НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126«Інформаційні системи та технології»
на тему: «Інтерактивна система управління та налаштування Telegramботів»

Виконав:
студент IV курсу, групи ІК-91
Новосельцев Олексій Олександрович _____

Керівник:
доктор технічних наук, професор,
Жураковський Богдан Юрійович _____

Рецензент:
доктор технічних наук, доцент
Жебка Вікторія Вікторівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма«Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Новосельцеву Олексію Олександровичу

1. Тема проєкту «Інтерактивна система управління та налаштування Telegram ботів», керівник проєкту Жураковський Богдан Юрійович,доктор технічних наук, професор, затверджені наказом по університету від 31 травня 2023 р. № 2101-с

2. Термін подання студентом проєкту: 12 червня 2023 року

3. Вихідні дані до проєкту:наукова література, технічне завдання для створення власного програмного забезпечення керування телеграм ботами.

4. Зміст пояснювальної записки: аналіз предметної області, оглядіснуючих рішень, опис обраних технологій, проєктування, розробка та розгортання програмного забезпечення, висновки.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): діаграма прецедентів, блок-схема алгоритму відновлення даних,ER-діаграма бази даних, діаграма послідновності.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження тематики роботи	13.03.2023	
2	Вивчення необхідних технологій	13.03.2023 – 18.03.2023	
3	Аналіз аналогічних рішень	19.03.2023 – 05.04.2023	
4	Пошук оптимального стеку для розробки	06.04.2023 – 10.04.2023	
5	Розробка архітектури системи	11.04.2023 – 23.04.2023	
6	Розробка програмного забезпечення	24.04.2023 – 10.06.2023	
7	Оформлення документації	10.06.2023 – 12.06.2023	

Студент Олексій НОВОСЕЛЬЦЕВ

Керівник Богдан ЖУРАКОВСЬКИЙ

АНОТАЦІЯ

Новосельцев О. О. Інтерактивна система управління та налаштування Telegram ботів. КПІ ім. Ігоря Сікорського, Київ, 2023.

Проєкт містить 67 с. тексту, 23 рисунків, 22 таблиці, посилання на 16 літературних джерелта 4 конструкторських документи.

Дипломний проєкт присвячений розробці комплексу задач з автоматизації та спрощення налаштування Telegram ботів

Проводено аналіз доменної обаласті, визнначені вимоги, побудована архітектура додатку, архітектура бази даних. Додаток було розгорнуто та протестовано

Ключові слова: TELEGRAM, TOKENS, BOT, OWASP, SECURITY

.

ABSTRACT

Novoseltsev O. O. Interactive System for Management and Configuration of Telegram Bots. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2023.

The project consists of 67 pages of text, 20 figures, 22 tables, references to 16 literary sources, and 4 design documents.

The diploma project is dedicated to the development of a comprehensive solution for automating and simplifying the configuration of Telegram bots.

Domain analysis was conducted, requirements were identified, application architecture and database architecture were built. The application was deployed and tested.

Keywords: TELEGRAM, TOKENS, BOT, OWASP, SECURITY.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК91.280БАК.005 ПЗ	Пояснювальна записка	62		
6	А3	ІК91.280БАК.005 Д1	Інтерактивна система	1		
7			управління та налаштування			
8			Telegram ботів. Діаграма			
9			прецендентів			
10	А3	ІК91.280БАК.005 Д2	Інтерактивна система	1		
11			управління та налаштування			
12			Telegram ботів. Блок-схема			
13			алгоритму відновлення версій			
14	А3	ІК91.280БАК.005 Д3	Інтерактивна система	1		
15			управління та налаштування			
16			Telegram ботів. ER-діаграма			
17			БД			
18	А3	ІК91.280БАК.005 Д4	Інтерактивна система	1		
19			управління та налаштування			
20			Telegram ботів. Діаграма			
21			послідовності			
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІК91.029БАК.005 ТП		
Розроб.		Новосельцев О.О.			Інтерактивна система управління та налаштування Telegram ботів. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Жураковський Б.Ю.				Т	
Затв.						КПІ ім. Ігоря Сікорського Група ІК-91	

Пояснювальна записка
до дипломного проєкту
на тему: «Інтерактивна система управління та налаштування Telegram ботів»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

User flow - візуальне відображення послідовності дій, які користувач здійснює в системі

ER-діаграма - Entity-Relationship діаграма. Діаграма, що показує зв’язок між таблицями в базі даних

БД – База даних

JWT – JavaScript Web Tokens.

UC –Приклад використання (UseCase)

FR – Функціональна вимога (Functional Requirement)

HTTP(S) – HyperText Transfer Protocol (Secure)

XSS – Cross-Site Scripting

OSI – Модель комунікаційних протоколів (Open Systems Interconnection)

ВСТУП

У сучасному світі месенджери і чат-боти відіграють все більш важливу роль у спілкуванні та взаємодії між користувачами. Telegram, який є одним з найпопулярніших месенджерів, пропонує багато можливостей для створення власних чат-ботів. Однак, процес їх створення та налаштування може бути викликаним для користувачів без технічних навичок та програмістського досвіду.

Telegram боти стали потужним інструментом для підтримки бізнесу, забезпечуючи автоматизацію процесів, взаємодію з клієнтами та надання послуг. Розробка інтерактивної системи дозволить бізнес-власникам без технічних навичок налаштовувати ботів, що відкриє нові можливості для автоматизації бізнес-процесів, покращення обслуговування клієнтів та збільшення ефективності.

Інтерактивна система управління та налаштування Telegram ботів дозволить користувачам налаштовувати різні функції та взаємодіяти з ботами у зручний спосіб.

Застосування Vue.js на клієнтській стороні і Node.js на сервері надасть розробникам зручний та ефективний інструментарій для розробки системи. Це спростить процес створення та підтримки системи, а також сприятиме швидкому впровадженню нових функціональних можливостей.

Метою даного проекту є спростити та зробити доступним процес створення, управління та налаштування Telegram ботів шляхом розробки інтерактивної системи. У кінці користувачі зможуть швидко і просто налаштовувати Telegram ботів, створюючи мапи діалогів, а також керувати ними, маючи потужну систему збереження історії.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Проєкт “інтерактивна система управління та налаштування Telegram ботів» є адміністративно-керівницькою системою. Адміністративно-керівницька система є комплексним рішенням, розробленим для забезпечення ефективного керування та налаштування різноманітних процесів. Вона надає можливість адміністраторам та власникам системи отримати повний контроль над функціональністю та діяльністю різних складових.

Одна з важливих характеристик адміністративно-керівницької системи - це централізоване управління. Вона забезпечує єдину точку доступу для адміністраторів, що дозволяє зручно керувати різними аспектами системи з одного місця. Це включає управління користувачами, ролями та правами доступу, конфігурацією налаштувань системи, моніторингом та аналізом даних.

Адміністративно-керівницька система також характеризується гнучкістю та розширюваністю. Вона має можливість адаптуватися до змінних потреб користувачів і дозволяє вносити зміни у налаштування, функціональність та інші аспекти системи з мінімальними зусиллями. Це дозволяє власникам системи пристосовувати її до своїх потреб та змінювати її відповідно до розвитку бізнесу або проекту.

У системі також можуть бути реалізовані функції моніторингу та аналізу. Це дозволяє адміністраторам отримувати статистику та звіти про різні параметри та показники системи. Наприклад, це може бути інформація про активність користувачів, час роботи системи, використання ресурсів, помилки та інші важливі дані. Аналіз таких даних дозволяє зрозуміти ефективність та продуктивність системи, виявляти проблеми та приймати відповідні рішення для їх вирішення.

Ці загальні характеристики допомагають забезпечити ефективне та продуктивне управління системою, забезпечуючи зручність та гнучкість для адміністраторів та користувачів.

1.2 Змістовний опис і аналіз предметної області

Предметна область цього проєкту включає в себе Telegram ботів та їх управління. Telegram боти відомі своєю зручністю та багатогранністю використання, що викликало широку популярність серед користувачів. Зручність Telegram ботів полягає у їх простоті та зручності взаємодії для користувачів. Боти забезпечують широкі можливості комунікації та виконання різноманітних завдань без необхідності встановлення додаткових програм або навчання складних інтерфейсів.

Переваги Telegram ботів полягають у їх простоті та доступності для користувачів. Взаємодія з ботами відбувається безпосередньо в месенджері Telegram, що має широке поширення і доступний на різних платформах. Користувачам не потрібно встановлювати окремий додаток або програмне забезпечення - вони можуть просто знайти бота в пошуку чи перейти за посиланням і почати спілкування.

Telegram боти також відрізняються своєю гнучкістю та розширюваністю. Вони можуть виконувати різні завдання, починаючи від надсилання інформації, відповіді на запити та виконання команд, до автоматизації певних процесів, створення опитувань або навіть надання послуг. Боти можуть бути використані для особистого використання, комунікації з брендами або підтримки бізнесу.

Зручність Telegram ботів також полягає в їх інтуїтивному та простому інтерфейсі. Більшість ботів пропонують текстову взаємодію або кнопкове меню, що дозволяє користувачам легко навігувати та вибирати потрібні опції. Крім того, Telegram підтримує візуальні елементи, такі як зображення,

графіки, відео або аудіофайли, що дозволяє ботам бути різноманітними та привабливими для користувачів.

Одна з ключових переваг Telegram ботів - це їх можливість інтеграції з іншими сервісами та додатками. Боти можуть надавати доступ до інформації, оновлювати статуси, відправляти повідомлення на зовнішні платформи, такі як соціальні мережі або CRM-системи. Це дає можливість легко об'єднувати різні сервіси та джерела інформації в одному місці.

У підсумку, Telegram боти відзначаються своєю зручністю завдяки простоті взаємодії, доступності, гнучкості, простому інтерфейсу та можливості інтеграції з іншими сервісами. Вони надають зручний і ефективний спосіб комунікації та виконання завдань у месенджері Telegram.

1.3 Аналіз існуючих рішень

Проведемо аналіз сучасного алгоритмічного забезпечення та технічних рішень, які застосовуються в області управління та налаштування Telegram-ботів. Також розглянемо доступні допоміжні програмні засоби, засоби розробки та наявні готові програмні рішення для реалізації такої системи.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Загальною метою аналізу алгоритмічних та технічних рішень є визначення найкращих практик та технологій, що допоможуть реалізувати ефективну та функціональну систему управління та налаштування Telegram-ботів.

Аналізуючи технічні рішення, які можуть бути використані в проекті управління та налаштування Telegram-ботів, можна розглянути наступні аспекти:

1. Архітектура системи:

- Розглянути різні архітектурні підходи, такі як клієнт-серверна архітектура або мікросервісна архітектура, для забезпечення модульності та масштабованості системи.

- Розподілені системи: розглянути можливість розподіленої архітектури для покращення надійності та швидкодії системи.

2. Зберігання та управління даними:

- Розглянути різні бази даних, такі як реляційні або NoSQL, для зберігання конфігурацій, даних користувачів та статистики.

- Розглянути можливості кешування даних для підвищення швидкодії системи та зменшення навантаження на базу даних.

3. Аутентифікація та авторизація:

- Розглянути різні методи автентифікації та авторизації, такі як використання токенів, JWT або OAuth, для забезпечення безпеки та контролю доступу до системи.

4. Комунікація з Telegram API:

- Розглянути можливості використання Telegram API для взаємодії з Telegram-ботами, зокрема для отримання та надсилання повідомлень, отримання статистики та керування налаштуваннями ботів.

5. Інтерфейс користувача:

- Розглянути можливості розробки зручного та інтуїтивно зрозумілого інтерфейсу користувача для управління та налаштування Telegram-ботів.

- Розглянути можливості використання фреймворків або бібліотек для швидкого створення інтерактивних інтерфейсів.

6. Моніторинг та аналітика:

- Розглянути можливості використання інструментів моніторингу та аналітики для збору та аналізу даних щодо активності ботів, їх продуктивності та взаємодії з користувачами.

7. Забезпечення безпеки:

- Розглянути можливості використання шифрування для захисту конфіденційної інформації та даних користувачів.

- Розглянути можливості використання заходів безпеки, таких як обмеження доступу до функцій бота або обробка небезпечних даних вхідних запитів.

Враховуючи вищезазначені аспекти, аналіз технічних рішень допоможе визначити оптимальні підходи та інструменти для реалізації системи управління та налаштування Telegram-ботів, незалежно від конкретних технологій використання.

