НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок для підтримки діяльності пункту обміну валют

Виконав студент IV курсу, групи ІТ-04
(шифр групи)
Кльова Микола Михайлович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник зав. каф. ІПІ, д.т.н., проф. Жаріков Е.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., Шимкович В.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Кльові Миколі Михайловичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для підтримки діяльності пункту обміну валют

керівник проєкту Жаріков Едуард В'ячеславович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 14 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Предпроектне обстеження предметної області. Аналіз предметної області та існуючих рішень, відомих програмних продуктів, алгоритмічних та технічних рішень. Опис бізнес-процесів та постановка задачі.
2) Розроблення вимог до програмного забезпечення. Варіанти використання програмного забезпечення. Аналіз системних вимог. Функціональні і нефункціональні вимоги.
3) Конструювання програмного забезпечення. Архітектура, обгрунтування вибору розробки та аналіз безпеки даних.
4) Аналіз якості та тестування.
5) Розгортання та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	15.03.2024	
3	Постановка та формалізація задачі	22.03.2024	
4	Розробка інформаційного забезпечення	01.04.2024	
5	Алгоритмізація задачі	14.04.2024	
6	Обґрунтування вибору використаних технічних засобів	20.04.2024	
7	Розробка програмного забезпечення	01.05.2024	
8	Налагодження програми	14.05.2024	
9	Виконання графічних документів	17.05.2024	
10	Оформлення пояснювальної записки	25.05.2024	
11	Подання ДП на попередній захист	04.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Микола КЛЬОВА

(підпис) (ініціали, прізвище)

Керівник _____ Едуард ЖАРІКОВ

(підпис) (ініціали, прізвище)

Анотація

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 62 таблиць, 100 рисунків та 12 джерел – загалом 105 сторінок.

Дипломний проєкт присвячений розробці системи для підтримки діяльності пункту обміну валют.

Мета розробки є створення застосунку для підтримки діяльності пункту обміну валют за рахунок впровадження боту в телеграмі для системи обліку для працівників і власника бізнесу.

Об'єкт дослідження: процесс отримання і видачі валюти обмінником.

Предмет дослідження: способи збереження валюти і інформації про неї у обміннику.

У розділі «Аналіз вимог до програмного забезпечення» розглядаються загальні положення, аналіз предметної області, дослідження існуючих технологій та успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, а також постановка задачі.

Розділ «Моделювання та конструювання програмного забезпечення» зосереджений на аналізі та моделюванні програмного забезпечення, описі архітектури, розробці програмного забезпечення та аналізі безпеки даних.

У розділі «Аналіз якості та тестування програмного забезпечення» здійснюється аналіз якості ПЗ, описуються процеси тестування та наводиться контрольний приклад.

Останній розділ «Розгортання та супровід програмного забезпечення» містить опис розгортання та підтримки програмного забезпечення.

КЛЮЧОВІ СЛОВА: БОТ, НБУ, TELEGRAM, КЛІЄНТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СЕРВЕРНА ЧАСТИНА, БАЗА ДАНИХ, АРХІТЕКТУРА.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 62 tables, 100 figures, and 12 sources, totaling 105 pages.

The diploma project is dedicated to the development of a system to support the operations of a currency exchange point.

The goal of the development is to create an application to support the activities of a currency exchange point by implementing a Telegram bot for an accounting system for employees and the business owner.

Object of the research: the process of receiving and issuing currency at the exchange point.

Subject of the research: methods of storing currency and information about it at the exchange point.

The section "Analysis of Software Requirements" covers general provisions, domain analysis, study of existing technologies and successful IT projects, analysis of software requirements, and task setting.

The section "Software Modeling and Design" focuses on software analysis and modeling, architecture description, software development, and data security analysis.

In the section "Quality Analysis and Software Testing," the software quality is analyzed, testing processes are described, and a control example is provided.

The final section "Software Deployment and Maintenance" contains a description of the deployment and maintenance of the software.

KEYWORDS: BOT, NBU, TELEGRAM, CLIENT, SOFTWARE, BACK-END, DATABASE, ARCHITECTURE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0406.045440.01.91	Технічне завдання	22	
3	А4	КПІ.ІТ-0406.045440.02.81	Пояснювальна записка	105	
4	А4	КПІ.ІТ-0406.045440.03.12	Текст програми	165	
5	А4	КПІ.ІТ-0406.045440.05.34	Керівництво користувача	37	
6	А3	КПІ.ІТ-0406.045440.06.99	Графічний матеріал	8	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0406.045440.00.90		
Розроб.		Кльова М.М.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Жаріков Е.В.					
Н.контр.		Родіонов П.Ю.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-04	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Вебзастосунок для підтримки діяльності пункту обміну валют

Технічне завдання

КПІ.ІТ-0406.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Едуард ЖАРІКОВ

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Микола КЛЬОВА

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ 3
2 ПІДСТАВА ДЛЯ РОЗРОБКИ 4
3 ПРИЗНАЧЕННЯ РОЗРОБКИ 5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6
4.1 Вимоги до функціональних характеристик 6
4.1.1 Користувацького інтерфейсу 6
4.1.2 Для клієнту: 15
4.1.3 Для адміністратора системи: 15
4.2 Вимоги до надійності 16
4.3 Умови експлуатації 16
4.3.1 Вид обслуговування 16
4.3.2 Обслуговуючий персонал 16
4.4 Вимоги до складу і параметрів технічних засобів 16
4.5 Вимоги до інформаційної та програмної сумісності 18
4.5.1 Вимоги до вхідних даних 18
4.5.2 Вимоги до вихідних даних 18
4.5.3 Вимоги до мови розробки 18
4.5.4 Вимоги до середовища розробки 18
4.5.5 Вимоги до представленню вихідних кодів 18
4.6 Вимоги до маркування та пакування 18
4.7 Вимоги до транспортування та зберігання 19
4.8 Спеціальні вимоги 19
5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 20
5.1 Попередній склад програмної документації 20
5.2 Спеціальні вимоги до програмної документації 20
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ 21
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 22

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: вебзастосунок для підтримки діяльності пункту обміну валют.

Галузь застосування: наведене технічне завдання поширюється на розробку системи підтримки роботи пункту обміну валют, котра використовується для підтримки дільності пункту обміну валют і телеграм боту для взаємодії з клієнтами.

Користувачі: власники пунктів обміну валют, які хочуть у зручному місці зберігати дані про роботу, мати більше контролю над пунктом та структурувати свій бізнес.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку для підтримки діяльності пункту обміну валют є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації та полегшенню роботи з пунктом обміну валют, ведення обліку і більшого контролю над бізнесом.

Метою розробки є покращення та автоматизація обліку роботи пункту обміну валюти за рахунок розроблення застосунку для візуалізації та обліку транзакцій, створення бронювань обміну валюти, а також для взаємодії з системою телеграм.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

- виведення вікна авторизації у системі. Прототип зображений на рисунку 4.1;

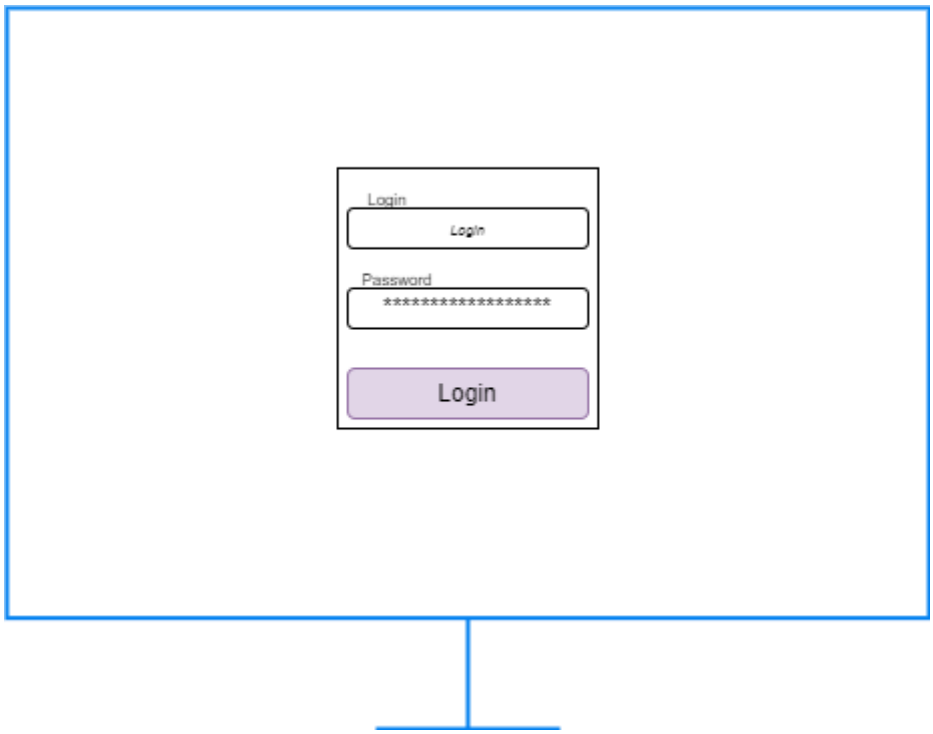


Рисунок 4.1 – Вікно авторизації у системі

- виведення головного меню після успішної авторизації у системі. Прототип зображений на рисунку 4.2;

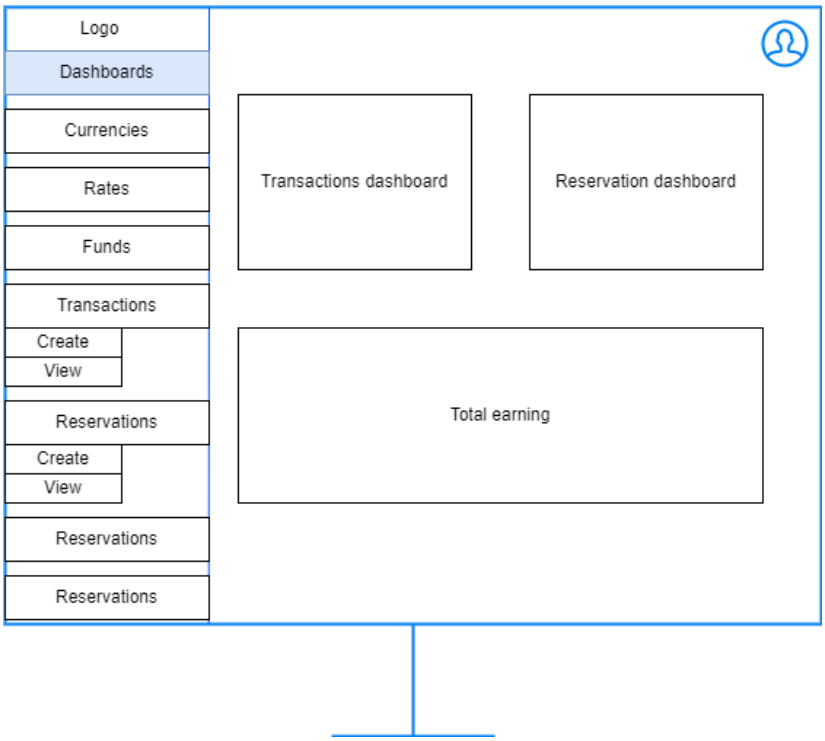


Рисунок 4.2 – Головне меню після успішної авторизації

- перегляд наявних валют у системі. Прототип зображений на рисунку 4.3;

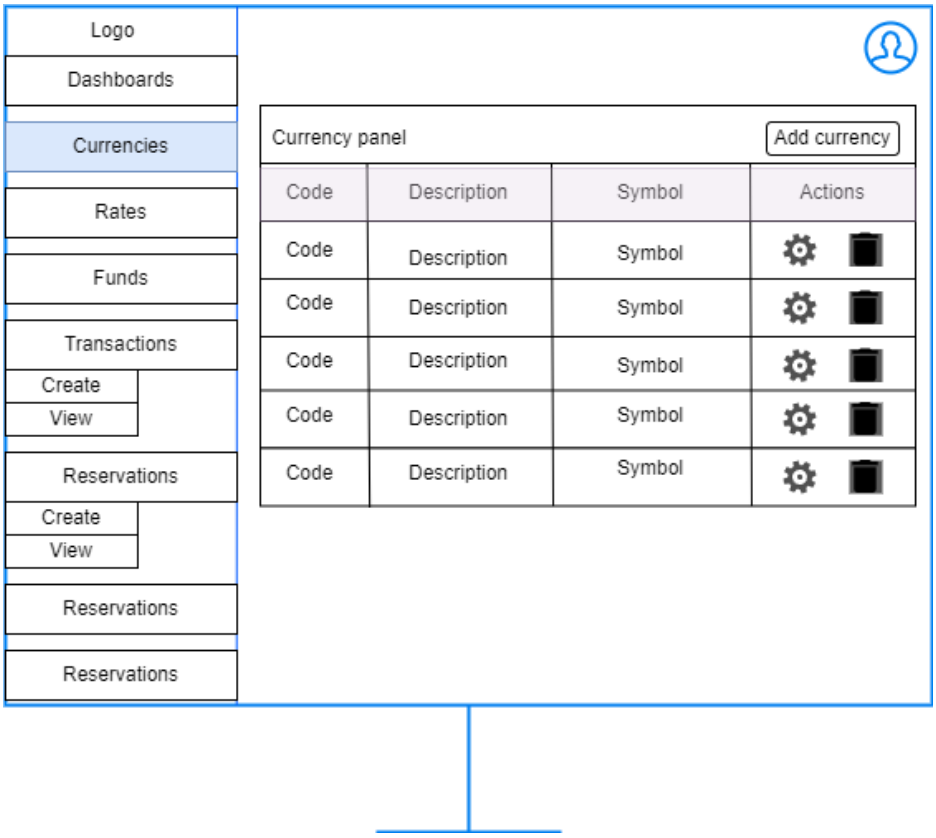


Рисунок 4.3 – Прототип списку наявних валют у системі

- створення нової валюти у системі. Прототип зображений на рисунку 4.4;

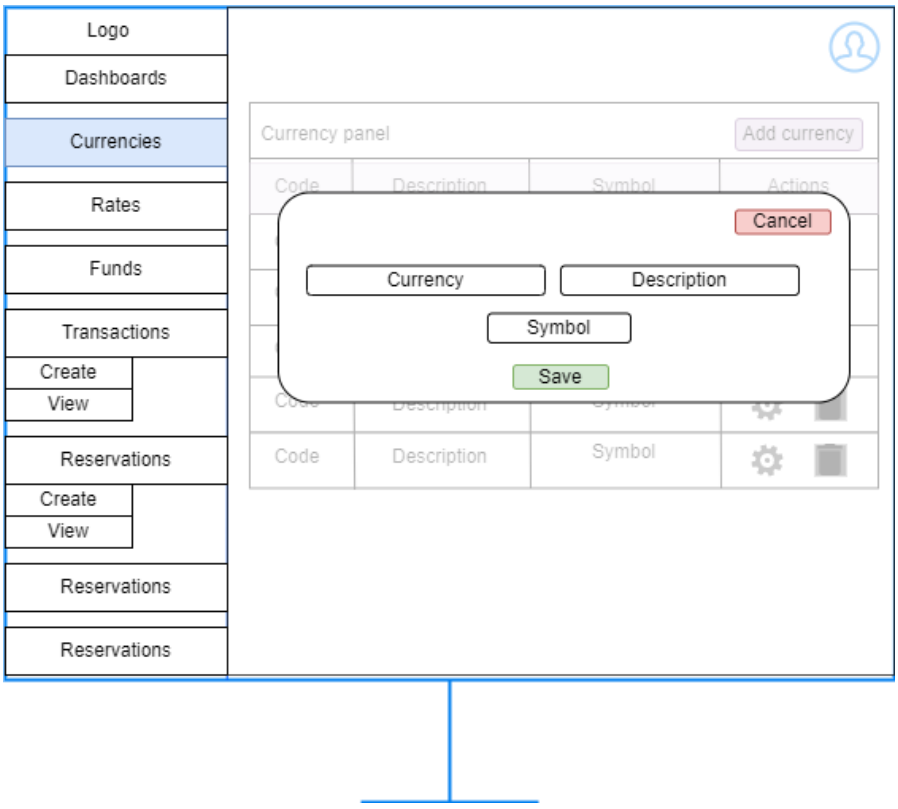


Рисунок 4.4 – Прототип створення валюти у системі

- оновлення валюти у системі. Прототип зображений на рисунку 4.5;

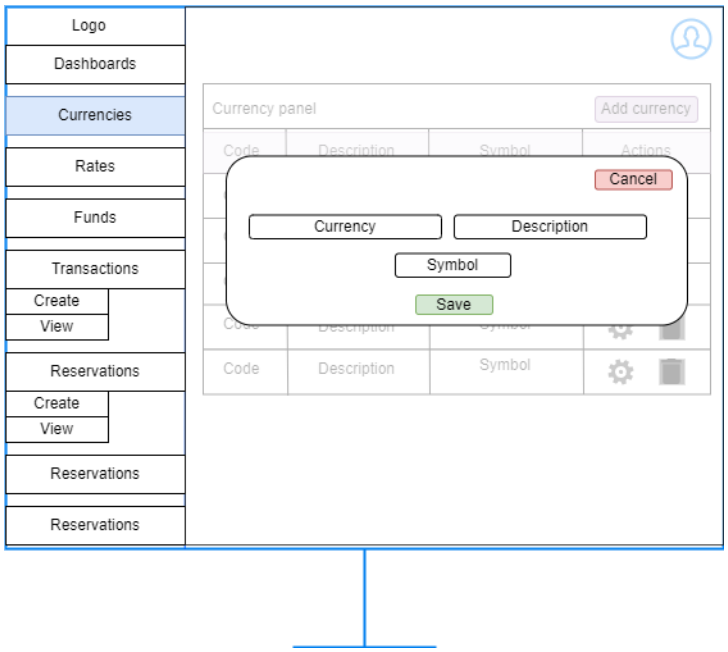


Рисунок 4.5 – Прототип оновлення валюти у системі

- перегляд списку курсів у системі. Прототип зображений на рисунку 4.6;

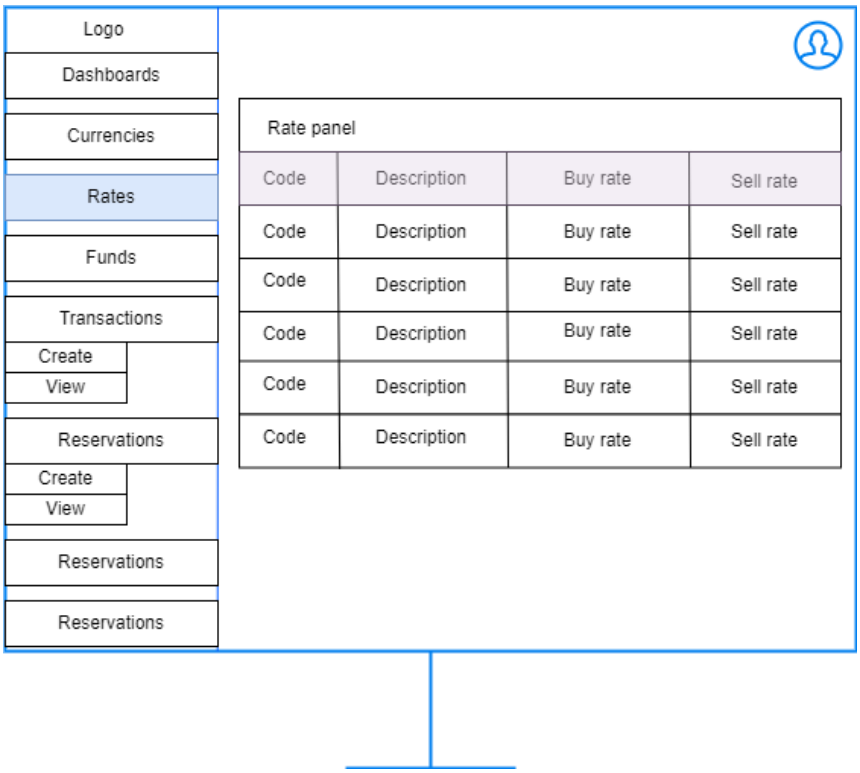


Рисунок 4.6 – Прототип перегляду списку курсів валют

- перегляд списку резервів у системі. Прототип зображений на рисунку 4.7;

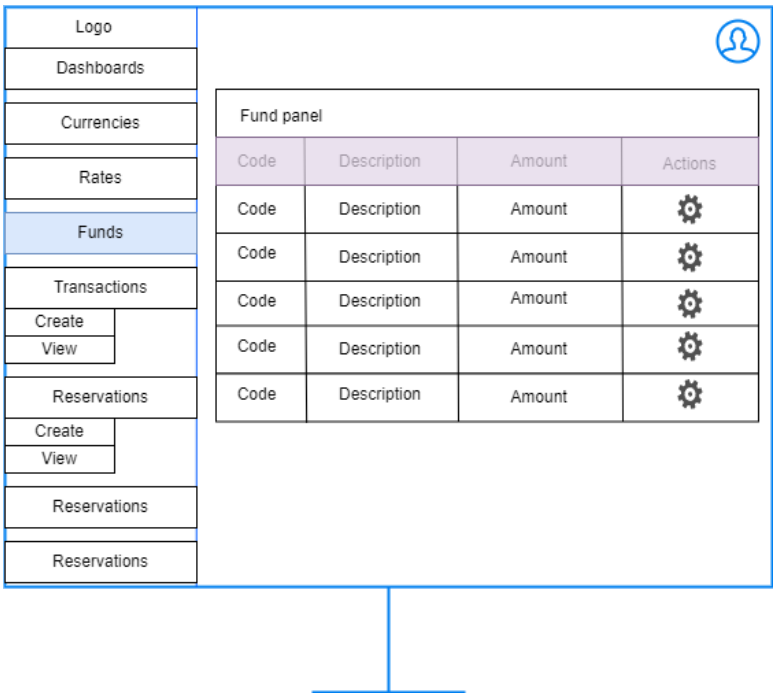


Рисунок 4.7 – Прототип перегляду списку резервів у системі

- оновлення резерву у системі. Прототип зображений на рисунку 4.8;

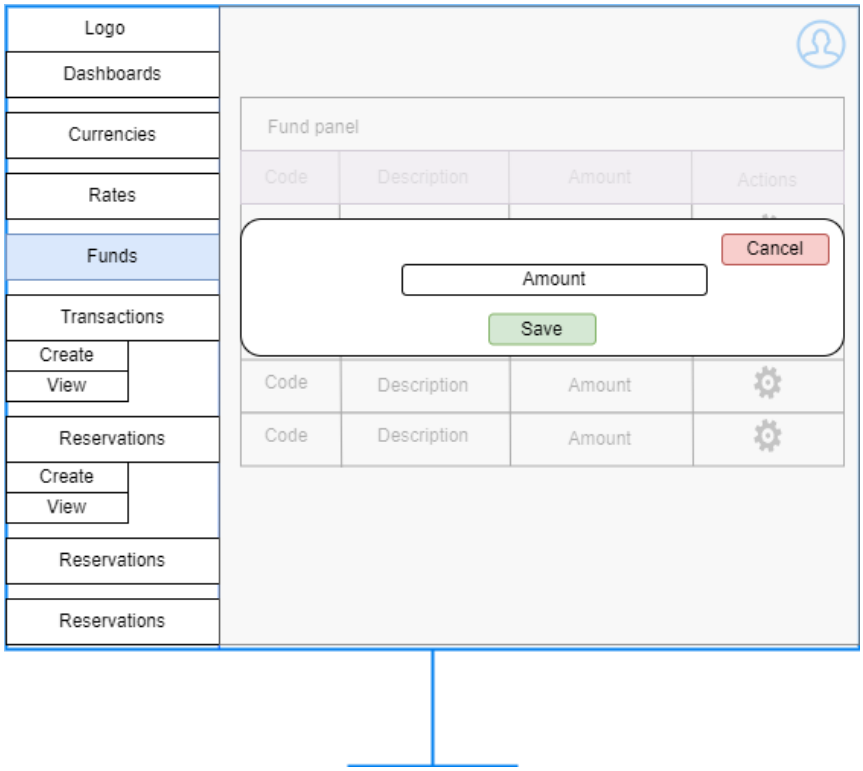


Рисунок 4.8 – Прототип оновлення резерву у системі

- перегляд бронювань у системі. Прототип зображений на рисунку 4.9;

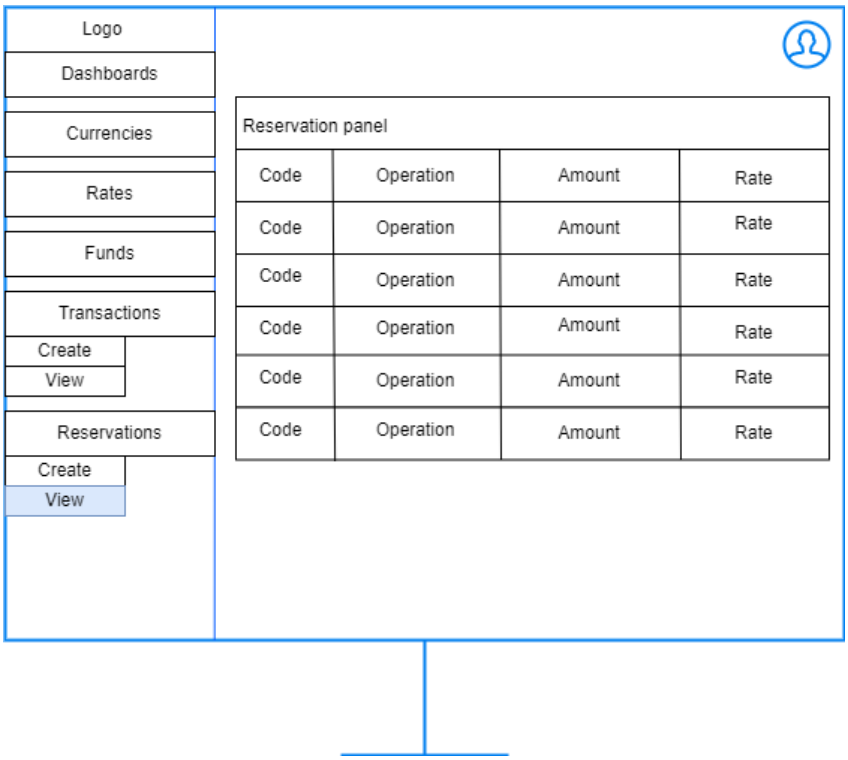


Рисунок 4.9 – Прототип перегляду бронювань у системі

- створення бронювання у системі. Прототип зображений на рисунку 4.10;

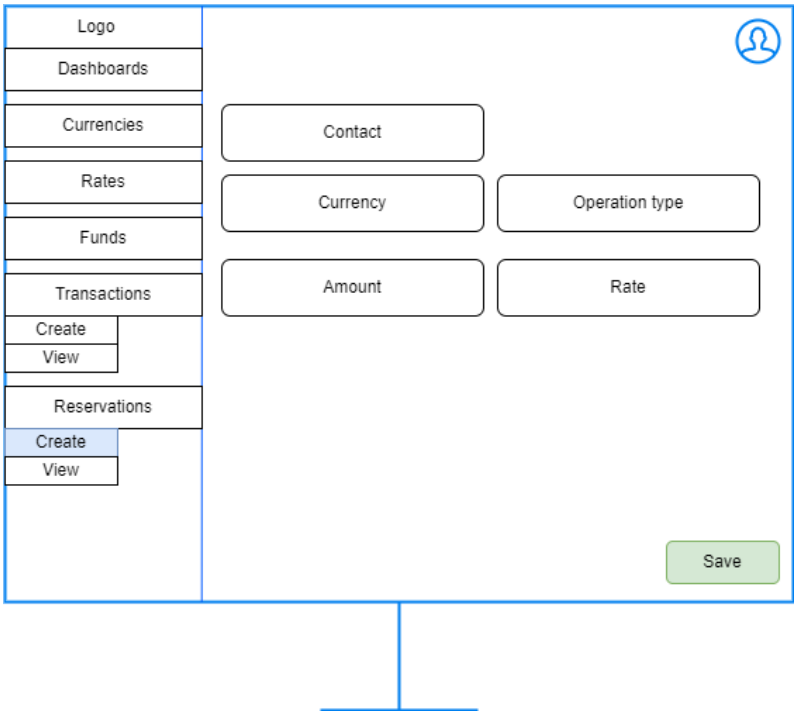


Рисунок 4.10 – Прототип створення бронювання у системі

- перегляд транзакцій у системі. Прототип зображений на рисунку 4.11;

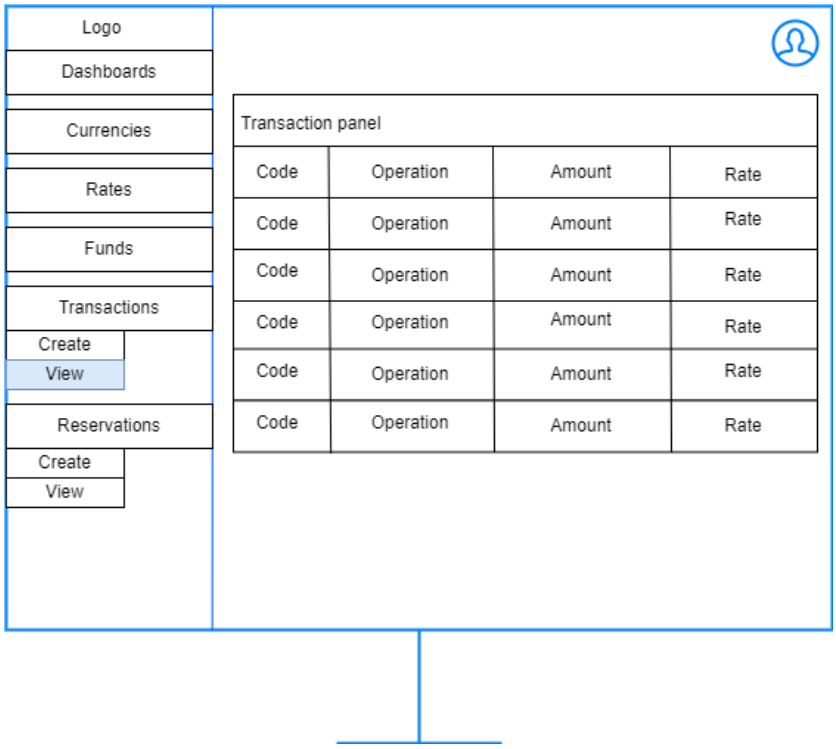


Рисунок 4.11 – Прототип перегляду транзакцій у системі

- створення транзакції у системі. Прототип зображений на рисунку 4.12;

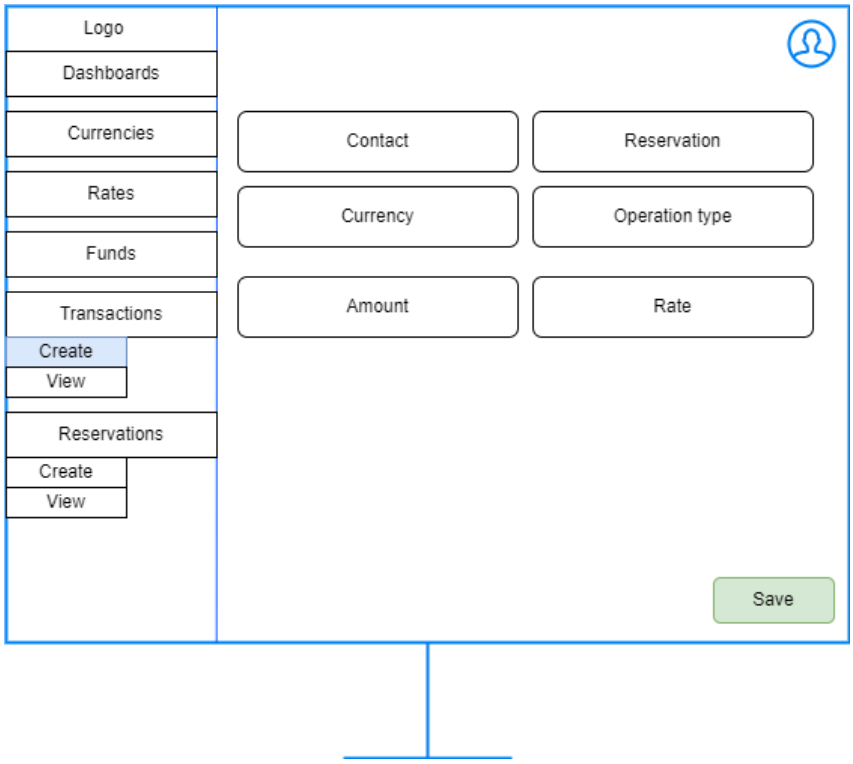


Рисунок 4.12 – Прототип створення транзакції у системі

- головне меню користувача telegram-боту. Прототип зображений на рисунку 4.13;



Рисунок 4.13 – Прототип меню користувача бота

- створення контаку користувачем telegram-боту. Прототип зображений на рисунку 4.14;



Рисунок 4.14 – Прототип форми створення контакту

- прототип перегляду контакту користувачем telegram-боту, рисунок 4.15;

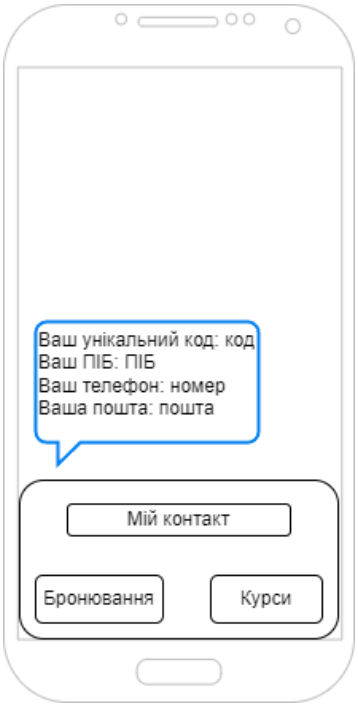


Рисунок 4.15 – Прототип форми перегляду контакту користувача

- прототип перегляду курсів користувачем telegram-боту. Прототип зображений на рисунку 4.16;

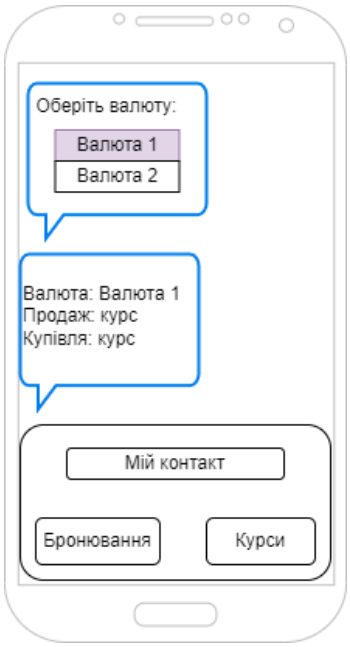


Рисунок 4.16 – Прототип форми перегляду валюти

- прототип створення бронювання валюти користувачем telegram-боту. Прототип зображений рисунку 4.17;

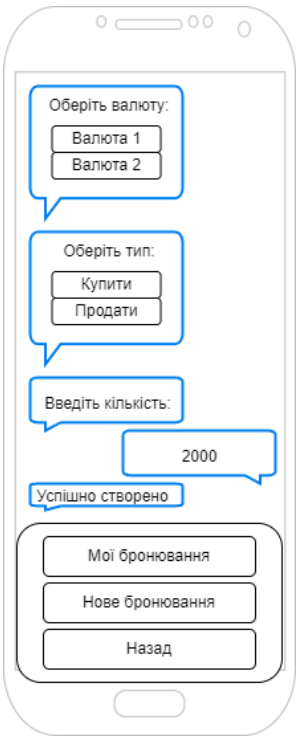


Рисунок 4.17 – Прототип форми створення бронювання користувачем бота

- прототип перегляду створених бронювань користувачем telegram-боту. Прототип зображений на рисунку 4.18;

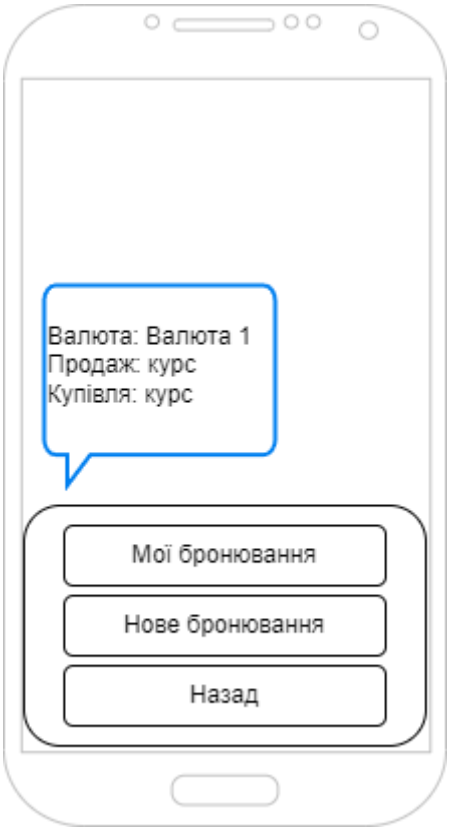


Рисунок 4.18 – Прототип форми перегляду створених бронювань користувачем telegram-бота

4.1.2 Для клієнту:

– створення бронювань;
– перегляд створених бронювань;
– перегляд актуальних курсів;
– створення свого контакту;
– перегляд створеного контакту.

4.1.3 Для адміністратора системи:

– створення валюти;
– редагування валюти;
– видалення валюти;
– редагування резервів;
– перегляд резервів.

– перегляд курсів;
– перегляд всіх бронювань;
– створення нового бронювання;
– оновлення бронювання;
– створення транзакції:
– перегляд всіх транзакцій;
– створення контакту;
– оновлення контакту.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Персонал повинен бути навчений для працювання з системою. Повинно бути 3 людини, касир – який відповідає за видачу валюти і внесення у систему інформації про це, обробка бронювань. Менеджер – створення нових валют, оновлення резервів. Власник – перегляд дохідності обмінного пункту.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп’ютерах.

Мінімальна конфігурація технічних засобів:

- тип процесору: Intel Core i5 або еквівалент;
- обсяг ОЗП: 14 ГБ;
- дисковий простір: 41 ГБ;
- підключення до мережі Інтернет: зі швидкістю від 20 мегабіт;
- операційна система: Windows Server 2016 або новіше, або Linux (наприклад, Ubuntu 18.04 або новіше);
- інші вимоги:
    1. .NET Core SDK 3.1 або новіше.
    2. Node.js 12.x або новіше.
    3. PostgreSQL 10 або новіше.
    4. Redis 5.x або новіше.
    5. Сучасні браузери (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
    6. SSL/TLS сертифікати для захисту HTTP-запитів (HTTPS).
    7. Налаштовані фаєрволи та правила доступу до баз даних і Redis.

Рекомендована конфігурація технічних засобів (та на якій виконувалась розробка):

- тип процесору: Intel Core i7 або еквівалент;
- обсяг ОЗП: 16 ГБ;
- дисковий простір: 100 ГБ;
- підключення до мережі Інтернет: зі швидкістю від 100 мегабіт;
- операційна система: Windows Server 2019 або новіше, або Linux (наприклад, Ubuntu 20.04 або новіше);
- інші вимоги:
    1. .NET Core SDK 5.0 або новіше.
    2. Node.js 14.x або новіше.
    3. PostgreSQL 12 або новіше.
    4. Redis 6.x або новіше.

5. Сучасні браузери (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
6. SSL/TLS сертифікати для захисту HTTP-запитів (HTTPS).
7. Налаштовані фаєрволи та правила доступу до баз даних і Redis.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows (Windows 10, Windows 11, Windows Server і т.д.) або Unix (Linux дистрибутиви, такі як Ubuntu, CentOS і т.д.).

4.5.1 Вимоги до вхідних даних

Особливі вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Особливі вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# з використанням фреймворку ASP.Net Core. JavaScript – з використанням фреймворку Vue.js та Vuetify.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Microsoft Visual Studio та JetBrains Web Storm.

4.5.5 Вимоги до представленню вихідних кодів

Вимогу до представлення вихідних кодів не висуваються.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема бази даних;
– архітектура програмного забезпечення;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Вебзастосунок для підтримки діяльності пункту обміну валют**

КПІ.ІТ-0406.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4
ВСТУП 5
1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Аналіз предметної області 7
1.2 Аналіз існуючих рішень 9
1.2.1 Аналіз відомих програмних продуктів 9
1.2.2 Аналіз відомих алгоритмічних та технічних рішень 15
1.2 Опис бізнес-процесів 20
1.3 Постановка задачі 23
Висновки до розділу 23
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24
2.1 Варіанти використання програмного забезпечення 24
2.2 Аналіз системних вимог 36
2.3 Розроблення функціональних вимог 37
2.4 Розроблення нефункціональних вимог 42
Висновки до розділу 43
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 44
3.1 Архітектура програмного забезпечення 44
3.2 Обґрунтування вибору засобів розробки 46
3.3 Конструювання програмного забезпечення 50
3.4 Аналіз безпеки даних 62
Висновки до розділу 63
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 64
4.1 Аналіз якості ПЗ 64
4.2 Опис процесів тестування 66
4.3 Опис контрольного прикладу 72

Висновки до розділу 97
5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 98
5.1 Розгортання програмного забезпечення 98
5.2 Супровід програмного забезпечення 101
Висновки до розділу 102
ВИСНОВКИ 103
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 104
ДОДАТОК А ЗВІТ ПОДІБНОСТІ 105

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface – прикладний програмний Інтерфейс
REST	–	Representational State Transfer – передача репрезентативного стану
TCP	–	Transmission Control Protocol – протокол керування передачею
HTML	–	HyperText Markup Language – мова гіпертекстової розмітки
IP	–	Internet Protocol –інтернет-протокол
URL	–	Uniform Resource Locator – єдиний вказівник на ресурс
JSON	–	JavaScript Object Notation – запис об'єктів JavaScript
NoSQL	–	Not Only Structured Query Language – не тільки структурована мова запитів
DB	–	Database – база даних
IT	–	Інформаційні технології
ЦП	–	Центральний процесор
Гб	–	Гігабайт
ПЗ	–	Програмне забезпечення
БД	–	База даних.

ВСТУП

Історія грошей починається ще тисячоліття тому. Перші банкноти з'явилися у Китаї з розвитком паперового виробництва. Китайським купцям було не зручно перевозити багато монет і розплачуватись ними, тому на початку 9 століття почалося їх виготовлення і у 10 столітті, вони вже стали повноцінною частиною економіки[1].

З часом вся Європа почала використовувати паперові гроші, навіть до сьогоднішнього дня. Сьогодні, у кожної країни є своя національна валюта, якою люди в її межах, можуть купувати речі.

Багато експортерів завозять гроші після продажу товарів за кордоном. Імпортери продають товари і вивозять кошти, після чого обмінюють їх на свою національну валюту. Іноземні інвестори вкладаються в проєкти, поза їх рідними країнами і також, обмінюють кошти. Ця потреба створила попит для заробітку на спекуляціях.

Купують валюту дешевше, продають дорожче – дохід на різниці, який має покрити податки, оренди і забезпечити дохід. Тому почали з'являтись установи, які надають послуги у обміну валют для великого бізнесу.

У людей, які не пов'язані з великим бізнесом, є також свої потреби. Деякі валюти є більш стабільними, ніж їх конкуренти у інших країнах, тому люди, для заощадження або збереження, у деяких випадках і примноження своїх коштів, купують їх за певним, визначеним економічним рівнем, курсом. Або мандруючи чужою країною, неможна купити речі у магазинах, використовуючи свою національну валюту, потрібно робити обмін, у спеціальних пунктах або банках, куди приносять певну валюту, а отримують необхідну, для певних, своїх цілей. Через відсутність зацікавленості великих фінансових установ у малих операціях, виникла потреба в створенні обмінних пунктів валют у людних місцях, що обслуговують широкі верстви населення.

З розвитком та розповсюдженням інтернету велика кількість сервісів перейшла в онлайн-простір. Зростаюча кількість користувачів надає перевагу вебзастосункам перед традиційними фізичними пунктами обміну валюти.

Інтернет-сервіси різних видів діяльності стають дедалі популярнішими завдяки своїй зручності та простоті використання, адже ними можна користуватися навіть через месенджери. Так звані чат-боти – це помічники, які спілкуються з користувачами за допомогою повідомлень. Перевагою є те, що користувач отримує відповідь майже миттєво, оскільки запити обробляються системою.

Однією з ключових сфер, що перейшла в онлайн, є обмін валют. Сучасні електронні пункти обміну валют дозволяють здійснювати операції швидко та безпечно, що особливо важливо в умовах економічної нестабільності. Власники таких сервісів отримують можливість ефективно адмініструвати бізнес, контролювати всі транзакції в режимі реального часу, а також відстежувати курси валют та налаштовувати вигідні пропозиції для клієнтів.

Для клієнтів електронні пункти обміну валют також мають значні переваги. Окрім зручності та швидкості обміну, вони надають додатковий рівень безпеки та прозорості. Клієнти можуть обмінювати великі суми за резервуванням, що зменшує ризик втрати коштів. Крім того, доступ до інформації про курси валют дозволяє клієнтам робити обґрунтовані фінансові рішення.

Основним завданням є покращення та автоматизація обліку роботи пункту обміну валюти за рахунок розроблення застосунку для візуалізації та обліку транзакцій, створення бронювань обміну валюти, а також для взаємодії з системою телеграм.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

В сучасному світі, де економіка є важливою складовою життя, люди цікавляться, як зберегти свої кошти. Забезпечення зручного інструменту для аналізу курсу валюти та створення бронювань валюти дозволяє користувачам бути більш інформованими та зважено висувати фінансові рішення.

З точки зору бізнесу, аналіз ефективності, облік та автоматизація всіх процесів є вкрай важливими складовими успішного управління. Для якісного аналізу та планування ризиків або прибутків, зручний інструмент є необхідною умовою успіху.

Предметна область аналізу ефективності і автоматизації пункту обміну валюти має велику важливість, з кількох причин:

- можливість відслідковування усіх транзацій, які проведені у розрізі днів, тижнів, місяців і років дає змогу планувати прибуток фірми, певні ризики і будувати плани на подальший оборот валют;
- збір інформації про користувачів надає можливість сформувати загальні характеристики клієнтів та будувати стратегії розвитку, включаючи цю інформацію;
- можливість відстежувати обсяги зарезервованої валюти протягом певного періоду має важливе значення у плануванні резервів на наступний період часу;
- агрегація всієї інформації про роботу бізнесу в одному місці дозволяє уникнути пропуску будь-якого важливого аспекту при проведенні аналізу.

Предметна область розробки програмного забезпечення для покращення та автоматизації обліку роботи пункту обміну валюти має значний потенціал на сучасному ринку. Ринок надає різноманітні програмні рішення, які допомагають власникам ефективно керувати бізнесом і будувати бізнес плани,

але більшість з них потребують більшого занурення у цю нішу та можуть бути поліпшені.

Зазвичай люди шукають зручні інструменти для вирішення фінансових питань, особливо, коли йдеться про обмін валют. У сучасному світі, де економіка постійно змінюється, програмне забезпечення для аналізу та автоматизації пункту обміну валют може стати справжньою перлиною для бізнесу. Воно дозволяє власникам пунктів обміну валют ефективно керувати фінансовими рішеннями, а також управляти ризиками та прибутками. Збір та аналіз інформації про транзакції допомагає у розумінні тенденцій ринку та плануванні майбутніх кроків. Також, можливість відслідковувати попит на певні валюти дозволяє точно визначити необхідні резерви. Отже, розробка програмного забезпечення для підтримки діяльності пункту обміну валют має великий потенціал у сучасному світі фінансів. Шляхи розвитку цієї області можуть включати пошук нових методів аналізу ринку, створення простих у використанні інтерфейсів та постійне підтримання безпеки даних користувачів.

У проєкті обрано два напрямки розробки програмного забезпечення: телеграм бот для клієнтського інтерфейсу та вебсайт для адміністрування. Телеграм бот надає зручний та простий інтерфейс для клієнтів, де вони можуть легко створювати бронювання та відстежувати курси валют. Сайт, у свою чергу, призначений для співробітників і власників бізнесу, де вони можуть проводити адміністративні операції, такі як додавання нових валют, встановлення курсів, управління резервами та транзакціями. Ці напрямки забезпечують комплексне вирішення завдань і задовольняє потреби різних категорій користувачів, надаючи їм зручність та ефективність у виконанні їхніх завдань.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні програмне забезпечення у даній області та технічні рішення, що можливо використати у реалізації вебзастосунку підтрики діяльності пункту обміну валют. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.2.1 Аналіз відомих програмних продуктів

У сфері програмного забезпечення для підтримки роботи і автоматизації бізнес-процесів існує багато різноманітних інструментів і сервісів. Деякі з них мають широку популярність серед користувачів:

- creatio, salesforce, zoho, надають різноманітні загальні інструменти для аналітики, автоматизації і підвищення ефективності серед бізнесів. Вони дозволяють вести звітність бізнесу, автоматизовувати процеси, підтримувати ефективну роботу;
- сервіси, які надають курси Національного банку, криптовалютних бірж, такі як: BankGovUa, Minfin, Privatbank, Binance, OKX.

Зважаючи на популярність і широкий функціонал, creatio визнана як одна з найефективніших платформ для автоматизації бізнес-процесів. Вона пропонує рішення для CRM, маркетингу, продажів та обслуговування клієнтів. Creatio дозволяє підприємствам оптимізувати свою діяльність, використовуючи інструменти для автоматизації процесів та підвищення продуктивності роботи. Більш націлена на low-code розробку. На рисунку 1.1 зображена головна сторінка для налаштування сторінок у системі Creatio[2].

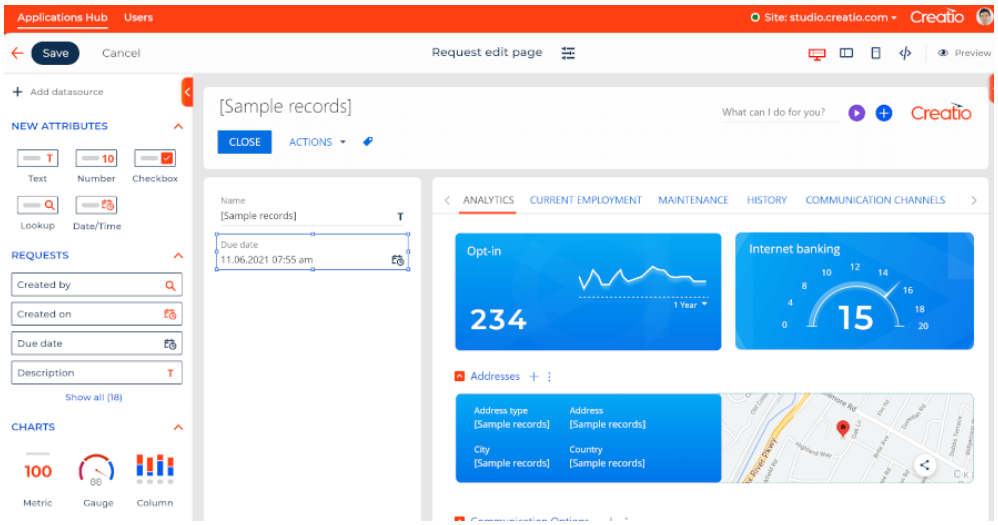


Рисунок 1.1 – Головна сторінка для налаштування сторінок у системі Creatio

Salesforce є лідером у сфері CRM та надає рішення для автоматизації маркетингу, продажів, обслуговування клієнтів та аналізу даних. Ця платформа допомагає підприємствам побудувати індивідуальні відносини з клієнтами та забезпечити ефективне управління бізнес-процесами. На рисунку 1.2 зображена головна сторінка для реєстрації звернень у CRM системі Salesforce[3].

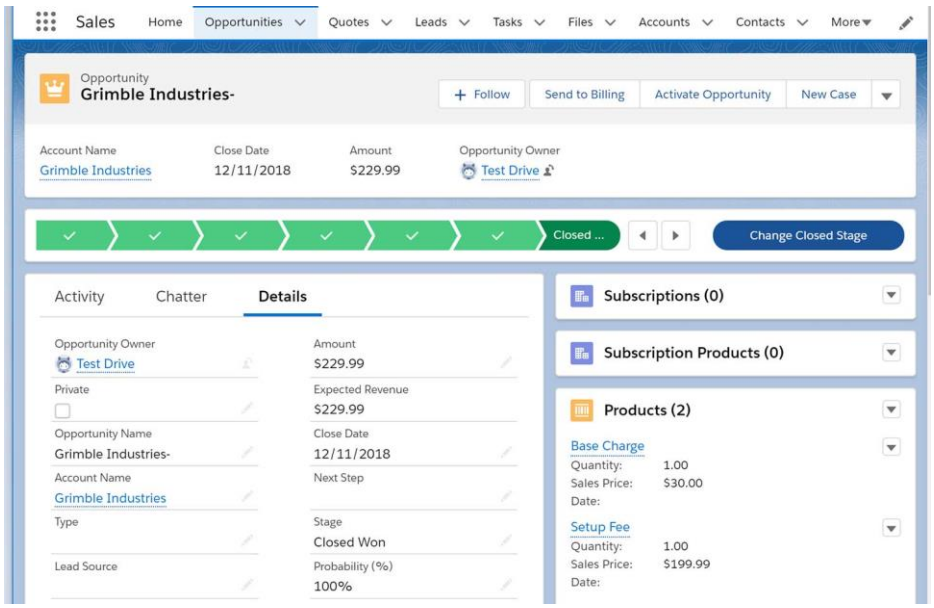


Рисунок 1.2 – Головна сторінка для реєстрації звернень у системі Salesforce

Zoho CRM - це інтегрована платформа для управління відносинами з клієнтами, яка допомагає підприємствам ефективно керувати маркетингом, продажами та обслуговуванням клієнтів. Вона надає широкий спектр інструментів для управління лідами, контактами, угодами та звітами, що допомагає підприємствам підвищити ефективність своїх бізнес-процесів та залучити нових клієнтів. На рисунку 1.3 зображені дашборди для відслідковування кількості і розмірів продажів у системі Zoho[4].

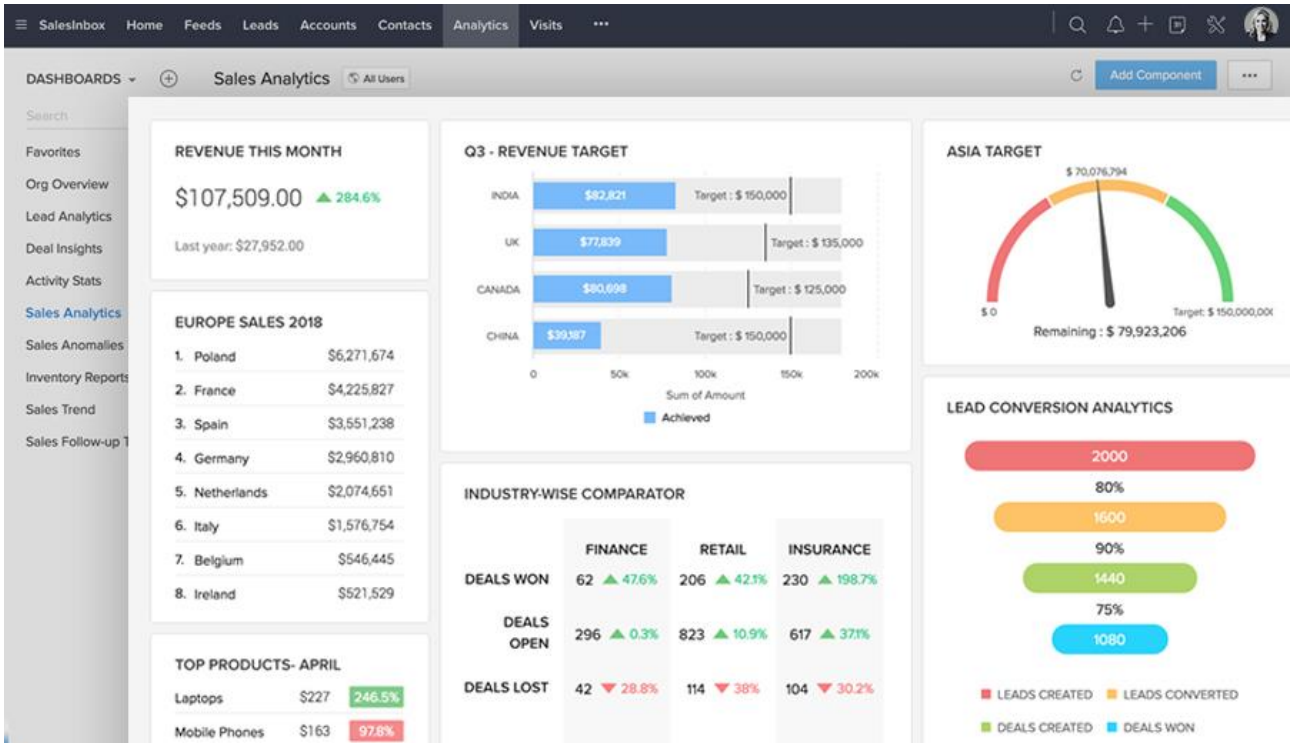


Рисунок 1.3 – Дашборди відслідковування кількості і розмірів продажів у системі Zoho

Кожна платформа має свої переваги та недоліки. Creatio – має гарний low-code функціонал, але важка у налаштуванні. Salesforce, Zoho – надають гнучкий функціонал і високий рівень безпеки, всі підтримують інтеграції зі сторонніми системами. Але є суттєві недоліки у всіх варіантах. Ціна обслуговування та розробки. Це дорогі рішення, які використовуються професіоналами і маленькому бізнесу багато з наявних функцій не потрібні.

Сервіси, такі як BankGovUa, Minfin, Privatbank, Binance і OKX, є одними з найпопулярніших і надійних джерел аналітики фінансових курсів в Україні.

Minfin – глобальний провайдер новин та аналітики, який пропонує новини у світі фінансів, агрегатор курсів валюти багатьох обмінних пунктів. За допомогою цієї інформації можна відслідковувати тенденції, встановлювати актуальний курс і моніторити ситуацію на ринку[5].

BankGovUa – національний банк України, надає інформацію про курс валют, який встановлений та регульований державою. Він надає просту інформацію, як: валюта – курс[6].

PrivatBank – найбільший банк України, який надає інформацію про курси валют, найнижчі серед всіх можливих комерційних установ[7].

Binance, OKX – криптовалютні біржі, які надають можливість аналізувати цифрові активи. На основі них можна будувати можливі майбутні курси обміну[8][9].

Ці сервіси відомі своєю надійністю, швидкістю оновлення інформації та широким покриттям різних фінансових ринків. Вони надають користувачам можливість бути в курсі останніх новин та трендів, що допомагає користувачам та інвесторам приймати обґрунтовані рішення. Крім того, ці сервіси зазвичай мають інтуїтивно зрозумілий інтерфейс, що дозволяє зручно переглядати та аналізувати інформацію.

Для порівняння проєкту з аналогами скористаємося таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Проєкт: Currexchange	Creatio	Salesforce	Zoho	Пояснення
Створення транзакції	+	+	+	+	Створення транзакції після прийому\ видачі валюти

Продовження таблиці 1.1

Додавання валют	+	+	+	+	Додавання валюти
Створення резервів	+	+	+	+	Створення резерву по проханню клієнта
Перегляд резервів і транзакцій	+	+	+	+	Перелік виконаних транзакцій \ виданих резервів
Створення курсів	+	+	+	+	Створення курсів
Створення користувача	+	+	+	+	Створення користувача
Телеграм бот для взаємодії з клієнтов	+	-	-	-	Первинний бот для створення користувачем бронювання
Зручність використання	+	+	-	-	Currexchange працює швидше через меншу кількість даних

Продовження таблиці 1.1

Легкість занурення у розробку	+	-	-	-	Currexchange написаний на ванільних технологіях, тому легше знайти рішення
Low-code підтримка	-	+	-	-	Налаштування системи та додавання нових функції без коду
Первинне налаштування	+	-	-	-	Для використання, потрібно налаштувати систему, додати об’єктну модель і т.д.
Безкоштовне використання	+	-	-	-	Це все не безкоштовні ресурси

Скориставшись порівняльною таблицею, можна зробити висновок, що запропонований функціонал вебзастосунку для підтримки діяльності пункту обміну валют зручніший і більш нішевий, аніж той, який наданий в інших програмних продуктах. Наявність ширшого функціоналу в поєднанні з інтуїтивно зрозумілим інтерфейсом позитивно впливає на ефективність бізнесу,

збільшуючи потік клієнтів, а також підвищує ефективність роботи самого пункту обміну валюти за рахунок мінімізації ручних процесів.

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Існує велика кількість архітектурних шаблонів, які можуть бути застосовані при розробці програмного забезпечення для пункту обміну валют. Один з таких шаблонів - це клієнт-серверна архітектура.

У клієнт-серверній архітектурі система розбивається на дві основні складові: клієнти і сервери. Клієнти виконують роль інтерфейсу, через який користувачі звертаються до сервера для отримання ресурсів, таких як дані, послуги або обробка операцій. Сервери, у свою чергу, надають ці ресурси та обробляють запити від клієнтів. Схема клієнт-серверної архітектури зображена на рисунку 1.4.

Основні переваги цієї архітектури включають:

- чітке визначення ролей та обов'язків клієнта та сервера, що спрощує розробку, тестування та управління системою;
- можливість масштабування системи шляхом додавання нових серверів або розподілу навантаження між серверами. Це дозволяє підтримувати високу продуктивність та швидкодію системи при зростанні обсягу даних та кількості користувачів;
- можливість сервера забезпечувати механізми автентифікації, авторизації та шифрування, що забезпечує більшу безпеку даних та захист від несанкціонованого доступу.

Архітектура клієнт-сервер має також свої недоліки:

- потреба в стабільному з'єднанні з сервером для отримання ресурсів може призвести до затримок та зниження продуктивності системи в разі проблем з мережевим з'єднанням;
- вразливість до відмови сервера або його недоступності може спричинити зупинку всієї системи та призвести до недоступності послуг для клієнтів;

- розгортання та управління розподіленою системою з багатьма серверами та клієнтами може бути складним завданням. Координація та синхронізація роботи серверів можуть викликати проблеми, особливо при розподілених транзакціях та кешуванні даних.

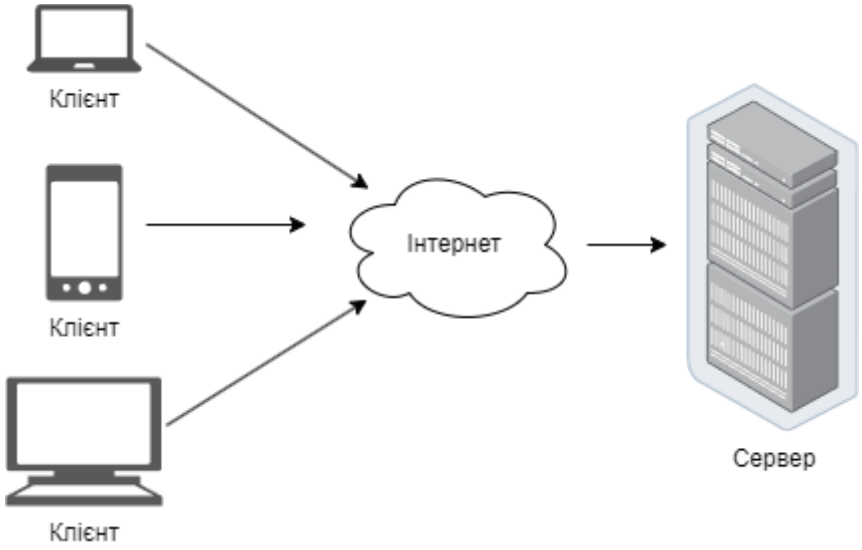


Рисунок 1.4 – Клієнт-серверна архітектура

Монолітна архітектура є традиційним підходом до розробки програмного забезпечення, де весь застосунок побудований як єдиний блок, який включає в себе всі компоненти та функціонал. У такій архітектурі весь код, логіка та інтерфейси розташовані разом, що може стати надмірно великим та складним для управління та розширення. Зміни в одній частині можуть мати вплив на всю систему. Схема монолітної архітектури зображена на рисунку 1.5.

Переваги:

- простота в розробці, оскільки всі компоненти розташовані разом і не потребують додаткових взаємодій чи комунікацій між ними;
- швидкість внесення змін, оскільки весь код знаходиться в одному місці, і немає необхідності розгортати окремі компоненти або виконувати складну інтеграцію;
- легкість розгортання, оскільки не потрібне складне налаштування або управління різними сервісами або серверами.

Недоліки:

- складність додавання нових функцій або змін до існуючих через велику взаємозалежність компонентів, що може призвести до виникнення проблем при модифікації;
- висока зчепленість між компонентами робить систему менш гнучкою та складною для розуміння та підтримки;
- помилки або проблеми в одному компоненті можуть мати вплив на всю систему, оскільки весь код знаходиться в єдиному блоку, що робить ізоляцію та локалізацію проблем складнішою.

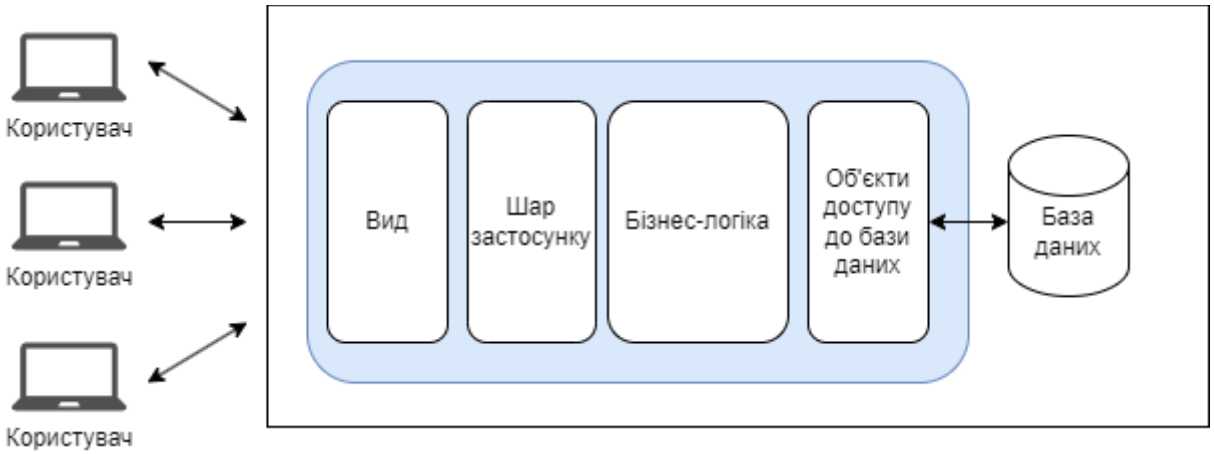


Рисунок 1.5 – Монолітна архітектура

MVP (Model-View-Presenter) - це шаблон архітектури для розробки користувацького інтерфейсу, що складається з трьох основних компонентів: Модель (Model), Вид (View) та Презентер (Presenter). Модель відповідає за обробку даних та бізнес-логіку, вид - за відображення даних та взаємодію з користувачем, а презентер виступає посередником між моделлю та представленням, оброблюючи дії користувача та оновлюючи модель та представлення за необхідності. Схема MVP архітектури зображена на рисунку 1.6.