1.3.2 Аналіз відомих програмних продуктів

Давайте розглянемо деякі аналоги до проекту системи управління та налаштування Telegram-ботів:

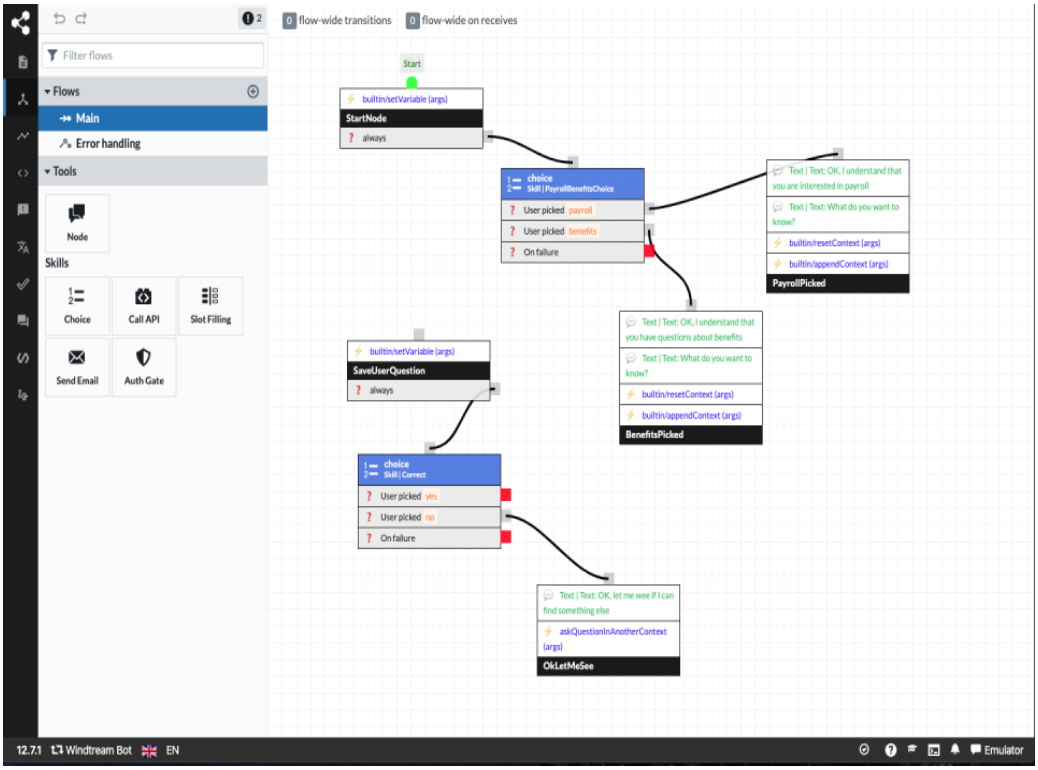

Рисунок 1.1 – Інтерфейс Botpress

1. Botpress: Botpress є відкритою платформою для розробки та управління чат-ботами. Вона надає можливості створення, навчання та розгортання ботів для різних каналів зв'язку, включаючи Telegram. Botpress

має вбудовані інструменти для налаштування ботів, аналітики та управління розгорнутими ботами.

Botpress є потужною платформою для розробки та управління чат-ботами з розширеними можливостями налаштування та аналітики. Він надає гнучкість і контроль над ботами, але може вимагати програмування та розгортання на власних серверах.

2. Chatfuel: Chatfuel є платформою для створення чат-ботів без програмування. Вона надає зручний інтерфейс для створення ботів та налаштування їх функціональності. Chatfuel підтримує інтеграцію з Telegram і надає можливість налаштування відповідей бота, управління підписниками та статистикою.

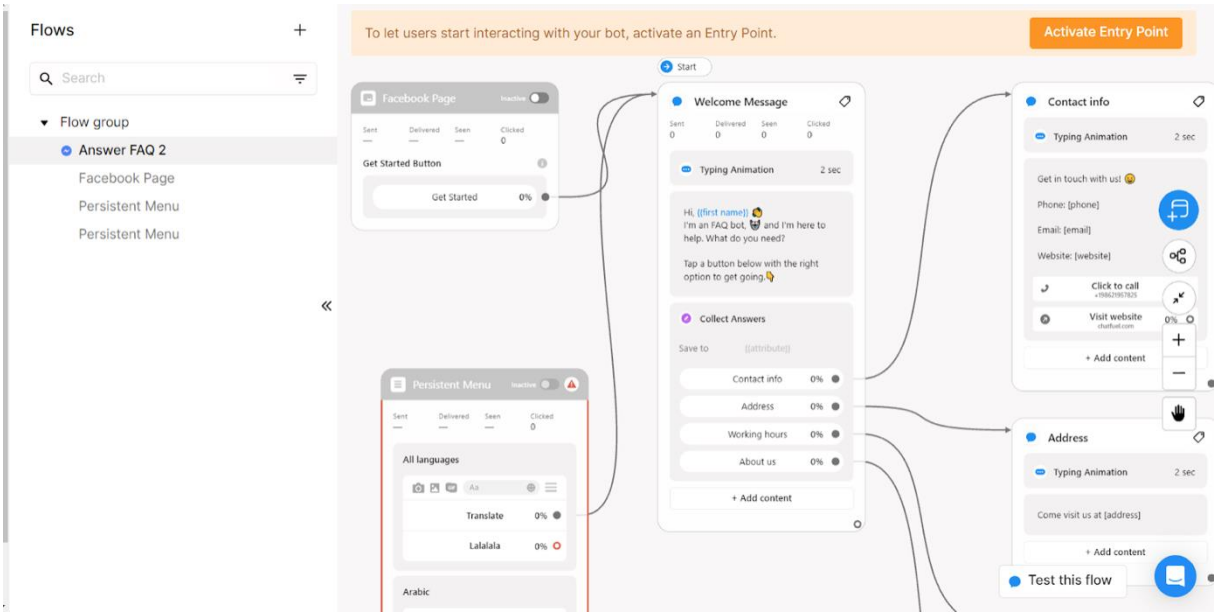

Рисунок 1.2 – Інтерфейс Chatfuel

Chatfuel підходить для швидкого створення простих чат-ботів без програмування. Він надає зручний інтерфейс та підтримку Telegram, але може бути обмежений у виборі функціональності та можливостей налаштування.

3. BotFather: BotFather є офіційним інструментом від Telegram для створення та налаштування ботів. Він надає зручний інтерфейс для створення нових ботів, надання їм унікальних ідентифікаторів та управління

налаштуваннями ботів. Однак, BotFather не надає розширених можливостей управління та аналітики, які можуть знадобитися в складніших проектах.

Рисунок 1.3 – Ідентифікатор BotFather

BotFather є простим інструментом для створення і базового налаштування Telegram-ботів, але може бути обмежений в розширених функціях та управлінні.

Залежно від потреб, можна обрати підхід, що найкраще відповідає вимогам щодо налаштування, аналітики та управління Telegram-ботами.

Висновки до розділу

Під час дослідження предметної області було проведено детальний аналіз існуючих алгоритмічних, технічних рішень та програмних продуктів, що допомогли глибше зрозуміти контекст та важливість проекту.

Результати аналізу підтверджують актуальність створення системи управління та налаштування telegram ботів. Зокрема, вони вказують на потенціал даного проекту для поліпшення комунікації та ефективності роботи в цій сфері. Наш аналіз дозволив визначити найкращі практики та можливості, які можуть бути успішно використані при реалізації даного проекту.

Отже, цей розділ є важливим підґрунтям для подальшої розробки та реалізації системи управління та налаштування telegram ботів.

2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

2.1 Загальні положення

Розділ формування вимог до системи є одним з ключових етапів в процесі розробки програмного продукту. У цьому розділі встановлюються вимоги, які система повинна виконувати для задоволення потреб користувачів і досягнення поставлених цілей проекту. Нижче наведені загальні положення, які слід враховувати при формуванні вимог до системи:

1. Аналіз потреб користувачів: Необхідно ретельно дослідити потреби і вимоги користувачів, їх очікування та вподобання. Це може включати в себе проведення спеціальних опитувань, інтерв'ю з потенційними користувачами, аналіз конкурентів та інші методи дослідження.

2. Визначення функціональних вимог: Функціональні вимоги описують, які функції повинна виконувати система. Це може включати функції взаємодії з користувачем, обробки даних, збереження і передачі інформації та інші операції, необхідні для досягнення мети системи.

3. Визначення нефункціональних вимог: Нефункціональні вимоги описують характеристики системи, які не пов'язані з конкретними функціями, але впливають на її ефективність, надійність, безпеку та інші аспекти. Це можуть бути вимоги до продуктивності, масштабованості, доступності, безпеки даних та інші аспекти.

Також можуть бути включені інші важливі аспекти, що відповідають конкретним потребам і особливостям проекту. Вимоги до системи повинні бути чіткими, специфічними, вимірюваними, досяжними та реалізованими в рамках виділених ресурсів та обмежень.

2.2 Аналіз потреб користувача

Визначення потреб користувача є важливим етапом цього проєкту, що дозволяє встановити вимоги до функціоналу системи та забезпечити максимальну задоволеність користувачів від її використання.

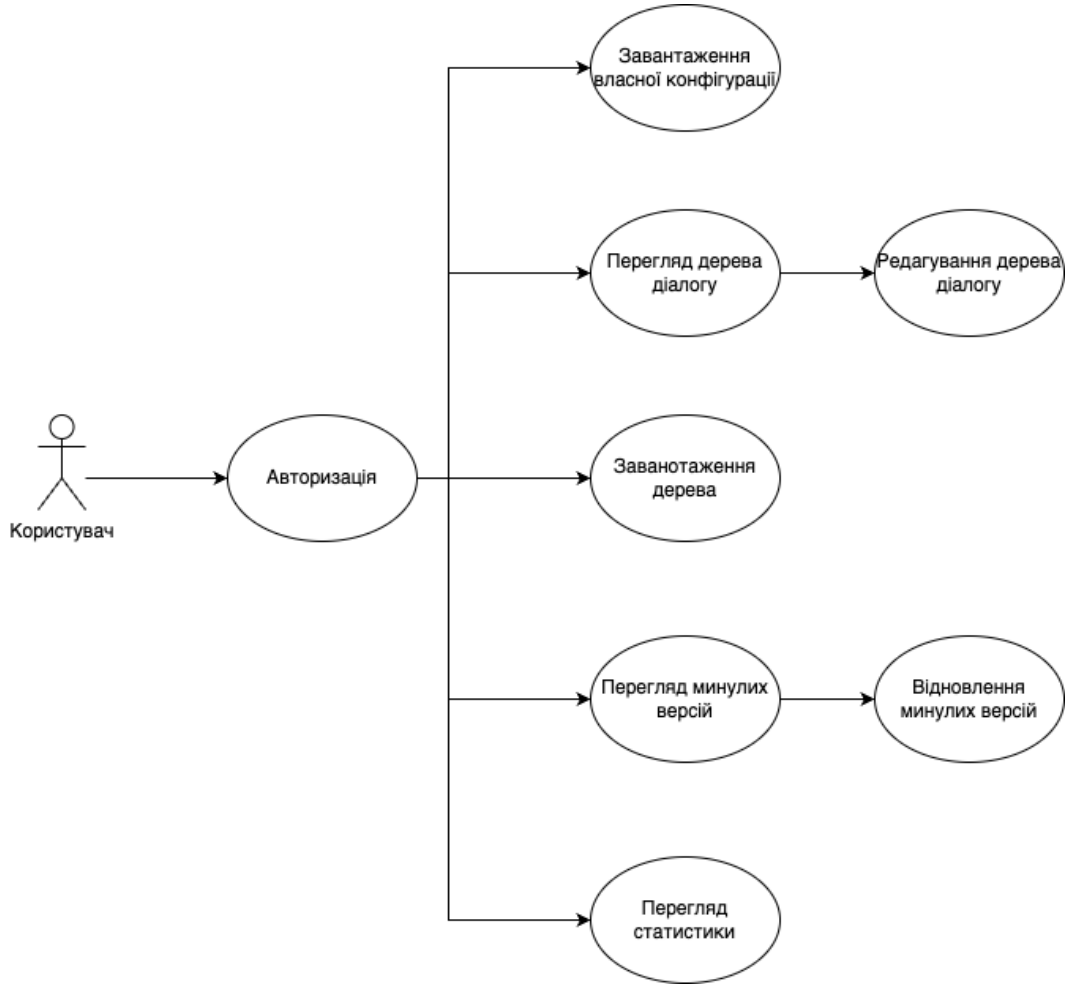

Рисунок 2.1 – Основні очікування користувача від системи

Таблиця 1 – Опис варіанту UC-01 використання додатку

Use case name	Завантаження власної json конфігурації
Use case ID	UC-01
Goals	Завантаження конфігурації на сервер, побудова на її основі мапи діалогу, відображення користувачу
Actors	Користувач

Trigger	Користувач натискає «Завантажити конфігурацію» після чого обирає JSONфайл
Pre-conditions	Користувач повинен бути авторизован у додадку
Flow of Events	Користувач натискає «Завантажити конфігурацію», після чого бачить форму завантаження JSON файлу і завантажує файл методом Drag-n-Drop або обирає самостійно
Extension	-
Post-Condition	Автоматично створюється мапа діалогу на основі структури

Таблиця 2 – Опис варіанту UC-02 використання додатку

Use case name	Авторізація в застосунку
Use case ID	UC-02
Goals	Авторизація користувача, збереження авторизаційного токена в Cookies
Actors	Користувач
Trigger	Користувач вперше зайшов у додаток
Pre-conditions	-
Flow of Events	Користувачить бачить форму авторизації, після чого вводить логін і пароль і тисне кнопку «Авторизуватись»
Extension	Якщо користувач не авторизується, він не зможе керувати ботами і налаштовувати мапи діалогів
Post-Condition	Користувач переходить на головну сторінку з деревом діалогу

Таблиця 3 – Опис варіанту UC-03 використання додатку

Use case name	Редагування дерева діалогу
Use case ID	UC-03
Goals	Створити власну конфігурацію діалогу з різними переходами
Actors	Користувач
Trigger	Користувач взаємодіє з деревом, використовуючи інтерактивні елементи
Pre-conditions	Користувач повинен бути авторизован у додатку і бути на головній сторінці додатку
Flow of Events	Користувач на головній сторінці натискає на прямокутний елемент мапи, після чого може або видалити, або відредагувати, або створити перехід на наступний елемент, після редагування необіхдно натиснути кнопку «Зберегти»
Extension	-
Post-Condition	Мапа діалогу відредагується

Таблиця 4 – Опис варіанту UC-04 використання додатку

Use case name	Відновлення минулих версій
Use case ID	UC-04
Goals	Переглянути та відновити минули версії мапи діалогу
Actors	Користувач
Trigger	Користувач переходить на сторінку «Резервні копії», після чого зліва обирає потрібну версію
Pre-conditions	Користувач повинен бути авторизован у додатку і бути сторінці резервних копій

Flow of Events	Користувач переходить на вкладку «Резервні копії», після чого злів обирає потрібну версію, потім користувач може переглянути попередню мапу діалогу і відновити її
Extension	-
Post-Condition	Мапа діалогу відновиться і на головній сторінці буде ця відновлена версія

Таблиця 5 – Опис варіанту UC-05 використання додатку

Use case name	Перегляд статистики бота
Use case ID	UC-05
Goals	Переглянути статистику бота
Actors	Користувач
Trigger	Користувач переходить на сторінку «Статистика»
Pre-conditions	Користувач повинен бути авторизован у додатку і бути сторінці «Статистика»
Flow of Events	Користувач переходить на сторінку статистики, після чого бачить статистику по цьому боту
Extension	-
Post-Condition	-

2.3 Функціональні вимоги

Функціональні вимоги до системи управління та налаштування telegram ботів визначаються на основі дослідження потреб користувачів ботів та забезпечують максимальну задоволеність користувачів від її використання.

Таблиця 6 – Функціональна вимога 1

Назва	Авторизація в додатку
Опис	Система надає можливість користувачу авторизуватись у додатку

Таблиця 7 – Функціональна вимога 2

Назва	Загрузка власної конфігурації
Опис	Система надає можливість завантажувати власну конфігурацію у JSONфайлі

Таблиця 8 – Функціональна вимога 3

Назва	Перегляд дерева діалогу
Опис	Система надає можливість користувачу переглянути дерево діалогу на головній сторінці з різним масштабуванням

Таблиця 9 – Функціональна вимога 4

Назва	Збероеження поточного стану
Опис	Система надає можливість користувачу безпечно зберегти поточну версію дерева

Таблиця 10 – Функціональна вимога 5

Назва	Редагування вузлів дерева
Опис	Система надає можливість користувачу видаляти/корегувати/робити потомків в кожному вузлі дерева

Таблиця 11 – Функціональна вимога 6

Назва	Перегляд минулих версій дерев діалогу
Опис	Система надає можливість користувачу переглянути минулі версії діалогу шляхом відображення мініатюрної версії дерева

Таблиця 12 – Функціональна вимога 7

Назва	Відновлення будь-якої минулої версії дерева
Опис	Система надає можливість користувачу відновити будь-яку минулу версію діалогу

Таблиця 13 – Функціональна вимога 8

Назва	Перегляд статистики
Опис	Система надає можливість користувачу переглянути статистику бота, тобто з якого ресурсу бот було знайдено

	A	B	C	D	E	F	G	H	I
1		FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8
2	UC-1	x							
3	UC-2		x						
4	UC-3			x	x	x			
5	UC-4						x	x	
6	UC-5								x

Рисунок 2.2 – Трасування вимог

2.4 Нефункціональні вимоги

Нефункціональні вимоги - це вимоги до системи, які не стосуються її функціональності, але визначають її характеристики, які необхідні для забезпечення оптимальної роботи та задоволеності користувачів. Визначення нефункціональних вимог є важливим етапом розробки системи, оскільки дозволяє забезпечити максимальну задоволеність користувачів від її використання та гарантувати якість роботи системи.

Основні нефункціональні вимогу до проекту:

- Високий рівень захисту від хакерських атак повинен бути забезпечений за допомогою сучасних технологій та заходів безпеки
- Простий та інтуїтивно зрозумілий інтерфейс повинен бути розроблений з урахуванням потреб користувачів та забезпечувати зручний доступ до всіх функцій

- Єдине оформлення на всіх сторінках додатку повинно бути використане для створення єдиної корпоративної ідентичності та поліпшення візуальної зручності користувачів
- UX повинен бути на високому рівні, задля легкого доступу до додатку людям з порушенням зору. Для цього, ми повинні забезпечити підтримку великих шрифтів, контрастних кольорів та інших функцій, що полегшують використання додатку для людей з особливими потребами.

Висновки до розділу

У даному розділі було розглянуто процес формування вимог до системи. Основна мета формування вимог - забезпечення високої якості та відповідності системи потребам користувачів. Для досягнення цієї мети, необхідно враховувати потреби користувачів, особливості предметної області, технічні можливості та обмеження.

Основні кроки у формуванні вимог включають збір і аналіз потреб користувачів, визначення функціональних та нефункціональних вимог, встановлення обмежень та визначення пріоритетів.

Урахування вимог до системи забезпечує раціональне планування та розробку програмного продукту, а також сприяє досягненню поставлених цілей та задоволенню потреб користувачів. Тому, формування вимог є ключовим етапом в процесі розробки системи і потребує уважного аналізу, документування та розгляду різних варіантів вимог.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

3.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура в проекті системи управління та налаштування Telegram-ботів передбачає розділення функціональності та обов'язків між клієнтською та серверною частинами системи. Нижче наведено детальний опис кожної з цих компонент:

1. Клієнтська частина:

- Клієнтська частина виконується на боці користувача і зазвичай представлена у вигляді веб-додатку або мобільного додатку.
- Використовуючи Vue.js (або іншу фронтенд-технологію), клієнтська частина відповідає за інтерфейс користувача і забезпечує взаємодію з користувачем.
- Клієнтська частина відправляє запити до серверної частини для отримання та оновлення даних ботів та їх налаштувань.

2. Серверна частина:

- Серверна частина виконується на веб-сервері та обробляє запити від клієнтської частини.
- Використовуючи Node.js (або іншу серверну технологію), серверна частина відповідає за обробку запитів, взаємодію з базою даних та керування логікою бізнес-логіки.
- Серверна частина здійснює комунікацію з Telegram API для обміну даними з ботами та забезпечує належне налаштування та управління ними.
- Крім того, серверна частина може включати модулі для автентифікації користувачів, обробки даних та логування подій.

3. Взаємодія між клієнтською та серверною частинами:

- Взаємодія між клієнтською та серверною частинами здійснюється за допомогою комунікаційного протоколу, такого як HTTP або WebSocket.

- Клієнтська частина відправляє запити до сервера, передаючи необхідні дані, такі як запити на отримання даних ботів, створення нових ботів або оновлення налаштувань.
- Серверна частина обробляє запити, виконує необхідні операції (наприклад, доступ до бази даних) та повертає відповідь клієнту.
- Дані, що передаються між клієнтом та сервером, можуть бути у форматі JSON або іншому форматі, який обумовлюється у специфікації проекту.
- Клієнт-серверна архітектура дозволяє розділити відповідальності між клієнтською та серверною частинами системи, що спрощує розробку, розгортання та підтримку проекту. Клієнтська частина відповідає за інтерфейс та взаємодію з користувачем, тоді як серверна частина забезпечує обробку запитів, доступ до даних та інші бізнес-логіку, пов'язану з управлінням та налаштуванням Telegram-ботів.

3.1.1 Архітектура серверної частини MVC

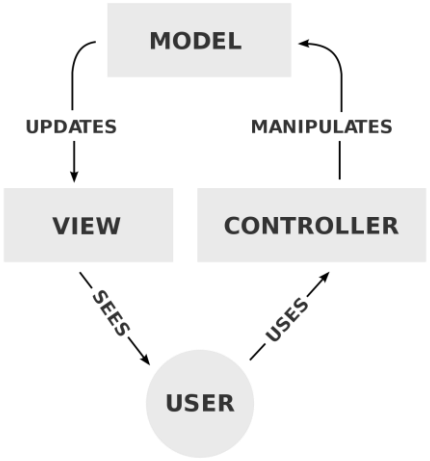

Рисунок 3.1 – Схема роботи архітектури MVC

MVC (Model-View-Controller) є популярним архітектурним шаблоном, який широко використовується для розробки програмного забезпечення, включаючи веб-додатки. Цей шаблон розділяє компоненти системи на три

основні частини: модель, представлення і контролер. Розглянемо кожну з них детально:

1. Модель (Model):

Модель відповідає за обробку даних, логіку бізнес-процесів та взаємодію з базою даних.

Вона представляє собою абстракцію даних, зберігаючи і управляючи їх станом.

Модель зазвичай містить методи для отримання, збереження, оновлення та видалення даних, а також для виконання розрахунків та інших операцій, пов'язаних з бізнес-логікою.

2. Представлення (View):

Представлення відповідає за відображення даних користувачу і обробку його взаємодії.

- Воно відображає дані з моделі у відповідному форматі для користувача (наприклад, HTML-сторінки або візуальні елементи інтерфейсу).
- Представлення також може містити логіку для валідації та форматування даних, а також для обробки користувацького вводу.

3. Контролер (Controller):

- Контролер відповідає за обробку запитів користувача та керування взаємодією між моделлю та представленням.
- Він приймає запити від користувача, виконує відповідні дії, оновлює модель та передає необхідні дані до представлення для відображення.
- Контролер також може містити логіку для маршрутизації, аутентифікації, авторизації та інших аспектів управління взаємодією з користувачем.

Принцип роботи в MVC полягає в тому, що користувач взаємодіє з представленням, відправляючи запити до контролера. Контролер обробляє запити, взаємодіє з моделлю для отримання необхідних даних або виконання операцій, а потім оновлює представлення з оновленими даними. При зміні

стану даних моделі, вона може повідомити представлення про необхідні зміни для оновлення відображення.

MVC забезпечує розділення відповідальностей між компонентами системи, сприяє розширюваності, підтримці кодування та тестуванні. Використання цього шаблону у проекті дозволяє легко керувати взаємодією з клієнтами, бізнес-логікою та збереженням даних.

3.1.2 Компонентна архітектура клієнтської частини

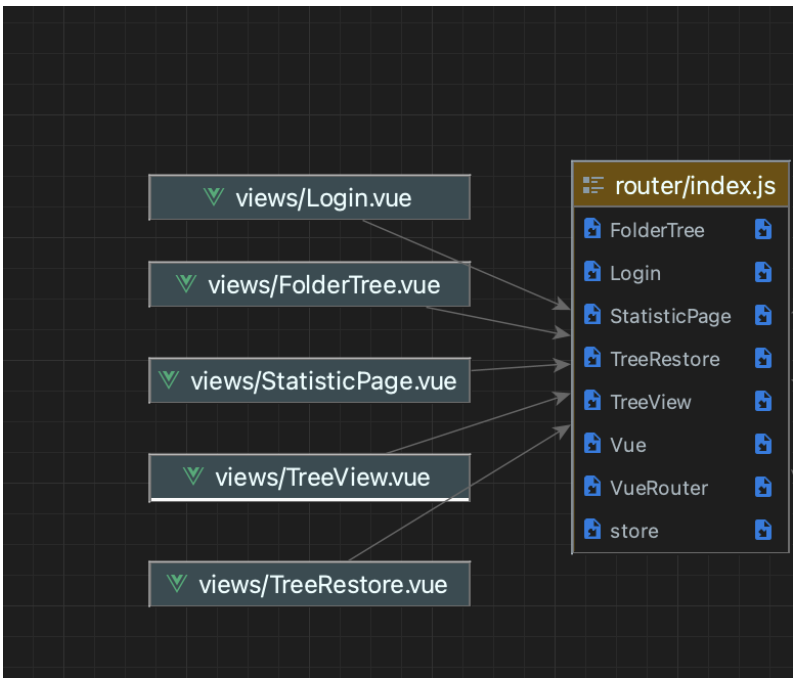

Рисунок 3.2 – Компонентна архітектура проєкту

Компонентна архітектура на клієнтській частині є підхідом до розробки програмного забезпечення, де функціональність поділяється на незалежні компоненти. Кожен компонент виконує конкретну функцію і може бути повторно використаний в різних частинах програми.