Переваги:

- чітке визначення ролей та обов'язків кожного компонента спрощує тестування, розробку та підтримку програмного забезпечення;

- компоненти можуть бути змінені або замінені без впливу на решту системи, що забезпечує більшу гнучкість та легкість розширення додатку;
- ефективне тестування кожного компонента окремо сприяє покращенню якості та стабільності програмного забезпечення.

Недоліки:

- порівняно з іншими архітектурними шаблонами, MVP може вимагати більшої кількості коду та складнішої організації компонентів;
- презентер як посередник між моделлю та представленням може призвести до збільшеної кількості комунікації та залежностей між компонентами;
- впровадження MVP може потребувати додаткових витрат на розробку, оскільки потрібно розробити всі необхідні компоненти та встановити взаємозв'язки між ними.

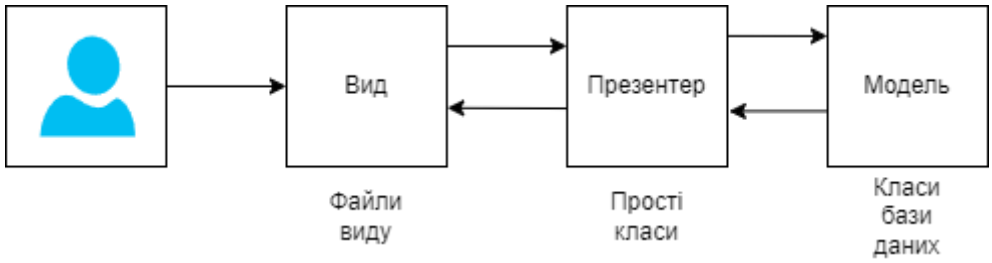


Рисунок 1.6 – MVP архітектура

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, де застосунок розбивається на невеликі незалежні сервіси, які функціонують незалежно один від одного та виконують обмежену кількість функцій. Кожен сервіс відповідає за виконання конкретної функції або послуги, що дозволяє підвищити гнучкість та масштабованість системи. Схема мікросервісної архітектури зображена на рисунку 1.7.

Переваги:

- мікросервіси дозволяють зменшити залежність між компонентами системи, що полегшує розробку, тестування та впровадження змін;
- кожен сервіс може бути розгорнутий, масштабований та оновлений незалежно від інших, що сприяє гнучкості та швидкості впровадження нових функцій або виправлень;
- мікросервіси дозволяють використовувати різні технології та мови програмування для кожного сервісу в залежності від його потреб, що сприяє оптимізації та покращенню продуктивності.

Недоліки:

- управління багатьма мікросервісами може стати складним завданням, оскільки потрібно вирішувати питання щодо моніторингу, керування та координації їх взаємодії;
- комунікація між сервісами може стати джерелом проблем, зокрема затримок та перевантажень мережі;
- розбиття додатку на багато невеликих сервісів може призвести до збільшення складності розробки та впровадження системи.

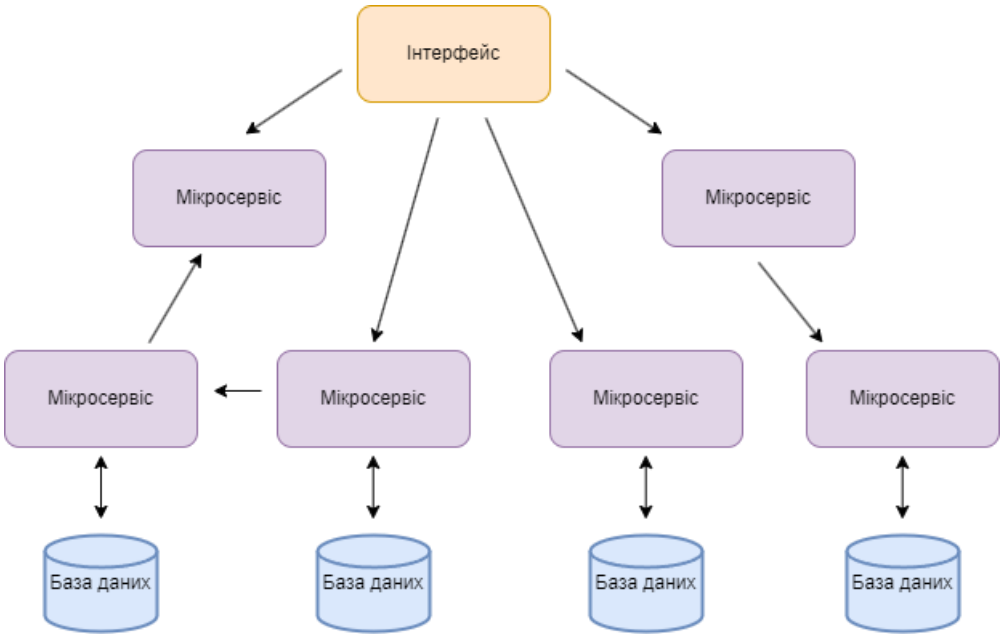


Рисунок 1.7 – Мікросервісна архітектура

1.2 Опис бізнес-процесів

Розробка програмного забезпечення для пункту обміну валют може бути представлена за допомогою моделювання бізнес-процесів за допомогою BPMN. Використання цих засобів дозволяє візуалізувати та надати точність для кожного кроку у процесі розробки, а також сприятиме кращому розумінню бізнес-дій.

Користувач заходить у систему обліку пункту обміну валют. Переходить на сторінку створення транзакції. Після заповнення всіх обов’язкових полів натискає на кнопку “Створити транзакцію”. Після чого запит з введенною інформацією відправляється на сервер, де перевіряється, чи вистачає наявного резерву для створення транзакції. У позитивному випадку – створюється транзакція та оновлюються резерви у відповідності до типу транзакції, курсу і кількості валюти. У негативному – користувачу буде виведено інформацію про те, що резерву не вистачає, щоб обслугувати дану транзакцію і вона не була створена. Бізнес-процес створення транзакції робітниками бізнесу зображений на рисунку 1.8.

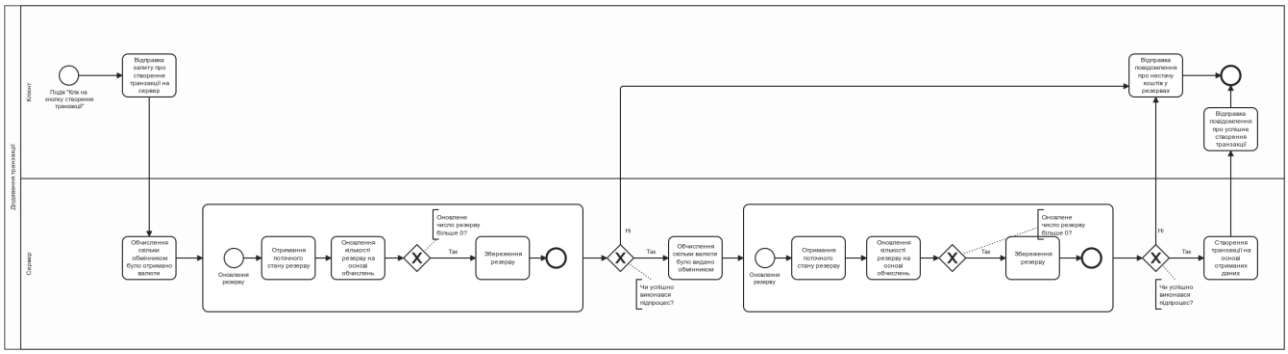


Рисунок 1.8 – Бізнес-процес створення транзакції робітниками бізнесу у системі

Для створення бронювання адміністратором він повинен зайти у систему. Після чого переходить на сторінку створення бронювання і вводить всі поля,

які є необхідними для заповнення і натискає на кнопку “Створити бронювання”. Система відправляє запит на сервер, де перевіряється вказаний контакт і валюта на наявність у системі. Після чого бронювання успішно створюється і система виводить вікно з інформацією про це. Бізнес-процес створення бронювання у системі, зображений на рисунку 1.9.

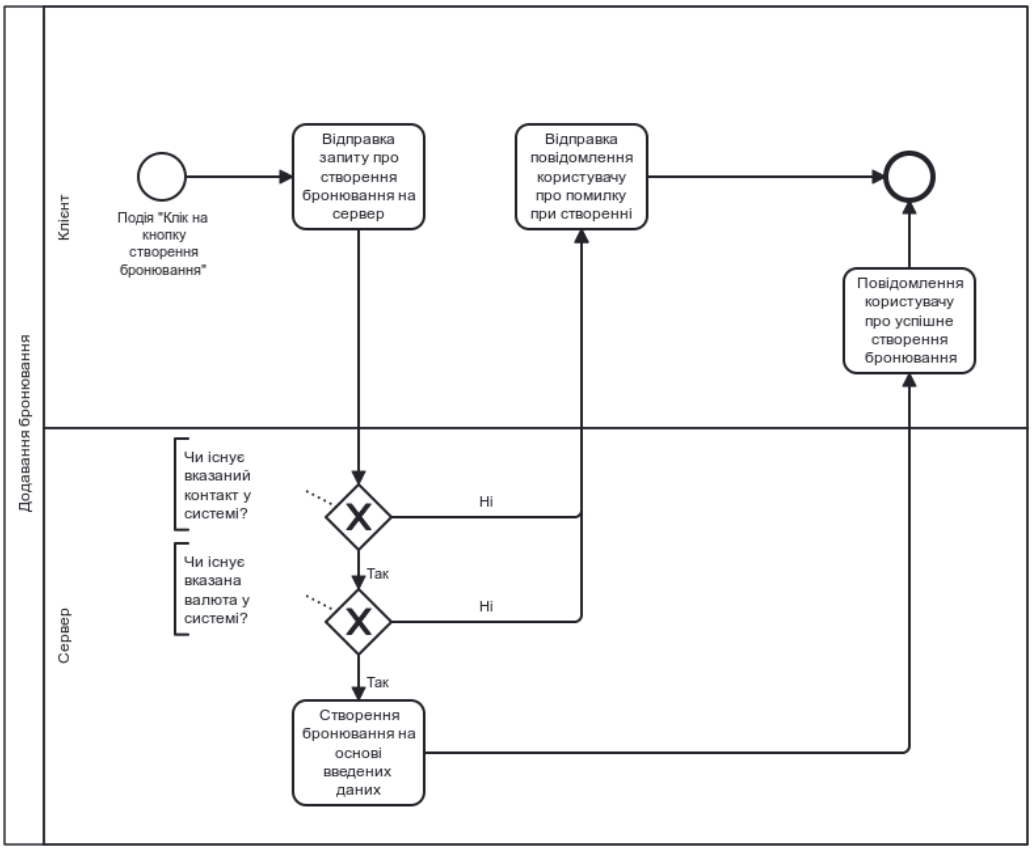


Рисунок 1.9 – Бізнес-процес створення бронювання у системі

Для створення валюти у системі, користувач повинен перейти на сторінку валют і натиснути на кнопку “Створити валюту”. Після чого, він вводить всі необхідні поля і натискає на кнопку “Створити валюту”. Клієнт відправляє дані на сервер, де у свою чергу він робить запит до Національного банку України і отримує відповідний курс валюти. Створює валюту, обчислює курс в залежності від бажаного відсотка прибутковості обмінного пункту, створює курс і резерв для цієї валюти. Бізнес-процес створення валюти у системі зображений на рисунку 1.10.

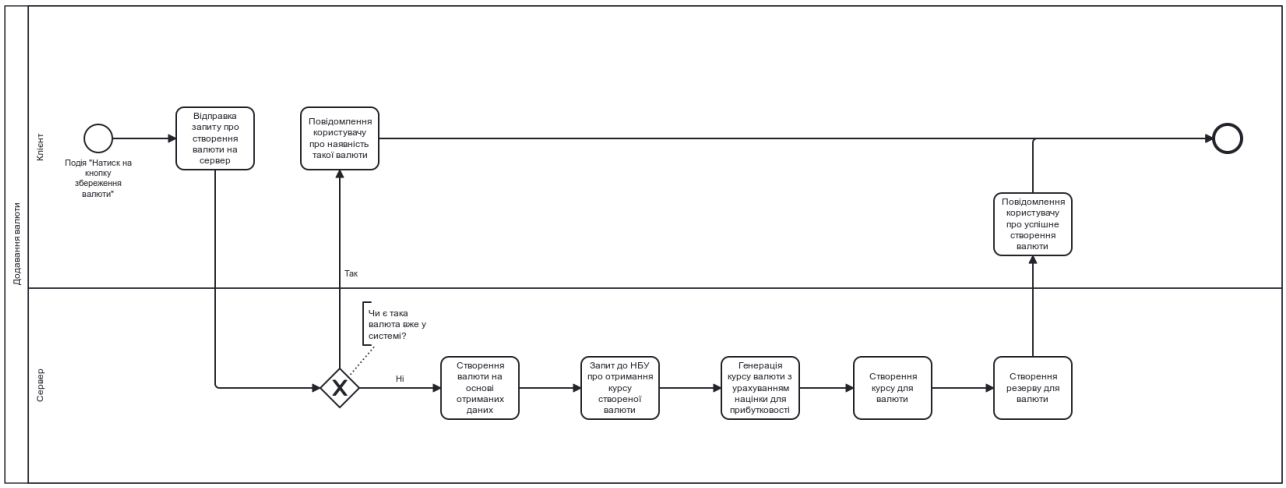


Рисунок 1.10 – Бізнес-процес створення валюти у системі

Для видалення валюти з системи, користувач повинен зайти на сторінку валют і натиснути на кнопку “Видалити валюту”. Після чого, клієнт робить запит на сервер, де він видаляє курс для цієї валюти і резерви. Потім відбувається видалення самої валюти. Бізнес-процес видалення валюти з системи зображений на рисунку 1.11.

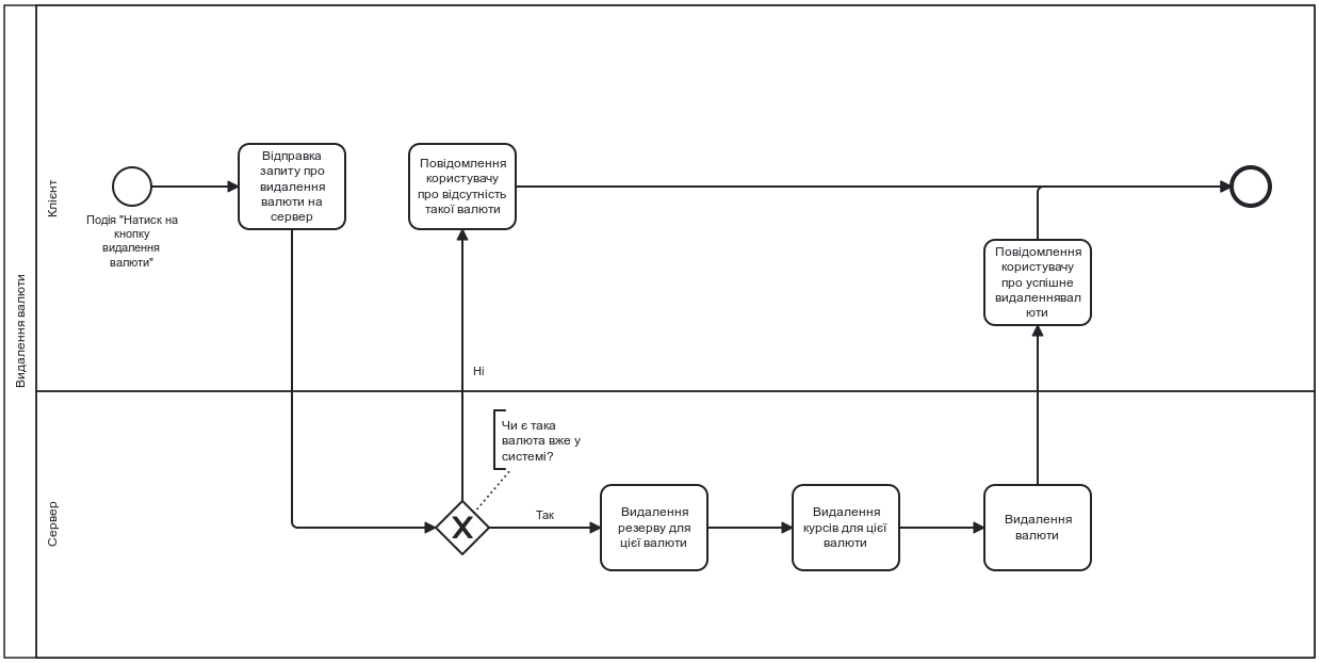


Рисунок 1.11 – Бізнес-процес видалення валюти з системи

1.3 Постановка задачі

Метою розробки є покращення та автоматизація обліку роботи пункту обміну валюти за рахунок розроблення застосунку для візуалізації та обліку транзакцій, створення бронювань обміну валюти, а також для взаємодії з системою телеграм.

Клієнт матиме можливість створити бронювання на певну валюту, переглянути актуальні курси, створити власний контакт. Працівникам і власникам бізнесу буде доступна можливість додавати нові валюти, оновлювати та видаляти їх, встановлювати курси, проводити транзакції та управляти резервами, а також обробляти бронювання.

Висновки до розділу

Автоматизація пункту обміну валют через вебзастосунок у середовищі Telegram та створення спеціалізованого вебсайту для управління валютним обміном - це важливі та перспективні рішення для оптимізації діяльності пункту обміну валют. Чат-боти в Telegram спрощують процес обміну валют, забезпечуючи автоматизацію ручних операцій та зменшуючи можливість помилок. Вони також дозволяють власникам залучати більше клієнтів та підвищувати продажі завдяки швидкій відповіді на запити та роботі в неробочий час.

У той же час, спеціалізований вебсайт надає власникам більш широкий набір інструментів для управління бізнесом. Він дозволяє створювати валюти, керувати курсами, відстежувати транзакції, забезпечувати можливість бронювання та управляти резервами валюти. Ці функції сприяють оптимізації робочих процесів та підвищують рівень обслуговування як для власників, так і для клієнтів.

Таким чином, обидва рішення представляють собою ефективні засоби автоматизації та управління пунктом обміну валют, забезпечуючи зручність, ефективність та надійність обміну валют.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Метою програмного забезпечення є покращення шляхом зменшення рутинних операцій і автоматизації обліку роботи пункту обміну валюти. Більше функцій можна побачити на рисунках 2.1-2.4.

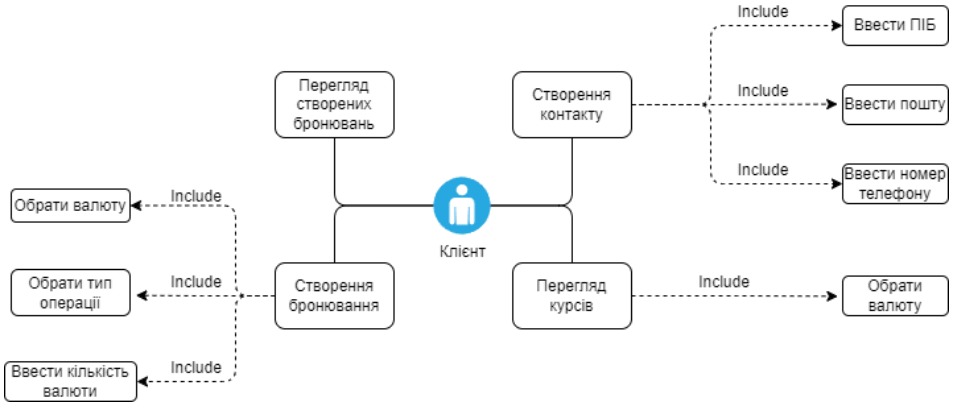


Рисунок 2.1 – Діаграма варіантів використання клієнта

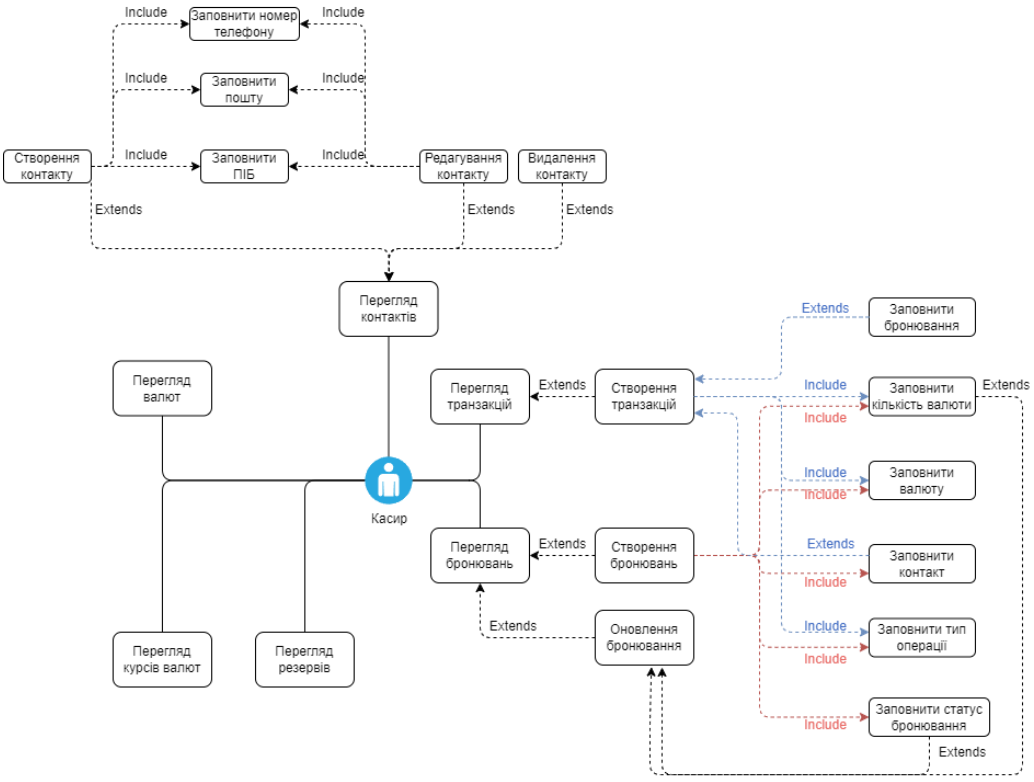


Рисунок 2.2 – Діаграма варіантів використання касира

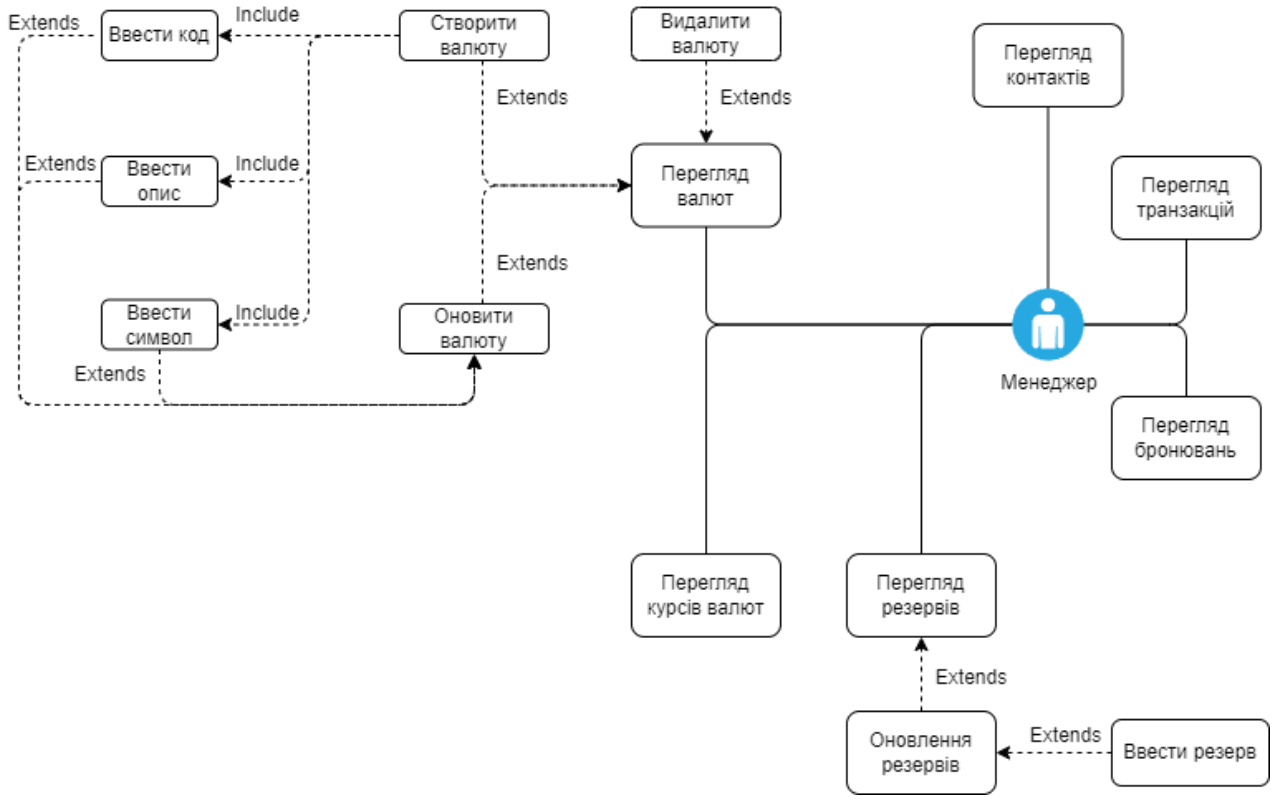


Рисунок 2.3 – Діаграма варіантів використання менеджера

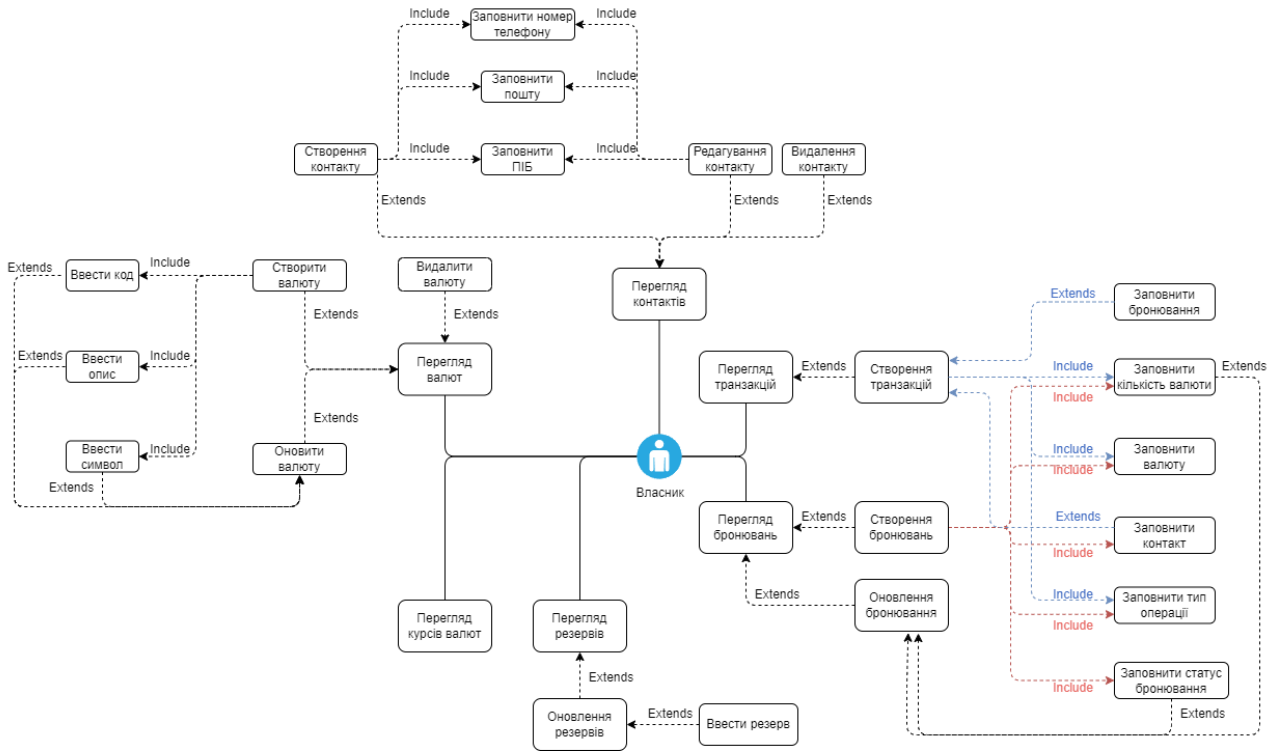


Рисунок 2.4 – Діаграма варіантів використання власника

В таблицях 2.1 - 2.19 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-1

Use case name	Створити контакт
Use case ID	UC-01
Goals	Створити контакт клієнта у базі даних
Actors	Клієнт
Trigger	Клієнт бажає внести свої дані у систему
Pre-conditions	-
Flow of Events	Клієнт настискає на кнопку “Мій контакт”, після чого, бот запитує клієнта його ПІБ, номер телефону та пошту. Після успішного створення контакту, користувач побачить повідомлення у боті.
Extension	Якщо клієнт відмовиться від створення контакту, то його поверне до меню налаштування.
Post-Condition	Клієнт успішно додав свій контакт у систему

Таблиця 2.2 – Варіант використання UC-2

Use case name	Створити бронювання
Use case ID	UC-02
Goals	Створити бронювання для певної валюти у системі
Actors	Клієнт
Trigger	Клієнт бажає створити бронювання у системі
Pre-conditions	Клієнт має створити контакт у системі
Flow of Events	Клієнт натискає на кнопку “Бронювання”, після чого, бот відкриває меню бронювань. Там він натискає на кнопку “Нове бронювання”. Після чого, бот, відправить список валют, з якими працює пункт обміну. Після того, як клієнт обрав необхідні валюти, система відправляє курс і 2 кнопки “Погодитись”, “Відхилити”. Після натискання кнопки “Погодитись” – система просить клієнта ввести кількість валюти для обміну, після чого, буде надіслано повідомлення про

	успішне створення бронювання. При натисканні на кнопку “Відхилити” – клієнт отримає інформацію, що створення його бронювання – скасовано і бот відкриє головне меню.

Продовження таблиці 2.2

Extension	Якщо клієнт натисне на кнопку “Погодитись” – бронювання успішно створиться у системі. При настисканні кнопки “Відхилити” – створення бронювання буде скасовано.
Post-Condition	Користувач успішно створив бронювання у системі.

Таблиця 2.3 – Варіант використання UC-3

Use case name	Переглянути створені бронювання
Use case ID	UC-03
Goals	Отримати з бази даних бронювання по певному користувачу
Actors	Клієнт
Trigger	Клієнт бажає переглянути свої бронювання
Pre-conditions	Клієнт повинен мати контакт у системі
Flow of Events	Клієнт натискає на кнопку “Бронювання”, де обирає пункт “Мої бронювання”. Після чого, клієнту буде виведено список його бронювань.
Extension	-
Post-Condition	Клієнт успішно переглянув список своїх бронювань

Таблиця 2.4 – Варіант використання UC-4

Use case name	Перегляд курсів валют з якими працює обмінник
Use case ID	UC-04
Goals	Отримати з бази даних інформацію про курси влют
Actors	Клієнт
Trigger	Клієнт бажає переглянути інформацію про валюти та курси з якими працює обмінний пункт
Pre-conditions	-
Flow of Events	Клієнт натискає на кнопку “Курси”, після чого, йому буде наданий список валюти, які приймає обмінник. Користувач обирає необхідну. Як тільки клієнт обере валюту для обміну,

	система надішле користувачу курс.
Extension	-
Post-Condition	Клієнт успішно отримав інформацію про курси і валюти

Таблиця 2.5 – Варіант використання UC-5

Use case name	Створення валюти
Use case ID	UC-05
Goals	Створити валюту у базі даних
Actors	Менеджер, Власник
Trigger	Менеджер, Власник має необхідність додати нову валюту у системі
Pre-conditions	Менеджер, Власник повинен бути авторизованим у системі
Flow of Events	Менеджер, Власник натискає на пункт валют і потім на кнопку “Add currencies”. Відкриється модальне вікно, де потрібно заповнити код валюти, опис та спеціальний символ. Після їх заповнення, менеджер або власник натискає на кнопку “Save” і валюта створюється у системі.
Extension	Якщо менеджер або власник не заповнить якесь поле з модального вікна він не зможе створити валюту.
Post-Condition	Менеджер або власник успішно створив валюту

Таблиця 2.6 – Варіант використання UC-6

Use case name	Перегляд валют
Use case ID	UC-06
Goals	Дістати з бази даних наявні валюти
Actors	Менеджер, Власник, Касир
Trigger	Користувач бажає переглянути валюти
Pre-conditions	Користувач повинен бути авторизованим у системі.
Flow of Events	Користувач натискає на пункт “Currencies” і йому буде

	відображено на сторінці список наявних валют
Extension	Якщо валют у системі немає, список буде порожнім
Post-Condition	Користувач успішно переглянув валюти

Таблиця 2.7 – Варіант використання UC-7

Use case name	Видалення валюти
Use case ID	UC-07
Goals	Видалити валюту з бази даних
Actors	Менеджер, Власник
Trigger	Користувач бажає видалити валюту

Провження таблиці 2.7

Pre-conditions	Користувач повинен бути авторизованим у системі.
Flow of Events	Користувач обирає пункт “Currencies” поруч з валютою, натискає на кнопку “Delete”.
Extension	-
Post-Condition	Користувач успішно видалив замовлення і був проінформований про це повідомленням у правому нижньому кутку екрану.

Таблиця 2.8 – Варіант використання UC-8

Use case name	Оновлення валюти у системі
Use case ID	UC-08
Goals	Оновити валюту у базі даних
Actors	Менеджер, Власник
Trigger	Користувач має необхідність внести зміни у валюту
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	Користувач обирає пункт “Currencies” і поруч з необхідною валютою, натискає на кнопку “Modify”. Після чого буде виведено модальне вікно, у якому він зможе обрати зміни і внести їх. Після чого він натискає на кнопку “Save” і буде проінформований про це повідомленням у правому нижньому кутку екрану про успішне оновлення валюти.

Extension	Якщо користувач залишить пусті поля, система видасть помилку про збереження валюти і не дасть йому її оновити.
Post-Condition	Користувач успішно оновив валюту у системі

Таблиця 2.9 – Варіант використання UC-9

Use case name	Створення курсу для валютної пари
Use case ID	UC-09
Goals	Створити курс для валютної пари у системі
Actors	Менеджер, Власник
Trigger	Користувач бажає створити курс для валютної пари
Pre-conditions	Користувач повинен бути авторизований і у системі повинно бути, хочаб 2 валюти

Продовження таблиці 2.9

Flow of Events	Користувач з головного меню обирає пункт “Rates” і потім “Create”. Система генерує степпер, де користувач повинен заповнити всі поля. Натискає на кнопку “Accepted currency” і після чого, йому буде наданий список доступних валют. Після того, як він обрав, натискає на кнопку “Returned currency” – де йому обираються інші валюти. Далі заповнює поле “Rate” і натискає на кнопку “Next”. Після чого, йому будуть виведені всі його дані, які він заповнив, але без можливості редагування. Після натискання на кнопку “Submit” – система створює курс для вказаної валютної пари.
Extension	Якщо користувач не заповнить всі поля, система не дасть створити курс. Якщо користувачу необхідно внести зміни у введених даних, він натискає на кнопку “Previous” і редагує.
Post-Condition	Користувач успішно створив валютну пару

Таблиця 2.10 – Варіант використання UC-10

Use case name	Оновлення курсу валют
Use case ID	UC-10

Goals	Оновити курс валют для створеної пари
Actors	Власник, менеджер
Trigger	Потрібно актуалізувати курс для певної пари
Pre-conditions	Користувач має бути авторизований у системі
Flow of Events	Користувач натискає на пункт “Rates” з головного меню і обирає пункт “View”. Після чого, біля необхідної валюти, користувач натискає на кнопку “Modify” і йому буде відображене модальне вікно з полем курсу, яке потрібно змінити. Після зміни – користувач буде проінформований про це повідомленням у правому нижньому кутку екрану.
Extension	Якщо користувач натисне на кнопку “Cancel” – зміни відміняться. Якщо користувач залишить поле курсу пустим, система не дасть оновити курс.
Post-Condition	Користувач оновив валютний курс

Таблиця 2.11 - Варіант використання UC-11

Use case name	Перегляд активних курсів
Use case ID	UC-11
Goals	Отримати список актуальних курсів
Actors	Касир, менеджер, власник
Trigger	Користувач бажає отримати список активних курсів
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	У головному меню користувач обирає пункт “Rates”, після чого, натискає на кнопку “View”. Далі система виведе активні курси валют.
Extension	Якщо курсів валют у системі немає, список буде порожнім.
Post-Condition	Користувач отримав список доступних курсів

Таблиця 2.12 - Варіант використання UC-12

Use case name	Створення транзакції
Use case ID	UC-12
Goals	Створити транзакцію у базі даних
Actors	Касир, менеджер, власник
Trigger	Касир, менеджер або власник бажає зафіксувати процесс обміну
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	В головному меню користувач обирає пункт Transactions. Після чого, повинен натиснути на кнопку “Create”. Система відобразить степпер, де на першому кроці він повинен заповнити поля: обрати валютний курс, кількість валюти, яка була видана. На другому кроці, користувач обирає контакта, якому видано валюту і бронювання, якщо валюта видавалася за бронюванням. На кроці 3, йому виводяться всі заблоковані поля, де він перевіряє правильність вводу даних. Після чого, натискає на кнопку “Submit”. Система відображає про успішне створення транзакції повідомлення у правому нижньому кутку екрану..
Extension	Якщо користувач не введе якісь поля, система проінформує користувача про не повноту введених даних.
Post-Condition	Користувач створив транзакцію у системі

Таблиця 2.13 - Варіант використання UC-13

Use case name	Перегляд створених транзакцій
Use case ID	UC-13
Goals	Переглянути транзакції у системі
Actors	Власник, менеджер, касир
Trigger	Користувач бажає переглянути проведені транзакції
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	Користувач у головному меню натискає на пункт “Transactions” и обирає підпункт “View”. Після чого, відкриється список з усіма транзакціями проведенеми.
Extension	-
Post-Condition	Користувач переглянув транзакції

Таблиця 2.14 – Варіант використання UC-14

Use case name	Створення бронювань
Use case ID	UC-14
Goals	Створити бронювання
Actors	Власник, менеджер, касир
Trigger	Користувач бажає створити нове бронювання
Pre-conditions	Користувач має бути авторизованим у системі
Flow of Events	Користувач у головному меню обирає пункт “Reservations” і підпункт “Create”. Після чого, система відправляє йому степпер, де на першому кроці, він повинен заповнити валютну пару і кількість забронюваної валюти і контакт, по якому створюється бронювання. На наступному кроці відображається всі заповнені поля без можливості редагування для перевірки введених значень. Після цього, користувач натискає на кнопку “Submit” і бронювання створюється у системі.
Extension	Якщо користувач пропустить якесь поле, система не дасть йому створити бронювання і покаже повідомлення у правому нижньому кутку екрану з помилкою.
Post-Condition	Користувач створив бронювання

Таблиця 2.15 – Варіант використання UC-15

Use case name	Оновлення бронювань
Use case ID	UC-15

Продовження таблиці 2.15

Goals	Оновити бронювання
Actors	Власник, менеджер, касир
Trigger	Користувач бажає оновити бронювання
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	Користувач у головному меню натискає на кнопку “Reservations” і там обирає пункт “View”. Потім, біля необхідного бронювання натискає кнопку “modify”. Після чого, вводить дані які хоче змінити і натискає на кнопку save.
Extension	Якщо користувач введе пусті дані, система не дасть йому

	оновити бронювання. При натисканні на cancel, зміни скасовуються і вікно закривається
Post-Condition	Користувач успішно оновив бронювання

Таблиця 2.16 – Варіант використання UC-16

Use case name	Видалення бронювань
Use case ID	UC-16
Goals	Видалити бронювання
Actors	Менеджер, власник, касир
Trigger	Користувач бажає відминити бронювання
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	Користувач у головному меню обирає пункт “Reservation” і далі переходить на вкладку “View”. Після чого, біля необхідного бронювання, натискає на кнопку “delete”. Після чого, бронювання видалиться і користувач отримає сповіщення.
Extension	-
Post-Condition	Адміністратор успішно видалив бронювання

Таблиця 2.17 – Варіант використання UC-17

Use case name	Створення контакту
Use case ID	UC-17
Goals	Створити контакт у системі
Actors	Менеджер, касир, власник
Trigger	Користувач бажає створити нового контакта у системі
Pre-conditions	Користувач має бути авторизований у системі

Продовження таблиці 2.17

Flow of Events	Користувач у головному меню натискає на пункт “Contacts” де потім натискає на кнопку “add contact” після чого, заповнює всі поля і натискає на кнопку “save”
Extension	Якщо користувач не заповнив усі поля, система не дасть створити контакта і проінформує його про це.
Post-Condition	Користувач успішно створив контакт

Таблиця 2.18 – Варіант використання UC-18

Use case name	Оновлення контакту
Use case ID	UC-18
Goals	Оновити створений контакт
Actors	Касир, менеджер, власник
Trigger	Користувач бажає оновити контакт
Pre-conditions	Користувач має бути авторизований у системі
Flow of Events	Користувач у головному меню натискає на кнопку “Contacts”, після чого, йому будуть відображені всі контакти системи. Біля необхідного – натискає на кнопку “modify”, потім йому буде відображене модальне вікно, у якому він повинен змінити необхідні дані. Після виконання цих дій, натискає на кнопку “Save”.
Extension	Якщо користувач не заповнив усі поля або залишив якесь пустим, система не дасть оновити контакт.
Post-Condition	Користувач оновив контакт

Таблиця 2.19 – Варіант використання UC-19

Use case name	Перегляд контактів
Use case ID	UC-19
Goals	Переглянути усі контакти системи
Actors	Касир, менеджер, власник
Trigger	Користувач бажає переглянути всі контакти у системі
Pre-conditions	Користувач повинен бути авторизованим у системі
Flow of Events	У головному меню користувач обирає пункт “Contacts”. Після його натискання, система покаже йому всі контакти, які у ній є.
Extension	Якщо контактів у системи немає, буде відображений пустий список.

Продовження таблиці 2.19

Post-Condition	Користувач успішно переглянув створені контакти

2.2 Аналіз системних вимог

Програмне забезпечення повинно функціонувати на сучасних персональних комп'ютерах та серверних системах. Рекомендовані характеристики можуть варіюватись в залежності від очікуваного навантаження, проте система повинна бути оптимізована для роботи навіть при значній кількості одночасних користувачів завдяки використанню Redis і PostgreSQL для ефективного керування даними.

Мінімальна конфігурація технічних засобів:

– процесор: Intel Core i3 чи еквівалентний;

– об'єм ОЗП: 8 Гб;

– жорсткий диск: 50 Гб вільного простору;

– підключення до мережі інтернет.

Рекомендована конфігурація технічних засобів:

– процесор: Intel Core i5 чи еквівалентний;

– об'єм ОЗП: 16 Гб;

– жорсткий диск: 100 Гб SSD;

– підключення до мережі інтернет.

Програмне забезпечення повинно працювати під управлінням наступних операційних систем:

– windows 10 чи пізніше;

– macOS 10.14 чи пізніше;

– unix-подібні системи (наприклад, Ubuntu 18.04 чи пізніше).

Процесор Intel Core i3 чи еквівалентний забезпечує достатню продуктивність для роботи кількох ASP.NET застосунків, фронтенду на Vue.js, та СУБД Redis і PostgreSQL. Для рекомендацій використовується Intel Core i5, оскільки це дозволить системі працювати більш стабільно при збільшеному навантаженні та додаткових сервісах.

8 Гб оперативної пам'яті є мінімальною необхідністю для забезпечення стабільної роботи всіх компонентів системи. 16 Гб оперативної пам'яті рекомендовано для кращої продуктивності та можливості масштабування.

50 Гб вільного місця достатньо для розгортання всіх необхідних компонентів та зберігання базових даних. SSD з 100 Гб місця забезпечить швидший доступ до даних та підвищить загальну швидкість роботи системи.

Надійне підключення до Інтернету є необхідним для забезпечення стабільної роботи вебзастосунків та баз даних.

Ці системні вимоги забезпечують оптимальний баланс між доступністю, продуктивністю і стабільністю роботи програмного забезпечення. Вони враховують можливості різних апаратних та програмних платформ, забезпечуючи ефективну роботу у широкому спектрі умов та середовищ.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.20 наведено загальну модель вимог, а в таблицях 2.21 – 2.37 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.5.

Таблиця 2.20 – Модель вимог у загальному вигляді

Id	Full description	Code	Priority	Workload	Risk	Status
1.	Авторизація \ аутентифікація у системі	REQ-1	1	0	-	Approved
2.	Перегляд списку валют	REQ-2	1	0	-	Approved
2.1	Додавання валюти	REQ-3	1	0	-	Approved

Продовження таблиці 2.20

2.2	Оновлення інформації про валюту	REQ-4	2	0	-	Approved
2.3	Видалення валюти	REQ-5	-	0	-	Approved

3.1	Додавання курсів валют	REQ-6	1	0	-	Approved
3.3	Видалення курсу	REQ-8	-	0	-	Approved
3.4	Перегляд списку курсів	REQ-9	1	0	-	Approved
4.1	Перегляд бронювань	REQ-10	1	0	-	Approved
4.2	Створення бронювання	REQ-11	1	0	-	Approved
4.3	Оновлення інформації про бронювання	REQ-12	2	0	-	Approved
5.1	Перегляд транзакцій	REQ-13	1	0	-	Approved
5.2	Створення нових транзакцій	REQ-14	1	0	-	Approved
6.1	Перегляд контактів	REQ-16	2	0	-	Approved
6.2	Створення нового контакту	REQ-17	2	0	-	Approved
6.3	Оновлення інформації про контакта	REQ-18	-	0	-	Approved
7	Виведення повідомлення користувачу	REQ-19	1	0	-	Approved

Таблиця 2.21 – Функціональна вимога FR-1

Назва	Авторизація \ аутентифікація у системі
Опис	Система повинна надавати можливість авторизуватись\аутентифікуватись у системі

Таблиця 2.22 – Функціональна вимога FR-2

Назва	Перегляд списку валют
Опис	Система повинна надавати можливість переглядати валюти які є у системі

Таблиця 2.23 – Функціональна вимога FR-3

Назва	Додавання валюти
Опис	Система повинна надавати можливість додати нову валюту до системи

Таблиця 2.24 – Функціональна вимога FR-4

Назва	Оновлення інформації про валюту
Опис	Система повинна надавати можливість оновлювати інформацію про валюту користувачем

Таблиця 2.25 – Функціональна вимога FR-5

Назва	Видалення валюти
Опис	Система повинна надавати можливість користувачеві видалити валюту.

Таблиця 2.26 – Функціональна вимога FR-7

Назва	Перегляд бронювань
Опис	Система повинна надавати можливість переглядати інформацію про створені бронювання

Таблиця 2.27 – Функціональна вимога FR-8

Назва	Створення бронювання
Опис	Система повинна надавати можливість користувачеві створити нове бронювання

Таблиця 2.28 – Функціональна вимога FR-9

Назва	Оновлення інформації про бронювання
Опис	Система повинна надавати можливість користувачеві оновити інформацію по певному бронюванню

Таблиця 2.29 – Функціональна вимога FR-10

Назва	Перегляд транзакцій
Опис	Система повинна надавати можливість користувачеві переглянути транзакції пункту обмінну.

Таблиця 2.30 – Функціональна вимога FR-11

Назва	Створення нових транзакцій
Опис	Система повинна надавати можливість користувачеві створювати нові транзакції

Таблиця 2.31 – Функціональна вимога FR-13

Назва	Перегляд контактів
Опис	Система повинна надавати можливість користувачу перегляди інформацію про створених контактів

Таблиця 2.32 – Функціональна вимога FR-14

Назва	Створення нового контакту
Опис	Система повинна надавати можливість користувачеві створити нового контакта

Таблиця 2.33 – Функціональна вимога FR-15

Назва	Оновлення інформації про контакт
Опис	Система повинна надавати можливість користувачеві оновлювати інформацію певного контакту

Таблиця 2.34 – Функціональна вимога FR-16

Назва	Створення нового курсу
Опис	Система повинна надавати можливість користувачеві створювати новий курс валют

Таблиця 2.35 – Функціональна вимога FR-17

Назва	Оновлення інформації про курс валюти
Опис	Система повинна надавати можливість користувачеві оновлювати інформацію про певний курс валюти

Таблиця 2.36 – Функціональна вимога FR-18

Назва	Видалення курсу
Опис	Система повинна надавати можливість користувачеві видаляти необхідний курс валют

Таблиця 2.37 – Функціональна вимога FR-19

Назва	Виведення повідомлення користувачу
Опис	Система повинна відображати повідомлення користувачу про успішне чи неуспішне виконання дії.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19
UC-1	+																+		+
UC-2	+	+							+		+					+			+
UC-3	+								+	+									+
UC-4	+								+			+							+
UC-5	+	+	+																+
UC-6	+	+																	+
UC-7	+				+														+
UC-8	+	+		+															+
UC-9	+	+					+		+										+
UC-10	+	+				+		+	+										+
UC-11	+	+							+										+
UC-12	+	+							+				+	+					+
UC-13	+	+							+				+		+	+			+
UC-14	+	+							+	+						+			+
UC-15	+	+							+	+						+			+
UC-16	+									+									+
UC-17	+																+		+
UC-18	+															+		+	+
UC-19	+															+			+

Рисунок 2.5 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Вебзастосунок передбачає використання користувацьких акаунтів та адміністративних прав, тому необхідно гарантувати їх безпеку та конфіденційність. Система повинна забезпечувати надійний захист даних користувачів від несанкціонованого доступу, використовуючи сучасні методи аутентифікації та авторизації.

Також важливо, щоб система управління базами даних (СУБД) могла ефективно обробляти можливі високі навантаження, оскільки вебзастосунок передбачає постійну взаємодію з базами даних. Це вимагає від СУБД високої продуктивності та масштабованості, щоб задовольнити потреби всіх користувачів, навіть під час пікових навантажень.

Додаткові нефункціональні вимоги.

Продуктивність: система повинна забезпечувати швидкий відгук на запити користувачів. Час завантаження сторінок не повинен перевищувати 2 секунд при нормальному навантаженні.

Масштабованість: архітектура системи повинна дозволяти горизонтальне та вертикальне масштабування для обробки зростаючої кількості користувачів і даних.

Надійність: система повинна забезпечувати безперервну роботу із часом безвідмовної роботи (Uptime) не менше 99.9%.

Безпека: застосунок повинен відповідати сучасним стандартам кібербезпеки, включаючи захист від SQL-ін'єкцій, XSS-атак та інших видів загроз. Всі дані, що передаються між клієнтом та сервером, повинні бути зашифровані.

Конфіденційність: персональні дані користувачів повинні бути захищені відповідно до стандартів GDPR або інших відповідних регуляцій.

Підтримка та обслуговування: система повинна мати можливість легкого оновлення і обслуговування без значних перерв у роботі.

Сумісність: програмне забезпечення повинно працювати на різних платформах і браузерах, включаючи мобільні пристрої.

Ці вимоги допоможуть забезпечити стабільну, безпечну і ефективну роботу вебзастосунку, задовольняючи при цьому високі очікування користувачів щодо продуктивності та безпеки.

Висновки до розділу

У цьому розділі представлено варіанти використання вебзастосунку, проведено детальний аналіз системних вимог, а також розроблено функціональні та нефункціональні вимоги. Розглянуто мінімальні та рекомендовані конфігурації апаратних засобів, необхідні для стабільної та ефективної роботи системи, яка включає кілька ASP.NET застосунків, фронтенд на Vue.js, бази даних Redis та PostgreSQL.

Проаналізовано необхідність підтримки різноманітних операційних систем, що дозволяє залучити максимальну аудиторію користувачів. Зокрема, програмне забезпечення повинно бути сумісним з Windows, MacOS, Unix, Android та iOS, забезпечуючи універсальність та зручність використання для користувачів із різними пристроями..

Таким чином, надано всебічний огляд системних вимог та функціональних можливостей вебзастосунку. Цей аналіз забезпечує фундамент для створення надійного, ефективного та зручного для користувачів програмного забезпечення. Описані вимоги дозволяють забезпечити високу продуктивність, надійність та безпеку системи, що є критичними для її успішного впровадження та експлуатації.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для розробленої системи обрано клієнт-серверну архітектуру. Архітектура клієнт-сервер описує систему, яка розміщує, передає та керує основними ресурсами та послугами, запитаними клієнтом. У цій моделі всі запити та послуги проходять через мережу, тому її ще називають моделлю мережевих обчислень або клієнт-серверною мережею.

Модель клієнт-сервер, відома також як архітектура клієнт-сервер, являє собою мережевий застосунок, що розподіляє завдання та навантаження між клієнтами і серверами, які можуть бути частинами однієї системи або з'єднаними через комп'ютерну мережу.

Архітектура клієнт-сервер зазвичай включає робочі станції, персональні комп’ютери або інші пристрої користувачів, підключені до центрального сервера через Інтернет або іншу мережу. Клієнт надсилає запит на дані, сервер приймає та обробляє цей запит, після чого повертає дані клієнту.

Цю модель також називають мережею клієнт-сервер або моделлю мережевих обчислень. На рисунку 3.1 зображено зв’язок між клієнтом та сервером, реалізованим у системі.

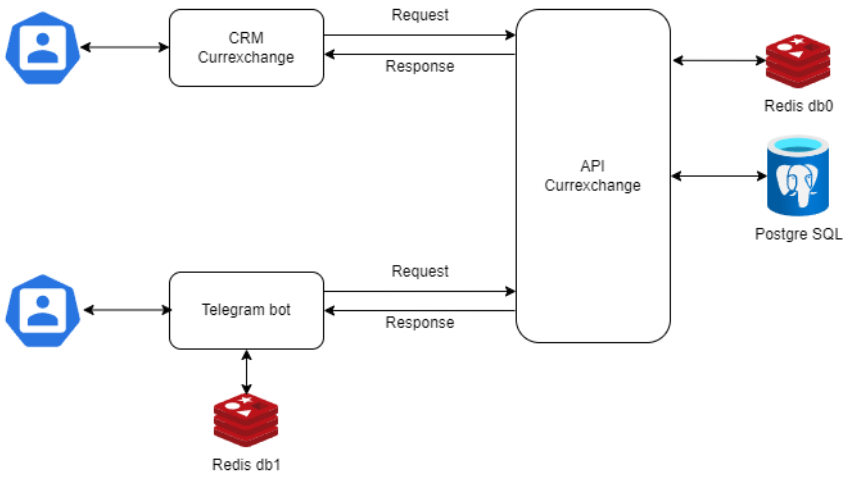


Рисунок 3.1 – Клієнт-серверна архітектура вебзастосунку

Архітектура застосунку побудована на базі MVC (Model-View-Controller) шаблону, який використовується для розробки програмного забезпечення з чітким розподілом логіки додатка на три компоненти: модель, представлення і контроллер, що допомагає організувати код і полегшує його підтримку та розвиток. Використання цього шаблону надає легкість у тестуванні, через те, що всі компоненти відокремлені і їх можна тестувати окремо.

На рисунку 3.2 нижче зображено паттерн проєктування, який використовувався для побудови архітектури.

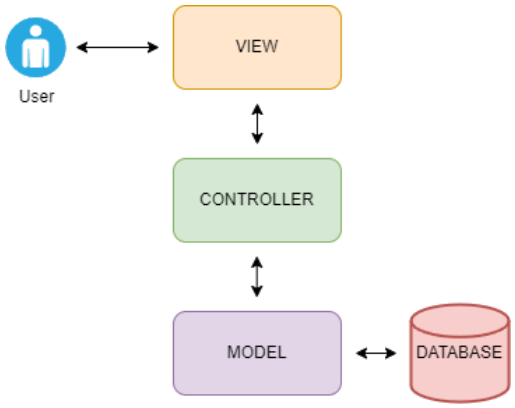


Рисунок 3.2 – MVC паттерн

Модель. Зберігає та управляє даними, виконує логіку бізнес-процесів. Інкапсулює дані додатку та правила їх обробки. Це можуть бути класи, що описують дані, які використовуються у додатку, моделі для бази даних, тощо.

Контроллер. Взаємодіє з користувачем через представлення. Обробляє введені дані, оновлює модель відповідно до дій користувача і визначає, яке представлення показати користувачеві. Обробляє HTTP-запити та викликає методи моделі та повертає відповіді.

Представлення. Відповідає за відображення даних користувачеві у зручному форматі. У вебдодатку представленням може бути HTML-сторінка, дані у вигляді JSON або XML.

У процесі розробки вебзастосунку використано стиль REST (Representational State Transfer). Це архітектурний стиль для взаємодії компонентів у розподіленій системі через мережу, який забезпечує спрощення і

підвищення ефективності архітектури. REST ґрунтується на певних принципах, які визначають, як компоненти повинні взаємодіяти один з одним.

Основна ідея REST полягає в тому, що ресурси системи ідентифікуються через URL, а їхнє маніпулювання здійснюється стандартними HTTP-методами: «GET» для отримання даних, «POST» для створення нових, «PUT» для оновлення існуючих і «DELETE» для видалення. Кожен ресурс має свій унікальний ідентифікатор, що дозволяє легко отримувати і змінювати інформацію.

Однією з основних переваг використання REST є його здатність спрощувати структуру системи, роблячи її легшою для розуміння і підтримки. Взаємодія між клієнтом і сервером у REST здійснюється через чітко визначені інтерфейси, що робить інтеграцію різних компонентів зручнішою.

Запити в REST-системах, які називаються «REST-запитами», передають дані як параметри HTTP-запитів.

Для вебслужб, які побудовані з урахуванням принципів REST, використовується термін «RESTful». Це означає, що такі служби відповідають правилам і стилю REST.

Таким чином, використання REST в системі забезпечило ефективну та спрощену архітектуру, що полегшує підтримку і масштабування вебзастосунку.

3.2 Обґрунтування вибору засобів розробки

Для створення вебзастосунку використано мову програмування C# разом з фреймворком ASP.NET Web API, а також JavaScript з бібліотеками Vue.js і Vuetify, HTML, та SCSS. Код був написаний у середовищах розробки Visual Studio та WebStorm. Нижче наведені обґрунтування вибору цих засобів розробки.

C# – це проста, сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона відзначається високою продуктивністю і гнучкістю, що робить її ідеальною для розробки складних веб-застосунків.

Основні переваги використання C# і ASP.NET Web API:

- подібність синтаксису до інших популярних мов програмування, таких як C++ і Java, що знижує криву навчання;
- об'єктно-орієнтований підхід забезпечує модульність і повторне використання коду;
- широка сфера застосування – від веб-додатків до ігор і системного програмного забезпечення;
- кросплатформність завдяки .NET Core, що дозволяє розгортати додатки на різних операційних системах (Windows, Linux, macOS);
- мова C# має вбудовані механізми захисту від поширених вразливостей, властивих низькорівневим мовам, таким як C і C++;
- зручність написання коду з інтелектуальними інструментами і автодоповненням у Visual Studio.

Для фронтенд-розробки обрано JavaScript разом з Vue.js та Vuetify.

Основні переваги цього вибору:

- фреймворк Vue.js – це прогресивний фреймворк для побудови інтерфейсів користувача, який відомий своєю простотою і гнучкістю. Він дозволяє створювати потужні односторінкові додатки з високою продуктивністю;
- vuetify – це бібліотека компонентів для Vue.js, яка забезпечує швидку розробку красивих і сучасних інтерфейсів, дотримуючись принципів матеріального дизайну;
- мова JavaScript – це універсальна мова для фронтенд-розробки, яка підтримується всіма сучасними браузерами. Її динамічність і велика кількість бібліотек роблять розробку швидкою та ефективною;
- мова розмітки HTML і SCSS – основи веб-розробки, що дозволяють створювати структуру сторінок і стилізувати їх відповідно до сучасних стандартів.

Використання Visual Studio та WebStorm. Visual Studio – це потужне інтегроване середовище розробки (IDE), розроблене Microsoft, яке надає безліч інструментів для розробки на C#. Основні переваги включають:

- розширені можливості налагодження і тестування, що забезпечують високу якість коду;
- інтеграція з Git і іншими системами контролю версій, що спрощує керування кодом;
- широка підтримка розширень, що дозволяє адаптувати IDE під конкретні потреби проєкту;

WebStorm – це IDE від JetBrains, спеціалізоване для веб-розробки з JavaScript. Його основні переваги:

- розширене автодоповнення коду і інтелектуальні підказки, що значно прискорюють розробку;
- вбудовані інструменти для роботи з фронтенд-фреймворками, такими як Vue.js;
- зручна інтеграція з популярними інструментами для тестування та налагодження коду.

Вибір СУБД

Для веб-застосунку обрано СУБД PostgreSQL та Redis. PostgreSQL використовується для зберігання основних даних, а Redis – для кешування і швидкого доступу до тимчасових даних. Для роботи з цими СУБД використано інструменти DataGrip і RedisInsight відповідно. Нижче наведено обґрунтування цього вибору.

Використання PostgreSQL. PostgreSQL – це потужна об’єктно-реляційна система управління базами даних з відкритим вихідним кодом. Вона відома своєю надійністю, функціональністю та відповідністю сучасним стандартам.

Основні переваги використання PostgreSQL:

- postgreSQL є безкоштовним і має активну спільноту розробників, що забезпечує регулярні оновлення та вдосконалення;

- включає роботу з транзакціями, підзапитами, тригерами та збереженими процедурами;
- дозволяє створювати власні типи даних, індекси та функції, що робить її дуже гнучкою;
- висока продуктивність і масштабованість забезпечує обробку великих обсягів даних і підтримку великих навантажень;

Сумісність з багатьма інструментами та фреймворками – легко інтегрується з різними мовами програмування та іншими технологіями.

Для управління та адміністрування PostgreSQL використовувався DataGrip – професійний інструмент для роботи з базами даних від JetBrains.

Основні переваги DataGrip:

- інтелектуальне автодоповнення коду підвищує продуктивність розробки;
- зручне управління схемами баз даних дозволяє легко створювати, змінювати та видаляти об'єкти баз даних;
- потужні можливості налагодження та тестування запитів включають профілювання запитів та аналіз продуктивності;
- підтримка роботи з декількома базами даних дозволяє одночасно працювати з різними СУБД, що зручно при складних архітектурах.

Використання Redis. Redis – це високопродуктивна система управління базами даних, яка працює в пам’яті і використовується для кешування та швидкого доступу до даних. Вона підтримує різні структури даних, такі як рядки, хеші, списки, множини та упорядковані множини.

Основні переваги використання Redis:

- надзвичайно висока швидкість обробки запитів завдяки зберіганню даних в оперативній пам'яті;
- підтримка різноманітних типів даних забезпечує гнучкість у роботі з даними;

- простота використання та налаштування дозволяє швидко інтегрувати Redis у будь-який проєкт;
- вбудована підтримка реплікації та стійкості до відмов забезпечує надійність і безперервну роботу системи;
- висока масштабованість дозволяє легко розширювати систему при зростанні навантаження.

Для моніторингу та управління Redis використовувався RedisInsight – інструмент з потужним інтерфейсом для аналізу і візуалізації даних. Основні переваги RedisInsight:

- зручний інтерфейс користувача полегшує роботу з даними і конфігураціями;
- потужні інструменти аналізу продуктивності дозволяють швидко знаходити і вирішувати проблеми;
- візуалізація даних у реальному часі надає наочне уявлення про роботу системи;
- підтримка всіх основних функцій Redis включає роботу з кластерами, реплікацією і налаштуваннями.
-

3.3 Конструювання програмного забезпечення

Для створення програмного забезпечення підвишення ефективності роботи пункту обміну валют використані сучасні технології, які забезпечують високу ефективність, надійність та зручність у використанні. У проєкті задіяні чотири основні ролі: власник, менеджер, касир та клієнт, кожна з яких має свої функціональні можливості та інтерфейс для взаємодії з системою.

Telegram Bot. Основним інтерфейсом для клієнтів є Telegram бот, розроблений за допомогою мови програмування C# та фреймворку ASP.NET. Telegram Bot API надає широкий набір інструментів та методів для створення інтерактивних ботів, що дозволяє ефективно взаємодіяти з користувачами,

обробляти їхні запити та виконувати різноманітні завдання. Бот реалізує функціонал створення бронювань та перегляду актуальних курсів валют.

Для кешування та збереження тимчасових даних використовується Redis. Це дозволяє швидко обробляти запити користувачів та забезпечує високу швидкодію бота. Використання Redis також дозволяє зберігати тимчасові дані, що важливо для забезпечення безперебійної роботи системи.

Адміністративний інтерфейс. Касир, менеджер та власник користуються адміністративним темплейтом Materio, який був модифікований під потреби проєкту. Materio є безкоштовним шаблоном, розробленим на основі технологій Vue.js та Vuetify, з використанням HTML, SCSS та JavaScript. Цей шаблон надає сучасний та інтуїтивно зрозумілий інтерфейс, який був розширений та адаптований для виконання всіх необхідних операцій:

- проведення транзакцій, касири можуть здійснювати операції з обміну валют, реєструвати їх у системі;
- створення валют, менеджери можуть додавати нові валюти до системи, налаштовувати курси та інші параметри;
- створення курсів, користувачі системи можуть створити курси під необхідні валюти;
- створення бронювань, користувач системи може створити бронювання під певний контакт.