- Компоненти:

Компоненти є основними будівельними блоками програми. Вони мають внутрішню логіку, стиль оформлення і можуть містити вкладені компоненти. Кожен компонент повинен мати чітко визначену відповідальність і виконувати

обмежений набір функцій. Компоненти можуть бути розміщені в окремих файлових структурах, що полегшує управління та розробку.

Кожен компонент може мати свої властивості, які можна передавати в компоненти як аргументи. Ці властивості називаються пропсами. Вони можуть бути передані в компоненти через різні способи, наприклад, як атрибути HTML-елементу або як параметри функції. Компоненти можуть використовувати пропси для збереження та передачі даних між компонентами.

- Комунікація між компонентами:

Компоненти можуть спілкуватися між собою через передачу даних і подій. Компоненти можуть відправляти події, які інші компоненти можуть підписатися і відповідно реагувати на них. Дані можуть бути передані від батьківського компонента до дочірніх компонентів через пропси або контекст.

Контекст - це механізм передачі даних вниз по дереву компонентів, який дозволяє уникнути передачі даних крізь кілька рівнів дочірніх компонентів. Контекст може бути використаний для передачі глобальних даних, таких як інформація про користувача або тема оформлення.

- Стан компонентів:

Компоненти можуть мати свій внутрішній стан, який може змінюватися протягом їх життєвого циклу або від реакції на взаємодію користувача. Збереження стану може бути реалізоване через локальний стан компонента або за допомогою глобальних станових управлінців, таких як Redux або Vuex.

Локальний стан - це стан, який зберігається безпосередньо в компоненті. Глобальні станові управлінці дозволяють зберігати стан за межами компонентів і керувати ним з одного місця. Це зменшує кількість пропсів, що потрібно передавати між компонентами, і полегшує управління станом.

- Маршутізація:

Для перехоплення і обробки URL-адрес та навігації між сторінками використовуються маршрутизатори. Маршрутизатори дозволяють визначати, які компоненти повинні бути відображені на основі URL-адреси.

Vue Router - це одна з найбільш популярних бібліотек маршрутизації для Vue. Вона дозволяє визначати маршрути, які відповідають URL-адресам, і відображати компоненти, які пов'язані з цими маршрутами.

- Стилізація:

Компоненти можуть мати свої властивості стилізації, які визначають їх зовнішній вигляд. Стилі можуть бути вбудовані в компоненти, використовуючи CSS або препроцесори, або ж можуть бути оформлені за допомогою компонентних бібліотек стилів, таких як CSS Modules або Styled Components.

CSS-модулі дозволяють визначати стилі, які застосовуються тільки до конкретного компонента, і уникнути конфлікту стилів між компонентами. Styled Components - це бібліотека, яка дозволяє визначати стилі прямо в компонентах, використовуючи CSS-синтаксис.

Компонентна архітектура на клієнтській частині забезпечує модульність, перевикористання коду, легкість управління і підтримку кодування. Вона дозволяє розробникам ефективно організовувати і структурувати клієнтську частину проекту, полегшує підтримку, розширення і тестування програмного забезпечення.

3.1.3 Мова програмування JavaScript

JavaScript є високорівневою, інтерпретованою мовою програмування, яка використовується для розробки веб-додатків і додає динамічність та інтерактивність на веб-сторінки. Основні характеристики цієї мови

1. Синтаксис:

- JavaScript має синтаксис, подібний до інших мов програмування, таких як Java або C++. Він використовує функції, змінні, оператори, цикли та умовні конструкції для керування виконанням програми.

2. Типи даних:

- JavaScript є динамічно типізованою мовою, що означає, що типи даних не визначаються заздалегідь. Змінні можуть містити різні типи даних, такі як рядки, числа, булеві значення, об'єкти та масиви.

3. Об'єктно-орієнтований підхід:

- JavaScript підтримує об'єктно-орієнтований підхід до програмування. Об'єкти в JavaScript можуть мати властивості і методи, і можуть бути використані для організації коду в логічні блоки.

4. Функціональне програмування:

- JavaScript також підтримує функціональне програмування, де функції можуть бути розглянуті як значення і передаватися як аргументи або повертатися як результати інших функцій.

5. Взаємодія з HTML та CSS:

- JavaScript може взаємодіяти з HTML-сторінками та CSS стилями. Він може змінювати вміст сторінок, обробляти події користувача, маніпулювати DOM-деревом і змінювати стилі сторінок.

6. Розширення можливостей:

- За допомогою JavaScript можна використовувати різноманітні бібліотеки та фреймворки, такі як React, Angular або Vue.js, для розробки веб-додатків. Ці інструменти спрощують роботу з JavaScript, надають зручні API для роботи з DOM, управління станом та будують модульну архітектуру.

7. Виконання на клієнті та на сервері:

- JavaScript використовується як мова програмування для веб-браузерів, де він виконується на стороні клієнта. Крім того, JavaScript може бути використаний на серверній стороні за допомогою платформи Node.js, що дозволяє розробляти серверні додатки та API.

JavaScript є широко використовуваною мовою програмування в веб-розробці. Його динамічність, гнучкість та широкі можливості дозволяють розробникам створювати інтерактивні та потужні веб-додатки з великою кількістю функцій та можливостей.

Висновки до розділу

В цьому розділі були обрані певні технології і архітектурні підходи для реалізації проєкту "Система управління та налаштування telegram ботів". Вибір цих технологій і підходів здійснювався з урахуванням конкретних вимог проекту і потреб користувачів.

На клієнтській стороні була обрана компонентна архітектура, яка дозволяє розбити функціональність на незалежні компоненти, що спрощує розробку, тестування і підтримку коду. Цей підхід дозволить надати гнучкість і масштабованість системи, а також полегшить зміну та розширення функціональності в майбутньому.

На серверній стороні було обрано архітектурний паттерн MVC (Model-View-Controller). Цей паттерн дозволяє розділити логіку додатку на модель, яка відповідає за збереження та обробку даних, представлення, яке відповідає за відображення даних користувачу, і контролер, який взаємодіє з моделлю та представленням. MVC паттерн сприяє структуруванню додатку, покращує його підтримку та розширення, а також дозволяє розділити ролі розробників у команді.

Мова програмування JavaScript була обрана для реалізації як на клієнтській, так і на серверній стороні. JavaScript є потужною мовою, яка має велику спільноту розробників, багатий екосистему засобів і бібліотек, що дозволяє швидко розробляти функціональний і інтерактивний інтерфейс користувача, а також забезпечувати потрібні функціональні можливості на сервері.

Обраний набір технологій і підходів має свої переваги, такі як гнучкість, масштабованість, структурованість коду і багатий функціонал. Враховуючи специфіку проєкту, обрані технології і архітектура є відповідними і допоможуть забезпечити ефективну розробку та роботу системи.

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Структура системи

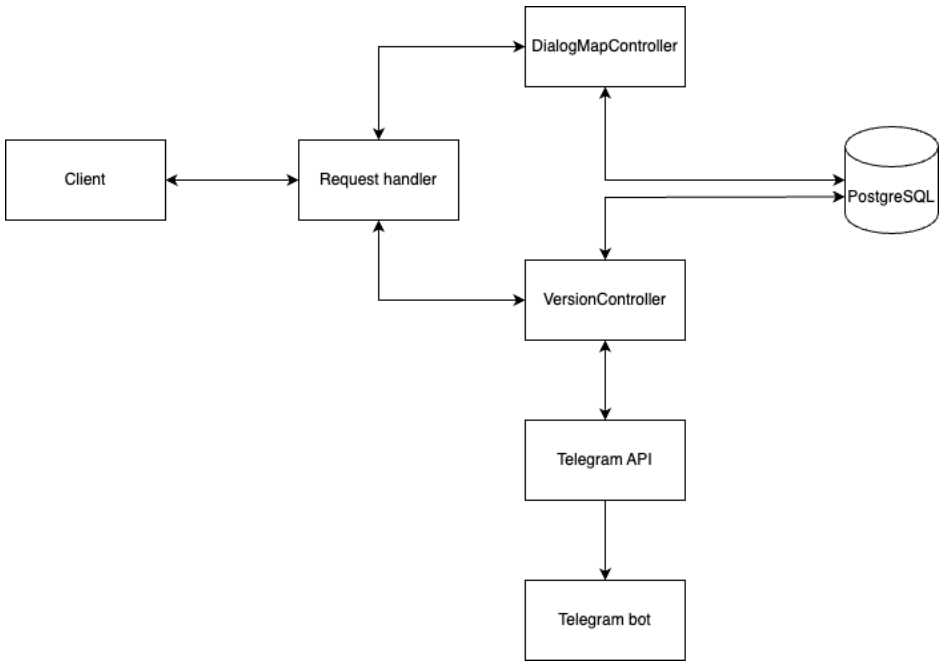

Рисунок 4.1 – Структура системи

Система складається з наступних елементів:

1. Client - це те, що бачить користувач коли заходить на сторінку, а також обробка валідації полей, відображення даних, надсилання запросів на сервер і малі обчислення

2. RequestHandler - відповідає за обробку вхідних HTTP-запитів, які надходять до серверної частини додатку. Його основна функція полягає в прийнятті запиту, обробці його даних та відповідному реагуванні на запит користувача. Також в даному додатку має в собі модуль авторизації

3. VersionController - є компонентом архітектури, відповідальним за керування версіями мапи діалогів. Він забезпечує можливість контролювати і управляти версіями різних діалогів, що дозволяє ефективно впроваджувати зміни, удосконалення та виправлення помилок у наявних деревах, визначає структуру діалогової мапи.

4. DialogMapController - обробляє метадані до дерева. Він допомагає визначити логіку та порядок взаємодії з користувачем або зовнішніми системами, а також керує потоком діалогу згідно з визначеною діалоговою мапою. Він контролює з`єднання з ботом, використовуючи токен доступу, та контролює поточну версію діалогу.

5. PostgreSQL – база даних. забезпечує структуроване зберігання даних у вигляді таблиць з різними типами даних. Це дозволяє ефективно зберігати інформацію, структурувати її за сутностями та встановлювати відношення між ними. Надає механізми для створення, оновлення, видалення та пошуку даних в базі даних. За допомогою мови запитів SQL, можна виконувати різноманітні операції над даними, такі як вибірка, фільтрація, сортування та об'єднання. Такод дозволяє виконувати складні запити до бази даних, використовуючи різні види операцій, такі як об'єднання, підзапити, групування та агрегації. Це дає можливість отримувати потрібну інформацію з бази даних шляхом складання складних запитів.

6. TelegramAPIservice - забезпечує взаємодію додатку або сервісу з TelegramMessenger за допомогою TelegramAPI. Він надає можливість використовувати функціональність Telegram, таку як надсилання повідомлень, отримання оновлень, управління ботами та багато іншого.

7. Telegrambot – власне бот, який ми бажаємо налаштувати

4.2 Структура бази даних

База даних додатку має наступну структуру:

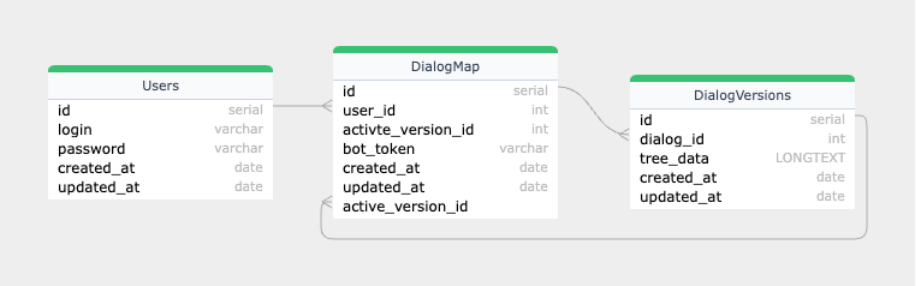

Рисунок 4.2 – структура бази даних

Таблиця 14 – Опис таблиці Users

Таблиця	Назва поля	Тип даних	Опис
Users	id	INTEGER	Ідентифікатор користувача
	login	VARCHAR(32)	Логін
	password	VARCHAR(32)	Пароль. Зашифровано
	updated_at	TIMESTAMP	Дата останнього оновлення даних користувача
	created_at	TIMESTAMP	Дата створення користувача

Ця таблиця зберігає інформацію про користувачів системи. Кожний користувач має унікальний ідентифікатор, логін та зашифрований пароль. Вона використовується для аутентифікації користувачів під час входу до системи.

Таблиця 15 – Опис таблиці DialogMap

Таблиця	Назва поля	Тип даних	Опис
DialogMap	id	INTEGER	Ідентифікатор мапи діалогу
	user_id	INTEGER	Зовнішній ключ, ідентифікатор користувача створившого мапу

	active_version_id	INTEGER	Зовнішній ключ, ідентифікатор активної версіїї
	bot_token	VARCHAR(255)	Токен бота. Зашифровано
	updated_at	TIMESTAMP	Дата останнього оновлення даних діалогу
	created_at	TIMESTAMP	Дата створення діалогу

Ця таблиця зберігає мапу діалог. Кожендіалогмаєунікальний ідентифікатор. Крім того, для кожного діалогу вказується ідентифікатор автора, який вказує на користувача, який створив цей діалог, а також токен бота, до якого прив`язан цей діалог.

Таблиця 16 – Опис таблиці DialogVersions

Таблиця	Назва поля	Тип даних	Опис
DialogVersions	id	INTEGER	Ідентифікатор версії діалогу
	dialog_id	INTEGER	Зовнішній ключ, ідентифікатор діалогу
	tree_data	LONGTEXT	JSON репрезентація дерева
	updated_at	TIMESTAMP	Дата останнього оновлення версії
	created_at	TIMESTAMP	Дата створення версії

Ця таблиця використовується для зберігання історії версій діалогів. Кожна версія має унікальний ідентифікатор і вказує на діалог, до якого вона належить за допомогою зовнішнього ключа. Також є JSONреалізація дерева. Додатково, для кожної версії вказується дата та час її створення.

4.3 Передача даних

Передача даних відбувається між клієнтською та серверною частинами за допомогою HTTP протоколу. Основний механізм передачі даних включає в себе наступні етапи:

1.Формування запиту: клієнтська частина (наприклад, веб-додаток, побудований з використанням Vue.js) формує HTTP запити для взаємодії з серверною частиною. Запит може містити метод (GET, POST, PUT, DELETE), URL-шлях, параметри запиту, заголовки тощо. Залежно від типу запиту, дані можуть бути включені у тілі запиту або як параметри запиту.

2.Відправка запиту: клієнтська частина відправляє сформований запит на сервер за допомогою HTTP протоколу. Запит передається через мережу з використанням веб-протоколу HTTP і доставляється до сервера.

3.Обробка запиту на сервері: серверна частина (заснована на Node.js) отримує запит і обробляє його. Запит передається до відповідного роутера або контролера, де виконуються потрібні дії, наприклад, обробка даних, звернення до бази даних тощо.

4.Обробка та маніпулювання даними: серверна частина взаємодіє з базою даних (PostgreSQL) для отримання або збереження даних. За необхідності, сервер може виконувати операції збору та маніпулювання даними, валідацію тощо.

5.Формування відповіді: серверна частина формує HTTP відповідь, яка містить дані, що будуть надіслані назад клієнтській частині. Відповідь може містити статусний код, заголовки та тіло відповіді з даними.

6.Прийом та обробка відповіді: клієнтська частина отримує HTTP відповідь від сервера і обробляє її. Залежно від потреби, дані можуть бути витягнуті з тіла відповіді та використані для оновлення відображення на клієнтському інтерфейсі.

Цей процес передачі даних використовується для забезпечення взаємодії між клієнтською та серверною частинами проекту. Він дозволяє передавати дані, виконувати операції збереження, отримання та маніпулювання даними, а також оновлювати клієнтський інтерфейс залежно від відповідей сервера.

4.3.1 HTTP протокол та модель OSI

HTTP (HypertextTransferProtocol) - це протокол передачі даних в мережі Інтернет. Він використовується для взаємодії між клієнтом (наприклад, веб-браузером) та сервером, забезпечуючи передачу гіпертекстових документів, зображень, відео та інших ресурсів.

HTTP базується на моделі OSI, яка визначає стандартизований набір протоколів для передачі даних між комп'ютерами у мережі. Модель OSI складається з сімох рівнів абстракції, кожен з яких виконує свої функції:

1. Фізичний рівень (Physical Layer):

Відповідає за фізичне підключення та передачу бітів по фізичному носію, такому як мережевий кабель або бездротовий сигнал.

2. Канальнийрівень (Data Link Layer):

Відповідає за передачу даних між прямо суміжніми вузлами мережі. Виконує функції контролю помилок, адресації та керування доступом до мережевого середовища.

3. Мережевий рівень (Network Layer):

Відповідає за маршрутизацію даних в мережі. Визначає шляхи доставки пакетів даних від відправника до отримувача.

4. Транспортний рівень (Transport Layer):

Забезпечує надійну передачу даних між кінцевими точками з використанням протоколів, таких як TCP (Transmission Control Protocol) або UDP (User Datagram Protocol).

5. Сеансовий рівень (Session Layer):

Встановлює, управляє та припиняє зв'язки між двома пристроями. Забезпечує механізми синхронізації та відновлення з'єднань.

6. Представлення (Presentation Layer):

Відповідає за перетворення даних у такий формат, що зрозумілий для програм. Виконує кодування, стиснення, шифрування та інші операції.

7. Застосування (Application Layer):

Найвищий рівень моделі OSI, який визначає протоколи, які використовуються для спілкування між програмами. HTTP є одним з протоколів на цьому рівні і відповідає за передачу гіпертекстових документів та інших ресурсів.

HTTP використовується для взаємодії між клієнтом та сервером шляхом відправки запитів (requests) з боку клієнта і отримання відповідей (responses) з боку сервера. Запити та відповіді містять заголовки (headers), які містять метадані про передачу даних, і тіло (body), яке містить фактичні дані.

Протокол HTTP використовує різні методи запитів, такі як GET, POST, PUT, DELETE, для виконання різних операцій над ресурсами на сервері. Запити можуть містити параметри, які передаються разом з URL або в тілі запиту.

HTTP також підтримує коди стану (status codes), які вказують на результат обробки запиту сервером, наприклад, 200 OK для успішного запиту або 404 Not Found, коли ресурс не знайдено.

Цей протокол є основою для багатьох веб-застосунків та послуг, і використовується для передачі даних, доступу до веб-сторінок, виконання запитів API та багатьох інших сценаріїв взаємодії в мережі Інтернет.

4.4 Архітектура безпеки системи

Архітектура безпеки в системі включає набір заходів та принципів, які спрямовані на захист системи від несанкціонованого доступу, збереження конфіденційності, цілісності та доступності даних. Основні аспекти архітектури безпеки включають:

4.4.1 Аутентифікація та авторизація

Система повинна мати механізми аутентифікації, які перевіряють ідентичність користувачів та їх права доступу до ресурсів системи. Авторизація визначає, до яких ресурсів або функціональності користувачі мають доступ на основі їх ролей та привілеїв.

Middleware - це шар або компонент програмного забезпечення, який знаходиться між додатком і операційною системою або між окремими компонентами додатку. Він виконується перед тим, як запит або відповідь досягає кінцевого пункту призначення, і дозволяє здійснювати обробку, перехоплення або зміну цього запиту або відповіді.

Після вдалої авторизації необхідно зберегти стан користувача, щоб в подальшому у нього не було необхідности вводити логін та пароль. Для цього ми будемо використовувати модуль JWT:

JSON Web Token (JWT) є відкритим стандартом для безпечної передачі інформації між сторонами у вигляді JSON об'єктів. JWT використовується для аутентифікації та авторизації користувачів в додатках і API. Давайте розглянемо основні поняття та принципи JWT:

1.Структура JWT: JWT складається з трьох основних частин - заголовка (Header), тіла (Payload) і підпису (Signature). Заголовок містить тип токену та алгоритм шифрування. Тіло містить корисну інформацію, таку як ідентифікатор користувача, ролі, час створення та час закінчення токену.

Підпис генерується на основі заголовка, тіла та секретного ключа, який відомий тільки серверу.

2.Аутентифікація за допомогою JWT: Після успішної аутентифікації користувача сервер генерує JWT і повертає його клієнту. Клієнт зберігає отриманий токен, наприклад, в локальному сховищі (наприклад, localStorage або sessionStorage) або в пам'яті браузера. При кожному наступному запиті до сервера, клієнт передає токен у заголовку або запиті. Сервер перевіряє цей токен, перевіряє його цілісність, дійсність та ролі користувача, і надає або відмовляє у доступі до запитуваного ресурсу.

3.Безпека JWT: JWT має вбудовану захист від подій зміни даних, оскільки підпис генерується на основі секретного ключа, який відомий тільки серверу. Також можна використовувати алгоритми шифрування для забезпечення конфіденційності даних в JWT. Однак, важливо зберігати секретний ключ на сервері в безпечному місці і не передавати його по мережі.

4.Переваги використання JWT: JWT є легким у використанні і передачі, оскільки він представлений у форматі JSON, що підтримується більшістю мов програмування. Він не потребує станових зберігаючих механізмів на сервері, оскільки всі необхідні дані містяться в самому токені. Крім того, JWT дозволяє реалізувати механізми одноразових токенів і зберігання історії токенів для більшої безпеки.

Загалом, JWT є потужним інструментом для реалізації аутентифікації в розподілених системах. Він надає зручний та безпечний спосіб передачі та перевірки даних користувача між клієнтом і сервером.

Маючи цілосне уявлення про механізм аутентифікації, можемо відобразити весь процес у наступній блок-схемі:

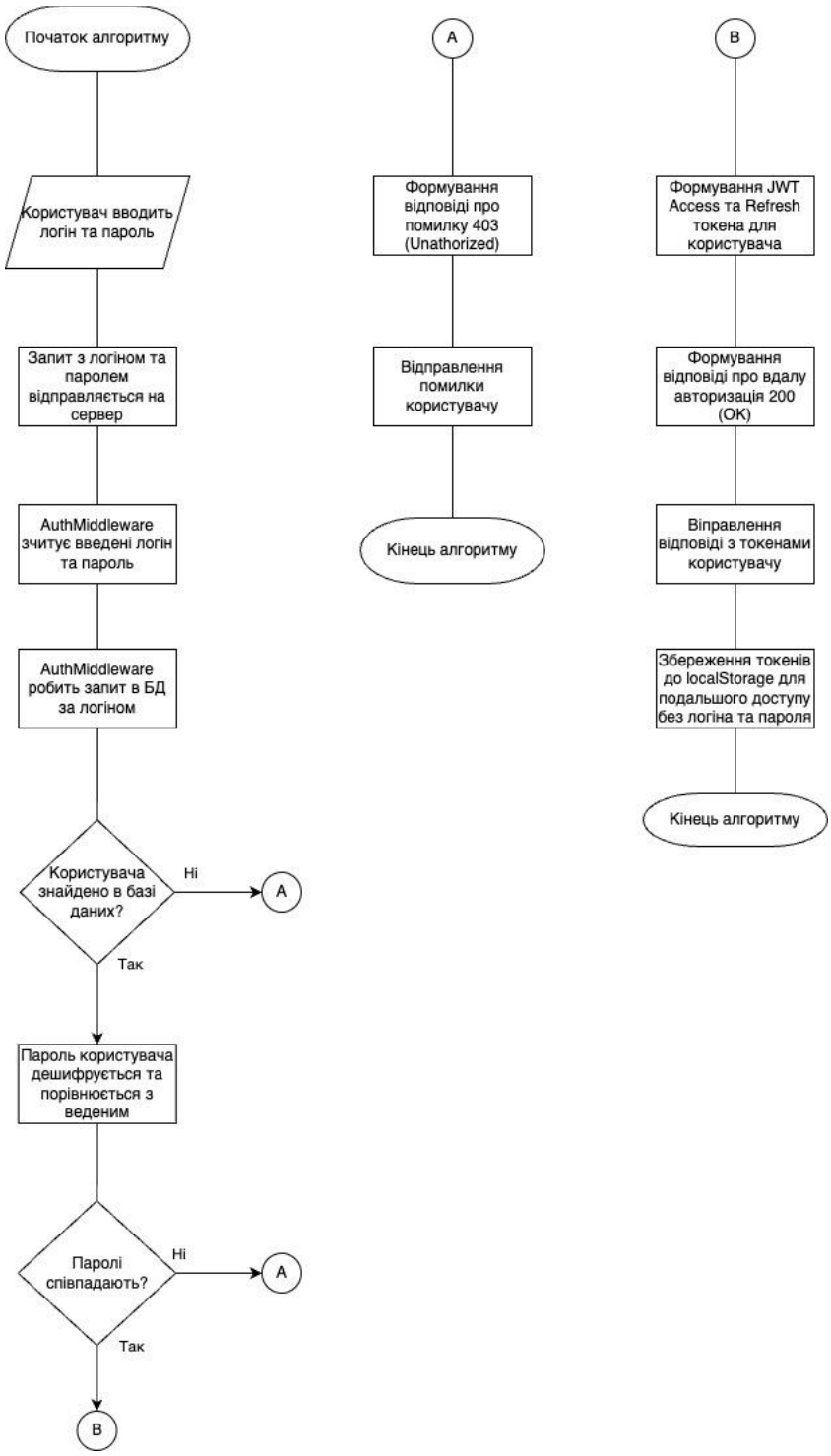

Рисунок 4.2 – Блок-схема алгоритму авторизації

4.4.2 Шифрування даних

Для забезпечення конфіденційності даних, особливо при їх передачі по мережі, система повинна використовувати шифрування. Шифрування дозволяє захистити дані від несанкціонованого доступу та перехоплення даних

в процесі використання додатком. Розгляенемо декілька основних технік шифрування даних:

1. Протокол HTTPS

Hypertext Transfer Protocol Secure (HTTPS) є протоколом забезпеченого обміну даними між клієнтом і сервером через мережу Інтернет. Він є захищеним варіантом звичайного HTTP і використовує шифрування для забезпечення конфіденційності та цілісності передачі даних.