СУБД. Для збереження даних була обрана реляційна СУБД PostgreSQL. Ця база даних забезпечує надійне збереження та швидкий доступ до інформації про користувачів, транзакції, курси валют та інші необхідні дані. Використання PostgreSQL дозволяє ефективно масштабувати систему та забезпечувати високу продуктивність навіть при великій кількості одночасних запитів.

Технології та методи:

1. C# та ASP.NET: використовуються для розробки серверної частини системи та Telegram бота. ASP.NET забезпечує зручність розробки та високу продуктивність вебдодатків.
2. Redis: використовується для кешування та збереження тимчасових даних, що дозволяє швидко обробляти запити та знижує навантаження на базу даних.
3. Vue.js та Vuetify: використовуються для розробки адміністративного інтерфейсу. Це сучасні фреймворки, що забезпечують зручність у розробці та високу якість користувацького інтерфейсу.
4. PostgreSQL: основна СУБД для збереження всіх постійних даних. PostgreSQL є потужною реляційною базою даних з багатим набором функцій для збереження та обробки даних.

Таким чином, поєднання цих технологій та методів дозволяє створити ефективне та надійне програмне забезпечення для пункту обміну валют, що забезпечує високу продуктивність, зручність у використанні та масштабованість рішення.

Опис таблиць СУБД наведено у рисунках 3.1-3.13. Схема СУБД наведена на рисунку 3.3.

Таблиця 3.1 – Опис таблиці Rates

Таблиця	Назва поля	Тип даних	Опис
Rates	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису

Продовження таблиці 3.1

	CurrencyId	uuid	Унікальний ідентифікатор базової валюти
	SellRate	numeric	Курс продажу
	BuyRate	numeric	Курс купівлі
	IsActive	boolean	Чи активний цей запис

Таблиця 3.2 – Опис таблиці Reservations

Таблиця	Назва поля	Тип даних	Опис
Reservations	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	ContactId	uuid	Унікальний ідентифікатор контаку
	RateId	uuid	Унікальний ідентифікатор курсу
	OperationId	uuid	Унікальний ідентифікатор операції
	TypeId	uuid	Унікальний ідентифікатор типу
	StatusId	uuid	Унікальний ідентифікатор статусу

Продовження таблиці 3.2

	Amount	numeric	Кількість зарезервованої валюти
	IsActive	boolean	Чи активний цей запис

Таблиця 3.3 – Опис таблиці Transactions

Таблиця	Назва поля	Тип даних	Опис
Transactions	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	ContactId	uuid	Унікальний ідентифікатор контакту
	RateLogId	uuid	Унікальний ідентифікатор курсу
	Amount	numeric	Кількість виданої валюти
	ReservationId	uuid	Унікальний ідентифікатор бронювання
	TypeId	uuid	Унікальний ідентифікатор типу
	OperationId	uuid	Унікальний ідентифікатор операції
	IsActive	boolean	Чи активний запис

Таблиця 3.4 – Опис таблиці Users

Таблиця	Назва поля	Тип даних	Опис
Users	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	ContactId	uuid	Унікальний ідентифікатор контакту
	RoleId	uuid	Унікальний ідентифікатор ролі
	IsActive	boolean	Чи активний запис

Таблиця 3.5 – Опис таблиці UserRoles

Таблиця	Назва поля	Тип даних	Опис
UserRoles	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	Name	text	Назва ролі

Таблиця 3.6 – Опис таблиці Contacts

Таблиця	Назва поля	Тип даних	Опис
Contacts	Id	uuid	Унікальний ідентифікатор

Продовження таблиці 3.6

	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	FullName	text	ПІБ контакту
	Email	text	Контактна пошта
	IsActive	boolean	Чи активний запис
	PhoneNumber	text	Контактний номер телефону
	IsBlacklist	boolean	Чи контакт у чорному списку

Таблиця 3.7 – Опис таблиці Funds

Таблиця	Назва поля	Тип даних	Опис
Funds	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	CurrencyId	uuid	Унікальний ідентифікатор валюти
	Amount	numeric	Кількість резерву
	IsActive	boolean	Чи активний запис

Таблиця 3.8 – Опис таблиці Currencies

Таблиця	Назва поля	Тип даних	Опис
Funds	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	BankGovId	int	Ідентифікатор валюти у національному банку України
	Code	text	Код валюти
	Description	text	Опис валюти
	IsActive	boolean	Чи активний запис
	Symbol	text	Символ валюти

Таблиця 3.9 – Опис таблиці IssuanceLogs

Таблиця	Назва поля	Тип даних	Опис
IssuanceLogs	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	ModifiedOn	timestamp	Дата модифікації запису
	CurrencyId	uuid	Унікальний ідентифікатор валюти

Продовження таблиці 3.9

	SellRate	numeric	Курс продажу
	BuyRate	numeric	Курс купівлі

Таблиця 3.10 – Опис таблиці ExchangeSettings

Таблиця	Назва поля	Тип даних	Опис
ExchangeSettings	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	PercantageTransactionIncome	numeric	Дохід за транзакцію у відсотках
	CurrencyId	uuid	Унікальний ідентифікатор базової валюти
	LowBalanceThreshold	numeric	Мінімальна кількість резерву
	HighTransactionThreshold	numeric	Максимальна транзакція

Таблиця 3.11 – Опис таблиці OperationTypes

Таблиця	Назва поля	Тип даних	Опис
OperationTypes	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	Name	text	Назва типу операції

Таблиця 3.12 – Опис таблиці ReservationStatuses

Таблиця	Назва поля	Тип даних	Опис
ReservationStatuses	Id	uuid	Унікальний ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	Name	text	Назва статусу бронювання

Таблиця 3.13 – Опис таблиці TransactionTypes

Таблиця	Назва поля	Тип даних	Опис
TransactionTypes	Id	uuid	Ідентифікатор запису
	CreatedOn	timestamp	Дата створення запису
	Name	text	Назва типу транзакції

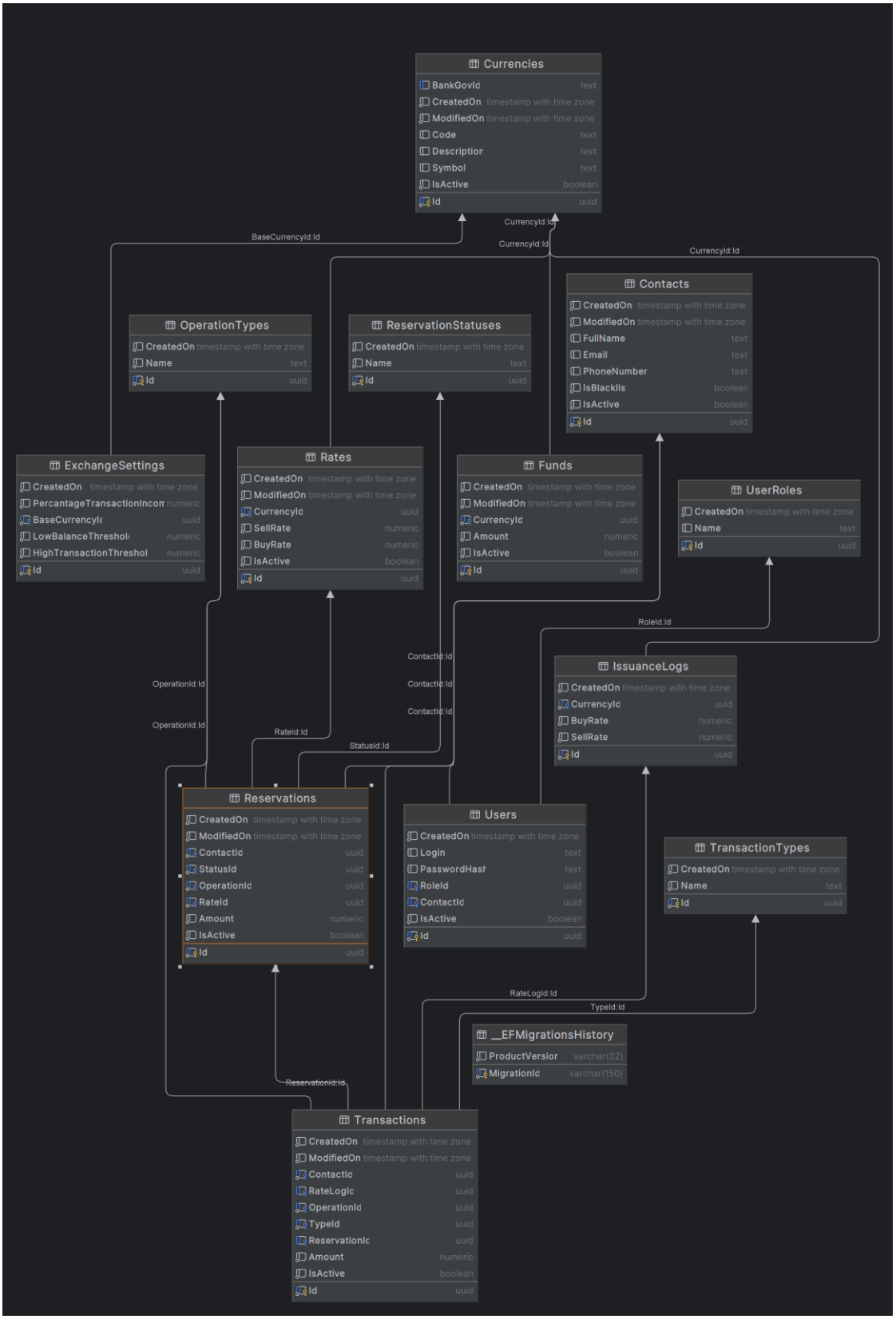


Рисунок 3.3 – Схема бази даних

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.14.

Таблиця 3.14 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio	Головне середовище розробки програмного забезпечення серверної частини проєкту.
2	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	Drawio	Draw.io є вебплатформою для створення діаграм, яка надає широкий спектр інструментів для візуалізації та моделювання різноманітних процесів, структур та даних. Ця утиліта дозволяє користувачам створювати професійні діаграми без необхідності у встановленні спеціалізованих програм, оскільки вона працює у веббраузері.
4	Redis Insight	Вебінтерфейс для адміністрування Redis, що дозволяє легко керувати ключами, переглядати дані та моніторити стан бази даних через веббраузер.
5	WebStorm	IDE для веброзробки від JetBrains, яке підтримує JavaScript, HTML, CSS та інші мови програмування для вебу.
6	DataGrip	Інтегроване середовище розробки (IDE) для роботи з базами даних від JetBrains.

Продовження таблиці 3.14

7	SourceTree	Графічний клієнт для роботи з Git та Mercurial.
8	Github	Вебплатформа для хостингу проєктів програмного забезпечення, яка базується на системі керування версіями Git.
9	ngrok	Інструмент, який надає можливість створення тунелю між локальним комп'ютером і Інтернетом.
10	Materio	Адміністративний шаблон для вебдодатків на Vue.js.

3.4 Аналіз безпеки даних

Під час розробки системи з Telegram ботом для покращення та автоматизації обліку пункту обміну валюти приділена значна увага безпеці даних.

Застосування Redis для зберігання тимчасових даних. Redis використовується для кешування та зберігання тимчасових даних, таких як сесійні дані клієнтів і станові повідомлення. Redis є потужним інструментом для швидкого доступу до даних із забезпеченням високої продуктивності. Для забезпечення безпеки Redis використовується разом з механізмами шифрування та обмеженням доступу до серверів Redis тільки для авторизованих додатків та сервісів.

Використання PostgreSQL для постійного зберігання даних. PostgreSQL використовується як основна база даних для зберігання даних про клієнтів, їх транзакції та інші важливі дані. PostgreSQL відомий своєю надійністю,

гнучкістю та можливістю використання різних механізмів автентифікації і шифрування для забезпечення конфіденційності і цілісності даних. Застосовується строгий контроль доступу до бази даних для забезпечення безпеки від несанкціонованого доступу.

Модифікація та безпека адміністративного інтерфейсу на основі Materio з Vue.js. Адміністративний інтерфейс на базі Materio, розроблений на Vue.js, використовується для управління та моніторингу всієї системи обміну валют. Впроваджені додаткові заходи безпеки, включаючи обмеження доступу до адміністративних функцій за різними ролями (власник, менеджер, касир), а також механізми аутентифікації з використанням токенів і сесійних ключів для забезпечення безпеки входу та діяльності користувачів.

Висновки до розділу

У розділі всебічно розглянуто ключові аспекти розробки програмного забезпечення для покращення та автоматизації обліку пункту обміну валюти за рахунок розроблення застосунку для візуалізації та обліку транзакцій, створення бронювань обміну валюти, а також для взаємодії з системою телеграм. Детальний аналіз охоплював кілька ключових підрозділів, що сприяло формуванню цілісного уявлення про процес створення, архітектурні рішення, вибір інструментів розробки, реалізацію програмного забезпечення та забезпечення безпеки даних. Вибір технологій, таких як C# з ASP.NET для реалізації бота та серверної частини системи, Vue.js для адміністративного інтерфейсу, був обгрунтований їхньою надійністю і можливістю забезпечення безпеки даних. Redis обрано для кешування та тимчасового зберігання даних, а PostgreSQL – для постійного зберігання інформації про транзакції та користувачів. Архітектурний підхід заснований на клієнт-серверній моделі забезпечив ефективну взаємодію користувачів системи з сервером. Особлива увага була приділена заходам безпеки, таким як контроль доступу до адміністративних функцій та використання механізмів аутентифікації для забезпечення конфіденційності та цілісності даних

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Оцінка якості програмного забезпечення та його ефективності є ключовим етапом у розробці програмних продуктів. Для забезпечення високих стандартів якості та виявлення потенційно проблемних зон необхідні інструменти для об'єктивної оцінки програмного забезпечення. У цьому контексті застосування різноманітних метрик стає незамінним. Метрики дозволяють кількісно оцінити різні аспекти програмного забезпечення, такі як розмір, складність, надійність та продуктивність, надаючи об'єктивні дані для аналізу та прийняття рішень щодо вдосконалення проєкту.

Деякі з поширених метрик якості:

– кількість рядків коду (Lines of Code, LOC): вимірює загальну кількість рядків коду в проєкті, що може бути використано для оцінки розміру програми та необхідних тестових зусиль;

– кількість дефектів (Defect Count): показує загальну кількість виявлених дефектів або помилок у програмному забезпеченні, допомагаючи оцінити його якість та вказуючи на проблемні області, що потребують уваги;

– покриття коду тестами (Code Coverage): визначає відсоток коду, що був протестований автоматизованими тестами, вказуючи на ефективність перевірки коду на наявність дефектів;

– складність коду (Code Complexity): оцінює складність коду на основі кількості гілок у керуючих структурах, глибини вкладеності та інших факторів, допомагаючи ідентифікувати складні для розуміння та підтримки частини коду;

– частота змін (Change Frequency): вимірює частоту змін у програмному забезпеченні, що вказує на стабільність коду та частоту впровадження нових функцій або виправлення помилок;

– показник повторного використання (Code Reusability): визначає, наскільки легко код може бути повторно використаний у інших частинах програми або проєктах, вказуючи на модульність та гнучкість кодової бази.

Для оцінки метрик якості використано платформу CodeScene[12], результати зображені на рисунках 4.1-4.3.

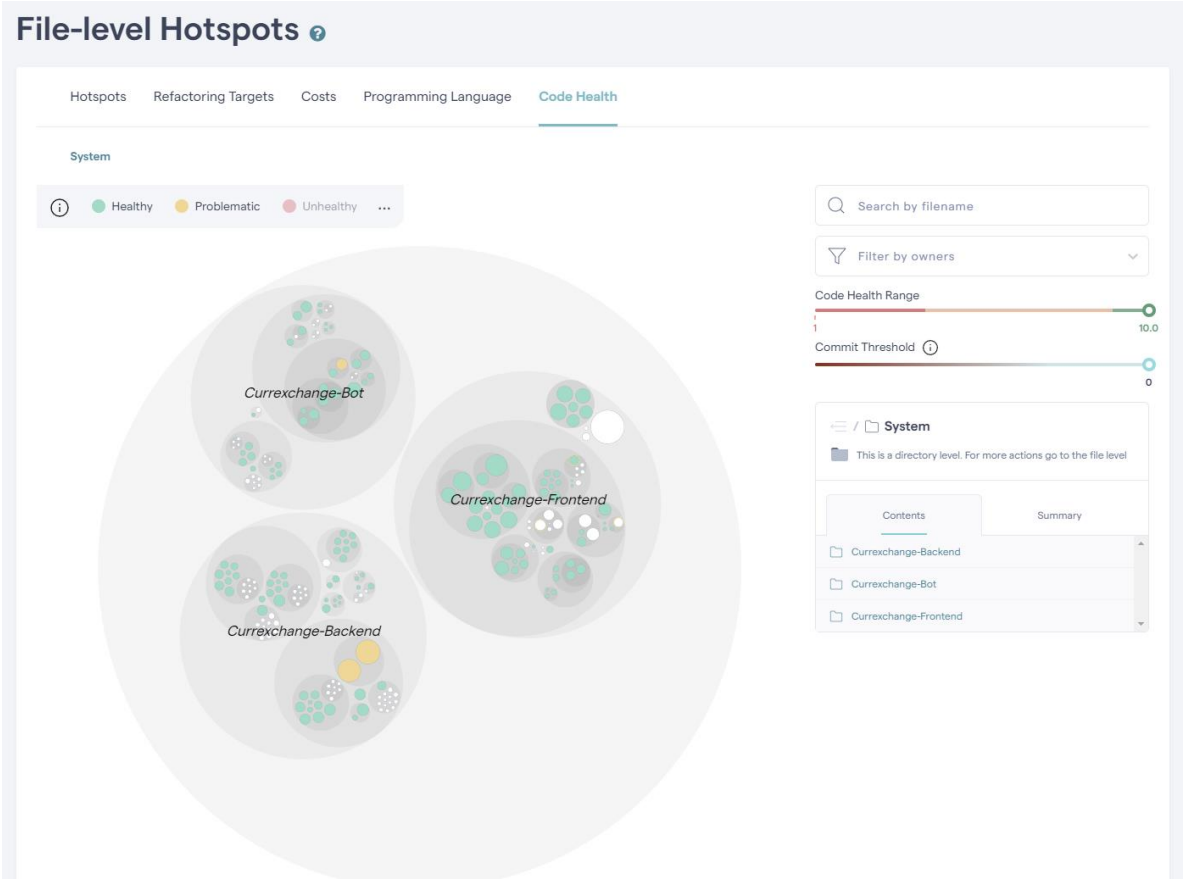


Рисунок 4.1 – Результат оцінки коду платформою CodeScene

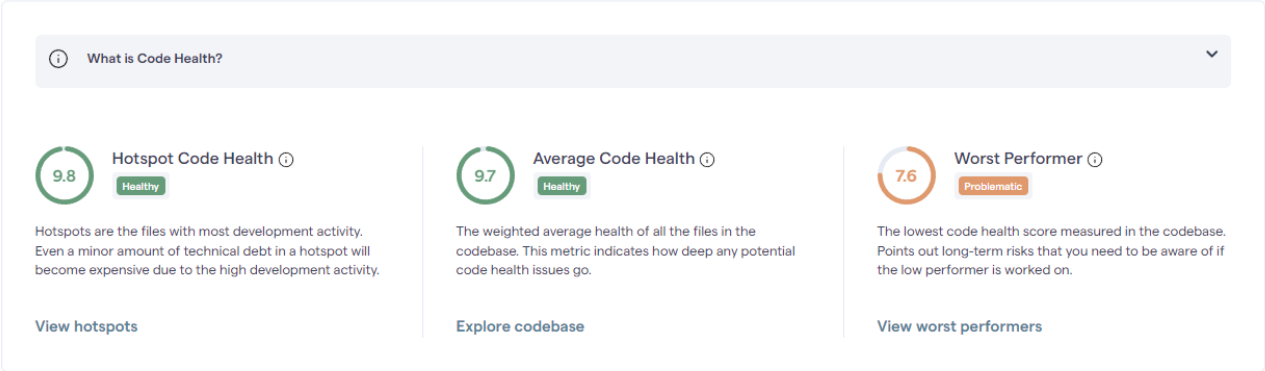


Рисунок 4.2 – Загальний результат оцінки коду платформою CodeScene

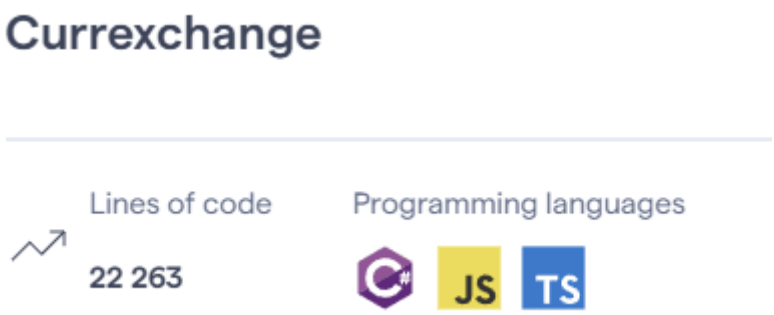


Рисунок 4.3 – Кількість рядків коду

4.2 Опис процесів тестування

Виконано ручне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1-4.10.

Таблиця 4.1 – Тест 1.1

Тест	Початок роботи з системою
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці логіну
Вхідні дані	-
Опис проведення тесту	Користувач вводить свій логін та пароль, потім натискає на кнопку “Login”.
Очікуваний результат	Система пускає користувача і виводить сторінку з дашбордами.
Фактичний результат	Система пускає користувача і виводить сторінку з дашбордами.

Таблиця 4.2 – Тест 1.2

Тест	Створення нової валюти у системі
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці валют
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Add currency” після чого, виводиться діалогове вікно з полями. Після чого, користувач заповнює всі поля та натискає на кнопку “Save”.
Очікуваний результат	Нова валюта додається до загального списку. У системі створюється курс для відповідної валюти і резерв.
Фактичний результат	Нова валюта додається до загального списку. У системі створюється курс для відповідної валюти і резерв.

Таблиця 4.3 – Тест 1.3

Тест	Створення нової транзакції
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці створення транзакції
Вхідні дані	-
Опис проведення тесту	Користувач заповнює відповідні поля для створення, натискає на кнопку “Next” і переходить на сторінку підтвердження введенних даних. Після чого, натискає на кнопку “Submit”.
Очікуваний результат	Користувача перенаправляє до списку всіх транзакцій. Створена транзакція відображається у загальному списку. Резерв валюти оновився відповідно до введенних даних.

Продовження таблиці 4.3

Фактичний результат	Користувача перенаправляє до списку всіх транзакцій. Створена транзакція відображається у загальному списку. Резерв валюти оновився відповідно до введенних даних.

Таблиця 4.4 – Тест 1.4

Тест	Створення нового бронювання
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на сторінці створення бронювання
Вхідні дані	-
Опис проведення тесту	Користувач заповнює відповідні поля для створення, натискає на кнопку “Next” і переходить на сторінку підтвердження введенних даних. Після чого, натискає на кнопку “Submit”.
Очікуваний результат	Користувача перенаправляє до списку всіх бронювань. Створене бронювання відображається у загальному списку.
Фактичний результат	Користувача перенаправляє до списку всіх транзакцій. Створена транзакція відображається у загальному списку.

Таблиця 4.5 – Тест 1.5

Тест	Створення нового контакту
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на сторінці створення контакту
Вхідні дані	-

Продовження таблиці 4.5

Опис проведення тесту	Користувач натискає на кнопку “Add contact”. Після чого, відкривається діалогове вікне з полями для заповнення. Користувач заповнює відповідні поля для створення, натискає на кнопку “Next” і переходить на сторінку підтвердження введенних даних. Після чого, натискає на кнопку “Submit”.
Очікуваний результат	Діалогове вікно закривається і загальний список контактів оновлюється і додається новий контакт.
Фактичний результат	Діалогове вікно закривається і загальний список контактів оновлюється і додається новий контакт.

Таблиця 4.6 – Тест 1.6

Тест	Початок роботи з ботом
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на сторінці з ботом
Вхідні дані	-
Опис проведення тесту	Користувач вводить команду “/start”.
Очікуваний результат	Бот запускається і відправляє користувачу головне меню.
Фактичний результат	Бот запускається і відправляє користувачу головне меню.

Таблиця 4.7 – Тест 1.7

Тест	Створення нового контакту у телеграм боті
Номер тесту	1.7

Продовження таблиці 4.7

Початковий стан системи	Користувач знаходиться на головному меню
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Створити контакт”. Після чого, бот відправляє питання, на які користувач відповідає.
Очікуваний результат	Контакт успішно створюється і користувач отримує текстове повідомлення про це у боті.
Фактичний результат	Контакт успішно створюється і користувач отримує текстове повідомлення про це у боті.

Таблиця 4.8 – Тест 1.8

Тест	Створення нового бронювання у телеграм боті
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на головному меню
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Бронювання”. Після чого, бот відправляє меню, яке відповідає за бронювання. Далі, користувач натискає на кнопку “Нове бронювання”. Бот надсилає список валют, користувач обирає необхідну. Далі, користувач обирає тип транзакції і вводить кількість валюти. Підтверджує введені дані.
Очікуваний результат	Бронювання успішно створюється у системі зі статусом “In progress”. Бот надсилає користувачу повідомлення.
Фактичний результат	Бронювання успішно створюється у системі зі статусом “In progress”. Бот надсилає користувачу повідомлення.

Таблиця 4.9 – Тест 1.9

Тест	Перегляд створених користувачем бронювань у телеграм боті
Номер тесту	1.9
Початковий стан системи	Користувач знаходиться на головному меню
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Бронювання”. Після чого, бот відправляє меню, яке відповідає за бронювання. Далі, користувач натискає на кнопку “Мої бронювання”.
Очікуваний результат	Бот надсилає список бронювань створених у системі.
Фактичний результат	Бот надсилає список бронювань створених у системі.

Таблиця 4.10 – Тест 1.10

Тест	Перегляд курсів пункту обміну у телеграм боті
Номер тесту	1.10
Початковий стан системи	Користувач знаходиться на головному меню
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Курси”. Після чого, бот відправляє список валют з якими працює пункт обміну. Користувач обирає необхідну валюту.
Очікуваний результат	Бот надсилає курс для обраної валюти.
Фактичний результат	Бот надсилає курс для обраної валюти.

4.3 Опис контрольного прикладу

Адміністратори системи мають свою систему обліку. Для її користування, їм потрібно перейти по посиланню на сайт, після чого ввести їх правильний логін та пароль, рисунок 4.4.

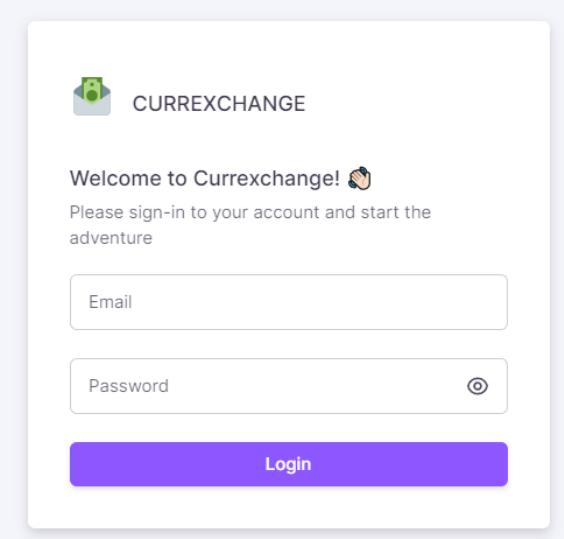


Рисунок 4.4 – Вікно авторизації у системі

Якщо користувач помилився, користувача не існує, деактивований або не правильні дані, то система сповістить його про це повідомленням у правому нижньому кутку червоним кольором з текстом помилки, рисунок 4.5.

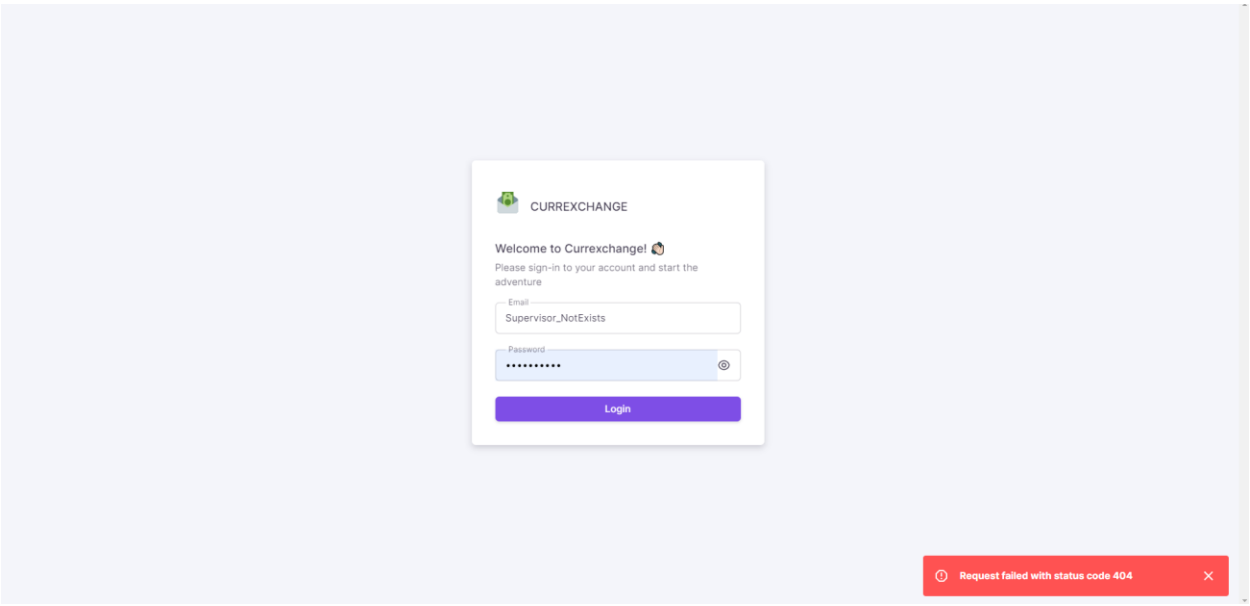


Рисунок 4.5 – Вікно авторизації після не вірного логіну\паролю

Якщо користувач ввів правильні дані, то система відкриє йому голове вікно з дашбордами, на яких зображена головна інформація. Перший блок це транзакції, де показано, скільки транзакцій проведено за весь час, сьогодні, вчора, минулий місяць та поточний. Також інформація у відсотках, на скільки змінилась кількісь у порявнянні сьогодні та вчора, поточного місяця та минулого. Другий блок містить інформацію про бронювання і має подібну структуру до транзакцій. Він включає дані про загальну кількість бронювань за весь час, кількість бронювань за сьогодні, вчора, поточний місяць та минулий місяць. Також порівняльна статистика у відсотках, на скільки збільшились кількість бронювань у відношення вчора до сьогодні і минулий місяць до теперішнього. Третій блок відповідає за загальний дохід обмінника. Тут можна побачити, скільки обмінник заробив за весь час, скільки зароблено за сьогодні, на цьому тижні і поточному місяці, рисунок 4.6.

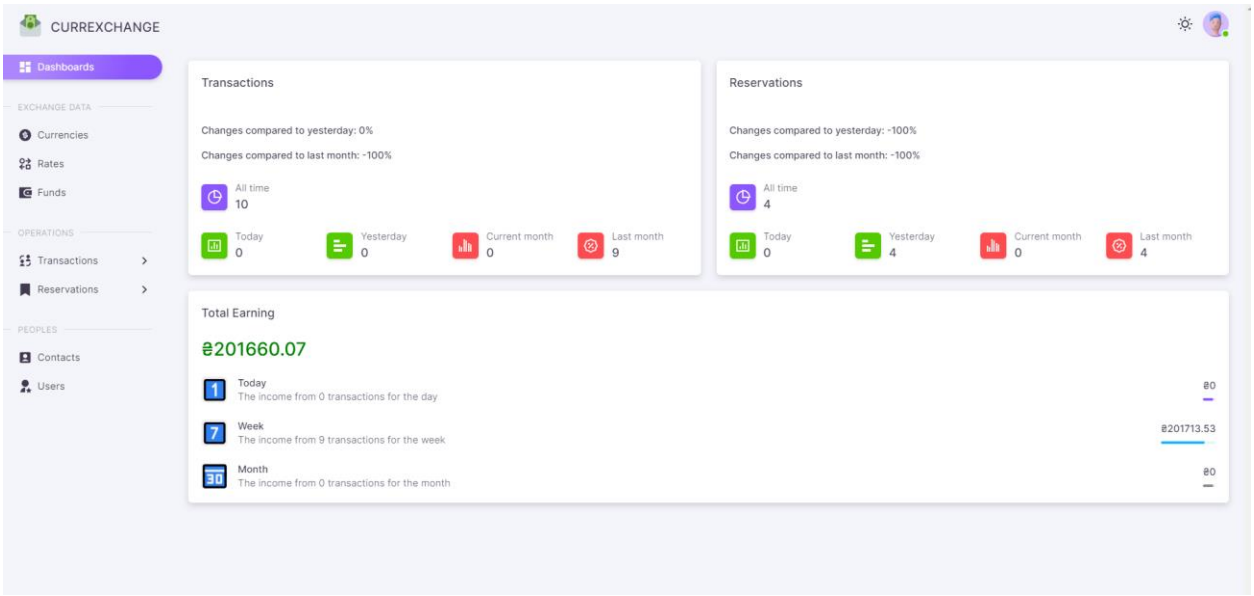


Рисунок 4.6 – Вікно дашбордів після успішної авторизації у системі

У системі є декілька ролей. Для того, щоб впевнетись, яка саме роль у користувача, достатньо натиснути на іконку користувача рисунок 4.6 і інформація про ім’я користувача і його роль, буде зображена у вигляді

випадаючого списку рисунок 4.8. Місце, куди потрібно натиснути для інформації зображено на рисунку 4.7.

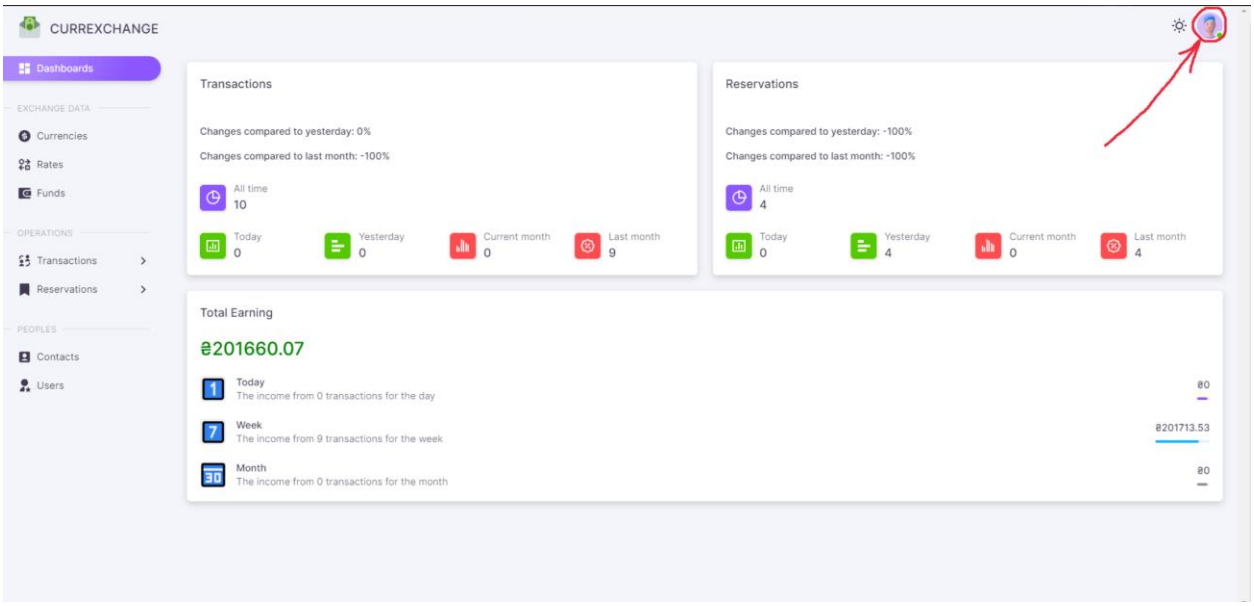


Рисунок 4.7 – Місце, куди потрібно натиснути для відображення інформації про аккаунт

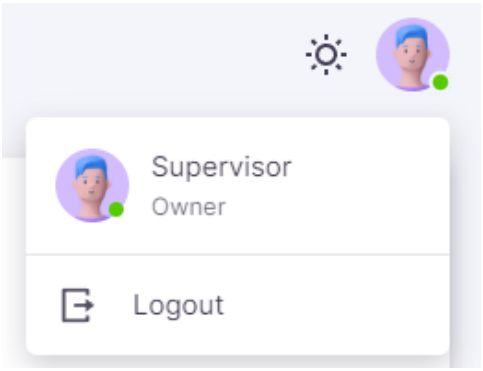


Рисунок 4.8 – Інформація про аккаунт користувача

Кожна роль має певні обмеження: власник може бачити всі сторінки, тоді як касир має доступ лише до обмеженого їх числа. Для всіх користувачів передбачене загальне меню, яке дозволяє виконувати певні дії, його зображено на рисунку 4.9.

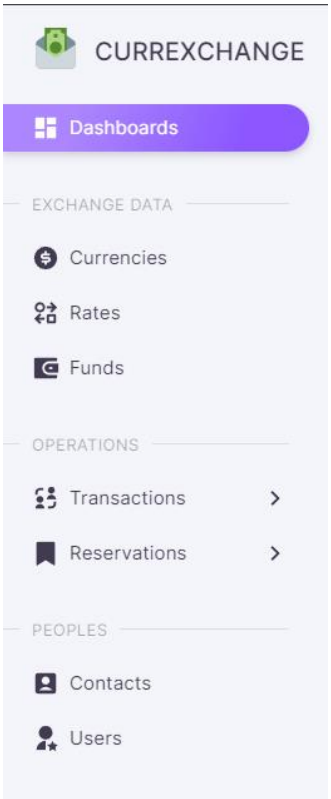


Рисунок 4.9 – Меню системи

Система працює з різними валютами, для керування якими існує розділ "Currencies". Після натискання на цей розділ користувач бачить перелік всіх валют у системі з відповідною інформацією. Кожен запис у списку містить дві кнопки: для редагування та видалення валюти. Для додавання нової валюти в систему передбачена основна кнопка "Add currency", рисунок 4.10.

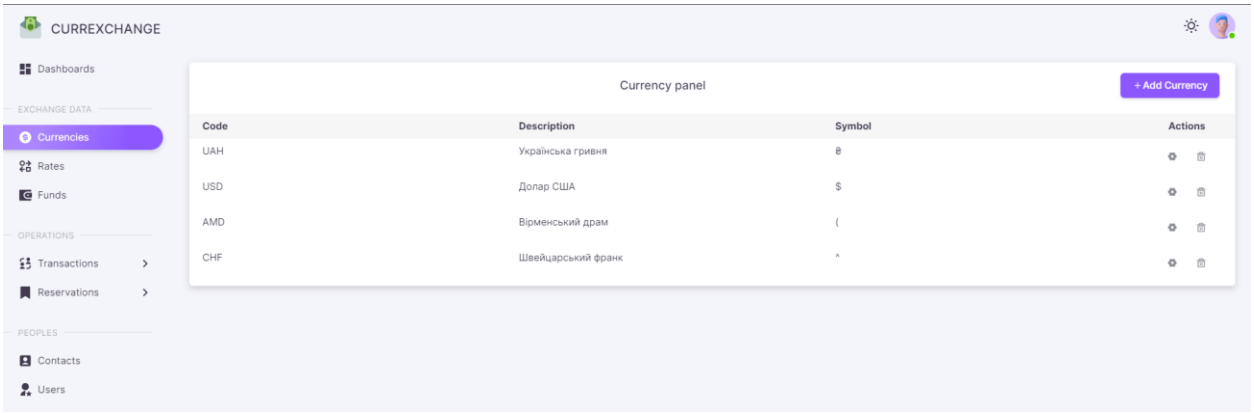


Рисунок 4.10 – Вікно для перегляду створених валют у системі

Для створення нової валюти у системі, потрібно натиснути на кнопку: “Add currency”. Після чого, користувачу відкриється діалогове вікно, яке зображене на рисунку 4.11. Це вікно можна закрити, натиснувши на кнопку “Close”. Для створення нової валюти, потрібно заповнити всі поля – вони є обов’язковими для заповнення і при спробі зберегти запис без їх заповнення, система видасть помилку і підкреслить ті поля, які користувач не заповнив, рисунок 4.12. Поле “Currency” – це безпосередньо код валюти. Користувач може створювати лише ті валюти, які обслуговує Національний банк України, тому при натисканні на це поле, буде виведений досить великий список для вибору необхідної валюти, приклад зображений на рисунку 4.13. Після того, як валюта була обрана, поле з описом автоматично буде заповнено інформацією, яка була отримана від Національного банку України, рисунок 4.14. Користувачу самому лише потрібно буде ввести символ, який асоціюється з цією валютою.

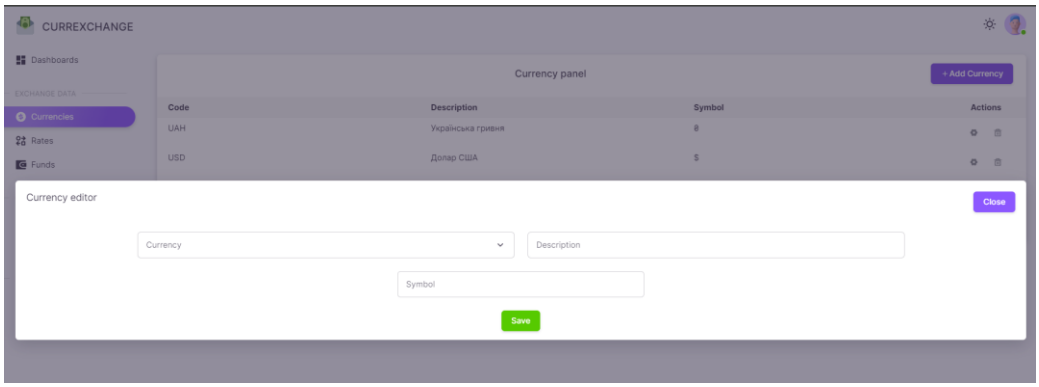


Рисунок 4.11 – Вікно створення нової валюти

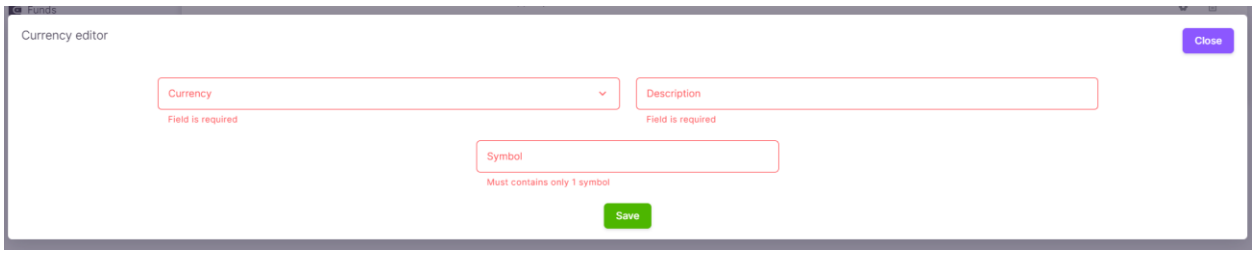


Рисунок 4.12 – Спроба збереження валюти з пустими полями

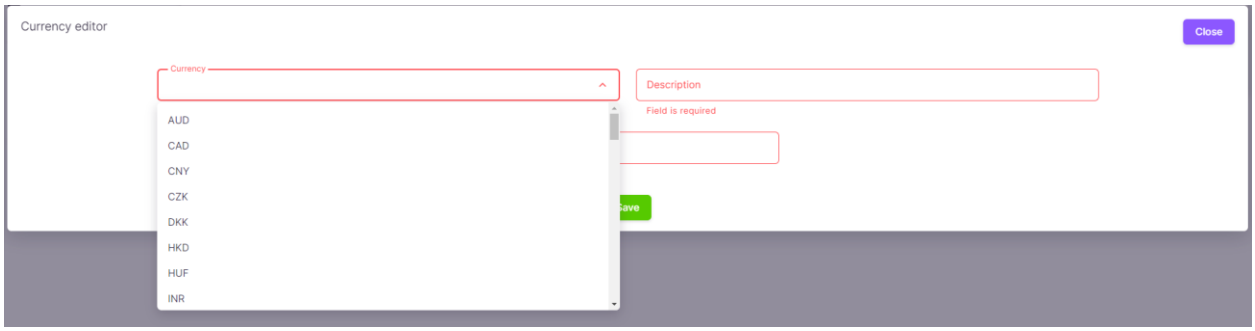


Рисунок 4.13 – Перелік валют ухвалений НБУ

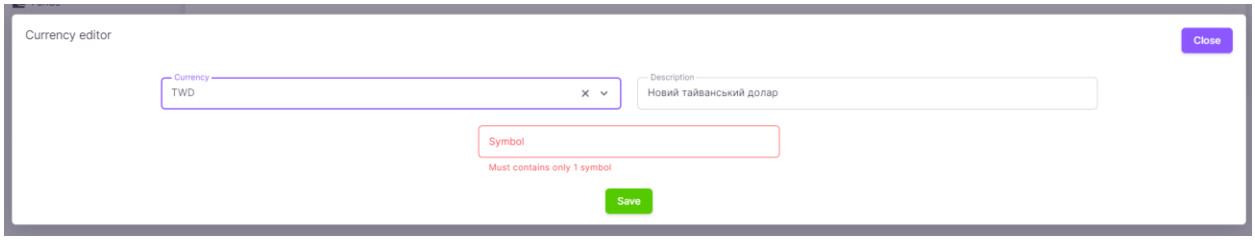


Рисунок 4.14 – Приклад автозаповнення поля з описом валюти

Після того, як користувач заповнив усі поля, він повинен натиснути на кнопку “Save”. Після чого, система створить цю валюту і оновить загальний список валют системи. Користувач отримує сповіщення зеленим кольором у нижньому правому кутку про успішність операції. Після того, як валюта була створена, система виконує запит про курс валюти до Національного банку України, робить обчислення, в залежності від відсотку, який хоче заробляти обмінник і створює новий курс. Для моніторингу кількості валют, також створюється і резерв під цю валюту, рисунок 4.15.

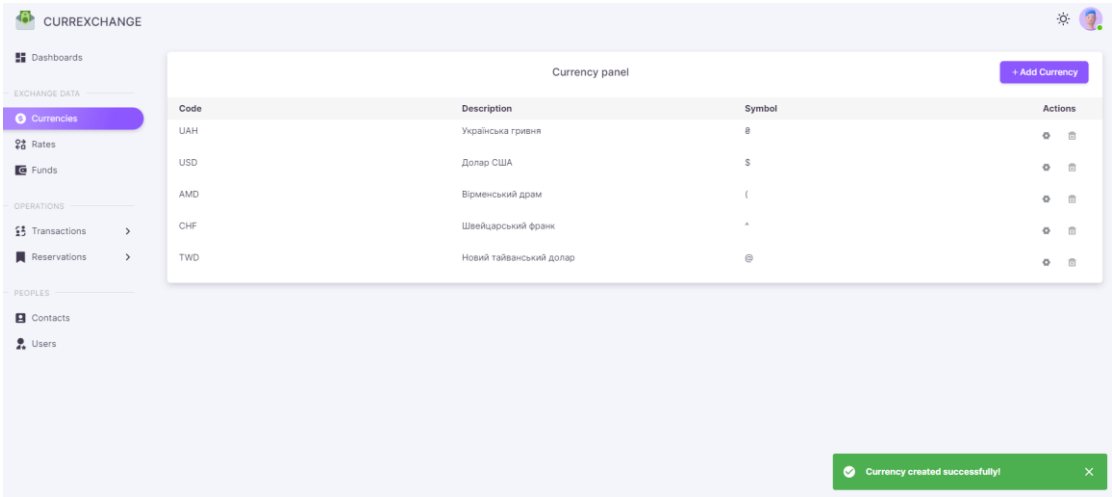


Рисунок 4.15 – Інформація про успішне додавання валюти у систему

Як зазначено вище, якщо користувач помилився з деякими полями при створенні валюти, він зможе успішно її усунути, натиснувши на шестерню біля необхідного запису для модифікування валюти, рисунок 4.16. Після кліку на цю кнопку, користувачу буде відкрито діалогове вікно з інформацією, яку він може змінити. Для закриття цього вікна, можна натиснути на кнопку “Cancel”. Але для збереження змін, користувач повинен натиснути на кнопку “Save”, рисунок 4.17.

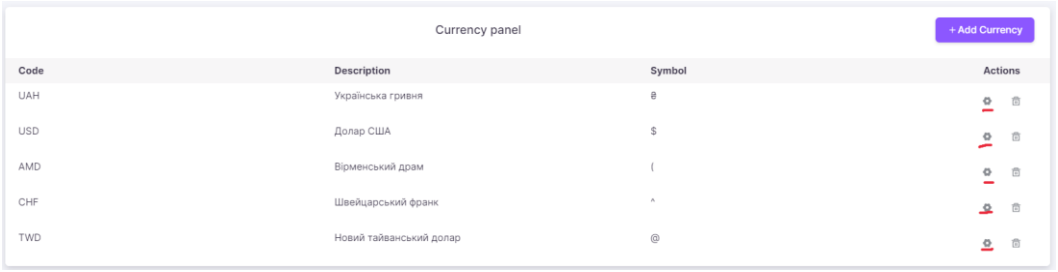


Рисунок 4.16 – Місце, куди потрібно натиснути для модифікації валют

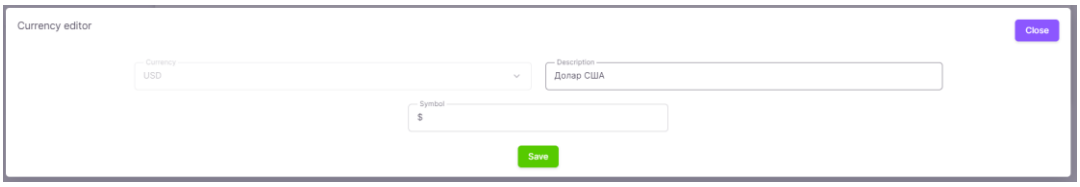


Рисунок 4.17 – Вікно редагування валюти

Якщо обмінник більше не збирається працювати з певною валютою з переліку, користувачу необхідно натиснути на кнопку для видалення валюти, вона зображена у вигляді кошику. Після цього, інформація про курси для цієї валюти, резерви буде видалена з системи. Список валюти оновиться.

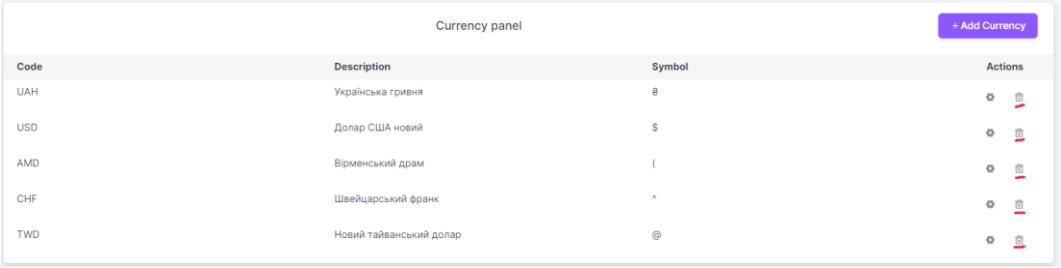


Рисунок 4.18 – Місце куди треба натискати для видалення валюти

Для того, щоб переглянути інформацію про курси валют, потрібно натиснути на кнопку “Rates” у лівому меню системи, рисунок 4.19. Система відкриє сторінку зі списком, де буде інформація про валюту та її курс продажу або купівлі обмінником, рисунок 4.20. Цю інформацію не можна видаляти та змінювати. Це обумовлено безпекою системи та цілісністю даних.

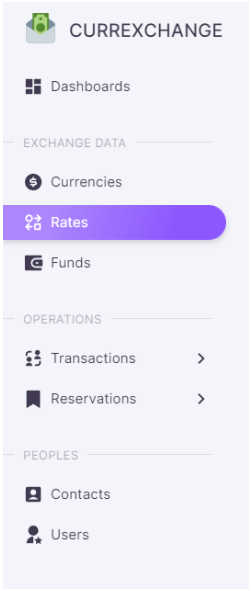


Рисунок 4.19 – Місце, куди потрібно натиснути для перегляду курсів валют

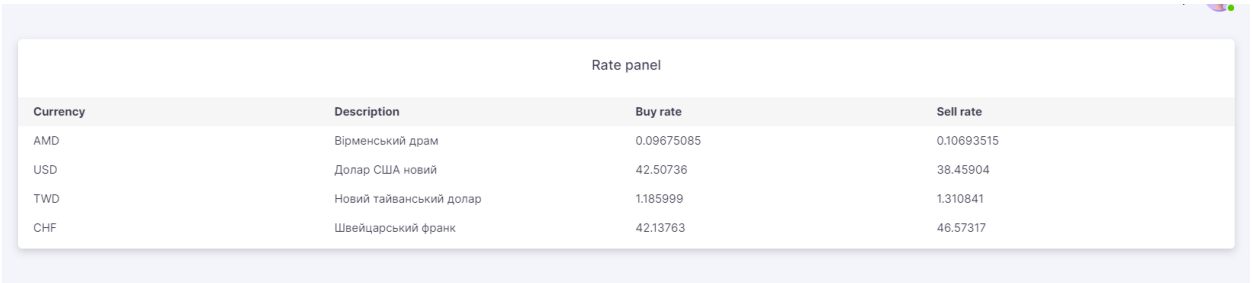


Рисунок 4.20 – Курси валют наявні у системі

Для управління резервами, є спеціальний пункт у меню “Funds”, рисунок 4.21. Там зображена інформація щодо резервів певних валют, рисунок 4.22. Його можна оновлювати, рисунок 4.23, так як він з’являється після додавання валюти у систему зі значенням 0, рисунок 4.24. Після видалення валюти, він так само видаляється, як і прив’язаний курс.

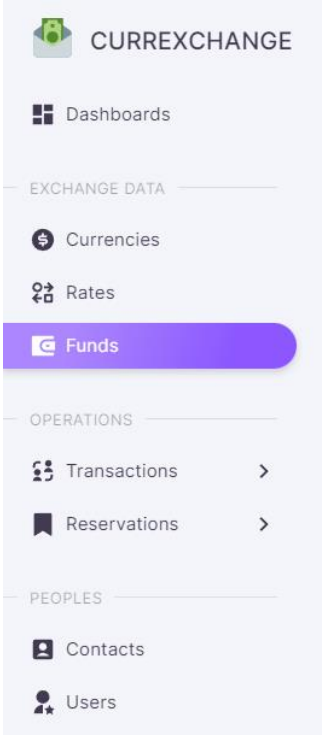


Рисунок 4.21 – Місце знаходження пункту резерву

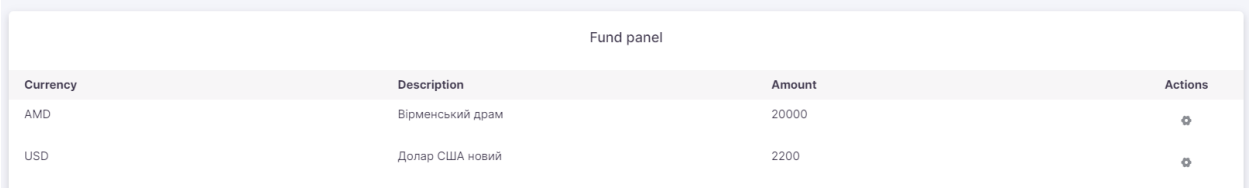


Рисунок 4.22 – Перелік резервів відповідно до валют у системі

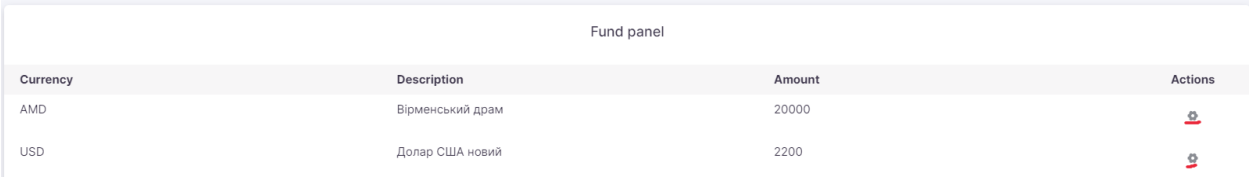


Рисунок 4.23 – Місце, куди потрібно натискати для оновлення резерву

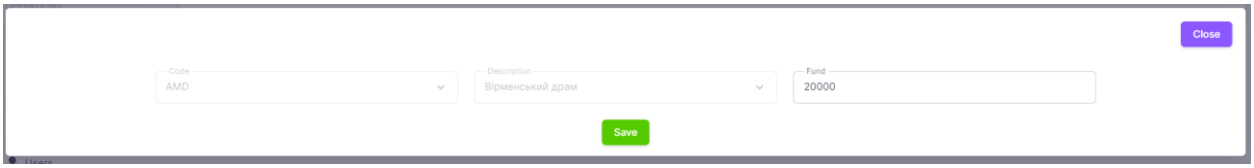


Рисунок 4.24 – Місце для редагування резерву

Система має можливість зберігати інформацію про своїх клієнтів. Вони представлені контактами. Їх можна створювати, змінювати та видаляти або заносити у чорний список. Для створення нового контакту, потрібно перейти у відповідний пункт меню, рисунок 4.25, і натиснути на кнопку “Add contact”, рисунок 4.26. Після чого, користувачу буде відкрито діалогове вікно, де він повинен заповнити поля і натиснути на кнопку збереження, рисунок 4.27. Після чого, створений користувач з’явиться у загальному списку контаків. Ці контакти можна як оновлювати, так і видаляти. Дії схожі з минулими пунктами.

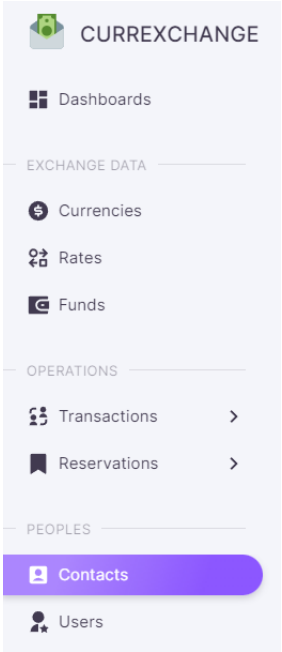


Рисунок 4.25 – Пункт контактів у меню

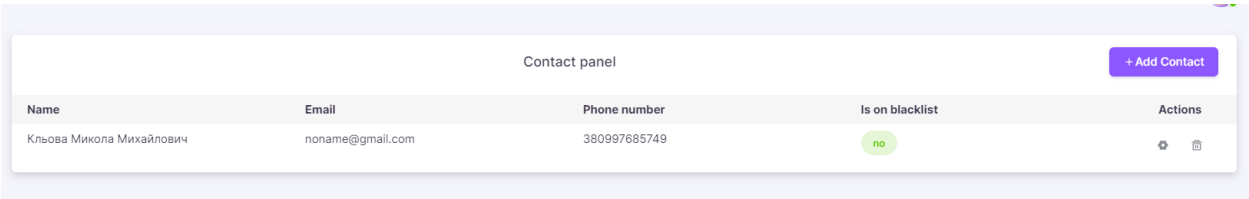


Рисунок 4.26 – Контакти системи

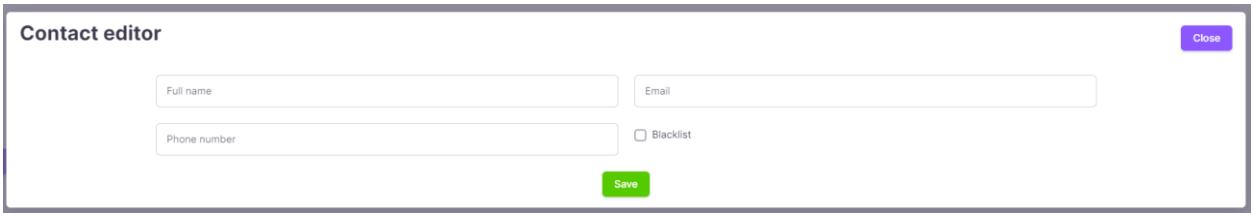


Рисунок 4.27 – Вікно для створення нового контакту

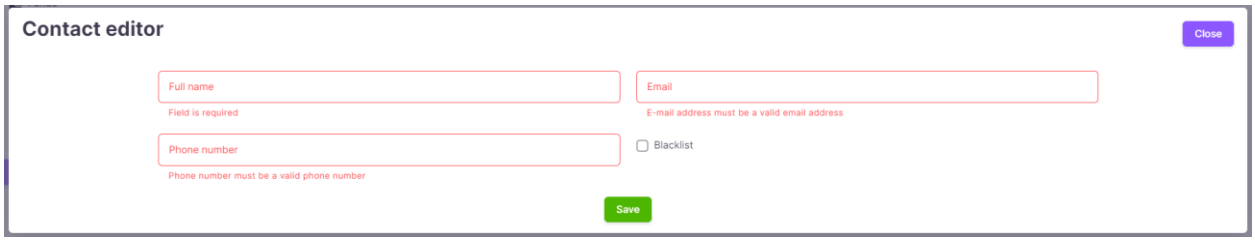


Рисунок 4.28 – Інформація про обов’язкові поля для заповнення

Перейдемо до найголовніших пунктів системи. Для того, щоб переглянути можливі дії з бронюваннями потрібно натиснути на пункт меню “Reservations”, рисунок 4.29, після чого, будуть відкриті 2 кнопки, пов’язані з бронюваннями “Create” і “View”, створення та перегляд – відповідно.

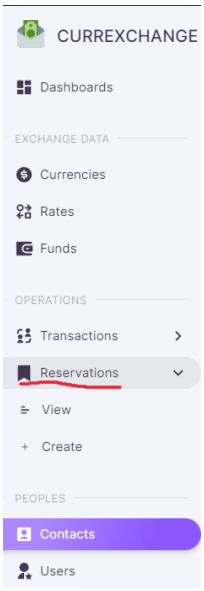


Рисунок 4.29 – Місце, куди потрібно натиснути для операцій з бронюваннями

Після кліку на підпункт “Create”, користувача буде перенаправлено на сторінку створення нового бронювання, рисунок 4.30-4.32.

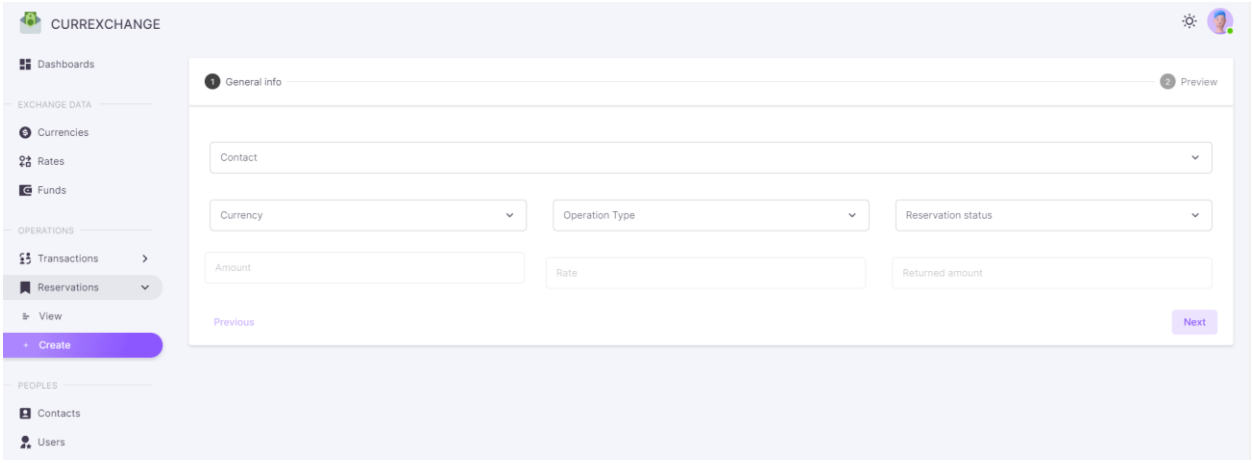


Рисунок 4.30 – Поля для заповнення для створення нового бронювання

Без заповнення необхідних полів, користувач не зможе перейти на наступний крок.

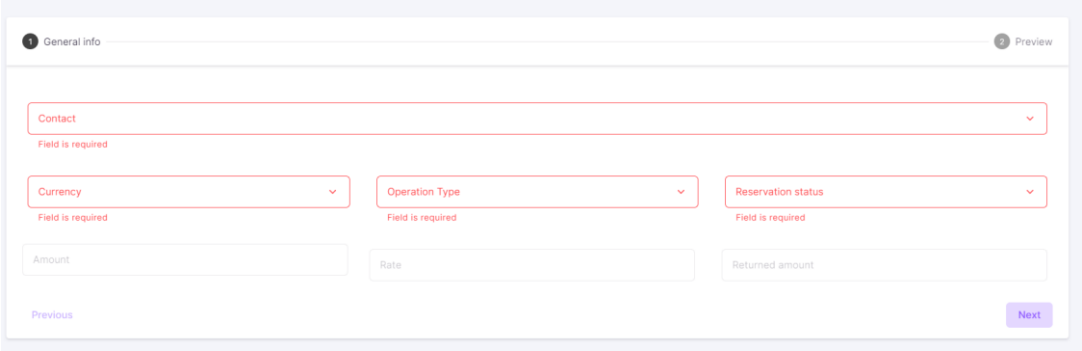


Рисунок 4.31 – Підказка, які поля є необхідними для заповнення

Після заповнення необхідних полів, потрібно натиснути на кнопку “Next”.