Основна роль HTTPS полягає в захисті інформації, яка передається між клієнтом і сервером в процесі взаємодії. Для досягнення безпеки, HTTPS використовує шифрування даних за допомогою протоколу Secure Sockets Layer (SSL) або його наступника Transport Layer Security (TLS). Ці протоколи дозволяють створити захищену "тунельну" з'єднання між клієнтом і сервером, де всі дані шифруються перед передачею.

При використанні HTTPS, клієнт і сервер взаємодіють за допомогою шифрованих сесійних ключів, які забезпечують конфіденційність даних і захист від перехоплення. Шифрування даних забезпечує їх захист від несанкціонованого доступу і недозволених змін.

Крім шифрування, HTTPS також використовує сертифікати безпеки, що підтверджують ідентичність сервера. Це дозволяє клієнтам перевірити, що вони спілкуються з правильним сервером і уникнути атак типу "перехоплення посередника". Сертифікати видані довіреними організаціями, які підтверджують, що власник сервера є законним.

У випадку використання HTTPS, URL-адреси починаються з "https://" замість звичайного "http://". Це дозволяє користувачам розпізнавати, що з'єднання з сайтом є захищеним і їх дані будуть зашифровані.

HTTPS є важливою складовою для забезпечення безпеки передачі даних в мережі Інтернет. Він забезпечує конфіденційність та цілісність даних, а також підтверджує ідентичність сервера, забезпечуючи безпечну і надійну комунікацію між клієнтом і сервером.

Загалом схема роботи HTTPSмає наступний вигляд:

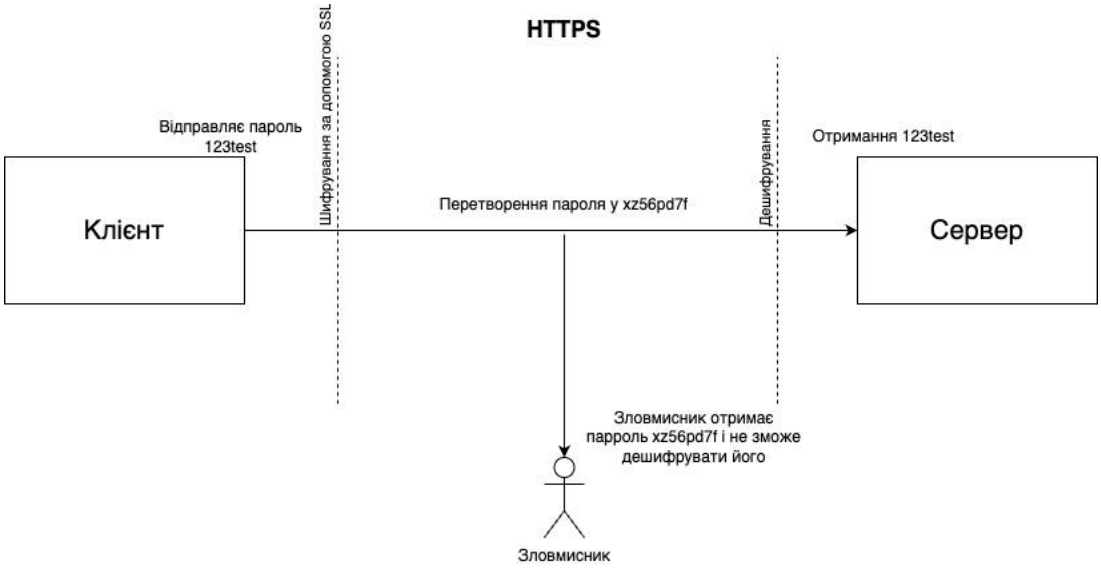

Рисунок 4.3 – Схема роботи HTTPS

Наш додаток ми будемо запускати у локальному середовищі, тому необхідності в HTTPSнема, бо дані не відправляються в глобальну мережу. Але при розгортанні додатку в хмарних середовищах, необхідно буде обрати SSL-сертифікат на сторінці середовища.

2. Шифрування даних перед збереженням їх у базу даних.

Шифрування даних перед збереженням у базі даних (БД) є важливою складовою для забезпечення безпеки та конфіденційності інформації. Воно дозволяє захистити дані від несанкціонованого доступу та забезпечити їх безпеку, навіть якщо БД потрапить у руки зловмисників.

Одним з найпоширеніших методів шифрування даних у БД є симетричне шифрування. При симетричному шифруванні використовується один і той же ключ для шифрування й дешифрування даних. Цей ключ повинен бути добре захищеним і доступним тільки авторизованим користувачам або програмам. Симетричне шифрування дозволяє швидко зашифрувати та розшифрувати дані, але ключ має бути обережно збережений, щоб уникнути його втрати або попадання в руки зловмисників.

Інший метод - асиметричне шифрування, використовує пару ключів: публічний ключ для шифрування і приватний ключ для розшифрування.

Публічний ключ може бути розповсюджений широко і доступний всім, хто бажає зашифрувати дані для відправки. Приватний ключ залишається конфіденційним і використовується тільки власником для розшифрування отриманих даних. Асиметричне шифрування є більш безпечним, оскільки приватний ключ важко підбирати, і навіть знаючи публічний ключ, зловмисник не зможе розшифрувати дані без приватного ключа.

Крім того, можна використовувати хеш-функції для шифрування даних перед збереженням у БД. Хеш-функція приймає вхідні дані і перетворює їх на унікальний хеш-код фіксованої довжини. Хеш-код не може бути зворотно перетворений на вихідні дані, тому його важко використовувати для відновлення оригінальних даних. Хеш-функції часто використовуються для збереження хешованих паролів, де під час аутентифікації використовується порівняння хеш-кодів.

При шифруванні даних перед збереженням у БД важливо вибрати надійний алгоритм шифрування та забезпечити безпечне збереження ключів. Крім того, необхідно використовувати правильні методи шифрування для конкретного типу даних, забезпечуючи належний рівень безпеки і конфіденційності.

Рисунок 4.4 – Схема роботи шифрування перед збереженням даних

4.4.3 Захист від атак

Система повинна мати комплекс заходів безпеки, які гарантують надійний захист від різноманітних атак, таких як кросс-сайтовий скриптінг (XSS), внедрення SQL-запитів, перехоплення сесій, введення зломаного коду та інших. Для цього можуть застосовуватися різноманітні технології та методики, такі як валідація вхідних даних, фільтрація та санітаризація введення користувачів, використання параметризованих запитів до бази даних, контроль доступу, шифрування та багато інших заходів безпеки на різних рівнях системи. Наприклад, можна застосувати технології, які дозволяють відстежувати та аналізувати поведінку користувачів, що дозволяє реагувати на потенційно небезпечні дії та запобігати можливим атакам. Також до заходів безпеки можуть входити регулярні аудити системи та виявлення вразливостей, а також розробка та імплементація політик безпеки, які забезпечать надійний захист вашої системи.

4.4.4 Моніторинг та журналювання

Система повинна мати механізми моніторингу безпеки, які виявляють незвичайну або підозрілу активність, включаючи спроби несанкціонованого доступу або атаки. Також важливим є журналювання подій та аудит, яке дозволяє відстежувати дії користувачів, зміни в системі та виявляти можливі проблеми безпеки.

Взагалом, провайдер хмарних рішень має вбудовані системи моніторингу, але якщо нам потрібна власна реалізація моніторінгу, треба розібратись в цьому питанні більш детально і підібрати потрібне рішення:

1.Логування: Один із способів моніторінгу - це логування подій та дій, які відбуваються в системі. Логи можуть містити інформацію про помилки, винятки, доступ до ресурсів та інші події, які є важливими для аналізу та

моніторінгу системи. Логи можна зберігати у відповідному форматі та аналізувати для виявлення проблем.

2.Метрики та моніторинг стану: Для вимірювання та моніторингу різних параметрів системи можна використовувати метрики. Метрики можуть включати кількість запитів, час відповіді, використання ресурсів, пам'ять, CPU і т.д. Ці дані можуть бути зібрані та візуалізовані за допомогою спеціалізованих інструментів для моніторингу, таких як Prometheus, Grafana, Elasticsearch і Kibana. Вони дозволяють відстежувати показники та виявляти аномалії для подальшого аналізу та оптимізації системи.

3.Автоматизовані сповіщення: При виникненні проблем або аномальних ситуацій в системі, можна налаштувати автоматичні сповіщення, які будуть надсилати повідомлення адміністраторам або відповідальним особам. Це дозволяє оперативно реагувати на проблеми та приймати необхідні заходи.

4.Аналіз журналів та метрик: Збір та аналіз журналів та метрик дозволяє виявляти тренди, патерни та аномалії в системі. Це може включати аналіз логів для виявлення проблемних запитів або помилок, аналіз метрик для виявлення навантаження на систему або визначення пікових навантажень.

5.Моніторинг зовнішніх служб: Якщо система взаємодіє з іншими зовнішніми сервісами або системами, важливо включити моніторинг цих залежностей. Це дозволяє виявляти проблеми з підключенням, відмови служб або зміни у їхній функціональності.

Ці рішення можуть бути реалізовані за допомогою спеціалізованих інструментів для моніторингу, які надають можливості збору, аналізу та візуалізації даних про систему. Вибір конкретного інструменту залежить від потреб проекту, його розмірів, складності та вимог до моніторингу.

Рисунок 4.5 – Інтерфейс системи моніторінгу Grafana

4.4.5 Резервне копіювання та відновлення

Система повинна мати механізми регулярного резервного копіювання даних та можливості їх відновлення в разі виникнення проблем, таких як втрата даних або пошкодження системи. Ці механізми допоможуть забезпечити безпеку вашої інформації та запобігти втратам даних в разі непередбачуваних обставин.У сучасному світі, провайдери хмарних рішень мають потужні системи резервного копіювання, які гарантують збереження важливої інформації у разі виникнення непередбачуваних ситуацій. Вони включають розподілені системи збереження версій та автоматичні системи копіювання, які можна підлаштувати під потреби проєкту. Наприклад, якщо ви плануєте зберігати величезний обсяг даних, вам можуть знадобитись системи збереження версій, які дозволять вам зберігати кілька копій ваших даних на різних серверах. Це забезпечить ще більшу надійність та безпеку вашої інформації.

4.4.6 Оновлення та патчі

Регулярні оновлення та застосування патчів до системи, програмного забезпечення та компонентів допомагають запобігати використанню вразливостей та забезпечувати високий рівень безпеки.

Висновки до розділу

У даному розділі розглянуто основні аспекти, пов'язані з розробкою інформаційної системи. Визначено структуру системи, включаючи клієнт-серверну архітектуру та компонентну модель на клієнтській частині.

Також розроблено схему базу даних, яка включає таблиці для збереження даних, необхідних для функціонування системи. Ця схема допомагає організувати структуроване зберігання даних та забезпечує їх доступність та цілісність.

У контексті передачі даних розглянуто HTTP протокол, який використовується для комунікації між клієнтом та сервером. HTTP забезпечує надійний обмін даними за допомогою запитів та відповідей. Також описано модель OSI, яка визначає сім рівнів комунікації в комп'ютерних мережах та допомагає забезпечити безперебійну передачу даних.

Архітектура безпеки системи розроблена з урахуванням основних принципів безпеки даних та захисту системи від несанкціонованого доступу.

Розробка інформаційної системи включає в себе різноманітні технології та рішення, що допомагають забезпечити функціональність, надійність та безпеку системи. Цей розділ виконує важливу роль у плануванні та реалізації проєкту, а також сприяє створенню ефективної та стабільної інформаційної системи.

5 РОЗГОРТАННЯ І ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗБЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання проекту - це процес підготовки та встановлення системи в середовищі виробництва. Етапи розгортання:

1. Підготовка середовища: Першим кроком є підготовка середовища, в якому буде розгортуватися проект. Це може бути фізичний сервер, хмарна інфраструктура або контейнеризоване середовище. Забезпечення необхідних ресурсів, налаштування мережі та доступу до системи - це важливі аспекти підготовки середовища. Для контейнерізації ми використовуємо docker. Docker контейнери забезпечують ізольоване середовище для додатків, що дозволяє уникати конфліктів залежностей та забезпечує стабільну роботу системи. Кожен контейнер має свій власний набір ресурсів, таких як пам'ять та обчислювальна потужність, що дозволяє керувати розподілом ресурсів та уникати впливу одного додатку на інші.

2. Встановлення залежностей: Для успішного розгортання проекту необхідно встановити всі необхідні залежності та середовища виконання. Це може включати пакети залежностей, базу даних, веб-сервери, сервери додатків та інші компоненти, необхідні для виконання проекту. Пакетний менеджер, що буде використаний у docker – npm.

3. Конфігурація: Наступним кроком є конфігурація проекту з урахуванням середовища розгортання. Це включає налаштування параметрів проекту, які залежать від специфіки середовища, таких як адреси бази даних, порти, налаштування безпеки тощо. Для цього ми використовуємо віртуальний сервер nginx

4. Завантаження коду: Код проекту повинен бути завантажений в середовище розгортання. Це може бути зроблено за допомогою систем контролю версій, пакетних менеджерів або інших механізмів. Код повинен бути розміщений у відповідних каталогах та налаштований для правильного виконання.

5. Налаштування бази даних: Необхідно створити базу даних, таблиці, індекси та інші необхідні елементи. Налаштування доступу до баз
и даних, встановлення прав доступу та забезпечення безпеки бази даних - це важливі аспекти розгортання.

6. Тестування та перевірка: Після завершення розгортання проекту важливо провести тестування та перевірку, щоб переконатися, що система працює належним чином. Це включає виконання тестів, перевірку функціональності, відповідності вимогам та виявлення можливих проблем.

5.1.1 Контейнеризація та віртуалізація

Контейнеризація та віртуалізація є двома розповсюдженими методами для розгортання та управління програмним забезпеченням. Вони забезпечують ізольоване середовище для виконання додатків, що дозволяє полегшити розгортання, масштабування та керування програмними системами.

1. Контейнеризація: Контейнеризація використовує контейнери для упаковки та виконання додатків з їхніми залежностями та середовищем. Контейнери ізолюють додаток та його компоненти від інших процесів та додатків, що дозволяє запускати додатки незалежно від конкретної операційної системи та середовища. Відомий контейнерний двигун Docker дозволяє створювати, розгортати та керувати контейнерами.

Контейнери дозволяють зберігати додатки та їхні залежності в одному пакеті, що робить розгортання та перенесення додатків між різними середовищами більш простим та надійним. Вони забезпечують стандартизований спосіб впакування, розгортання та управління додатками, що спрощує процес розробки та тестування.

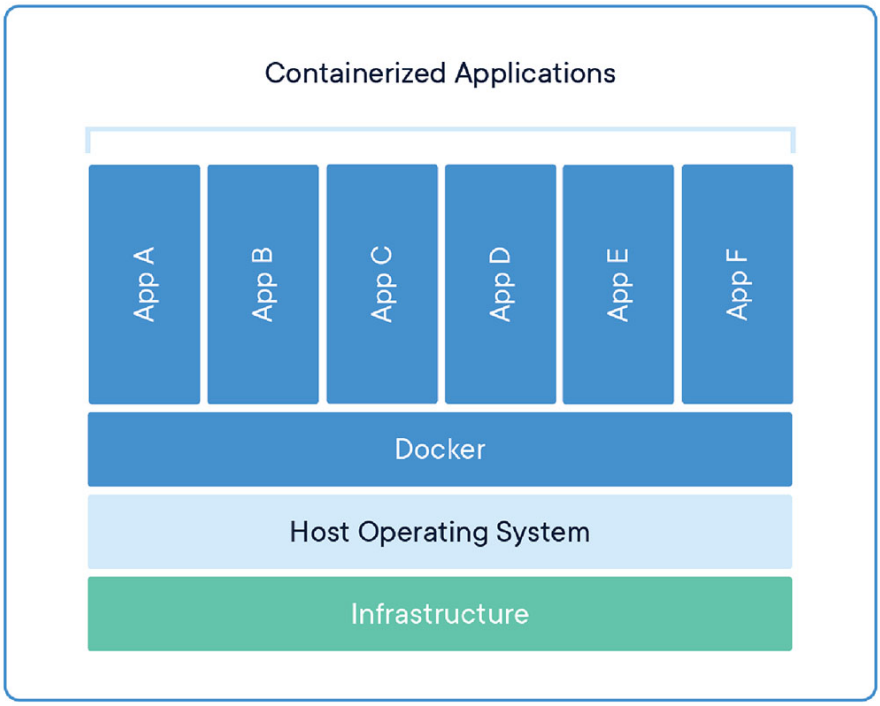

Рисунок 5.1 – Схема роботи контейнеризації

2. Віртуалізація: Віртуалізація передбачає створення віртуальних версій апаратного та програмного забезпечення, що дозволяє виконувати багато віртуальних машин (ВМ) на одному фізичному сервері. Кожна віртуальна машина має власну операційну систему та ресурси, які ізольовані від інших ВМ на сервері.

Віртуалізація дозволяє ефективно використовувати апаратні ресурси сервера, розділяючи їх між різними ВМ. Кожна ВМ може працювати незалежно, маючи свої власні налаштування та середовище. Це дозволяє гнучко керувати та масштабувати систему, забезпечуючи ізоляцію між різними додатками та сервісами.

Контейнеризація може бути використана для упакування та розгортання самої системи управління, а також окремих компонентів, які використовуються у процесі налаштування ботів. Віртуалізація може бути використана для розділення різних складових системи, таких як бази даних або сервери для керування ботами, забезпечуючи їхню ізоляцію та ефективне використання ресурсів сервера.

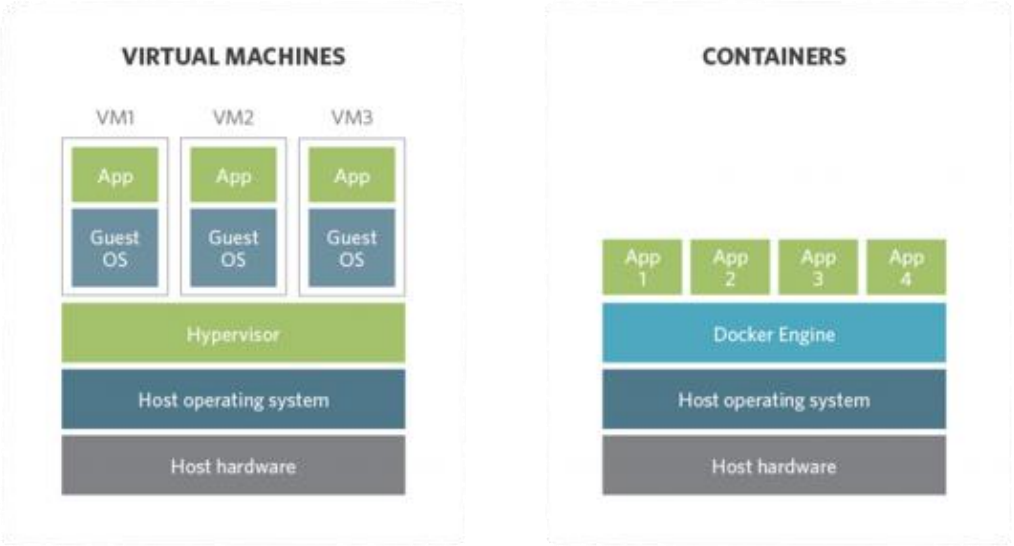

Рисунок 5.2 – Принципова різниця контейнеризації та віртуалізації

5.2 Тестування програмного забезпечення

5.2.1 Мануальне тестування програмного забезпечення

Мануальне тестування є процесом активної взаємодії з програмним продуктом з метою перевірки його функціональності, коректності та відповідності вимогам. Цей підхід включає ручну перевірку різних аспектів системи і вимагає від тестера ретельного аналізу вимог і активного взаємодії з програмним продуктом.

Під час мануального тестування, тестер виконує такі дії:

- Аналіз вимог: Тестер ретельно вивчає вимоги до системи, включаючи функціональні та нефункціональні вимоги. Це допомагає зрозуміти очікувану поведінку системи і встановити базові критерії для перевірки.

- Створення тестової документації: Тестер розробляє план тестування, сценарії тестування та іншу необхідну документацію для систематичного тестування програмного продукту.

- Виконання тестових сценаріїв: Тестер послідовно виконує задані тестові сценарії, спостерігає за реакцією програмного продукту і перевіряє його відповідність очікуваному результату.

- Виявлення та документування помилок: Якщо тестер виявляє некоректну реакцію або неправильну роботу програмного продукту, він ретельно документує помилку, вказує її опис, кроки для відтворення та іншу необхідну інформацію.

- Тестування на різних платформах та конфігураціях: Тестер перевіряє роботу програмного продукту на різних платформах, операційних системах та конфігураціях, щоб забезпечити його сумісність та стабільність.

- Звітність: Тестер готує звіти про виконані тести, виявлені помилки та результати перевірки. Ця інформація допомагає розробникам виправити помилки і поліпшити якість програмного продукту.

Мануальне тестування забезпечує можливість виявлення проблем і помилок, які можуть бути пропущені автоматичними тестами. Воно дозволяє отримати реальний досвід користувача і переконатися в правильній роботі програмного продукту перед його випуском.

Наведу приклад звіту з тестування цього проекту:

Таблиця 17 – Опис мануального тесту 1

Тест	Авторизація користувача
Модуль	Авторизація
Номер тесту	1.1
Початковий стан системи	Користувач вперше заходить на сторінку
Вхідні данні	Логін та пароль
Опис проведення тесту	Користувач вводить логін та пароль у необхідні поля, після чого натискає кнопку «Увійти»
Очікуваний результат	- У разі введення вірних даних: перехід на головну сторінку - У разі введення невірних: повідомлення про помилку

Фактичний результат	- У разі введення вірних даних: перехід на головну сторінку - У разі введення невірних: повідомлення про помилку

Таблиця 18 – Опис мануального тесту 2

Тест	Робота дерева діалогу
Модуль	Дерево діалогу
Номер тесту	2.1
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку
Вхідні данні	-
Опис проведення тесту	Користувач намагається мишкою переміщати дерево, змінювати масштаб
Очікуваний результат	Дерево працює коректно
Фактичний результат	Дерево працює коректно

Таблиця 19 – Опис мануального тесту 3

Тест	Редагування пустого вузлу
Модуль	Дерево діалогу
Номер тесту	2.2
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку
Вхідні данні	-

Опис проведення тесту	Користувач намагається змінити дані пустого вузлу, який буде відображено як тільки він зайде на сторінку і зберегти зміни, натиснувши на кнопку «Зберегти»
Очікуваний результат	Редагування вузла відбувається коректно, зміни збережено
Фактичний результат	Редагування вузла відбувається коректно, зміни збережено

Таблиця 20 – Опис мануального тесту 4

Тест	Додання нащадків
Модуль	Дерево діалогу
Номер тесту	2.3
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку
Вхідні данні	-
Опис проведення тесту	Користувач намагається створити нащадки до найпершого вузла і зберегти зміни, натиснувши на кнопку «Зберегти»
Очікуваний результат	Нащадки додані коректно, зміни збережено
Фактичний результат	Нащадки додані коректно, зміни збережено

Таблиця 21 – Опис мануального тесту 5

Тест	Видалення нащадків
Модуль	Дерево діалогу
Номер тесту	2.4
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку, створив нащадків

Вхідні данні	-
Опис проведення тесту	Користувач намагається видалити нащадків і зберегти зміни, натиснувши на кнопку «Зберегти»
Очікуваний результат	Нащадки видалені коректно, зміни збережено
Фактичний результат	Нащадки видалені коректно, зміни збережено

Таблиця 22 – Опис мануального тесту 6

Тест	Імпортування конфігурації
Модуль	Дерево діалогу
Номер тесту	2.5
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку
Вхідні данні	Користувач має JSONфайл з коректним деревом
Опис проведення тесту	Користувач намагається імпортувати конфігурацію, натиснувши кнопку «Імпортувати конфігурацію» і обрати файл
Очікуваний результат	- при правильній структурі файлу: дерево імпортується, початкове дерево заміщується - при неправильній структурі файлу: дерево не імпортується, виникає повідомлення про помилку
Фактичний результат	- при правильній структурі файлу: дерево імпортується, початкове дерево заміщується - при неправильній структурі файлу: дерево не імпортується, виникає повідомлення про помилку

Таблиця 23 – Опис мануального тесту 7

Тест	Завантаження дерева
Модуль	Дерево діалогу
Номер тесту	2.6
Початковий стан системи	Користувач авторизувався і зайшов на головну сторінку, також існує дерево з непустими вузлами
Вхідні данні	Дерево з непустими вузлами
Опис проведення тесту	Користувач натискає кнопку «Завантажити»
Очікуваний результат	Завантаження JSONфайлу з коректною структурою
Фактичний результат	Завантаження JSONфайлу з коректною структурою

Таблиця 24 – Опис мануального тесту 8

Тест	Перегляд версії
Модуль	Відновлення версій
Номер тесту	3.1
Початковий стан системи	Користувач авторизувався і зайшов на сторінку відновлення версій, також він повинен мати попередні збереженні версії дерев
Вхідні данні	Користувач має попередні збережені версії
Опис проведення тесту	Користувач намагається обрати потрібну версію, після чого бачить зменшену версію дерево і намагається повторити тест 2.1 з деревом
Очікуваний результат	Версії у вигляді дат відображаються, дерево у версії відображається і правильно реагує на взаємодію

Фактичний результат	Версії у вигляді дат відображаються, дерево у версії відображається і правильно реагує на взаємодію