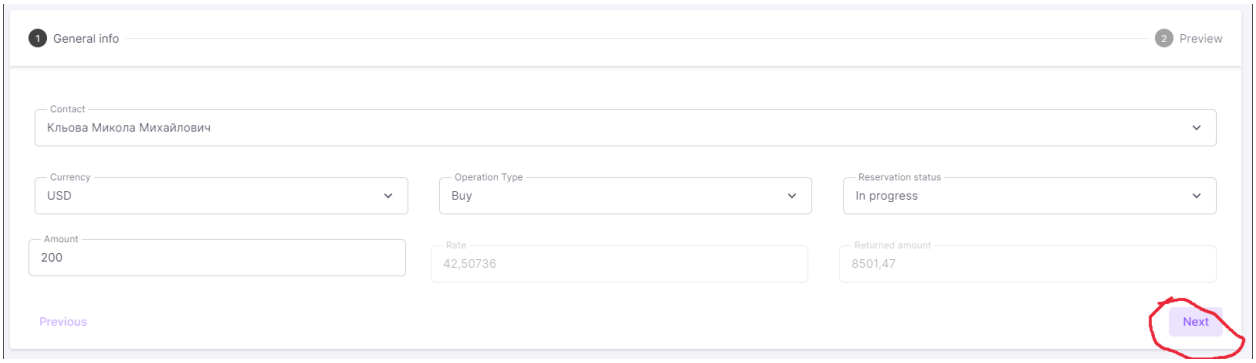


Рисунок 4.32 – Місце, куди потрібно натиснути для переходу на інший крок

Користувач повинен впевнитись у правильності введених даних, тому буде зображено вікно підтвердження, де буде зображено все введені поля, але заблоковані для редагування, рисунок 4.33.

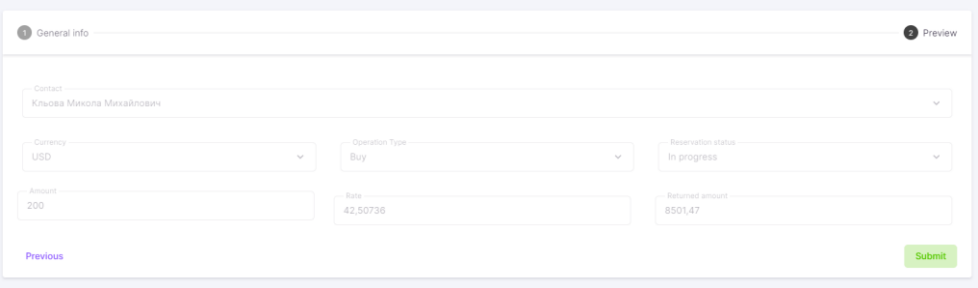


Рисунок 4.33 – Крок для перевірки правильності введених даних

Якщо помічено, що якісь дані не є вірними, користувач може повернутися на попередній крок і виправити їх, натиснувши на кнопку “Previous”, рисунок 4.34.

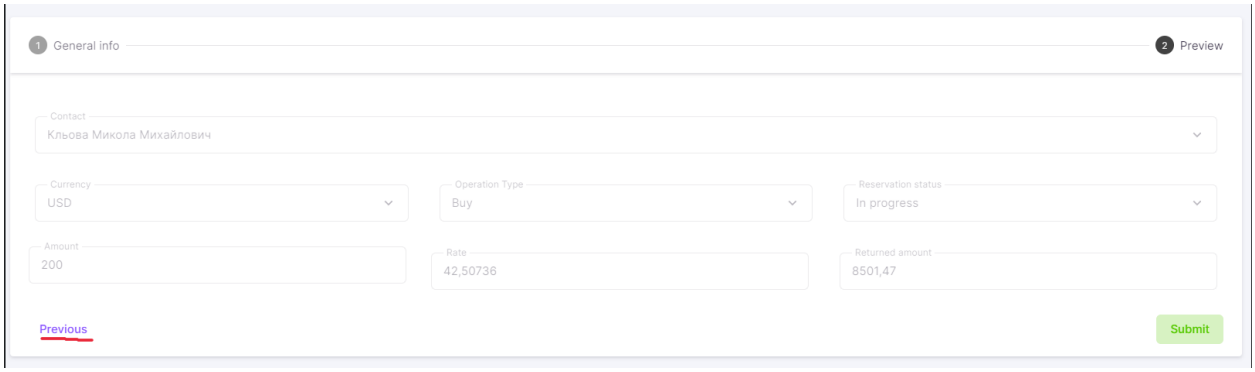


Рисунок 4.34 – Місце, куди потрібно натиснути для виправлення даних у полях

Після переконання, що всі дані є правильними, потрібно натиснути на кнопку “Submit”, після чого, користувача перенаправить на сторінку всіх бронювань у системі і він буде проінформований про успішність створення бронювання, рисунок 4.35.

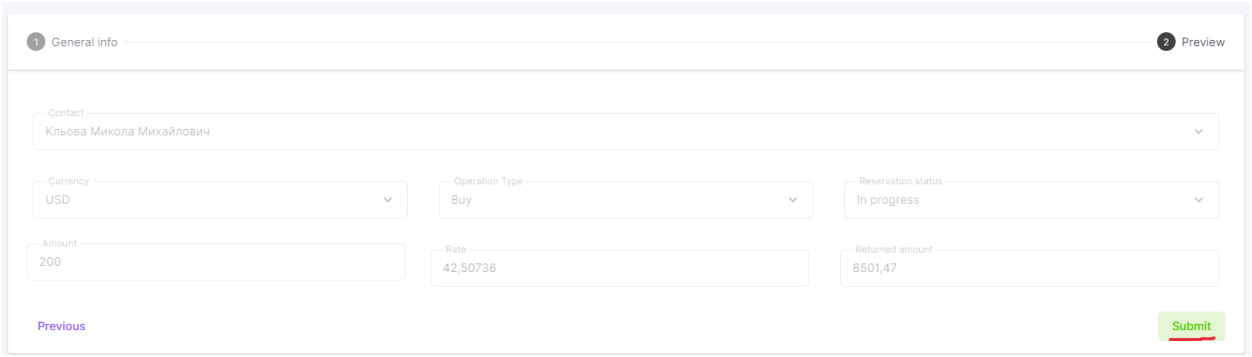


Рисунок 4.35 – Місце, куди потрібно настиснути для збереження бронювання у системі

На минулу сторінку можна потрапити натиснувши на пункт меню у “Reservations” – “View”, рисунок 4.36.

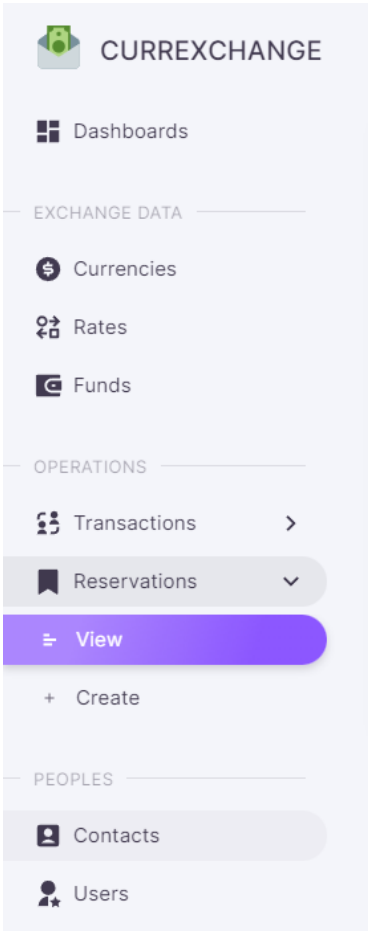


Рисунок 4.36 – Пункт меню для перегляду створених бронювань у системі

Коли клієнт приходить у пункт обміну, користувач повинен занести у систему інформацію про те, яка валюта була продана або куплена, по якому курсу і яка кількість. Для цього є спеціальний пункт у меню “Transactions”, рисунок 4.37.

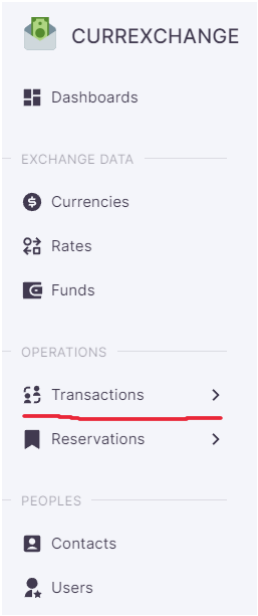


Рисунок 4.37 – Пункт меню для транзакцій проведених пунктом обміну

Для створення нової транзакції, потрібно розгорнути цей пункт і натиснути на пункт “Create”, рисунок 4.38.

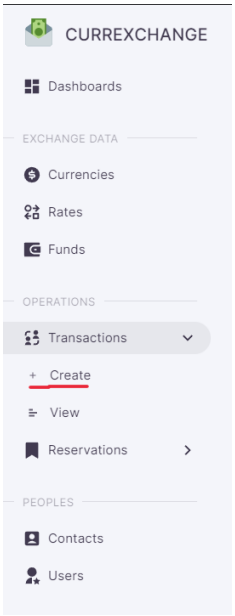


Рисунок 4.38 – Пункт меню для створення транзакції

Після натискання на пункт створення транзакції, користувачу буде відкрите вікно з полями, необхідними для заповнення, рисунок 4.39.

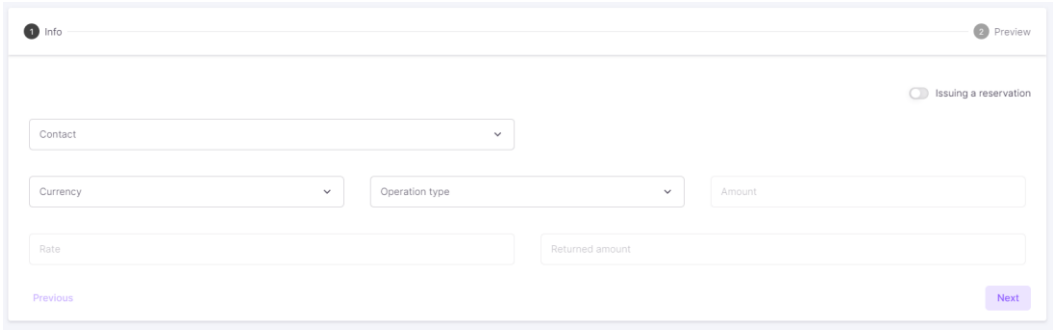


Рисунок 4.39 – Вікно для створення транзакції у системі

Якщо користувач пропустить обов’язкові поля для заповнення, система його повідомить про це і не дасть перейти на наступний крок, рисунок 4.40.

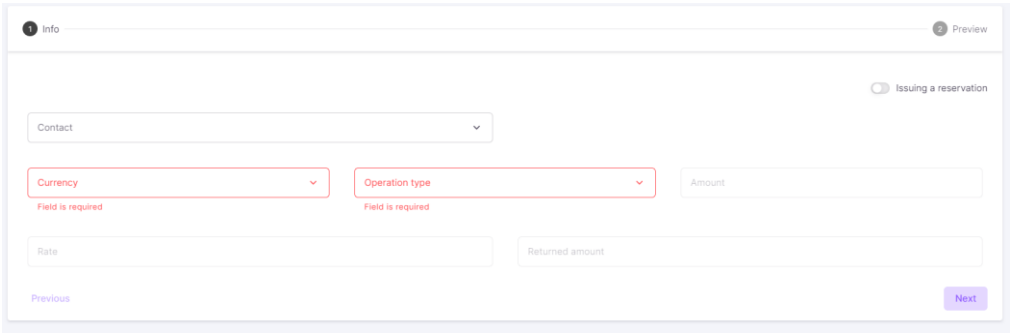


Рисунок 4.40 – Обов’язкові поля для заповнення

Після заповнення полів, треба перейти на наступний крок, перегляду введених даних, рисунок 4.41-4.42.

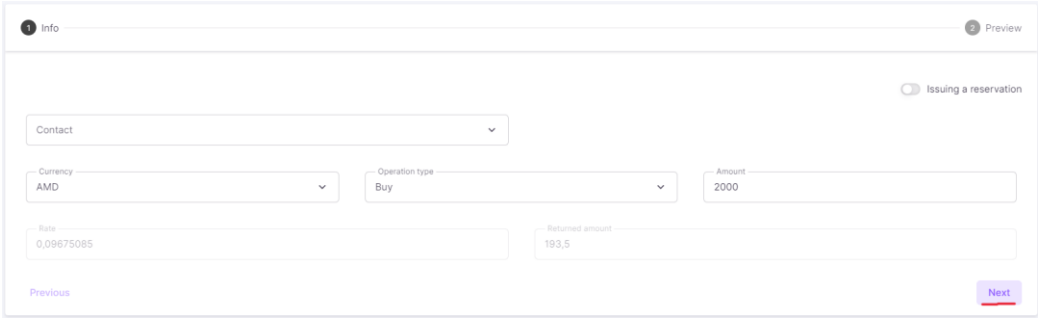


Рисунок 4.41 – Місце, куди треба натиснути для переходу на наступний крок

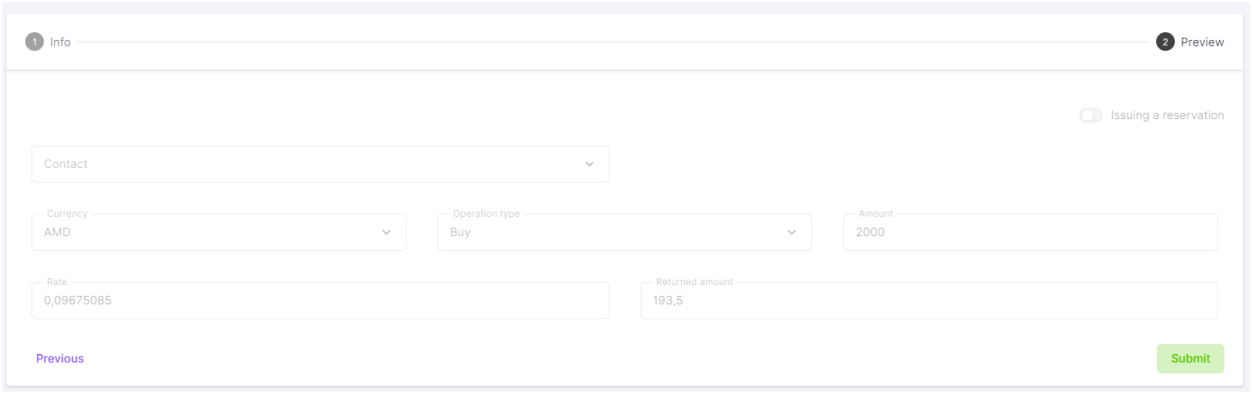


Рисунок 4.42 – Крок перегляду введеної інформації

Після того, як користувач впевнився у правильності занесеної інформації, він натискає на кнопку “Submit”, рисунок 4.43, після чого, він буде перенаправлений на сторінку перегляду транзакцій у буде проінформований повідомленням у правому нижньому кутку про успішне створення транзакції.

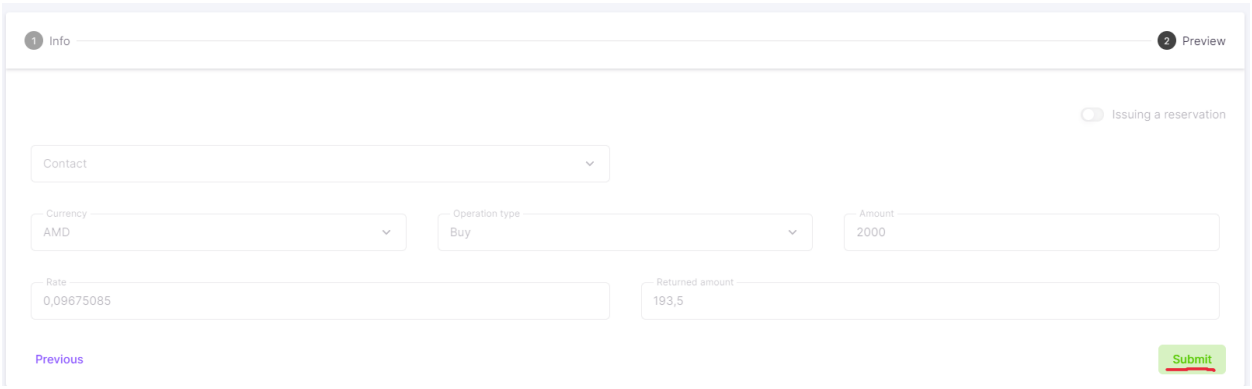


Рисунок 4.43 – Місце, куди потрібно натиснути для створення транзакції

Є ситуація, коли людина приходить з певним бронюванням і у транзакції потрібно вказати, що вона належить до певного бронювання, тоді при створенні транзакції включаємо пункт “Issuing a reservation”, де додається поле з бронюваннями, яке користувач повинен вибрати для транзакції і наступні кроки є ідентичними зі звичайним створенням транзакції, рисунок 4.44.

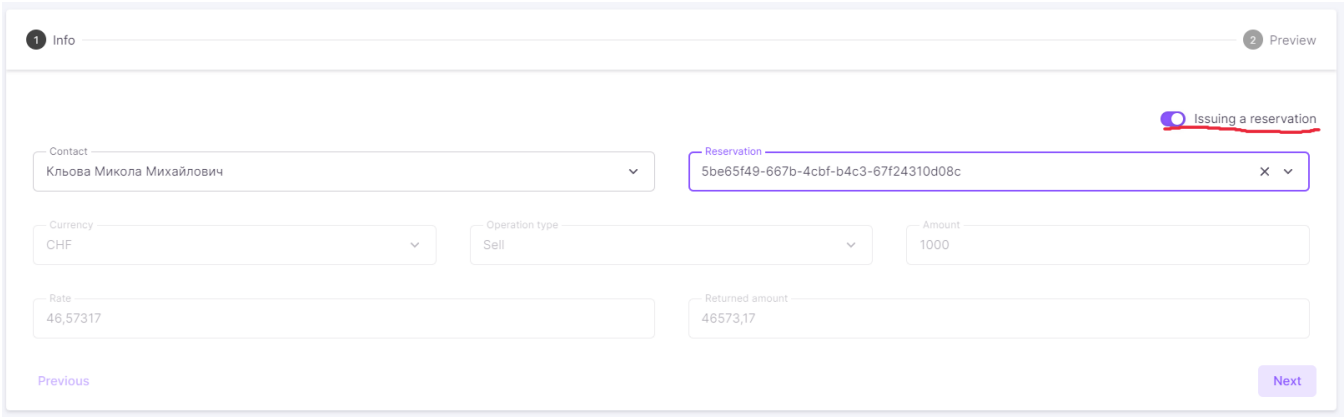


Рисунок 4.44 – Кнопка для активування режиму видачі бронювання

Для того, щоб переглянути проведені транзакції обмінником, потрібно натиснути на кнопку “View” у пункті “Transactions”, рисунки 4.45-4.46.

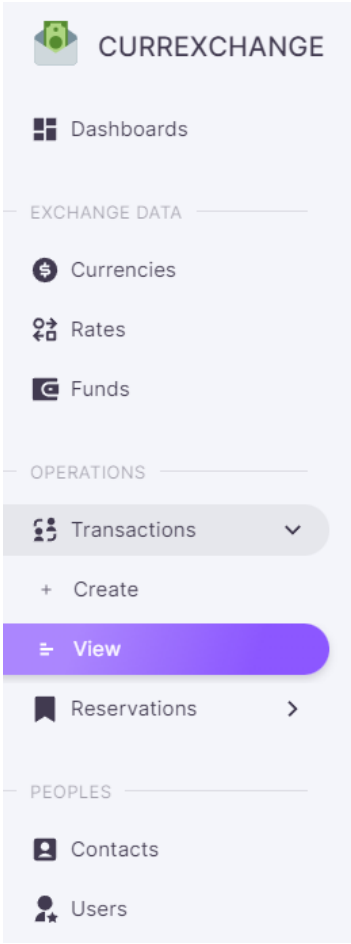


Рисунок 4.45 – Кнопка для перегляду проведених транзакцій

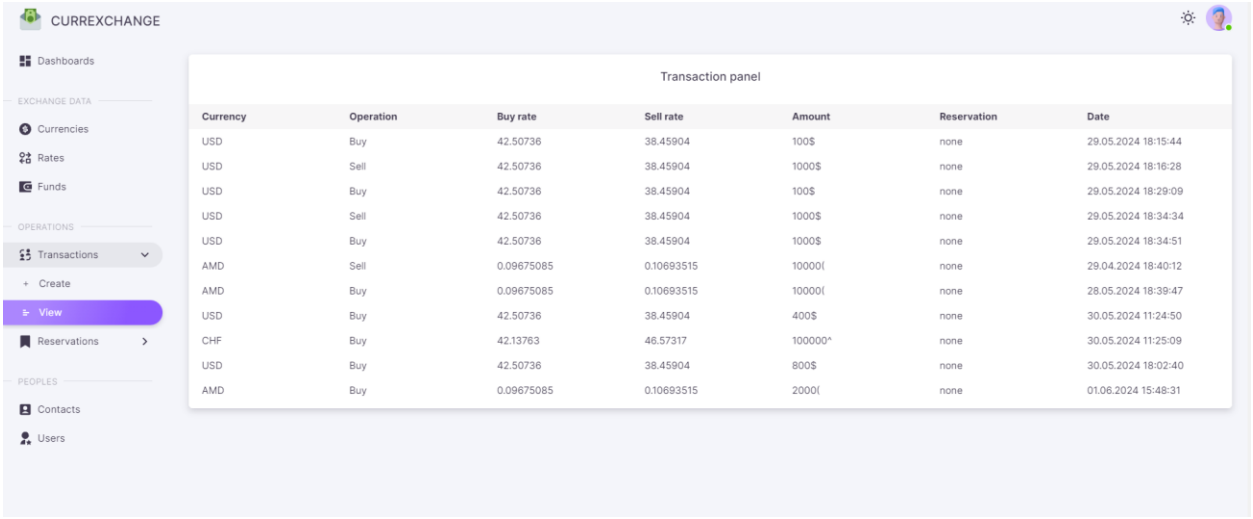


Рисунок 4.46 – Меню з проведеними транзакціями

Після завершення роботи у системі, користувач повинен вийти з неї, натиснувши на кнопку “Logout”, де його перенаправить на сторінку авторизації, рисунок 4.47.

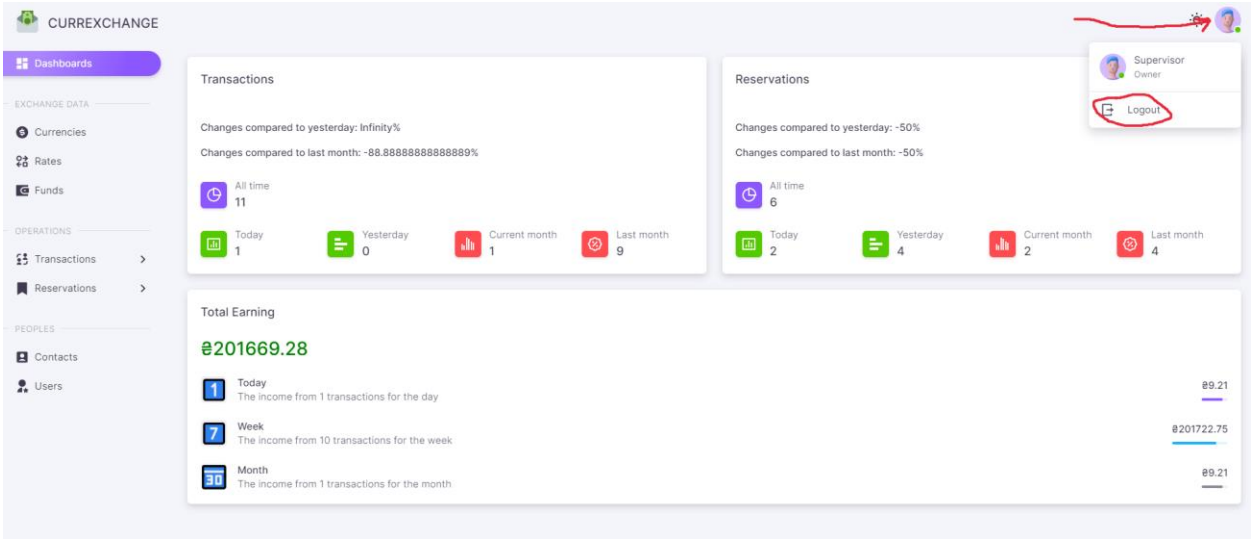


Рисунок 4.47 – Кнопка, яку потрібно натиснути для виходу з системи

Система також передбачає використання телеграм боту для клієнтів пункту обміну валют. Для того, щоб використовувати цей функціонал, клієнт повинен зайти на бот пункту обміну валют і після чого виконати команду /start, після виконання якої, клієнту буде відображене головне меню, рисунок 4.48.

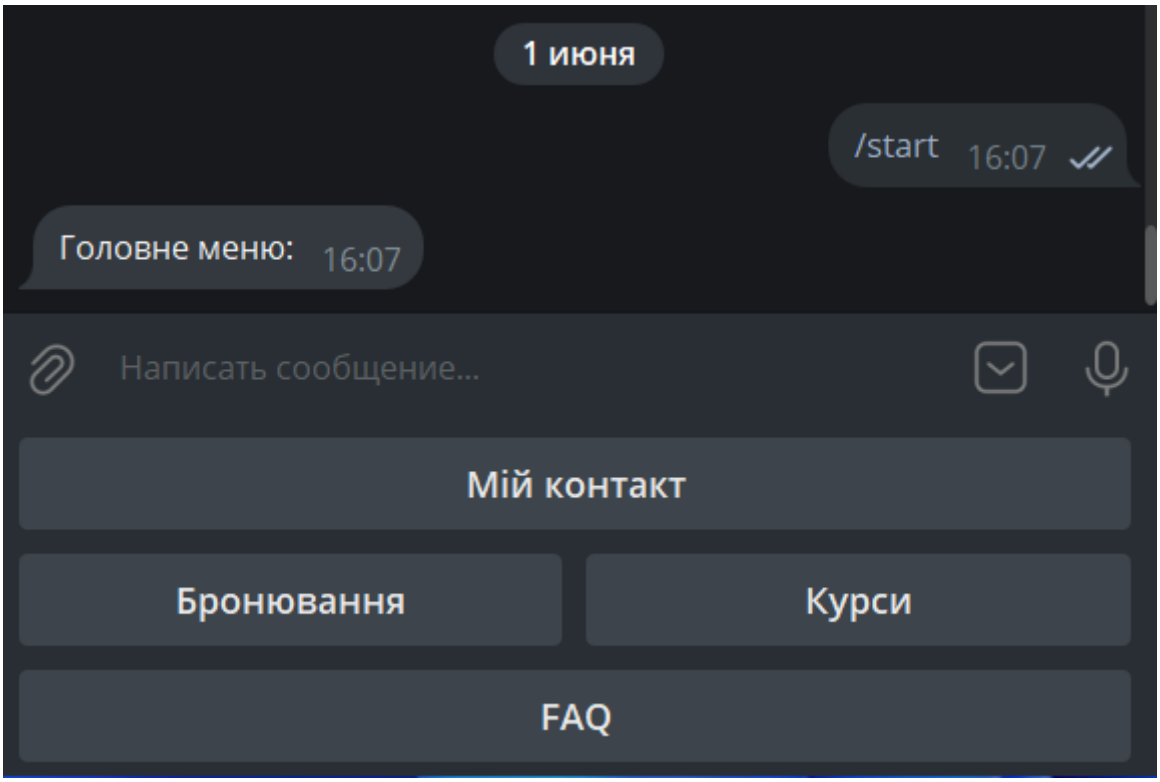


Рисунок 4.48 – Головне меню боту

Для того, щоб користуватись можливостями боту, потрібно створити свій контакт, тому після натискання на кнопку “Мій контакт”, рисунок 4.49, користувачу повинно буде заповнити необхідні дані.

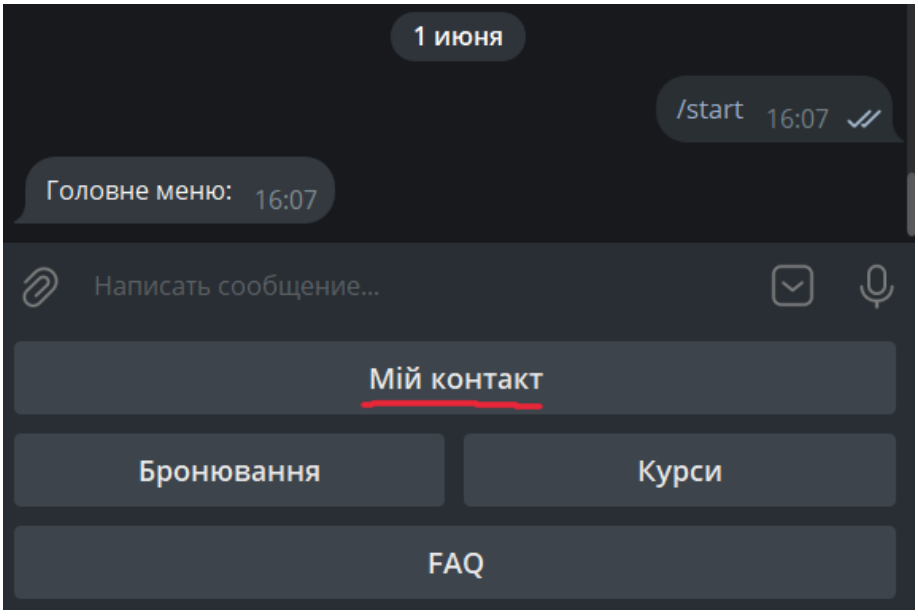


Рисунок 4.49 – Кнопка для створення мого контаку

Після відповідей на всі питання система створить новий контакт, рисунок 4.50.

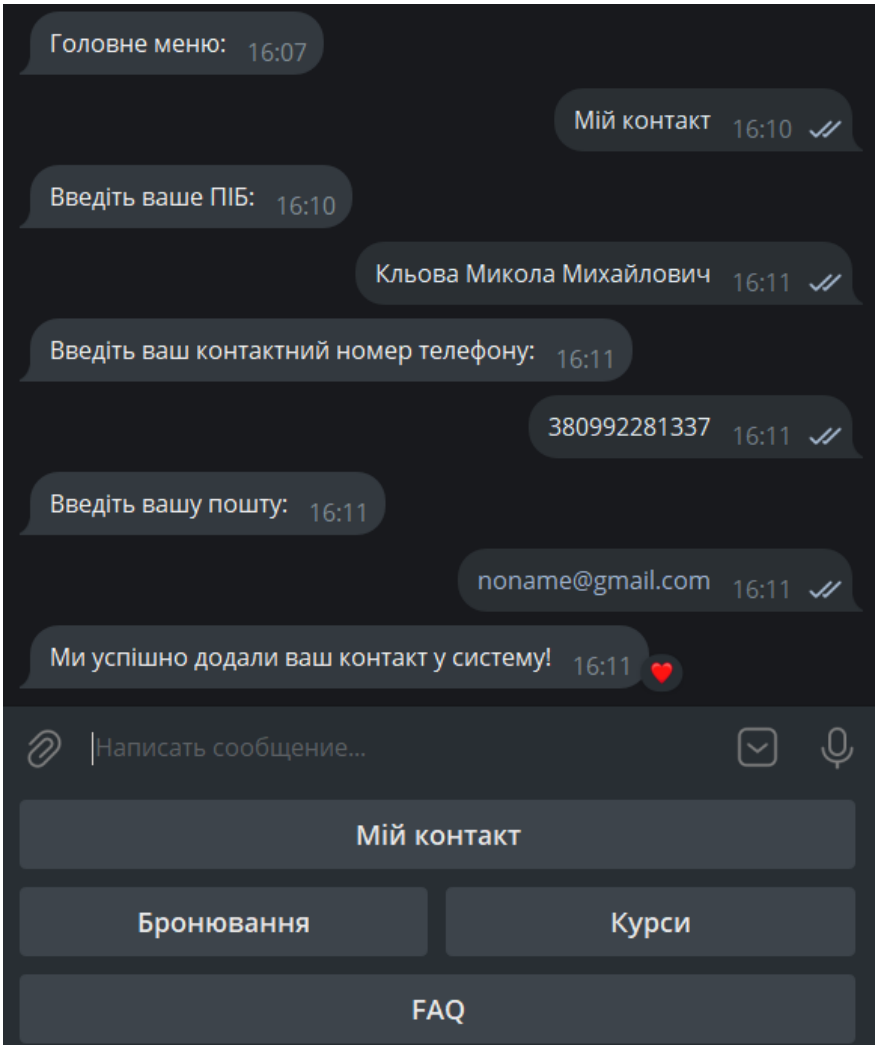


Рисунок 4.50 – Перелік питань для створення контакту

Після цього, користувачу можна буде використовувати систему повноцінно. Для ознайомлення з курсами валют, які представлені обмінником, потрібно натиснути на кнопку “Курси”, рисунок 4.51.

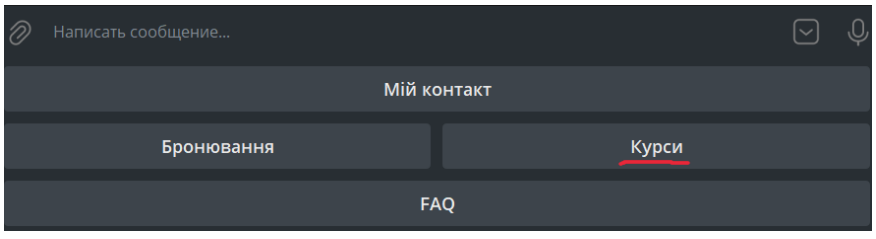


Рисунок 4.51 – Кнопка, на яку потрібно натиснути для перегляду курсів валют які представлені обмінником

Після чого, користувачу буде наданий список валют, з якого він потрібен обрати ту, яка його цікавить, у поточному випадку, це “USD”, рисунки 4.52 – 4.53.

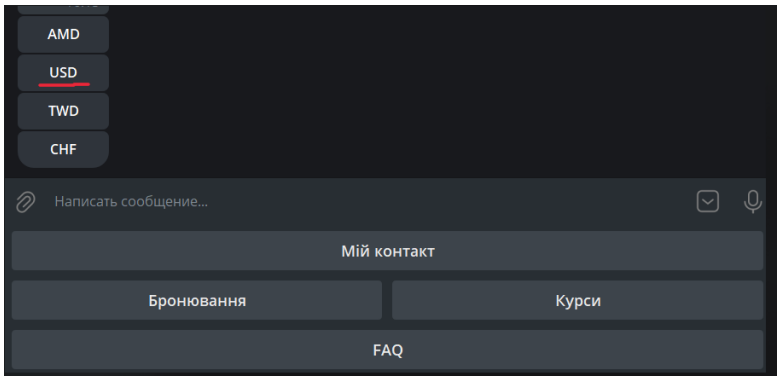


Рисунок 4.52 – Кнопка, на яку треба натиснути, для перегляду курсів валюти USD

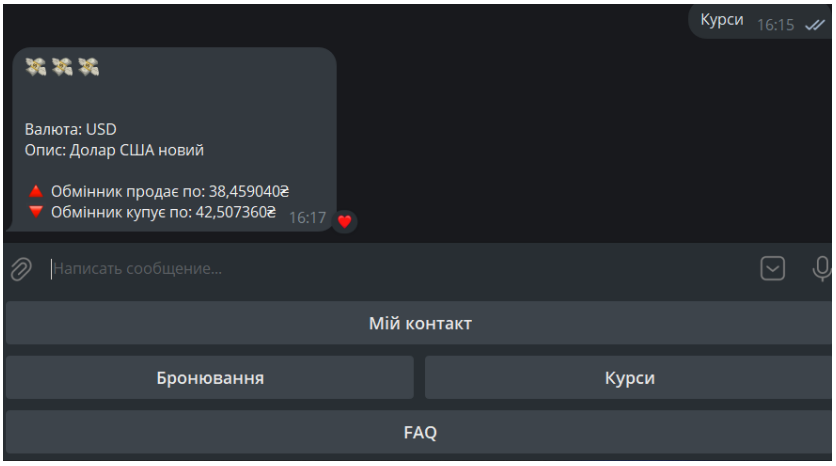


Рисунок 4.53 – Результат вибору валюти

Для того, щоб оперувати бронюваннями, потрібно обрати пункт у головному меню “Бронювання”, рисунки 4.54 – 4.55.

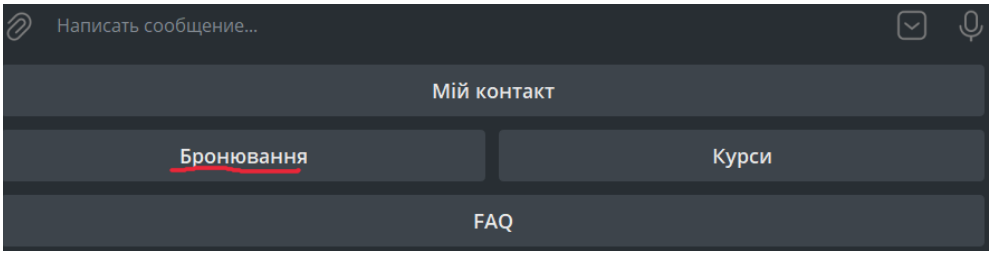


Рисунок 4.54 – Пункт для переходу у меню бронювань

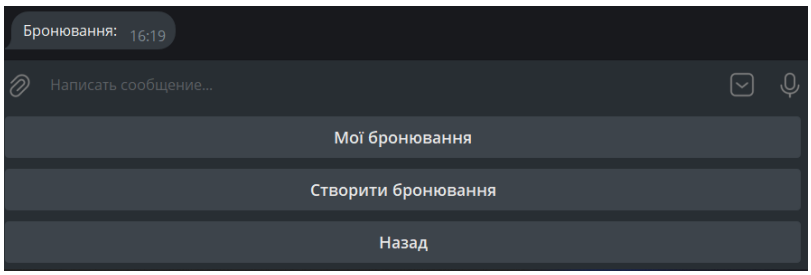


Рисунок 4.55 – Меню бронювання

Щоб створити нове бронювання, потрібно натиснути на кнопку “Створити бронювання”, рисунок 4.56.

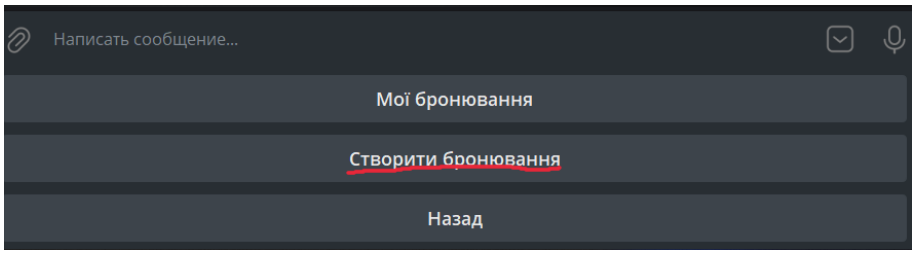


Рисунок 4.56 – Кнопка для створення нового бронювання

Після цього, користувачу буде запроповновано валюти, по яким він може створити бронювання, рисунок 4.57.

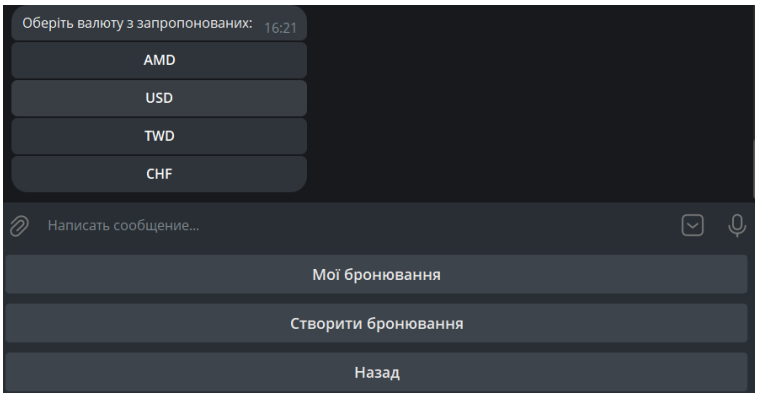


Рисунок 4.57 – Валюти доступні для бронювання

Після вибору необхідної валюти, потрібно вибрати операцію, яка буде здійснюватись, рисунок 4.58.

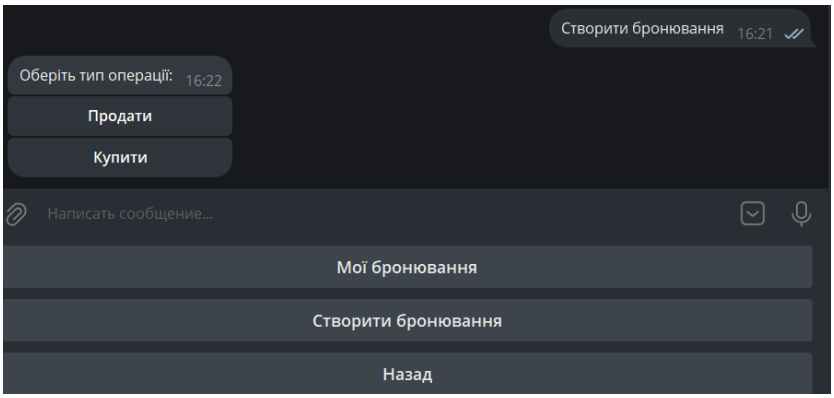


Рисунок 4.58 – Операції для бронювання

Як обрано необхідну дію, треба ввести суму валюти, яку користувач хоче забронювати, рисунок 4.59.

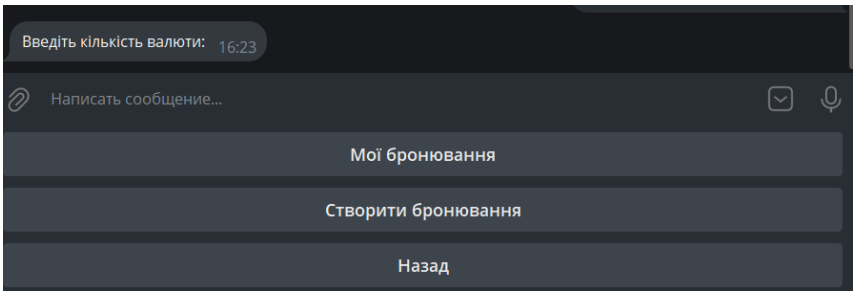


Рисунок 4.59 – Введення кількості валюти

Після успішного вводу кількості валюти, у нас буде вікно підтвердження, з актуальним курсом на даний момент, рисунок 4.60.



Рисунок 4.60 – Підтвердження бронювання

Після підтвердження введених даних, створюється бронювання і користувач проінформований повідомленням, рисунок 4.61.

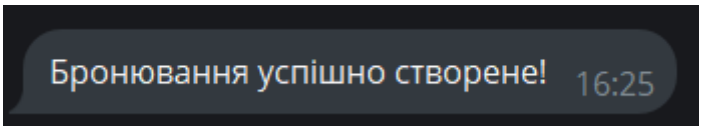


Рисунок 4.61 – Інформація про успішно створене бронювання

Інформацію про створене бронювання, можна переглянути у пункті бронювань, настиснувши на кнопку “Мої бронювання”, рисунки 4.62 – 4.63.

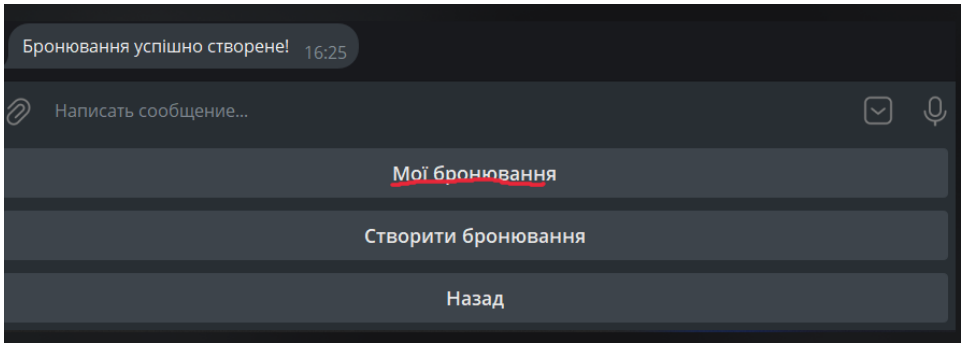


Рисунок 4.62 – Кнопка для перегляду створених бронювань

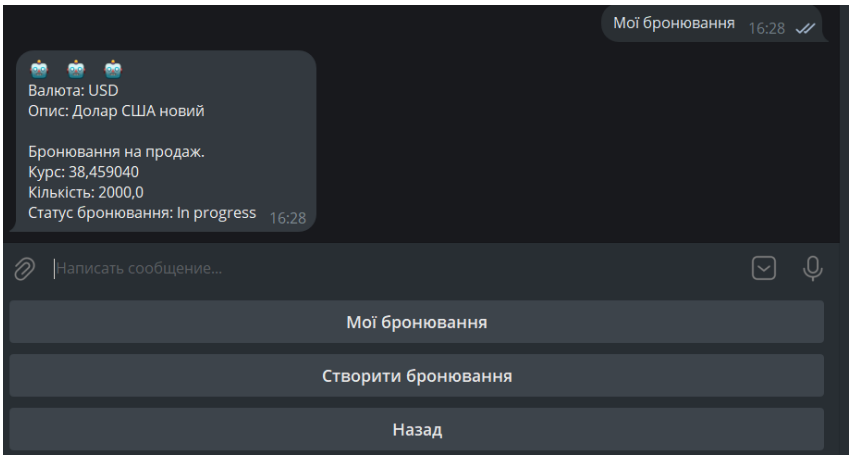


Рисунок 4.63 – Повідомлення з створеними бронюваннями у системі для цього контакту

Всі дані з приводу нового контакту і бронювань також будуть відображені адміністраторам у їх системі, де вони уже оперуватимуть ними за потреби.

Висновки до розділу

В цьому розділі проведено всебічний аналіз якості програмного забезпечення з використанням різних метрик та інструментів. Застосовано платформу CodeScene для оцінки коду проєкту за такими параметрами, як кількість рядків коду, кількість дефектів, складність коду та повторне використання.

Результати аналізу показали, що:

- код відповідає прийнятим стандартам і не містить зайвих рядків;
- виявлено кілька незначних дефектів, які швидко виправлені;
- код має прийнятний рівень складності, що сприяє легкості його підтримки та розширення;
- виявлено декілька областей, де можна покращити повторне використання коду.

Платформа CodeScene виявила проблеми з метрикою Maintainability, що свідчить про наявність певних труднощів у підтримці коду. Однак, по всім іншим метрикам, зазначеним у розділі, проблем не виявлено.

Окрім автоматизованого аналізу, проведено низку мануальних тестів для перевірки основних функцій програмного забезпечення. Тести охоплювали введення користувачем хибної інформації, наприклад, некоректних назв валют

На основі проведеного аналізу та тестування, внесено необхідні корективи та адаптовано код згідно результатів тестів, що дозволило підвищити загальну якість програмного забезпечення.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Клієнтську та серверну частини програмного забезпечення вирішено розгорнути на платформі Windows Server 2022. Для перенесення коду застосовано платформу GitHub.

Для розгортання бази даних, потрібно з офіційного сайту PostgreSql завантажити інсталятор. Офіційний сайт інсталятору зображений на рисунку 5.1[10].

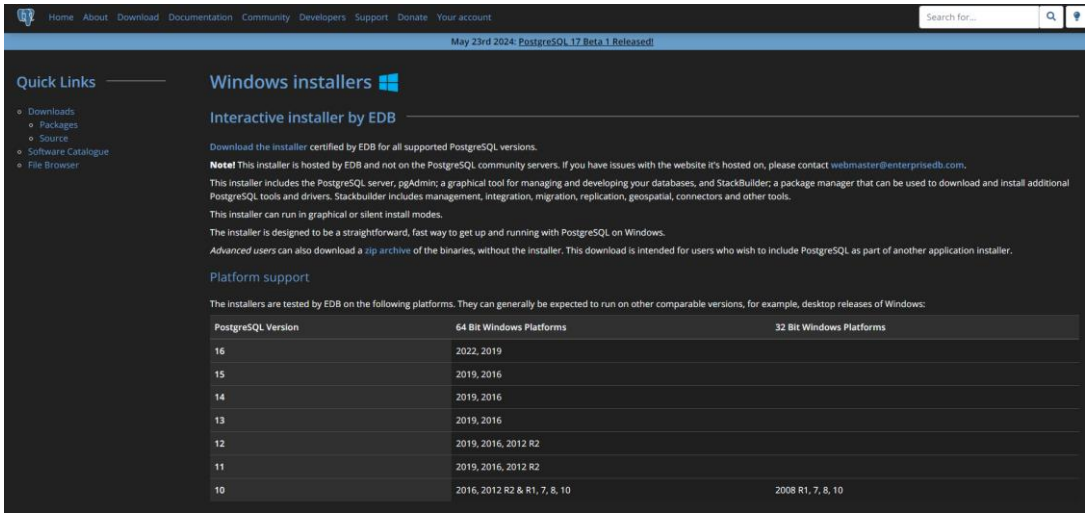


Рисунок 5.1 – Офіційний інсталятор PostgreSql

Після завантаження інсталятору, запускаємо його і робимо встановлення, рисунок 5.2.

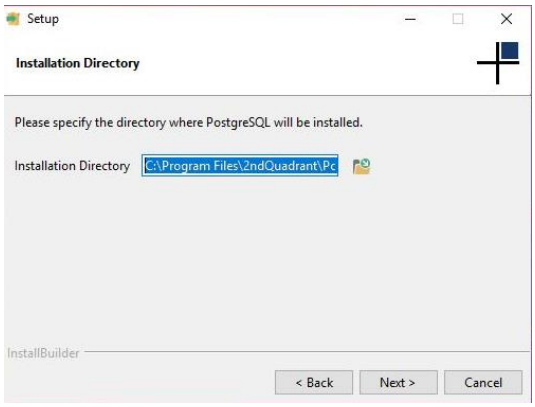


Рисунок 5.2 – Встановлення бази даних

Далі, потрібно встановити редіс. Для цього відкриємо powershell з правами адміністратора та виконуємо команду на рисунку 5.3.

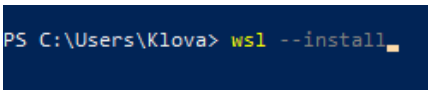


Рисунок 5.3 – Команда для встановлення wsl

Далі, з меню пуск, відкриємо встановлений дистрибутив рисунок 5.4.

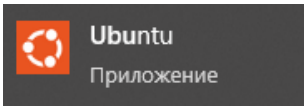


Рисунок 5.4 – Ubuntu застосунок

Виконаємо команди, які зображені на рисунку 5.5.

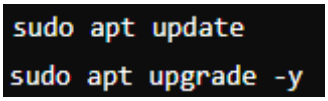


Рисунок 5.5 – Команди для оновлення Ubuntu пакетів

Встановимо Redis, тому потрібно виконати команди на рисунку 5.6:

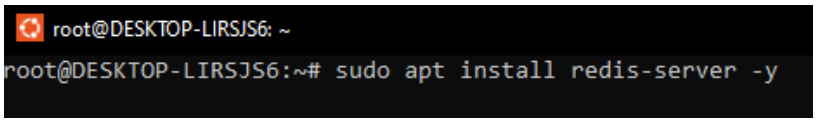


Рисунок 5.6 – Команда для встановлення Redis

Після успішного встановлення запутимо редіс командою з рисунку 5.7:

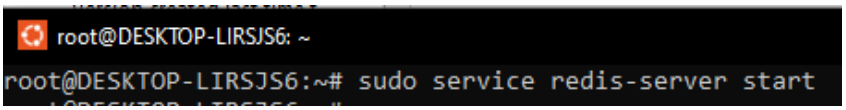


Рисунок 5.7 – Команда для запуску Redis серверу

Впевнимось, що Redis працює, тому виконаємо команду на рисунку 5.8:

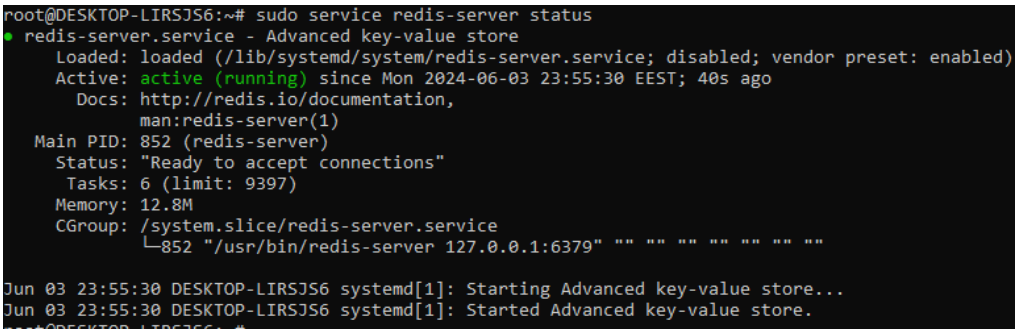


Рисунок 5.8 – Команда для перевірки роботи Redis серверу

Встановимо git з офіційного сайту рисунок 5.9[11].

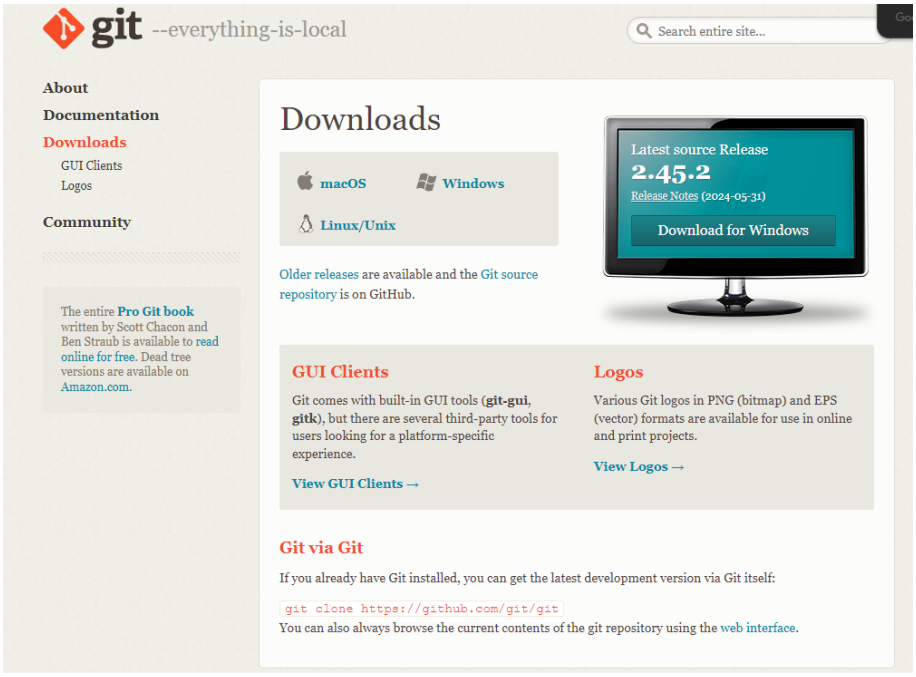


Рисунок 5.9 – Офіційний сайт git

Завантажимо проєкт з github репозиторію рисунки 5.10-5.12.

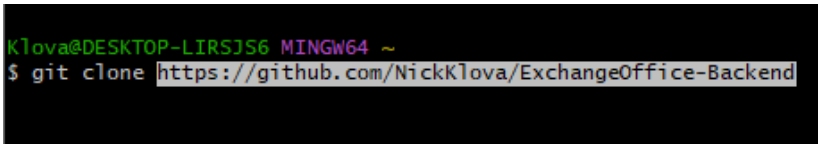


Рисунок 5.10 – Клонування репозиторію серверної частини з github

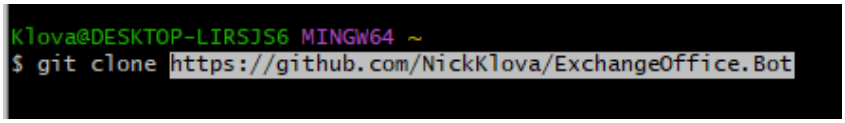


Рисунок 5.11 – Клонування репозиторію боту з github



Рисунок 5.12 – Клонування репозиторію клієнта з github

Переходимо у папку з проєктами серверної частини і боту, після виконуємо команди на рисунках 5.13 – 5.18.

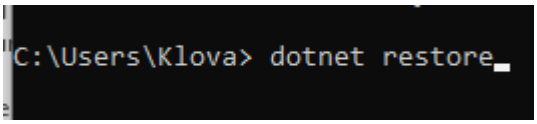


Рисунок 5.13 – Встановлення залежностей

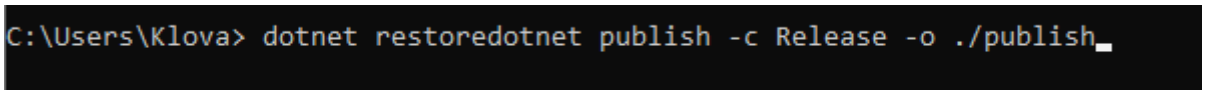


Рисунок 5.14 – Публікація проєкту

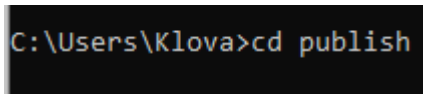


Рисунок 5.15 – Переходимо в папку з опублікованим проєктом

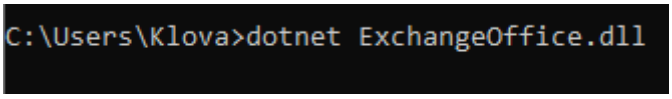


Рисунок 5.16 – Запускаємо проєкт

Після успішного розгортання серверних застосунків, переходимо у папку з клієнтською частиною і виконуємо команди на рисунках 5.17-5.18.

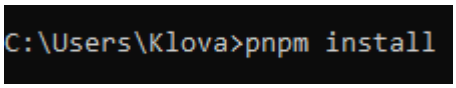


Рисунок 5.17 – Встановлюємо залежності



Рисунок 5.18 – Запускаємо сайт

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Висновки до розділу

У цьому розділі детально розглянуто процес розгортання клієнтської та серверної частин програмного забезпечення на платформі Windows Server 2022 з використанням коду з репозиторію GitHub.

Розгортання бази даних:

Для розгортання СУБД PostgreSQL завантажено офіційний інсталятор з сайту PostgreSQL. Після завантаження інсталяційного файлу виконано встановлення СУБД.

Встановлення Redis:

Для встановлення Redis використано WSL (Windows Subsystem for Linux). Спочатку виконано команду для встановлення WSL з PowerShell з правами адміністратора. Після встановлення WSL, з меню "Пуск" відкрито дистрибутив Ubuntu, де виконано команди для оновлення пакетів та встановлення Redis. Після успішного встановлення Redis, сервер запущено і перевірено його роботу.

Встановлено Git з офіційного сайту. З GitHub клоновано репозиторії серверної частини, боту та клієнта.

Після клонування репозиторіїв, виконано команди для встановлення залежностей, публікації проєкту та запуску серверних застосунків.

Після успішного розгортання серверної частини, виконано команди для встановлення залежностей та запуску клієнтського сайту.

Таким чином, весь процес розгортання програмного забезпечення успішно виконано, включаючи налаштування СУБД PostgreSQL, встановлення та налаштування Redis, клонування необхідних репозиторіїв з GitHub, та розгортання клієнтської і серверної частин застосунку. Це дозволяє забезпечити повноцінне функціонування програмного забезпечення на платформі Windows Server 2022.

ВИСНОВКИ

Під час виконання дипломного проєкту досліджено ринок систем обліку пунктів обміну валюти і побудовано власне програмне забезпечення для покращення та автоматизації обліку роботи пункту обміну валюти з урахуванням переваг та недоліків існуючих програмних рішень. Розроблена система має більш інтуїтивно зрозумілий інтерфейс, зручний доступ до даних та безкоштовне використання.

Створене програмне забезпечення вклшючає в себе функціонал для клієнта: додавання свого контакту, перегляд та створення свого бронювання, перегляд курсів валюти та валюти з якими працює пункт обміну. Робітники бізнесу мають можливість створювати, редагувати та додавати нові валюти у систему, переглядати курси валюти та наявні резерви. Переглядати, створювати або оновлювати транзакції і бронювання валюти. Можна зробити висновок, що мета роботи була успішно досягнута.

Розглянуто та описано аспекти розробки програмного забезпечення для покращення та автоматизації обліку роботи пункту обміну валюти на основі шаблону клієнт-серверної архітектури.

Результатом виконання проєкту є спроєктована система для автоматизації та обліку роботи пунктів обміну валюти. В якості середовища розробки використано Microsoft Visual Studio та JetBrains Web Storm. У якості СУБД використано PostgreSql і Redis.

Досліджувана тема програмного забезпечення для обліку та автоматизації роботи пункту обміну валюти є актуальною на ринку систем обліку, через необхідність зменшення ручних операцій. Перспективи вивчення полягають в подальшому розвитку і вдосконаленні програмного забезпечення. Для детальнішого аналізу ефективності роботи бізнесу можна додати різні дашборди у систему та розширити функціонал бота та самої системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Violity shop [Електронний ресурс] – Режим доступу до ресурсу: https://violity.shop/
2) Creatio [Електронний ресурс] – Режим доступу до ресурсу: https://www.creatio.com
3) Salesforce [Електронний ресурс] – Режим доступу до ресурсу: https://www.salesforce.com
4) Zoho [Електронний ресурс] – Режим доступу до ресурсу: https://www.zoho.com
5) Minfin [Електронний ресурс] – Режим доступу до ресурсу: https://minfin.com.ua
6) Національний банк України [Електронний ресурс] – Режим доступу до ресурсу: https://bank.gov.ua
7) PrivatBank [Електронний ресурс] – Режим доступу до ресурсу: https://privatbank.ua
8) Криптовалютна біржа Binance [Електронний ресурс] – Режим доступу до ресурсу: https://www.binance.com
9) Криптовалютна біржа OKX [Електронний ресурс] – Режим доступу до ресурсу: https://www.okx.com/
10) PostgreSql [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org
11) Git [Електронний ресурс] – Режим доступу до ресурсу: https://www.git-scm.com
12) CodeScene [Електронний ресурс] – Режим доступу до ресурсу: https://codescene.io

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016330531

Дата перевірки:
07.06.2024 07:56:49 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2024 10:28:50 EEST

ID користувача:
76913

Назва документа: IT-04_Кльова_ПЗ

Кількість сторінок: 103 Кількість слів: 12860 Кількість символів: 100923 Розмір файлу: 4.68 MB ID файлу: 1016130102

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

10.7%
Схожість

Найбільша схожість: 4.46% з джерелом з Бібліотеки (ID файлу: 1016123222)

1.81% Джерела з Інтернету 198 ..... Сторінка 105

10.7% Джерела з Бібліотеки 416 ..... Сторінка 107

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Підозріле форматування 24 сторінки

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Вебзастосунок для підтримки діяльності пункту обміну валют

Текст програми

КПІ.ІТ-0406.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Едуард ЖАРІКОВ

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Микола КЛЬОВА

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
Back-end: https://github.com/NickKlova/Currexchange-Backend
Front-end: https://github.com/NickKlova/Currexchange-Frontend
Telegram bot: https://github.com/NickKlova/Currexchange-Bot

1. BACK-END ЧАСТИНА ПРОГРАМИ

Файл ContactRepository.cs:

```
public class ContactRepository : BaseRepository, IContactRepository {
	#region Fields: Private

	private readonly DataAccessContext _context;
	private readonly IMapper _mapper;

	#endregion

	#region Constructors: Public

	public ContactRepository(DataAccessContext context, IMapper mapper) {
		_context = context;
		_mapper = mapper;
	}

	#endregion

	#region Methods: Public

	public async Task<IEnumerable<Contact>> GetContactsAsync() {
		return await Task.FromResult(_context.Contacts.Where(x =>
x.IsActive == true).AsNoTracking());
	}
```

```
        public async Task<Contact> GetContactAsync(Guid id) {
                var entity = await _context.Contacts.FindAsync(id);
                if(entity == null || entity.IsActive == false) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Contact with such id not found");
                }
                return entity;
        }
        public async Task<Contact> AddContactAsync(Contact entity) {
                SetDefaultValues(entity);
                await _context.Contacts.AddAsync(entity);
                await _context.SaveChangesAsync();
                return entity;
        }
        public async Task<Contact> UpdateContactAsync(Contact entity) {
                var oldEntity = await _context.Contacts.FindAsync(entity.Id);
                if (oldEntity == null || oldEntity.IsActive == false) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Contact with such id not found");
                }
                _mapper.Map(entity, oldEntity);
                await _context.SaveChangesAsync();
                return entity;
        }
        public async Task<Contact> DeactivateContactAsync(Guid id) {
                var entity = await _context.Contacts.FindAsync(id);
                if (entity == null || entity.IsActive == false) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Contact with such id not found");
                }
```

```
            entity.IsActive = false;
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Contact> DeleteContactAsync(Guid id) {
            var entity = await _context.Contacts.FindAsync(id);
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Contact with such id not found");
            }
            _context.Contacts.Remove(entity);
            await _context.SaveChangesAsync();
            return entity;
        }

        #endregion
}
```

Файл CurrencyRepository.cs:

```
public class CurrencyRepository : BaseRepository, ICurrencyRepository {
        #region Fields: Private

        private readonly DataAccessContext _context;
        private readonly IMapper _mapper;

        #endregion

        #region Constructors: Public

        public CurrencyRepository(DataAccessContext context, IMapper mapper) {
```

```
            _context = context;
            _mapper = mapper;
        }

        #endregion

        #region Methods: Public

        public async Task<IEnumerable<Currency>> GetCurrenciesAsync() {
            return await Task.FromResult(_context.Currencies
                .Where(x=>x.IsActive==true)
                .AsNoTracking());
        }
        public async Task<IEnumerable<Currency>> GetDeletedCurrenciesAsync()
{
            return await Task.FromResult(_context.Currencies
                .Where(x => x.IsActive == false)
                .AsNoTracking());
        }
        public async Task<Currency> GetCurrencyAsync(Guid id) {
            var entity = await _context.Currencies
                .Where(x=>x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Currency with such id not found");
            }
            return entity;
        }
        public async Task<Currency> AddCurrencyAsync(Currency entity) {
```

```
            SetDefaultValues(entity);
            await _context.Currencies.AddAsync(entity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Currency> UpdateCurrencyAsync(Currency entity) {
            var oldEntity = await _context.Currencies
                .Where(x=>x.Id == entity.Id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (oldEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Currency with such id not found");
            }
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Currency> ActivateCurrencyAsync(Currency entity) {
            var oldEntity = await _context.Currencies
                .Where(x => x.Id == entity.Id)
                .FirstOrDefaultAsync();
            if (oldEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Currency with such id not found");
            }
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Currency> DeleteCurrencyAsync(Guid id) {
```

```
            var entity = await _context.Currencies
                .Where(x=>x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Currency with such id not found");
            }
            SetDeleteDefaultValues(entity);
            await _context.SaveChangesAsync();
            return entity;
        }

        #endregion
}
```

Файл ExchangeSettingsRepository.cs:

```
public class ExchangeSettingsRepository : IExchangeSettingsRepository {
        private readonly DataAccessContext _context;
        public ExchangeSettingsRepository(DataAccessContext context) {
            _context = context;
        }

        public async Task<ExchangeSetting> GetSettingAsync() {
            var entity = await _context.ExchangeSettings
                .Include(x => x.BaseCurrency)
                .FirstOrDefaultAsync();
            if(entity == null) {
                throw new RecordNotFoundException(500, "DataAccess",
"SysSetting doesn't found!");
            }
```

```
            return entity;
        }

        public Task<ExchangeSetting> UpdateSettingAsync(ExchangeSetting
setting) {
            throw new NotImplementedException();
        }
}
```

Файл FundRepository.cs:

```
public class FundRepository : BaseRepository, IFundRepository {
        #region Fields: Private

        private readonly DataAccessContext _context;
        private readonly IMapper _mapper;

        #endregion

        #region Constructors: Public

        public FundRepository(DataAccessContext context, IMapper mapper) {
            _context = context;
            _mapper = mapper;
        }

        #endregion

        #region Methods: Public

        public async Task<IEnumerable<Fund>> GetDeletedFundsAsync() {
```

```
            return await Task.FromResult(_context.Funds
                .Include(x => x.Currency)
                .Where(x => x.IsActive == false)
                .AsNoTracking());
        }
        public async Task<IEnumerable<Fund>> GetFundsAsync() {
            return await Task.FromResult(_context.Funds
                .Include(x => x.Currency)
                .Where(x => x.IsActive == true)
                .AsNoTracking());
        }
        public async Task<Fund> GetFundAsync(Guid id) {
            var entity = await _context.Funds
                .Include(x => x.Currency)
                .Where(x=>x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such id not found");
            }
            return entity;
        }
        public async Task<Fund> GetFundByCurrencyIdAsync(Guid currencyId) {
            var entity = await _context.Funds
                .Include(x => x.Currency)
                .Where(x => x.CurrencyId == currencyId && x.IsActive ==
true)
                .FirstOrDefaultAsync();
            if (entity == null) {
```

```
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such currency id not found");
            }
            return entity;
        }
        public async Task<Fund> AddFundAsync(Fund entity) {
            SetDefaultValues(entity);
            await _context.Funds.AddAsync(entity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Fund> UpdateFundByCurrencyIdAsync(Guid
currencyId, decimal amount) {
            var entity = await _context.Funds
                .Where(x => x.CurrencyId == currencyId && x.IsActive ==
true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such currency id not found in database, so entity didn't updated");
            }
            entity.ModifiedOn = DateTime.UtcNow;
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Fund> UpdateFundAsync(Fund entity) {
            var oldEntity = await _context.Funds
                .Include(x=>x.Currency)
                .Where(x=>x.Id == entity.Id && x.IsActive == true)
                .FirstOrDefaultAsync();
```

```
            if (oldEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such id not found");
            }
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return oldEntity;

        }
        public async Task<Fund> ActivateDeletedFundAsync(Fund entity) {
            var oldEntity = await _context.Funds
                .Where(x => x.Id == entity.Id && x.IsActive == false)
                .FirstOrDefaultAsync();
            if (oldEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such id not found");
            }
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Fund> DeleteFundAsync(Guid id) {
            var entity = await _context.Funds
                .Where(x=>x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Fund with such id not found");
            }
            SetDeleteDefaultValues(entity);
```

```
            await _context.SaveChangesAsync();
            return entity;
        }

        #endregion
}
```

Файл RateRepository.cs:

```
public class RateRepository : BaseRepository, IRateRepository {
        #region Fields: Private

        private readonly DataAccessContext _context;
        private readonly IMapper _mapper;

        #endregion

        #region Constructors: Public

        public RateRepository(DataAccessContext context, IMapper mapper) {
            _context = context;
            _mapper = mapper;
        }

        #endregion

        #region Methods: Public

        public async Task<IEnumerable<Rate>> GetRatesAsync() {
            return await Task.FromResult(_context.Rates
                    .Include(x => x.Currency)
```

```
                .Where(x => x.IsActive == true)
                .AsNoTracking());
        }
        public async Task<IEnumerable<Rate>> GetDeletedRatesAsync() {
            return await Task.FromResult(_context.Rates
                .Include(x => x.Currency)
                .Where(x => x.IsActive == false)
                .AsNoTracking());
        }
        public async Task<Rate> GetRateAsync(Guid id) {
            var entity = await _context.Rates
                .Include(x => x.Currency)
                .Where(x => x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Rate with such id not found");
            }
            return entity;
        }
        public async Task<Rate> GetRateByCurrencyAsync(Guid currencyId) {
            var entity = await _context.Rates
                .Include(x=>x.Currency).Where(x => x.CurrencyId ==
currencyId
            && x.IsActive == true).FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Rate with such base and target currencies id not found");
            }
            return entity;
```

```
        }
        public async Task<Rate> AddRateAsync(Rate entity) {
                SetDefaultValues(entity);
                await _context.Rates.AddAsync(entity);
                await _context.SaveChangesAsync();
                return entity;
        }
        public async Task<Rate> UpdateRateAsync(Rate entity) {
                var oldEntity = await _context.Rates
                        .Include(x=>x.Currency)
                        .Where(x=>x.Id == entity.Id && x.IsActive == true)
                        .FirstOrDefaultAsync();
                if (oldEntity == null) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Rate with such id not found");
                }
                _mapper.Map(entity, oldEntity);
                await _context.SaveChangesAsync();
                return entity;
        }
        public async Task<Rate> ActivateDeletedRateAsync(Rate entity) {
                var oldEntity = await _context.Rates
                        .Include(x => x.Currency)
                        .Where(x => x.Id == entity.Id && x.IsActive == false)
                        .FirstOrDefaultAsync();
                if (oldEntity == null) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Rate with such id not found");
                }
                _mapper.Map(entity, oldEntity);
```

```
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Rate> DeleteRateAsync(Guid id) {
            var entity = await _context.Rates
                .Include(x => x.Currency)
                .Where(x => x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess", "A
rate with such id was not found in the database");
            }
            SetDeleteDefaultValues(entity);
            await _context.SaveChangesAsync();
            return entity;
        }

        #endregion
}
```

Файл ReservationRepository.cs:

```
public class ReservationRepository : BaseRepository, IReservationRepository {
        private readonly DataAccessContext _context;
        private readonly IMapper _mapper;
        public ReservationRepository(DataAccessContext context, IMapper
mapper) {
            _context = context;
            _mapper = mapper;
        }
```

```
        public async Task<IEnumerable<Reservation>> GetReservationsAsync() {
                return await Task.FromResult(_context.Reservations
                        .Include(x => x.OperationType)
                        .Include(x => x.Status)
                        .Include(x => x.Contact)
                        .Include(x => x.Rate)
                        .ThenInclude(r => r.Currency)
                        .Where(x=>x.IsActive)
                        .AsNoTracking());

        }
        public async Task<Reservation> GetReservationAsync(Guid id) {

                var entity = await _context.Reservations
                        .Include(x=>x.OperationType)
                        .Include(x=>x.Status)
                        .Include(x => x.Contact)
                        .Include(x => x.Rate)
                        .ThenInclude(r => r.Currency)
                        .Where(x => x.Id == id && x.IsActive == true)
                        .AsNoTracking()
                        .FirstOrDefaultAsync();
                if (entity == null) {
                        throw new RecordNotFoundException(404, "DataAccess",
"Reservation with such id not found");
                }
                return entity;
        }
        public async Task<Reservation> AddReservationAsync(Reservation entity)
{
```

```
            SetDefaultValues(entity);
            await _context.Reservations.AddAsync(entity);
            await _context.SaveChangesAsync();
            return entity;

        }
        public async Task<Reservation> UpdateReservationAsync(Reservation
entity) {
            var oldEntity = await _context.Reservations
                .Include(x => x.OperationType)
                .Include(x => x.Status)
                .Include(x => x.Contact)
                .Include(x => x.Rate)
                .ThenInclude(r => r.Currency)
                .Where(x => x.Id == entity.Id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (oldEntity == null) {
                throw new RecordNotFoundException(404, "DataLayer",
"Reservation with such id not found");
            }
            var operationStatus = await _context.OperationTypes.Where(x =>
x.Id == entity.OperationId).FirstOrDefaultAsync();
            if (operationStatus == null) {
                throw new RecordNotFoundException(404, "DataLayer",
"Reservation with such id not found");
            }
            entity.OperationType = operationStatus;
            var status = await _context.ReservationStatuses.Where(x => x.Id ==
entity.StatusId).FirstOrDefaultAsync();
            if (status == null) {
```

```
                throw new RecordNotFoundException(404, "DataLayer",
"Reservation with such id not found");
            }
            entity.Status = status;
            var contact = await _context.Contacts.Where(x => x.Id ==
entity.ContactId).FirstOrDefaultAsync();
            if (contact == null) {
                throw new RecordNotFoundException(404, "DataLayer",
"Reservation with such id not found");
            }
            entity.Contact = contact;
            var rate = await _context.Rates.Where(x => x.Id ==
entity.RateId).FirstOrDefaultAsync();
            if (rate == null) {
                throw new RecordNotFoundException(404, "DataLayer",
"Reservation with such id not found");
            }
            entity.Rate = rate;
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Reservation> DeleteReservationAsync(Guid id) {
            var entity = await _context.Reservations
                .Where(x=>x.Id==id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Reservation with such id not found");
            }
```

```
            SetDeleteDefaultValues(entity);
            await _context.SaveChangesAsync();
            return entity;
        }
}
```

Файл TransactionRepository.cs:

```
public class TransactionRepository : BaseRepository, ITransactionRepository {
        private DataAccessContext _context;
        private IMapper _mapper;
        public TransactionRepository(DataAccessContext context, IMapper mapper)
{
            _context = context;
            _mapper = mapper;
        }

        public async Task<IEnumerable<Transaction>> GetTransactionsAsync() {
            return await Task.FromResult(_context.Transactions
                .Where(x=>x.IsActive == true)
                .Include(x=>x.Contact)
                .Include(x=>x.RateLog)
                .Include(x=>x.RateLog.Currency)
                .Include(x=>x.TransactionType)
                .Include(x=>x.OperationType)
                .Include(x=>x.Reservation)
                .AsNoTracking());
        }
        public async Task<Transaction> CreateTransactionAsync(Transaction
entity) {
            SetDefaultValues(entity);
```

```
            SetDefaultValues(entity.RateLog);
            entity.RateLogId = entity.RateLog.Id;
            await _context.Transactions.AddAsync(entity);
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<Transaction> UpdateTransactionAsync(Transaction
entity) {
            var oldEntity = await _context.Transactions
                .Where(x=>x.Id == entity.Id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if(oldEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Transaction with such id didn't found");
            }
            _mapper.Map(entity, oldEntity);
            await _context.SaveChangesAsync();
            return oldEntity;
        }
        public async Task<Transaction> DeleteTransactionAsync(Guid id) {
            var entity = await _context.Transactions
                .Where(x=>x.Id == id && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Transaction with such id didn't found");
            }
            SetDeleteDefaultValues(entity);
            await _context.SaveChangesAsync();
            return entity;
```

```
        }
}
```

Файл UserRepository.cs:

```
public class UserRepository : BaseRepository, IUserRepository {
        private readonly DataAccessContext _context;
        public UserRepository(DataAccessContext context) {
                _context = context;
        }

        public async Task<IEnumerable<User>> GetUsersAsync() {
                return await Task.FromResult(_context.Users
                        .Include(x => x.Contact)
                        .Include(x => x.Role));
        }
        public async Task<User> GetUserAsync(string login) {
                var entity = await _context.Users
                        .Include(x=>x.Contact)
                        .Include(x=>x.Role)
                        .Where(x=>x.Login == login && x.IsActive == true)
                        .FirstOrDefaultAsync();
                if (entity == null) {
                        throw new RecordNotFoundException(404, "DataAccess",
"User with such login not found");
                }
                return entity;
        }
        public async Task<User> AddUserAsync(User entity) {
                SetDefaultValues(entity);
                await _context.Users.AddAsync(entity);
```

```
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<User> ChangeUserRoleAsync(string login, Guid roleId)
{
            var entity = await _context.Users
                .Where(x => x.Login == login && x.IsActive == true)
                .FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"User with such id not found");
            }
            var roleEntity = await _context.UserRoles.FindAsync(roleId);
            if (roleEntity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"Role with such id not found");
            }
            entity.RoleId = roleId;
            entity.Role = roleEntity;
            await _context.SaveChangesAsync();
            return entity;
        }
        public async Task<User> DeleteUserAsync(string login) {
            var entity = await _context.Users.Where(x=>x.Login ==
login).FirstOrDefaultAsync();
            if (entity == null) {
                throw new RecordNotFoundException(404, "DataAccess",
"User with such id not found");
            }
            SetDeleteDefaultValues(entity);
```

```
            await _context.SaveChangesAsync();
            return entity;
        }

}
```

Файл UserRoleRepository.cs:

```
public class UserRoleRepository : IUserRoleRepository {
        private readonly DataAccessContext _context;
        public UserRoleRepository(DataAccessContext context) {
            _context = context;
        }

        public async Task<IEnumerable<UserRole>> GetRolesAsync() {
            return await
Task.FromResult(_context.UserRoles.AsNoTracking().AsEnumerable());
        }
}
```

Файл DataAccessContext.cs:

```
public class DataAccessContext : DbContext {
        public DataAccessContext(DbContextOptions<DataAccessContext>
options) : base(options) { }

        public DbSet<Currency> Currencies { get; set; }
        public DbSet<Rate> Rates { get; set; }
        public DbSet<Fund> Funds { get; set; }
        public DbSet<Contact> Contacts { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<UserRole> UserRoles { get; set; }
```

```
public DbSet<Transaction> Transactions { get; set; }
public DbSet<Reservation> Reservations { get; set; }
public DbSet<OperationType> OperationTypes { get; set; }
public DbSet<ReservationStatus> ReservationStatuses { get; set; }
public DbSet<TransactionType> TransactionTypes { get; set; }
public DbSet<IssuanceLog> IssuanceLogs { get; set; }
public DbSet<ExchangeSetting> ExchangeSettings { get; set; }
protected override void OnModelCreating(ModelBuilder modelBuilder) {
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Rate>()
        .HasOne(r => r.Currency)
        .WithMany()
        .HasForeignKey(r => r.CurrencyId)
        .IsRequired()
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<Fund>()
        .HasOne(r => r.Currency)
        .WithMany()
        .HasForeignKey(r => r.CurrencyId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<User>()
        .HasOne(r => r.Role)
        .WithMany()
        .HasForeignKey(r => r.RoleId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<User>()
```

```
            .HasOne(r=>r.Contact)
            .WithMany()
            .HasForeignKey(r=>r.ContactId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<User>()
            .Property(r => r.ContactId)
            .IsRequired(false);

        modelBuilder.Entity<Transaction>()
            .HasOne(r => r.Contact)
            .WithMany()
            .HasForeignKey(r => r.ContactId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Transaction>()
            .HasOne(r => r.RateLog)
            .WithMany()
            .HasForeignKey(r => r.RateLogId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Transaction>()
            .HasOne(r => r.OperationType)
            .WithMany()
            .HasForeignKey(r => r.OperationId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Transaction>()
            .HasOne(r=>r.TransactionType)
            .WithMany()
```

```
            .HasForeignKey(r => r.TypeId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Reservation>()
            .HasOne(r => r.Contact)
            .WithMany()
            .HasForeignKey(r => r.ContactId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Reservation>()
            .HasOne(r => r.Rate)
            .WithMany()
            .HasForeignKey(r => r.RateId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Reservation>()
            .HasOne(r => r.Status)
            .WithMany()
            .HasForeignKey(r => r.StatusId);

        modelBuilder.Entity<Reservation>()
            .HasOne(r => r.OperationType)
            .WithMany()
            .HasForeignKey(r => r.OperationId);

        modelBuilder.Entity<UserRole>()
            .HasData(InitialDataExtension.GetUserRoles());

        modelBuilder.Entity<Currency>()
            .HasData(InitialDataExtension.GetCurrencies());
```

```
            modelBuilder.Entity<Currency>()
                .HasIndex(e => e.BankGovId)
                .IsUnique();

            modelBuilder.Entity<Fund>()
                .HasData(InitialDataExtension.GetFunds());

            modelBuilder.Entity<ReservationStatus>()
                .HasData(InitialDataExtension.GetReservationStatuses());

            modelBuilder.Entity<OperationType>()
                .HasData(InitialDataExtension.GetOperationTypes());

            modelBuilder.Entity<TransactionType>()
                .HasData(InitialDataExtension.GetTransactionTypes());

            modelBuilder.Entity<ExchangeSetting>()
                .HasData(InitialDataExtension.GetExchangeSettings());
        }
}
```

Файл ContactManager.cs:

```
public class ContactManager : IContactManager {
    private readonly IContactService _service;
    public ContactManager(IContactService service) {
        _service = service;
    }

    public async Task<IEnumerable<ContactDto>> GetContactsAsync() {
```

```
            return await _service.GetContactsAsync();
        }
        public async Task<ContactDto> GetContactAsync(Guid id) {
            return await _service.GetContactAsync(id);
        }
        public async Task<ContactDto> AddContactAsync(InsertContactDto entity)
{
            return await _service.AddContactAsync(entity);
        }
        public async Task<ContactDto> UpdateContactAsync(Guid id,
InsertContactDto entity) {
            return await _service.UpdateContactAsync(id, entity);
        }
        public async Task<ContactDto> DeactivateContactAsync(Guid id) {
            return await _service.DeactivateContactAsync(id);
        }
        public async Task<ContactDto> DeleteContactAsync(Guid id) {
            return await _service.DeleteContactAsync(id);
        }
}
```

Файл CurrencyManager.cs:

```
public class CurrencyManager : ICurrencyManager {
        #region Fields: Private
        private readonly ICurrencyService _service;
        private readonly IFundManager _fundManager;
        private readonly IRateManager _rateManager;
        private readonly IBankGovManager _bankGovManager;
        private readonly IMapper _mapper;
        #endregion
```

```
        #region Constructors: Public
        public CurrencyManager(ICurrencyService service, IFundManager
fundManager, IRateManager rateManager, IBankGovManager bankGovManager,
IMapper mapper) {
                _service = service;
                _fundManager = fundManager;
                _bankGovManager = bankGovManager;
                _rateManager = rateManager;
                _mapper = mapper;
        }
        #endregion

        #region Methods: Private

        private async Task<FundDto> CreateFundAsync(Guid currencyId) {
                var fundEntity = new InsertFundDto() {
                        CurrencyId = currencyId,
                        Amount = 0
                };
                return await _fundManager.AddFundAsync(fundEntity);
        }
        private async Task<BankGovRate> GetBankGovRate(string bankGovId) {
                var rates = await _bankGovManager.GetRatesAsync();
                var rate = rates.Where(x => x.Id == bankGovId).FirstOrDefault();
                if (rate == null) {
                        // TODO: Change exception to global commons
                        throw new Exception("");
                }
                return rate;
```

```
    }
    private async Task<RateDto> CreateRateAsync(Guid currencyId, string
bankGovId) {
        var rate = await GetBankGovRate(bankGovId);
        var rateEntity = new InsertRateDto() {
            CurrencyId = currencyId,
            // TODO: % of income change to another place
            BuyRate = rate.Rate * 0.95m,
            SellRate = rate.Rate * 1.05m,
        };
        return await _rateManager.AddRateAsync(rateEntity);
    }

    private async Task ActivateDeletedCurrency(CurrencyDto entity,
InsertCurrencyDto data) {
        await _service.ActivateCurrencyAsync(entity.Id, data);
        var deletedFunds = await _fundManager.GetDeletedFundsAsync();
        var deletedFund = deletedFunds.Where(x => x.Currency != null &&
x.Currency.Id == entity.Id).FirstOrDefault();
        if (deletedFund != null) {
            var insertFund = _mapper.Map<InsertFundDto>(deletedFund);
            await
_fundManager.ActivateDeletedFundAsync(deletedFund.Id, insertFund);
        }

        var deletedRates = await _rateManager.GetDeletedRatesAsync();
        var deletedRate = deletedRates.Where(x => x.Currency != null &&
x.Currency.Id == entity.Id).FirstOrDefault();
        if (deletedRate != null) {
            if (string.IsNullOrEmpty(entity.BankGovId)) {
```

```
                    // TODO: Change exception to global commons
                    throw new Exception("");
                }
                var rate = await GetBankGovRate(entity.BankGovId);
                var rateEntity = new InsertRateDto() {
                    CurrencyId = entity.Id,
                    // TODO: % of income change to another place
                    BuyRate = rate.Rate * 0.95m,
                    SellRate = rate.Rate * 1.05m,
                };
                await _rateManager.ActivateDeletedRateAsync(deletedRate.Id,
rateEntity);
            }
        }

        #endregion

        #region Methods: Public

        public async Task<IEnumerable<CurrencyDto>> GetCurrenciesAsync() {
            return await _service.GetCurrenciesAsync();
        }
        public async Task<IEnumerable<BankGovCurrency>>
GetBankGovCurrencies() {
            return await _bankGovManager.GetCurrenciesAsync();
        }
        public async Task<CurrencyDto> GetCurrencyAsync(Guid id) {
            return await _service.GetCurrencyAsync(id);
        }
```

```
        public async Task<CurrencyDto> AddCurrencyAsync(InsertCurrencyDto
data) {
                var deletedCurrencies = await _service.GetDeletedCurrenciesAsync();
                var deletedCurrency = deletedCurrencies.Where(x=>x.Code ==
data.Code).FirstOrDefault();
                if (deletedCurrency != null) {
                        await ActivateDeletedCurrency(deletedCurrency, data);
                }

                var currency = await _service.AddCurrencyAsync(data);
                await CreateFundAsync(currency.Id);
                if (!string.IsNullOrEmpty(currency.BankGovId)) {
                        await CreateRateAsync(currency.Id, currency.BankGovId);
                }
                return currency;
        }
        public async Task<CurrencyDto> UpdateCurrencyAsync(Guid id,
InsertCurrencyDto entity) {
                return await _service.UpdateCurrencyAsync(id, entity);
        }
        public async Task<CurrencyDto> DeleteCurrencyAsync(Guid id) {
                var fund = await _fundManager.GetFundByCurrencyIdAsync(id);
                await _fundManager.DeleteFundAsync(fund.Id);
                return await _service.DeleteCurrencyAsync(id);
        }

        #endregion
}
```

Файл ExchangeSettingManager.cs:

```
public class ExchangeSettingManager : IExchangeSettingManager {
	private readonly IExchangeSettingService _service;
	public ExchangeSettingManager(IExchangeSettingService service) {
		_service = service;
	}

	public async Task<ExchangeSettingDto> GetSettingAsync() {
		return await _service.GetSettingAsync();
	}
}
```

Файл FundManager.cs:

```
public class FundManager : IFundManager {
	#region Fields: Private

	private readonly IFundService _service;

	#endregion

	#region Constructors: Public

	public FundManager(IFundService service) {
		_service = service;
	}

	#endregion

	#region Methods: Public

	public async Task<IEnumerable<FundDto>> GetDeletedFundsAsync() {
```

```
            return await _service.GetDeletedFundsAsync();
        }
        public async Task<IEnumerable<FundDto>> GetFundsAsync() {
            return await _service.GetFundsAsync();
        }
        public async Task<FundDto> GetFundAsync(Guid id) {
            return await _service.GetFundAsync(id);
        }
        public async Task<FundDto> GetFundByCurrencyIdAsync(Guid
currencyId) {
            return await _service.GetFundByCurrencyIdAsync(currencyId);
        }
        public async Task<FundDto> AddFundAsync(InsertFundDto entity) {
            return await _service.AddFundAsync(entity);
        }
        public async Task<FundDto> UpdateFundByCurrencyIdAsync(Guid
currencyId, decimal amount) {
            return await _service.UpdateFundByCurrencyIdAsync(currencyId,
amount);
        }
        public async Task<FundDto> UpdateFundAsync(Guid id, InsertFundDto
entity) {
            return await _service.UpdateFundAsync(id, entity);
        }
        public async Task<FundDto> ActivateDeletedFundAsync(Guid id,
InsertFundDto entity) {
            return await _service.ActivateDeletedFundAsync(id, entity);
        }
        public async Task<FundDto> DeleteFundAsync(Guid id) {
            return await _service.DeleteFundAsync(id);
```

```
        }

        #endregion
}
```

Файл RateManager.cs:

```
public class RateManager : IRateManager {
        #region Fields: Private

        private readonly IRateService _service;

        #endregion

        #region Constructors: Public

        public RateManager(IRateService service) {
                _service = service;
        }

        #endregion

        #region Methods: Public

        public async Task<IEnumerable<RateDto>> GetRatesAsync() {
                return await _service.GetRatesAsync();
        }
        public async Task<IEnumerable<RateDto>> GetDeletedRatesAsync() {
                return await _service.GetDeletedRatesAsync();
        }
        public async Task<RateDto> GetRateAsync(Guid id) {
```

```
            return await _service.GetRateAsync(id);
        }
        public async Task<RateDto> GetRateByCurrencyAsync(Guid currencyId) {
            return await _service.GetRateByCurrencyAsync(currencyId);
        }
        public async Task<RateDto> AddRateAsync(InsertRateDto entity) {
            return await _service.AddRateAsync(entity);
        }
        public async Task<RateDto> UpdateRateAsync(Guid id, InsertRateDto
entity) {
            return await _service.UpdateRateAsync(id, entity);
        }
        public async Task<RateDto> DeleteRateAsync(Guid id) {
            return await _service.DeleteRateAsync(id);
        }
        public async Task<RateDto> ActivateDeletedRateAsync(Guid id,
InsertRateDto entity) {
            return await _service.ActivateDeletedRateAsync(id, entity);
        }

        #endregion
}
```

Файл ReservationManager.cs:

```
public class ReservationManager : IReservationManager {
        private readonly IReservationService _service;
        public ReservationManager(IReservationService service) {
            _service = service;

        }
```

```
        public async Task<ReservationDto>
AddReservationAsync(InsertReservationDto reservation) {
                return await _service.AddReservationAsync(reservation);
        }

        public async Task<ReservationDto> DeleteReservationAsync(Guid id) {
                return await _service.DeleteReservationAsync(id);
        }

        public async Task<IEnumerable<ReservationDto>>
GetReservationsAsync() {
                return await _service.GetReservationsAsync();
        }

        public async Task<ReservationDto> UpdateReservationAsync(Guid id,
InsertReservationDto reservation) {
                return await _service.UpdateReservationAsync(id, reservation);
        }

        public async Task<ReservationDto> GetReservationAsync(Guid id) {
                return await _service.GetReservationAsync(id);
        }

        public async Task<IEnumerable<ReservationDto>>
GetReservationsByContact(Guid contactId) {
                var reservations = await _service.GetReservationsAsync();
                return reservations.Where(x=>x.Contact != null && x.Contact.Id ==
contactId);
        }
```

Файл TransactionManager.cs:

```
public class TransactionManager : ITransactionManager {
        #region Fields: Private

        private readonly ITransactionService _service;
        private readonly IFundManager _fundManager;
        private readonly IExchangeSettingManager _settingManager;
        private readonly IMapper _mapper;
        private readonly IReservationManager _reservationManager;

        #endregion

        #region Constructors: Public

        public TransactionManager(ITransactionService service, IFundManager
fundManager, IExchangeSettingManager settingManager, IMapper mapper,
IReservationManager reservationManager) {
                _service = service;
                _fundManager = fundManager;
                _settingManager = settingManager;
                _mapper = mapper;
                _reservationManager = reservationManager;
        }

        #endregion

        #region Methods: Private

        private async Task UpdateCurrenciesFundsAsync(InsertTransactionDto
transaction) {
```

```
            var currencyFund = await
_fundManager.GetFundByCurrencyIdAsync(transaction.RateLog.CurrencyId);
            if (transaction.OperationId == OperationTypeConstants.Buy) {
                await UpdateFundForBuyTypeAsync(currencyFund,
transaction);
                return;
            }
            await UpdateFundForSellTypeAsync(currencyFund, transaction);
    }

    private async Task UpdateFundForSellTypeAsync(FundDto fund,
InsertTransactionDto transaction) {
            var setting = await _settingManager.GetSettingAsync();
            var baseFund = await
_fundManager.GetFundByCurrencyIdAsync(setting.BaseCurrencyId);
            var acceptedAmount = Math.Round(transaction.Amount *
transaction.RateLog.SellRate);
            baseFund.Amount = baseFund.Amount + acceptedAmount;
            var insertBaseFund = _mapper.Map<InsertFundDto>(baseFund);
            await _fundManager.UpdateFundAsync(baseFund.Id,
insertBaseFund);

            var returnedAmount = transaction.Amount;
            fund.Amount = fund.Amount - returnedAmount;
            var insertFund = _mapper.Map<InsertFundDto>(fund);
            await _fundManager.UpdateFundAsync(fund.Id, insertFund);
    }

    private async Task UpdateFundForBuyTypeAsync(FundDto fund,
InsertTransactionDto transaction) {
```

```
                var setting = await _settingManager.GetSettingAsync();
                var acceptedAmount = transaction.Amount;
                fund.Amount = fund.Amount + acceptedAmount;
                var insertFund = _mapper.Map<InsertFundDto>(fund);
                await _fundManager.UpdateFundAsync(fund.Id, insertFund);

                var baseFund = await
_fundManager.GetFundByCurrencyIdAsync(setting.BaseCurrencyId);
                var returnedAmount = Math.Round(transaction.Amount *
transaction.RateLog.BuyRate);
                baseFund.Amount = baseFund.Amount - returnedAmount;
                var insertBaseFund = _mapper.Map<InsertFundDto>(baseFund);
                await _fundManager.UpdateFundAsync(baseFund.Id,
insertBaseFund);
        }

        #endregion

        #region Methods: public

        public async Task<IEnumerable<TransactionDto>>
GetTransactionsAsync() {
                return await _service.GetTransactionsAsync();
        }
        public async Task<TransactionDto>
CreateTransactionAsync(InsertTransactionDto transaction) {
                if(transaction.TypeId == TransactionTypeConstants.Reservation) {
                        var reservationId = transaction.ReservationId ?? Guid.Empty;
                        var reservation = await
_reservationManager.GetReservationAsync(reservationId);
```

```
                var insertReservation =
_mapper.Map<InsertReservationDto>(reservation);
                insertReservation.StatusId =
ReservationStatusConstants.Issued;
                await
_reservationManager.UpdateReservationAsync(reservationId, insertReservation);
            }
            await UpdateCurrenciesFundsAsync(transaction);
            return await _service.CreateTransactionAsync(transaction);
        }

        #endregion
}
```

Файл UserManager.cs:

```
public class UserManager : IUserManager {
        private readonly IUserService _service;
        public UserManager(IUserService service) {
            _service = service;
        }

        public async Task<IEnumerable<UserDto>> GetUsersAsync() {
            return await _service.GetUsersAsync();
        }
        public async Task<AuthUserDto> LoginAsync(string login, string
password) {
            var user = await _service.GetUserAsync(login);
            if (user == null) {
                throw new Exception("Not found");
            }
```

```
            var isPassValid = JwtTokenProvider.VerifyPassword(password,
user.PasswordHash);
            if (!isPassValid) {
                throw new Exception("Pass not valid");
            }
            var authUser = new AuthUserDto();
            authUser.User = user;
            authUser.Token = JwtTokenProvider.GenerateToken(login,
user.Role.Name);
            return await Task.FromResult(authUser);
        }

        public async Task RegisterAsync(InsertUserDto data) {
            var passwordHash =
JwtTokenProvider.HashPassword(data.PasswordHash);
            data.PasswordHash = passwordHash;
            await _service.AddUserAsync(data);
        }

        public async Task ChangeUserRoleAsync(string login, Guid roleId) {
            await _service.ChangeUserRoleAsync(login, roleId);
        }

        public async Task DeactivateUserAsync(string login) {
            await _service.DeleteUserAsync(login);
        }
}
```

Файл UserRoleManager.cs:

```
public class UserRoleManager : IUserRoleManager {
        private readonly IUserRoleService _service;
        public UserRoleManager(IUserRoleService service) {
                _service = service;
        }

        public Task<IEnumerable<UserRoleDto>> GetRolesAsync() {
                return _service.GetRolesAsync();
        }
}
```

2. FRONT-END ЧАСТИНА ПРОГРАМИ

Файл Contacts.vue:

```
<script setup>
import contactService from '@/js/services/contactService'
import ContactModalWindow from
'../../../views/modals/ContactModalWindow.vue'
import { ref } from 'vue'
import { useSnackbar, Vue3Snackbar } from 'vue3-snackbar'

const snackbar = useSnackbar()
const contacts = ref([])
const isShownContactDialog = ref({})
const isLoading = ref(false)

onMounted(() => {
  setContacts()
  isShownContactDialog.value["new"] = false
})
```

```
function setContacts() {
  contactService.getContacts()
    .then(data => {
      contacts.value = data
    })
}

function pushCreatedContactToList(createdContact) {
  contacts.value.push(createdContact)
  snackbar.add({
    type: 'success',
    text: 'Contact created successfully!',
  })
  let closeObject = {
    id: "new",
    state: true,
  }
  closeContactDialog(closeObject)
}

const closeContactDialog = value => {
  isShownContactDialog.value[value.id] = !value.state
}

function updateContactInList(updatedContact) {
  let index = contacts.value.findIndex(cont => cont.id === updatedContact.id)
  if(index !== -1) {
    contacts.value[index] = updatedContact
```

```
    let closeObject = {
      id: updatedContact.id,
      state: true,
    }
    closeContactDialog(closeObject)
    snackbar.add({
      type: 'success',
      text: 'Contact updated successfully!',
    })
  }
}

async function deleteContact(id) {
  try {
    isLoading.value = true
    let data = await contactService.deleteContact(id)
    let contactIndex = contacts.value.findIndex(item => item.id === data.id)
    if (contactIndex !== -1) {
      contacts.value.splice(contactIndex, 1)
      snackbar.add({
        type: 'success',
        text: 'Contact deleted successfully!',
      })
    }
  } catch (error) {
    snackbar.add({
      type: 'error',
      text: error.message,
    })
  } finally {
```

```
    isLoading.value = false
  }
}
</script>

<template>
  <VCard>
    <!-- Header -->
    <VCardTitle>
      <div class="d-flex pa-2 align-center justify-space-between w-100">
        <div class="text-center flex-grow-1">
          Contact panel
        </div>
        <VBtn @click="isShownContactDialog['new']=true">
          <VIcon icon="ri-add-line" />
          <span>Add contact</span>
        </VBtn>

        <VDialog v-model="isShownContactDialog['new']">
          <ContactModalWindow
            @pushCreatedContactToList="pushCreatedContactToList"
            @closeAddContactDialog="closeContactDialog"
          />
        </VDialog>
      </div>
    </VCardTitle>
    <!--  -->
    <!-- List of currencies -->
    <VList>
      <!-- List title -->
```

```
<VListItem class="bg-var-theme-background">
  <VRow>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Name
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Email
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Phone number
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Is on blacklist
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
      <VListItemTitle class="font-weight-bold">
        Actions
      </VListItemTitle>
    </VCol>
```

```
  </VRow>
</VListItem>
<!-- -->

<!-- List content -->
<VListItem v-if="contacts.length === 0">
  <VRow>
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
      <VIcon
        icon="ri-database-2-fill"
        class="mr-3"
      />
      No data
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="contacts.length > 0"
  v-for="contact in contacts"
  :key="contact.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ contact.fullName }}
      </VListItemTitle>
```

```
</VCol>
<VCol>
  <VListItemTitle>
    {{ contact.email }}
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle>
    {{ contact.phoneNumber }}
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle>
    <VChip :color="contact.isBlacklist ? 'error' : 'success'">
      {{ contact.isBlacklist ? "yes" : "no" }}
    </VChip>
  </VListItemTitle>
</VCol>
<VCol
  cols="1"
  class="d-flex justify-center"
>
  <!-- Update currency button -->
  <VBtn
    color="secondary"
    variant="text"
    @click="isShownContactDialog[contact.id] = true"
  >
    <VIcon icon="ri-settings-4-fill" />
  </VBtn>
```

```
          <VDialog v-model="isShownContactDialog[contact.id]">
            <ContactModalWindow
              :contact="contact"
              @updateContactInList="updateContactInList"
              @closeAddContactDialog="closeContactDialog"
            />
          </VDialog>
          <!-- -->

          <!-- Delete currency button -->
          <VBtn
            color="secondary"
            variant="text"
            @click="deleteContact(contact.id)"
          >
            <VIcon icon="ri-delete-bin-5-line" />
          </VBtn>
          <!-- -->
        </VCol>
      </VRow>
    </VListItem>
  <!-- -->
  </VList>
<!-- -->
</VCard>

<VDialog
  v-model="isLoading"
  :persistent="isLoading"
```

```
  >
    <VProgressCircular
      v-if="isLoading"
      color="primary"
      :size="100"
      :width="10"
      indeterminate
      class="ma-auto"
    />
  </VDialog>

  <Vue3Snackbar
    bottom
    right
    duration="2000"
  />
</template>
```

Файл Currencies.vue:

```
<script setup>
import { ref, onMounted } from 'vue'
import CurrencyModalWindow from
'../../../views/modals/CurrencyModalWindow.vue'
import currencyService from '@/js/services/currencyService'
import { useSnackbar, Vue3Snackbar } from 'vue3-snackbar'

const snackbar = useSnackbar()

const actualCurrencies = ref([])
const isLoading = ref(false)
```

```
//region dialog

const isShownCurrencyDialog = ref({})

const closeCurrencyDialog = value => {
  isShownCurrencyDialog.value[value.id] = !value.state
}

const updateCurrencyInList = updatedCurrency => {
  let index = actualCurrencies.value.findIndex(currency => currency.id ===
updatedCurrency.id)
  if(index !== -1) {
    let foundCurrency = actualCurrencies.value[index]
    foundCurrency.code = updatedCurrency.code
    foundCurrency.description = updatedCurrency.description
    foundCurrency.symbol = updatedCurrency.symbol

    let closeObject = {
      id: updatedCurrency.id,
      state: true,
    }
    closeCurrencyDialog(closeObject)
    snackbar.add({
      type: 'success',
      text: 'Currency updated successfully!',
    })
  }
}
```

```
const pushCreatedCurrencyToList = createdCurrency => {
  let currency = {
    id: createdCurrency.id,
    code: createdCurrency.code,
    description: createdCurrency.description,
    symbol: createdCurrency.symbol,
  }

  actualCurrencies.value.push(currency)
  let closeObject = {
    id: "new",
    state: true,
  }
  closeCurrencyDialog(closeObject)
  snackbar.add({
    type: 'success',
    text: 'Currency created successfully!',
  })
}

//endregion

onMounted(async () => {
  await setActualCurrencies()
  isShownCurrencyDialog.value["new"] = false
})

//region value setters

async function setActualCurrencies() {
```

```
  actualCurrencies.value = await currencyService.getCurrencies()
}

//endregion

async function deleteCurrency(id) {
  try {
    //snackbar.clear()
    isLoading.value = true
    let deletedCurrency = await currencyService.deleteCurrency(id)
    let deletedCurrencyIndex = actualCurrencies.value.findIndex(item => item.id
=== deletedCurrency.id)
    if (deletedCurrencyIndex !== -1) {
      actualCurrencies.value.splice(deletedCurrencyIndex, 1)
    }
    snackbar.add({
      type: 'success',
      text: 'Currency deleted successfully!',
    })
  } catch (error) {
    snackbar.add({
      type: 'error',
      text: error.message,
    })
  } finally {
    isLoading.value = false
  }

}
</script>
```

```
<template>
  <VCard>
    <!-- Header -->
    <VCardTitle>
      <div class="d-flex pa-2 align-center justify-space-between w-100">
        <div class="text-center flex-grow-1">
          Currency panel
        </div>
        <VBtn @click="isShownCurrencyDialog['new'] = true" class="ml-auto">
          <VIcon icon="ri-add-line" />
          <span>Add currency</span>
        </VBtn>

        <VDialog v-model="isShownCurrencyDialog['new']">
          <CurrencyModalWindow
            :currencies="actualCurrencies"
            @push-created-currency-to-list="pushCreatedCurrencyToList"
            @close-add-currency-dialog="closeCurrencyDialog"
          />
        </VDialog>
      </div>
    </VCardTitle>
    <!--  -->

    <!-- List of currencies -->
    <VList>
      <!-- List title -->
      <VListItem class="bg-var-theme-background">
        <VRow>
```

```
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Code
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Description
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Symbol
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
      <VListItemTitle class="font-weight-bold">
        Actions
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>
<!-- -->

<!-- List content -->
<VListItem v-if="actualCurrencies.length === 0">
  <VRow>
```

```
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
      <VIcon
        icon="ri-database-2-fill"
        class="mr-3"
      />
      No data
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="actualCurrencies.length > 0"
  v-for="currency in actualCurrencies"
  :key="currency.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ currency.code }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ currency.description }}
      </VListItemTitle>
    </VCol>
    <VCol>
```

```
  <VListItemTitle>
    {{ currency.symbol }}
  </VListItemTitle>
</VCol>
<VCol
  cols="1"
  class="d-flex justify-center"
>
  <!-- Update currency button -->
  <VBtn
    color="secondary"
    variant="text"
    @click="isShownCurrencyDialog[currency.id] = true"
  >
    <VIcon icon="ri-settings-4-fill" />
  </VBtn>

  <VDialog v-model="isShownCurrencyDialog[currency.id]">
    <CurrencyModalWindow
      :currency="currency"
      @update-currency-in-list="updateCurrencyInList"
      @close-add-currency-dialog="closeCurrencyDialog"
    />
  </VDialog>
  <!-- -->

  <!-- Delete currency button -->
  <VBtn
    color="secondary"
    variant="text"
```

```
            @click="deleteCurrency(currency.id)"
          >
            <VIcon icon="ri-delete-bin-5-line" />
          </VBtn>
          <!-- -->
        </VCol>
      </VRow>
    </VListItem>
    <!-- -->
  </VList>
  <!-- -->
</VCard>

<VDialog
  v-model="isLoading"
  :persistent="isLoading"
>
  <VProgressCircular
    v-if="isLoading"
    color="primary"
    :size="100"
    :width="10"
    indeterminate
    class="ma-auto"
  />
</VDialog>

<Vue3Snackbar
  bottom
  right
```

```
    duration="2000"
  />
</template>
```

Файл ViewFunds.vue:

```
<script setup>
import fundService from '@/js/services/fundService'
import FundModalWindow from '../../../views/modals/FundModalWindow.vue'
import { Vue3Snackbar, useSnackbar } from "vue3-snackbar"

const snackbar = useSnackbar()

const funds = ref([])
const isShownFundDialog = ref({})

onMounted(() => {
  setFunds()
})

function setFunds() {
  fundService.getFunds().then(data=> {
    funds.value = data
  })
}

const closeFundDialog = value => {
  isShownFundDialog.value[value.id] = !value.state
}

const updateFundInList = updatedFund => {
```

```
  let index = funds.value.findIndex(fund => fund.id === updatedFund.id)
  if(index !== -1) {
    funds.value[index] = updatedFund
    let closeObject = {
      id: updatedFund.id,
      state: true,
    }
    closeFundDialog(closeObject)

    snackbar.add({
      type: "success",
      text: "Fund is updated successfully!"
    })
  }
}
</script>

<template>
  <VCard>
    <VCardTitle class="d-flex justify-center pa-5">
      Fund panel
    </VCardTitle>

    <VList>
      <VListItem class="bg-var-theme-background">
        <VRow>
          <VCol>
            <VListItemTitle class="font-weight-bold">
              Currency
            </VListItemTitle>
```

```
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Description
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Amount
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
      <VListItemTitle class="font-weight-bold">
        Actions
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>

<VListItem v-if="funds.length === 0">
  <VRow>
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
      <VIcon
        icon="ri-database-2-fill"
```

```
        class="mr-3"
      />
      No data
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="funds.length > 0"
  v-for="fund in funds"
  :key="fund.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ fund.currency.code }}
      </VListItemTitle>
    </VCol>
    <VCol>
      {{ fund.currency.description }}
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ fund.amount }}
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
```

```
            <!-- Update currency button -->
            <VBtn
              color="secondary"
              variant="text"
              @click="isShownFundDialog[fund.id] = true"
            >
              <VIcon icon="ri-settings-4-fill" />
            </VBtn>

            <VDialog v-model="isShownFundDialog[fund.id]">
              <FundModalWindow
                :fund="fund"
                @updateFundInList="updateFundInList"
                @closeModifyFundDialog="closeFundDialog"
              />
            </VDialog>
            <!-- -->
          </VCol>
        </VRow>
      </VListItem>
    </VList>
  </VCard>

  <Vue3Snackbar
    bottom
    right
    duration="2000"
  ></Vue3Snackbar>
</template>
```

Файл CreateRate.vue:

```
<script setup>
import { computed, ref, onMounted, watch } from 'vue'
import currencyService from '@/js/services/currencyService'
import rateService from '@/js/services/rateService'
import { useRouter } from 'vue-router'
import { Vue3Snackbar, useSnackbar } from "vue3-snackbar"
import isVarEmpty from "@/js/helper"

const snackbar = useSnackbar()
const router = useRouter()
const items = reactive(['Rate information', 'Confirmation'])
const step = ref(1)
const isLastItem = computed(() => step.value === items.length)

const acceptedCurrency = ref({
  id: null,
  code: null,
  description: null,
  symbol: null,
})

const returnedCurrency = ref({
  id: null,
  code: null,
  description: null,
  symbol: null,
})

const currencies = ref([])
```

```
const rate = ref(null)

const acceptedCurrencies = computed(() => {
  if (returnedCurrency.value !== null) {
    return [...currencies.value].filter(curr=>curr.id !== returnedCurrency.value.id)
  }

  return [...currencies.value]
})

const returnedCurrencies = computed(() => {
  if (acceptedCurrency.value !== null) {
    return [...currencies.value].filter(curr=>curr.id !== acceptedCurrency.value.id)
  }

  return [...currencies.value]
})

onMounted(() => {
  setCurrencies(currencies)
})

function isAllRequiredFieldFill() {
  return !(isVarEmpty(acceptedCurrency.value) ||
isVarEmpty(returnedCurrency.value) || isVarEmpty(rate.value))
}

function setCurrencies(reactiveVariable) {
  return currencyService.getCurrencies()
    .then(data => {
```

```
      reactiveVariable.value = data
    })
}

const nextOrSubmit = () => {
  if (isLastItem.value) {
    if (isAllRequiredFieldFill()) {
      return createRate()
    }
    let errorObj = {
      type: 'error',
      text: 'Fill all fields',
    }
    snackbar.add(errorObj)

    return
  }

  step.value++
}

function createRate() {
  let data = {
    "baseCurrencyId": acceptedCurrency.value.id,
    "targetCurrencyId": returnedCurrency.value.id,
    "sellRate": rate.value,
    "buyRate": rate.value,
  }
  rateService.createRate(data)
    .then(() => {
```

```
      router.push('/base')
    })
}
</script>

<template>
  <Vue3Snackbar
    bottom
    right
    :duration="2000"
  />
  <VCard>
    <VCardTitle>
      <h1 class="pa-5">
        Rate editor
      </h1>
    </VCardTitle>
    <VStepper
      v-model="step"
      :items="items"
    >
      <VStepperWindow v-if="step === 1">
        <VRow>
          <VCol class="pa-5">
            <VAutocomplete
              v-model="acceptedCurrency"
              label="Accepted currency"
              :items="acceptedCurrencies"
              return-object
              clearable
```

```
        item-title="code"
      />
    </VCol>
    <VCol class="pa-5">
      <VAutocomplete
        v-model="returnedCurrency"
        label="Returned currency"
        :items="returnedCurrencies"
        return-object
        clearable
        item-title="code"
      />
    </VCol>
  </VRow>
  <VRow>
    <VCol class="pa-5">
      <VTextField
        v-model="rate"
        label="Rate"
      />
    </VCol>
  </VRow>
</VStepperWindow>

<VStepperWindow v-if="step === 2">
  <VRow>
    <VCol class="pa-5">
      <VTextField
        v-model="acceptedCurrency.code"
        label="Accepted currency"
```

```
        disabled
      />
    </VCol>
    <VCol class="pa-5">
      <VTextField
        v-model="returnedCurrency.code"
        label="Returned currency"
        disabled
      />
    </VCol>
  </VRow>
  <VRow>
    <VCol class="pa-5">
      <VTextField
        v-model="rate"
        label="Rate"
        disabled
      />
    </VCol>
  </VRow>
</VStepperWindow>

<template #next>
  <VBtn
    :disabled="false"
    :color="isLastItem ? 'success' : 'primary'"
    @click="nextOrSubmit"
  >
    {{ isLastItem ? 'Submit' : 'Next' }}
  </VBtn>
```

```
      </template>
    </VStepper>
  </VCard>
</template>
```

Файл ViewRates.vue:

```
<script setup>
import rateService from '@/js/services/rateService'

const rates = ref([])

onMounted(() => {
  setRates()
})

function setRates() {
  rateService.getRates()
    .then(data => {
      rates.value = data
    })
}
</script>

<template>
  <VCard>
    <VCardTitle class="d-flex justify-center pa-5">
      Rate panel
    </VCardTitle>
    <VList>
      <VListItem class="bg-var-theme-background">
```

```
  <VRow>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Currency
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Description
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Buy rate
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Sell rate
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>

<VListItem v-if="rates.length === 0">
  <VRow>
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
```

```
      <VIcon
        icon="ri-database-2-fill"
        class="mr-3"
      />
      No data
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="rates.length > 0"
  v-for="rate in rates"
  :key="rate.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ rate.currency.code }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ rate.currency.description }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ rate.buyRate }}
      </VListItemTitle>
    </VCol>
```

```
          <VCol>
            <VListItemTitle>
              {{ rate.sellRate }}
            </VListItemTitle>
          </VCol>
        </VRow>
      </VListItem>
    </VList>
  </VCard>
</template>
```

Файл CreateReservation.vue:

```
<script setup>
import { computed, reactive, ref, watch } from 'vue'
import rateService from '@/js/services/rateService'
import contactService from '@/js/services/contactService'
import currencyService from '@/js/services/currencyService'
import { Vue3Snackbar, useSnackbar } from "vue3-snackbar"
import { useRouter } from 'vue-router'
import reservationService from '@/js/services/reservationService'
import isVarEmpty from '@/js/helper'
import { operationTypes } from '@/js/consts/operationTypesConsts'
import { reservationStatuses } from '@/js/consts/reservationStatusesConsts'
import { validationRules } from '@/js/validationRules.js'

const router = useRouter()
const snackbar = useSnackbar()

const isLoading = ref(false)
```

```
const isFormValid = ref(false)
const form = ref(null)

const contacts = ref([])
const currencies = ref([])

const contact = ref(null)
const reservationStatus = ref(null)
const currency = ref(null)
const amount = ref(null)
const returnedAmount = ref(null)
const rate = ref(null)
const operationType = ref(null)

const rateTextValue = computed(() => {
  if (operationType.value !== null && rate.value !== null) {
    if (operationType.value.id === "e3b0c442-98fc-1c14-9afd-2fb77c6076f6") {
      return rate.value.buyRate
    }
    return rate.value.sellRate
  }
  return ""
})

const isAmountDisabled = computed(() => {
  if (step.value > 1) {
    return true
  }

  return currency.value === null || operationType.value === null
```

```
})

const items = reactive(['General info', 'Preview'])
const step = ref(1)
const isLastItem = computed(() => step.value === items.length)

const nextOrSubmit = async () => {
  if (isLastItem.value) {
    return await createReservation()
  }
  if (step.value === 1 && !isFormValid.value) {
    form.value.validate()
    return
  }
  step.value++
}

onMounted(async () => {
  await setContacts()
  await setCurrencies()
})

//region watchers

watch(amount, (newValue) => {
  returnedAmount.value = Math.round((newValue * rateTextValue.value) * 100) /
100
})

watch(currency, async (newValue) => {
```

```
  if (newValue !== null) {
    return await getRate(newValue.id)
  }
  rate.value = null
  rateTextValue.value = null
})

//endregion

//region value setters

async function setContacts() {
  await contactService.getContacts()
    .then(data => {
      contacts.value = data
    })
}

async function setCurrencies() {
  await currencyService.getCurrencies()
    .then(data => {
      currencies.value = data
    })
}

async function getRate(currencyId) {
  await rateService.getRateByCurrency(currencyId)
    .then(data => {
      rate.value = data
    })
```

```
}

//endregion

const isFieldDisabled = computed(() => {
  if (step.value > 1) {
    return true
  }
  return false
})

//region validation rules

const contactValidationRules = computed(() => {
  let model = contact.value
  let rules = []

  rules.push(validationRules.isFieldRequired(model))
  return rules
})

const currencyValidationRules = computed(() => {
  let model = currency.value
  let rules = []

  rules.push(validationRules.isFieldRequired(model))
  return rules
})

const reservationStatusValidationRules = computed(() => {
```

```
  let model = reservationStatus.value
  let rules = []

  rules.push(validationRules.isFieldRequired(model))
  return rules
})

const operationTypeValidationRules = computed(() => {
  let model = operationType.value
  let rules = []

  rules.push(validationRules.isFieldRequired(model))
  return rules
})

const amountValidationRules = computed(() => {
  let model = amount.value
  let rules = []

  if(!isAmountDisabled.value) {
    rules.push(validationRules.isFieldRequired(model))

    let isPositiveAmount = typeof parseFloat(model) === 'number' &&
!isNaN(parseFloat(model)) && parseFloat(model) > 0
    rules.push((isPositiveAmount) || 'Amount must be a positive number')
  }

  return rules
})
```

```
//endregion

function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function createReservation() {
  let data = {
    contactId: contact.value.id,
    rateId: rate.value.id,
    operationId: operationType.value.id,
    statusId: reservationStatus.value.id,
    amount: amount.value,
  }
  isLoading.value = true
  let reservationCreatedSuccessfully = false
  try {
    await reservationService.createReservation(data)
    snackbar.add({
      type: 'success',
      text: `Reservation created successfully!`,
    })
    reservationCreatedSuccessfully = true
  } catch (error) {
    snackbar.add({
      type: 'error',
      text: error.message,
    })
  } finally {
    isLoading.value = false
```

```
    if(reservationCreatedSuccessfully) {
      await delay(1000)
      await router.push('/reservations/get')
    }
  }
}
</script>

<template>
  <VStepper
    v-model="step"
    :items="items"
  >
    <VStepperWindow>
      <VForm
        v-model="isFormValid"
        ref="form"
      >
        <VRow>
          <VCol class="pa-5">
            <VAutocomplete
              v-model="contact"
              label="Contact"
              :items="contacts"
              item-title="fullName"
              clearable
              :rules="contactValidationRules"
              return-object
              :disabled="isFieldDisabled"
            />
```

```
  </VCol>
</VRow>
<VRow>
  <VCol class="pa-5">
    <VAutocomplete
      v-model="currency"
      label="Currency"
      :items="currencies"
      item-title="code"
      return-object
      :rules="currencyValidationRules"
      clearable
      :disabled="isFieldDisabled"
    />
  </VCol>
  <VCol class="pa-5">
    <VSelect
      v-model="operationType"
      label="Operation Type"
      :items="operationTypes"
      item-title="name"
      :rules="operationTypeValidationRules"
      return-object
      :disabled="isFieldDisabled"
    />
  </VCol>
  <VCol class="pa-5">
    <VSelect
      v-model="reservationStatus"
      label="Reservation status"
```

```
      :items="reservationStatuses"
      item-title="name"
      :rules="reservationStatusValidationRules"
      return-object
      :disabled="isFieldDisabled"
    />
  </VCol>
</VRow>
<VRow>
  <VCol>
    <VTextField
      v-model="amount"
      hide-details="auto"
      type="number"
      label="Amount"
      :rules="amountValidationRules"
      :disabled="isAmountDisabled"
    />
  </VCol>
  <VCol class="pa-5">
    <VTextField
      v-model="rateTextValue"
      hide-details="auto"
      type="number"
      label="Rate"
      disabled
    />
  </VCol>
  <VCol class="pa-5">
    <VTextField
```

```
            v-model="returnedAmount"
            hide-details="auto"
            type="number"
            label="Returned amount"
            disabled
           />
          </VCol>
        </VRow>
      </VForm>
  </VStepperWindow>

  <template #next>
    <VBtn
      :disabled="false"
      :color="isLastItem ? 'success' : 'primary'"
      @click="nextOrSubmit"
    >
      {{ isLastItem ? 'Submit' : 'Next' }}
    </VBtn>
  </template>
</VStepper>

<VDialog
  v-model="isLoading"
  :persistent="isLoading"
>
  <VProgressCircular
    v-if="isLoading"
    color="primary"
    :size="100"
```

```
      :width="10"
      indeterminate
      class="ma-auto"
    />
  </VDialog>

  <Vue3Snackbar
    bottom
    right
    :duration="2000"
  />
</template>
```

Файл ViewReservations.vue:

```
<script setup>
import reservationService from '@/js/services/reservationService'
import CurrencyModalWindow from
'../../../views/modals/CurrencyModalWindow.vue'
import { ref } from 'vue'
import ReservationModalWindow from
'../../../views/modals/ReservationModalWindow.vue'

const isShownReservationDialog = ref({})

const closeReservationDialog = value => {
  isShownReservationDialog.value[value.id] = !value.state
}

const updateReservationInList = updatedReservation => {
  let index = reservations.value.findIndex(reservation => reservation.id ===
```

```
updatedReservation.id)
  if(index !== -1) {
    reservations.value[index].status = updatedReservation.status

    let closeObject = {
      id: updatedReservation.id,
      state: true,
    }
    closeReservationDialog(closeObject)
  }
}

const reservations = ref([])

onMounted(() => {
  setReservations()
})

function setReservations() {
  reservationService.getReservations()
    .then(data => {
      reservations.value = data
    })
}

function deleteReservation(id) {
  reservationService.deleteReservation(id)
    .then(deletedReservation => {
      let reservationIndex = reservations.value.findIndex(item => item.id ===
deletedReservation.id)
```

```
      if (reservationIndex !== -1) {
        reservations.value.splice(reservationIndex, 1)
      }
    })
}

function getColor(reservation) {
  if (reservation.status.id === '12345678-9abc-def0-1234-56789abcdef0') {
    return 'error'
  }
  if (reservation.status.id === '8e4f2c7d-91b3-4a5f-9b6d-3e7a8b9c1d0e') {
    return 'success'
  }
  if (reservation.status.id === 'a1b2c3d4-e5f6-7890-abcd-ef1234567890') {
    return 'secondary'
  }
  return 'primary'
}
</script>

<template>
  <VCard>
    <VCardTitle>
      <h1 class="pa-5">Reservation panel</h1>
    </VCardTitle>
    <VList>
      <VListItem class="bg-var-theme-background">
        <VRow>
          <VCol>
            <VListItemTitle class="font-weight-bold">
```

```
    Currency
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
    Operation type
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
    Rate
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
    Amount
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
    Returned amount
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
    Contact
  </VListItemTitle>
</VCol>
<VCol>
  <VListItemTitle class="font-weight-bold">
```

```
        Reservation status
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Date
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
      <VListItemTitle class="font-weight-bold">
        Actions
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-for="reservation in reservations"
  :key="reservation.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ reservation.rate.currency.code }}
      </VListItemTitle>
    </VCol>
    <VCol>
```

```
          <VListItemTitle>
            {{ reservation.operationType.name }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ reservation.operationType.id === "e3b0c442-98fc-1c14-9afd-
2fb77c6076f6" ? reservation.rate.buyRate : reservation.rate.sellRate }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ reservation.amount }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ reservation.operationType.id === "e3b0c442-98fc-1c14-9afd-
2fb77c6076f6" ? Math.round((reservation.rate.buyRate *
reservation.amount)*100)/100 : Math.round((reservation.rate.sellRate *
reservation.amount)*100)/100 }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ reservation.contact.fullName }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
```

```
            <VChip :color="getColor(reservation)">
              {{ reservation.status.name }}
            </VChip>
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ new Date(reservation.createdOn).toLocaleDateString() + " " + new
Date(reservation.createdOn).toLocaleTimeString() }}
          </VListItemTitle>
        </VCol>
        <VCol
          cols="1"
          class="d-flex justify-center"
        >
          <!-- Update reservation button -->
          <VBtn
            color="secondary"
            variant="text"
            @click="isShownReservationDialog[reservation.id] = true"
          >
            <VIcon icon="ri-settings-4-fill" />
          </VBtn>

          <VDialog v-model="isShownReservationDialog[reservation.id]">
            <ReservationModalWindow
              :reservation="reservation"
              @update-reservation-in-list="updateReservationInList"
              @close-reservation-dialog="closeReservationDialog"
            />
```

```
            </VDialog>
            <!-- -->
          </VCol>
        </VRow>
      </VListItem>
    </VList>
  </VCard>
</template>
```

Файл CreateTransaction.vue:

```
<script setup>
import { computed, reactive, ref, watch } from 'vue'

import rateService from '@/js/services/rateService'
import transactionService from '@/js/services/transactionService'
import currencyService from '@/js/services/currencyService'
import reservationService from '@/js/services/reservationService'
import contactService from '@/js/services/contactService'

import { Vue3Snackbar, useSnackbar } from 'vue3-snackbar'
import { useRouter } from 'vue-router'

import { operationTypes } from '@/js/consts/operationTypesConsts.js'
import { validationRules } from '@/js/validationRules.js'

const router = useRouter()
const snackbar = useSnackbar()

//region stepper fields
```

```
const items = reactive(['Info', 'Preview'])
const step = ref(1)
const isLastItem = computed(() => step.value === items.length)

//endregion

//region form fields

const isFormValid = ref(false)
const form = ref(null)

//endregion

//region fields

const isLoading = ref(false)
const isReservationType = ref(false)
const contacts = ref([])
const reservations = ref([])
const currencies = ref([])

const contact = ref()
const currency = ref(null)
const reservation = ref(null)
const operationType = ref(null)
const rate = ref(null)
const rateTextValue = ref(null)
const amount = ref(null)
const returnedAmount = ref(null)
```

```
//endregion

//region next or submit stepper

const nextOrSubmit = async () => {
  if (isLastItem.value) {
    return createTransaction()
  }
  if (step.value === 1 && !isFormValid.value) {
    form.value.validate()
    return
  }
  step.value++
}

//endregion

onMounted(async () => {
  await setContacts()
  await setCurrencies()
})

//region watchers

watch(contact, async (newValue) => {
  if (contact.value !== null) {
    await setReservations(contact.value.id)
  }
  if (newValue === null) {
    reservations.value = []
```

```
  }
})

watch(currency, async (newValue) => {
  if (!isReservationType.value) {
    if (currency.value !== null) {
      await setRate(newValue.id)
      rateTextValue.value = null
      operationType.value = null
    }
    if (currency.value === null) {
      rateTextValue.value = null
      operationType.value = null
      rate.value = null
      return
    }
  }
})

watch(operationType, async (newValue) => {
  if (!isReservationType.value) {
    if (newValue !== null) {
      if (newValue.id === "c2d4e3f6-63a4-4d58-bc63-ad6e2e7b3fb6") {
        return rateTextValue.value = rate.value.sellRate
      }
      return rateTextValue.value = rate.value.buyRate
    }
    rateTextValue.value = null
  }
})
```

```
watch(amount, (newValue) => {
  returnedAmount.value = Math.round((newValue * rateTextValue.value) * 100) /
100
})

watch(reservation, (newValue) => {
 currency.value = newValue.rate.currency
  operationType.value = newValue.operationType
  rate.value = newValue.rate
  amount.value = newValue.amount
  if (newValue.operationType.id === "e3b0c442-98fc-1c14-9afd-2fb77c6076f6") {
    rateTextValue.value = newValue.rate.buyRate
  }
  if (newValue.operationType.id === "c2d4e3f6-63a4-4d58-bc63-ad6e2e7b3fb6")
{
    rateTextValue.value = newValue.rate.sellRate
  }

})

watch(isReservationType, () => {
  form.value.reset()
})

//endregion

//region blocking fields

const isDisabledForPreview = computed(() => {
```

```
  if (isLastItem.value) {
    return true
  }
  return false
})

const isReservationFieldDisabled = computed(() => {
  if(isReservationType.value) {
    return true
  }
  if(isFieldDisabled.value) {
    return true
  }
  return false
})

const isFieldDisabled = computed(() => {
  if(isDisabledForPreview.value) {
    return true
  }
  return false
})

const isAmountDisabled = computed(() => {
  if (isDisabledForPreview.value) {
    return true
  }
  if(isReservationType.value) {
    return true
  }
```

```
  return currency.value === null || operationType.value === null
})

//endregion

//region value setters

async function setContacts() {
  await contactService.getContacts()
    .then(data => {
      contacts.value = data
    })
}

async function setCurrencies() {
  await currencyService.getCurrencies()
    .then(data => {
      currencies.value = data
    })
}

async function setReservations(contactId) {
  await reservationService.getReservationsByContact(contactId).then(data =>
reservations.value = data)
}

async function setRate(currencyId) {
  await rateService.getRateByCurrency(currencyId)
    .then(data => rate.value = data)
}
```

```
//endregion

//region validation rules

const contactValidationRules = computed(() => {
  let model = contact.value
  let rules = []

  if (isReservationType.value) {
    rules.push(validationRules.isFieldRequired(model))
  }
  return rules
})

const reservationValidationRules = computed(() => {
  let model = reservation.value
  let rules = []

  if (isReservationType.value) {
    rules.push(validationRules.isFieldRequired(model))
  }
  return rules
})

const amountValidationRules = computed(() => {
  let model = amount.value
  let rules = []

  if (!isReservationType.value && !isAmountDisabled.value) {
```

```
    rules.push(validationRules.isFieldRequired(model))

    let isPositiveAmount = typeof parseFloat(model) === 'number' &&
!isNaN(parseFloat(model)) && parseFloat(model) > 0
    rules.push((isPositiveAmount) || 'Amount must be a positive number')
  }
  return rules
})

const currencyValidationRules = computed(() => {
  let model = currency.value
  let rules = []

  if (!isReservationType.value) {
    rules.push(validationRules.isFieldRequired(model))
  }
  return rules
})

const operationTypeValidationRules = computed(() => {
  let model = operationType.value
  let rules = []

  if (!isReservationType.value) {
    rules.push(validationRules.isFieldRequired(model))
  }
  return rules
})

//endregion
```

```
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function createTransaction() {
  let data = {}
  if (isReservationType.value) {
    data = {
      contactId: contact.value.id,
      rateLog: {
        currencyId: rate.value.currency.id,
        buyRate: rate.value.buyRate,
        sellRate: rate.value.sellRate,
      },
      typeId: "D4F6E2A1-9C3B-4D8E-7F2A-5C1E3B4F6A9D",
      operationId: operationType.value.id,
      reservationId: reservation.value.id,
      amount: amount.value,
    }
  }
  if(!isReservationType.value) {
    data = {
      rateLog: {
        currencyId: rate.value.currency.id,
        buyRate: rate.value.buyRate,
        sellRate: rate.value.sellRate,
      },
      operationId: operationType.value.id,
      typeId: "F2C1A3E5-4D6B-4E8F-9A7C-5B1D3E4F6A7B",
```

```
      amount: amount.value,
    }
    if (!!contact.value) {
      data.contactId = contact.value.id
    }
  }

  isLoading.value = true
  let transactionCreatedSuccessfully = false
  try {
    await transactionService.createTransaction(data)
    snackbar.add({
      type: 'success',
      text: 'Transaction created successfully!',
    })
    transactionCreatedSuccessfully = true
  } catch (error) {
    snackbar.add({
      type: 'error',
      text: error.message,
    })
  } finally {
    isLoading.value = false
    if(transactionCreatedSuccessfully) {
      await delay(1000)
      await router.push('/transactions/get')
    }
  }
}
</script>
```

```
<template>
  <VStepper
    v-model="step"
    :items="items"
  >
    <VStepperWindow>
      <VSwitch class="pb-5 d-flex justify-end"
        v-model="isReservationType"
        label="Issuing a reservation"
         :disabled="isFieldDisabled"
      />
      <VForm
        v-model="isFormValid"
        ref="form"
      >
        <VRow>
          <VCol class="pa-5">
            <VAutocomplete
              v-model="contact"
              label="Contact"
              :items="contacts"
              item-title="fullName"
              return-object
              :disabled="isFieldDisabled"
              :rules="contactValidationRules"
              clearable
            />
          </VCol>
          <VCol class="pa-5">
```

```
    <VAutocomplete
      v-model="reservation"
      v-if="isReservationType"
      label="Reservation"
      :items="reservations"
      item-title="id"
      return-object
      :disabled="isFieldDisabled"
      :rules="reservationValidationRules"
      clearable
    />
  </VCol>
</VRow>
<VRow>
  <VCol class="pa-5">
    <VAutocomplete
      v-model="currency"
      label="Currency"
      :items="currencies"
      item-title="code"
      return-object
      clearable
      :disabled="isReservationFieldDisabled"
      :rules="currencyValidationRules"
    />
  </VCol>
  <VCol class="pa-5">
    <VAutocomplete
      v-model="operationType"
      label="Operation type"
```

```
      :items="operationTypes"
      item-title="name"
      return-object
      clearable
      :disabled="isReservationFieldDisabled"
      :rules="operationTypeValidationRules"
    />
  </VCol>
  <VCol class="pa-5">
    <VTextField
      v-model="amount"
      hide-details="auto"
      type="number"
      label="Amount"
      :disabled="isAmountDisabled"
      :rules="amountValidationRules"
    />
  </VCol>
</VRow>
<VRow>
  <VCol class="pa-5">
    <VTextField
      v-model="rateTextValue"
      hide-details="auto"
      type="number"
      label="Rate"
      disabled
      :disabled="isReservationFieldDisabled"
    />
  </VCol>
```

```
        <VCol class="pa-5">
          <VTextField
            v-model="returnedAmount"
            hide-details="auto"
            type="number"
            label="Returned amount"
            disabled
            :disabled="isReservationFieldDisabled"
          />
        </VCol>
      </VRow>
    </VForm>
  </VStepperWindow>

  <template #next>
    <VBtn
      :disabled="false"
      :color="isLastItem ? 'success' : 'primary'"
      @click="nextOrSubmit"
    >
      {{ isLastItem ? 'Submit' : 'Next' }}
    </VBtn>
  </template>
</VStepper>

<VDialog
  v-model="isLoading"
  :persistent="isLoading"
>
  <VProgressCircular
```

```
      v-if="isLoading"
      color="primary"
      :size="100"
      :width="10"
      indeterminate
      class="ma-auto"
    />
  </VDialog>

  <Vue3Snackbar
    bottom
    right
    :duration="2000"
  />
</template>
```

Файл ViewTransaction.vue:

```
<script setup>
import transactionService from '@/js/services/transactionService'

const transactions = ref([])

onMounted(() => {
  setTransactions()
})

function setTransactions() {
  transactionService.getTransactions()
    .then(data => {
      transactions.value = data
```

```
    })
}
</script>

<template>
  <VCard>
    <VCardTitle class="d-flex justify-center pa-5">
      Transaction panel
    </VCardTitle>
    <VList>
      <VListItem class="bg-var-theme-background">
        <VRow>
          <VCol>
            <VListItemTitle class="font-weight-bold">
              Currency
            </VListItemTitle>
          </VCol>
          <VCol>
            <VListItemTitle class="font-weight-bold">
              Operation
            </VListItemTitle>
          </VCol>
          <VCol>
            <VListItemTitle class="font-weight-bold">
              Buy rate
            </VListItemTitle>
          </VCol>
          <VCol>
            <VListItemTitle class="font-weight-bold">
              Sell rate
```

```
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Amount
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Reservation
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Date
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>

<VListItem v-if="transactions.length === 0">
  <VRow>
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
      <VIcon
        icon="ri-database-2-fill"
        class="mr-3"
      />
```

```
      No data
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="transactions.length > 0"
  v-for="transaction in transactions"
  :key="transaction.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ transaction.rateLog.currency.code }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ transaction.operationType.name }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ transaction.rateLog.buyRate }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ transaction.rateLog.sellRate }}
      </VListItemTitle>
```

```
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ transaction.amount + transaction.rateLog.currency.symbol }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ transaction.reservation == null ? "none" : transaction.reservation.id }}
          </VListItemTitle>
        </VCol>
        <VCol>
          <VListItemTitle>
            {{ new Date(transaction.createdOn).toLocaleDateString() + " " + new
Date(transaction.createdOn).toLocaleTimeString() }}
          </VListItemTitle>
        </VCol>
      </VRow>
    </VListItem>
  </VList>
 </VCard>
</template>
```

Файл Users.vue:

```
<script setup>
import CurrencyModalWindow from
'../../../views/modals/CurrencyModalWindow.vue'
import { useSnackbar, Vue3Snackbar } from 'vue3-snackbar'
import userService from '@/js/services/userService'
import { ref } from 'vue'
```

```
import UserModalWindow from '../../../views/modals/UserModalWindow.vue'

const snackbar = useSnackbar()

const users = ref([])
const isLoading = ref(false)

//region dialog

const isShownUserDialog = ref({})

//endregion

onMounted(async () => {
  await setUsers()
})

async function setUsers() {
  try {
    users.value = await userService.getAllUsers()
  } catch (error) {

  }
}

const updateUserInList = updatedUser => {
  let index = users.value.findIndex(usr => usr.id === updatedUser.id)
  if(index !== -1) {
    users.value[index] = updatedUser
```

```
    let closeObject = {
      id: updatedUser.id,
      state: true,
    }
    closeUserDialog(closeObject)
    snackbar.add({
      type: 'success',
      text: 'User updated successfully!',
    })
  }
}
const closeUserDialog = value => {
  isShownUserDialog.value[value.id] = !value.state
}
</script>

<template>
  <VCard>
    <!-- Header -->
    <VCardTitle>
      <div class="d-flex pa-2 align-center justify-space-between w-100">
        <div class="text-center flex-grow-1">
          User panel
        </div>
        <VBtn @click="isShownUserDialog['new'] = true" class="ml-auto">
          <VIcon icon="ri-add-line" />
          <span>Add user</span>
        </VBtn>

        <VDialog v-model="isShownUserDialog['new']">
```

```
      <UserModalWindow
        :users="users"
        :isDialogShown="isShownUserDialog"
      >

      </UserModalWindow>
    </VDialog>
  </div>
</VCardTitle>
<!--  -->

<!-- List of currencies -->
<VList>
  <!-- List title -->
  <VListItem class="bg-var-theme-background">
    <VRow>
      <VCol>
        <VListItemTitle class="font-weight-bold">
          Login
        </VListItemTitle>
      </VCol>
      <VCol>
        <VListItemTitle class="font-weight-bold">
          Contact
        </VListItemTitle>
      </VCol>
      <VCol>
        <VListItemTitle class="font-weight-bold">
          Role
        </VListItemTitle>
```

```
    </VCol>
    <VCol>
      <VListItemTitle class="font-weight-bold">
        Status
      </VListItemTitle>
    </VCol>
    <VCol
      cols="1"
      class="d-flex justify-center"
    >
      <VListItemTitle class="font-weight-bold">
        Actions
      </VListItemTitle>
    </VCol>
  </VRow>
</VListItem>
<!-- -->

<!-- List content -->
<VListItem v-if="users.length === 0">
  <VRow>
    <VCol
      cols="12"
      class="d-flex justify-center pa-15"
    >
      <VIcon
        icon="ri-database-2-fill"
        class="mr-3"
      />
      No data
```

```
    </VCol>
  </VRow>
</VListItem>

<VListItem
  v-if="users.length > 0"
  v-for="user in users"
  :key="user.id"
>
  <VRow>
    <VCol>
      <VListItemTitle>
        {{ user.login }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        {{ user.contact.fullName }}
      </VListItemTitle>
    </VCol>
    <VCol>
      <VListItemTitle>
        <VChip
          color="primary"
          label
        >
          {{ user.role.name }}
        </VChip>
      </VListItemTitle>
    </VCol>
```

```
<VCol>
  <VListItemTitle>
    <VChip :color="user.isActive ? 'success' : 'error'">
      {{ user.isActive ? 'active' : 'not active' }}
    </VChip>
  </VListItemTitle>
</VCol>
<VCol
  cols="1"
  class="d-flex justify-center"
>
  <!-- Update currency button -->
  <VBtn
    color="secondary"
    variant="text"
    @click="isShownUserDialog[user.id] = true"
  >
    <VIcon icon="ri-settings-4-fill" />
  </VBtn>

  <VDialog v-model="isShownUserDialog[user.id]">
    <UserModalWindow
      :user="user"
      :users="users"
      :isDialogShown="isShownUserDialog"
    />
  </VDialog>
  <!-- -->

  <!-- Delete currency button
```

```
          <VBtn
            color="secondary"
            variant="text"
            @click="deleteCurrency(currency.id)"
          >
            <VIcon icon="ri-delete-bin-5-line" />
          </VBtn>-->
          <!-- -->
        </VCol>
      </VRow>
    </VListItem>
    <!-- -->
  </VList>
  <!-- -->
</VCard>

<VDialog
  v-model="isLoading"
  :persistent="isLoading"
>
  <VProgressCircular
    v-if="isLoading"
    color="primary"
    :size="100"
    :width="10"
    indeterminate
    class="ma-auto"
  />
</VDialog>
```

```
  <Vue3Snackbar
    bottom
    right
    duration="2000"
  />
</template>
```

3. TELEGRAM BOT

Файл BaseReservationCreator.cs:

```
public abstract class BaseReservationCreator : BaseStepperHandler {
        #region Fields: Private

        private readonly ICacheManager _cacheManager;
        private readonly IRateManager _rateManager;
        private readonly string _tempReservationDataKey;

        #endregion

        #region Contructors: Public

        public BaseReservationCreator(IManagerProvider provider) : base(provider)
{
                _cacheManager = provider.GetCacheManager();
                _rateManager = provider.GetRateManager();
                _tempReservationDataKey = "TempReservationData";
        }

        #endregion

        #region Methods: Private
```

```
private string GetTempReservationDataKey(string key) {
        return _tempReservationDataKey + ":" + key;
}

#endregion

#region Methods: Protected

protected async Task SetContactIdValue(string key, string contactId) {
        var cacheKey = GetTempReservationDataKey(key);
        var data = new TempReservationData() {
                ContactId = new Guid(contactId)
        };

        var json = JsonConvert.SerializeObject(data, Formatting.Indented);
        await _cacheManager.SetAsync(cacheKey, json);
}

protected async Task SetCurrencyValue(string key, string currencyId) {
        var rate = await _rateManager.GetRateAsync(key, currencyId);
        if (rate == null) {
                throw new Exception();
        }

        var cacheKey = GetTempReservationDataKey(key);
        var json = await _cacheManager.GetAsync(cacheKey);
        if (string.IsNullOrEmpty(json)) {
                throw new Exception();
        }
```

```
            var entity =
JsonConvert.DeserializeObject<TempReservationData>(json);
            if (entity == null) {
                throw new Exception();
            }

            entity.Rate = rate;
            var modifiedJson = JsonConvert.SerializeObject(entity,
Formatting.Indented);
            await _cacheManager.SetAsync(cacheKey, modifiedJson);
        }

        protected async Task SetOperationTypeValue(string key, string
operationTypeId) {
            var cacheKey = GetTempReservationDataKey(key);
            var json = await _cacheManager.GetAsync(cacheKey);
            if (string.IsNullOrEmpty(json)) {
                throw new Exception();
            }

            var entity =
JsonConvert.DeserializeObject<TempReservationData>(json);
            if (entity == null) {
                throw new Exception();
            }

            entity.OperationId = new Guid(operationTypeId);
            var modifiedJson = JsonConvert.SerializeObject(entity,
Formatting.Indented);
```

```
            await _cacheManager.SetAsync(cacheKey, modifiedJson);
        }

        protected async Task SetAmountValue(string key, string amount) {
            var cacheKey = GetTempReservationDataKey(key);
            var json = await _cacheManager.GetAsync(cacheKey);
            if (string.IsNullOrEmpty(json)) {
                throw new Exception();
            }

            var entity =
JsonConvert.DeserializeObject<TempReservationData>(json);
            if (entity == null) {
                throw new Exception();
            }

            var isParcedToDecimal = decimal.TryParse(amount, out var
decimalValue);
            if(!isParcedToDecimal) {
                throw new Exception();
            }

            entity.Amount = decimalValue;
            var modifiedJson = JsonConvert.SerializeObject(entity,
Formatting.Indented);
            await _cacheManager.SetAsync(cacheKey, modifiedJson);
        }

        protected async Task<TempReservationData>
GetTempReservationDataAsync(string key) {
```

```
            var cacheKey = GetTempReservationDataKey(key);
            var json = await _cacheManager.GetAsync(cacheKey);
            if (string.IsNullOrEmpty(json)) {
                throw new Exception();
            }

            var entity =
JsonConvert.DeserializeObject<TempReservationData>(json);
            if (entity == null) {
                throw new Exception();
            }
            return entity;
        }

        protected async Task ClearTempReservationDataAsync(string key) {
            var cacheKey = GetTempReservationDataKey(key);
            await _cacheManager.DeleteAsync(cacheKey);
        }

        #endregion
}
```

Файл BaseStepperHandler.cs:

```
public abstract class BaseStepperHandler {
        #region Fields: Private

        private readonly ICacheManager _cacheManager;

        #endregion
```

```
        #region Contructors: Public

        public BaseStepperHandler(IManagerProvider provider) {
                _cacheManager = provider.GetCacheManager();
        }

        #endregion

        #region Methods: Protected

        protected async Task ConfigureStepperAsync(object key, StepperInfo
config) {
                var stringKey = key.ToString();
                if (stringKey == null) {
                        throw new Exception();
                }
                await NextOrFinishStepAsync(stringKey, config);
        }

        protected async Task DeleteStepperAsync(object key) {
                var stringKey = key.ToString();
                if (string.IsNullOrEmpty(stringKey)) {
                        return;
                }
                await _cacheManager.DeleteAsync(stringKey);
        }

        protected async Task NextOrFinishStepAsync(object key, StepperInfo?
value = null) {
                var stringKey = key.ToString();
```

```
            if (string.IsNullOrEmpty(stringKey)) {
                return;
            }
            if (value == null) {
                var valueJson = await _cacheManager.GetAsync(stringKey);
                if (valueJson == null) {
                    return;
                }
                value =
JsonConvert.DeserializeObject<StepperInfo>(valueJson);
                if (value == null) {
                    return;
                }
            }
            value!.CurrentStep++;
            if (value.CurrentStep > value.StepsCount) {
                await _cacheManager.DeleteAsync(stringKey);
                return;
            }
            var jsonValue = JsonConvert.SerializeObject(value);
            await _cacheManager.SetAsync(stringKey, jsonValue);
        }

        #endregion
}
```

Файл BackMessageHandler.cs:

```
[TextMessageHandler(ReservationMenuTitles.BackToMainMenu)]
public class BackMessageHandler : ITextHandler {
        #region Fields: Private
```

```
        private ITelegramBotClient _bot;

        #endregion

        #region Constructors: Public

        public BackMessageHandler(IManagerProvider provider) {
                _bot = provider.GetTelegramBotClient();
        }

        #endregion

        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
                var chatId = request?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                await _bot.SendTextMessageAsync(chatId, text: "Головне меню:",
replyMarkup: MainMenu.GetKeyboard());
        }

        #endregion
}
```

Файл StartMessageHandler.cs:

```
[TextMessageHandler("/start")]
```

```
public class StartMessageHandler : ITextHandler {
	#region Fields: Private

	private ITelegramBotClient _bot;

	#endregion

	#region Constructors: Public

	public StartMessageHandler(IManagerProvider provider) {
		_bot = provider.GetTelegramBotClient();
	}

	#endregion

	#region Methods: Public

	public async Task ExecuteAsync(Update request) {
		var chatId = request?.Message?.Chat.Id;
		if (chatId == null) {
			return;
		}

		await _bot.SendTextMessageAsync(chatId, text: "Головне меню:",
replyMarkup: MainMenu.GetKeyboard());
	}

	#endregion
}
```

Файл ContactMessageHandler.cs:

```
[TextMessageHandler(MainMenuTitles.MyContact)]
public class ContactMessageHandler : BaseStepperHandler, ITextHandler {
	#region Fields: Private

	private readonly ITelegramBotClient _bot;
	private readonly IContactManager _contactManager;
	private readonly ICacheManager _cacheManager;
	private readonly string _contactTempDataKey;

	#endregion

	#region Constructors: Public

	public ContactMessageHandler(IManagerProvider managerProvider) :
base(managerProvider) {
		_bot = managerProvider.GetTelegramBotClient();
		_contactManager = managerProvider.GetContactManager();
		_cacheManager = managerProvider.GetCacheManager();
		_contactTempDataKey = "ContactTempData";
	}

	#endregion

	#region Methods: Private

	private string GetContactMsg(Core.Models.Contact contact) {
		return "☎ ☎ ☎\n\n" +
			$"Особистий ключ: {contact.Id}\n" +
			$"ПІБ: {contact.FullName}\n" +
```

```
                $"Номер телефону: {contact.PhoneNumber}\n" +
                $"Пошта: {contact.Email}\n";
        }

        #endregion

        #region Methods: Private

        private string GetTempContactDataKey(string key) {
                return _contactTempDataKey + ":" + key;
        }

        private async Task SaveFullNameAsync(string key, string value) {
                var cacheKey = GetTempContactDataKey(key);
                var contact = new Core.Models.Contact() {
                        FullName = value
                };
                var json = JsonConvert.SerializeObject(contact);
                await _cacheManager.SetAsync(cacheKey, json);
        }

        private async Task SavePhoneNumberAsync(string key, string value) {
                var cacheKey = GetTempContactDataKey(key);
                var json = await _cacheManager.GetAsync(cacheKey);
                if (string.IsNullOrEmpty(json)) {
                        throw new Exception();
                }

                var entity =
JsonConvert.DeserializeObject<Core.Models.Contact>(json);
```

```
            if (entity == null) {
                throw new Exception();
            }

            entity.PhoneNumber = value;
            var resultJson = JsonConvert.SerializeObject(entity);
            await _cacheManager.SetAsync(cacheKey, resultJson);
        }

        private async Task<Core.Models.Contact>
GetSavedContactWithEmail(string key, string value) {
            var cacheKey = GetTempContactDataKey(key);
            var json = await _cacheManager.GetAsync(cacheKey);
            if (string.IsNullOrEmpty(json)) {
                throw new Exception();
            }

            var entity =
JsonConvert.DeserializeObject<Core.Models.Contact>(json);
            if (entity == null) {
                throw new Exception();
            }
            entity.Email = value;

            await _cacheManager.DeleteAsync(cacheKey);

            return entity;
        }

        #endregion
```

```
        #region Methods: Public

        [TextStepperHandler(MainMenuTitles.MyContact, 1)]
        public async Task ExecuteAsync(Update request) {
                var chatId = request?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }
                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }
                var contact = await _contactManager.GetContactAsync(key);
                if (contact != null) {
                        var registeredContactMessage = GetContactMsg(contact);
                        await _bot.SendTextMessageAsync(chatId,
registeredContactMessage, replyMarkup: MainMenu.GetKeyboard());
                        return;
                }

                var config = new StepperInfo() {
                        CurrentStep = 1,
                        StepsCount = 4,
                        Name = MainMenuTitles.MyContact,
                };
                await ConfigureStepperAsync(chatId, config);

                await _bot.SendTextMessageAsync(chatId, "Введіть ваше ПІБ:",
replyMarkup: MainMenu.GetKeyboard());
```

```
    }

    [TextStepperHandler(MainMenuTitles.MyContact, 2)]
    public async Task ProcessUsernameParameterAsync(Update request) {
        var chatId = request?.Message?.Chat.Id;
        if (chatId == null) {
            return;
        }
        var key = chatId.ToString();
        if (string.IsNullOrEmpty(key)) {
            return;
        }

        var text = request?.Message?.Text;
        if (string.IsNullOrEmpty(text)) {
            return;
        }
        await SaveFullNameAsync(key, text);

        await _bot.SendTextMessageAsync(chatId, "Введіть ваш
контактний номер телефону:", replyMarkup: MainMenu.GetKeyboard());
        await NextOrFinishStepAsync(chatId);
    }

    [TextStepperHandler(MainMenuTitles.MyContact, 3)]
    public async Task ProcessPhoneNumberParameterAsync(Update request) {
        var chatId = request?.Message?.Chat.Id;
        if (chatId == null) {
            return;
        }
```

```
            var key = chatId.ToString();
            if (string.IsNullOrEmpty(key)) {
                return;
            }

            var text = request?.Message?.Text;
            if (string.IsNullOrEmpty(text)) {
                return;
            }
            await SavePhoneNumberAsync(key, text);

            await _bot.SendTextMessageAsync(chatId, "Введіть вашу пошту:",
replyMarkup: MainMenu.GetKeyboard());
            await NextOrFinishStepAsync(chatId);
        }

        [TextStepperHandler(MainMenuTitles.MyContact, 4)]
        public async Task FillEmail(Update request) {
            var chatId = request?.Message?.Chat.Id;
            if (chatId == null) {
                return;
            }
            var key = chatId.ToString();
            if (string.IsNullOrEmpty(key)) {
                return;
            }
            var msg = await _bot.SendTextMessageAsync(chatId, "Додаємо ваш
контакт у систему □", replyMarkup: MainMenu.GetKeyboard());
            var text = request?.Message?.Text;
            if (string.IsNullOrEmpty(text)) {
```

```
                return;
            }

            var contact = await GetSavedContactWithEmail(key, text);
            await _contactManager.CreateContactAsync(contact, key);
            await _bot.DeleteMessageAsync(chatId, msg.MessageId);
            await _bot.SendTextMessageAsync(chatId, "Ми успішно додали
ваш контакт у систему!", replyMarkup: MainMenu.GetKeyboard());
            await NextOrFinishStepAsync(chatId);
        }

        #endregion
}
```

Файл RateCurrencyHandler.cs:

```
[CallbackMessageHandler(Callbacks.RateCurrency)]
public class RateCurrencyHandler : ICallbackHandler {
        #region Fields: Private

        private readonly ITelegramBotClient _bot;
        private readonly IRateManager _manager;
        private readonly ICacheManager _cacheManager;

        #endregion

        #region Constructors: Public

        public RateCurrencyHandler(IManagerProvider provider) {
            _bot = provider.GetTelegramBotClient();
            _manager = provider.GetRateManager();
```

```
        _cacheManager = provider.GetCacheManager();
}

#endregion

#region Methods: Private

private string GetRateTextMessage(Core.Models.Rate data) {
        var currencyCode = data.Currency?.Code;
        var currencyDescription = data.Currency?.Description;
        var sellRate = data.SellRate;
        var buyRate = data.BuyRate;
        if (currencyCode == null || currencyDescription == null) {
                return "Щось пішло не так... ☹";
        }
        return $"💸\t💸\t💸\n\n\n" +
                $"Валюта: {currencyCode}\n" +
                $"Опис: {currencyDescription}\n\n" +
                $"▲ Обмінник продає по: {sellRate}₴\n" +
                $"▼ Обмінник купує по: {buyRate}₴";
}

#endregion

#region Methods: Public

public async Task ExecuteAsync(Update request) {
        var chatId = request?.CallbackQuery?.Message?.Chat.Id;
        if (chatId == null) {
                return;
```

```
            }

            var key = chatId.ToString();
            if (string.IsNullOrEmpty(key)) {
                return;
            }

            var callback = request?.CallbackQuery?.Data;
            if (string.IsNullOrEmpty(callback)) {
                return;
            }

            var splittedCallback = callback.Split('|');
            if (splittedCallback.Length != 2) {
                return;
            }

            var currencyId = splittedCallback[1];
            var rate = await _manager.GetRateAsync(key, currencyId);
            var rateTextMsg = GetRateTextMessage(rate);

            var lastMessageId = await _cacheManager.GetLastMessage(key);
            await _bot.DeleteMessageAsync(chatId, messageId:
Convert.ToInt32(lastMessageId));

            await _bot.SendTextMessageAsync(chatId, text: rateTextMsg,
replyMarkup: MainMenu.GetKeyboard());
        }

        #endregion
```

Файл RateMessageHandler.cs:

```
[TextMessageHandler(MainMenuTitles.Rates)]
public class RateMessageHandler : ITextHandler {
	#region Fields: Private

	private ITelegramBotClient _bot;
	private IRateManager _rateManager;
	private ICacheManager _cacheManager;
	private string _cbCurrencyPrefix;

	#endregion

	#region Constructors: Public

	public RateMessageHandler(IManagerProvider provider) {
		_bot = provider.GetTelegramBotClient();
		_rateManager = provider.GetRateManager();
		_cacheManager = provider.GetCacheManager();
		_cbCurrencyPrefix = "RateCurrency";
	}

	#endregion


	#region Methods: Private

	public static InlineKeyboardMarkup GetKeyboard(IDictionary<string,
string> values) {
		var keyboard = new List<List<InlineKeyboardButton>>();
		foreach (var key in values.Keys) {
```

```
                var row = new List<InlineKeyboardButton>();
                var value = values[key];
                var button = InlineKeyboardButton.WithCallbackData(text:
value, callbackData: key);
                row.Add(button);
                keyboard.Add(row);
            }
            return new InlineKeyboardMarkup(keyboard);
        }

        private InlineKeyboardMarkup
GetAcceptedCurrenciesKeyboard(IEnumerable<Core.Models.Rate> rates) {
            var currencies = rates.Select(x => x.Currency).Where(x => x !=
null).Cast<Currency>();
            var dictionarifiedValues = GetDictionarifiedValues(currencies);
            return GetKeyboard(dictionarifiedValues);
        }

        private IDictionary<string, string>
GetDictionarifiedValues(IEnumerable<Currency> currencies) {
            var dictionary = new Dictionary<string, string>();
            foreach (var currency in currencies) {
                var key = _cbCurrencyPrefix + "|" + currency.Id.ToString();
                var value = currency.Code == null ? "none" : currency.Code;
                dictionary.TryAdd(key, value);
            }
            return dictionary;
        }

        #endregion
```

```
        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
                var chatId = request?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var loadingMessage = await _bot.SendStickerAsync(chatId, sticker:
InputFile.FromFileId("CAACAgIAAxkBAAEMKDdmTIUGYME5y_0bBsx_6NR
c8lsoVAACRF4AAi3haEi5O6MC2jMvlTUE"));
                var rates = await _rateManager.GetRatesAsync(key);
                await _bot.DeleteMessageAsync(chatId, loadingMessage.MessageId);

                if (rates.Count() == 0) {
                        await _bot.SendTextMessageAsync(chatId, text: "test",
replyMarkup: MainMenu.GetKeyboard());
                        return;
                }

                var keyboard = GetAcceptedCurrenciesKeyboard(rates);
                var msg = await _bot.SendTextMessageAsync(chatId, text: "test",
replyMarkup: keyboard);
                await _cacheManager.SaveMessageAsync(key, msg.MessageId);
```

```
        }

        #endregion
}
```

Файл ReservationConfirmationHandler.cs:

```
[CallbackMessageHandler(Callbacks.Approve)]
public class ReservationApproveHandler : BaseReservationCreator,
ICallbackHandler {
        #region Fields: Private

        private readonly ITelegramBotClient _bot;
        private readonly ICacheManager _cacheManager;
        private readonly IReservationManager _reservationManager;

        #endregion

        #region Constructors: Public

        public ReservationApproveHandler(IManagerProvider provider) :
base(provider) {
                _bot = provider.GetTelegramBotClient();
                _cacheManager = provider.GetCacheManager();
                _reservationManager = provider.GetReservationManager();
        }

        #endregion

        #region Methods: Public
```

```
        public async Task ExecuteAsync(Update request) {
                var chatId = request?.CallbackQuery?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var callback = request?.CallbackQuery?.Data;
                if (string.IsNullOrEmpty(callback)) {
                        return;
                }

                var lastMessageId = await _cacheManager.GetLastMessage(key);
                await _bot.DeleteMessageAsync(chatId, messageId:
Convert.ToInt32(lastMessageId));

                var tempReservationData = await
GetTempReservationDataAsync(key);
                var data = new InsertReservation() {
                        ContactId = tempReservationData.ContactId,
                        OperationId = tempReservationData.OperationId,
                        RateId = tempReservationData.Rate.Id,
                        StatusId = new Guid("a1b2c3d4-e5f6-7890-abcd-
ef1234567890"), // in progress
                        Amount = tempReservationData.Amount
                };
```

```
            var msg = await _bot.SendStickerAsync(chatId, sticker:
InputFile.FromFileId("CAACAgIAAxkBAAEMOZ1mWahnBXeULfsC9JtS6IlJuz
98KwACaBEAAmZ7kUgsXv7fxzaNIDUE"));
            await _reservationManager.CreateReservationAsync(data);
            await _bot.DeleteMessageAsync(chatId, msg.MessageId);
            await _bot.SendTextMessageAsync(chatId, text: "Бронювання
успішно створене!");
        }

        #endregion
}

[CallbackMessageHandler(Callbacks.Reject)]
public class ReservationRejectHandler : BaseReservationCreator,
ICallbackHandler {
        #region Fields: Private

        private readonly ITelegramBotClient _bot;
        private readonly ICacheManager _cacheManager;

        #endregion

        #region Constructors: Public

        public ReservationRejectHandler(IManagerProvider provider) :
base(provider) {
            _bot = provider.GetTelegramBotClient();
            _cacheManager = provider.GetCacheManager();
        }
```

```
        #endregion

        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
                var chatId = request?.CallbackQuery?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var callback = request?.CallbackQuery?.Data;
                if (string.IsNullOrEmpty(callback)) {
                        return;
                }

                var lastMessageId = await _cacheManager.GetLastMessage(key);
                await _bot.DeleteMessageAsync(chatId, messageId:
Convert.ToInt32(lastMessageId));

                await _bot.SendTextMessageAsync(chatId, text: "Створення
бронювання скасовано!");
                await ClearTempReservationDataAsync(key);
        }
```

```
        #endregion
}
```

Файл ReservationCurrencyHandler.cs:

```
[CallbackMessageHandler(Callbacks.ReservationCurrency)]
public class ReservationCurrencyHandler : BaseReservationCreator,
ICallbackHandler {
        #region Fields: Private

        private readonly ITelegramBotClient _bot;
        private readonly ICacheManager _cacheManager;

        #endregion

        #region Constructors: Public

        public ReservationCurrencyHandler(IManagerProvider provider) :
base(provider) {
                _bot = provider.GetTelegramBotClient();
                _cacheManager = provider.GetCacheManager();
        }

        #endregion

        #region Methods: Private

        private InlineKeyboardMarkup GetOperationTypeKeyboard() {
                var buttons = new List<List<InlineKeyboardButton>>() {
                        new List<InlineKeyboardButton>() {
```

```
                        InlineKeyboardButton.WithCallbackData(text:
"Продати", callbackData:
$"{Callbacks.OperationType}|{Constants.OperationType.Buy}")
                    },new List<InlineKeyboardButton>() {
                        InlineKeyboardButton.WithCallbackData(text:
"Купити", callbackData:
$"{Callbacks.OperationType}|{Constants.OperationType.Sell}")
                    }
            };
            return new InlineKeyboardMarkup(buttons);
        }

        #endregion

        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
            var chatId = request?.CallbackQuery?.Message?.Chat.Id;
            if (chatId == null) {
                return;
            }

            var key = chatId.ToString();
            if (string.IsNullOrEmpty(key)) {
                return;
            }

            var callback = request?.CallbackQuery?.Data;
            if (string.IsNullOrEmpty(callback)) {
                return;
```

```
            }

            var splittedCallback = callback.Split('|');
            if (splittedCallback.Length != 2) {
                return;
            }

            var currencyId = splittedCallback[1];
            await SetCurrencyValue(key, currencyId);

            var lastMessageId = await _cacheManager.GetLastMessage(key);
            await _bot.DeleteMessageAsync(chatId, messageId:
Convert.ToInt32(lastMessageId));

            var message = await _bot.SendTextMessageAsync(chatId, text:
"Оберіть тип операції:", replyMarkup: GetOperationTypeKeyboard());
            await _cacheManager.SaveMessageAsync(key, message.MessageId);
        }

        #endregion
}
```

Файл ReservationOperationHandler.cs:

```
[CallbackMessageHandler(Callbacks.OperationType)]
public class ReservationOperationHandler : BaseReservationCreator,
ICallbackHandler {
        #region Fields: Private

        private readonly ITelegramBotClient _bot;
        private readonly ICacheManager _cacheManager;
```

```
        #endregion

        #region Constructors: Public

        public ReservationOperationHandler(IManagerProvider provider) :
base(provider) {
                _bot = provider.GetTelegramBotClient();
                _cacheManager = provider.GetCacheManager();
        }

        #endregion

        #region Methods: Private

        private async Task<string> GetConfirmationInfoText(string key) {
                var entity = await GetTempReservationDataAsync(key);
                if (entity.Rate == null || entity.Rate.Currency == null) {
                        throw new Exception();
                }
                var textRate = entity.OperationId == new Guid(OperationType.Buy) ?
entity.Rate.BuyRate : entity.Rate.SellRate;
                var textOperationType = entity.OperationId == new
Guid(OperationType.Buy) ? "Покупка" : "Продаж";
                return "!?   !?   !?\n\n" +
                        $"Валюта: {entity.Rate.Currency.Code}\n" +
                        $"Опис: {entity.Rate.Currency.Description}\n\n" +
                        $"Курс: {textRate}\n" +
                        $"Тип операції: {textOperationType}";
        }
```

```
        private InlineKeyboardMarkup GetConfirmationKeyboard() {
                var buttons = new List<List<InlineKeyboardButton>>() {
                        new List<InlineKeyboardButton>() {
                                InlineKeyboardButton.WithCallbackData(text: "👍",
callbackData: Callbacks.Approve),
                                InlineKeyboardButton.WithCallbackData(text: "👎",
callbackData: Callbacks.Reject)
                        }
                };
                return new InlineKeyboardMarkup(buttons);
        }

        #endregion

        #region Methods: Public

        [TextStepperHandler(Callbacks.OperationType, 1)]
        public async Task ExecuteAsync(Update request) {
                var chatId = request?.CallbackQuery?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var callback = request?.CallbackQuery?.Data;
```

```
            if (string.IsNullOrEmpty(callback)) {
                return;
            }

            var splittedCallback = callback.Split('|');
            if (splittedCallback.Length != 2) {
                return;
            }

            var operationId = splittedCallback[1];
            await SetOperationTypeValue(key, operationId);

            var lastMessageId = await _cacheManager.GetLastMessage(key);
            await _bot.DeleteMessageAsync(chatId, messageId:
Convert.ToInt32(lastMessageId));

            var message = await _bot.SendTextMessageAsync(chatId, text:
"Введіть кількість валюти:");
            await _cacheManager.SaveMessageAsync(key, message.MessageId);

            var config = new StepperInfo() {
                CurrentStep = 1,
                StepsCount = 2,
                Name = Callbacks.OperationType,
            };
            await ConfigureStepperAsync(chatId, config);
        }

        [TextStepperHandler(Callbacks.OperationType, 2)]
        public async Task ConfirmationAsync(Update request) {
```

```
            var chatId = request?.Message?.Chat.Id;
            if (chatId == null) {
                return;
            }

            var key = chatId.ToString();
            if (string.IsNullOrEmpty(key)) {
                return;
            }

            var amount = request?.Message?.Text;
            if (string.IsNullOrEmpty(amount)) {
                throw new Exception();
            }
            await SetAmountValue(key, amount);

            var text = await GetConfirmationInfoText(key);
            var message = await _bot.SendTextMessageAsync(chatId, text: text,
replyMarkup: GetConfirmationKeyboard());
            await _cacheManager.SaveMessageAsync(key, message.MessageId);
            await NextOrFinishStepAsync(chatId);
        }

        #endregion
}
```

Файл MyReservationsMessageHandler.cs:

```
[TextMessageHandler(ReservationMenuTitles.MyReservations)]
public class MyReservationsMessageHandler : ITextHandler {
        #region Fields: Private
```

```
        private ITelegramBotClient _bot;
        private IReservationManager _reservationManager;
        private IContactManager _contactManager;

        #endregion

        #region Constructors: Public

        public MyReservationsMessageHandler(IManagerProvider provider) {
                _bot = provider.GetTelegramBotClient();
                _reservationManager = provider.GetReservationManager();
                _contactManager = provider.GetContactManager();
        }

        #endregion

        #region Methods: Private

        private string
GetReservationsText(IEnumerable<Core.Models.Reservation> reservations) {
                var resultText = "";
                foreach(var reservation in reservations) {
                        var textRate = reservation.OperationType.Id == new
Guid(Constants.OperationType.Buy) ? reservation.Rate.BuyRate :
reservation.Rate.SellRate;
                        var textOperationType = reservation.OperationType.Id == new
Guid(Constants.OperationType.Buy) ? "покупку" : "продаж";

                        resultText += "□    □    □\n" +
```

```
                        $"Валюта: {reservation.Rate.Currency.Code}\n" +
                        $"Опис: {reservation.Rate.Currency.Description}\n\n" +
                        $"Бронювання на {textOperationType}.\n" +
                        $"Курс: {textRate}\n" +
                        $"Кількість: {reservation.Amount}\n" +
                        $"Статус бронювання:
{reservation.Status.Name}\n\n\n\n";
                }
                return resultText;
        }

        #endregion

        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
                var chatId = request?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var contact = await _contactManager.GetContactAsync(key);
                if (contact == null) {
```

```
                await _bot.SendTextMessageAsync(chatId, text: "Додайте
контакт у систему, без нього неможливо створити бронювання 😈",
replyMarkup: MainMenu.GetKeyboard());
                return;
            }

            var reservations = await
_reservationManager.GetReservationsAsync(contact.Id.ToString());
            if (reservations.Count() == 0) {
                await _bot.SendTextMessageAsync(chatId, text: "У вас немає
активних бронювань, створіть нове 😈", replyMarkup:
ReservationMenu.GetKeyboard());
                return;
            }

            var msgText = GetReservationsText(reservations);
            await _bot.SendTextMessageAsync(chatId, text: msgText,
replyMarkup: ReservationMenu.GetKeyboard());
        }

        #endregion
}
```

Файл NewReservationMessageHandler.cs:

```
[TextMessageHandler(ReservationMenuTitles.CreateReservation)]
public class NewReservationMessageHandler : BaseReservationCreator,
ITextHandler {
        #region Fields: Private

        private ITelegramBotClient _bot;
```

```
        private IRateManager _rateManager;
        private IContactManager _contactManager;
        private ICacheManager _cacheManager;
        private string _cbPrefixReservationCurrency;

        #endregion

        #region Constructors: Public

        public NewReservationMessageHandler(IManagerProvider provider) :
base(provider) {
                _bot = provider.GetTelegramBotClient();
                _rateManager = provider.GetRateManager();
                _contactManager = provider.GetContactManager();
                _cacheManager = provider.GetCacheManager();
                _cbPrefixReservationCurrency = Callbacks.ReservationCurrency;
        }

        #endregion

        #region Methods: Private

        public static InlineKeyboardMarkup GetKeyboard(IDictionary<string,
string> values) {
                var keyboard = new List<List<InlineKeyboardButton>>();
                foreach (var key in values.Keys) {
                        var row = new List<InlineKeyboardButton>();
                        var value = values[key];
                        var button = InlineKeyboardButton.WithCallbackData(text:
value, callbackData: key);
```

```
                row.Add(button);
                keyboard.Add(row);
            }
            return new InlineKeyboardMarkup(keyboard);
        }

        private InlineKeyboardMarkup
GetAcceptedCurrenciesKeyboard(IEnumerable<Core.Models.Rate> rates) {
            var currencies = rates.Select(x => x.Currency).Where(x => x !=
null).Cast<Currency>();
            var dictionarifiedValues = GetDictionarifiedValues(currencies);
            return GetKeyboard(dictionarifiedValues);
        }

        private IDictionary<string, string>
GetDictionarifiedValues(IEnumerable<Currency> currencies) {
            var dictionary = new Dictionary<string, string>();
            foreach (var currency in currencies) {
                var key = _cbPrefixReservationCurrency + "|" +
currency.Id.ToString();
                var value = currency.Code == null ? "none" : currency.Code;
                dictionary.TryAdd(key, value);
            }
            return dictionary;
        }

        #endregion

        #region Methods: Public
```

```
        public async Task ExecuteAsync(Update request) {
                var chatId = request?.Message?.Chat.Id;
                if (chatId == null) {
                        return;
                }

                var key = chatId.ToString();
                if (string.IsNullOrEmpty(key)) {
                        return;
                }

                var contact = await _contactManager.GetContactAsync(key);
                if (contact == null) {
                        await _bot.SendTextMessageAsync(chatId, text: "Додайте
контакт у систему, без нього неможливо створити бронювання 😇",
replyMarkup: MainMenu.GetKeyboard());
                        return;
                }
                await SetContactIdValue(key, contact.Id.ToString());

                var rates = await _rateManager.GetRatesAsync(key);
                if (rates.Count() == 0) {
                        await _bot.SendTextMessageAsync(chatId, text: "Напишить
пізніше, обмінник нічого не може вам запропонувати 😇", replyMarkup:
ReservationMenu.GetKeyboard());
                        return;
                }

                var keyboard = GetAcceptedCurrenciesKeyboard(rates);
```

```
            var msg = await _bot.SendTextMessageAsync(chatId, text: "Оберіть валюту з запропонованих:", replyMarkup: keyboard);
            await _cacheManager.SaveMessageAsync(key, msg.MessageId);
        }

        #endregion
}
```

Файл ReservationsMessageHandler.cs:

```
[TextMessageHandler(MainMenuTitles.Reservations)]
public class ReservationsMessageHandler : ITextHandler {
        #region Fields: Private

        private ITelegramBotClient _bot;

        #endregion

        #region Constructors: Public

        public ReservationsMessageHandler(IManagerProvider provider) {
            _bot = provider.GetTelegramBotClient();
        }

        #endregion

        #region Methods: Public

        public async Task ExecuteAsync(Update request) {
            var chatId = request?.Message?.Chat.Id;
            if (chatId == null) {
```

```
                return;
            }

            await _bot.SendTextMessageAsync(chatId, text: "Бронювання:",
replyMarkup: ReservationMenu.GetKeyboard());
        }

        #endregion
}
```

Файл HandlerMiddleware.cs:

```
public class HandlerMiddleware {
        #region Fields: Private

        private readonly RequestDelegate _next;
        private readonly Dictionary<string, ITextHandler> _handlers;
        private readonly Dictionary<string, ICallbackHandler> _callbackHandlers;

        #endregion

        #region Constructors: Public

        public HandlerMiddleware(RequestDelegate next,
IEnumerable<ITextHandler> handlers, IEnumerable<ICallbackHandler>
cbHandlers) {
            _next = next;
            _handlers = handlers
                .Select(h => new { Handler = h, Attribute =
h.GetType().GetCustomAttribute<TextMessageHandlerAttribute>() })
                .Where(x => x.Attribute != null)
```

```
                .ToDictionary(x => x.Attribute!.Text, x => x.Handler);
            _callbackHandlers = cbHandlers
                .Select(h => new { Handler = h, Attribute =
h.GetType().GetCustomAttribute<CallbackMessageHandlerAttribute>() })
            .Where(x => x.Attribute != null)
                .ToDictionary(x => x.Attribute!.Text, x => x.Handler);
        }

        #endregion

        #region Methods: Private

        private async Task<Update?> GetUpdateFromRequest(Stream requestBody)
{
            var body = await GetRequestBody(requestBody);
            return JsonConvert.DeserializeObject<Update>(body);
        }

        private async Task<string> GetRequestBody(Stream body) {
            var data = await new StreamReader(body, leaveOpen:
true).ReadToEndAsync();
            if (body.CanSeek)
                body.Position = 0;
            return data;
        }

        #endregion

        #region Methods: Public
```

```
        public async Task InvokeAsync(HttpContext context) {
                context.Request.EnableBuffering();
                var update = await GetUpdateFromRequest(context.Request.Body);

                if (update == null) {
                        return;
                }

                var text = update?.Message?.Text;
                if (!string.IsNullOrEmpty(text) && _handlers.TryGetValue(text, out
var handler)) {
                        await handler.ExecuteAsync(update!);
                        context.Request.Headers["handler-executed"] = "1";
                }

                var callback = update?.CallbackQuery?.Data;
                if (!string.IsNullOrEmpty(callback)) {
                        var searcher = callback.Contains('|') ? callback.Substring(0,
callback.IndexOf('|')) : callback;
                        if (!string.IsNullOrEmpty(callback) &&
_callbackHandlers.TryGetValue(searcher, out var callbackHandler)) {
                                await callbackHandler.ExecuteAsync(update!);
                        }
                }

                await _next(context);
        }

        #endregion
}
```

Файл StepperHandlerMiddleware.cs:

```
public class StepperHandlerMiddleware {
	#region Fields: Private

	private readonly RequestDelegate _next;
	private readonly ICacheClient _cache;
	private readonly IManagerProvider _managerProvider;

	#endregion

	#region Constructors: Public

	public StepperHandlerMiddleware(RequestDelegate next, ICacheClient
cacheClient, IManagerProvider managerProvider) {
		_next = next;
		_cache = cacheClient;
		_managerProvider = managerProvider;
	}

	#endregion

	#region Methods: Private

	private async Task ExecuteTextHandlerNextStep(Update request,
StepperInfo info) {
		var stepperKey = GetUserStepper(request);
		if (string.IsNullOrEmpty(stepperKey))
			return;

		var stepperName = info.Name;
```

```
            if(string.IsNullOrEmpty(stepperName)) {
                return;
            }
            var types = GetTypesWithTextHandlerAttribute(stepperName);

            foreach (var type in types) {
                MethodInfo[] methods = type.GetMethods(BindingFlags.Public
| BindingFlags.Instance);
                foreach (var method in methods) {
                    var attributes =
method.GetCustomAttributes(typeof(TextStepperHandlerAttribute), false);
                    foreach (var attribute in attributes) {
                        var typedAttribute =
(TextStepperHandlerAttribute)attribute;
                        if (typedAttribute.Name == info.Name &&
typedAttribute.Step == info.CurrentStep) {
                            var instance =
Activator.CreateInstance(type, _managerProvider);
                            await
Task.FromResult(method.Invoke(instance, new[] { request }));
                        }
                    }
                }
            }
        }

        private IEnumerable<Type> GetTypesWithTextHandlerAttribute(string
stepperName) {
            Assembly assembly = Assembly.GetExecutingAssembly();
            var types = assembly.GetTypes();
```

```
            return types.Where(t =>
                t.GetCustomAttributes(false).Any(attr =>
                    (attr is TextMessageHandlerAttribute textAttr &&
textAttr.Text == stepperName) ||
                    (attr is CallbackMessageHandlerAttribute callbackAttr
&& callbackAttr.Text == stepperName)));
        }

        private string? GetUserStepper(Update request) {
            return request?.Message?.Chat?.Id.ToString();
        }

        private async Task<Update?> GetUpdateFromRequest(Stream requestBody)
{
            var body = await GetRequestBody(requestBody);
            return JsonConvert.DeserializeObject<Update>(body);
        }

        private async Task<string> GetRequestBody(Stream body) {
            var data = await new StreamReader(body, leaveOpen:
true).ReadToEndAsync();
            if (body.CanSeek)
                body.Position = 0;
            return data;
        }

        #endregion

        #region Methods: Public
```

```
        public async Task InvokeAsync(HttpContext context) {
                var handlerExecutedHeader =
context.Request.Headers.TryGetValue("handler-executed", out var
isHandlerExecuted);
                if (!string.IsNullOrEmpty(isHandlerExecuted)) {
                        await _next(context);
                }

                context.Request.EnableBuffering();
                var update = await GetUpdateFromRequest(context.Request.Body);
                if (update == null) {
                        return;
                }

                string? userStepper = GetUserStepper(update);
                if (string.IsNullOrEmpty(userStepper)) {
                        return;
                }

                var stepInfoJson = await _cache.GetAsync(userStepper);
                if (!string.IsNullOrEmpty(stepInfoJson)) {
                        var stepInfo =
JsonConvert.DeserializeObject<StepperInfo>(stepInfoJson);
                        if (stepInfo == null) {
                                return;
                        }

                        await ExecuteTextHandlerNextStep(update, stepInfo);
                }
```

```
            await _next(context);
        }

        #endregion
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Вебзастосунок для підтримки діяльності пункту обміну валют

Програма та методика тестування

КПІ.ІТ-0406.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Едуард ЖАРІКОВ

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Микола КЛЬОВА

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ 3

2 МЕТА ТЕСТУВАННЯ 4

3 МЕТОДИ ТЕСТУВАННЯ 5

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ 6

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є додаток для підтримки дільності пункту обміну валют, створеного за допомогою фреймворків Vue.js, Vuetify, ASP. NET Core з використанням середовищ програмування Microsoft Visual Studio, JetBrains WebStorm та мов програмування JavaScript та C#.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка створення бронювань у telegram-боті користувачем;
- перевірка створення бронювань у системі адміністратором;
- перевірка оновлення статусу бронювань у системі адміністратором;
- перевірка відображення бронювань у системі для адміністратору;
- перевірка відображення створених бронювань у telegram-боті для клієнту;
- перевірка створення транзакцій адміністратором у системі;
- перевірка відображення створених бронювань у системі адміністратору;
- перевірка створення транзакцій по певному бронюванню адміністратором у системі;
- перевірка створення валют у системі адміністратором;
- перевірка оновлення валюти адміністратором у системі;
- перевірка видалення валюти адміністратором з системи;
- перевірка відображення валют у системі для адміністратору;
- перевірка відображення курсів валют для адміністратора у системі;
- перевірка відображення курсів валют у telegram-боті для клієнта;
- перевірка відображення резервів валют для адміністратора у системі;
- перевірка оновлення резерву для валюти адміністратором у системі;
- перевірка створення контакту у системі адміністратором;
- перевірка створення контакту у telegram-боті клієнтом;
- перевірка створення нового користувача системи адміністратором;
- перевірка оновлення ролі для певного користувача у системі;
- перевірка правильності розрахунків у дашбордах для адміністраторів системи;
- Ззнаходження проблем, помилок і недоліків з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування, перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки», об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;
- тестування «білої скриньки», об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;
- тестування «сірої скриньки», об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування інтерфейсу користувача;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ПІТРИМКИ ДІЯЛЬНОСТІ ПУНКТУ ОБМІНУ ВАЛЮТ

Керівництво користувача

КПІ.ІТ-0406.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Едуард ЖАРІКОВ

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Микола КЛЬОВА

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ 3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ 4
2.1 Системні вимоги для коректної роботи 4
2.2 Завантаження застосунку 5
2.3 Перевірка коректної роботи 6
3 ВИКОНАННЯ ПРОГРАМИ 7
3.1 Взаємодія з системою адміністраторам 7
3.2 Взаємодія з telegram-ботом клієнтам 30

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

“Currexchange” – система, яка призначена для підтримки діяльності пункту обміну валют. Ця програма надає зручний інструмент, для для відслідковування операцій, які проводить обмінний пункт кожен день. Проведення транзакцій та збереження їх у історії, створення бронювань, відслідковування та зміна їх статусів, можливість бачити з будь-якого куточка світу кількість валюти у резервах та курси валют, які отримуються з національного банку України і вже з розраховані з певним відсотком для дохідності обміннику. Можливість переглянути дохід пункту обміну за певні проміжки часу та кількість проведених транзакцій або опрацьованих бронювань адміністраторами. Система надає зручний інтерфейс та набір функцій для зручного використання та аналізу фінансової та бізнес інформації.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- Windows Server 2016 або новіше, або Linux (наприклад, Ubuntu 18.04 або новіше);
- .NET Core SDK 3.1 або новіше (рекомендовано .NET 5 або новіше);
- Node.js 12.x або новіше;
- PostgreSQL 10 або новіше;
- Redis 5.x або новіше;
- сучасні браузери (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari);
- стабільне інтернет-з'єднання для взаємодії між компонентами системи та телеграм-ботом;
- SSL/TLS сертифікати для захисту HTTP-запитів (HTTPS).
- налаштовані фаєрволи та правила доступу до баз даних і Redis;
- загалом мінімально необхідна оперативна пам'ять: 4 ГБ (сервер) + 4 ГБ (PostgreSQL) + 4 ГБ (Redis) + 2 ГБ (телеграм-бот) = 14 ГБ;
- серверна частина (ASP.NET Core): мінімум 10 ГБ (для ОС, .NET SDK та інших залежностей);
- клієнтська частина (Vue.js): мінімум 1 ГБ (для Node.js, npm-пакетів та інших залежностей);
- PostgreSQL: мінімум 20 ГБ (для даних, залежно від обсягу бази даних);
- Redis: мінімум 5 ГБ (для кешу та інших залежностей);
- телеграм-бот (ASP.NET Core): мінімум 5 ГБ (для ОС, .NET SDK та інших залежностей);
- загалом мінімально необхідний дисковий простір: 10 ГБ (сервер) + 1 ГБ (клієнт) + 20 ГБ (PostgreSQL) + 5 ГБ (Redis) + 5 ГБ (телеграм-бот) = 41 ГБ;

2.2 Завантаження застосунку

Треба з репозиторію склонувати проект back-end, front-end, telegram bot. Завантажити базу даних postgresql та redis. У проекті back-end запустити створення міграції, командою на рисунку 2.1.

```
PS D:\Study\Diploma\ExchangeOffice> dotnet ef migrations add InitialMigration -s ExchangeOffice.API -p ExchangeOffice.DataAccess
```

Рисунок 2.1 – Створення міграції для серверної частини застосунку

І далі, потрібно створити базу даних, командою на рисунку 2.2.

```
PS D:\Study\Diploma\ExchangeOffice> dotnet ef database update -s ExchangeOffice.API -p ExchangeOffice.DataAccess
```

Рисунок 2.2 – Створення бази даних

Після чого, можна запускати серверну частину застосунку, а потім клієнтську. Для встановлення telegram боту, потрібно у платформі telegram, перейти до bot father і виконати дії, які зображені на рисунку 2.3.

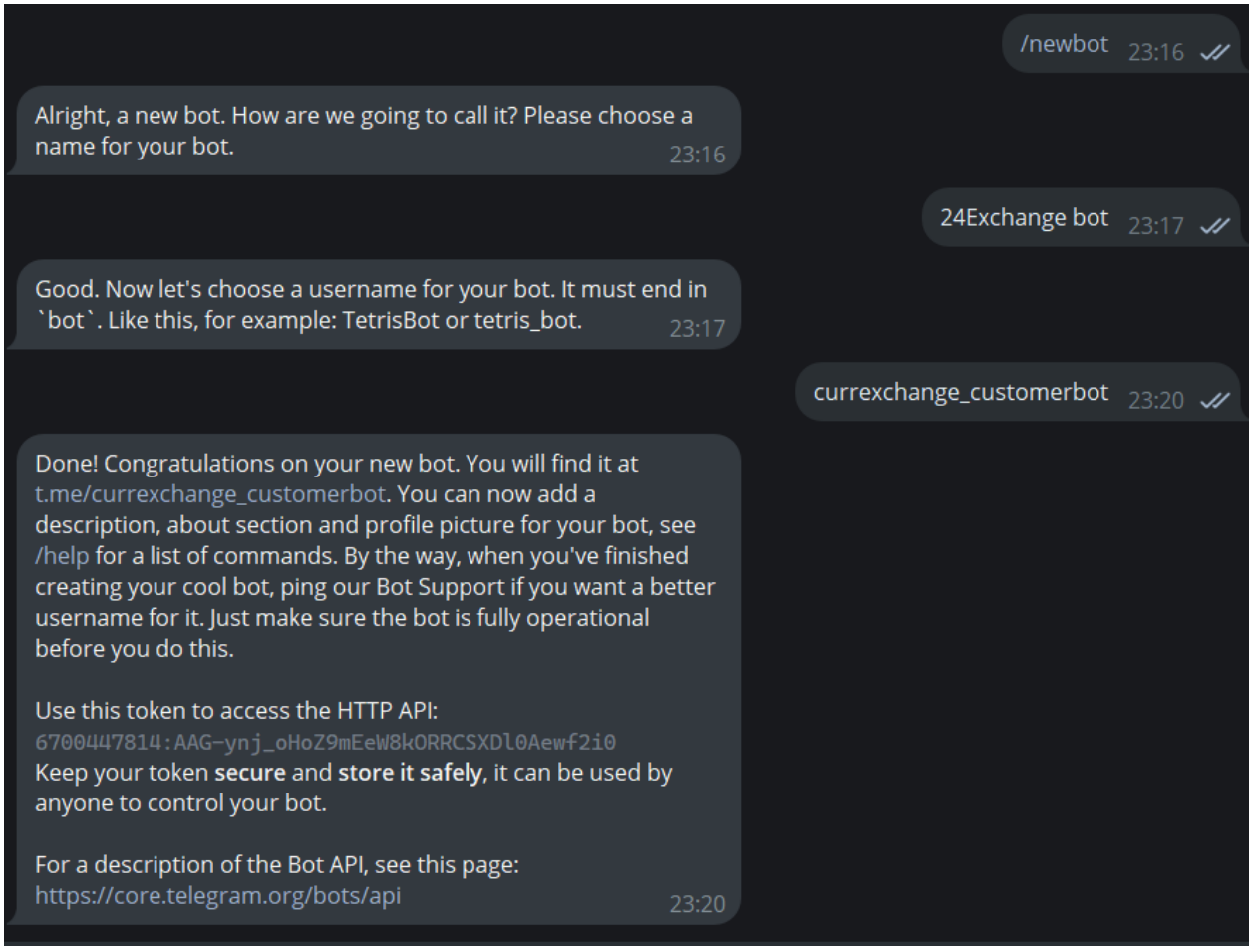


Рисунок 2.3 – Створення нового боту на платформі Telegram

Після чого, отриманий токен, вставити у застосунок telegram bot та запустити його.

2.3 Перевірка коректної роботи

Після розгортання системи, переходимо по посиланню на сайт, там наявний доступ до системи для адміністратора. Клієнту потрібно використовувати тег, який надіслав bot father.

3 ВИКОНАННЯ ПРОГРАМИ

3.1 Взаємодія з системою адміністраторам

У системі наявні 3 ролі – власник, менеджер та касир. Кожна роль має спільні можливості, а деякі, більш високий доступ до системи. Розглянемо на ролі власника, інструкцію взаємодії з системою, а для менеджера та касира, вона буде спільна, але без деякий функцій.

Власник переходить по посиланню, яке він вказав під час розгортання застосунку, рисунок 3.1.

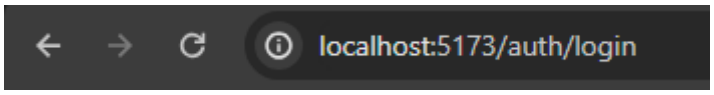


Рисунок 3.1 – Посилання на систему

Після того, як користувач перейшов по посиланню, йому буде відображене вікно для авторизації у системі, рисунок 3.2.

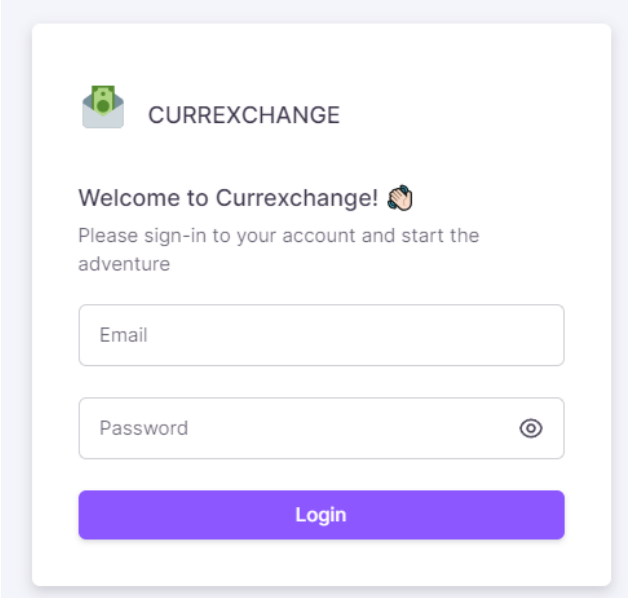


Рисунок 3.2 – Вікно авторизації у системі

Якщо користувач ввів не правильний пароль, то система повідомить йому про це, рисунок 3.3.

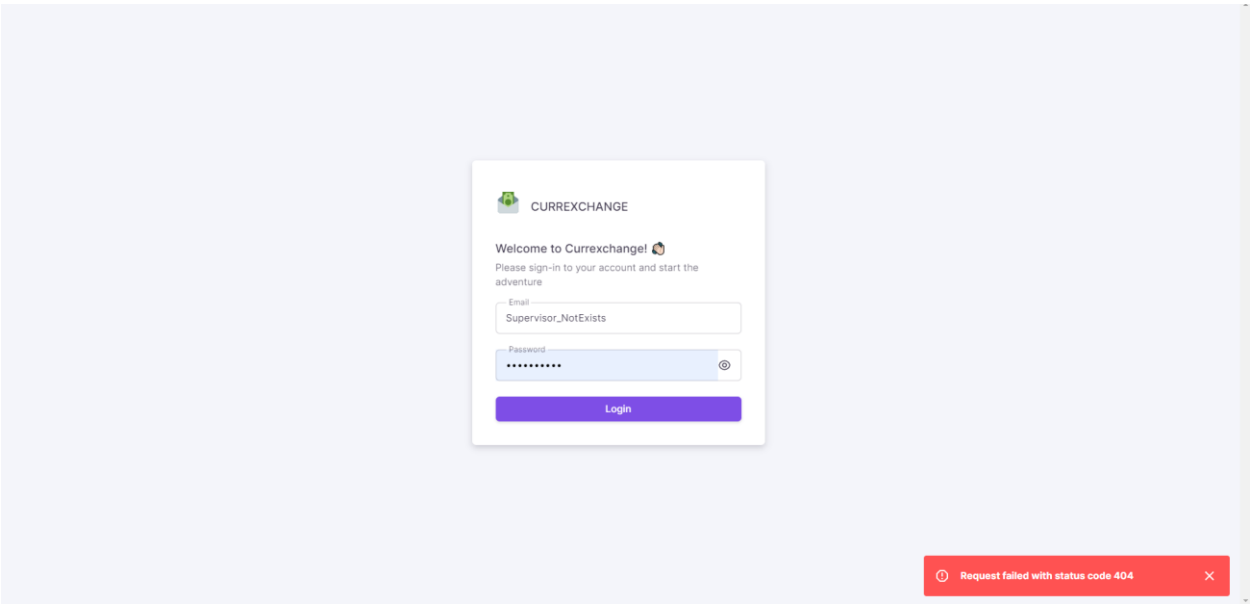


Рисунок 3.3 – Вікно авторизації після не вірного логіну або паролю

Якщо все добре, то відкривається головне вікно, де власнику відображені дашборди з головною інформацією про пункт обміну, рисунок 3.4.

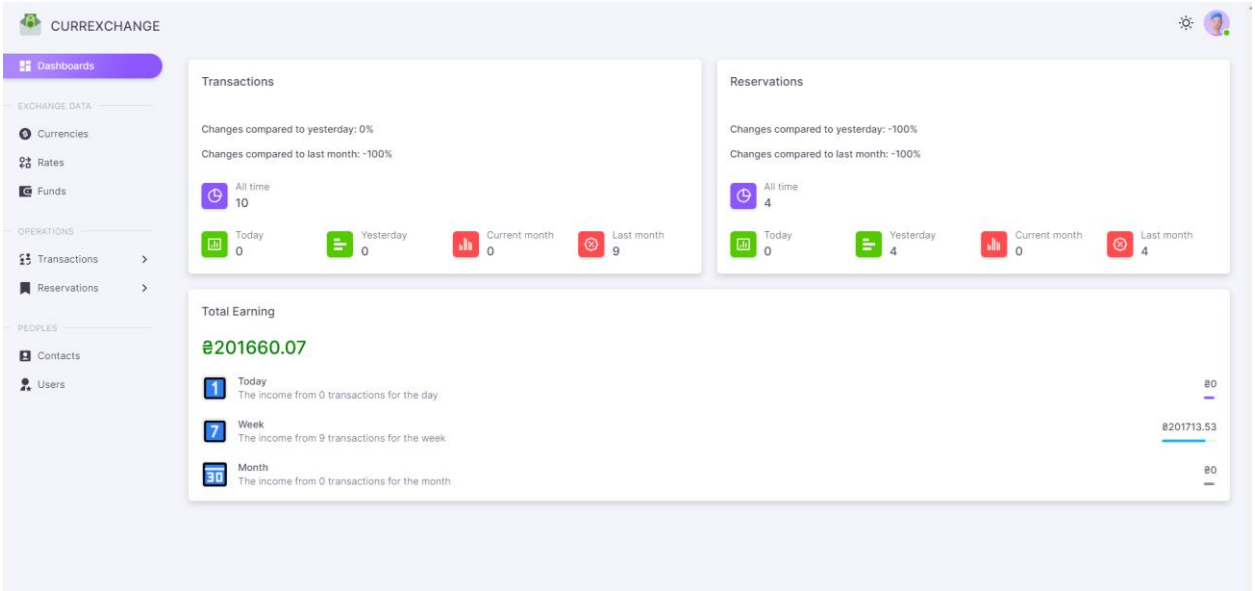


Рисунок 3.4 – Вікно дашбордів після успішної авторизації у системі

Адміністратор може впевнитись у тому, що він під своїм аккаунтом, натиснувши на іконку користувача, рисунок 3.5.

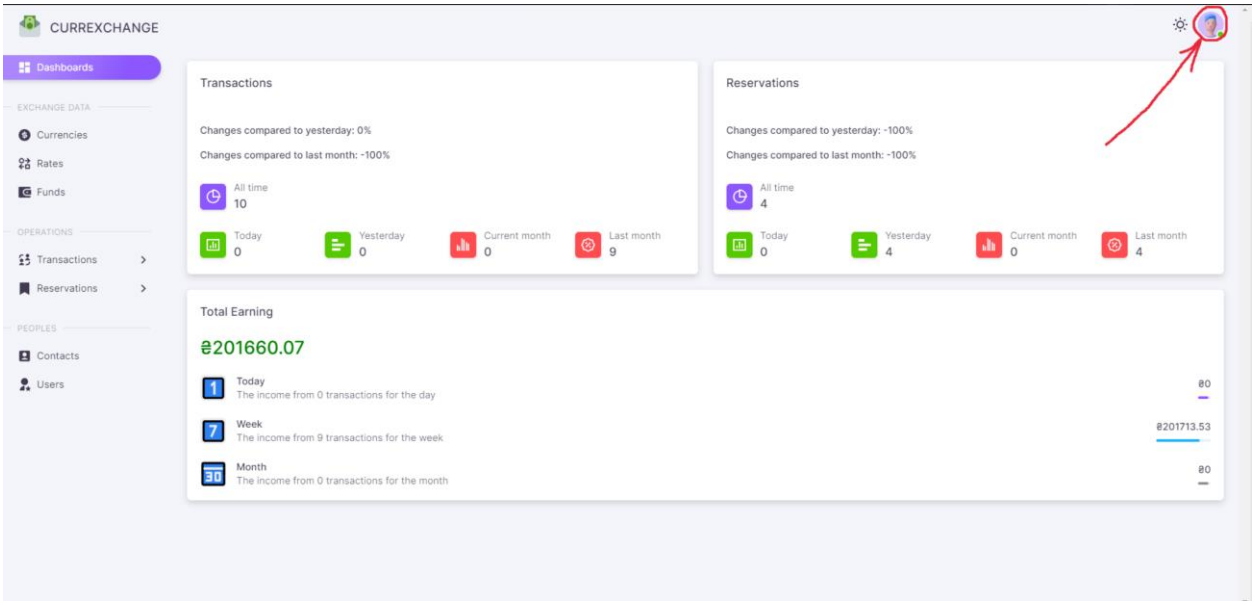


Рисунок 3.5 – Місце, куди потрібно натиснути для відображення інформації про аккаунт

Після натискання, буде відображений логін користувача та його роль у системі, рисунок 3.6.

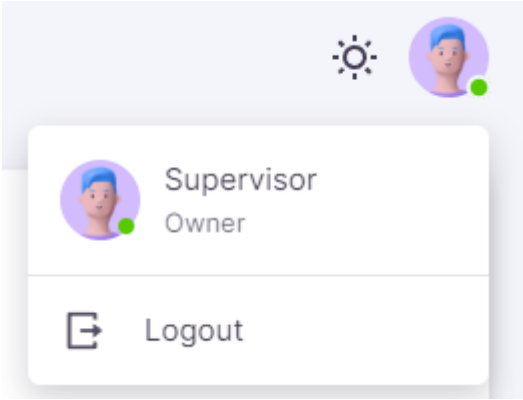


Рисунок 3.6 – Інформація про аккаунт користувача

Для взаємодії з системою, далі користувач повинен оперувати кнопками з меню у лівому краю екрану, рисунок 3.7.

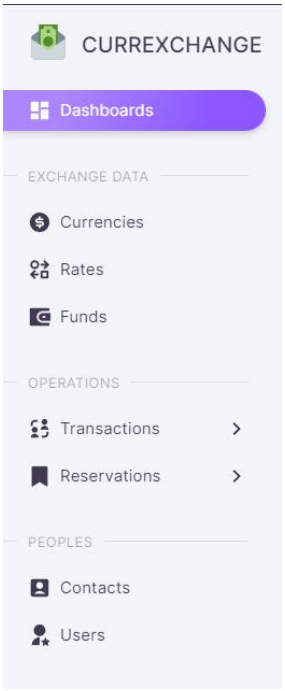


Рисунок 3.7 – Меню системи

Почнемо з пункту “Currencies”. На сторінці є можливість переглянути створені валюти у системі, рисунок 3.8.

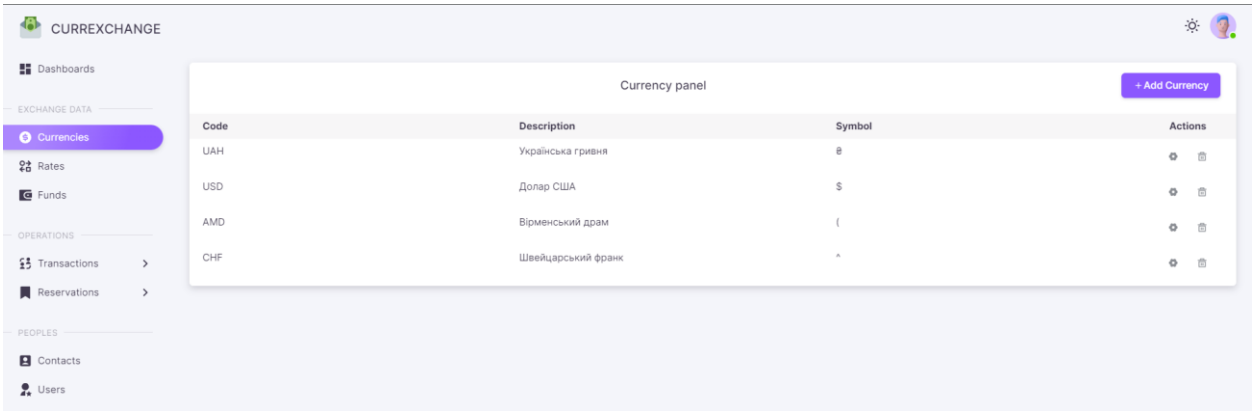


Рисунок 3.8 – Вікно для перегляду створених валют у системі

Для створення нової валюти у системі, потрібно натиснути на кнопку: “Add currency”, рисунок 3.9.

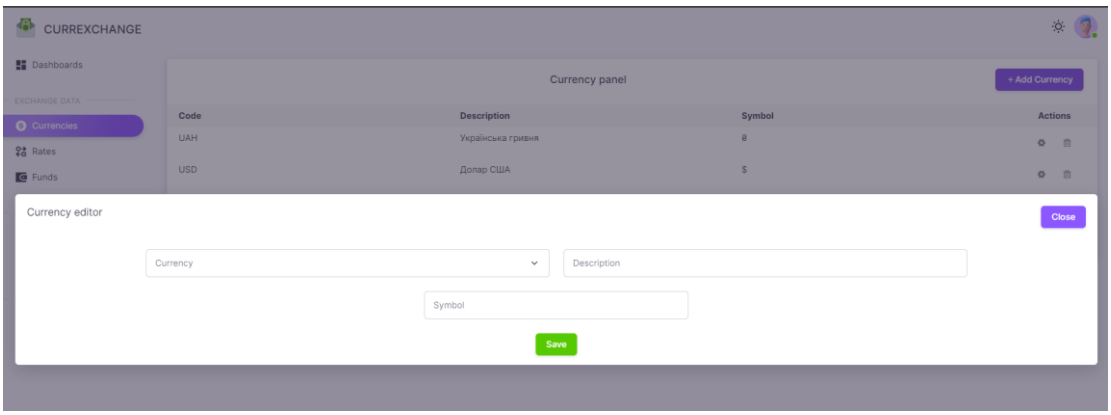


Рисунок 3.9 – Вікно створення нової валюти

На цьому пункті можна закрити це вікно та повернутись до рисунку 3.9. А також зберегти нову валюту. При спробі створити валюту з пустими полями, система підкаже, які потрібно заповнити, рисунок 3.10.

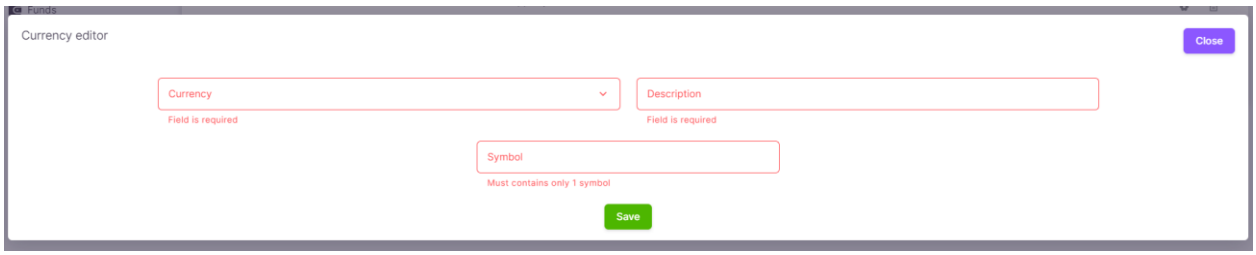


Рисунок 3.10 – Спроба збереження валюти з пустими полями

Валюту можна створити лише ту, яка ухвалена Національним банком України, тому після натискання на поле “Currency” – буде наданий список усіх валют, які представлені Національним банком України, рисунок 3.11.

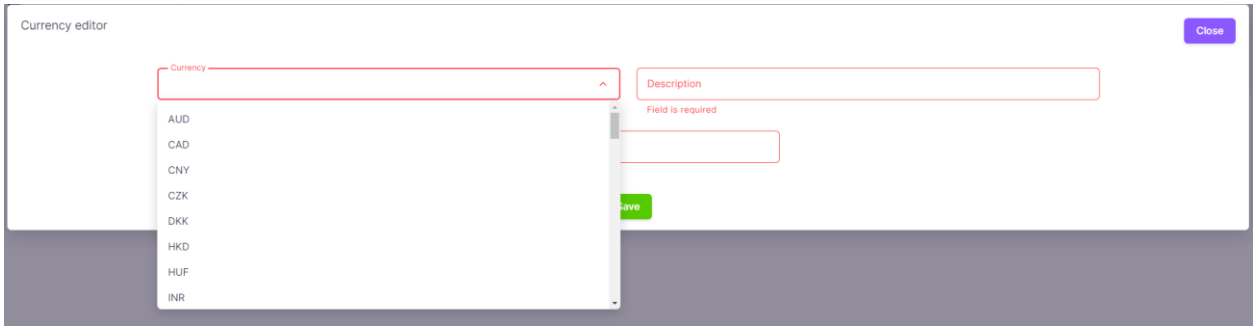


Рисунок 3.11 – Перелік валют ухвалений Національним банком України

Обравши необхідну валюту для створення, поле “Description” – буде заповнене описом валюти з переліку Національного банку України, це значення можна відредагувати на своє, рисунок 3.12.

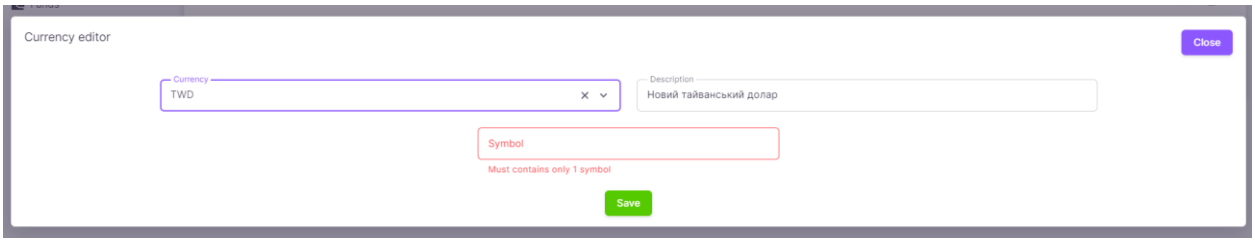


Рисунок 3.12 – Приклад автозаповнення поля з описом валюти

Заповнивши всі обов’язкові поля, треба натиснути на кнопку “Save” і валюта буде успішно додана у систему, про що буде свідчити інформація у правому нижньому кутку, рисунок 3.13.

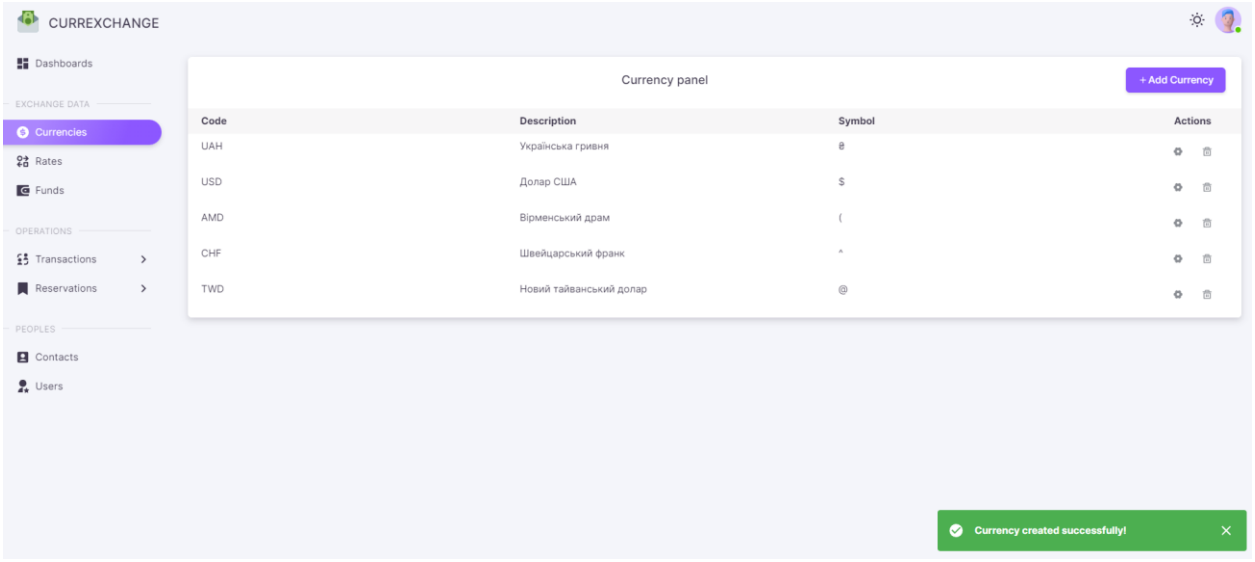


Рисунок 3.13 – Інформація про успішне додавання валюти у систему

При певній помилці, деякі дані про валюту можна відредагувати, натиснувши на шестерню біля необхідної валюти, рисунок 3.14.

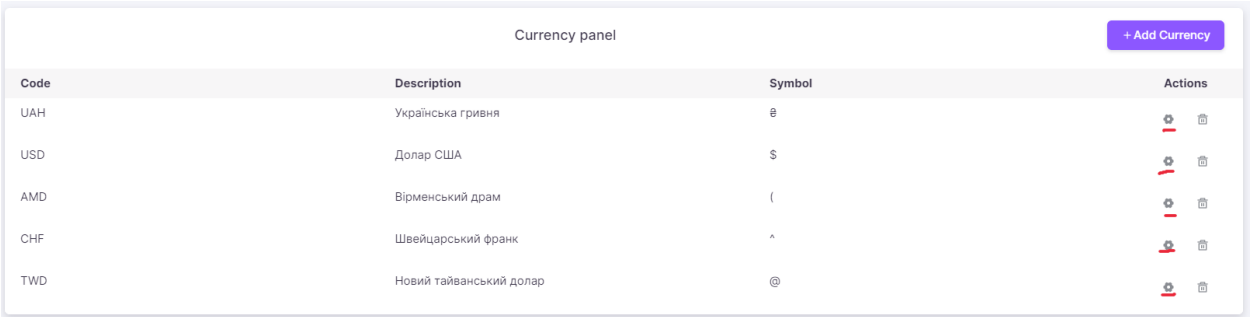


Рисунок 3.14 – Місце, куди потрібно натиснути для модифікації валют

Після кліку, буде відкрито відкно для редагування валюти з заповненими полями, рисунок 3.15.

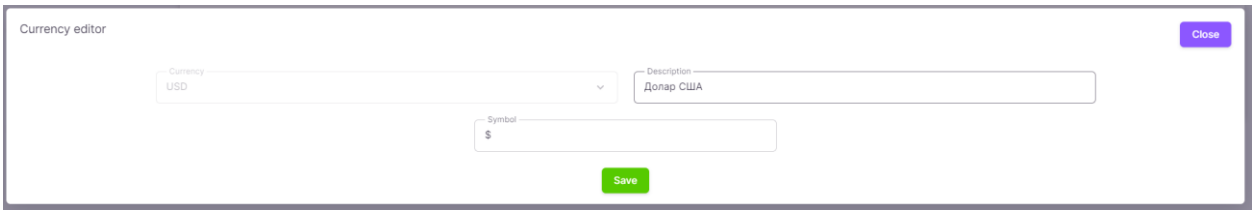


Рисунок 3.15 – Вікно редагування валюти

Після того, як змінено необхідні поля, натискаємо на кнопку “Save” і валюта буде оновлена у загальному списку з інформацією про оновлененя у правому нижньому кутку, рисунок 3.16.

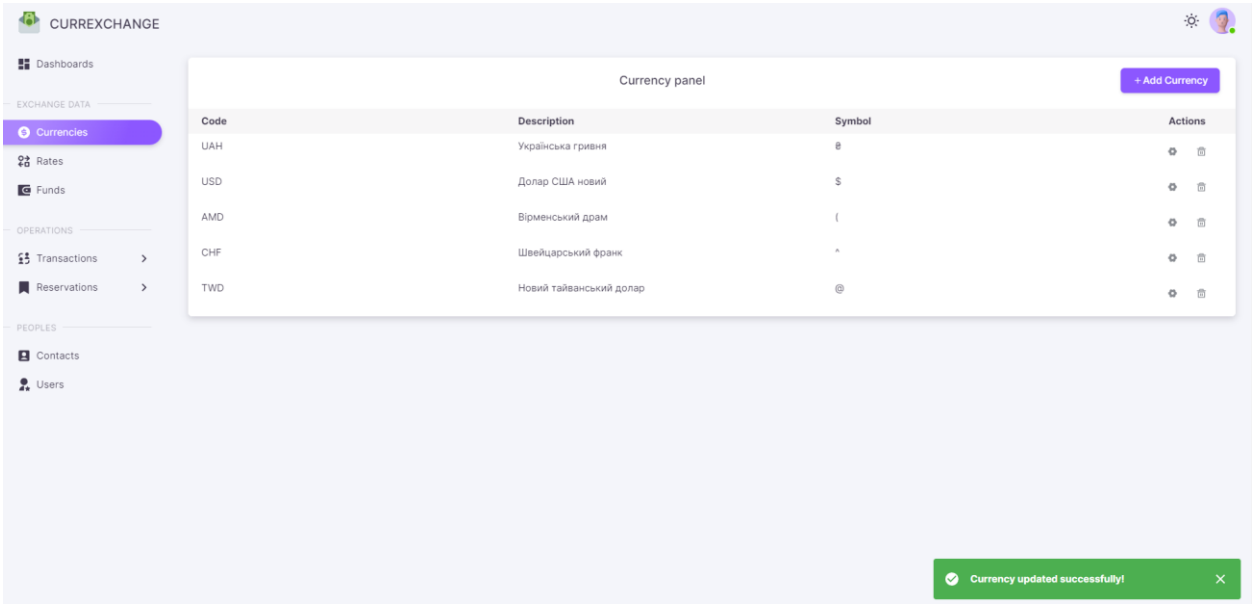


Рисунок 3.16 – Успішне оновлення валюти

Якщо обмінник більше не збирається працювати з певною валютою з переліку, потрібно натиснути на кошик біля відповідної валюти, рисунок 3.17.

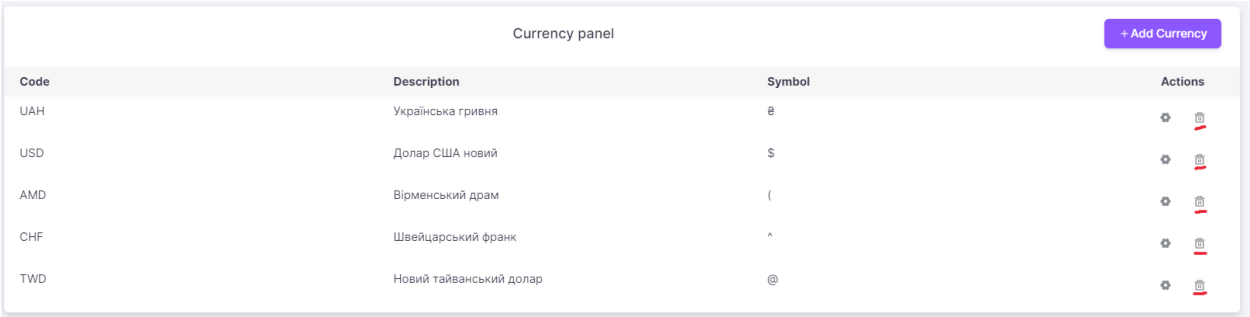


Рисунок 3.17 – Місце куди треба натискати для видалення валюти

Після чого, валюта буде видалена зі списку і відповідна інформація з’явиться у правому нижньому кутку, рисунок 3.18.

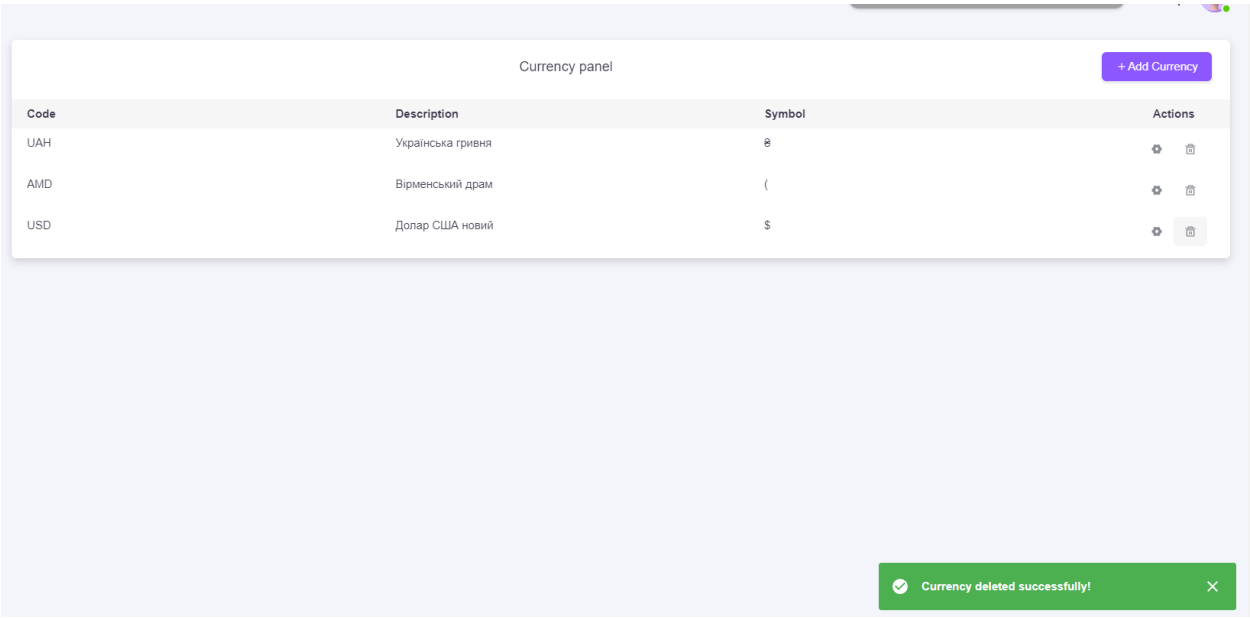


Рисунок 3.18 – Успішне видалення валюти

Після того, як валюта створилась або видалилась, у відповідному вікні, можна переглянути, курс валюти з певною націнкою або його відсутність у разі видалення, натиснувши на пункт меню “Rates”, рисунок 3.19.

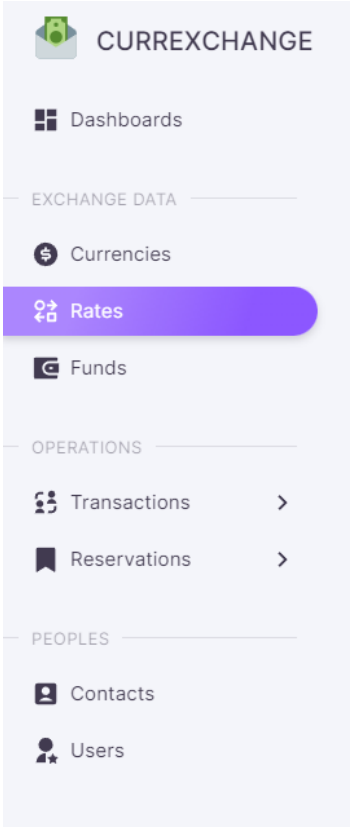


Рисунок 3.19 – Місце, куди потрібно натиснути для перегляду курсів валют

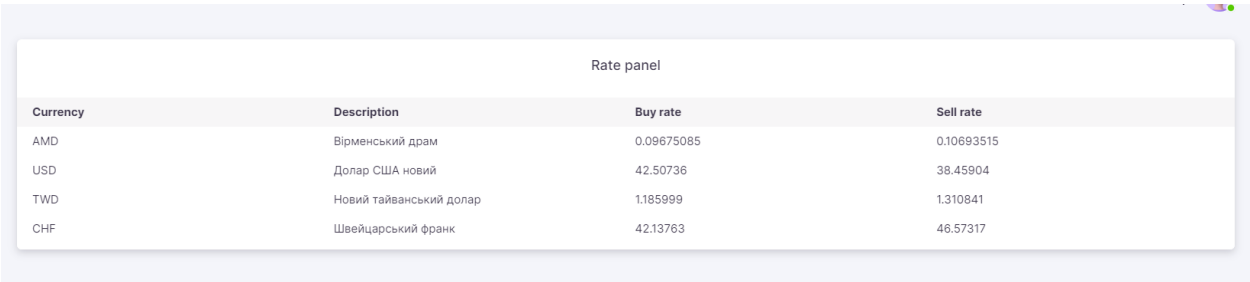


Рисунок 3.20 – Курси валют наявні у системі

Для управління резервами, є спеціальний пункт у меню “Funds”. Там зображена інформація щодо резервів певних валют. Його можна оновлювати, так як він з’являється після додавання валюти у систему зі значенням 0. Після видалення валюти, він так само видаляється, як і курс валюти.

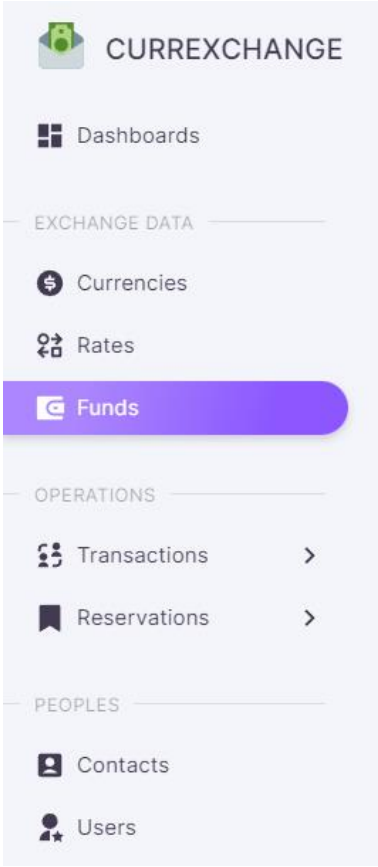


Рисунок 3.21 – Місце знаходження пункту резерву

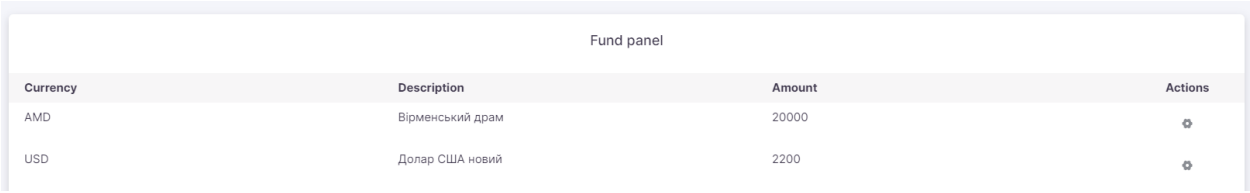


Рисунок 3.22 – Перелік резервів відповідно до валют у системі

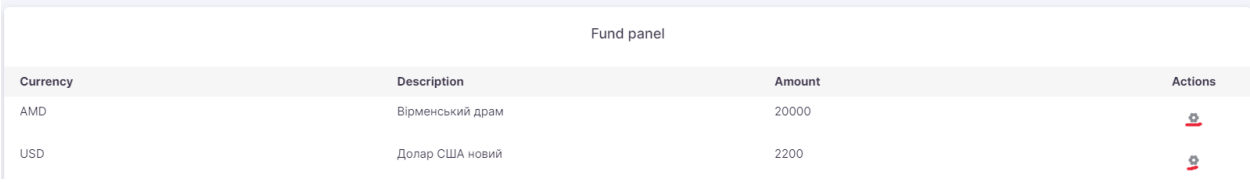


Рисунок 3.23 – Місце, куди потрібно натискати для оновлення резерву

Якщо користувачу потрібно актуалізувати резерв, він натискає на шестерню і йому відкривається вікно для оновлення, рисунок 3.24.

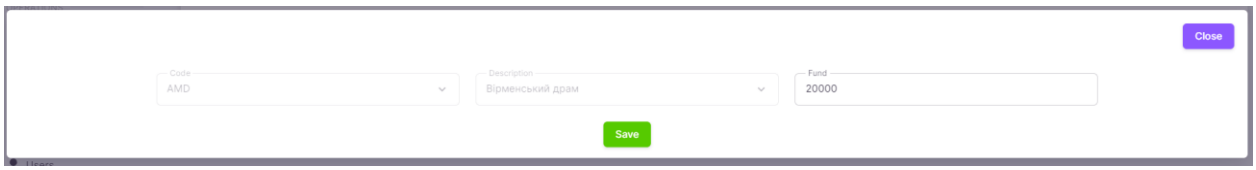


Рисунок 3.24 – Місце для редагування резерву

Після оновлення резерву, потрібно натиснути на кнопку “Save” і відповідний резерв буде оновлено та буде зображена інформація про успішність цього кроку, рисунок 3.25.

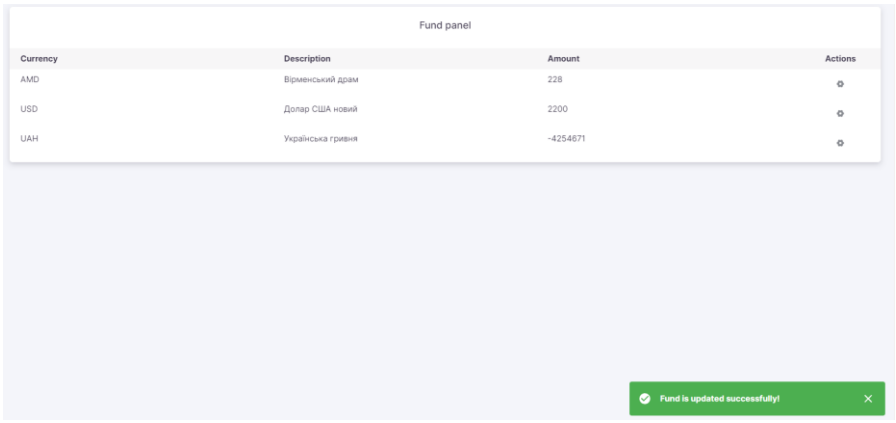


Рисунок 3.25 – Успішне оновлення резерву валюти

У системі є контакти – це люди, які роблять бронювання або постійні клієнти, яких можна створити або переглянути. Для цього є відповідний пункт у меню “Contacts”, рисунок 3.26.

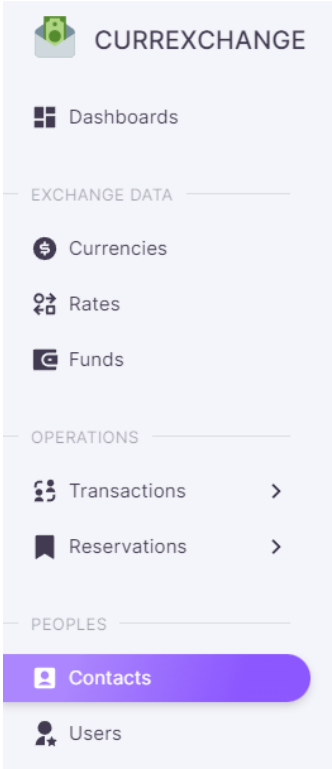


Рисунок 3.26 – Пункт контактів у меню

Після того, як було перейдено у цей розділ, користувачу буде зображено всі контакти пункту обміну валют, рисунок 3.27.

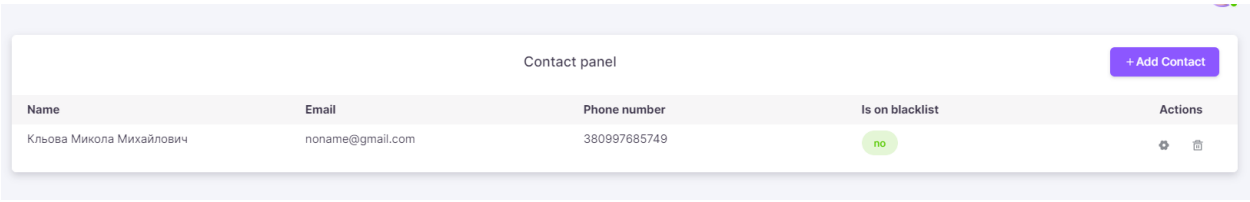


Рисунок 3.27 – Контакти системи

Для створення контакту, є відповідна кнопка “Add contact”, після натискання на яку, буде відкрите вікно з полями, які необхідно заповнити для створення, рисунок 3.28.

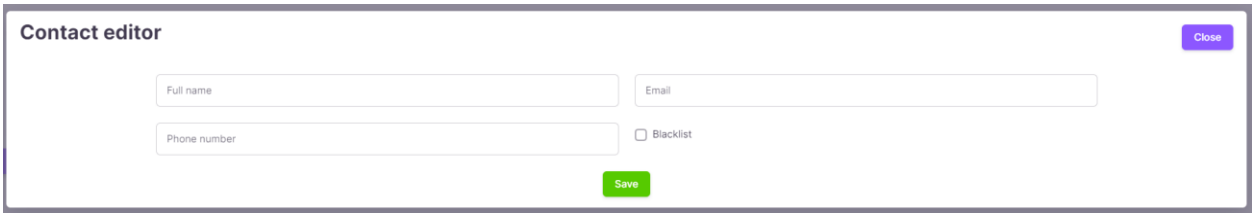


Рисунок 3.28 – Вікно для створення нового контакту

Є певні обов’язкові поля для заповнення, якщо їх пропустити, система після натискання на кнопку “Save” повідомить користувача про це, рисунок 3.29.

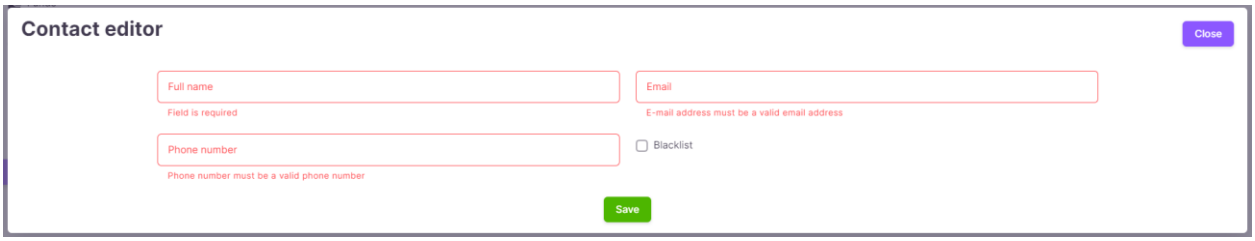


Рисунок 3.29 – Інформація про обов’язкові поля для заповнення

Після заповнення всіх обов’язкових полів і натискання на кнопку “Save”, контакт створиться у системі, буде доданий до загального списку контактів і користувач буде про це проінформований у правому нижньому кутку, рисунок 3.30.

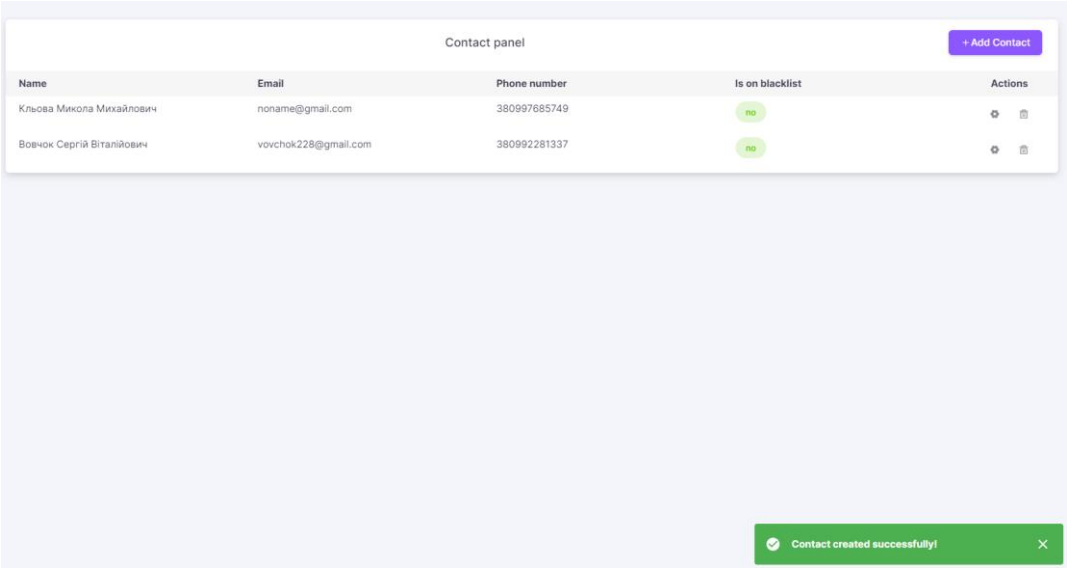


Рисунок 3.30 – Успішне створення контакту

Для оновлення контакту, потрібно натиснути на шестерню біля необхідного, рисунок 3.31.

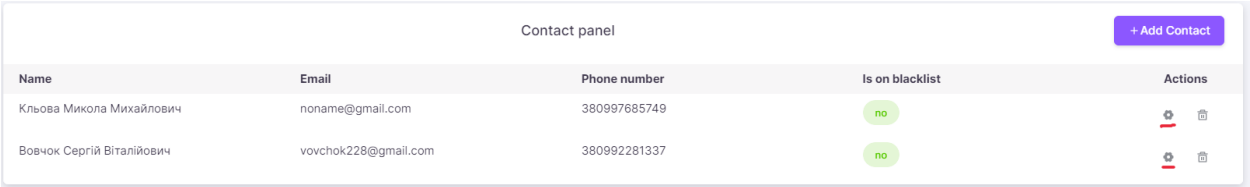


Рисунок 3.31 – Місце, куди потрібно натиснути для оновлення контакту

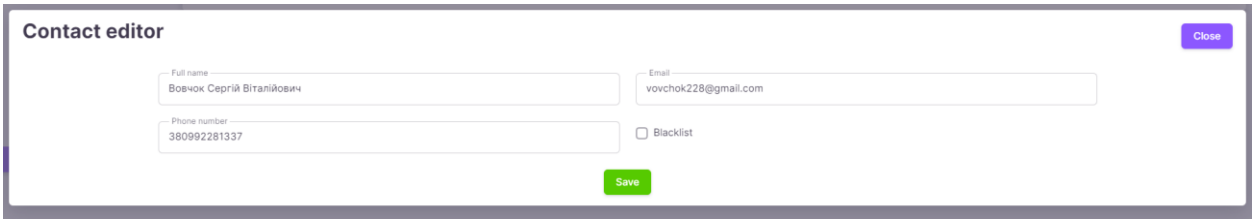


Рисунок 3.32 – Вікно оновлення контакту

Після оновлення необхідних полів, потрібно натиснути на кнопку “Save” і інформація оновиться у загальному списку і користувач буде проінформований про це у правому нижньому кутку, рисунок 3.33.

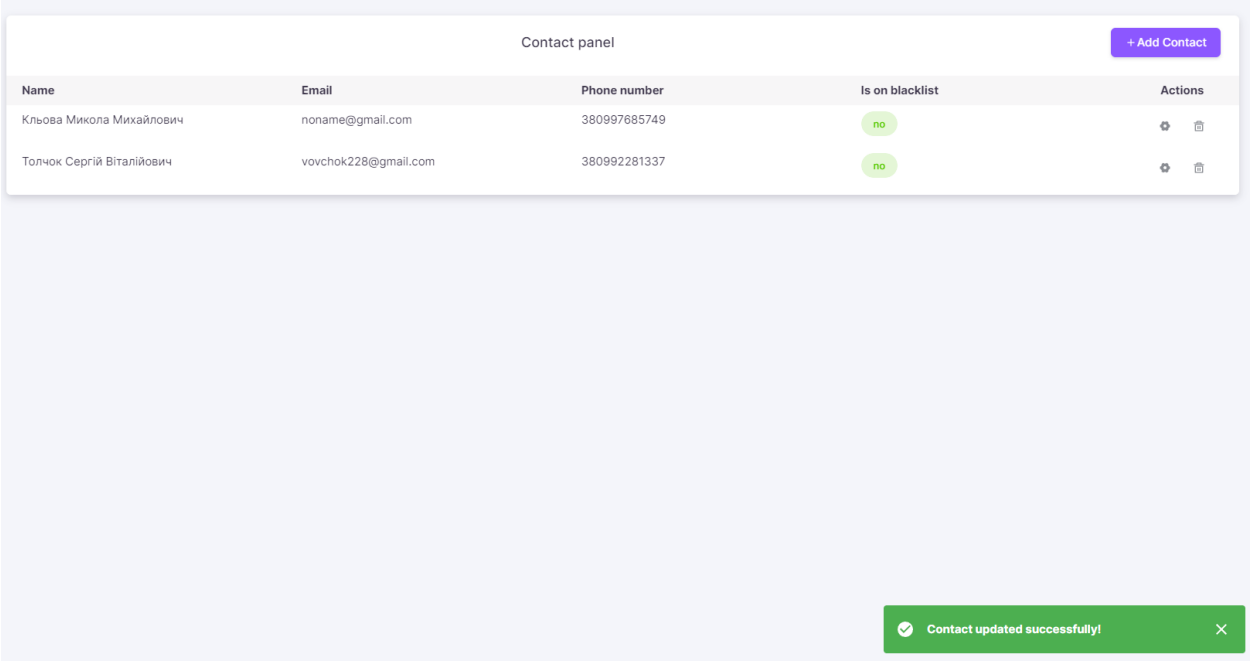


Рисунок 3.33 – Успішне оновлення контакту

Для видалення певного контаку з системи, потрібно натиснути на кошик, біля необхідного, рисунок 3.34.

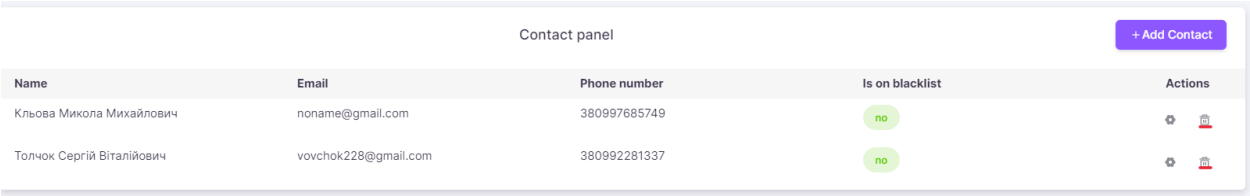


Рисунок 3.34 – Місце, куди потрібно натиснути для видалення контакту

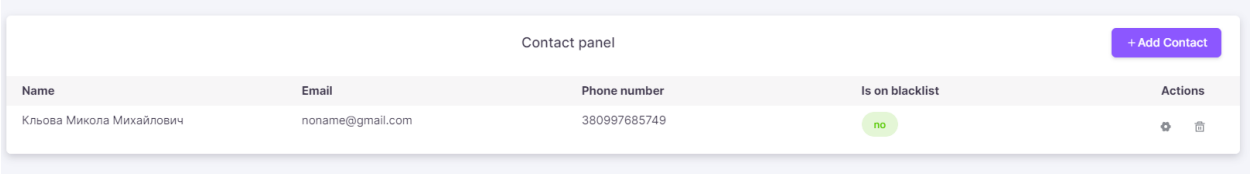


Рисунок 3.35 – Успішне видалення контакту

Для того, щоб переглянути можливі дії з бронюваннями потрібно натиснути на пункт меню “Reservations”, після чого, будуть відкриті 2 кнопки, пов’язані з бронюваннями “Create” і “View”, створення та перегляд – відповідно, рисунок 3.36.

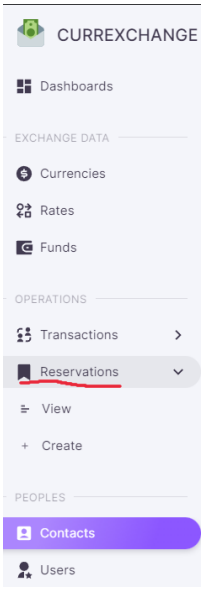


Рисунок 3.36 – Місце, куди потрібно натиснути для операцій з бронюваннями

Після кліку на підпункт “Create”, користувача буде перенаправлено на сторінку створення нового бронювання, рисунок 3.37.

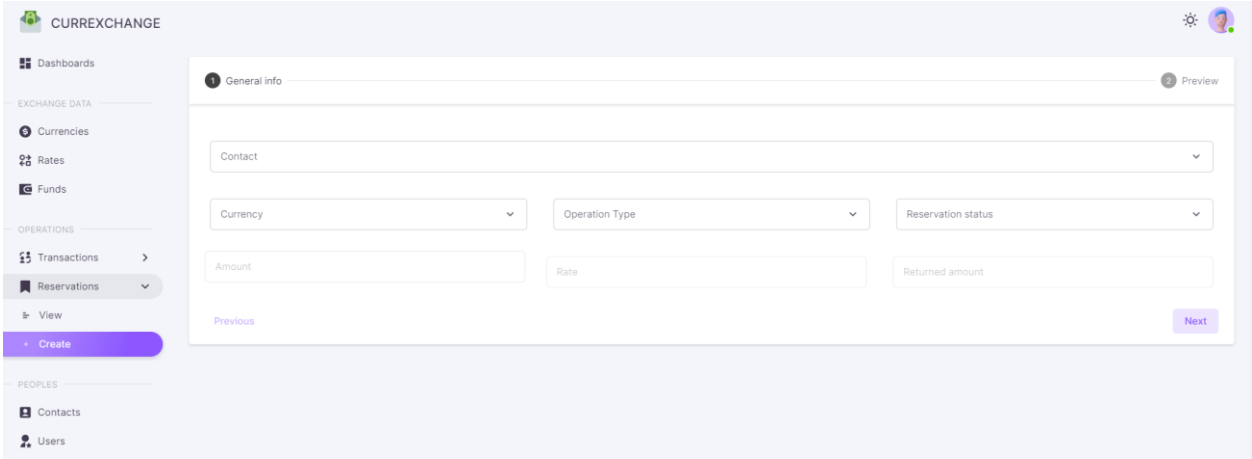


Рисунок 3.37 – Поля для заповнення для створення нового бронювання

Без заповнення необхідних полів, користувач не зможе перейти на наступний крок, рисунок 3.38.

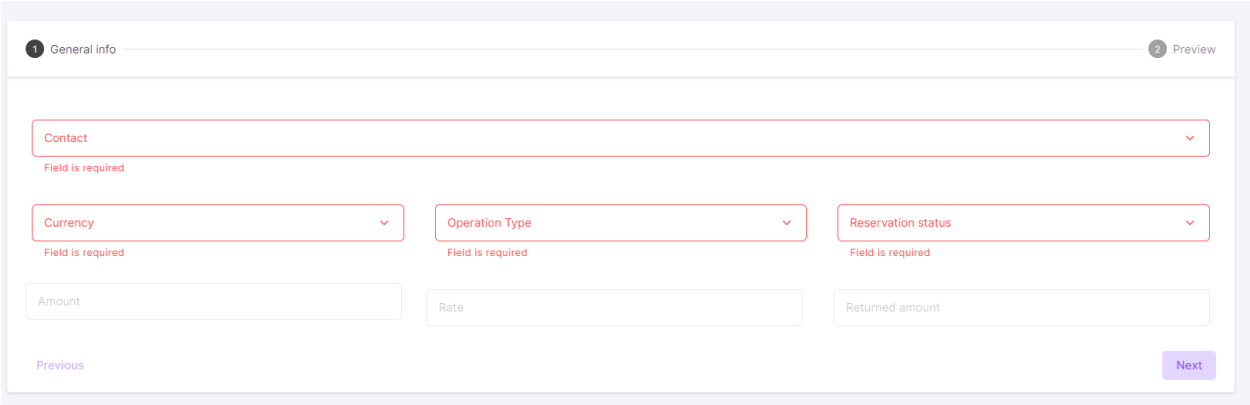


Рисунок 3.38 – Підказка, які поля є необхідними для заповнення

Після заповнення необхідних полів, потрібно натиснути на кнопку “Next”, рисунок 3.39.

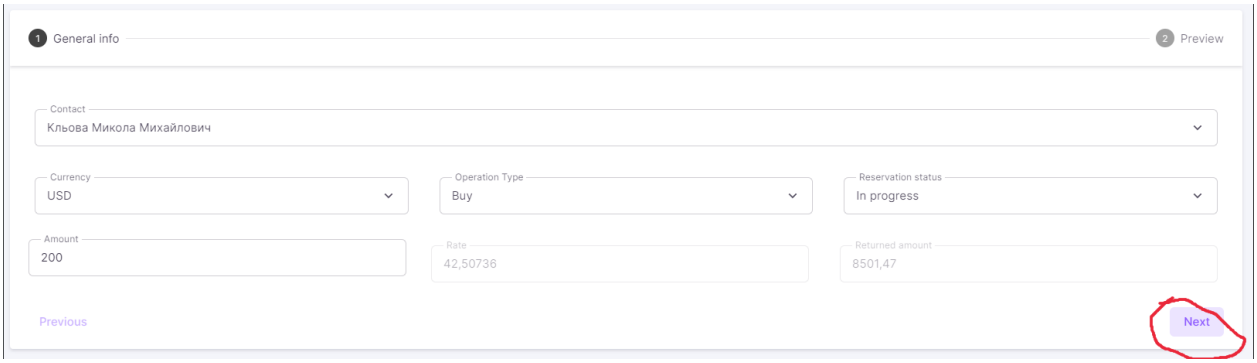


Рисунок 3.39 – Місце, куди потрібно натиснути для переходу на інший крок

Користувач повинен впевнитись у правильності введених даних, тому буде зображено вікно підтвердження, де буде зображено всі введені поля, але заблоковані для редагування, рисунок 3.40.

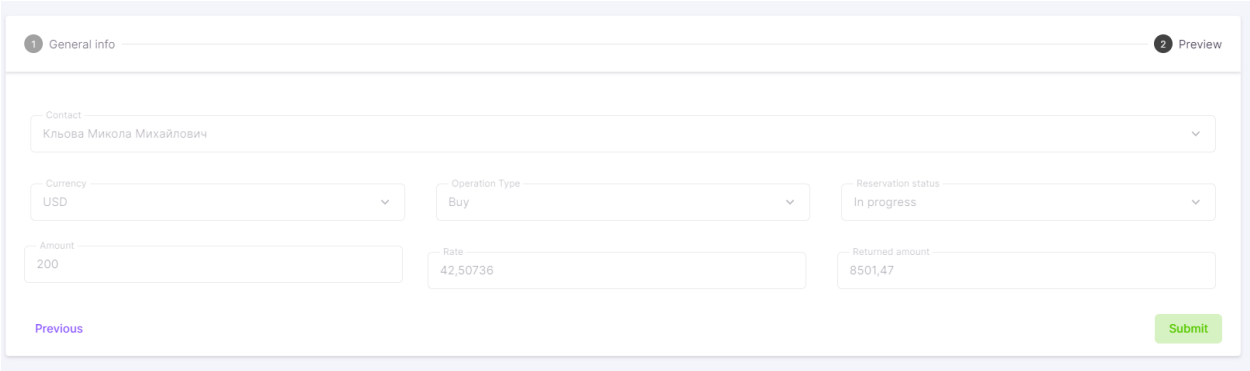


Рисунок 3.40 – Крок для перевірки правильності введених даних

Якщо було помічено, що якісь дані не є вірними, користувач може повернутися на попередній крок і виправити їх, натиснувши на кнопку “Previous”, рисунок 3.41.

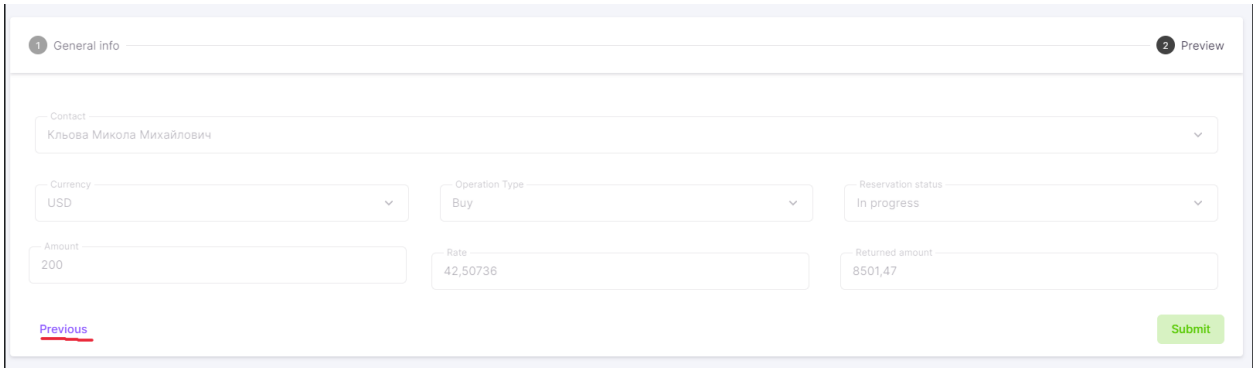


Рисунок 3.41 – Місце, куди потрібно натиснути для виправлення даних у полях

Після переконання, що всі дані є правильними, потрібно натиснути на кнопку “Submit”, після чого, користувача перенаправить на сторінку всіх бронювань у системі і він буде проінформований про успішність створення бронювання, рисунок 3.42.

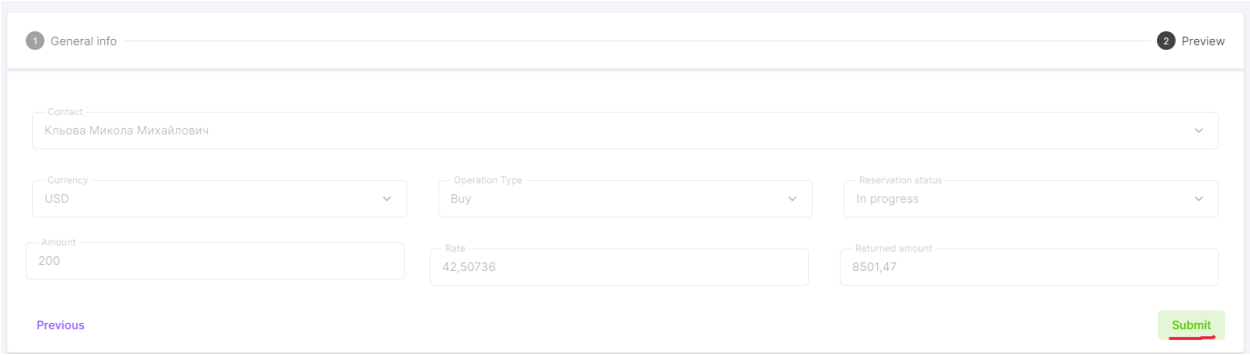


Рисунок 3.42 – Місце, куди потрібно настиснути для збереження бронювання у системі

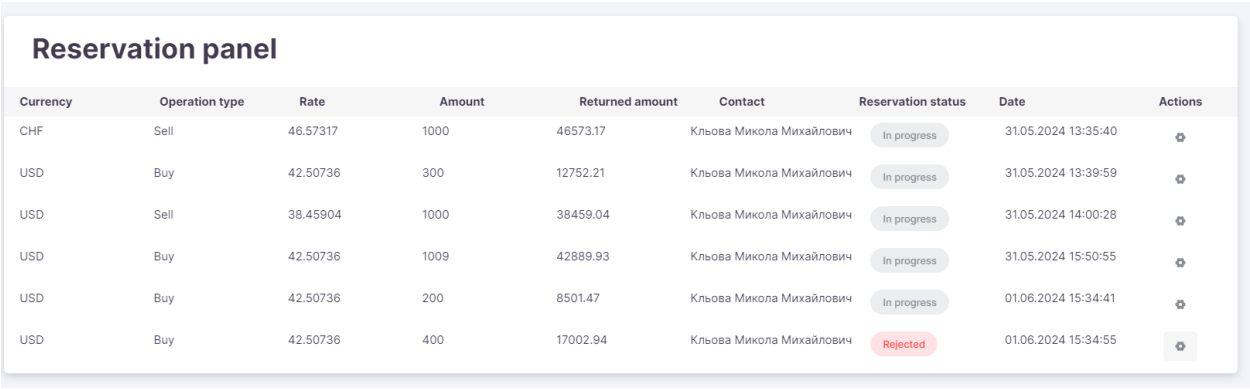


Рисунок 3.43 – Успішне створення бронювання

На минулу сторінку можна потрапити натиснувши на пункт меню у “Reservations” – “View”, рисунок 3.44.

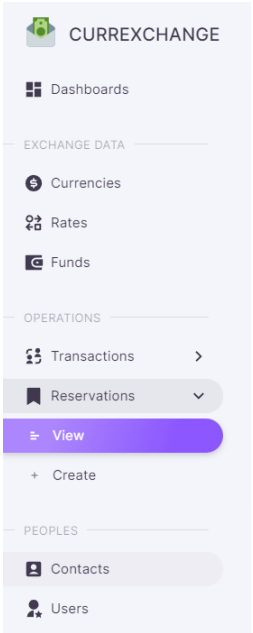


Рисунок 3.44 – Пункт меню для перегляду створених бронювань у системі

Коли клієнт приходить у пункт обміну, користувач повинен занести у систему інформацію про те, яка валюта була продана або куплена, по якому курсу і яка кількість. Для цього є спеціальний пункт у меню “Transactions”, рисунок 3.45.

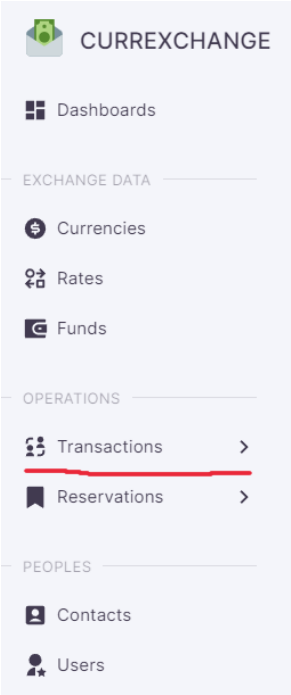


Рисунок 3.45 – Пункт меню для транзакцій проведених пунктом обміну

Для створення нової транзакції, потрібно розгорнути цей пункт і натиснути на пункт “Create”, рисунок 3.46.

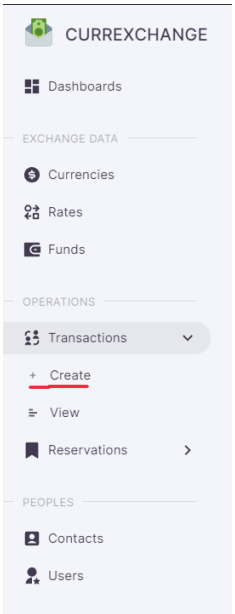


Рисунок 3.46 – Пункт меню для створення транзакції

Після натискання на пункт створення транзакції, користувачу буде відкрите вікно з полями, необхідними для заповнення, рисунок 3.47.

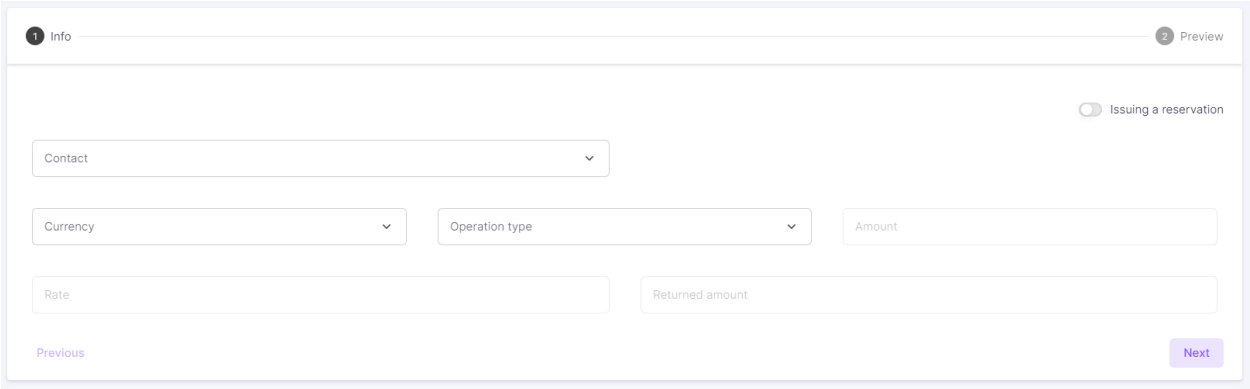


Рисунок 3.47 – Вікно для створення транзакції у системі

Якщо користувач пропустить обов’язкові поля для заповнення, система його повідомить про це і не дасть перейти на наступний крок, рисунок 3.48.

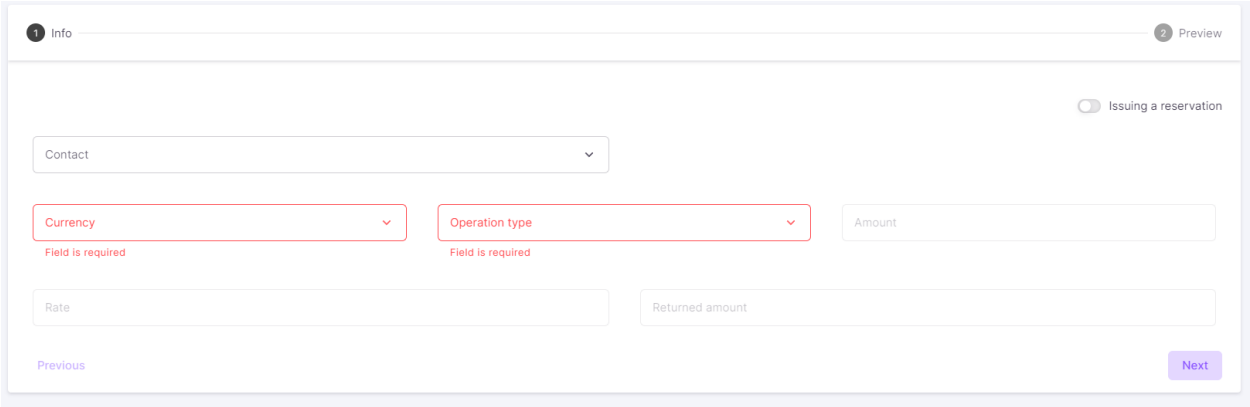


Рисунок 3.48 – Обов’язкові поля для заповнення

Після заповнення полів, треба перейти на наступний крок, перегляду введених даних, рисунок 3.49.

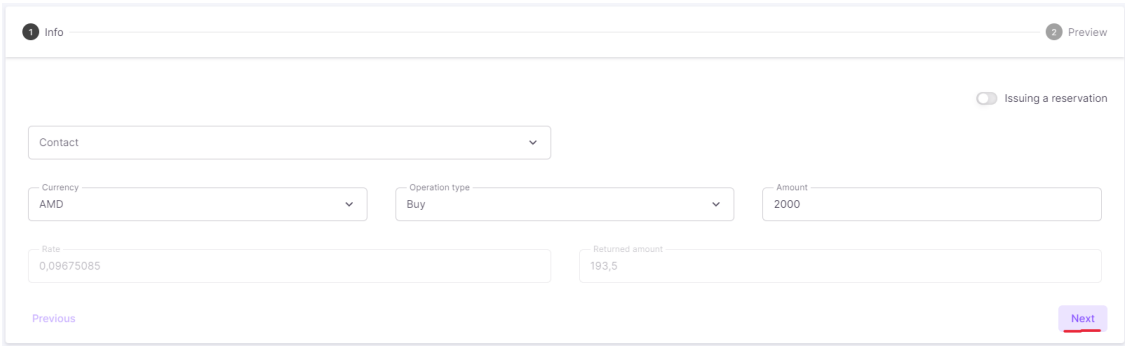


Рисунок 3.49 – Місце, куди треба натиснути для переходу на наступний крок

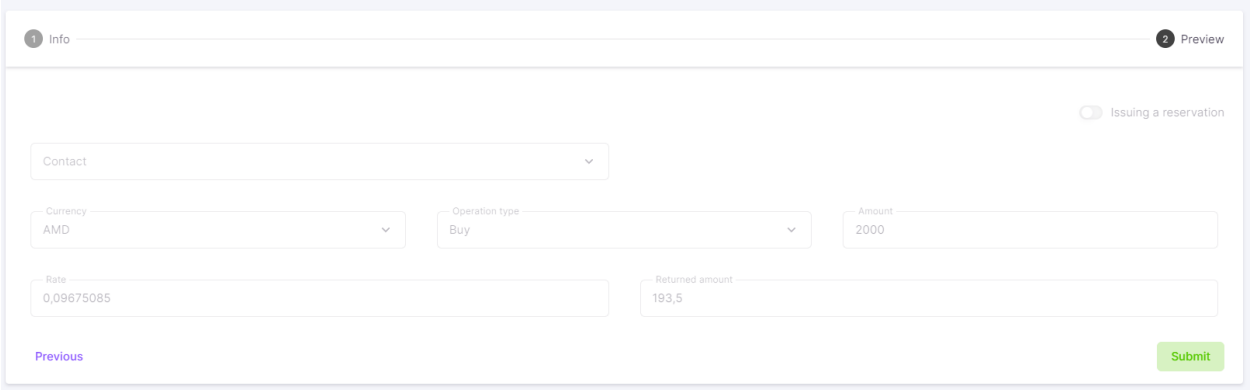


Рисунок 3.50 – Крок перегляду введеної інформації

Після того, як користувач впевнився у правильності занесеної інформації, він натискає на кнопку “Submit”, після чого, він буде перенаправлений на сторінку перегляду транзакцій у буде проінформований повідомленням у правому нижньому кутку про успішне створення транзакції, рисунок 3.51.

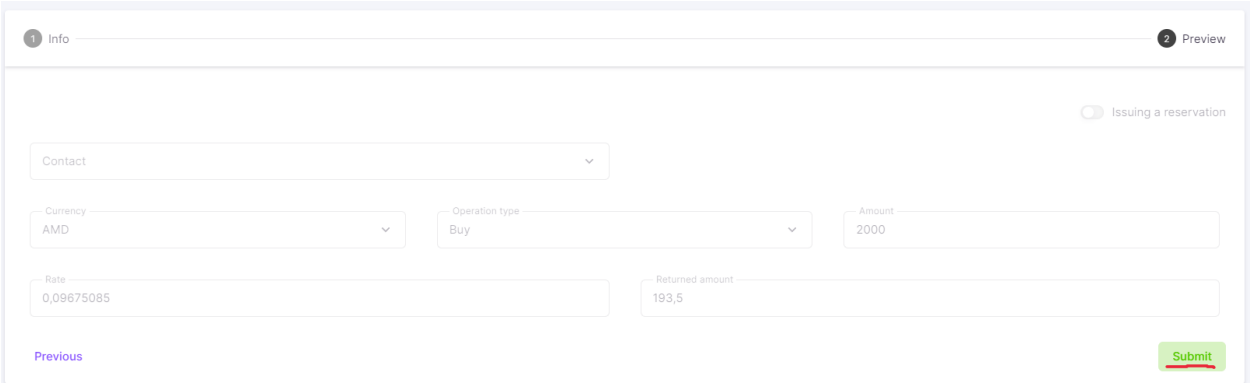


Рисунок 3.51 – Місце, куди потрібно натиснути для створення транзакції

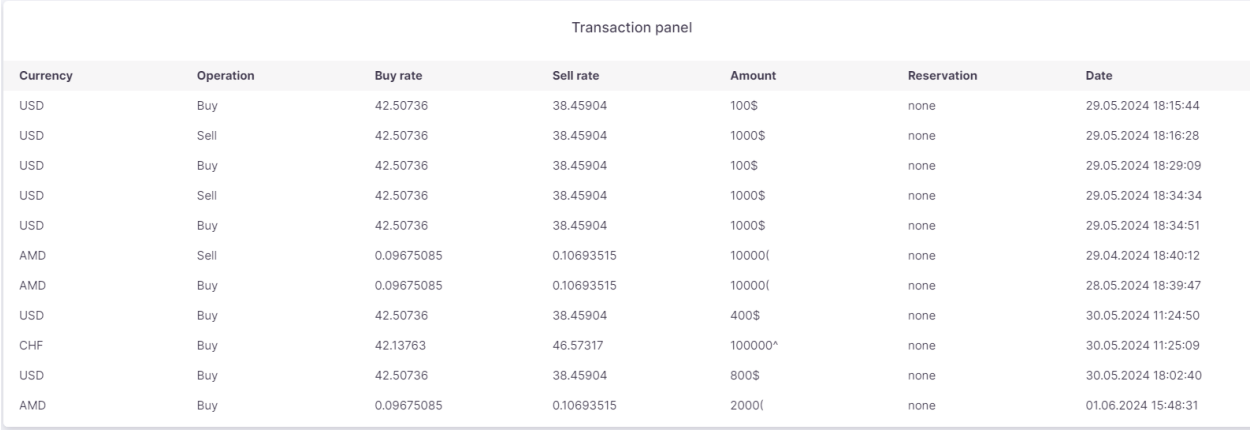


Рисунок 3.52 – Успішне створення транзакції

Є ситуація, коли людина приходить з певним бронюванням і у транзакції потрібно вказати, що вона належить до певного бронювання, тоді при створенні транзакції включаємо пункт “Issuing a reservation”, де додається поле з бронюваннями, яке повинні вибрати для транзакції і наступні кроки є ідентичними зі звичайним створенням транзакції, рисунок 3.53.

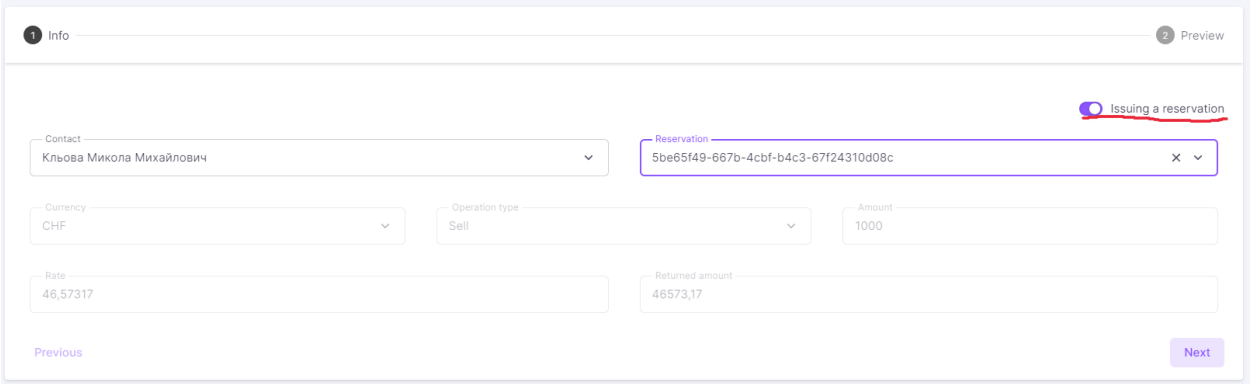


Рисунок 3.53 – Кнопка для активування режиму видачі бронювання

Для того, щоб переглянути проведені транзакції обмінником, потрібно натиснути на кнопку “View” у пункті “Transactions”, рисунок 3.54.

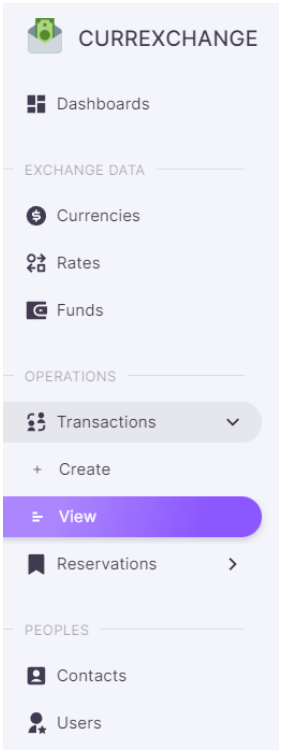


Рисунок 3.54 – Кнопка для перегляду проведених транзакцій

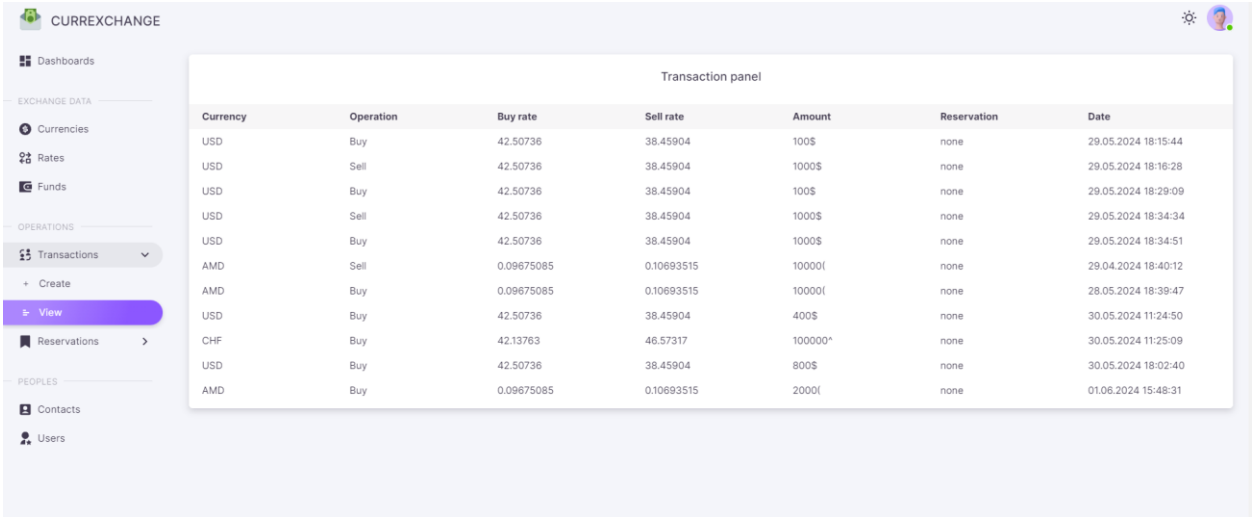


Рисунок 3.55 – Меню з проведеними транзакціями

Після завершення роботи у системі, користувач повинен вийти з неї, натиснувши на кнопку “Logout”, де його перенаправить на сторінку авторизації, рисунок 3.56.

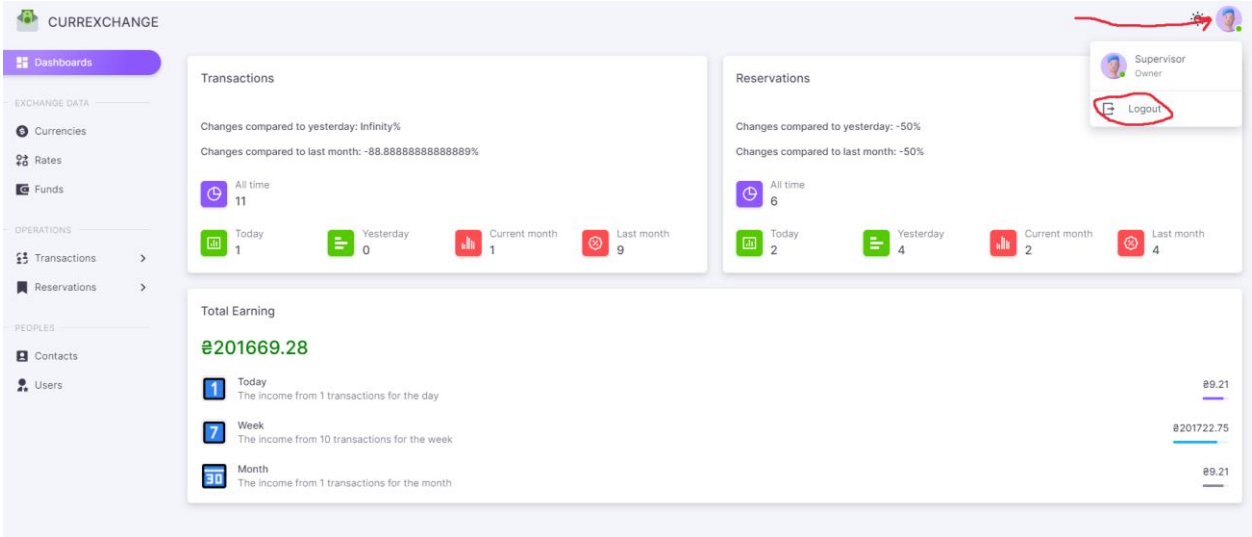


Рисунок 3.56 – Кнопка, яку потрібно натиснути для виходу з системи

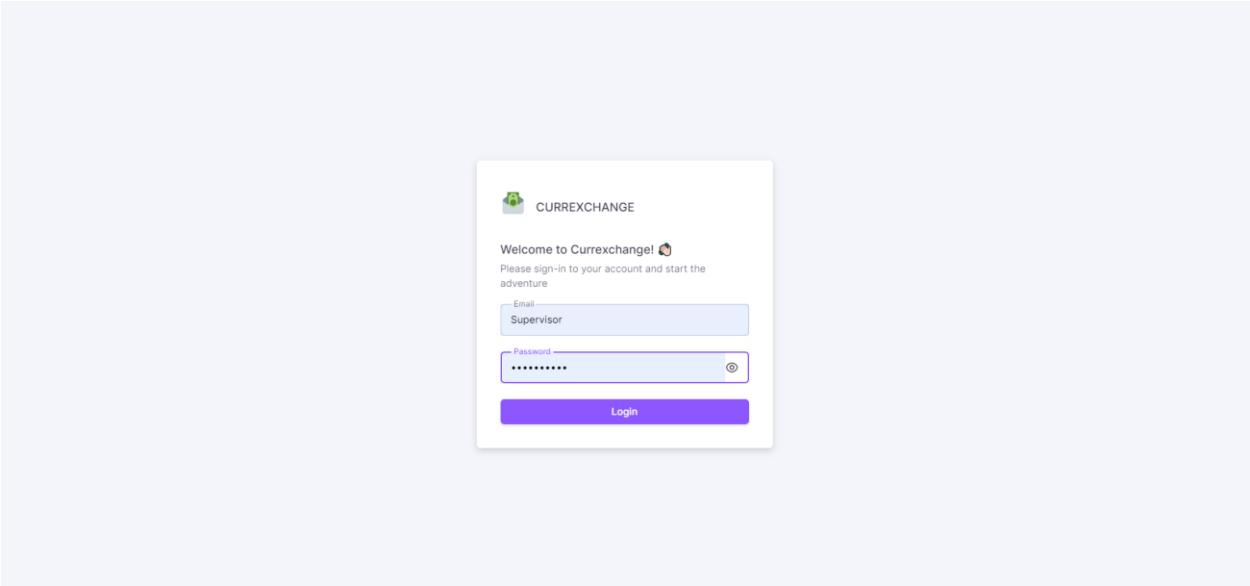


Рисунок 3.57 – Результат виходу з системи

3.2 Взаємодія з telegram-ботом клієнтам

Під час розгортання систем, власник створив назву для telegram-боту взаємодії з клієнтами, у нашому випадку, це: @currexchange_customerbot.

Клієнт заходить у телеграм, та у пошуку, обирає бота по назві @currexchange_customerbot, рисунок 3.58.

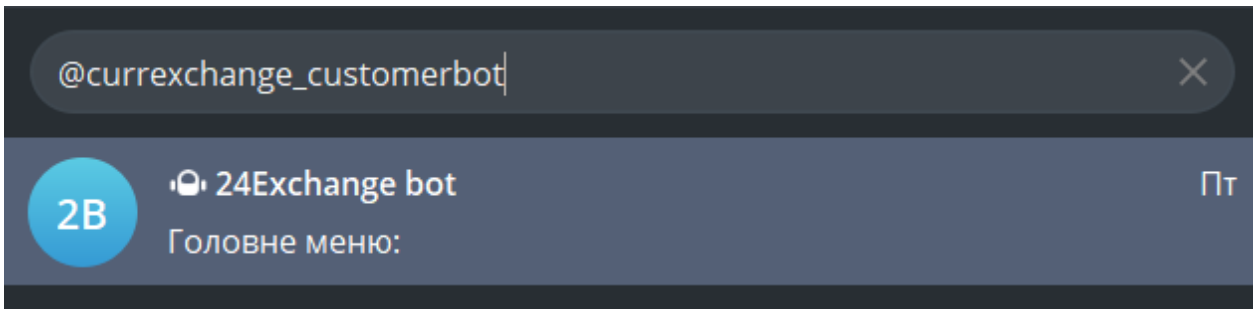


Рисунок 3.58 – Пошук боту на платформі Telegram

Треба перейти у нього і виконати команду /start, після виконання клієнту буде відображене головне меню, рисунок 3.59.

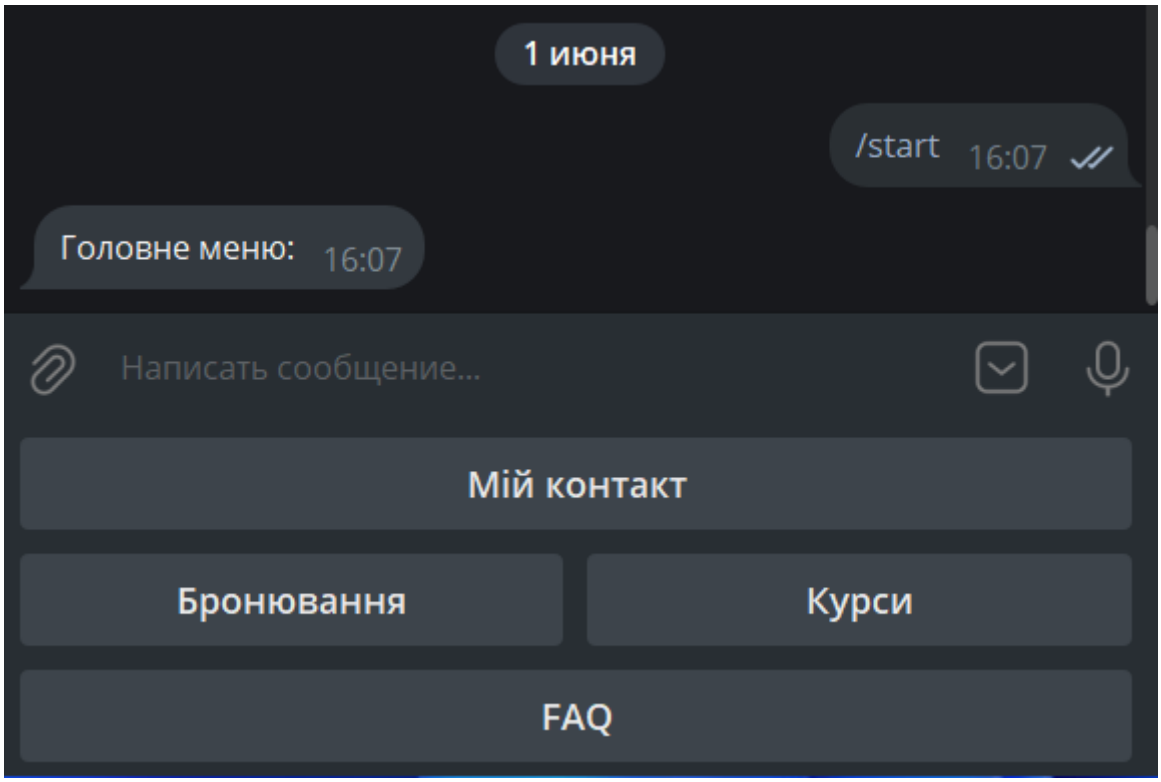


Рисунок 3.59 – Головне меню боту

Для того, щоб користуватись можливостями боту, потрібно створити свій контакт, тому після натискання на кнопку “Мій контакт”, користувачу необхідно заповнити дані, рисунок 3.60.

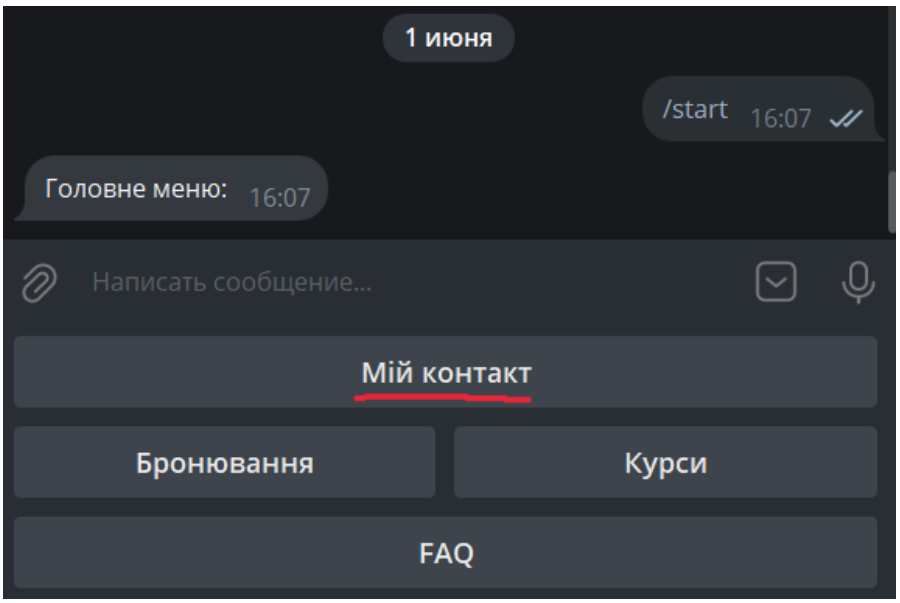


Рисунок 3.60 – Кнопка для створення мого контаку

Після відповідей на всі питання, система створить нового контакта, рисунок 3.61.

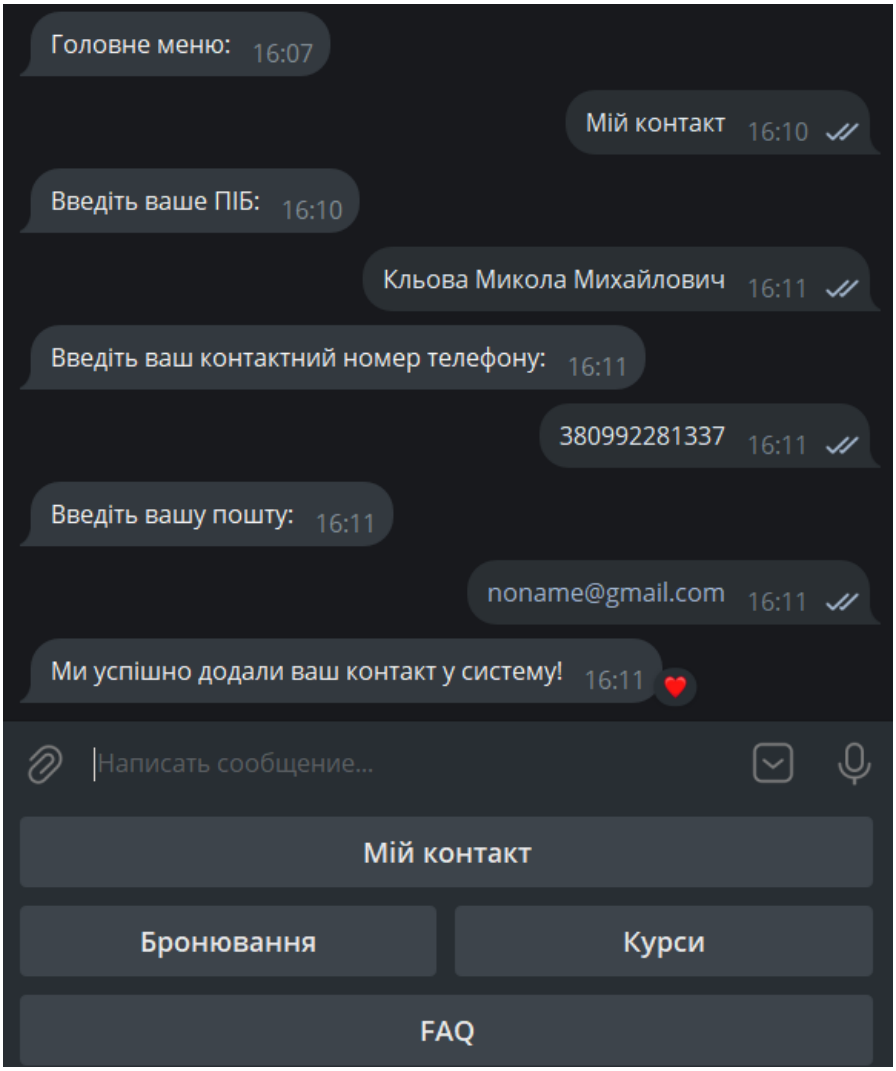


Рисунок 3.61 – Перелік питань для створення контакту

Після чого, користувач може впевнитись, в успішно створеному контакті, натиснувши на кнопку “Мій контакт”, ще раз, рисунок 3.62.

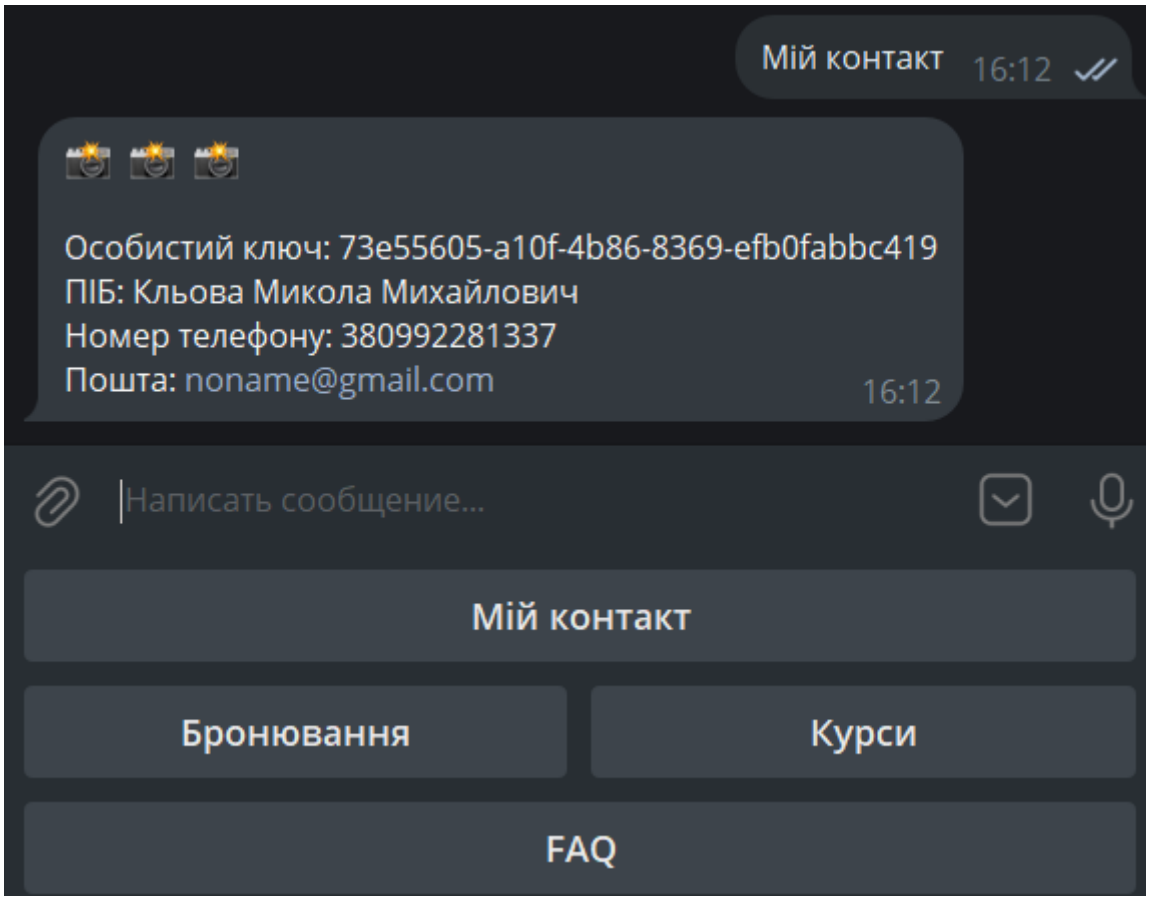


Рисунок 3.62 – Перевірка успішно створеного контакту

Після цього, користувачу можна використовувати систему повноцінно. Для ознайомлення з курсами валют, які представлені обмінником, потрібно натиснути на кнопку “Курси”, рисунок 3.63.

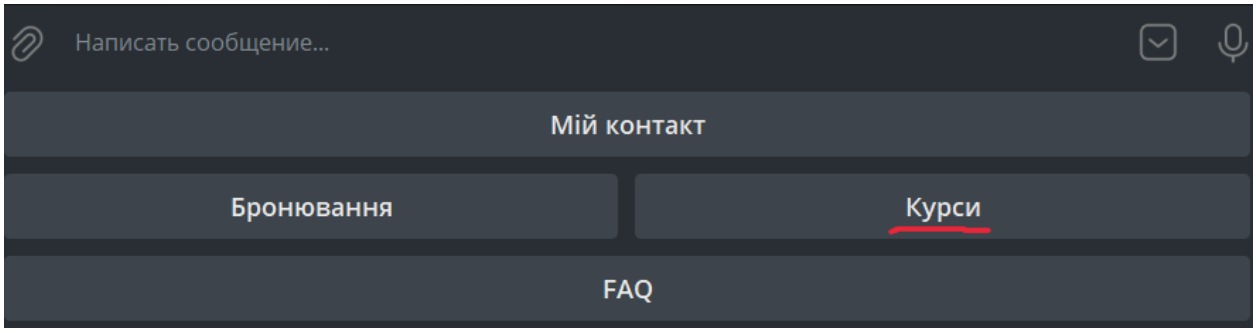


Рисунок 3.63 – Кнопка, на яку потрібно натиснути для перегляду курсів валют, які представлені обмінником

Після чого, користувачу наданий список валют, з якого він потрібен обрати ту, яка його цікавить, у нашому випадку, це “USD”, рисунок 3.64.

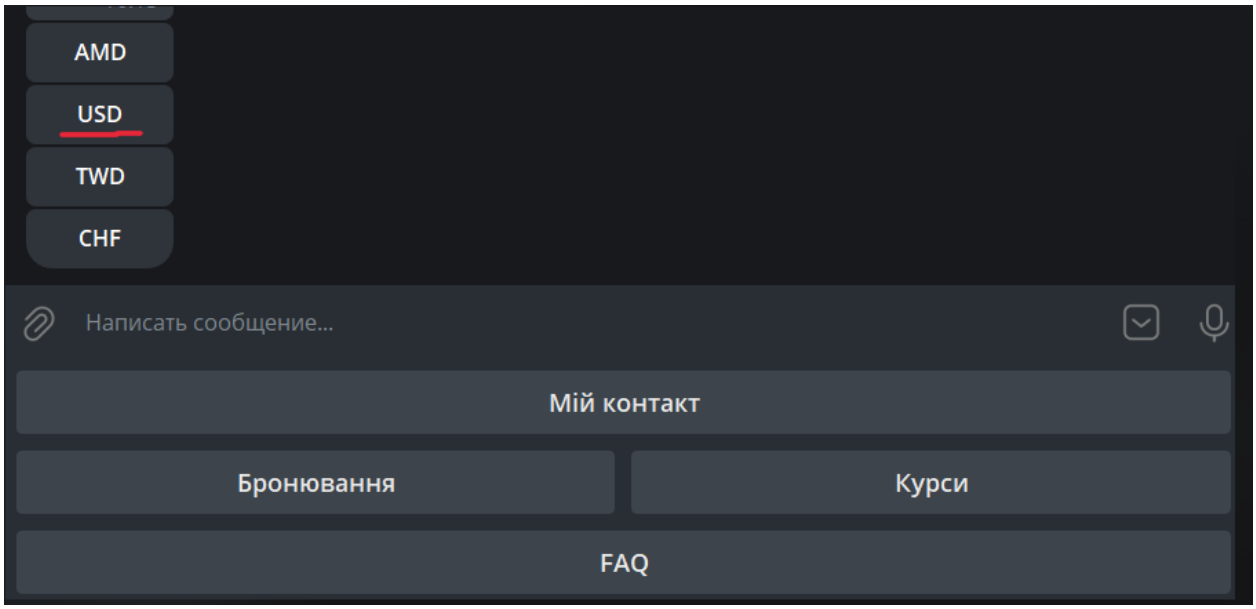


Рисунок 3.64 – Кнопка, на яку треба натиснути, для перегляду курсів валюти USD

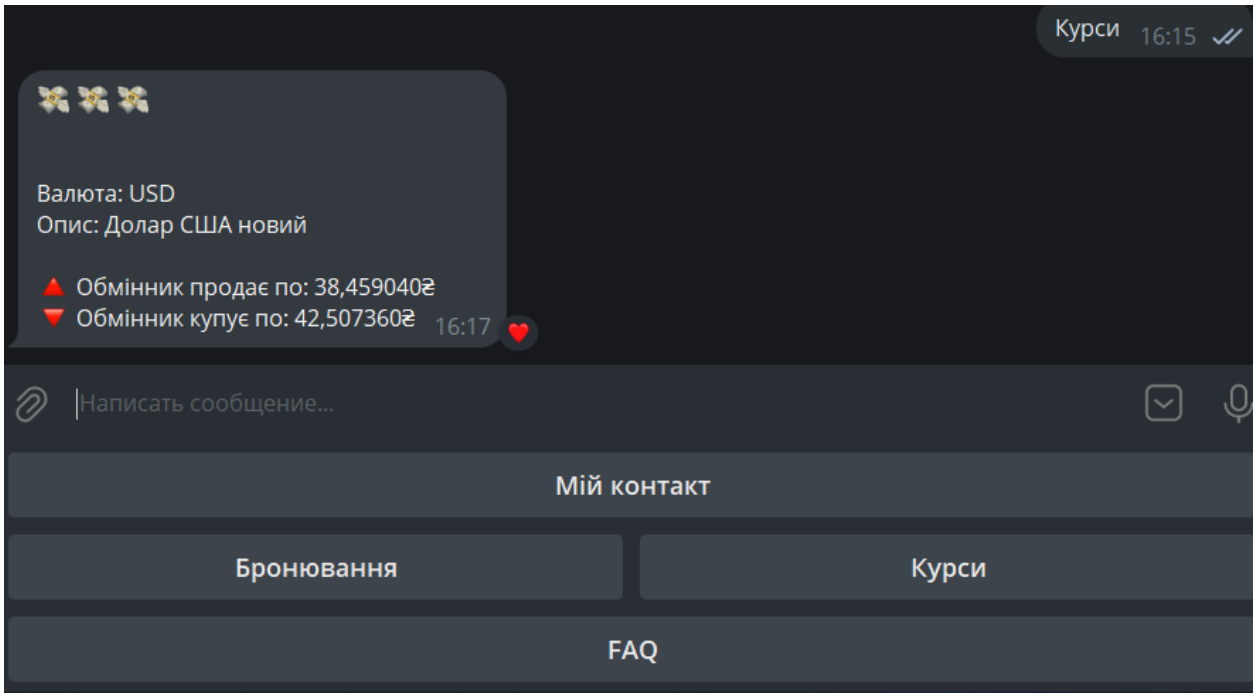


Рисунок 3.65 – Результат вибору валюти

Для того, щоб користуватись функціоналом бронювання, потрібно обрати пункт у головному меню “Бронювання”, рисунок 3.66.

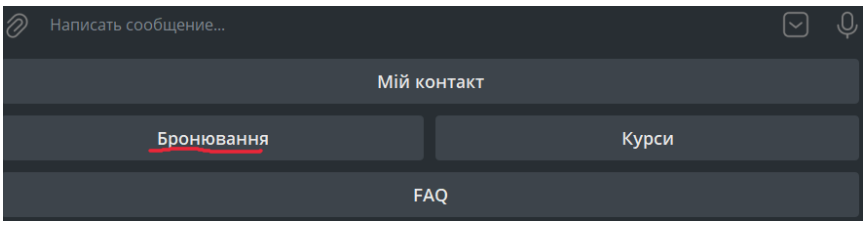


Рисунок 3.66 – Пункт для переходу у меню бронювань

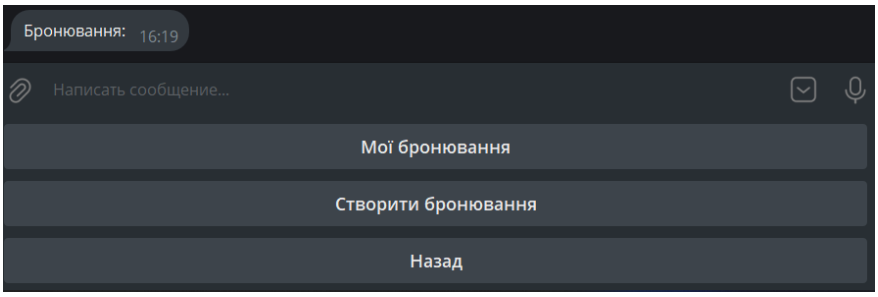


Рисунок 3.67 – Меню бронювання

Щоб створити нове бронювання, потрібно натиснути на кнопку “Створити бронювання”, рисунок 3.68.

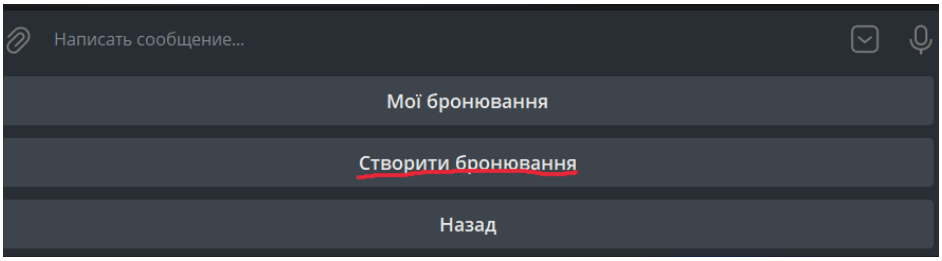


Рисунок 3.68 – Кнопка для створення нового бронювання

Після цього, користувачу запроповновано валюти, по яким він може створити бронювання, рисунок 3.69.

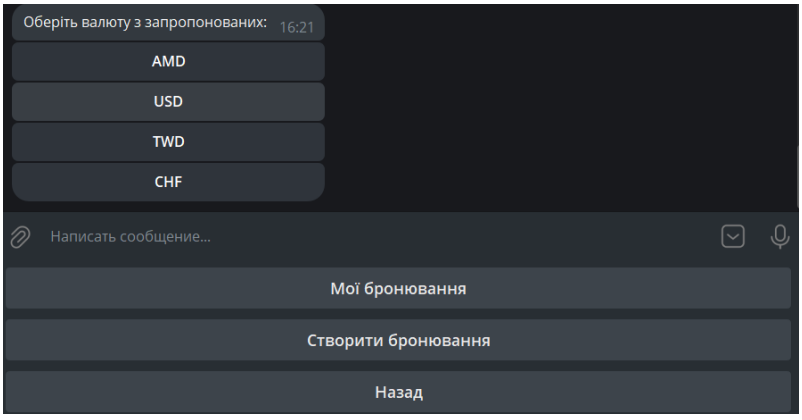


Рисунок 3.69 – Валюти доступні для бронювання

Після вибору необхідної валюти, потрібно вибрати операцію, яка необхідна для виконання, рисунок 3.70.

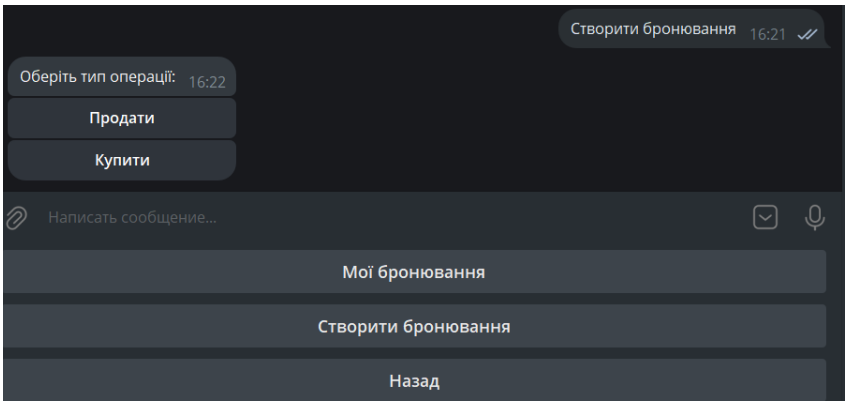


Рисунок 3.70 – Операції для бронювання

Після того, як обрано необхідну дію, треба ввести кількість валюти, яку користувач хоче забронювати, рисунок 3.71.

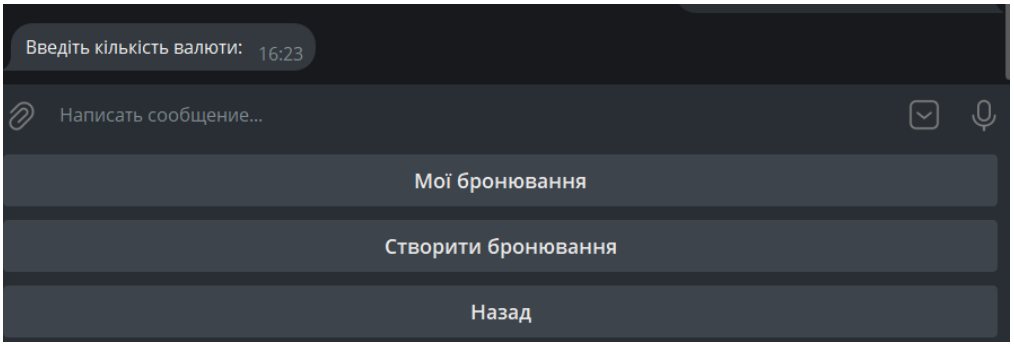


Рисунок 3.71 – Введення кількості валюти

Після успішного вводу кількості валюти, виведено вікно підтвердження, з актуальним курсом на даний момент, рисунок 3.72.



Рисунок 3.72 – Підтвердження бронювання

Після підтвердження введених даних, створюється бронювання і користувач проінформований повідомленням, рисунок 3.73.

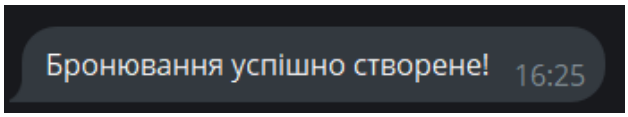


Рисунок 3.73 – Інформація про успішно створене бронювання

Інформацію про створене бронювання, можна переглянути у пункті бронювань, настиснувши на кнопку “Мої бронювання”, рисунок 3.74.

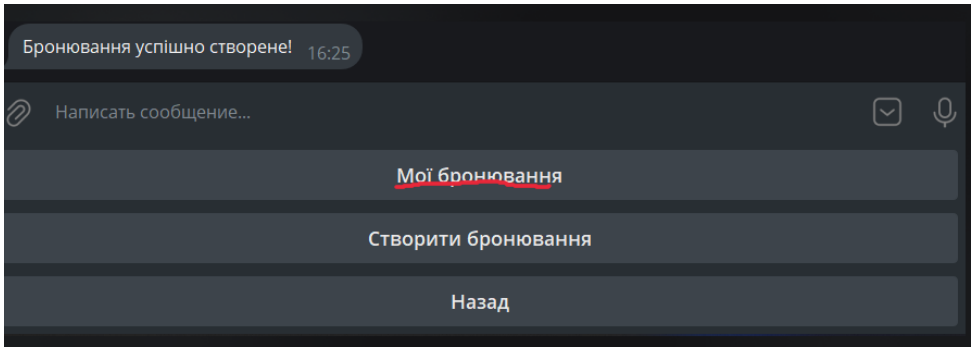


Рисунок 3.74 – Кнопка для перегляду створених бронювань

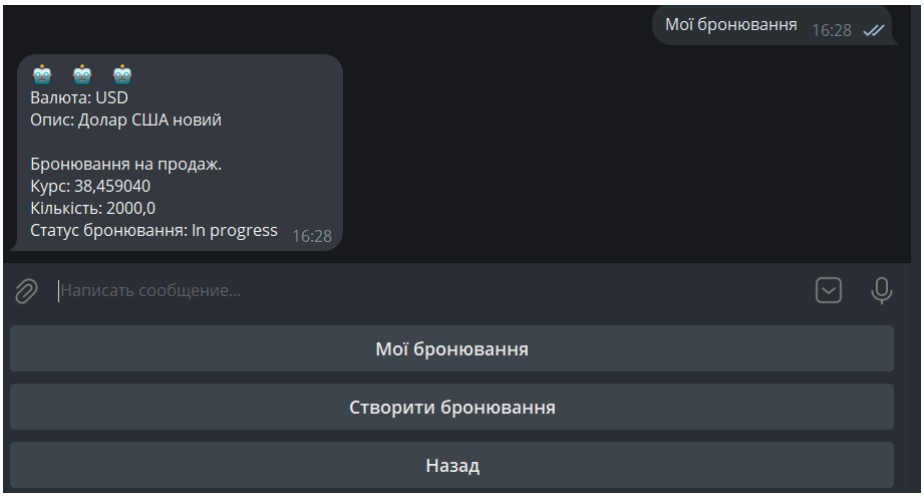


Рисунок 3.75 – Повідомлення з створеними бронюваннями у системі для цього контакту

Всі дані, з приводу нового контакту і бронювань, також будуть відображені адміністраторам у їх системі, де вони уже оперуватимуть ними за потреби.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ ПУНКТУ ОБМІНУ ВАЛЮТ

Графічний матеріал

КПІ.ІТ-0406.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Едуард ЖАРІКОВ

Нормоконтроль:
_____ Павло РОДІОНОВ

Виконавець:
_____ Микола КЛЬОВА

Київ – 2024

КПІ.ІТ-0406.045440.06.99.СБД



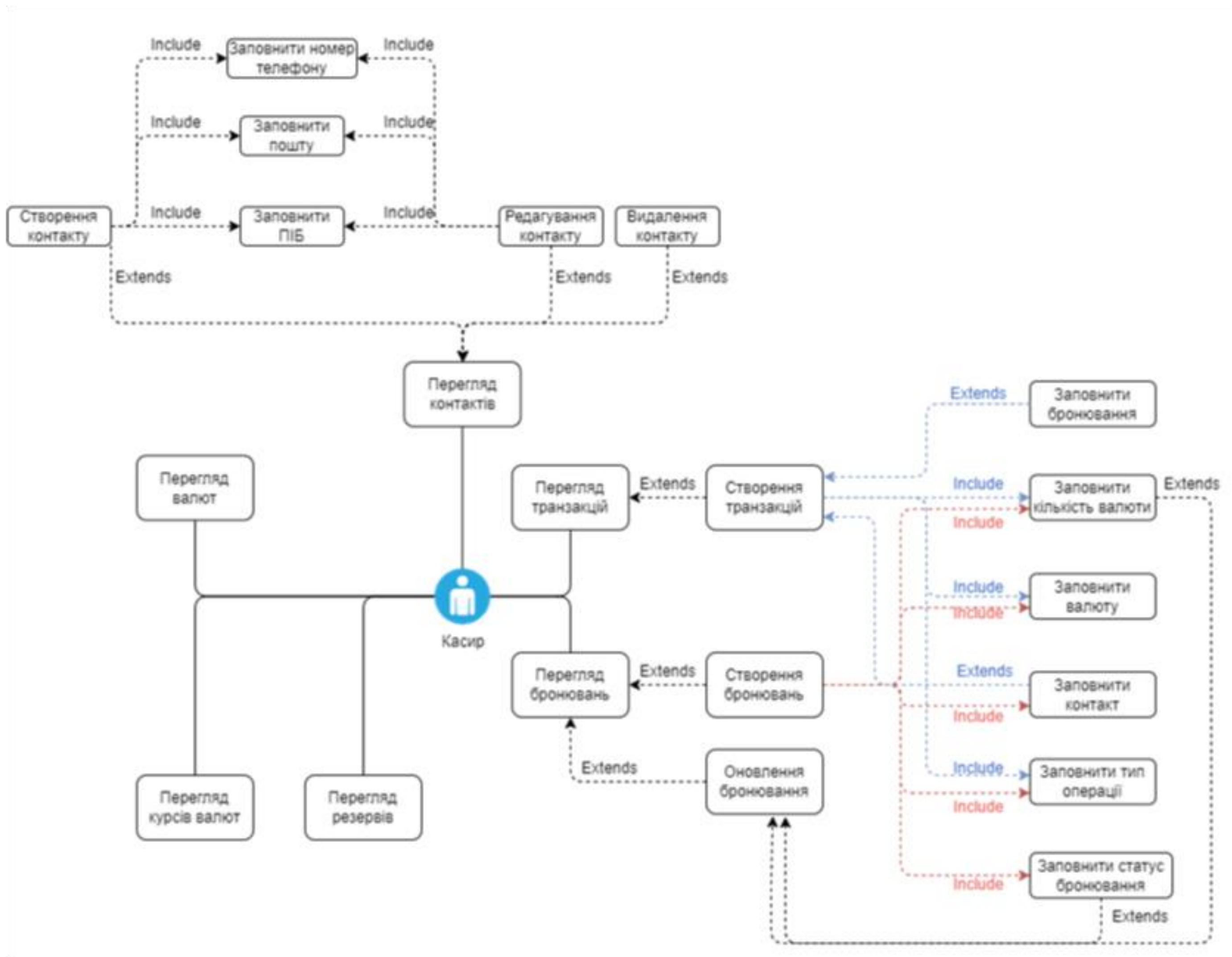
					КПІ.ІТ-0406.045440.06.99.СБД			
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Кльова М.М.			Схема бази даних	Літера	Маса	Масштаб
Перевірив		Жаріков Е.В.						
Т. контр.						Аркуш		Аркушів
					Веб-застосунок для підтримки діяльності пункту обміну валют	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04		
Н. контр.		Родіонов П.Ю.						
Затвердив		Жаріков Е.В.						

КПІ.ІТ-0406.045440.06.99.ССК

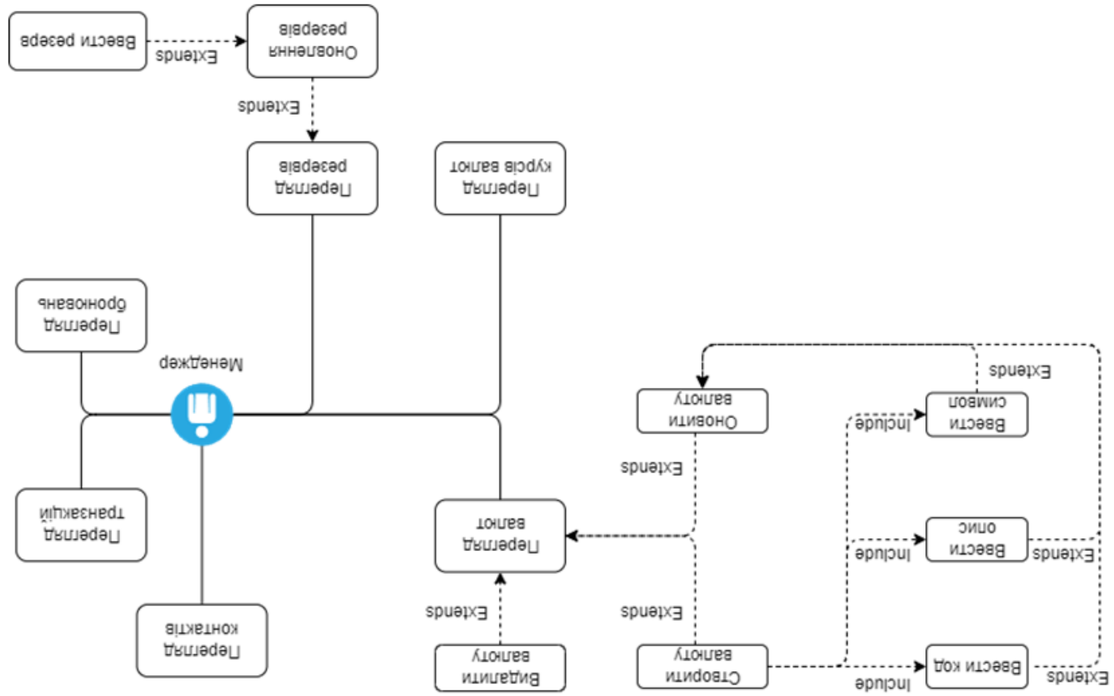


					КПІ.ІТ-0406.045440.06.99.ССК		
					Діаграма варіантів використання клієнта	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Кльова М.М.				Аркуш	Аркушів
Перевірив		Жаріков Е.В.					
Т. контр.							
					Веб-застосунок для підтримки діяльності пункту обміну валют	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04	
Н. контр.		Родіонов П.Ю.					
Затвердив		Жаріков Е.В.					

КПІ.ІТ-0406.045440.06.99.ССВ

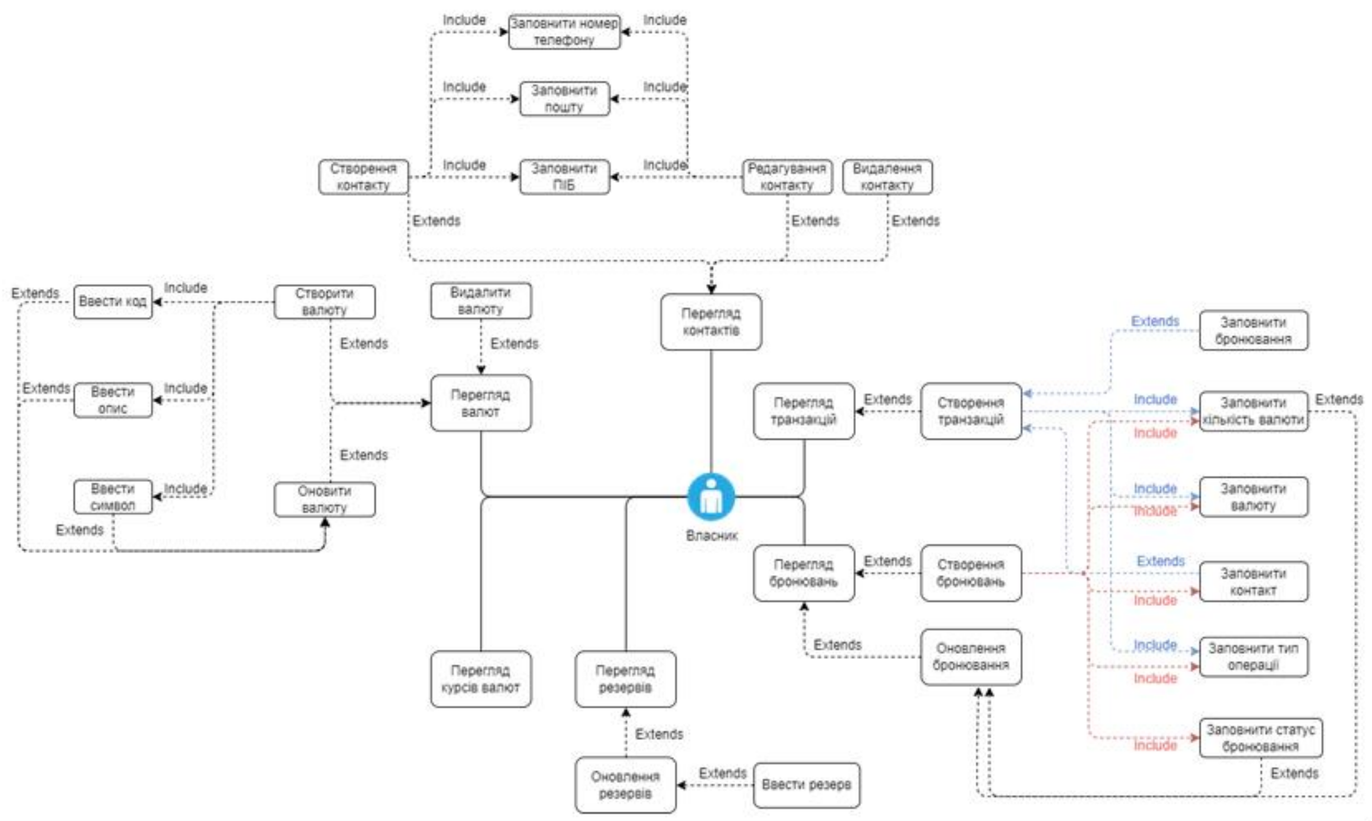


					КПІ.ІТ-0406.045440.06.99.ССВ			
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Кльова М.М.			Діаграма варіантів використання касиру	Літера	Маса	Масштаб
Перевірив		Жаріков Е.В.						
Т. контр.						Аркуш	Аркушів	
					Веб-застосунок для підтримки діяльності пункту обміну валют	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04		
Н. контр.		Родіонов П.Ю.						
Затвердив		Жаріков Е.В.						



										КПІ.ІТ-0406.045440.06.99.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Діаграма варіантів використання менеджеру					Літера	Маса	Масштаб
Розробив		Клькова М.М.										
Перевірив		Жарікова Е.В.										
Т. контр.										Аркуш	Аркушів	
Н. контр.		Родіонов П.Ю.			Веб-застосунок для підтримки діяльності пункту обміну валют					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04		
Затвердив		Жарікова Е.В.										

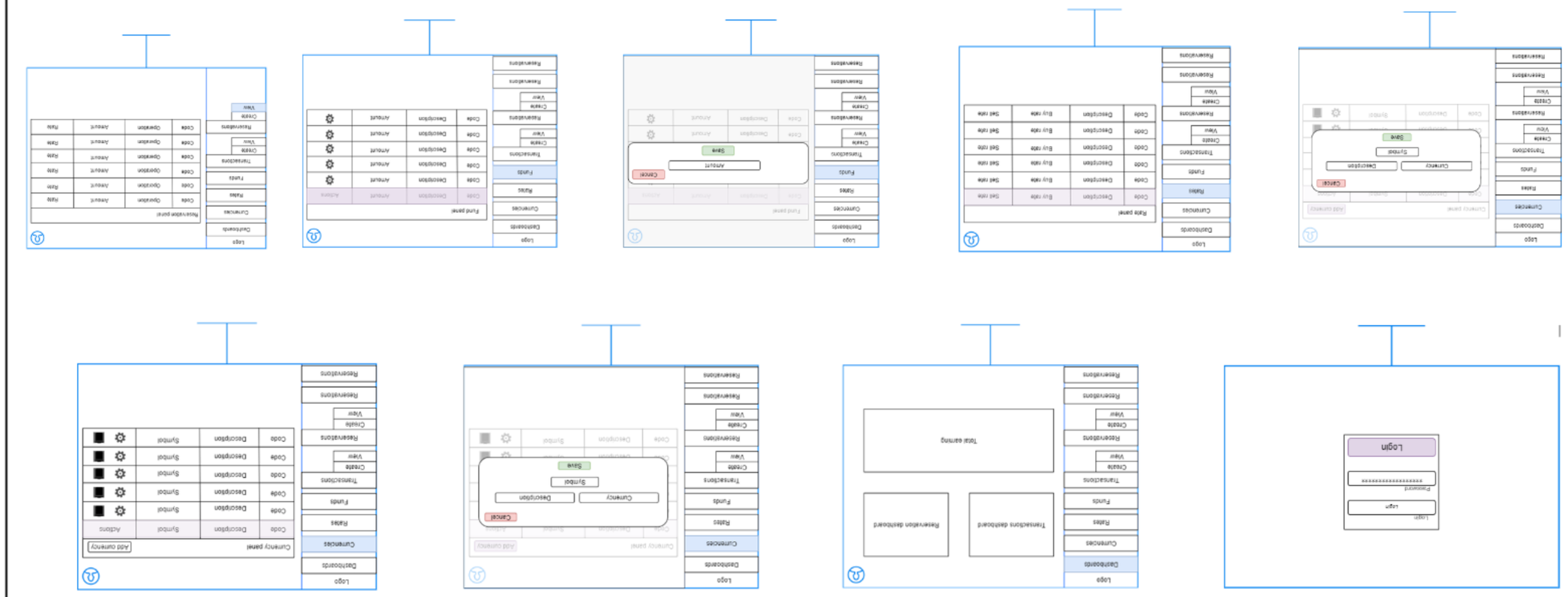
КПІ.ІТ-0406.045440.06.99.ССВ



					КПІ.ІТ-0406.045440.06.99.ССВ
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Кльова М.М.			Діаграма варіантів використання менеджеру
Перевірив		Жаріков Е.В.			
Т. контр.					
Н. контр.		Родіонов П.Ю.			Веб-застосунок для підтримки діяльності пункту обміну валют
Затвердив		Жаріков Е.В.			

Літера	Маса	Масштаб
Аркуш	Аркушів	

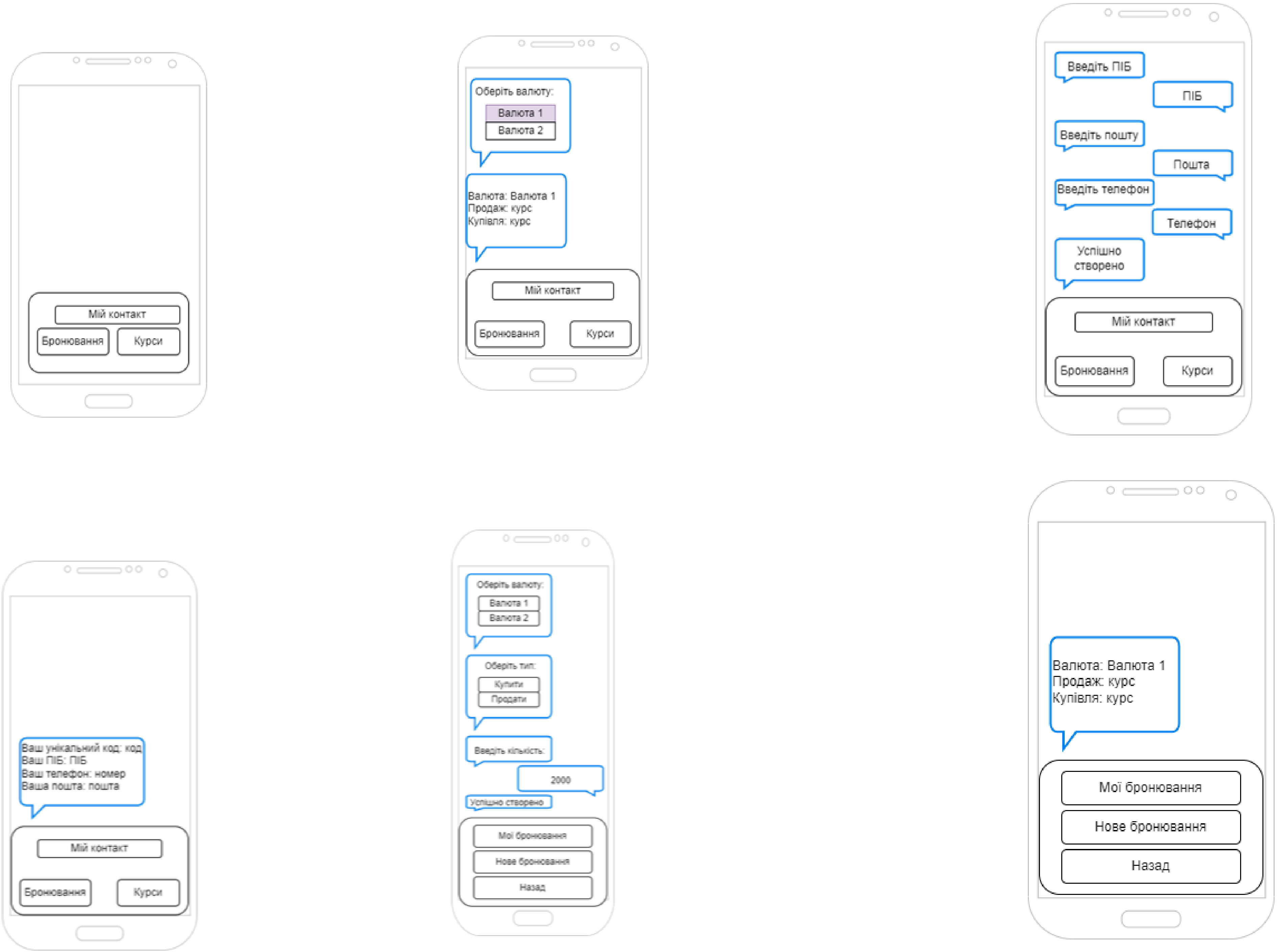
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-04



Зм.	Арк.	№ докумeнта	Підпис	Дата	Кресленнa вигляду екранних форм			
Розробив		Клюєва М.М.				Літера	Маса	Масштаб
Перевірив		Жаріков Е.В.						
Т. контр.						Аркуш	Аркушів	
Н. контр.		Родіонов П.Ю.			Веб-застосунок для підтримки діяльності пункту обміну валют	КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІТ-04		
Затвердив		Жаріков Е.В.						

КПІ.ІТ-0406.045440.06.99.КЕ

КПІ.ІТ-0406.045440.06.99.КЕ



Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-0406.045440.06.99.КЕ		
Розробив		Кльова М.М.			Креслення вигляду екранних форм	Літера	Маса	Масштаб
Перевірив		Жаріков Е.В.				Аркуш	Аркушів
Т. контр.					Веб-застосунок для підтримки діяльності пункту обміну валют	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04	
Н. контр.		Родіонов П.Ю.					
Затвердив		Жаріков Е.В.					