Таблиця 25 – Опис мануального тесту 9

Тест	Відновлення версії
Модуль	Відновлення версій
Номер тесту	3.1
Початковий стан системи	Користувач авторизувався і зайшов на сторінку відновлення версій, також він повинен мати попередні збереженні версії дерев
Вхідні данні	Користувач має попередні збережені версії
Опис проведення тесту	Користувач намагається обрати потрібну версію, після чого натискає кнопку «Відновити дерево» і підтверджує відновлення
Очікуваний результат	Версія відновилась, користувача повернуто на головну сторінку
Фактичний результат	Версія відновилась, користувача повернуто на головну сторінку

5.2.2 Дослідження userflow

Тепер пройдемось по шляху, який буде проходити користувач, вирішивши скористатись нашим програмним забезпеченням. Вперше зайшовши на сторінку, він побаче вікно авторизації:

Рисунок 5.3 – Вікно авторизації

Припустимо, що у користувача вже існує акаунт до якого вже підключено бота. Він вводить правильні дані, після чого заходить на головну сторінку:

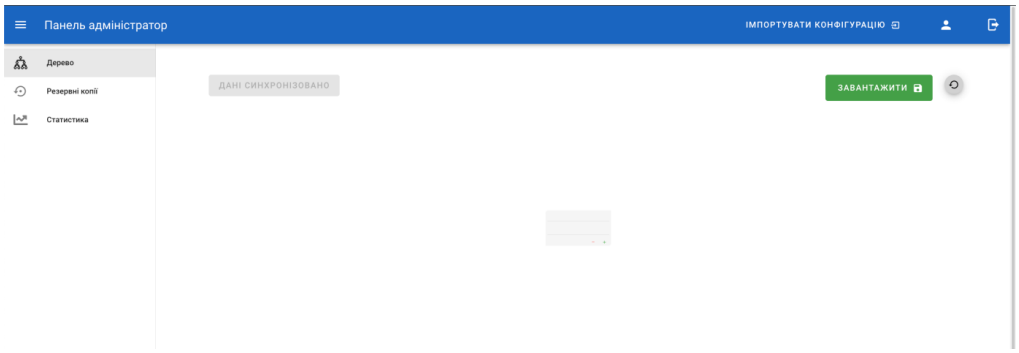

Рисунок 5.4 – Головне меню з пустим деревом

Він бачить пустий вузол, який він може почати редагувати:

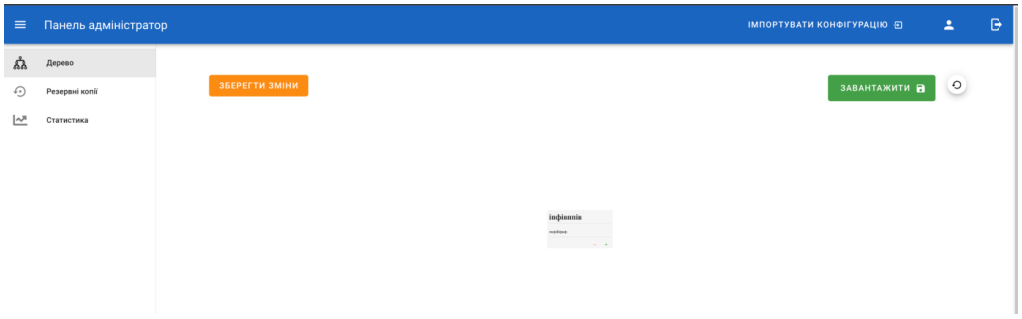

Рисунок 5.5 – Відредагований вузол

Як бачимо, після зміни вузла стала активна кнопка «Зберегти зміни». Вона необхідно для збереження змін на стороні сервера. Також користувач може добавити нащадків у дерево:

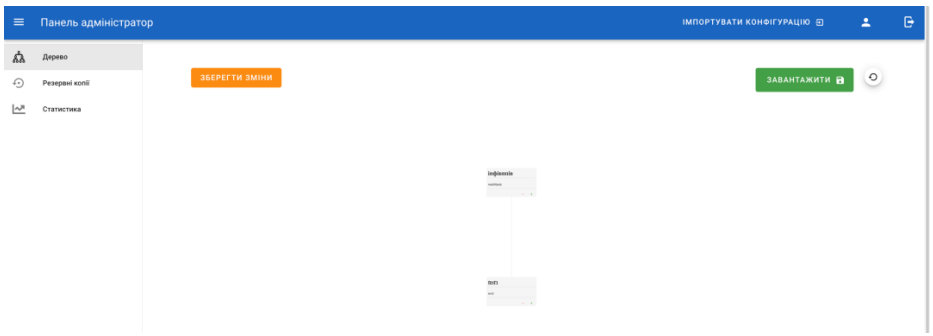

Рисунок 5.6 – Відображення нащадка

Після цього він може завантажити поточне дерево, натиснувши кнопку «Завантажити»:

Рисунок 5.7 – Контент завантаженого файлу

Після збереження файлу, користувач зможе зайти до розділу відновлення версій і там побачити список всіх доступних версій з датою збереження:

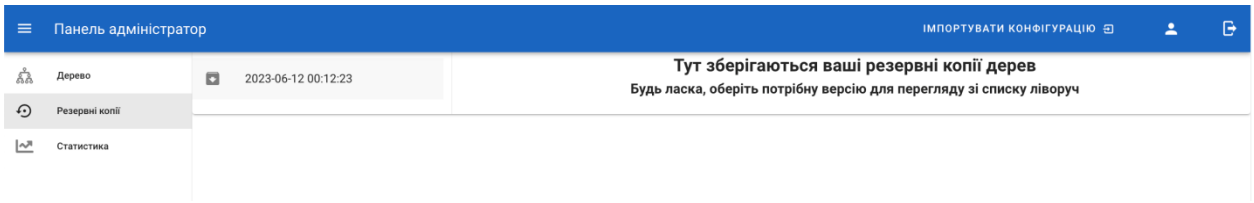

Рисунок 5.8 – Розділ відновлення версій

Після натискання на будь-яку версію, користувач зможе переглянути меншу версію дерева:

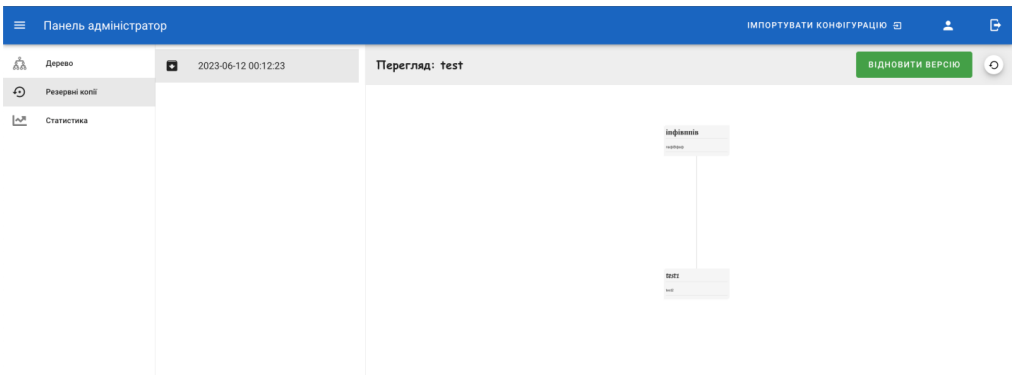

Рисунок 5.9 – Мініатюрна версія дерева

Перед відновленням, користувач повинен підтвердити свій намір

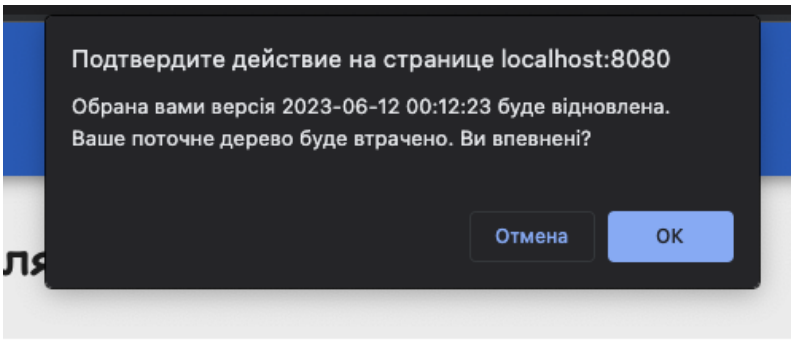

Рисунок 5.10 – Запит про намір відновлення

Висновки до розділу

Цей розділ ми описали два основних аспекта підтримки програмного забезпечення. Це розгортання та тестування додатку.

На першому етапі проведено необхідні дії щодо встановлення та налаштування програмного забезпечення на відповідних серверах або середовищах. Зокрема, проведено налаштування інфраструктури, встановлення необхідних залежностей, налаштування бази даних та інших компонентів системи. В результаті цих заходів забезпечено безперебійну роботу системи та готовність до проведення тестування.

Для перевірки коректності роботи системи проведено мануальні тести. Цей процес включав перевірку функціональності, взаємодії різних

компонентів, перевірку відповідності системи вимогам та очікуванням користувачів. Крім того, проведено введення тестових даних, виконання дій в системі та перевірку отриманих результатів. Завдяки цьому процесу виявлено та виправлено деякі помилки, що забезпечило поліпшення функціональності та якості роботи системи.

На останньому етапі проведено аналіз та опис шляхів, які користувач може пройти в системі, від початкової реєстрації або входу до виконання певних функцій та завершення роботи в системі. Перевірено зручність, логіку та шляхи взаємодії з користувачем. Цей процес допоміг забезпечити зручну та логічну роботу системи для користувачів.

Таким чином, під час розгортання та тестування програмного забезпечення здійснено ряд необхідних заходів, що дозволили забезпечити відповідність системи вимогам, поліпшити її функціональність, забезпечити зручну та безперебійну роботу системи для користувачів

ВИСНОВКИ

В процесі розробки інформаційної системи для управління та налаштування телеграм ботів ретельно розглянуто різні аспекти проекту. Основною метою проєкту є створення ефективної та надійної системи, яка надає можливість керувати та налаштовувати ботів в зручний та безпечний спосіб.

Структура системи побудовано на основі клієнт-серверної архітектури. На клієнтській стороні використовується фреймворк Vue.js, що дозволяє реалізувати користувацький інтерфейс і взаємодію з користувачами. Серверна частина реалізована на основі Node.js, що забезпечує обробку запитів та виконання бізнес-логіки.

Для зберігання та управління даними використовується PostgreSQL - потужна та надійна реляційна база даних. Модель бази даних розроблено з урахуванням вимог проекту та включає таблиці для користувачів, мап діалогу, версій.

Для передачі даних між клієнтом та сервером використовується HTTP протокол. Це стандартний протокол передачі даних в інтернеті, який забезпечує надійну та безпечну комунікацію між клієнтською та серверною частинами системи.

Архітектура безпеки розроблена з метою захисту конфіденційності, цілісності та доступності даних. Одним з ключових елементів безпеки є механізм аутентифікації за допомогою JWT (JSON Web Token). Цей механізм дозволяє перевірити автентичність користувача та забезпечити доступ до відповідних ресурсів системи.

Для забезпечення безпеки передачі даних використано протокол HTTPS. Цей протокол використовує шифрування за допомогою SSL/TLS протоколу для захисту даних під час їх передачі по мережі.

Розгортання проекту здійснюється за допомогою Docker, що дозволяє забезпечити однакове та незалежне виконання системи на різних середовищах.

Це забезпечує швидке та просте розгортання системи на нових серверах або у віртуальних середовищах.

Для забезпечення якості програмного забезпечення проводиться мануальне тестування, під час якого тестувачі вручну перевіряють роботу системи та виконують різноманітні сценарії взаємодії з системою.

Усі вищезазначені аспекти були ретельно розглянуті та реалізовані в проєкті з метою створення ефективної, безпечної та зручної системи для управління та налаштування telegram ботів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація Vue.js URL: https://vuejs.org/
2. Офіційна документація Node.js. URL: https://nodejs.org/
3. Офіційна документація PostgreSQL. URL: https://www.postgresql.org/
4. MDN Web Docs. URL: https://developer.mozilla.org/
5. StackOverflow. URL: https://stackoverflow.com/
6. Express.js офіційна документація. URL: https://expressjs.com/
7. JavascriptWebTokens офіційна документація. URL: https://jwt.io/
8. HTTPS офіційна документація. URL: https://tools.ietf.org/html/rfc2818
9. Docker офіційна документація. URL: https://docs.docker.com/
10. Selenium офіційна документація. URL: https://www.selenium.dev/documentation/
11. OWASP (Open Web Application Security Project). URL: https://owasp.org/
12. Рекомендації з безпеки веб-додатків. URL: https://owasp.org/www-project-web-security-testing-guide/
13. Проектування баз даних: концептуальне моделювання. URL: https://www.routledge.com/Conceptual-Data-Modeling-and-Database-Design-A-Fully-Algorithmic-Approach/Mancas/p/book/9781774635452
14. Блог TroyHunt - відомого експерта з кібербезпеки. URL: https://www.troyhunt.com/
15. Рекомендації з безпеки веб-додатків від Mozilla. URL: https://infosec.mozilla.org/guidelines/web_security
16. Рекомендації OWASP для захисту веб-додатків від атак XSS. URL:https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet