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

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *"Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій"*

Виконав: студент IV курсу, групи *ІП-61 Кушка Михайло Олександрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст.вик, Недашківський Є. А.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

доц. каф. ТК, к.т.н., Кисленко Ю. І.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

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

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
12.06.2020 15:46:32 EEST

Дата звіту:
12.06.2020 15:52:26 EEST

ID перевірки:
1004000091

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Kushka_ip61

ID файлу: 1004013089 Кількість сторінок: 40 Кількість слів: 7269 Кількість символів: 52224 Розмір файлу: 105.65 KB

1.76% Схожість

Найбільша схожість: 1.65% з джерело бібліотеки. ID файлу: 1000052312

1.62% Схожість з Інтернет джерелами	12	Page 42
1.76% Текстові збіги по Бібліотеці акаунту	36	Page 42

0.41% Цитат

Цитати	1	Page 43

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___” _____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Кущі Михайлу Олександровичу
(прізвище, ім’я, по батькові)

1. Тема проєкту « *Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій*»
керівник проєкту *Недашківський Євген Анатолійович, ст. вик*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с
2. Термін подання студентом проєкту «08» *червня 2020 року* **3. Вихідні дані до проєкту**
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення,
3) Аналіз якості та тестування програмного забезпечення
4) Впровадження та супровід програмного забезпечення
5) Керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна діяльності

3) Схема бази даних

4) Схема структурна бізнес процесу

5) Схема структурна послідовностей

6) Креслення вигляду екранних форм

7) Креслення вигляду звітних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*17.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*24.03.2020*	
3.	*Постановка та формалізація задачі*	*27.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*03.04.2020*	
5.	*Алгоритмізація задачі*	*10.04.2020*	
6.	*Моделювання програмного забезпечення*	*17.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*24.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*30.04.2020*	
9.	*Розробка програмного забезпечення*	*08.05.2020*	
10.	*Налагодження програми*	*15.05.2020*	
11.	*Виконання графічних документів*	*20.05.2020*	
12.	*Оформлення пояснювальної записки*	*27.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*09.06.2020*	
15.	*Подання ДП на основний захист*	*15.06.2020*	

Студент _____ Михайло КУШКА

(підпис)

Керівник _____ Євген НЕДАШКІВСЬКИЙ

(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6116.045490.02.81	Пояснювальна записка	71	
3	А4	КПІ.ІП-6116.045490.03.91	Технічне завдання	12	
4	А4	КПІ.ІП-6116.045490.04.51	Програма та методика тестування	4	
5	А4	КПІ.ІП-6116.045490.05.33	Керівництво програміста	3	
6	А4	КПІ.ІП-6116.045490.06.34	Керівництво користувача	1	
7	А4	КПІ.ІП-6116.045490.07.13	Опис програми	92	
8	А3	КПІ.ІП-6116.045490.08.СС	Схема структурна варіантів використань	1	
	А3	КПІ.ІП-6116.045490.09.СС	Схема структурна діяльності	1	
	А3	КПІ.ІП-6116.045490.10.КЕ	Креслення вигляду екранних форм	2	
	А3	КПІ.ІП-6116.045490.11.СС	Схема структурна послідовностей	1	
	А3	КПІ.ІП-6116.045490.12.СБД	Схема бази даних	1	
	А3	КПІ.ІП-6116.045490.13.КЗ	Креслення вигляду звітних форм	1	
	А3	КПІ.ІП-6116.045490.14.СС	Схема структурна бізнес процесу	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6116.045490.00.90		
Розробн.		Кушка М. О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Недашківський Є. А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка до дипломного проєкту: 203 с., 11 рис., 9 табл., 6 додатки, 22 джерела.

Метою даного проєкту було розробити мобільний додаток для платформи iOS у вигляді месенджеру, що призначений для консультантів у різних галузях та їх замовників, але також може використовуватися і просто для спілкування як звичайний месенджер. Оплата консультаційних послуг відбувається за допомогою мікротранзакцій у криптовалюті, завдяки чому забезпечується можливість користуватися сервісом без жодної довіри як до нього самого, так і до консультанта (замовника). Таким чином використання мікротранзакцій та блокчейн-технології дозволяє усунути проблему з довірою в сфері консультаційних послуг.

Для успішного використання цієї розробки достатньо пояснити користувачу загальні кроки для надання (отримання) консультацій за допомогою сервісу. Єдине, що потрібен знати користувач для користування сервісом – це як створити Ефіріум гаманець, поповнити його та переказувати кошти на іншу адресу.

КЛЮЧОВІ СЛОВА: БЛОКЧЕЙН-ТЕХНОЛОГІЯ, МІКРОТРАНЗАКЦІЯ, ЕФІРІУМ, КОНСУЛЬТАЦІЙНІ ПОСЛУГИ, МЕСЕНДЖЕР, IOS

ABSTRACT

An explanatory note to the dissertation: 203 page, 11 pictures, 9 tables, 6 appendixes, 22 resources.

The purpose of the project was to develop a messenger as a mobile application for the iOS platform. The application is intended for consultants from any field and their clients but can also be used for chatting as an ordinary messenger. The payment is conducted utilizing transactions in cryptocurrency by which a consultant and a client can use the service without any trust in it or each other. As a result, the usage of microtransactions and blockchain technology enables removing a problem with trust in the consultancy field.

In order to ensure that the application runs successfully, it is enough to explain to its users the basic principles of consultancy providing (getting) through the service. The only things that the users have to know before they run the application are how to create an Ethereum wallet, to replenish it, and to transfer the funds to any other address.

KEYWORDS: BLOCKCHAIN TECHNOLOGY, MICROTRANSACTION, ETHEREUM, CONSULTING, MESSENGER, IOS

Пояснювальна записка
до дипломного проєкту

на тему: *«Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій»*

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Блокчейн (блокчейн-технологія) – це набір блоків, що з'єднані між собою подібно до ланцюга, який містить інформацію та зростає з часом. Також відомий як розподілений реєстр 1) Blockchain technology: beyond bitcoin.

Смарт-контракт – будь-яка комп'ютерна програма, що виконується на блокчейні. Термін "смарт-контракт" визначено Сабо у 1994 році, як "комп'ютеризований протокол транзакцій, що виконує умови контракту" 2).

IoT (Internet Of Things, Інтернет речей) – система взаємопов'язаних електронних пристроїв, що під'єднані до інтернету та обмінюються інформацією один з одним без необхідності безпосереднього людського втручання.

Ефіріум – це платформа для розподілених обчислень з відкритим вихідним кодом та можливістю написання смарт-контрактів, що базується на блокчейні.

Майнер – особа, що бере участь у формуванні нових блоків у блокчейні, за що (зазвичай) отримує винагороду.

Мікротранзакція / мікроплатіж (у сфері блокчейну) – це транзакція на невелику суму (проте точний розмір залежить від сфери застосування; узагальнено можна вважати суму менше $20), дані якої криптографічно "підписується" ініціатором транзакції, проте не публікуються до блокчейну, а лише передається отримувачу.

Bitcoin Script – це мова програмування, що є Тюрінг-неповною, базується на принципі стеку та використовує мінімалістичні програми, що мають жорсткі обмеження.

ICO (Initial Coin Offering) – еквівалент у криптовалютній індустрії до Initial Public Offering (IPO). ICO є способом збору коштів, де компанія, що

збирається зібрати кошти для створення нового додатку, сервісу чи, фактично, будь-чого іншого запускає ICO. Зацікавлені інвестори можуть інвестувати в проєкт купуючи токени, що були випущені компанією. Ці токени можуть мати деяку прив'язку до продукту, що розробляється, але вони також можуть бути вкладом в компанію чи проєкт.

API (Application Programming Interface) – набір функцій, які знаходяться на одному сервері, але можуть бути викликані з різних додатків чи сервісів.

wei – найменша одиниця виміру в Ефіріум блокчейні, що дорівнює 1e-18 ETH.

KeyChain – система керування приватними даними та паролями, що була розроблена компанією Apple для своїх операційних систем.

див. – дивись.

Нода блокчейну – пристрій, що містить повну історію усіх транзакцій певного блокчейну та постійно синхронізується з мережею, додаючи нові транзакції, таким чином підтримуючи копію блокчейну актуальною.

"Сіль" (паролю) – невелика обрана випадковим чином послідовність байт, що при додаванні до паролю за допомогою звичайної конкатенації дозволяє уникнути дублювання хешу однакових паролів при зберіганні у базі даних. Тобто навіть якщо у двох користувачів буде однаковий пароль, то за допомогою "солі" їх хеш буде різний.

напр. – наприклад.

і т. д. – і так далі.

CRUD операція – CRUD розшифровується як Create Read Update Delete (створення, читання, оновлення, видалення) та означає базові 4 характеристики API, без яких його роботу вважати повноцінною не можна.

IDE – інтегрована середа розробки (Integrated Development Environment). Програма, що призначена для полегшення процесу написання коду та створення комп'ютерних програм.

укр. – українською.

англ. – англійською.

ВСТУП

На сьогоднішній день блокчейн-технологія та пов'язані з нею розробки, наприклад, смарт-контракти, набувають усе більшого розповсюдження. Основними перевагами цієї технології є надійність та незмінність . Завдяки цьому програми (смарт-контракти), що працюють на блокчейні завжди виконуються відповідно до свого програмного коду, який неможливо змінити після того, як відповідний застосунок було опубліковано до блокчейну. Більш того, блокчейн дозволяє зменшити сумніви, що могли б виникнути щодо роботи смарт-контракту, оскільки зазвичай його код публічно доступний усім бажаючим, а тому будь-хто може переконатися, що смарт-контракт не містить жодних прихованих властивостей та йому можна повністю довіряти. Загалом невід'ємними характеристиками блокчейну є прозорість, гнучкість, можливість перевірки та безпека 3).

Сфера консультаційних послуг онлайн, як і будь-яка інша, має свої проблеми. Наприклад, споживач ніяк не може бути повністю застрахований від поганої якості надання послуг, користувачі можуть мати низький рівень довіри до сервісу, завищені ціни на консультації через складність входу на ринок консультаційних послуг онлайн та створення адекватної конкуренції. Проте однією з найбільших серед цих проблем є недовіра людей один до одного: завжди є консультанти, які не є експертами у галузі, хоча за таких себе видають, або ж замовники, які прагнуть отримати консультацію, не заплативши за неї. Незважаючи на те, що блокчейн-технологія є новою (на момент створення проєкту їй лише трохи більше десяти років), вона може бути використана у сфері надання консультаційних послуг онлайн для розв'язання проблем, які в іншому випадку вирішити було б неможливо. Блокчейн може допомогти не лише у сфері консультаційних послуг онлайн, а й у багатьох інших, наприклад IoT 4), охорони здоров'я 5). На сьогоднішній день блокчейн

також активно застосовується до різноманітних фінансових сфер таких, як: бізнес-послуги, розміщення фінансових активів, передбачення цін на біржах та економічних операцій 6).

Для оплати послуг консультанта, замовник платить за кожне повідомлення від консультанта під час їхньої розмови. У Ефіріум блокчейні, на якому й була реалізована система, за кожну транзакцію потрібно платити комісію майнерам, та підтвердження транзакції займає певний час 7). Саме тому були використані мікротранзакції для оплати таких повідомлень, оскільки вони відбуваються фактично миттєво та є абсолютно безкоштовними для створення.

Першочерговою сферою застосування розробленого додатку є сфера консультаційних послуг. Причому, це може бути як навчання англійській мові онлайн, так і використання онлайн фітнес тренерами, що набуває все більшого розповсюдження останнім часом. Також розробка може бути використана для надання бізнес консультацій у різних сферах (будівництво, ІТ, юриспруденція та багато інших). Проте на цьому область застосувань не закінчується. Проєкт можна розширити для використання в будь-якій системі, де присутня особа, що надає інформацію онлайн, та особа, що зацікавлена в цій інформації. Проте використання в інших сферах вимагатиме переоцінку бізнес моделі, що означає додаткові ресурси: як часові, так і матеріальні.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Блокчейн-технологія вперше з'явилася трохи більше, ніж 10 років тому з моменту написання цієї роботи – у 2009 році. Це відбулося з появою криптовалюти Біткойн, що була розроблена людиною, або групою людей, що приховувалися за псевдонімом Сатоші Накомото 8).

З того часу з'явилося багато інших криптовалют, але з них хочу виділити одну – Ефіріум. Ця криптовалютна платформа була розроблена організацією "Ethereum Foundation" у 2015 році. З деталями й особливостями її застосування можна ознайомитись у документації до продукту – "Yellow Paper" 9). Її суттєвою відмінністю у порівнянні з Біткойн блокчейном є те, що на Ефіріум можна створювати смарт-контракти. Власне, не вдаючись в деталі, смарт-контрактами називають програми (застосунки), що виконуються у блокчейні, який захищає їх від будь-яких атак, спрямованих на зміну програмного коду, а тому такі програми завжди виконуються відповідно до того, як вони були написані 10). Саме завдяки можливості створення смарт-контрактів мною й було обрано Ефіріум як блокчейн платформу для створення програмного продукту.

Слід зазначити, що Біткойн блокчейн теж містить можливість написання смарт-контрактів, але вони пишуться на мові Script, що не є Тюрінг-повною мовою програмування. Це означає, що така мова не містить усіх базових елементів, щоб нею можна було описати будь-яку задачу (як це можуть робити звичайні мови програмування, до яких ми звикли, такі, як C++ чи Python). Наприклад, Script містить лише один стек, у який можна поміщувати операції та дані, які пізніше можна (за правилами стеку) звідтіля діставати. З цієї особливості видно чому саме мова не є Тюрінг-повною. Адже таким чином

неможливо реалізувати, скажімо цикли, а отже достатньо складно створити будь-яку програму, що не є вкрай примітивною 11).

Ефіріум, в свою чергу, дозволяє писати смарт-контракти на мові програмування Solidity, що є Тюрінг-повною високорівневою об'єктно орієнтованою мовою програмування подібною до JavaScript. Мова є достатньо молодою, а тому не має усіх зручних конструкцій, які присутні у тому ж JavaScript 12). Проте, незважаючи на цей невеликий недолік, Solidity дозволяє реалізовувати навіть достатньо складну бізнес модель за допомогою смарт-контрактів. У 2017 році було надзвичайно популярним проведення ICO (Initial Coin Offering) 13) за допомогою смарт-контрактів, проте зараз цей ажіотаж суттєво зменшився даючи дорогу вже більш складним смарт-контактам. Приклад простого смарт-контракту, написаного мовою програмування Solidity, можна побачити на Рисунок 1.1.

```
1  pragma solidity ^0.6.6;
2
3  contract Hello {
4
5      string name;
6
7      constructor() public {
8          name = 'Alice';
9      }
10
11     function get() public view returns(string memory storedName) {
12         return name;
13     }
14
15     function set(string memory _name) public returns(bool success) {
16         name = _name;
17         return true;
18     }
19
20 }
```

Рисунок 1.1 – Смарт-контракт для зберігання та отримання рядка з даними на мові для смарт-контрактів Solidity

Для ґрунтовного розуміння моєї роботи слід також пояснити що таке мікротранзакції (у блокчейні). Класично мікротранзакції роблять завдяки

відкриттю платіжних каналів між учасниками (сторонами, що платять та сторонами, що отримують) 14). Мікротранзакцією зазвичай називають транзакцію на відносно невелику суму. Проте у контексті блокчейн-технології це поняття має дещо ширший зміст. Щоб ґрунтовно його зрозуміти, потрібно спершу мати уявлення про те, як відбувається транзакція.

Кожен користувач мережі має приватний ключ, який є послідовністю випадковим чином згенерованих байтів (наприклад, 256 байтів). Цей ключ користувач зберігає у таємниці від інших. З приватного ключа математичними перетвореннями можна отримати адресу, яка і буде ідентифікувати конкретного користувача, та на яку інші користувачі зможуть надсилати криптовалюту. Адреса у криптовалюті Ефіріум складається з 42 символів і завжди починається з "0x". Приклад приватного ключа та адреси зображено на Рисунок 1.2. Проте жодними маніпуляціями з адресою не можна отримати приватний ключ (за лінійний, чи субекспоненційний час), про що важливо пам'ятати. У момент, коли власник приватного ключа хоче здійснити транзакцію, він "підписує" цим ключем відповідний набір даних, а саме, хто є одержувачем переказу, яка сума платежу тощо. Маючи таку транзакцію будь-хто може перевірити, що вона була дійсно "підписана" власником приватного ключа відповідної адреси. Після "підписання" транзакція може бути опублікована до блокчейну, де її додають до блоку, після чого вона набуває статусу опублікованої 15).

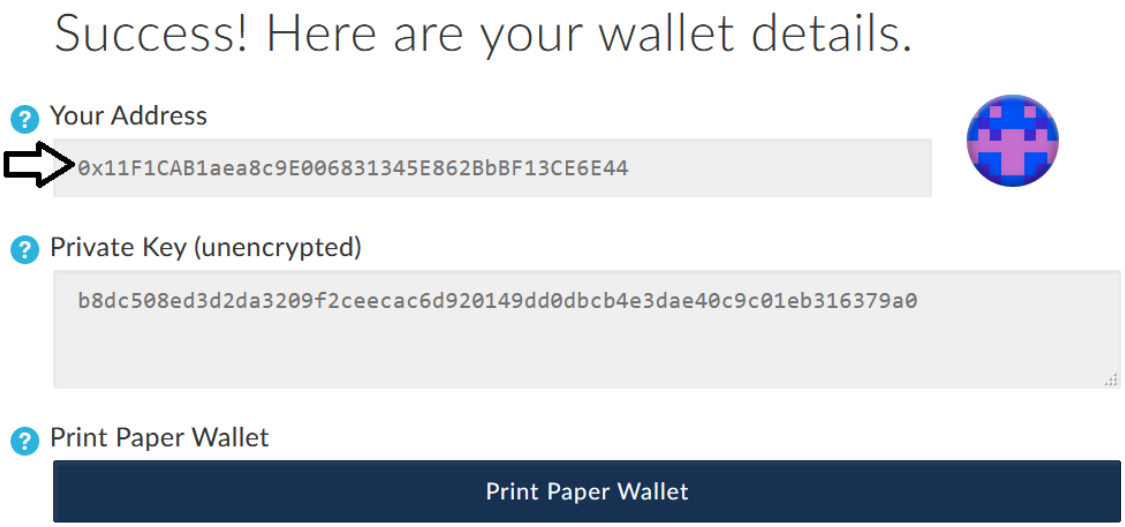


Рисунок 1.2 – Приклад адреси та приватного ключа користувача криптовалюти Ефіріум

З мікротранзакцією все відбувається так само, але без етапу публікації до мережі. Це робиться з метою економії коштів (оскільки для публікації потрібно заплатити комісію майнерам для її додавання) та часу, адже публікація займає певний час (в Ефіріумі приблизно 20 секунд залежно від розміру комісії). Кожна наступна мікротранзакція робиться на суму

$$S_{last} = S_{prev} + S_{curr} \quad (1.1)$$

, де S_{prev} – сума попереднього неопублікованого мікроплатежу (або 0, якщо такий платіж відсутній);

S_{curr} – сума поточного мікроплатежу.

Таким чином, можна здійснювати безліч мікротранзакцій практично миттєво без жодної комісії, а коли S_{last} стає достатньо великою з точки зору одержувача коштів (у нашому випадку особи, яка надає консультацію), він публікує останній платіж (на суму S_{last}) до блокчейну, таким чином закріплюючи своє право на володіння коштами у розподіленому реєстрі 16).

1.2 Змістовний опис і аналіз предметної області

Проєкт, по суті, являє собою iOS додаток у вигляді месенджеру, що має можливості звичайного месенджеру (такі, як знаходити користувача, переписуватися з ним, створювати обліковий запис тощо) на додачу до спеціальних для надання та отримання консультаційних послуг. Таким чином існують наступні види чатів:

– безкоштовний. Звичайний чат, де ніхто нікому не платить, а він створений лише для спілкування;

– заблокований. Якщо людина є, скажімо, дуже відомим викладачем французької мови, то вона може обрати опцію, що щоб їй написала незнайома людина, остання мусить заплатити певну фіксовану суму для того, щоб це зробити. Це може бути потрібно для уникнення спам-повідомлень таким користувачам;

– розблокований. Після того, як користувач заплатив необхідну суму Заблокований цей чат між ним та консультантом стає Розблокованим та тепер учасники чату можуть переписуватися безкоштовно;

– я плачу. Після того, як замовник з консультантом домовилися про дату та час консультації у Безкоштовному чи Розблокованому чаті, створюється чат для замовника виду "Я плачу", де за кожне повідомлення від консультанта замовник платить певну суму в криптовалюті, що залежить від кількості символів у повідомленні. Ціна за символ тексту у повідомленні обирається консультантом. Оплата відбувається у вигляді мікротранзакції, щоразу як замовник прочитує повідомлення від консультанта;

– мені платять. Коли з консультантом є чат виду "Я плачу", то для консультанта такий чат має вид "Мені платять". Тобто за кожне повідомлення від консультанта у такому чаті він отримує мінротранзакцію від замовника.

Оскільки основною особливістю розробки є спосіб оплати консультацій, то зупинюся на цьому процесі детальніше. Нехай існує чат між двома користувачами – Алісою (замовник) та Бобом (консультант). Відповідно вид чату є "Я плачу" та "Мені платять" для Аліси та Боба відповідно. Для того, щоб Боб міг приймати мікротранзакції від Аліси, мусять виконатися такі умови:

– у Аліси мусить бути деяка сума на смарт-контракті для платформи, що є достатньою для консультації (тобто поки Аліса зможе оплачувати нові повідомлення від Боба – нові мікротранзакції від неї будуть створюватися);

– ці кошти мусять бути "заморожені" на цьому смарт-контракті на декілька днів (наприклад 5). Для "заморозки" коштів Алісі лише потрібно викликати відповідний метод на смарт-контракті. Ця "заморозка" необхідна для того, щоб уникнути шахрайства з боку Аліси у подальшому. Способи атаки на систему будуть наведені пізніше.

Якщо вищезазначені умови виконано, то після того, як Боб почне писати Алісі, вона почне платити йому за усі непрочитані повідомлення у чаті. Причому за повідомлення від Аліси до Боба ніхто платити не буде. Оплата відбувається у вигляді мікротранзакцій наступним чином:

– рахується загальна кількість символів у всіх непрочитаних повідомленнях від Боба до Аліси, усі такі повідомлення позначаються як прочитані;

– поточна сума до оплати розраховується, як:

$$S_{curr} = \text{rate} * \text{total\_characters} \quad (1.2)$$

, де rate – ціна за символ, що встановлюється Бобом для усіх його чатів;

total_characters – загальна кількість символів у непрочитаних повідомленнях від Боба до Аліси;

– за формулою 1.1 розраховується сума, яку Аліса мусить вказати у мікротранзакції до Боба;

– формується Ефіріум транзакція від Аліси до Боба, де вона вказує:

а) адресу від кого платіж (свою Ефіріум адресу, що була створена для неї разом з приватним ключем під час її реєстрації у сервісі);

б) адресу до кого платіж (Ефіріум адресу Боба);

в) суму платежу (S_{last});

г) функцію, яка викликається у смарт контракті (це потрібно для того, щоб бути впевненим, що відбувається саме переказ коштів з такими параметрами, а не щось інше; в даному випадку це функція transfer);

– Аліса підписує транзакцію з вищезазначеними параметрами своїм приватним ключем та передає підписану транзакцію Бобу;

– Боб, отримавши мікротранзакцію, перевіряє, що всі параметри, що вона містить правильні, та за допомогою адреси Аліси перевіряє, що, дійсно, саме вона підписала цю транзакцію. Також він перевіряє, що у Аліси є "заморожені" кошти та що їх достатньо для оплати суми, що зазначена у мікротранзакції;

– якщо усі вищеперелічені кроки були успішними, то мікротранзакція вважається успішно надісланою.

Такі мікротранзакції можуть відбуватися доти, доки Боб не вирішить, що сума, яку йому вказала Аліса в останній мікротранзакції надто велика (скажімо, перевищує $200). Тоді Боб прийме рішення про те, що

мікротранзакцію необхідно опублікувати до блокчейну. Це відбувається наступним чином:

– Боб публікує останню мікротранзакцію від Аліси до блокчейну. Ця транзакція буде містити усю суму, що йому "винна" Аліса;

– транзакція деякий час (близько 20 секунд, але це значення може варіюватися в залежності від багатьох факторів) буде додаватися до нового блоку (а значить і до блокчейну) майнерами, причому за комісію останнім заплатить Аліса, оскільки саме вона підписувала цю транзакцію;

– після додавання до блокчейну Боб матиме гарантію перерахунку коштів, оскільки його баланс у смарт-контракті вже було збільшено.

Якщо після цього Боб, скажімо, вирішить вивести ці гроші з розробленої мною системи, то він просто здійснить виклик відповідного методу смарт-контракту та його гроші перерахуються на будь-яку адресу, що він вкаже (а отже, наприклад, і на будь-який Ефіріум гаманець від стороннього розробника).

Після усього вищезазначеного якщо Боб знову буде консультувати Алісу, то все буде аналогічно, оскільки за формулою 1.1 S_{prev} буде дорівнювати нулю, бо не буде жодної неопублікованої до блокчейну мікротранзакції.

Використовуючи описані підходи користувачі можуть повністю не довіряти один одному чи сервісу (trustless схема), але все одно зможуть користуватися сервісом та отримувати (чи надавати) консультаційні послуги. Блокчейн уможливлює trustless взаємодію і за іншого застосування, наприклад в машинному навчанні 17). Я вважаю, що це надзвичайно важлива конкурентна перевага розробленого сервісу у порівнянні з будь-якими іншими для надання консультаційних послуг онлайн.

Сервіс є **trustless**, оскільки:

– якщо замовник хоче обманути консультанта і створити невірну транзакцію, то останній одразу про це дізнається, оскільки кожна мікротранзакція перевіряється автоматично додатком консультанта при отриманні, а тому будь-які подібні махінації не вдасться зробити;

– якщо консультант надає неякісні послуги, чи просто пише повідомлення, щоб заробити таким чином на замовнику, то останній може відразу видалити чат з таким консультантом, тим самим переставши йому платити;

– якщо сервіс спробує забрати собі кошти користувачів, то нічого не вийде, оскільки у нього не зберігаються приватні ключі користувачів, а смарт-контракт, що реально містить гроші користувачів має відкритий висхідний код та працює на блокчейні, а тому він не може виконувати жодні функції, що не прописані в коді.

Нижче наведено діаграми діяльності для реєстрації користувача (Рисунок 1.3), входу (Рисунок 1.4) та загальних дій з месенджером (Рисунок 1.5).

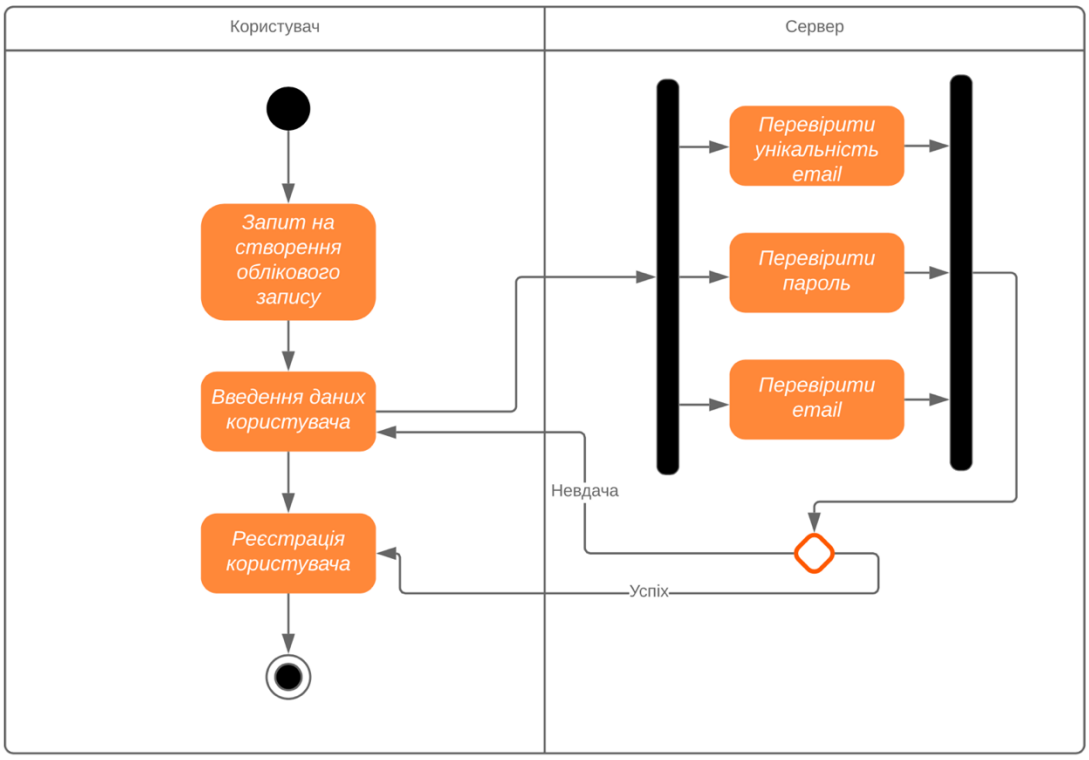


Рисунок 1.3 – Схема структурна діяльності: Реєстрація користувача

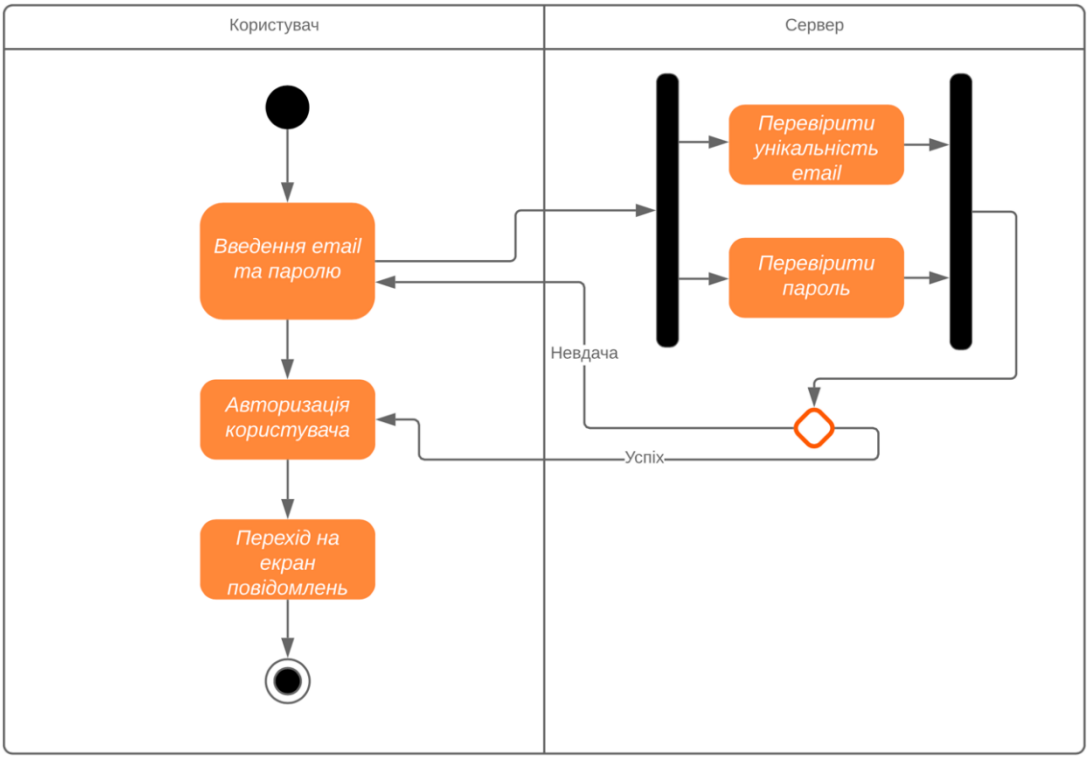


Рисунок 1.4 – Схема структурна діяльності: Вхід користувача

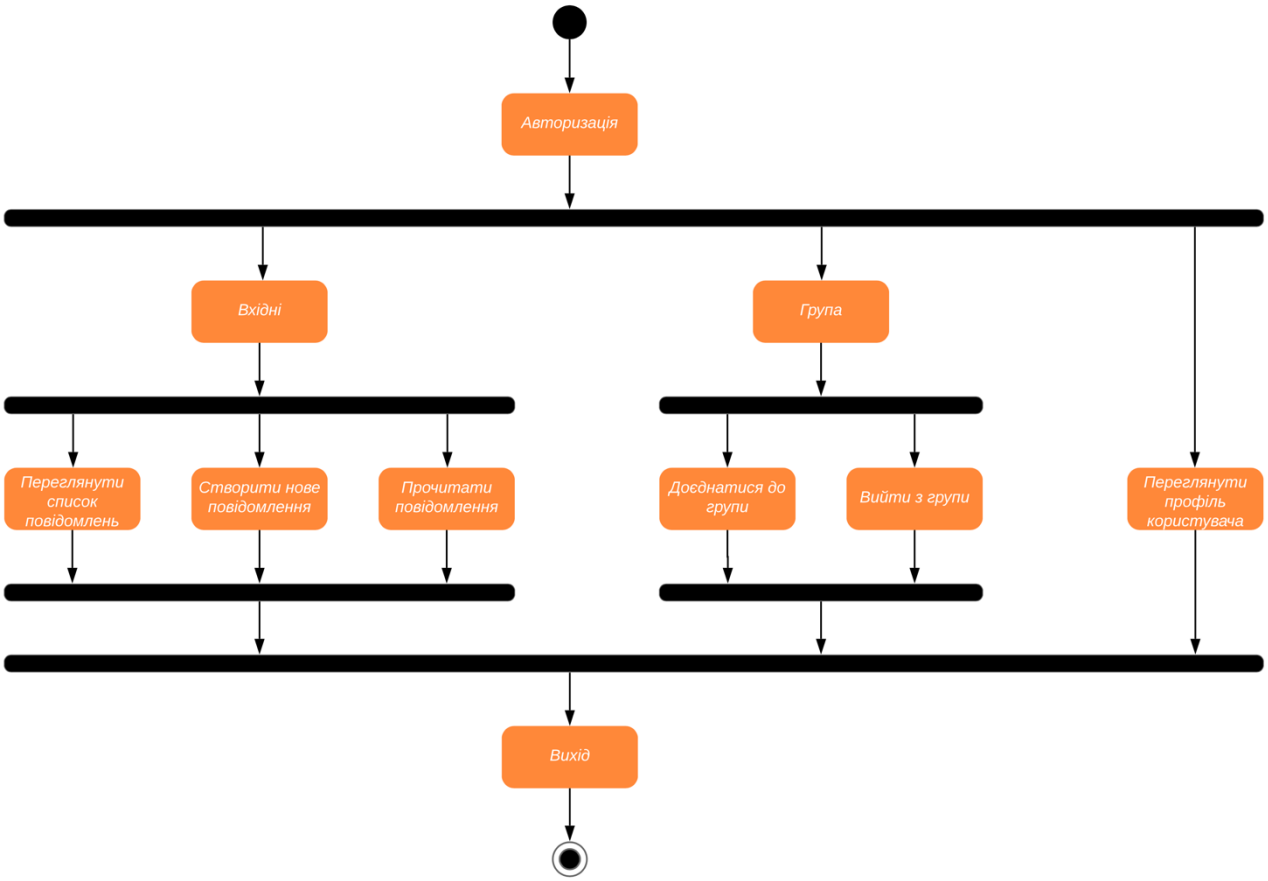


Рисунок 1.5 – Схема структурна діяльності: загальні дії з месенджером

1.3 Аналіз успішних IT-проєктів

1.3.1 Аналіз відомих технічних рішень

Для оплати консультаційних послуг онлайн переважно використовують стандартний підхід, що слабо варіюється від рішення до рішення.

При реєстрації (чи пізніше в налаштуваннях) користувач підв'язує до сервісу свою банківську картку, з якої, в ході консультацій, знімаються гроші (чи нараховуються, якщо користувач надає консультації). Крім того, оплата зазвичай здійснюється один раз після консультації, що також суттєво збільшує можливості зловмисника. Тобто таке технічне рішення має одразу цілу низку недоліків у порівнянні з розробленим сервісом, а саме:

– у разі, якщо під час консультації замовник спорожнить (чи заблокує) свою картку, то консультація для нього буде безкоштовною;

– у разі, якщо консультант є невідомим, то користувачі можуть йому недовіряти, оскільки вони не є впевненими у якості надання послуг;

– у разі, якщо у сервісі відбудеться збій, чи, що ще гірше, він виявиться зловмисним – користувачі ніяк не зможуть захистити свої кошти.

1.3.2 Аналіз відомих програмних продуктів

Слід почати з того, що дослідивши сферу консультаційних послуг онлайн, мені не вдалося знайти жодного рішення, що було б хоч чимось схоже на дану розробку, а саме: оплата відбувалася б у криптовалюті, чи оплата відбувалася б за допомогою мікроплатежів у криптовалюті.

Першим програмним продуктом, на якому я хотів би зупинитися є онлайн сервіс для викладання англійської мови – SayABC 18).

Переваги сервісу:

– зручний веб-сайт, авторизувавшись на якому викладач бачить свій розклад, час до заняття та інші корисні метрики;

– сервіс сам піклується про те, щоб знаходити викладачу заняття (тобто цей сервіс не для фрілансерів, а є роботою на повний робочий день).

Недоліки:

– викладач не може почати викладання одразу після реєстрації, а йому потрібно зачекати 10 - 14 днів для його перевірки сервісом та підписання з ними договору;

– необхідність підписання договору з сервісом;

– необхідність працювати за чітким графіком; усі заняття відбуваються рівно 40 хвилин;

– викладач змушений довіряти сервісу;

– сервіс здійснює виплату лише раз на місяць (для порівняння у моєму сервісі консультант сам вирішує коли, та як часто, публікувати мікротранзакції та виводити гроші та свій криптовалютний гаманець).

Іншим сервісом, що вартий уваги є International Online Consulting 19). Це платформа для надання бізнес консультацій у різних сферах, а саме: бухгалтерії, податків, юриспруденції, маркетингу та інтернаціоналізації.

Переваги:

– один сервіс для цілої низки сфер;

– відеодзвінки для надання консультацій.

Недоліки:

– консультант не може почати консультування одразу після реєстрації, потрібен час на його підтвердження;

– консультант змушений довіряти сервісу;

– сервіс здійснює виплату за фіксованим графіком.

Останнім сервісом, на який хочеться звернути увагу є наш вітчизняний – B2B Consult 20). Це всеукраїнський онлайн сервіс консультацій із законодавства.

Переваги:

– можливість задавання питання одразу на сервіс, а не конкретному консультанту, що може бути вигідно;

– для консультанта можливість обрання на яке саме питання відповідати;

– консультант може працювати в будь-який час та будь-яку кількість годин.

Недоліки:

– консультант не може обрати ціну за яку він працює. Він може лише обирати питання, де вже вказана ціна за відповідь на нього;

– після надання відповіді консультантом на запитання він отримує оплату від того, хто написав питання, але відповідь на нього є публічно доступною;

– консультант змушений довіряти сервісу.

1.4 Аналіз вимог до програмного забезпечення

Користувач системи за допомогою додатку взаємодіє з розробленим мною API для використання усіх функцій месенджеру таких, як авторизація, реєстрація, надсилання повідомлення, перегляд повідомлень, зміни даних профілю тощо. Також користувач може викликати методи цього API для взаємодії з Ефіріум блокчейном. Такими методами є, наприклад, перегляд балансу користувача, підписування мікротранзакції, публікація мікротранзакції тощо. Детальніше з функціями додатку можна ознайомитися за допомогою діаграми варіантів використання, що зображена на Рисунок 1.6.

Рисунок 1.6 – Схема структурна варіантів використання

1.4.1 Розроблення функціональних вимог

Функціональні вимоги до сервісу можна побачити у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги до сервісу

Номер	**Назва**	**Опис**
FR01	Реєстрація	Користувач мусить мати змогу зареєструватися у системі ввівши: – адресу електронної пошти (обов'язково); – пароль (обов'язково); – ім'я; – друге ім'я; – прізвище; – дату народження у форматі (рррр-мм-дд); – опис; – ключові слова для його пошуку іншими користувачами. Шлях доступу: /auth/register Варіанти відповіді API: – HTTP-статус: 200 { status: 'success', userId: ідентифікатор користувача, address: Ефіріум адреса користувача, prKey: приватний ключ користувача,

Продовження таблиці 1.1

Номер	Назва	Опис

		token: токен для доступу до методів API для зареєстрованих користувачів
		};
		– HTTP-статус: 409
		{
		status: 'error',
		message: 'Користувач з такою адресою пошти вже існує, спробуйте іншу'
		};
		– HTTP-статус: 500.

FR02	Авторизація	Користувач мусить мати змогу авторизуватися у системі після введення чинних адреси електронної пошти та паролю, за умови, що користувач зареєстрований у системі.
		Шлях доступу: /auth/login
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success',
		userId: ідентифікатор користувача,
		token: токен для доступу до методів API для зареєстрованих користувачів
		};
		– HTTP-статус: 401
		{

Продовження таблиці 1.1

Номер	Назва	Опис
		status: 'error', message: 'Ваш емейл чи пароль введені невірно' }; HTTP-статус: 500.

FR03	Отримання списку чатів	Користувач мусить мати змогу отримати список своїх чатів після передачі токену.
		Шлях доступу: /chat/chatList
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success', chats: [{
		userId: ідентифікатор користувача чату,
		chatId: ідентифікатор чату,
		chatType: тип чату,
		fromUser: ідентифікатор користувача-відправника,
		toUser: ідентифікатор користувача-отримувача,
		firstName: ім'я / undefined,
		middleName: друге ім'я / undefined,
		lastName: прізвище / undefined,
		avatar: фото профілю закодоване у форматі base64 / undefined,
		lastMsgText: текст останнього

Продовження таблиці 1.1

Номер	Назва	Опис
		повідомлення / undefined, lastMsgTime: дата та час останнього повідомлення / undefined }, ...] }; HTTP-статус: 500.

FR04	Отримання повідомлень чату	Користувач мусить мати змогу отримати усі повідомлення чату після передачі токену.
		Шлях доступу: /chat/messages
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success',
		amount: сума останньої неопублікованої мікротранзакції / undefined,
		messages: [{
		msgId: ідентифікатор повідомлення,
		userId: ідентифікатор користувача,
		text: текст повідомлення,
		isRead: true/false - чи прочитано повідомлення,
		time: час створення повідомлення
		}, ...]
		};
		– HTTP-статус: 500.

Продовження таблиці 1.1

Номер	Назва	Опис

FR05	Відправка повідомлення	Користувач мусить мати змогу надсилати повідомлення іншому користувачу передавши токен, ідентифікатор чату та текст повідомлення.
		Шлях доступу: /chat/message
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success',
		createdAt: дата та час створення повідомлення
		};
		– HTTP-статус: 500.

FR06	Читання повідомлень чату	Користувач мусить мати змогу помітити усі повідомлення в чаті як прочитані передавши токен та ідентифікатор чату.
		Шлях доступу: /chat/readMessages
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success'
		};
		– HTTP-статус: 500.

Продовження таблиці 1.1

Номер	Назва	Опис

FR07	Перегляд свого балансу Ефіріум адреси	Користувач мусить мати змогу переглянути баланс на своїй Ефіріум адресі, передавши токен.
		Шлях доступу: /bc/balanceInAddress
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success',
		currency: 'ETH',
		balanceInEth: баланс користувача в Ефірах
		};
		– HTTP-статус: 500.

FR08	Перегляд свого балансу на смарт-контракті сервісу	Користувач мусить мати змогу переглянути свій баланс, що зберігає смарт-контракт сервісу передавши токен.
		Шлях доступу: /bc/balanceInContract
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'success',
		currency: 'ETH',
		balanceInEth: баланс користувача в Ефірах
		};
		– HTTP-статус: 500.

Продовження таблиці 1.1

Номер	Назва	Опис

FR09	Поповнення смарт-контракту	Користувач мусить мати змогу поповнити смарт-контракт сервісу з своєї Ефіріум-адреси передавши токен; суму в Ефірах, на яку він хоче поповнити смарт-контракт; приватний ключ адреси, з якої мусить відбутися переказ коштів.
		Шлях доступу: /bc/replenishContract
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'mining...',
		txHash: хеш транзакції
		};
		– HTTP-статус: 500.

FR10	Перевірка мікроплатежу	Користувач мусить мати змогу перевірити отриману мікротранзакцію на коректність даних, що він містить передавши: – токен; – підписану мікротранзакцію; – Ефіріум адресу, яка підписувала мікротранзакцію; – адресу отримувача; – суму переказу (у wei). Шлях доступу: /bc/verifyTransfer Варіанти відповіді API: – HTTP-статус: 200

Продовження таблиці 1.1

Номер	Назва	Опис

		{ status: 'success',
		msg: 'Транзакція чинна'
		};
		– HTTP-статус: 200
		{
		status: 'error',
		msg: 'Параметри транзакції та параметри, що були передані різняться'
		};
		– HTTP-статус: 500.

FR11	Підписання мікроплатежу	Користувач мусить мати змогу підписати мікротранзакцію з вказаними параметрами передавши: – токен; – підписану мікротранзакцію; – Ефіріум адресу, яка підписувала мікротранзакцію; – адресу отримувача; – суму переказу (в ETH). Шлях доступу: /bc/signTransfer Варіанти відповіді API: – HTTP-статус: 200 { status: 'success', rawTx: підписану мікротранзакцію,

Продовження таблиці 1.1

Номер	Назва	Опис

		totalAmount: сума мікротранзакції
		};
		– HTTP-статус: 200
		{
		status: 'error',
		message: 'Недостатньо коштів'
		};
		– HTTP-статус: 500.

FR12	Публікація мікроплатежу	Користувач мусить мати змогу опублікувати підписану мікротранзакцію до Ефіріум блокчейну, передавши токен та ідентифікатор мікроплатежу.
		Шлях доступу: /bc/publishTransfer
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: 'mining...',
		txHash: хеш транзакції
		};
		– HTTP-статус: 500.

Продовження таблиці 1.1

Номер	Назва	Опис

FR13	Перегляд усіх неопублікованих мікроплатежів	Користувач мусить мати змогу переглянути усі неопубліковані мікротранзакції, що були здійснені у чаті.
		Шлях доступу: /bc/transfers
		Варіанти відповіді API:
		– HTTP-статус: 200
		{
		status: success,
		txs: [{
		txId: ідентифікатор мікротранзакції,
		fullName: повне ім'я користувача,
		direction: in/out - напрям мікротранзакції (до користувача, чи від нього),
		amount: сума мікротранзакції (в ETH),
		createdAt: дата та час створення мікротранзакції
		}, ...]
		};
		– HTTP-статус: 500.

Базуючись на функціональних вимогах була побудована матриця трасування вимог, з якою можна ознайомитися на Рисунок 1.7.

Варіант використання / Вимога	FR01	FR02	FR03	FR04	FR05	FR06	FR07	FR08	FR09	FR10	FR11	FR12	FR13
Реєстрація	✅												
Авторизація		✅											
Робота з чатом			✅	✅	✅	✅							
Робота з блокчейном							✅	✅	✅			✅	✅
Підготовка мікроплатежу										✅	✅		

Рисунок 1.7 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

До сервісу висуваються наступні нефункціональні вимоги:

– мобільний додаток мусить працювати на iPad сьомого покоління та iPad Pro 9.7" третього покоління обидва лише у горизонтальній орієнтації екрану, якщо на них встановлена iPadOS 13.2 чи вище;

– сервіс мусить працювати лише за умови підключення до інтернету;

– API мусить давати відповідь на запити протягом не більш, ніж 5 секунд за умови навантаження до 100 одночасних запитів;

– iPad мусить мати принаймні 0.5 гігабайт вільної пам'яті для роботи додатку.

1.5 Постановка задачі

1.5.1 Призначення розробки

Розробка призначена для надання консультаційних послуг онлайн за допомогою блокчейн-технології та мікротранзакцій.

Метою розробки є спрощення надання консультаційних послуг онлайн та збільшення рівня довіри сторін одна до одної.

1.5.2 Цілі та завдання розробки

Завданням розробки є спрощення процесу надання консультаційних послуг онлайн, уникнення необхідності довіряти сервісу чи консультантам. Для досягнення такого завдання було використано блокчейн-технологію та мікротранзакції. Цілями розробки було:

– розробити iPadOS додаток у вигляді месенджеру;
– розробити API для функцій месенджеру;
– розробити API для взаємодії з блокчейном;
– розробити смарт-контракт та опублікувати його до Ефіріум блокчейну.

1.6 Висновки по розділу

У цьому розділі була описана моя робота як у цілому, так і деталізовано про сам процес оплати та надання консультаційних послуг за допомогою розробленого додатку. Були проаналізовані технічні рішення, що часто застосовуються у цій сфері онлайн, а також був проведений аналіз декількох конкурентів, що хоч і сильно відрізняються від даної розробки, але є найбільш близькими серед сервісів, що вдалося знайти. Також були детально описані функціональні вимоги до системи, а також нефункціональні.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

За першого запуску додатку перед користувачем з'являється екран авторизації та можливість зареєструватися. Під час реєстрації, після введення даних загальних даних користувача для нього створюється приватний ключ для Ефіріум блокчейну та цей ключ одразу зберігається у KeyChain на iPad. На додачу до цього користувачу показується його адреса новоствореного гаманця та QR-код для можливості поповнення балансу адреси вже на етапі реєстрації. Якщо ж у користувача вже є обліковий запис, то він авторизується у системі використовуючи свої пошту та пароль.

Після реєстрації/авторизації користувач потрапляє на екран чатів та повідомлень. Тут він має змогу або написати повідомлення у вже існуючі чати, або знайти користувача, та створити з ним чат. Детальніше про типи чатів, їх особливості та спосіб оплати консультаційних послуг див. розділ 1.2 Змістовний опис і аналіз предметної області.

З екрану чатів та повідомлень користувач має змогу перейти до налаштувань, де він може змінювати свої реєстраційні дані, мову інтерфейсу; переглядати свій баланс у смарт-контракті системи та непідтверджені мікротранзакції з інформацією про них. Використовуючи список непідтверджених мікротранзакцій користувач може опублікувати будь-яку з них за умови, що він є отримувачем мікротранзакції, а не її відправником. Після підтвердження опублікованої транзакції майнерами, користувач повідомляється про це, та його баланс оновлюється.

Діаграму бізнес процесу можна побачити на Рисунок 2.1.

Рисунок 2.1 – Схема бізнес процесу

2.2 Архітектура програмного забезпечення

Програмне забезпечення складається з наступних частин:

– клієнтська частина у вигляді мобільного додатку для операційної системи iPadOS, що запускається на Apple iPad різних версій;

– серверна частина, що, в свою чергу, містить у собі:

а) API для роботи усіх функцій, що пов'язані з операціями месенджеру, які є в розробці. Сюди відноситься робота з обліковими записами користувачів; збереження назв чатів, що ці користувачі мають; збереження історію повідомлень цих чатів;

б) API для взаємодії з блокчейном, що є окремо виділеним API, з яким взаємодіє API для месенджеру завдяки використання інтерфейсу. Тут знаходяться безпосередньо методи виклику функцій смарт-контракту та деякі функції для отримання інших даних, наприклад балансу адреси користувача у Ефіріум блокчейні (не у смарт-контракті);

в) база даних PostgreSQL, що була використана для зберігання даних користувачів. У ній зберігаються усі дані, що потрібні для API для месенджеру (дані користувачів та їх чатів), а також Ефіріум-адресу користувача та список його транзакцій. Інші ж дані отримуються безпосередньо з блокчейну, що і є, по суті, базою даних.

Схему файлів проєкту можна побачити на Рисунок 2.2.

Рисунок 2.2 – Схема файлів проєкту

Крім того для отримання даних з Ефіріум блокчейну, чи публікації транзакцій був використаний провайдер блокчейну – Infura. Цей сервіс має високопродуктивні сервери з швидким підключенням до мережі Інтернет. На цих серверах зберігається та підтримується в актуальному стані повна нода Ефіріум блокчейну, що значить що там зберігається копія повністю всіх блоків блокчейну починаючи від найпершого, що було створено 30 липня 2015 року. В середньому кожен новий блок з'являється у мережі (майниться) кожні 10 секунд, чим і пояснюється необхідність швидкого Інтернет-з'єднання. Саме тому було обрано використовувати провайдер блокчейну, а не розгортати власний вузол мережі на сервері, оскільки в такому разі до серверу одразу

висувається багато вимог з продуктивності (як значних постійних ресурсів як диску, так і оперативної пам'яті).

Архітектуру проєкту можна побачити на Рисунок 2.3.

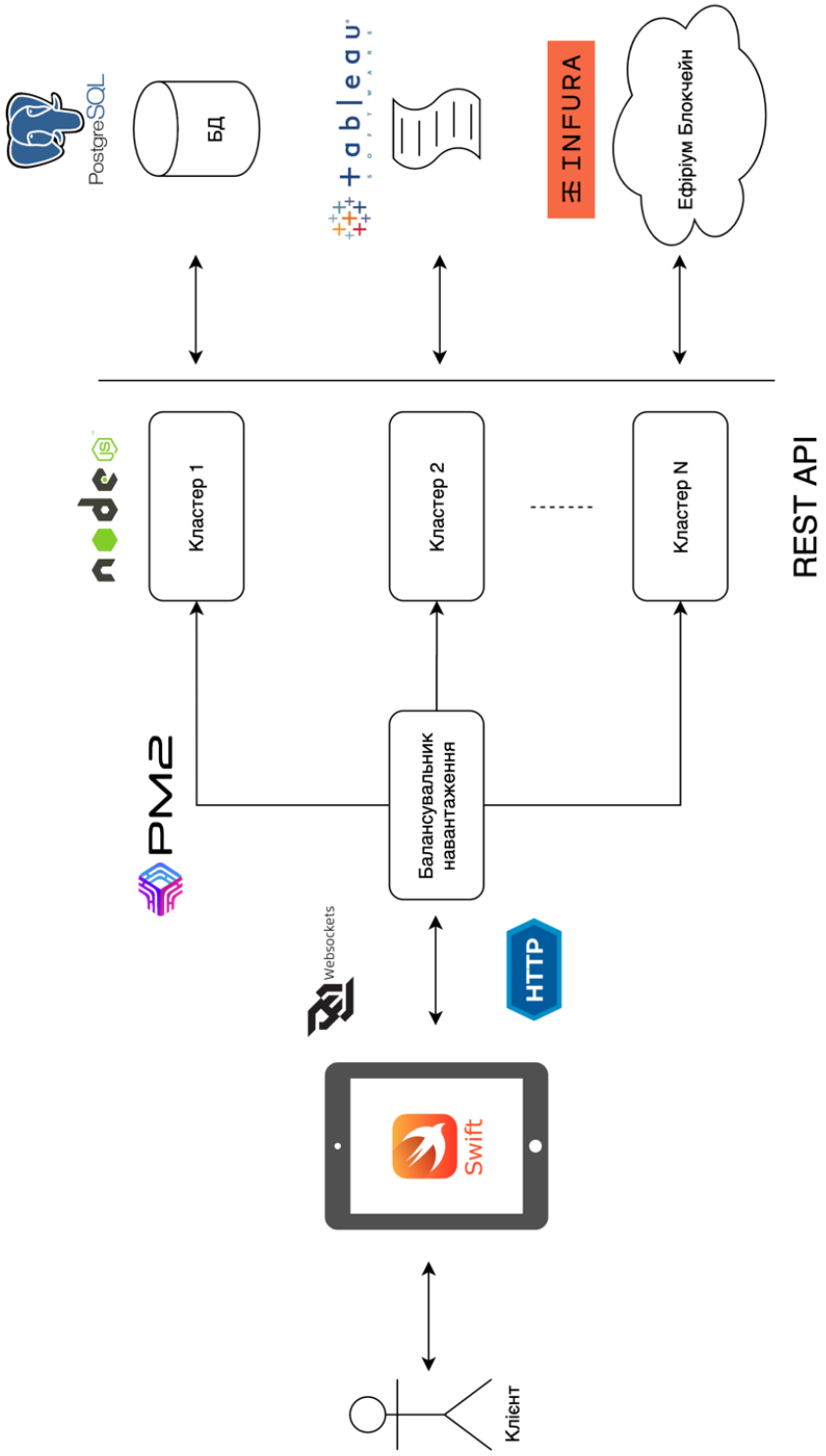


Рисунок 2.3 – Схема архітектури проєкту

Базу даних, що використовується в даному проєкті – реляційну базу даних PostgreSQL було обрана через високу ступінь надійності та постійність формату даних, що зберігаються до неї. Вона має задовільну швидкодію, а це було необхідно, оскільки при активному використанні користувачами сервісу у неї буде надходити велика кількість даних у вигляді повідомлень у чатах від користувачів. Тому вимоги до швидкодії до неї висувалися більші, ніж для використання у сервісах без функціоналу месенджеру. У подальшому також, у разі активного використання сервісу користувачами, матиме сенс використати нереляційну базу даних для швидкого збереження та доступу до повідомлень чату, а PostgreSQL залишити для зберігання більш важливих та цінних даних, що зберігаються для сервісу. Саму ж схему бази даних можна побачити на Рисунок 2.4.

Рисунок 2.4 – Схема бази даних

База даних містить таблиці, що відповідають за зберігання даних користувача (табл. User), таблиць для даних месенджеру (табл. ChatUser, Chat, ChatFile, ChatMessage) та таблиць для даних, що пов'язані з блокчейном (табл. Wallet та Transaction). Ключовою таблицею є таблиця User, що містить основні дані користувача. У користувача є гаманець (табл. Wallet) та він може бути відправником чи отримувачем транзакції (табл. Transaction). Оскільки у кожного чату може бути багато користувачів, а кожен користувач може бути учасником багатьох чатів, то було створено проміжну таблицю ChatUser, що логічно зв'язує чати з їх користувачами.

Наведу опис кожної таблиці бази даних. Їх можна побачити у таблицях Таблиця 2.1 - Таблиця 2.7.

Таблиця 2.1 – Опис таблиці User

Колонка	**Опис**	**Тип даних**	**Ключ**
UserId	Ідентифікатор	int4	PK
Email	Електронна адреса	varchar(256)	
FirstName	Ім'я	varchar(256)	
MiddleName	Друге ім'я	varchar(256)	
LastName	Прізвище	varchar(256)	
BirthDate	Дата народження	date	
Description	Опис користувача для його пошуку	text	
AvatarBase64	Картинка профілю користувача у форматі base64	text	

Продовження таблиці 2.1

Колонка	Опис	Тип даних	Ключ
PasswordHash	Хеш паролю	varchar(256)	
Version	Версія шифрування паролю	int2	
IsCompromised	Відмітка чи було пароль скомпрометовано	bool	

Таблиця 2.2 – Опис таблиці Chat

Колонка	Опис	Тип даних	Ключ
ChatId	Ідентифікатор	int4	PK
ChatType	Тип чату між користувачами	ChatType	
FromUser	Ідентифікатор того, хто створив чат (для отримання мікроплатежів)	int4	FK
ToUser	Ідентифікатор того, хто отримувач повідомлень чату (для створення мікроплатежів)	int4	FK

Продовження таблиці 2.2

Колонка	Опис	Тип даних	Ключ
CreatedAt	Дата створення чату	timestamp(6)	

Таблиця 2.3 – Опис таблиці ChatUser

Колонка	Опис	Тип даних	Ключ
ChatUserId	Ідентифікатор	int4	PK
ChatId	Ідентифікатор чату	int4	FK
UserId	Ідентифікатор користувача у чаті	int4	FK

Таблиця 2.4 – Опис таблиці ChatMessage

Колонка	Опис	Тип даних	Ключ
ChatMessageId	Ідентифікатор	int4	PK
UserId	Ідентифікатор користувача-відправника повідомлення	int4	FK
ChatId	Ідентифікатор чату	int4	FK
MessageType	Тип повідомлення	MessageType	

Продовження таблиці 2.4

Колонка	Опис	Тип даних	Ключ
MessageText	Текст повідомлення	text	
IsRead	Відмітка чи є повідомлення прочитаним	bool	
CreatedAt	Час створення повідомлення	timestamp(6)	

Таблиця 2.5 – Опис таблиці ChatFile

Колонка	Опис	Тип даних	Ключ
ChatFileId	Ідентифікатор	int4	PK
ChatMessageId	Ідентифікатор повідомлення, що є файлом	int4	FK
Filename	Назва файлу	varchar(50)	

Таблиця 2.6 – Опис таблиці Transaction

Колонка	Опис	Тип даних	Ключ
TransactionId	Ідентифікатор	int4	PK
FromUserId	Ідентифікатор від якого користувача транзакція	int4	FK

Продовження таблиці 2.6

Колонка	Опис	Тип даних	Ключ
ToUserId	Ідентифікатор до якого користувача транзакція	int4	FK
ForMessageId	Ідентифікатор для якого повідомлення транзакція	int4	FK
TransactionAmountWei	Сума транзакції у wei	text	
TransactionStatus	Статус транзакції	TransactionStatus	
RawTransaction	Повні дані транзакції	varchar(1024)	
TransactionHash	Хеш транзакції	char(66)	

Таблиця 2.7 – Опис таблиці Wallet

Колонка	Опис	Тип даних	Ключ
WalletId	Ідентифікатор	int4	PK
UserId	Ідентифікатор користувача-власника гаманця	int4	FK
Address	Ефіріум адреса користувача	char(42)	

Також були створені власні типи даних у базі даних. Їх опис наведено у таблиці 2.8.

Таблиця 2.8 – Опис власних типів у базі даних

Тип	Можливі значення	Призначення
ChatType	'free', 'locked', 'unlocked', 'paying', 'group'	Розрізнення різних видів чатів в залежності від необхідності платити, типу оплати та кількості учасників
MessageType	'text message', 'file message'	Розрізнення різних видів повідомлень для того, щоб дізнатися чи потрібно брати дані з таблиці ChatFile, чи ні
TransactionStatus	'unpublished', 'outdated', 'mining', 'mined'	Розрізнення різних видів статусів транзакції для відображення їх статусу відповідним чином

Сервер, що працює з базою даних написаний на NodeJS. Оскільки API планується для використання багатьма клієнтами одночасно, то був використаний балансувальник навантаження NodeJS додатку – PM2. Він дозволяє API працювати на багатьох ядрах комп'ютера у багатьох процесах, таким чином значно збільшуючи швидкодію та відмовостійкість сервера.

Для зв'язку клієнта з сервером клієнт використовує переважно HTTP-запити, а для отримання нових повідомлень використовується вебсокетне з'єднання (websockets). А саме: для надсилання будь-яких запитів до сервера використовуються POST та GET HTTP запити. В той час, як для месенджеру прийнято використовувати вебсокетне з'єднання для усіх видів запитів я його використовую лише там, де це є необхідним, а саме лише для отримання нових повідомлень. Оскільки вебсокетне з'єднання потрібно підтримувати увесь час "відкритим" та кількість одночасних підключень зазвичай суттєво менше, ніж кількість запитів до сервера. Втім зовсім без вебсокетного з'єднання майже не обійтись. Серед альтернатив можна зазначити технології Long Pooling для отримання інформації від сервера у момент її отримання сервером, а не у момент запиту клієнтом даних, проте цей метод має свої недоліки та зупинятися детально я на ньому не буду. Таким чином для отримання нових повідомлень користувачем у момент, коли йому їх надіслали, а не в момент запиту клієнтом списку повідомлення було використане вебсокетне з'єднання.

У зв'язку з розробленням trustless сервісу у архітектурному рішенні також варто згадати спосіб передачі криптовалюти від одного користувача системи іншому. Тоді, як у розділі 1.2. Змістовний опис і аналіз предметної області був описаний весь процес надання консультаційних послуг, то тут було зосереджено увагу на способі передачі та публікації мікроплатежів, з яким можна ознайомитися на Рисунок 2.5. Існує чотири сторони: замовник (Аліса), отримувач (Боб), посередник у вигляді сервера та смарт-контракт у блокчейні. Аліса взаємодіє зі смарт-контрактом (а отже і з блокчейном) лише для поповнення власного балансу (цього кроку можна буде уникнути, якщо це не її перша консультація та у неї залишилося достатньо коштів з минулої консультації). Боб, в свою чергу, взаємодіє з блокчейном теж лише один раз для публікації останньої мікротранзакції до блокчейну для закріплення за собою права володіння коштами, що він отримав від Аліси під час

консультації. Хоча цей крок теж можна уникнути, оскільки у разі надання консультацій Алісі на регулярній основі він може не публікувати до мережі мікротранзакцію щоразу після завершення консультації.

Коли ж Аліса створює мікротранзакцію та передає її Бобу, то вона за це ніяк не платить, бо взаємодія з блокчейном не відбувається, а отже платіж проходить безкоштовно та миттєво. Після отримання мікроплатежу Боб перевіряє усі її дані і у разі їх валідності він продовжує консультацію з Алісою. І так само відбуваються й наступні транзакції. Коли ж Боб публікує відповідну мікротранзакцію від Аліси, то він за це не платить, а платить Аліса, оскільки саме вона підписала цю мікротранзакцію своїм приватним ключем тим самим даючи згоду на переказ відповідної суми Бобу.

Рисунок 2.5 – Схема структурна послідовностей мікроплатежів системи

Було створено аналітичні звіти у додатку Tableau, що відображає основну аналітику сервісу. Приклад звітів можна побачити на Рисунок 2.6.

Рисунок 2.6 – Схема аналітичний звітів

2.3 Аналіз безпеки даних

Безпеку сервісу можна розділити на багато складових: безпека даних на клієнті, на сервері, безпека передачі даних тощо. Зупинюся на кожному пункті детальніше.

Єдині приватні дані, що зберігаються на пристрої користувача – це його приватний ключ. Він зберігається в Apple KeyChain, що вважається достатньо безпечним сховищем приватних даних на пристроях з операційною системою від компанії Apple.

З точки зору даних, що зберігаються на сервері – це дані користувачів, що зберігаються у базі даних. Крім звичайних способів захисту бази даних, як то створення окремого користувача з спеціальними правами для доступу до бази даних, були використані певні додаткові способи захисту. Оскільки з даних користувача, мабуть, найбільш важливим є пароль, то його захист і була першочергова задача. У базі даних зберігається хеш паролю у вигляді Argon2(SHA512(пароль) + "сіль"). Тобто користувач може придумати та використовувати пароль довільної довжини, що складається з довільних символів – сервіс не накладає на нього жодних обмежень "зверху". Щодо ж обмежень "знизу", то застосовуються наступні обмеження:

- довжина паролю мусить бути не менше 8 символів;
- у паролі мусить бути принаймні один символ у нижньому регістрі;
- у паролі мусить бути принаймні один символ у верхньому регістрі;
- у паролі мусить бути принаймні один спеціальний символ регістрі;
- у паролі мусить бути принаймні одна цифра.

Якщо пароль пройшов усі ці перевірки, то його хеш (sha512) відправляється на сервер. Тут для нього створюється "сіль", яка додається до паролю перед виконанням хешування за допомогою Argon2. Щоправда цей алгоритм створений таким чином, що він для хешування потребує багато ресурсів

комп'ютера (оперативної пам'яті та місця на диску), причому ці параметри можна вказувати. Це зроблено з тією метою, щоб якщо зловмисник отримає копію бази даних, то на підбір пароля, що відповідав би хешу, який зберігається у базі даних, зловмиснику довелось би витрачати дуже багато часу. Скажімо, якщо на хешування за допомогою Argon2 комп'ютеру потрібно 512 МБ пам'яті на диску та воно займає 2 секунди на сервері, що щоб перебрати пароль мінімальної довжини (тобто 8 символів) зловмиснику знадобиться:

$$avg\_num\_options = UNICODE_{size} * pass\_min\_len / 2 \quad (2.1)$$

, де *avg_num_options* – в середньому кількість варіантів пароля, що потрібно буде перебрати зловмиснику;

$UNICODE_{size}$ – кількість видимих символів (тобто таких, які теоретично може ввести користувач) у таблиці UNICODE 21);

pass_min_len – мінімальна довжина паролю, що дозволена в системі.

Знаючи середню кількість варіантів пароля та час, що потрібен на перевірку одного пароля можемо порахувати загальний час підбору паролю (лише в одному потоці комп'ютера):

$$t\_pass_{total} = avg\_num\_options * t_{one} \quad (2.2)$$

, де t_pass_{total} – загальний час в середньому, що потрібен для підбору одного пароля в одному потоці;

$avg\_num\_options$ – з попередньої формули;

t_{one} – середній час хешування одного пароля, щоб отримати хеш формату, що зберігається у базі даних.

Тобто для розробленого сервісу, підставивши числа у формулу 2.1 отримуємо 143696 * 8 / 2 = 574784. А підставивши значення з формули 2.1 у формулу 2.2 отримаємо 574784 * 2 ≈ 319 годин на підбір лише одного пароля в середньому, що робить процес перебору невигідним.

Щоправда, на практиці для злому паролю використовуються так звані "райдужні таблиці", що містять список найбільш часто вживаних користувачами паролів, але навіть використовуючи такий спосіб ціна атаки все ще буде досить високою. Проте, за необхідності можна додати ще одну перевірку на пароль, що придумує користувач, а саме щоб його пароль не був у топ 1000 найбільш вживаних паролів. Це дозволить збільшити ціну атаки, щоправда тоді сервіс може втрачати користувачів через занадто жорсткі вимоги до паролю.

Ще одним захистом паролю є колонки версії паролю та чи був він скомпрометований. Колонка версії паролю потрібна для того, щоб якщо схема хешування, що використовується наразі, а саме Argon2(SHA512(пароль) + "сіль") виявиться недостатньою у зв'язку з тим, що будуть розроблені більш потужні комп'ютери, чи з будь-яких інших причин, то цю схему хешування можна було б замінити на іншу. Робиться це наступним чином. За замовчуванням колонка версії містить версію 1. Якщо ми замінили спосіб хешування, то коли користувач увійде в систему наступного разу, то ми після перевірки його паролю за старою схемою збережемо у базі даних його пароль, що буде хешований за новою схемою та збільшимо значення у полі версії паролю на 1. Таким чином його пароль вже буде збережений більш безпечним чином, і наступного разу, коли користувач входитиме до системи, то для перевірки його паролю вже буде використовуватися нова схема хешування.

Іншою колонкою у базі даних є булівська відмітка чи був пароль скомпрометований. Якщо ми знаємо, що базу даних було скопійовано зловмисником, то ми ставимо відмітку у значення "істина" для всіх записів у таблиці. Коли користувач намагається увійти у систему за своїм паролем, то система його просить підтвердити особу іншим чином (електронною поштою, сервісом з подвійної автентифікації тощо). Після успішного підтвердження

особи користувач змушений змінити свій пароль на новий, таким чином роблячи копію, що є у зловмисника неактуальною.

Якщо все ж зловмиснику вдасться отримати доступ до бази даних та підібрати паролі користувачів, то він усе одно не зможе отримати доступ до коштів користувачів, оскільки приватний ключ користувача зберігається на мобільному пристрої в користувача, а тому навіть маючи пароль жодного доступу до коштів у зловмисника не буде.

Щодо захисту з'єднання між клієнтом та сервером, то за запуску сервісу на основному (не тестовому) сервері потрібно змінити протокол передачі даних з HTTP на HTTP/2 з TLS 1.3 для шифрування трафіку. Трафік сокетів (websockets) шифрується, оскільки використовується з'єднання wss. Також для ускладнення неавторизованого доступу до API у нього є ключ доступу, який однаковий для всіх клієнтів. Більш того для кожного клієнту після авторизації надається токен для авторизації на сервісі на певну кількість днів, причому цей токен повністю автентифікує клієнта у сервісі.

2.4 Висновки по розділу

У цьому розділі була розглянута технічна реалізація проєкту, спосіб роботи сервісу та його безпека. Таким після цього розділу у читача мусить бути чітке розуміння того, як саме додаток спроєктований та які технології він використовує, а також мусить бути зрозуміло чому були обрані ті, чи інші технології. Крім того розділ 2.3 Аналіз безпеки даних пояснює технічні рішення для захисту приватних даних користувача, а саме приватного ключа та паролю, оскільки це є найважливіші дані, з якими працює сервіс.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Тестовий план

З метою аналізу якості проєкту був розроблений тестовий план, який має на меті описати як мусить відбуватися процес тестування системи. Тестовий план було розроблено згідно з стандартом IEEE 829 22).

3.1.1 Вступ

Цей тест план створено для тестування месенджеру на платформі iPadOS, що базується на Ефіріум мікроплатежах та має на меті наступне:

– визначити утиліти для тестування та увесь процес тестування;

– визначити які функції будуть тестуватися, встановити часові очікування та поставити вимоги до тестувального середовища;

– визначити як тести будуть виконуватися.

3.1.2 Об'єкти тестування

Системи, що будуть тестуватися включають у себе iOS додаток, а також API, що було написане на NodeJS. Додаток має бути протестований на усіх основних версіях iOS від 10 до 12 (напр. 10.x.x, 11.x.x і т. д.) та основній версії iPadOS 13. API мусить бути протестоване лише у середовищі, у якому воно буде працювати після запуску проєкту, що є Ubuntu Server 18.04.3 LTS, 64 bit.

3.1.3 Функції, що мусять бути протестовані

Функції, що мусять бути протестовані (з точки зору користувача) включають наступне:

– реєстрація у додатку;

– авторизація у додатку;

– отримання списку чатів;
– отримання повідомлень чату;
– надсилання повідомлення у чаті типу "Безкоштовний";
– надсилання повідомлення у чаті типу "Я плачу";
– надсилання повідомлення у чаті типу "Мені платять";
– надсилання повідомлення у чаті типу "Розблокований";
– читання повідомлень чату;
– перегляд свого балансу Ефіріум адреси;
– перегляд свого балансу на смарт-контракті сервісу;
– поповнення смарт-контракту;
– перевірка мікроплатежу;
– підписання мікроплатежу;
– публікація мікроплатежу;
– перегляд усіх неопублікованих мікроплатежів.

3.1.4 Функції, що не мусять бути протестовані

Дизайн додатку не мусить бути протестований. Також середня пропускна спроможність Ефіріум блокчейну, а також середній час майнингу опублікованої транзакції не мусить бути протестоване, оскільки блокчейн буде використаний лише для публікації мікроплатежів до мережі, а тому, на відміну від створення мікроплатежів, публікація не мусить бути швидкою.

3.1.5 Підхід

Тести будуть проводитися для кожного прецеденту (test case), що буде збережено до хмарного сервісу для тестування під назвою TestLodge. Тестувальник буде запускати тести в TestLodge та помічати кожен прецедент як Пройдений/Невдалий/Пропущений. Тестувальник мусить додати деякі

нотатки щодо отриманих результатів та додати будь-які інші деталі, що можуть бути корисними якщо це можливо.

Коли тест помічений як "Невдалий баг-репорт" буде створений автоматично в трекері помилок, що вбудований в TestLodge.

Після завершення необхідно переглянути звіти з тестування в TestLodge та виправити помилки, що були знайдені якщо це необхідно.

3.1.6 Критерій Пройдений/Невдалий

Увесь основний функціонал системи мусить працювати згідно з очікуваннями та визначеними прецедентами. У системі не мусить бути виявлено жодних критичних дефектів та кінцевий користувач мусить мати змогу успішно зареєструватися, авторизуватися та переписуватися з іншими користувачами за допомогою повідомлень та публікувати мікротранзакції до блокчейну за запитом.

Після завершення процесу тестування у системі не мусить бути жодних нерозв'язаних критичних проблем; не більше, ніж 5 нерозв'язаних проблем середнього пріоритету; не більше, ніж 25 нерозв'язаних проблем низького пріоритету. Крім того 85% від тестів функціональності дотатку мусять бути пройденими так само, як і усі інтеграційні та юніт тести.

3.1.7 Критерії призупинки

Процес тестування мусить бути призупинено негайно якщо система має проблеми з авторизацією користувача, або відмову у будь-якій базовій CRUD операції.

3.1.8 Результати тестування

Після завершення тестування усі його результати будуть збережені у TestLodge, після чого потрібно буде запустити цей звіт з метою перевірки того, що все працює належним чином.

3.1.9 Задачі тестування

Наступні кроки мусять бути виконані:

– підготовлений План тестування;

– записані функціональні вимоги до системи;

– тестувальне середовище готове для проведення тестів (а саме є тестові дані, тестові облікові записи користувачів, тестова Ефіріум мережа тощо);

– усі тести пройдено;

– підготовлено звіт з результатами тестування.

3.1.10 Вимоги до середовища

Додаток для тестування мусить мати декілька облікових записів користувачів з деяким балансом у криптовалюті (тестового Ефіріум). У API мусить бути активована змінна оточення під назвою "development".

3.1.11 Відповідальність

Відповідальним за дотримання розкладу тестування, підтримання належної якості сервісу та усвідомлення ризиків тестування є розробник.

3.1.12 Розклад

Тестування починається за 4 тижні до дати запуску продукту. Перший етап тестування мусить бути завершений протягом 1 тижня.

3.1.13 Ризики та непередбачувані ситуації

У разі, якщо перший етап тестування не завершено протягом 1 тижня – виправлення помилок та фінальний етап тестування може бути затримано. Якщо це відбудеться, то тестування користувачами буде зсунуто, а це вплине на дату запуску продукту.

3.1.14 Затвердження

Право вирішувати коли завершувати процес тестування чи переходити на наступний етап залишається за розробником.

3.2 Діаграма тест плану

Для кращого розуміння безпосередньо тестів, що можна провести над сервісом було створено діаграму, що зображує тест план до сервісу у вигляді деревовидної структури (Рисунок 3.1).

Рисунок 3.1 – Схема тест плану до проєкту

3.3 Результати тестування

Було розроблено тести для тестування запитів API (запити див. у розділі 1.4.1 Розроблення функціональних вимог). Результатом виконання API тестів є успішне проходження 13 з 13 тестів успішно.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Програмне забезпечення складається з клієнтської та серверної частин. Крім того потрібно опублікувати смарт-контракт до основної мережі Ефіріум (Ethereum Main Net), щоб користувачі мали змогу проводити операції з криптовалютою Ефіріум, а не з її тестовим аналогом. Тому розгортання додатку слід розглядати, як розгортання трьох вищезазначених компонентів, що потрібно розгортати у порядку слідування нижче.

4.1.1 Публікація смарт-контакту до основної мережі Ефіріум

Для публікації смарт-контракту потрібно виконати наступне:

– встановити браузер під керуванням рушія Chromium;

– встановити розширення MetaMask (https://metamask.io/) у цей браузер;

– поповнити гаманець з основної мережі Ефіріум, що було створено при активації розширення на принаймні 0,01 ETH;

– увійти на сайт онлайн Ефіріум IDE – Remix (https://remix.ethereum.org/) та вставити код контракту в поле для коду;

– на панелі в Remix обрати опцію компіляції, після завершення якої натиснути кнопку "Опублікувати" та обрати провайдер Ефіріум ноди – розширення MetaMask, що було встановлене на кроці 2;

– після підтвердження транзакції у MetaMask та завершення процесу майнингу зберегти адресу новоствореного смарт-контракту для подальшого використання.

4.1.2 Розгортання API

Серверна частина написана на NodeJS та використовує з'єднання з базою даних та провайдер блокчейн ноди – Infura. Для розгортання API потрібно мати:

– комп'ютер, що підключено до мережі Інтернет з операційною системою Linux, MacOS чи Windows; розміром жорсткого диску не менше, ніж 16 ГБ; об'ємом оперативної пам'яті не менше, ніж 4 ГБ. Швидкість підключення до мережі Інтернет мусить бути не менше, ніж 400 Мб/сек для відвантаження файлів та 600 Мб/сек для завантаження файлів;

– на комп'ютері мусить бути встановлена NodeJS версії 13.7.x або більше та його пакетний менеджер npm версії 6.13.x або більше;

– на комп'ютері мусить бути встановлена база даних PostgreSQL версії 10 або більше;

– на сайті Infura(https://infura.io/) мусить бути створено обліковий запис та отримано ключі для підключення до основної мережі Ефіріум (Ethereum Main Net).

Після задоволення вищезазначених вимог можна починати процес встановлення API:

– виконати команди з файлу database.sql для створення необхідних таблиць у базі даних;

– створити файл змінної середовища (.env) згідно з шаблоном файлу .env.example та внести в нього наступне:

а) "NODE_ENV" встановити у значення "production";

б) заповнити відповідні змінні для підключення до бази даних;

в) вказати адресу смарт-контракту (див. розділ 4.1.1);

г) вказати адресу основної мережі Ефіріум, що надається Infura.

– перейти в командному рядку до директорії, де знаходиться код API та виконати команду "npm install" для встановлення залежностей;

– запустити додаток виконавши команду "npm start".

4.1.3 Розгортання мобільного додатку на iPad

Клієнтською частиною є мобільний додаток для Apple iPad під керуванням iPadOS. Для встановлення цього додатку потрібно мати Apple комп'ютер з операційною системою MacOS та встановленою програмою XCode на ньому. Крім того на комп'ютері мусить бути встановлений пакетний менеджер pod. iPad, на який мусить бути встановлений мобільний додаток мусить бути підключений до комп'ютера за допомогою USB-кабелю. Після задоволення вищезазначених вимог процес встановлення мобільного додатку складається з наступних кроків:

– відкрити код мобільного додатку в XCode;

– змінити за необхідності IP адресу серверу та порти для HTTP та вебсокетного з'єднань;

– запустити програму Термінал та перейти в директорію, де знаходиться код мобільного додатку. У терміналі виконати команду "pod install" для встановлення необхідних бібліотек, що використовує мобільний додаток;

– запустити код, після чого мобільний додаток мусить бути встановлений на iPad.

4.2 Робота з програмним забезпеченням

Див. документи "Керівництво програміста" та "Керівництво користувача".

ВИСНОВКИ

В ході даної роботи було спроєктовано дизайн усіх екранів iOS додатку на базі якого був розроблений власне додаток. Для його коректної роботи було розроблено API, що складається з двох частин: API для месенджеру та для взаємозв'язку зі смарт-контрактом та блокчейном. Крім того було створено та опубліковано до блокчейну смарт-контракт, що містить логіку для роботи сервісу. Як результат було розроблено робочий сервіс для надання та отримання консультаційних послуг за допомогою мікротранзакцій з можливістю учасникам не довіряти до постачальника послуг чи сервісу. Для зручного бізнес-аналізу метрик розробки було створено аналітичні звіти за допомогою Tableau.

Для створення такого продукту була приділена увага питанням безпеки, оскільки це є однією з ключових переваг даного додатку, а також є стандартом для будь-якої високоякісної розробки. Крім того для створення якісного продукту був описаний тестовий план до проєкту згідно з стандартом IEEE 829. Також були написані інструкції для впровадження та супроводу проєкту; інструкції користування сервісом для користувача та програміста. Усі заплановані цілі були досягнуті.

ПЕРЕЛІК ПОСИЛАНЬ

1) Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V. Blockchain technology: beyond bitcoin. Appl. Innovation 2 / Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V. – 2016. – С. 6-10.
2) N. Szabo Smart contracts / N. Szabo., 1994.
3) K. Christidis, M. Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things / K. Christidis, M. Devetsikiotis. // IEEE Access. – 2016. – №4. – С. 2292 – 2303.
4) D. W. Kravitz, J. Cooper. Securing user identity and transactions symbiotically: IoT meets blockchain / D. W. Kravitz, J. Cooper. // 2017 Global Internet of Things Summit (GIoTS). – 2017.
5) A. Hasselgren, K. Kralevska, D. Gligoroski. Blockchain in healthcare and health sciences — A scoping review / A. Hasselgren, K. Kralevska, D. Gligoroski. // International Journal of Medical Informatics. – 2020. – №134.
6) Haferkorn M., Quintana Diaz J.M. Seasonality and Interconnectivity Within Cryptocurrencies - An Analysis on the Basis of Bitcoin, Litecoin and Namecoin. / Haferkorn M., Quintana Diaz J.M. // Lugmayr A. (eds) Enterprise Applications and Services in the Finance Industry. FinanceCom 2014. Lecture Notes in Business Information Processing, vol 217. Springer, Cham. – 2015. – №217. – С. 106–120.
7) Sampson J. Secret digital coin mining and trading is a threat to your business / Sampson J. // Computer Fraud & Security. – 2018. – №2018. – С. 8–10.
8) S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system / S. Nakamoto., 2008.

9) Dr. G. Wood. The "Yellow Paper": Ethereum's formal specification [Електронний ресурс] / Dr. G. Wood. – 2019. – Режим доступу до ресурсу: https://ethereum.github.io/yellowpaper/paper.pdf.
10) Z. Zheng, S. Xiea, H. Daib. An overview on smart contracts: Challenges, advances and platforms / Z. Zheng, S. Xiea, H. Daib. // Future Generation Computer Systems. – 2020. – №105. – С. 475–491.
11) Möser M., Eyal I., Gün Sirer E. Bitcoin Covenants / Möser M., Eyal I., Gün Sirer E. // Clark J., Meiklejohn S., Ryan P., Wallach D., Brenner M., Rohloff K. (eds) Financial Cryptography and Data Security. FC 2016. Lecture Notes in Computer Science. – 2016. – №9604. – С. 126–141.
12) Chris Dannen Introducing Ethereum and Solidity / Dannen C. – Berkley: Apress, 2017. – (1).
13) Chohan U. W. Initial coin offerings (ICOs): Risks, regulation, and accountability / Chohan U. W., 2019. – (Cryptofinance and Mechanisms of Exchange).
14) Burchert C., Decker C., Wattenhofer R. Scalable funding of bitcoin micropayment channel networks. / Burchert C., Decker C., Wattenhofer R. // Royal Society open science. – 2018. – №5.
15) Heilman E., Baldimtsi F., Goldberg, S. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. / Heilman E., Baldimtsi F., Goldberg, S. // International conference on financial cryptography and data security. Springer, Berlin, Heidelberg.. – 2016. – С. 43–60.
16) D. Rhodes How Blockchain Is Improving Micropayments Capabilities [Електронний ресурс] / D. Rhodes – 2019. – Режим доступу до ресурсу: https://komodoplatform.com/micropayments/.

17) Kurtulmus A. B., Daniel K. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. / Kurtulmus A. B., Daniel K. // arXiv preprint arXiv. – 2018. – №1802.
18) Teach English Online from Home with SayABC! [Електронний ресурс] – Режим доступу до ресурсу: https://t.sayabc.com/.
19) International Online Consulting [Електронний ресурс] – Режим доступу до ресурсу: https://www.oone.eu/.
20) B2B Consult: Всеукраїнський онлайн сервіс консультацій із законодавства [Електронний ресурс] – Режим доступу до ресурсу: https://b2bconsult.ua/.
21) Unicode [Електронний ресурс] / Wikipedia – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Unicode.
22) Lohr Claire 829-1998 - IEEE Standard for Software Test Documentation [Електронний ресурс] / C. Lohr – 1998. – Режим доступу до ресурсу: https://standards.ieee.org/standard/829-1998.html.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МЕСЕНДЖЕР ДЛЯ НАДАННЯ КОНСУЛЬТАЦІЙНИХ ПОСЛУГ НА ОСНОВІ БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА МІКРОТРАНЗАКЦІЙ

Технічне завдання

КПІ.ІП-6116.045490.03.91

“ПОГОДЖЕНО”

Керівник проєкту:

Недашківський Є. А.

Нормоконтроль:

Ліщук К. І.

Виконавець:

Кушка М. О.

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій.

Галузь застосування: Уся сфера консультаційних послуг онлайн (бізнес-консультування, фітнес-консультування, викладання іноземної мови онлайн тощо).

Наведене технічне завдання поширюється на розробку програмного забезпечення для надання консультаційних послуг онлайн за допомогою мікротранзакцій у криптовалюті [КПІ.ІП-6116.045490.03.91], котра використовується для надання консультаційних послуг будь-якого характеру онлайн та призначена для спрощення надання таких послуг та збільшення рівня довіри сторін одна до одної.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки месенджеру для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій є завдання на дипломне проєктування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для надання консультаційних послуг онлайн за допомогою блокчейн-технології та мікротранзакцій.

Метою розробки є спрощення надання консультаційних послуг онлайн та збільшення рівня довіри сторін одна до одної.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- реєстрація;
- авторизація;
- отримання списку чатів;
- отримання повідомлень чату;
- відправка повідомлення;
- читання повідомлень чату;
- перегляд свого балансу Ефіріум адреси;
- перегляд свого балансу на смарт-контракті сервісу;
- поповнення смарт-контракту;
- перевірка мікроплатежу;
- підписання мікроплатежу;
- публікація мікроплатежу;
- перегляд усіх неопублікованих мікроплатежів.

4.1.1.2 Для адміністратора системи:

- можливість надсилати HTTP-запити до API;
- переглядати аналітичні звіти про сервіс за допомогою Tableau.

4.1.2 Розробку виконати на платформі iPadOS/Linux

4.1.3 Додаткові вимоги:

- відображати QR-код новоствореної адреси під час реєстрації користувача;

– оновлення списку чатів та неопублікованих мікротранзакцій мусить відбуватися за допомогою жесту проведення пальцем зверху вниз.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

При реєстрації користувач не може ввести пароль, що не задовольняє вимогам з безпеки паролю, а саме: пароль менше восьми символів, без жодної літери у верхньому регістрі, без жодної літери у нижньому регістрі, без жодної цифри, без жодного спеціального символа.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

Для забезпечення цілісності інформації база даних мусить бути приведена до третьої нормальної форми.

4.2.4 Забезпечити постійний доступ до мережі Інтернет під час користування програмним забезпеченням.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування

4.3.3 Обслуговуючий персонал

Не вимагається.

4.4 Вимоги до складу і параметрів технічних засобів

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

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору ..... Apple A10 Fusion або потужніший.

4.4.2.2 Об‘єм ОЗП..... 3 ГБ або більше.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства iPadOS.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: HTTP-запит до сервера.

4.5.3 Результати повинні бути представлені в наступному форматі: HTTP-відповідь з сервера.

4.5.4 Програмне забезпечення (API) повинно обробляти HTTP-запити, що були надіслані лише з розробленого мобільного додатку, що встановлений на пристрої.

Клієнтська частина була розроблена на мові програмування Swift, а серверна – на NodeJS, що працює на сервері під керуванням операційної системи Linux/MacOS/Windows.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Розгорнути серверну частину на підходящому сервері, а клієнтську – "зібрати" та завантажити на відповідний пристрій за допомогою IDE Xcode.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

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

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 60 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Програма та методика тестування.

5.3.4 Керівництво програміста.

5.3.5 Керівництво користувача.

5.3.6 Опис програми.

5.4 Графічна частина повинна бути виконана на листках формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна варіантів використань.

5.4.2 Схема структурна діяльності.

5.4.3 Схема бази даних.

5.4.4 Схема структурна бізнес процесу.

5.4.5 Схема структурна послідовностей.

5.4.6 Креслення вигляду екранних форм.

5.4.7 Креслення вигляду звітних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1	Вивчення літератури за тематикою проєкту	17.03.2020	
2	Розробка технічного завдання	27.03.2020	Технічне завдання
3	Аналіз вимог та уточнення специфікацій	03.04.2020	Специфікації програмного забезпечення
4	Проєктування структури програмного забезпечення, проєктування компонентів	17.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5	Програмна реалізація програмного забезпечення	08.05.2020	Тексти програмного забезпечення

6	Тестування програмного забезпечення	15.05.2020	Тести, результати тестування
7	Розробка матеріалів графічної частини проєкту	20.05.2020	Графічний матеріал проєкту
8	Розробка матеріалів текстової частини проєкту	27.05.2020	Пояснювальна записка
9	Оформлення технічної документації проєкту	28.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

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

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МЕСЕНДЖЕР ДЛЯ НАДАННЯ КОНСУЛЬТАЦІЙНИХ ПОСЛУГ НА ОСНОВІ БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА МІКРОТРАНЗАКЦІЙ

Програма та методика тестування

КПІ.ІП-6116.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є. А. Недашківський

Нормоконтроль:

_____ К. І. Ліщук

Виконавець:

_____ М. О. Кушка

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний додаток у вигляді месенджеру для надання консультаційних послуг онлайн під платформу iPadOS, API написане на NodeJS та смарт-контракт для Ефіріум блокчейну розроблений на Solidity.

2 МЕТА ТЕСТУВАННЯ

Як результат тестування мусить бути перевірено наступне:

– відповідність роботи iPadOS додатку та API функціональним вимогам;

– наявність достатнього, згідно з вимогами, рівня безпеки даних, куди відноситься, але цим не обмежується:

а) зберігання паролів та приватних даних користувачів;

б) зберігання приватних ключів користувачів;

в) авторизований доступ до API;

г) коректність та атомарність роботи функцій смарт-контракту;

д) захищеність бази даних;

– відповідність системи Технічному завданню.

Тестування програмного забезпечення мусить відбуватися згідно з Планом тестування.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування розробленого продукту виконується згідно з методикою White box testing, що дозволяє краще створювати тести, оскільки відомо де програмний продукт вірогідно може породжувати помилки.

Перш за все програмний продукт мусить пройти Smoke testing, після чого можуть виконуватися функціональні тести. Для зменшення кількості помилок на етапі розробки доцільно використовувати юніт тести. Тестування інтерфейсу не відбувається через складність написання вимог до такого тестування та відсутність принципових відмінностей в інтерфейсі додатку від інтерфейсу звичайних месенджерів. Детальніше про методи тестування див. розділ 3 Аналіз якості та тестування програмного забезпечення Пояснювальної записки.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування викликів API доцільно проводити за допомогою утиліти Postman через її відносну простоту інтерфейсу та зручність. Решту тестів слід виконувати за допомогою утиліти Katalon Studio. Детальніше про засоби та порядок тестування див. розділ 3 Аналіз якості та тестування програмного забезпечення Пояснювальної записки.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МЕСЕНДЖЕР ДЛЯ НАДАННЯ КОНСУЛЬТАЦІЙНИХ ПОСЛУГ НА ОСНОВІ БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА МІКРОТРАНЗАКЦІЙ

Керівництво програміста

КПІ.ІП-6116.045490.05.33

“ПОГОДЖЕНО”

Керівник проєкту:

Є. А. Недашківський

Виконавець:

М. О. Кушка

Нормоконтроль:

К. І. Ліщук

Київ – 2020 року

Опис параметрів запитів та відповіді сервера див. у розділі **Error! Reference source not found. Error! Reference source not found.**.

– Методи API для облікових записів:

а) реєстрація

Endpoint: /auth/register:

Метод: POST

Запит: { email, [firstName], [middleName], [lastName], [birthDate], pass, [description], [keywords] };

б) авторизація

Endpoint: /auth/login

Метод: POST

Запит: { email, pass }.

– Методи API для месенджеру:

а) отримання списку чатів

Endpoint: /chat/chatList

Метод: GET

Запит: { token };

б) отримання повідомлень чату

Endpoint: /chat/messages

Метод: GET

Запит: { chatId, token };

в) відправка повідомлення

Endpoint: /chat/message

Метод: POST

Запит: { token, chatId, message };

г) читання повідомлень чату

Endpoint: /chat/readMessages

Метод: POST

Запит: { token, chatId }.

– Методи API для блокчейну:

а) перегляд балансу Ефіріум адреси

Endpoint: /bc/balanceInAddress

Метод: GET

Запит: { token };

б) перегляд балансу на смарт-контракті сервісу

Endpoint: /bc/balanceInContract

Метод: GET

Запит: { token };

в) поповнення смарт-контракту

Endpoint: /bc/replenishContract

Метод: POST

Запит: { token, amountEth, prKey };

г) перевірка мікроплатежу

Endpoint: /bc/verifyTransfer

Метод: GET

Запит: { token, rawTx, from, to, amount };

д) підписання мікроплатежу

Endpoint: /bc/signTransfer

Метод: POST

Запит: { token, from, to, amount, prKey };

е) публікація мікроплатежу

Endpoint: /bc/publishTransfer

Метод: POST

Запит: { token, txId };

ж) перегляд усіх неопублікованих мікроплатежів

Endpoint: /bc/transfers

Метод: GET

Запит: { token }.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МЕСЕНДЖЕР ДЛЯ НАДАННЯ КОНСУЛЬТАЦІЙНИХ ПОСЛУГ НА ОСНОВІ БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА МІКРОТРАНЗАКЦІЙ

Керівництво користувача

КПІ.ІП-6116.045490.06.34

“ПОГОДЖЕНО”

Керівник проєкту:

Є. А. Недашківський

Виконавець:

М. О. Кушка

Нормоконтроль:

К. І. Ліщук

Київ – 2020 року

При першому вході в додаток користувачу потрібно створити обліковий запис для користування сервісом. При реєстрації від або вона вводять свої реєстраційні дані, з яких є обов'язковими лише адреса електронної пошти та пароль. Після реєстрації для користувача створюється обліковий запис та Ефіріум адреса, яку можна поповнити за допомогою додатку будь-якого стороннього розробника Ефіріум гаманця.

Увійшовши до облікового запису користувач потрапляє на основний екран додатку – екран з списком чатів, яких у нього поки що немає. Для пошуку консультанта користувач вводить дані для пошуку, якими можуть бути: ім'я користувача, його друге ім'я, прізвище чи ключові слова, які були вказані консультантом. Після успішного знаходження консультанта можна розпочати з ним чат (детальніше про типи чатів див. розділ 1.2 Змістовний опис і аналіз предметної області Пояснювальної записки). Зазвичай це буде безкоштовний чат, де з консультантом можна буде домовитися про час та деталі консультації.

Консультації проводяться в платних чатах, де замовник платить консультанту за кожен символ у повідомленнях консультанта базуючись на ціні символа консультанта. Формули розрахунку вартості повідомлення наведені у розділі 1.2 Змістовний опис і аналіз предметної області Пояснювальної записки. Гроші з замовника списуються автоматично за фактом читання повідомлень.

У будь-який момент консультант може у налаштуваннях додатку опублікувати останню мікротранзакцію будь-якого чату, де йому платять. Після майнингу такої транзакції сума на рахунку консультанта у смарт-контракті оновиться. Крім того у налаштуваннях є змога змінити деякі свої реєстраційні дані, а також змінити мову інтерфейсу додатку (українська або англійська).

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МЕСЕНДЖЕР ДЛЯ НАДАННЯ КОНСУЛЬТАЦІЙНИХ ПОСЛУГ НА ОСНОВІ БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА МІКРОТРАНЗАКЦІЙ

Опис програми

КПІ.ІП-6116.045490.07.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є. А. Недашківський

Нормоконтроль:

_____ К. І. Ліщук

Виконавець:

_____М. О. Кушка

Київ – 2020 року

Тексти програмного коду

Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій

(Найменування програми (документа))

DVD-R

(Вид носія даних)

92 арк, 511 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

ЗМІСТ

1 ПРОГРАМНИЙ КОД API

1.1 Блокчейн API

index.js

```
module.exports = {
    web3: require('./singletons/web3'),
    contract: require('./singletons/contract'),
}
```

errors.js

```
module.exports = {
    Web3Error: class Web3Error extends Error {
        constructor(message) {
            super(message)
            this.name = 'Web3 Error'
        }
    }
}
```

smart-contract.js

```
const fs = require('fs')
const path = require('path')
const { bc } = require('../config')
const Web3 = require('web3')
const web3 = new Web3(new Web3.providers.HttpProvider(bc.infura))

module.exports = class Contract {

    constructor({ abi, address }) {
        this.abi = JSON.parse(
            fs.readFileSync(path.join(__dirname, abi)))
        this.address = address
        this.gas = 21000 // 2000000
        this.gasPrice = 20000000000 // 20 Gwei
        this.contract = new web3.eth.Contract(this.abi, this.address)
    }

    instance() { return this.contract }

    balanceOf(address) {
        return this.contract.methods.balanceOf(address).call()
    }
```

```
    deposit(from, prKey, value) {
        return this.__send(from, prKey, 'deposit', [], value)
    }

    transfer({ from, prKey, to, amount }) {
        return this.__send(from, prKey, 'transfer', [to, amount])
    }

    withdraw(from, prKey, amount) {
        return this.__send(from, prKey, 'withdraw', [amount])
    }

    __send(from, prKey, method, params, value = 0) {
        const query = this.contract.methods[method](...params)
        const encodedABI = query.encodeABI()

        return web3.eth.accounts.signTransaction(
            {
                nonce: web3.eth.getTransactionCount(from),
                data: encodedABI,
                from,
                gas: this.gas,
                gasPrice: this.gasPrice,
                to: this.contract.options.address,
                value
            },
            prKey,
            false,
        )
    }
}
```

singletons/index.js

```
module.exports = {
    contract: require('./contract'),
    web3: require('./web3'),
}
```

singletons/contract.js

```
const { bc } = require('../../config')
const path = require('path')
const Contract = require('../smart-contract')

const contract = new Contract({
    abi: path.join('data', 'abi.json'),
```

```
    address: bc.contractAddr,
})

module.exports = contract
```

singletons/web3.js

```
const { bc } = require('../../config')
const Web3 = require('web3')
const web3 = new Web3(new Web3.providers.HttpProvider(bc.infura))

module.exports = web3
```

1.2 Месенджер API

app.js

```
const express = require('express')
const bodyParser = require('body-parser')
const routes = require('./routes')
const config = require('./config')

const port = config.port || 3000
const app = express()
const server = require('http').createServer(app)
const io = new (require('./utils/socket'))(server)

app.use(bodyParser.json({ limit: '5mb' }))
app.use(bodyParser.urlencoded({ extended: true }))
app.use((req, res, next) => { req.io = io; next() })

app.use('/', routes)
io.start()

server.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

routes.js

```
const express = require('express')
const { auth, bc, chat } = require('./controllers')
const { verifyAppToken } = require('./middleware')

// eslint-disable-next-line new-cap
const r = express.Router()

r.post('/auth/register',                          auth.register)
r.post('/auth/login',                             auth.login)
```

```
r.post('/auth/updateUserData',     verifyAppToken, auth.updateUserData)
r.get('/auth/myProfile',           verifyAppToken, auth.myProfile)

r.get('/chat/chatList',            verifyAppToken, chat.chatsList)
r.get('/chat/messages',            verifyAppToken, chat.messages)
r.get('/chat/unreadMessages',      verifyAppToken, chat.unreadMessages)
r.get('/chat/totalEthAmount',      verifyAppToken, chat.totalEthAmount)
r.post('/chat/message',            verifyAppToken, chat.addMessage)
r.post('/chat/readMessages',       verifyAppToken, chat.readMessages)

r.get('/bc/balanceInAddress',      verifyAppToken, bc.balanceInAddress)
r.get('/bc/balanceInContract',     verifyAppToken, bc.balanceInContract)
r.post('/bc/replenishContract',    verifyAppToken, bc.replenishContract)
r.post('/bc/signTransfer',         verifyAppToken, bc.signTransfer)
r.post('/bc/signTransferByUserId', verifyAppToken, bc.signTransferByUserId)
r.get('/bc/verifyTranfer',         verifyAppToken, bc.verifyTransfer)
r.post('/bc/publishTransfer',      verifyAppToken, bc.publishTransfer)
r.get('/bc/transfers',             verifyAppToken, bc.transfers)

module.exports = r
```

middleware.js

```
const crypto = require('./utils/crypto')
const { verifyToken } = require('./utils/crypto')

const verifyAppToken = (req, res, next) => {
    const token = req.body.token ? req.body.token : req.query.token
    crypto.verifyToken(token)
        .then(decoded => {
            req.body.decoded = decoded
            next()
        })
        .catch(() => {
            res.status(401).send({
                status: 'unauthorized',
                message: 'Invalid token provided'
            })
        })
}

const verifySocketToken = (socket, next) => {
    const token = socket.handshake.query.token
    verifyToken(token)
        .then(({ userId }) => {
            console.log({ userId })
            socket.userId = userId
```

```
            next()
        })
        .catch(() =>
            console.error('Invalid token is provided via a socket connetion'))
}

module.exports = {
    verifyAppToken,
    verifySocketToken,
}
```

errors.js

```
module.exports = {
    Web3Error: class Web3Error extends Error {
        constructor(message) {
            super(message)
            this.name = 'Web3 Error'
        }
    },
    BreakPromiseChainError: class BreakPromiseChainError extends Error {
        constructor(msg) {
            super(msg)
            this.name = 'BreakPromiseChainError'
        }
    },
}
```

logger.js

```
const { createLogger, format, transports } = require('winston')
require('winston-daily-rotate-file')
const config = require('./config')
const fs = require('fs')
const path = require('path')
const util = require('util')

const env = config.env || 'development'
const logDir = path.join(__dirname, 'logs')

// Create the log directory if it does not exist
if (!fs.existsSync(logDir)) {
    fs.mkdirSync(logDir)
}

const dailyRotateFileTransportResults = new transports.DailyRotateFile({
    filename: `${logDir}/%DATE%-results.log`,
    datePattern: 'YYYY-MM-DD',
```

```
    maxSize: '20m',
    maxFiles: '14d'
})

const dailyRotateFileTransportErrors = new transports.DailyRotateFile({
    filename: `${logDir}/%DATE%-errors.log`,
    datePattern: 'YYYY-MM-DD',
    maxSize: '20m',
    maxFiles: '14d'
})

const logger = createLogger({
    // To the file
    level: 'debug',
    format: format.combine(
        format.timestamp({ format: 'HH:mm:ss' }),
        format.printf(info =>
            util.format(info.timestamp, info.level, info.message)
        )
    ),

    // On the screen
    transports: [
        new transports.Console({
            level: env === 'development' ? 'debug' : 'info',
            format: format.combine(
                format.colorize(),
                format.printf(info =>
                    util.format(info.timestamp, info.level, info.message))
            )
        }),
        dailyRotateFileTransportResults
    ],
    exceptionHandlers: [
        new transports.Console({
            level: 'error',
            format: format.combine(
                format.colorize(),
                format.printf(info =>
                    util.format(info.timestamp, info.level, info.message))
            )
        }),
        dailyRotateFileTransportErrors
    ]
})

module.exports = logger
```

.env.example

```
NODE_ENV=development

APP_PORT=80
JWT_PR_KEY=

PGUSER=
PGHOST=
PGPASSWORD=
PGDATABASE=
PGPORT=5432

CONTRACT_ADDRESS=
INFURA_ROPSTEN=
JWT_EXPIRE_TIME=

DEV_APP_PORT=8080
DEV_JWT_PR_KEY=

DEV_PGUSER=
DEV_PGHOST=
DEV_PGPASSWORD=
DEV_PGDATABASE=
DEV_PGPORT=

DEV_CONTRACT_ADDRESS=
DEV_INFURA_ROPSTEN=
DEV_JWT_EXPIRE_TIME=
```

controllers/index.js

```
module.exports = {
    auth: require('./auth.controller'),
    chat: require('./chat.controller'),
    bc: require('./bc.controller'),
}
```

controllers/auth.controller.js

```
const { auth } = require('../services')
const { generateToken } = require('../utils/crypto')
const logger = require('../logger')

const register = (req, res, next) => {
    const {
        email, pass, firstName, middleName, lastName, avatar, birthDate,
        keywords, description
```

```
    } = req.body

    const keywordsArr = keywords ? keywords.split(',').map(x => x.trim()) : []

    auth.register(
        email, pass, firstName, middleName, lastName, birthDate, keywordsArr,
        description, avatar
    )
        .then(({ userId, address, prKey }) => {
            logger.info(`Register result: ${!!userId}. User id: ${userId}`)

            if (userId) {
                generateToken(userId)
                    .then(token =>
                        res.status(200).send({
                            status: 'success',
                            userId,
                            address,
                            prKey,
                            token
                        }))
            } else res.status(409).send({
                status: 'error',
                message: 'There is already a user with such username. Maybe, \
that\'s your old account'
            })

            next()
        })
        .catch(err => {
            console.error({ err })
            res.sendStatus(500) && next(console.error(err))
        })
}

const login = (req, res, next) => {
    auth.login(req.body)
        .then(userId => {
            logger.info(`Login result (user id): ${userId}`)

            if (userId) {
                generateToken(userId)
                    .then(token =>
                        res.status(200).send({
                            status: 'success',
                            userId,
                            token
```

```
                        }))
            } else res.status(401).send({
                status: 'error',
                message: 'Your email or password is incorrect'
            })

            next()
        })
        .catch(err => {
            console.error({ err })
            res.sendStatus(500)
        })
}

const updateUserData = (req, res) => {
    const keywords = req.body.keywords
    const keywordsArr = keywords ? keywords.split(',').map(x => x.trim()) : []
    auth.updateUserData({
        userId: req.body.decoded.userId,
        keywords: keywordsArr,
        description: req.body.description
    })
        .then(() => res.status(200).send({
            status: 'success',
            message: 'User data were updated'
        }))
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

const myProfile = (req, res) => {
    auth.myProfile(req.body.decoded.userId)
        .then(([x]) => {
            const date = new Date(x.BirthDate)
            const year = date.getFullYear()
            const month = date.getMonth()
            const day = date.getDay()

            res.status(200).send({
                status: 'success',
                email: x.Email,
                firstName: x.FirstName,
                middleName: x.MiddleName,
                lastName: x.LastName,
                birthDate: `${year} ${month} ${day}`,
```

```
                description: x.Description,
                avatar: x.AvatarBase64
            })
        })
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

module.exports = {
    register,
    login,
    updateUserData,
    myProfile,
}
```

controllers/bc.controller.js

```
const { bc } = require('../services')
const { toEth } = require('../utils/bc')
const logger = require('../logger')

const balanceInAddress = (req, res) => {
    bc.balanceInAddress(req.body.decoded.userId)
        .then(balance => {
            const balanceInEth = toEth(balance)
            logger.debug(`User balance: ${balanceInEth}`)
            res.status(200).send({
                status: 'success',
                currency: 'ETH',
                balanceInEth
            })
        })
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

const balanceInContract = (req, res) => {
    bc.balanceInContract(req.body.decoded.userId)
        .then(balance => {
            const balanceInEth = `${toEth(balance)}`
            logger.debug(`User balance: ${balanceInEth}`)
            res.status(200).send({
                status: 'success',
```

```
                currency: 'ETH',
                balanceInEth
            })
        })
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

const replenishContract = (req, res) => {
    bc.replenishContract({
        userId: req.body.decoded.userId,
        amountEth: parseFloat(req.body.amountEth),
        prKey: req.body.prKey
    })
        .then(hash => {
            logger.debug({ hash })
            res.status(200).send({
                status: 'mining...',
                txHash: hash
            })
        })
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

const signTransfer = (req, res) => {
    bc.signTransfer(req.body)
        .then(tx => {
            logger.debug({ tx })
            res.status(200).send({
                status: 'success',
                rawTx: tx.rawTransaction
            })
        })
        .catch(err => {
            if (err.code === 'INVALID_ARGUMENT' && err.arg === 'amount') {
                return res.send({
                    status: 'error',
                    message: 'Insufficient funds'
                })
            }
            console.error(err)
            res.sendStatus(500)
```

```
        })
}

const signTransferByUserId = (req, res) => {
    bc.signTransferByUserId({
        msgId: req.body.msgId,
        fromUserId: req.body.decoded.userId,
        toUserId: req.body.toUserId,
        amount: req.body.amount,
        prKey: req.body.prKey
    })
        .then(([totalAmount, tx]) => {
            logger.debug({ totalAmount, tx })
            req.io.emit('upd-amount', ({
                fromUserId: req.body.decoded.userId,
                toUserId: req.body.toUserId,
                amount: `${toEth(totalAmount)}`
            }))
            res.status(200).send({
                status: 'success',
                rawTx: tx.rawTransaction,
                totalAmount
            })
        })
        .catch(err => {
            if (err.code === 'INVALID_ARGUMENT' && err.arg === 'amount') {
                return res.send({
                    status: 'error',
                    message: 'Insufficient funds'
                })
            }
            console.error(err)
            res.sendStatus(500)
        })
}

const verifyTransfer = (req, res) => {
    const { rawTx, from, to, amount } = req.body
    const isGood = bc.verifyTransfer(rawTx, from, to, amount)
    console.log({ isGood })
    if (isGood) {
        res.status(200).send({
            status: 'success',
            msg: 'The transaction is valid'
        })
    } else {
        res.status(200).send({
```

```
            status: 'error',
            msg: 'The transaction params and your params are different'
        })
    }
}

const publishTransfer = (req, res) => {
    bc.publishTransfer({
        userId: req.body.decoded.userId,
        txId: req.body.txId,
        socket: req.io
    })
        .then(hash => {
            logger.debug({ hash })
            res.status(200).send({
                status: 'mining...',
                txHash: hash
            })
        })
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })
}

const transfers = (req, res) =>
    bc.transfers(req.body.decoded.userId)
        .then(txs =>
            res.status(200).send({
                status: 'success',
                txs
            }))
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })

module.exports = {
    balanceInAddress,
    balanceInContract,
    replenishContract,
    signTransfer,
    signTransferByUserId,
    verifyTransfer,
    publishTransfer,
    transfers,
}
```

controllers/chat.controller.js

```
const { chat } = require('../services')

const chatsList = (req, res) => {
    chat.chatsList({ userId: req.body.decoded.userId })
        .then(chats => {
            res.status(200).send({
                status: 'success',
                chats
            })
        })
        .catch(err => {
            console.error(err)
            res.status(500).send({
                status: 'error',
                message: 'Error while receiving the list of chats from the db'
            })
        })
}

const messages = (req, res) => {
    chat.messages(req.query.chatId)
        .then(([amount, messages]) => {
            res.status(200).send({
                status: 'success',
                amount: amount ? `${amount}` : undefined,
                messages
            })
        })
        .catch(err => {
            console.error(err)
            res.status(500).send({
                status: 'error',
                message: 'Error with reading chat messages from the db'
            })
        })
}

const unreadMessages = (req, res) => {
    chat.unreadMessages(req.query.chatId, req.query.userId)
        .then(messages => {
            res.status(200).send({
                status: 'success',
                messages
            })
        })
```

```
        .catch(err => {
            console.error(err)
            res.status(500).send({
                status: 'error',
                message: 'Error with reading chat messages from the db'
            })
        })
}

const totalEthAmount = (req, res) =>
    chat.totalEthAmount(req.query.chatId)
        .then(amount =>
            res.status(200).send({
                status: 'success',
                amount
            }))
        .catch(err => {
            console.error(err)
            res.sendStatus(500)
        })

const addMessage = (req, res) => {
    chat.addMessage({
        chatId: req.body.chatId,
        userId: req.body.decoded.userId,
        text: req.body.message
    })
        .then(({ chatMsgId, createdAt, chatUsers, amount }) => {
            console.log({ chatMsgId })
            req.io.emit('new-message', ({
                msgId: chatMsgId,
                userId: req.body.decoded.userId,
                chatId: req.body.chatId,
                message: req.body.message,
                amount,
                createdAt
            }))
            res.status(200).send({
                status: 'success',
                createdAt
            })
        })
        .catch(err => {
            console.error(err)
            res.status(500).send({
                status: 'error',
                message: 'Error with adding a new message to the db'
```

```
            })
        })
}

const readMessages = (req, res) =>
    chat.readMessages(req.body.chatId, req.body.decoded.userId)
        .then(res.status(200).send({ status: 'success' }))
        .catch(err => {
            console.error(err)
            res.status(500).send({
                status: 'error',
                message: 'Error with updating messages status'
            })
        })

module.exports = {
    chatsList,
    messages,
    unreadMessages,
    totalEthAmount,
    addMessage,
    readMessages,
}
```

services/index.js

```
module.exports = {
    auth: require('./auth.service'),
    chat: require('./chat.service'),
    bc: require('./bc.service'),
}
```

services/chat.service.js

```
const { chat } = require('../db')
const { dateToLabel } = require('../utils')
const { toEth } = require('../utils/bc')

const chatsList = ({ userId }) => new Promise((resolve, reject) =>
    chat.getPersonalChats(userId)
        .then(chats => {
            chats = chats.map(x => ({
                userId: x.UserId,
                chatId: x.ChatId,
                chatType: x.ChatType,
                fromUser: x.FromUser,
                toUser: x.ToUser,
                firstName: x.FirstName,
```

```
                lastName: x.LastName,
                avatar: x.AvatarBase64,
                lastMsgText: x.MessageText,
                lastMsgTime: x.CreatedAt
            }))
            chats = chats.map(obj => {
                obj.lastMsgTime = dateToLabel(obj.lastMsgTime)
                return obj
            })
            resolve(chats)
        })
        .catch(reject))

const messages = chatId => new Promise((resolve, reject) =>
    Promise.all([
        chat.getTotalAmount(chatId),
        chat.getMessages(chatId)
    ])
        .then(([totalAmount, msgs ]) => {
            msgs = msgs.map(x => ({
                msgId: x.ChatMessageId,
                userId: x.UserId,
                text: x.MessageText,
                isRead: x.IsRead,
                time: `${x.CreatedAt}`.slice(16, 21)
            }))
            if (totalAmount.length)
                resolve([toEth(totalAmount[0].TransactionAmountWei), msgs])
            else
                resolve([undefined, msgs])
        })
        .catch(reject))

const unreadMessages = (chatId, userId) => new Promise((resolve, reject) =>
    chat.getUnreadMessages(chatId, userId)
        .then(msgs => {
            msgs = msgs.map(x => ({
                userId: x.UserId,
                text: x.MessageText,
                time: `${x.CreatedAt}`.slice(16, 21)
            }))
            resolve(msgs)
        })
        .catch(reject))

const totalEthAmount = chatId => new Promise((resolve, reject) =>
    chat.getTotalAmount(chatId)
```

```
        .then(totalAmount =>
            resolve(totalAmount.length ?
                `${toEth(totalAmount[0].TransactionAmountWei)}` : 0))
        .catch(reject))

const addMessage = ({ chatId, userId, text }) =>
    new Promise((resolve, reject) =>
        Promise.all([
            chat.addMessage(chatId, userId, text),
            chat.getChatUsers(chatId),
            chat.getTotalAmount(chatId)
        ])
            .then(([
                [{ CreatedAt, ChatMessageId }],
                chatUsers,
                totalAmount
            ]) => {
                const amount = totalAmount.length ?
                    `${toEth(totalAmount[0].TransactionAmountWei)}` :
                    undefined
                resolve({
                    chatMsgId: ChatMessageId,
                    createdAt: `${CreatedAt}`.slice(16, 21),
                    chatUsers,
                    amount
                })
            })
            .catch(reject))

const getMessageById = msgId => chat.getMessageById(msgId)

const readMessages = (chatId, userId) => chat.readMessages(chatId, userId)

module.exports = {
    chatsList,
    messages,
    unreadMessages,
    totalEthAmount,
    addMessage,
    getMessageById,
    readMessages,
}
```

services/bc.service.js

```
const { web3 } = require('../bc')
const db = require('../db/bc.db')
```

```
const { contract } = require('../bc/singletons')
const logger = require('../logger')
const { checkSignedFunc } = require('../utils/bc')
const { getDateTime } = require('../utils')
const { toWei } = require('../utils/bc')

const balanceInAddress = userId =>
    new Promise((resolve, reject) =>
        db.getUserAddress(userId)
            .then(([{ Address: addr }]) => {
                logger.debug(`User address: ${addr}`)
                return web3.eth.getBalance(addr)
            })
            .then(balance => resolve(balance))
            .catch(err => {
                console.error({ err })
                reject(err)
            }))

const balanceInContract = userId =>
    new Promise((resolve, reject) =>
        db.getUserAddress(userId)
            .then(res => {
                const address = res[0].Address
                logger.debug(`User address: ${address}`)
                return contract.balanceOf(address)
            })
            .then(balance => resolve(balance))
            .catch(err => {
                console.error({ err })
                reject(err)
            }))

const replenishContract = ({ userId, amountEth, prKey }) =>
    new Promise((resolve, reject) =>
        db.getUserAddress(userId)
            .then(([{ Address: addr }]) =>
                contract.deposit(addr, prKey, toWei(amountEth)))
            .then(({ rawTransaction }) => {
                console.log({ rawTransaction })
                web3.eth.sendSignedTransaction(rawTransaction)
                    .once('transactionHash', hash => {
                        logger.debug(`Transaction hash: ${hash}`)
                        resolve(hash)
                    })
                    .on('error', reject)
            })
```

```
            .catch(reject))

const signTransfer = ({ from, to, amount, prKey }) =>
    new Promise((resolve, reject) =>
        contract.transfer({ from, to, amount: toWei(amount), prKey })
            .then(tx => {
                logger.debug({ tx })
                const isGood = checkSignedFunc({
                    rawTx: tx.rawTransaction,
                    from,
                    params: ['address', 'uint256'],
                    paramsCheck: [to, toWei(amount)],
                    func: 'transfer'
                })

                console.log({ isGood })

                if (isGood) resolve(tx)
                else reject()
            })
            .catch(reject))

const signTransferByUserId = ({ msgId, fromUserId, toUserId, amount, prKey }) =>
    new Promise((resolve, reject) =>
        Promise.all([
            db.getUserAddress(fromUserId),
            db.getUserAddress(toUserId),
            db.lastUnpublishedTxAmountForChat(fromUserId, toUserId)
        ])
            .then(([
                [{ Address: from }],
                [{ Address: to }],
                txAmount
            ]) => {
                console.log({ msgId, fromUserId, toUserId, amount, prKey })
                const lastAmount = txAmount.length ?
                    txAmount[0].TransactionAmountWei : 0
                console.log({ txAmount, lastAmount })
                const fullAmount = parseInt(lastAmount) + parseInt(amount)
                console.log({ fullAmount })
                return Promise.all([
                    contract.transfer({
                        from,
                        to,
                        amount: fullAmount,
                        prKey
                    }),
```

```
                    from,
                    to,
                    fullAmount
                ])
            })
            .then(([tx, from, to, fullAmount]) => {
                const isGood = checkSignedFunc({
                    rawTx: tx.rawTransaction,
                    from,
                    params: ['address', 'uint256'],
                    paramsCheck: [to, fullAmount],
                    func: 'transfer'
                })

                console.log({ isGood })
                console.log({ tx })

                if (isGood) {
                    return Promise.all([
                        db.saveTransfer(
                            fromUserId,
                            toUserId,
                            msgId,
                            fullAmount,
                            tx.rawTransaction
                        ),
                        db.setAllTxsAsOutdated(fromUserId, toUserId),
                        fullAmount,
                        tx
                    ])
                } else reject(new Error('The transaction is not valid'))
            })
            .then(([,, fullAmount, tx]) => resolve([fullAmount, tx]))
            .catch(reject))

const publishTransfer = ({ userId, txId, socket }) =>
    new Promise((resolve, reject) => {
        db.getRawTxById(txId)
            .then(([{ TransactionStatus: status, RawTransaction: rawTx }]) => {
                console.log({ status, rawTx })
                if (status !== 'unpublished')
                    reject(new Error('The tx status differs from unpublished'))
                else
                    web3.eth.sendSignedTransaction(rawTx)
                        .once('transactionHash', hash => {
                            logger.debug(`Transaction hash: ${hash}`)
                            // Change db tx status to mining
```

```
                            db.changeTxStatusTo('mining', txId)
                            resolve(hash)
                        })
                        .once('confirmation', (confNumber, receipt) => {
                            const txHash = receipt.transactionHash
                            console.log(`Confiramation number: ${confNumber}`)
                            console.log(`Tx hash: ${txHash}`)

                            socket.emit('tx-confirmed', ({ userId, txHash }))

                            // Change db tx status to mined
                            db.changeTxStatusTo('mined', txId)
                        })
                        .once('error', reject)
                })
                .catch(reject)
        })

    const verifyTransfer = (rawTx, from, to, amount) =>
        checkSignedFunc({
            rawTx,
            from,
            func: 'transfer',
            params: ['address', 'uint256'],
            paramsCheck: [to, amount]
        })

    const transfers = userId => new Promise((resolve, reject) =>
        db.getUnpublishedTransfers(userId)
            .then(txs => {
                txs = txs.map(tx => ({
                    txId: tx.TransactionId,
                    fullName: tx.FullName,
                    direction: tx.Direction,
                    amount: `${tx.TransactionAmountWei / 10 ** 18}`,
                    createdAt: getDateTime(tx.CreatedAt)
                }))
                resolve(txs)
            })
            .catch(reject))

    module.exports = {
        balanceInAddress,
        balanceInContract,
        replenishContract,
        signTransfer,
        signTransferByUserId,
```

```
    verifyTransfer,
    publishTransfer,
    transfers,
}
```

services/auth.service.js

```
const { auth } = require('../db')
const { hash, verify } = require('../utils/crypto')
const { BreakPromiseChainError } = require('../errors')
const logger = require('../logger')

const register = (email, pass, firstName, middleName, lastName, birthDate,
    keywords, description, avatar) =>
    new Promise((resolve, reject) =>
        hash(pass)
            .then(passwordHash => {
                if (!firstName) firstName = undefined
                if (!middleName) middleName = undefined
                if (!lastName) lastName = undefined
                if (!birthDate) birthDate = undefined
                if (!description) description = undefined
                if (!avatar) avatar = undefined

                logger.debug({ passwordHash })

                auth.checkUniqueness(email)
                    .then(uniqueness => {
                        if (!uniqueness) {
                            resolve(0)
                            throw new BreakPromiseChainError()
                        }
                        return
                    })
                    .then(() => auth.registerUser(
                        email,
                        firstName,
                        middleName,
                        lastName,
                        birthDate,
                        description,
                        avatar,
                        passwordHash
                    ))
                    .then(res => {
                        const userId = res[0].UserId
                        logger.debug({ userId })
```

```
                        return auth.setUserKeywords(userId, keywords)
                    })
                    .then(userId => auth.createWallet(userId))
                    .then(({ userId, address, prKey }) =>
                        resolve({ userId, address, prKey }))
                    .catch(err => {
                        if (err.name !== 'BreakPromiseChainError') reject(err)
                    })
            }))

const login = ({ email, pass }) => new Promise((resolve, reject) =>
    auth.login(email)
        .then(res => {
            logger.debug({ note: 'DB auth.login', res })

            if (!res.length) resolve(false)
            if (!pass) resolve(false)

            verify(pass, res[0].PasswordHash)
                .then(result => {
                    logger.debug(result)
                    if (result) resolve(res[0].UserId)
                    else resolve(0)
                })
                .catch(err => reject(err))
        })
        .catch(err => reject(err)))

const updateUserData = ({ userId, keywords, description }) =>
    new Promise((resolve, reject) =>
        auth.setUserKeywords(userId, keywords)
            .then(userId => auth.setUserDescription(userId, description))
            .then(resolve)
            .catch(reject))

const myProfile = userId =>
    new Promise((resolve, reject) =>
        auth.myProfile(userId)
            .then(resolve)
            .catch(reject))

module.exports = {
    register,
    login,
    updateUserData,
    myProfile,
}
```

db/index.js

```
module.exports = {
    auth: require('./auth.db'),
    chat: require('./chat.db'),
    bc: require('./bc.db'),
}
```

db/pool.js

```
const { db } = require('../config')
const { Pool } = require('pg')

const pool = new Pool({
    user: db.user,
    host: db.host,
    database: db.database,
    password: db.password,
    port: db.port
})

pool.on('error', err => console.error(`Unexpected error on idle client ${err}`))

module.exports = pool
```

db/chat.db.js

```
const { query } = require('../utils/db')

const getPersonalChats = userId =>
    query(`
        select "User"."UserId", "Chat"."ChatId", "ChatType", "FromUser",
            "ToUser", "FirstName", "LastName", "AvatarBase64",
            "MessageText", "LastMsgs"."CreatedAt"
        from "ChatUser"
        inner join "Chat" on "Chat"."ChatId" = "ChatUser"."ChatId"
        inner join "User" on "User"."UserId" = "ChatUser"."UserId"

        join (
            select "tmp"."ChatId", "MessageText", "CreatedAt" from "ChatMessage"
            join (
            select "ChatId", max("ChatMessageId") as "ChatMessageId"
            from "ChatMessage"
            where "ChatId" in
                (select "ChatId" from "ChatUser" where "UserId" = $1)
            group by "ChatId"
            ) as "tmp" on "ChatMessage"."ChatMessageId" = "tmp"."ChatMessageId"
```

```
        ) as "LastMsgs" on "Chat"."ChatId" = "LastMsgs"."ChatId"

        where "Chat"."ChatId" in (
            select "ChatId" from "ChatUser" where "UserId" = $1
        )
            and "ChatType" != 'group'
            and "ChatUser"."UserId" != $1
        order by "LastMsgs"."CreatedAt" desc;`, [userId])

const getMessages = chatId =>
    query(`
        select "ChatMessageId", "ChatUser"."UserId", "MessageText", "IsRead",
            "CreatedAt"
        from "ChatMessage"
        join "ChatUser" on "ChatUserId" = "ChatMessage"."UserId"
        where "ChatMessage"."ChatId" = $1
        order by "CreatedAt";`, [chatId])

const getTotalAmount = chatId =>
    query(`
        select "TransactionAmountWei" from "Transaction"
        join (
            select "FromUser", "ToUser"
            from "Chat"
            where "ChatId" = $1
        ) as "tmp"
        on "FromUserId" = "tmp"."FromUser" and "ToUserId" = "tmp"."ToUser"
        where "TransactionStatus" = 'unpublished';`, [chatId])

const getUnreadMessages = (chatId, userId) =>
    query(`
        select "UserId", "MessageText", "CreatedAt"
        from "ChatMessage"
        where "ChatId" = $1 and "IsRead" = false and "UserId" != $2
        order by "CreatedAt";`, [chatId, userId])

const addMessage = (chatId, userId, text) =>
    query(`
        insert into "ChatMessage"("ChatId", "UserId", "MessageText")
        values (
            $1, (
                select "ChatUserId" from "ChatUser"
                where "ChatId" = $1 and "UserId" = $2
            ),
            $3
        ) returning "CreatedAt", "ChatMessageId";`, [chatId, userId, text])
```

```
const getMessageById = id =>
    query(`
        select "ChatId", "UserId", "MessageText", "CreatedAt"
        from "ChatMessage"
        where "ChatMessageId" = $1;`, [id])

const getChatUsers = chatId =>
    query('select "UserId" from "ChatUser" where "ChatId" = $1;', [chatId])

const readMessages = (chatId, userId) =>
    query(`
    update "ChatMessage"
    set "IsRead" = true
    where "ChatId" = $1 and "UserId" != $2;`, [chatId, userId])

module.exports = {
    getPersonalChats,
    getMessages,
    getTotalAmount,
    getUnreadMessages,
    addMessage,
    getMessageById,
    getChatUsers,
    readMessages,
}
```

db/bc.db.js

```
const { query } = require('../utils/db')

const getUserAddress = userId =>
    query('select "Address" from "Wallet" where "UserId" = $1', [userId])

const saveTransfer = (from, to, msgId, amount, tx) =>
    query(`
        insert into "Transaction" (
            "FromUserId",
            "ToUserId",
            "ForMessageId",
            "TransactionAmountWei",
            "RawTransaction"
        ) values ($1, $2, $3, $4, $5);`,
    [from, to, msgId, amount, tx])

const setAllTxsAsOutdated = (fromUser, toUser) =>
    query(`
        update "Transaction"
```

```
        set "TransactionStatus" = 'outdated'
        where "FromUserId" = $1 and
            "ToUserId" = $2 and
            "TransactionStatus" = 'unpublished';`, [fromUser, toUser])

const getRawTxById = txId =>
    query(`
        select "TransactionStatus", "RawTransaction"
        from "Transaction"
        where "TransactionId" = $1;`, [txId])

const lastUnpublishedTxAmountForChat = (from, to) =>
    query(`
        select "TransactionAmountWei" from "Transaction"
        where "FromUserId" = $1 and
              "ToUserId" = $2 and
              "TransactionStatus" = 'unpublished'
        order by "TransactionAmountWei"::bigint desc
        limit 1;`, [from, to])

const getUnpublishedTransfers = userId =>
    query(`
        select
        case
            when "FromUserId" = $1 then 'out'
            else 'in'
        end as "Direction",
        case
            when "FromUserId" = $1 then (
                select concat("FirstName", ' ', "MiddleName", ' ', "LastName")
                    as "FullName"
                from "User"
                where "UserId" = "Transaction"."ToUserId"
            )
            else (
                select concat("FirstName", ' ',
                              -- "MiddleName", ' ',
                              "LastName")
                    as "FullName"
                from "User"
                where "UserId" = "Transaction"."FromUserId"
            )
        end,
        "TransactionAmountWei", "CreatedAt", "TransactionId"
    from "Transaction"
    where ("FromUserId" = $1 or "ToUserId" = $1)
        and "TransactionStatus" = 'unpublished'
```

```
    order by "CreatedAt" desc;`, [userId])

const changeTxStatusTo = (status, txId) =>
    query(`
        update "Transaction"
        set "TransactionStatus" = $1
        where "TransactionId" = $2;`, [status, txId])

module.exports = {
    getUserAddress,
    setAllTxsAsOutdated,
    saveTransfer,
    getRawTxById,
    lastUnpublishedTxAmountForChat,
    getUnpublishedTransfers,
    changeTxStatusTo,
}
```

db/auth.db.js

```
const logger = require('../logger')
const { query } = require('../utils/db')
const { generateKeys } = require('../utils/bc')

const checkUniqueness = email => new Promise((resolve, reject) =>
    query('select * from "User" where "Email" = $1', [email])
        .then(users => {
            if (users.length) resolve(false)
            resolve(true)
        })
        .catch(err => reject(err)))

const registerUser = (email, firstName, middleName, lastName, birthDate,
    description, avatar, pass) =>
    query(`
        insert into "User" (
            "Email", "FirstName", "MiddleName", "LastName",
            "BirthDate", "Description", "AvatarBase64", "PasswordHash"
        ) values ($1, $2, $3, $4, $5, $6, $7, $8) returning "UserId"`,
    [email, firstName, middleName, lastName, birthDate, description, avatar,
        pass])

const setUserKeywords = (userId, keywords) => new Promise((resolve, reject) => {
    const queries = []

    keywords.forEach(keyword =>
        queries.push(query(`
```

```
            insert into "UserKeyword" ("UserId", "UserKeyword")
            values ($1, $2)`,
        [userId, keyword])))

    Promise.all(queries)
        .then(() => resolve(userId))
        .catch(err => reject(err))
})

const setUserDescription = (userId, description) =>
    query(`
        update "User" set "Description" = $1
        where "UserId" = $2;`, [description, userId])

const createWallet = userId => new Promise((resolve, reject) => {
    const [address, prKey] = generateKeys()
    logger.debug({ address, prKey })

    query(`insert into "Wallet" ("UserId", "Address")
        values ($1, $2) returning "Address"`, [userId, address])
        .then(() => resolve({ userId, address, prKey }))
        .catch(err => reject(err))
})

const login = email => query('select * from "User" where "Email" = $1', [email])

const myProfile = userId => query(`
    select "Email", "FirstName", "MiddleName", "LastName", "BirthDate",
    "Description", "AvatarBase64"
    from "User"
    where "UserId" = $1;`, [userId])

module.exports = {
    checkUniqueness,
    registerUser,
    setUserKeywords,
    setUserDescription,
    createWallet,
    login,
    myProfile,
}
```

utils/index.js

```
const constructSuffix = (diff, measurement, suffixes) => {
    const m = `${Math.floor(diff / measurement)}`
    const last = parseInt(m[m.length - 1])
```

```
    let suffix = ''
    if (last === 0)
        suffix = suffixes[0]
    else if (parseInt(m) >= 11 && parseInt(m) <= 19)
        suffix = suffixes[0]
    else if (last === 1)
        suffix = suffixes[1]
    else if (last <= 4)
        suffix = suffixes[2]
    else if (last > 4)
        suffix = suffixes[0]

    return `${m} ${suffix}`
}

const dateToLabel = date => {
    const curr = new Date()
    const diff = Math.floor((curr - date) / 1000) // difference in seconds

    const minute = 60
    const hour = 60 * minute
    const day = 24 * hour
    const week = 7 * day
    const month = 30 * day
    const year = 12 * month

    let label = ''
    if (diff < minute) {
        label = 'щойно'
    } else if (diff < hour)
        label = constructSuffix(
            diff, minute, ['хвилин', 'хвилину', 'хвилини'])
    else if (diff < day)
        label = constructSuffix(
            diff, hour, ['годин', 'годину', 'години'])
    else if (diff < week)
        label = constructSuffix(
            diff, day, ['днів', 'день', 'дні'])
    else if (diff < month)
        label = constructSuffix(
            diff, week, ['тижнів', 'тиждень', 'тижні'])
    else if (diff < year)
        label = constructSuffix(
            diff, month, ['місяців', 'місяць', 'місяці'])
    else
        label = constructSuffix(
```

```
            diff, year, ['років', 'рік', 'роки'])

    return label
}

const getDateTime = d => {
    const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
        'Sep', 'Oct', 'Nov', 'Dec']
    let res = `${d.getDate()} ${months[d.getMonth()]} ${d.getFullYear()}`
    res += ` ${d.getHours().toString().length === 1 ?
        '0' + d.getHours() : d.getHours()}:`
    res += `${d.getMinutes().toString().length === 1 ?
        '0' + d.getMinutes() : d.getMinutes()}:`
    res += `${d.getSeconds().toString().length === 1 ?
        '0' + d.getSeconds() : d.getSeconds()}`

    return res
}

module.exports = {
    db: require('./db'),
    crypto: require('./crypto'),
    bc: require('./bc'),
    dateToLabel,
    getDateTime,
}
```

utils/socket.js

```
const socketio = require('socket.io')
const logger = require('../logger')
const { verifySocketToken } = require('../middleware')

class Socket {

    constructor(server) {
        this.io = socketio(server)
        this.connections = {}
    }

    start() {
        this.io.use(verifySocketToken)

        this.io.on('connection', socket => {
            logger.debug(`
    New user
        userId: ${socket.userId}
```

```
            socketId: ${socket.id}`)
                this.connections[socket.userId] = socket.id
                logger.debug(this.connections)
            })

            this.io.on('disconnect', args => {
                logger.debug('A client is disconnected')
                console.log({ args })
            })
        }

        emit(event, params) {
            this.io.emit(event, params)
        }

        emitSmart(chatUsers, event, params) {
            logger.debug(this.connections)
            for (const userId in chatUsers) {
                this.io.to(this.connections[userId]).emit(event, params)
            }
        }
    }

    module.exports = Socket
```

utils/db.js

```
const pool = require('../db/pool')

const query = (queryString, params = []) => new Promise((resolve, reject) =>
    pool
        .connect()
        .then(client => client
            .query(queryString, params)
            .then(res => {
                client.release()
                resolve(res.rows)
            })
            .catch(err => {
                client.release()
                reject(err.stack)
            })))

module.exports = {
    query,
}
```

utils/crypto.js

```
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const config = require('../config')
const logger = require('../logger')

const hash = sha3 => new Promise((resolve, reject) =>
    bcrypt.hash(sha3, 13, (err, hash) => {
        if (err) reject(err)
        resolve(hash)
    }))

const verify = (sha3, hash) => new Promise((resolve, reject) => {
    bcrypt.compare(sha3, hash)
        .then(res => {
            logger.debug(`Bcrypt result: ${res}`)
            resolve(res)
        })
        .catch(err => reject(err))
})

const generateToken = (userId, expiresIn = config.jwt.expiresIn) =>
    new Promise((resolve, reject) =>
        jwt.sign({ userId }, config.jwt.key, { expiresIn }, (err, token) => {
            if (err) reject(err)
            logger.debug(`JWT token: ${token}`)
            resolve(token)
        }))

const verifyToken = token => new Promise((resolve, reject) =>
    jwt.verify(token, config.jwt.key, (err, decoded) => {
        if (err) reject(err)
        logger.debug({ decoded })
        resolve(decoded)
    }))

module.exports = {
    hash,
    verify,
    generateToken,
    verifyToken,
}
```

utils/bc.js

```
const abi = require('ethereumjs-abi')
const txDecoder = require('ethereum-tx-decoder')
```

```
const logger = require('../logger')
const { web3 } = require('../bc/singletons')
const { Web3Error } = require('../errors')

const toEth = amount => Math.round(amount / 10 ** 13) / 10 ** 5

const toWei = amount => parseInt(amount * 10 ** 18)

const generateKeys = () => {
    const { address, privateKey } = web3.eth.accounts.create()
    return [address, privateKey]
}

const txParams = (rawTx, types = []) => {
    const decodedTx = txDecoder.decodeTx(rawTx)

    const rawData = decodedTx.data
    const funcHex = rawData.slice(0, 10)
    const data = rawData.slice(10)

    return [funcHex, ...abi.rawDecode(types, Buffer.from(data, 'hex'))]
}

const verifySigner = (address, rawTx) => {
    const signer = web3.eth.accounts.recoverTransaction(rawTx)

    return signer === address
}

const checkSignedFunc = ({ rawTx, from, func, params, paramsCheck }) => {
    console.log({ rawTx, from, func, params, paramsCheck })
    const args = txParams(rawTx, params)
    logger.debug({ args })
    const signature = `${func}(${params.join(',')})`
    console.log({ signature })

    if (!verifySigner(from, rawTx)) return false
    // Is called a required function?
    if (web3.utils.sha3(signature).substr(0, 10) !== args[0]) return false
    if (paramsCheck[0].toLowerCase() !== `0x${args[1]}`) return false
    if (parseInt(paramsCheck[1]) !== parseInt(args[2].toString())) return false

    return true
}

const handleWeb3Error = err => {
```

```
    const [, type, vmType, info] = err.toString().split(':').map(x => x.trim())

    if (type !== 'Returned error')
        return new Web3Error(`Unknown error type: ${err}`)
    if (vmType !== 'VM Exception while processing transaction')
        return new Web3Error(`Unknown error VM type: ${vmType}`)

    return new Web3Error(info)
}

module.exports = {
    toEth,
    toWei,
    checkSignedFunc,
    handleWeb3Error,
    verifySigner,
    txParams,
    generateKeys,
}
```

config/index.js

```
require('dotenv').config()

const env = process.env.NODE_ENV || 'development'
const config = require(`./${env}`)

module.exports = config
```

config/test.js

```
module.exports = {
    env: 'test',
    db: {
        user: process.env.DEV_PGUSER,
        host: process.env.DEV_PGHOST,
        database: process.env.DEV_PGDATABASE,
        password: process.env.DEV_PGPASSWORD,
        port: process.env.DEV_PGPORT,
    },
    defaultUrl: process.env.DEFAULT_URL,
}
```

config/development.js

```
module.exports = {
    env: 'development',
    port: process.env.DEV_APP_PORT,
```

```
    db: {
        user: process.env.DEV_PGUSER,
        host: process.env.DEV_PGHOST,
        database: process.env.DEV_PGDATABASE,
        password: process.env.DEV_PGPASSWORD,
        port: process.env.DEV_PGPORT,
    },
    jwt: {
        key: process.env.DEV_JWT_PR_KEY,
        expiresIn: process.env.DEV_JWT_EXPIRE_TIME,
    },
    bc: {
        infura: process.env.DEV_INFURA_ROPSTEN,
        contractAddr: process.env.DEV_CONTRACT_ADDRESS
    },
}
```

config/production.js

```
module.exports = {
    env: 'production',
    db: {
        user: process.env.PGUSER,
        host: process.env.PGHOST,
        database: process.env.PGDATABASE,
        password: process.env.PGPASSWORD,
        port: process.env.PGPORT,
    },
    defaultUrl: process.env.DEFAULT_URL,
}
```

2 ПРОГРАМНИЙ КОД СМАРТ-КОНТРАКТУ

```
/**
 *Submitted for verification at Etherscan.io on 2020-01-27
*/

pragma solidity ^0.6.1;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
```

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * NOTE: This is a feature of the next version of OpenZeppelin Contracts.
     * @dev Get it via `npm install @openzeppelin/contracts@next`.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but
the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
```

```
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom
message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.

     * NOTE: This is a feature of the next version of OpenZeppelin Contracts.
     * @dev Get it via `npm install @openzeppelin/contracts@next`.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
     * Reverts when dividing by zero.
     *
```

```
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * NOTE: This is a feature of the next version of OpenZeppelin Contracts.
     * @dev Get it via `npm install @openzeppelin/contracts@next`.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

contract Micropayment {

    using SafeMath for uint256;

    event Transfer(address indexed from, address indexed to, uint256 value);

    mapping(address => uint256) private _balances;

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function deposit() public payable returns(bool) {
         _balances[msg.sender] = _balances[msg.sender].add(msg.value);
```

```
        return true;
    }

    function transfer(address recipient, uint256 amount) public returns(bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    function withdraw(uint256 amount) public returns(uint256) {
        uint256 balance = balanceOf(msg.sender);
        require(amount <= balance, "Withdrawal amount exceeds balance");
         _balances[msg.sender] = _balances[msg.sender].sub(amount);
        msg.sender.transfer(amount);

        return _balances[msg.sender];
    }

    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "Transfer from the zero address");
        require(recipient != address(0), "Transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "Transfer amount exceeds
balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

}
```

3 ПРОГРАМНИЙ КОД IPADOS ДОДАТКУ

ViewControllers\LoginViewController.swift

```
//
//  LoginViewController.swift
//  crypto-chat
//
//  Created by Kushka Misha on 2/4/20.
//  Copyright © 2020 Misha Kushka. All rights reserved.
//

import UIKit
import CryptoKit
import Alamofire
import Loaf

struct Login: Encodable {
    let email: String
    let pass: String
}

class LoginViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var emailInputLabel: UIView!
    @IBOutlet weak var passInputLabel: UIView!
    @IBOutlet weak var signInBigLabel: UILabel!
    @IBOutlet weak var emailInputField: UITextField!
    @IBOutlet weak var passInputField: UITextField!

    @IBAction func showSignUpScreen(_ sender: Any) { navigateToScreen(screenName:
"SignUpScreen")}

    override func viewDidLoad() {
        super.viewDidLoad()

        emailInputLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        passInputLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        signInBigLabel.transform = CGAffineTransform(rotationAngle: -CGFloat.pi / 2)
        emailInputField.attributedPlaceholder = NSAttributedString(string:
"email@example.com", attributes: [NSAttributedString.Key.foregroundColor:
UIColor.init(displayP3Red: 255/255, green: 255/255, blue: 255/255, alpha: 0.5)])
        passInputField.attributedPlaceholder = NSAttributedString(string: "password",
attributes: [NSAttributedString.Key.foregroundColor: UIColor.init(displayP3Red:
255/255, green: 255/255, blue: 255/255, alpha: 0.5)])
//        emailInputField.text = "bob"
//        passInputField.text = "bob"
```

```
        passInputField.delegate = self
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        print("enter is pressed")
        signIn()
        return true
    } // By "Enter" press...

    @IBAction func signIn(_ sender: Any) { self.signIn() }

    func signIn() {
        guard let data = passInputField.text?.data(using: .utf8) else { return }
        let passHash = SHA512.hash(data: data).hexStr
        debugPrint(passHash)
        let login = Login(email: emailInputField.text ?? "", pass: passHash)

        AF.request("http://localhost:8080/auth/login",
                   method: .post,
                   parameters: login,
                   encoder: JSONParameterEncoder.default).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let dict = data as! NSDictionary
                    let status = dict["status"] as! String
                    print(dict)
                    if (status == "success") {
                        // Save as global variables
                        jwt = dict["token"] as! String
                        userId = "\(dict["userId"]!)"

                        let vc = UIStoryboard.init(name: "Main", bundle:
Bundle.main).instantiateViewController(withIdentifier: "MessagesScreen") as!
ChatViewController
                        vc.modalPresentationStyle = .fullScreen
                        self.present(vc, animated: true, completion: nil)
                    } else {
                        Loaf(NSLocalizedString("loginError", comment: ""), state:
.error, sender: self).show()
                    }
                case .failure(let error):
                    Loaf(NSLocalizedString("connectionError", comment: ""), state:
.error, sender: self).show()
                    print(error.localizedDescription)
            }
        }
```

```
    }

}
```

ViewControllers\SignUpViewController.swift

```
import UIKit

class SignUpViewController: UIViewController {

    @IBOutlet weak var emailLabel: UIView!
    @IBOutlet weak var firstNameLabel: UIView!
    @IBOutlet weak var middleNameLabel: UIView!
    @IBOutlet weak var lastNameLabel: UIView!
    @IBOutlet weak var passLabel: UIView!
    @IBOutlet weak var repeatPassLabel: UIView!
    @IBOutlet weak var birthDateLabel: UIView!

    @IBOutlet weak var emailInputField: UITextField!
    @IBOutlet weak var firstNameInputField: UITextField!
    @IBOutlet weak var middleNameInputField: UITextField!
    @IBOutlet weak var lastNameInputField: UITextField!
    @IBOutlet weak var passInputField: UITextField!
    @IBOutlet weak var repeatPassInputField: UITextField!
    @IBOutlet weak var birthDateInputField: UITextField!

    @IBOutlet weak var emailGlowingView: UIView!
    @IBOutlet weak var passGlowingView: UIView!
    @IBOutlet weak var repeatPassGlowingView: UIView!

    @IBOutlet weak var userImage: UIImageView!
    @IBOutlet weak var uploadUserPhotoButton: UIButton!

    let imagePicker = UIImagePickerController()

    override var preferredStatusBarStyle : UIStatusBarStyle {
        return UIStatusBarStyle.lightContent
        //return UIStatusBarStyle.default   // Make dark again
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        setStatusBarBackgroundColor(color : purple)
        self.view.backgroundColor = purple
        imagePicker.delegate = self
        setupInputFields()
```

```
    }

    @IBAction func showNextScreen(_ sender: Any) { if checkInputData() {
registerUser() } }

    @IBAction func uploadUserPhoto(_ sender: Any) { chooseImageFromDevice() }

}
```

ViewControllers\SignUp2ViewController.swift

```
import Loaf
import UIKit
import SwiftKeychainWrapper

class SignUp2ViewController: UIViewController {

    @IBOutlet weak var keywordsCollectionView: UICollectionView!
    @IBOutlet weak var descriptionTitleView: UIView!
    @IBOutlet weak var ethereumAddrTitleView: UIView!
    @IBOutlet weak var copyEthAddrButton: UIButton!
    @IBOutlet weak var descriptionTextBox: UITextView!
    @IBOutlet weak var ethAddrTextField: UITextField!
    @IBOutlet weak var qrCodeImage: UIImageView!
    @IBOutlet weak var qrCodeView: UIView!

    let keywords: [String] = ["consulting", "blockchain", "smart contracts",
"teaching", "+"]
    let descriptionText = NSLocalizedString("yourDescription", comment: "")

    var userId: String = ""
    var address: String = ""
    var token: String = ""

    override var preferredStatusBarStyle : UIStatusBarStyle {
        return UIStatusBarStyle.lightContent
        //return UIStatusBarStyle.default   // Make dark again
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setStatusBarBackgroundColor(color : purple)
        self.view.backgroundColor = purple

        print(userId)
        print(address)
        print(token)
```

```
        guard let prKey: String = KeychainWrapper.standard.string(forKey: "prKey")
else { return }
        print(prKey)

        keywordsCollectionView.backgroundColor = purple

        qrCodeImage.image = generateQRCode(from: "eth:\(address)")
        setupInputFields()
    }

    @IBAction func copyEthAddr(_ sender: Any) {
        UIPasteboard.general.string = ethAddrTextField.text
        Loaf("Copied to the clipboard", state: .success, sender: self).show()
    }

    @IBAction func updateUserData(_ sender: Any) {
        finishUserSignUp()
    }


}
```

ViewControllers\ChatViewController.swift

```
import UIKit
import SwiftKeychainWrapper


struct LoadChats: Encodable {
    let userId: Int
}

class ChatViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var searchUserBar: UISearchBar!
    @IBOutlet weak var chatsTableView: UITableView!
    @IBOutlet weak var msgsTableView: UITableView!
    @IBOutlet weak var noChatsSelectedLabel: UILabel!
    @IBOutlet weak var noChatsSelectedView: UIView!

    @IBOutlet weak var topBarView: UIView!
    @IBOutlet weak var topBarUsername: UILabel!
    @IBOutlet weak var topBarAvatar: UIImageView!
    @IBOutlet weak var topBarChatType: UIImageView!
    @IBOutlet weak var topBarVideoCall: UIImageView!
    @IBOutlet weak var topBarAudioCall: UIImageView!
    @IBOutlet weak var topBarEthereum: UILabel!
```

```
    @IBOutlet weak var sendMessageTextField: UITextField!
    @IBOutlet weak var sendMessageButton: UIButton!
    @IBOutlet weak var sendMessageView: UIView!

    var username: String = ""
    var selectedChat: Chat = Chat(chatId: "", socketId: "", name: "", chatType: "",
fromUser: "", avatar: UIImage(), lastMsgText: "", lastMsgTime: "", chatTypeImage:
UIImage(), chatTypeSelectedImage: UIImage())
    var chats: [Chat] = []
    var filteredChats: [Chat] = []
    var msgs: [Message] = []
    var chatSelected: Bool = false
    let CHARACTER_PRICE = 0.00001 // ETH = about $0.001

    var isSearchBarEmpty: Bool {
        return searchUserBar.text?.isEmpty ?? true
    }

    var isFiltering: Bool {
        return !isSearchBarEmpty
    }

    lazy var refreshControl: UIRefreshControl = {
        let refreshControl = UIRefreshControl()
        refreshControl.addTarget(self, action:
            #selector(ChatViewController.handleRefresh(_:)),
                                 for: UIControl.Event.valueChanged)
        refreshControl.tintColor = UIColor.init(red: 122/255, green: 140/255, blue:
255/255, alpha: 1)

        return refreshControl
    }()

    @objc func handleRefresh(_ refreshControl: UIRefreshControl) {
//        establishSocketConnection()
        loadChats()
        refreshControl.endRefreshing()
    }

    @IBAction func goToSettings(_ sender: Any) {
        let vc = UIStoryboard.init(name: "Main", bundle:
Bundle.main).instantiateViewController(withIdentifier: "SettingsScreen") as!
SettingsViewController
        self.present(vc, animated: true, completion: nil)
    }
```

```
    override var preferredStatusBarStyle : UIStatusBarStyle {
        return UIStatusBarStyle.lightContent
        //return UIStatusBarStyle.default   // Make dark again
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setStatusBarBackgroundColor(color : purple)
        topBarView.backgroundColor = purple
        topBarAvatar.cornerRadius = 25

        searchUserBar.backgroundImage = UIImage()
        searchUserBar.searchBarStyle = UISearchBar.Style.prominent
        searchUserBar.searchTextField.backgroundColor = UIColor.white
        searchUserBar.placeholder = NSLocalizedString("searchForAUser", comment: "")
        searchUserBar.delegate = self

        chatsTableView.delegate = self
        chatsTableView.dataSource = self
        msgsTableView.delegate = self
        msgsTableView.dataSource = self
        sendMessageTextField.delegate = self

        noChatsSelectedLabel.text = NSLocalizedString("noChatsSelected", comment: "")
        noChatsSelectedLabel.backgroundColor = purple
        noChatsSelectedLabel.padding = UIEdgeInsets(top: 10, left: 10, bottom: 10,
right: 10)
        noChatsSelectedLabel.layer.masksToBounds = true
        noChatsSelectedLabel.layer.cornerRadius = 10

        msgsTableView.rowHeight = UITableView.automaticDimension
        chatsTableView.rowHeight = 76

        sendMessageTextField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("typeYourMsgHere", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: UIColor.init(displayP3Red: 255/255, green:
255/255, blue: 255/255, alpha: 0.5)])

        self.chatsTableView.addSubview(self.refreshControl)

        isChatSelected(status: false)
        establishSocketConnection()
        listen4NewMessages()
        listen4AmountChanges()
        loadChats()
    }
```

```
    func textFieldShouldReturn(_ textField: UITextField) -> Bool { sendMsg(); return
true } // By "Enter" press...
    @IBAction func sendMsg(_ sender: Any) { sendMsg() } // By send button press...

    func sendMsg() {
        let msg = sendMessageTextField.text!

        if (msg != "") {
            sendMessage(message: msg)

            // Current time
            let date = Date()
            let calendar = Calendar.current
            let hour = calendar.component(.hour, from: date)
            let minutes = calendar.component(.minute, from: date)

            msgs.append(Message(msgId: "", userId: userId, msg: msg, isRead: false,
time: "\(hour):\(minutes)"))

            sendMessageTextField.text = ""
            updateMessages()
        }
    }

}
```

ViewControllers\SettingsViewController.swift

```
import UIKit

class SettingsViewController: UIViewController {

    @IBOutlet weak var avatar: UIImageView!
    @IBOutlet weak var fullName: UILabel!
    @IBOutlet weak var birthDate: UILabel!

    @IBOutlet weak var keywordsCollectionView: UICollectionView!
    @IBOutlet weak var descriptionTitleView: UIView!
    @IBOutlet weak var descriptionTextView: UITextView!
    @IBOutlet weak var langSegmentedControl: UISegmentedControl!
    @IBOutlet weak var txTableView: UITableView!
    @IBOutlet weak var userBalance: UIButton!

    @IBOutlet weak var headerDate: UILabel!
    @IBOutlet weak var headerAmount: UILabel!

    let descriptionText = NSLocalizedString("noDescription", comment: "")
```

```
    let keywords: [String] = ["consulting", "blockchain", "smart contracts",
"teaching", "+"]
    let languages = [["Українська", "uk"], ["English", "en"]]
    let months = ["january", "february", "march", "april", "may", "june", "july",
"august", "september", "october", "november", "december"]

    var currentLangCode: String = Locale.current.languageCode ?? ""
    var txs: [Tx] = []
    var selectedTxIndex: Int = -1

    lazy var refreshControl: UIRefreshControl = {
        let refreshControl = UIRefreshControl()
        refreshControl.addTarget(self, action:
            #selector(ChatViewController.handleRefresh(_:)),
                                 for: UIControl.Event.valueChanged)
        refreshControl.tintColor = white

        return refreshControl
    }()

    @objc func handleRefresh(_ refreshControl: UIRefreshControl) {
        self.getUnpublishedTxs()
        refreshControl.endRefreshing()
    }

    override var preferredStatusBarStyle : UIStatusBarStyle {
        return UIStatusBarStyle.lightContent
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        let rightSwipe = UISwipeGestureRecognizer(target: self, action:
#selector(rightSwipeAction(swipe:)))
        rightSwipe.direction = UISwipeGestureRecognizer.Direction.right
        self.view.addGestureRecognizer(rightSwipe)

        setStatusBarBackgroundColor(color : purple)
        self.view.backgroundColor = purple

        listenForGlobalSocketMsgs()
        setupInputFields()
        getUnpublishedTxs()
    }

    @IBAction func changeLang(_ sender: Any) {
```

```
        if currentLangCode != languages[langSegmentedControl.selectedSegmentIndex][1]
{
            print("The language was changed")

            // Alert to restart app to change the app language
            let alertController = UIAlertController(title:
NSLocalizedString("langChange", comment: ""), message:
NSLocalizedString("langChangeDescr", comment: ""), preferredStyle: .alert)
            let okAction = UIAlertAction(title: NSLocalizedString("yes", comment: ""),
style: UIAlertAction.Style.default) {
                UIAlertAction in
                NSLog("OK Pressed")

                self.currentLangCode =
self.languages[self.langSegmentedControl.selectedSegmentIndex][1]
                UserDefaults.standard.set([self.currentLangCode], forKey:
"AppleLanguages")
                UserDefaults.standard.synchronize()

                exit(0)
            }
            let cancelAction = UIAlertAction(title: NSLocalizedString("restartLater",
comment: ""), style: UIAlertAction.Style.cancel) {
                UIAlertAction in
                self.selectCorrectLang()
                NSLog("Cancel Pressed")
            }
            alertController.addAction(okAction)
            alertController.addAction(cancelAction)
            self.present(alertController, animated: true, completion: nil)
        }
        print(currentLangCode)
    }

    @IBAction func publishTx(_ sender: Any) {
        if (self.selectedTxIndex != -1) {
            let alertController = UIAlertController(title:
NSLocalizedString("attention", comment: ""), message: NSLocalizedString("publishTx",
comment: ""), preferredStyle: .alert)
            let okAction = UIAlertAction(title: NSLocalizedString("publish", comment:
""), style: UIAlertAction.Style.default) {
                UIAlertAction in
                // Publish tx
                self.publishTransaction(tx: self.txs[self.selectedTxIndex])
                print("OK Pressed")
            }
```

```
            let cancelAction = UIAlertAction(title: NSLocalizedString("cancel",
comment: ""), style: UIAlertAction.Style.cancel) {
                UIAlertAction in
                print("Cancel Pressed")
            }
            alertController.addAction(okAction)
            alertController.addAction(cancelAction)
            self.present(alertController, animated: true, completion: nil)
        }
    }

    @IBAction func refreshUserBalance(_ sender: Any) {
        self.loadUserBalance()
    }

    @objc func rightSwipeAction(swipe: UISwipeGestureRecognizer) {
        NSLog("right swipe action")

        let vc = UIStoryboard.init(name: "Main", bundle:
Bundle.main).instantiateViewController(withIdentifier: "MessagesScreen") as!
ChatViewController
        self.present(vc, animated: true, completion: nil)
    }
}
```

ViewControllers\ProductInfoViewController.swift

```
import UIKit

class ProductInfoViewController: UIViewController {

    @IBOutlet weak var githubUrl: UIButton!

    override var preferredStatusBarStyle : UIStatusBarStyle {
        return UIStatusBarStyle.lightContent
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func githubOpen(_ sender: Any) {
        if let url = URL(string: "https://www.github.com/kushkamisha") {
            UIApplication.shared.open(url)
        }
    }
```

```
}
```

CustomElements/InputBox.swift

```
//
//  InputBox.swift
//  crypto-chat
//
//  Created by Kushka Misha on 2/4/20.
//  Copyright © 2020 Misha Kushka. All rights reserved.
//

import UIKit

class InputBox: UIView {
    //initWithFrame to init view from code
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    //initWithCode to init view from xib or storyboard
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    //common func to init our view
    private func setupView() {
        backgroundColor = UIColor(red: 122, green: 140, blue: 255, alpha: 0)

        cornerRadius = 10
        layer.borderColor = UIColor.white.cgColor
        layer.borderWidth = 1
    }
}
```

CustomElements/MyButton.swift

```
import Foundation
import UIKit

class MyButton: UIButton {
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }
```

```
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    private func setupView() {
        backgroundColor = UIColor.white
        cornerRadius = 10
    }
}
```

CustomElements/Colors.swift

```
import UIKit

let translucentWhite = UIColor.init(displayP3Red: 255/255, green: 255/255, blue:
255/255, alpha: 0.5)
let purple = UIColor.init(red: 122/255, green: 140/255, blue: 255/255, alpha: 1)
let white = UIColor.white
let black = UIColor.black
```

Objects/Chat.swift

```
import Foundation
import UIKit

class Chat {

    var chatId: String
    var socketId: String
    var name: String
    var chatType: String
    var fromUser: String
    var avatar: UIImage
    var lastMsgText: String
    var lastMsgTime: String
    var chatTypeImage: UIImage
    var chatTypeSelectedImage: UIImage

    init (chatId: String, socketId: String, name: String, chatType: String, fromUser:
String, avatar: UIImage, lastMsgText: String, lastMsgTime: String, chatTypeImage:
UIImage, chatTypeSelectedImage: UIImage) {
        self.chatId = chatId
        self.socketId = socketId
        self.name = name
```

```
        self.chatType = chatType
        self.fromUser = fromUser
        self.avatar = avatar
        self.lastMsgText = lastMsgText
        self.lastMsgTime = lastMsgTime
        self.chatTypeImage = chatTypeImage
        self.chatTypeSelectedImage = chatTypeSelectedImage
    }
}
```

Objects/Message.swift

```
import Foundation

class Message {

    var msgId: String
    var userId: String
    var msg : String
    var isRead: Bool
    var time : String

    init(msgId: String, userId: String, msg: String, isRead: Bool, time: String) {
        self.msgId = msgId
        self.userId = userId
        self.msg = msg
        self.isRead = isRead
        self.time = time
    }
}
```

Objects/UserData.swift

```
import UIKit

class UserData {

    var email: String
    var pass: String
    var firstName: String
    var middleName: String
    var lastName: String
    var birthDate: String
    var photo: UIImage
    var address: String
    var prKey: String
```

```
    init(
        email: String,
        pass: String,
        firstName: String = "",
        middleName: String = "",
        lastName: String = "",
        birthDate: String = "",
        address: String = "",
        prKey: String = "",
        photo: UIImage = UIImage()
    ) {
        self.email = email
        self.pass = pass
        self.firstName = firstName
        self.middleName = middleName
        self.lastName = lastName
        self.birthDate = birthDate
        self.photo = photo
        self.address = address
        self.prKey = prKey
    }
}
```

Objects/Tx.swift

```
class Tx {

    var id: String = ""
    var date: String = ""
    var userName: String = ""
    var direction: String = ""
    var amount: String = ""

    init(id: String, date: String, userName: String, direction: String, amount:
String) {
        self.id = id
        self.date = date
        self.userName = userName
        self.direction = direction
        self.amount = amount
    }
}
```

Services/Socket.swift

```
import Foundation
```

```
import SocketIO

class Socket {

    var socket: SocketIOClient!
    static var manager: SocketManager?
    var token: String
    var isConnected: Bool = false

    init(token: String) {
        self.token = token
        Socket.self.manager = SocketManager(socketURL: URL(string:
"http://localhost:8080")!, config: [.log(false), .compress, .connectParams(["token":
token])])
    }

    func connect() {
        self.socket = Socket.self.manager?.defaultSocket

        self.socket.on(clientEvent: .connect) { data, ack in
            print("connected to external server")
            self.isConnected = true
        }

        self.socket.on(clientEvent: .disconnect) { data, ack in
            self.isConnected = false
        }

        self.socket.onAny { data in
            print("\n\ndata")
            print(data)
        }

        self.socket.connect()
    }

    func socketOn(event: String, callback success: @escaping (_ data: Any) -> Void) {
        self.socket.on(event) { (dataArr, ack) in
            print(dataArr) // don't remove. Otherwise messages will be added multiple
times to the msgs list
            success(dataArr)
        }
    }

}
```

Services/ChatTables.swift

```
import UIKit

extension ChatViewController: UITableViewDataSource, UITableViewDelegate {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        if (tableView == msgsTableView) {
            return msgs.count
        } else {
            if isFiltering { return filteredChats.count }
            return chats.count
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        if (tableView == chatsTableView) {
            let cell = tableView.dequeueReusableCell(withIdentifier: "ChatCell") as!
ChatCell
            let chat: Chat
            if isFiltering {
                chat = filteredChats[indexPath.row]
            } else {
                chat = chats[indexPath.row]
            }
            cell.setChat(chat: chat)
            cell.selectionColor = purple

            return cell
        }

        let msg = msgs[indexPath.row]
        if (msg.userId == userId) {
            let cell = tableView.dequeueReusableCell(withIdentifier:
"MessageRightCell") as! MessageRightCell
            cell.setMessage(message: msg)
            return cell
        } else {
            let cell = tableView.dequeueReusableCell(withIdentifier:
"MessageLeftCell") as! MessageLeftCell
            cell.setMessage(message: msg)
            return cell
        }
    }
```

```
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        if (tableView == chatsTableView) {
            self.isChatSelected(status: true)
            self.selectedChat = isFiltering ? filteredChats[indexPath.row] :
chats[indexPath.row]

            print(self.selectedChat.name)
            print(self.selectedChat.chatId)

            self.topBarEthereum.isHidden = self.selectedChat.chatType != "paying" ?
true : false

            getMessages()

            topBarUsername.text = self.selectedChat.name
            topBarAvatar.image = self.selectedChat.avatar
            topBarChatType.image = self.selectedChat.chatTypeSelectedImage
        }
    }

    func updateMessages() {
        self.msgsTableView.reloadData()
        if self.msgs.count > 0 {
            self.msgsTableView.scrollToRow(at: IndexPath(item:self.msgs.count-1,
section: 0), at: .bottom, animated: false)
        }
    }

}

extension ChatViewController: UISearchBarDelegate {

    func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
        filter(searchText, category: searchBar.selectedScopeButtonIndex)
    }

    func filter(_ searchText: String, category: Int = 0) {
        filteredChats = chats.filter({ (chat: Chat) -> Bool in
            return chat.name.lowercased().contains(searchText.lowercased())
        })
        chatsTableView.reloadData()
    }
}

Services/ChatHelper.swift
//
//  http.ext.swift
```

```
//  crypto-chat
//
//  Created by Kushka Misha on 2/24/20.
//  Copyright © 2020 Misha Kushka. All rights reserved.
//

import Alamofire
import SwiftyJSON
import UIKit
import SwiftKeychainWrapper

extension ChatViewController {

    func establishSocketConnection() {
        // Establish a socket connection
        socket = Socket.init(token: jwt)
        socket.connect()
    }

    func listen4NewMessages() {
        listenForGlobalSocketMsgs()
        socket.socket.on("new-message") { data, ack in
            print("event new-message")
            let message = JSON(data)[0]
            print(message)
            if message["userId"].stringValue != userId &&
message["chatId"].stringValue == self.selectedChat.chatId {
                self.msgs.append(Message(
                    msgId: message["msgId"].stringValue,
                    userId: message["userId"].stringValue,
                    msg: message["message"].stringValue,
                    isRead: message["isRead"].boolValue,
                    time: message["createdAt"].stringValue
                ))

                if self.selectedChat.chatType == "paying" {
                    self.topBarEthereum.text = "\(message["amount"].stringValue) ETH"
                }

                self.pay4Msgs()
                self.updateMessages()
            }
        }
    }

    func listen4AmountChanges() {
        socket.socket.on("upd-amount") { data, ack in
```

```
            print("event upd-amount")
            let event = JSON(data)[0]

            if userId == event["toUserId"].stringValue &&
                self.selectedChat.fromUser == event["fromUserId"].stringValue &&
                self.selectedChat.chatType == "paying"
            {
                self.topBarEthereum.text = "\(event["amount"].stringValue) ETH"
            }
        }
    }

    func loadChats() {
        // Send request to load chats to the server
        AF.request("http://localhost:8080/chat/chatList",
                   parameters: ["token": jwt]).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let chats = JSON(data)["chats"]
                    self.chats = []
                    for (_, chat) in chats {
                        var chatTypeImage = UIImage()
                        var chatTypeSelectedImage = UIImage()

                        switch(chat["chatType"].stringValue) {
                            case "free":
                                chatTypeImage =  imageLiteral(resourceName: "free")
                                chatTypeSelectedImage =  imageLiteral(resourceName:
"free white")
                                break
                            case "locked":
                                chatTypeImage =  imageLiteral(resourceName: "locked-
purple")
                                chatTypeSelectedImage =  imageLiteral(resourceName:
"locked-white")
                                break
                            case "unlocked":
                                chatTypeImage =  imageLiteral(resourceName:
"unlocked")
                                chatTypeSelectedImage =  imageLiteral(resourceName:
"unlocked white")
                                break
                            case "paying":
                                if userId == chat["fromUser"].stringValue {
                                    chatTypeImage =  imageLiteral(resourceName:
"ethereum-out-purple")
```

```
                                    chatTypeSelectedImage =
imageLiteral(resourceName: "ethereum-out-white")
                                } else {
                                    chatTypeImage =  imageLiteral(resourceName:
"ethereum-in-purple")
                                    chatTypeSelectedImage =
imageLiteral(resourceName: "ethereum-in-white")
                                }
                                break
                            case "group":
                                break
                            default:
                                break
                        }

                        let name = "\(chat["firstName"].stringValue)
\(chat["lastName"].stringValue)"
                        let imageBase64String = chat["avatar"].stringValue
                        let imageData = Data(base64Encoded: imageBase64String)
                        let avatar = UIImage(data: imageData ?? Data())

                        let arr =
chat["lastMsgTime"].stringValue.components(separatedBy: " ")
                        let timeLabel = arr.count == 2 ? "\(arr[0])
\(NSLocalizedString(arr[1], comment: ""))" : NSLocalizedString(arr[0], comment: "")

                        self.chats.append(Chat(
                            chatId: chat["chatId"].stringValue,
                            socketId: "",
                            name: name,
                            chatType: chat["chatType"].stringValue,
                            fromUser: chat["fromUser"].stringValue,
                            avatar: avatar ??  imageLiteral(resourceName: "user-
default"),
                            lastMsgText: chat["lastMsgText"].stringValue,
                            lastMsgTime: timeLabel,
                            chatTypeImage: chatTypeImage,
                            chatTypeSelectedImage: chatTypeSelectedImage
                        ))
                    }
                    self.chatsTableView.reloadData()
                    break
                case .failure(let error):
                    print(error.localizedDescription)
            }
        }
    }
```

```
func getMessages() {
    AF.request("http://localhost:8080/chat/messages",
               parameters: [
                   "token": jwt,
                   "chatId": self.selectedChat.chatId
               ]).responseJSON { response in
        switch response.result {
            case .success(let data):
                let json = JSON(data)
                let msgs = json["messages"]
                self.msgs = []
                for (_, msg) in msgs {
                    if (msg["text"].stringValue != "") {
                        self.msgs.append(Message(
                            msgId: msg["msgId"].stringValue,
                            userId: msg["userId"].stringValue,
                            msg: msg["text"].stringValue,
                            isRead: msg["isRead"].boolValue,
                            time: msg["time"].stringValue
                        ))
                    }
                }

                let amount = json["amount"].stringValue
                if (self.selectedChat.chatType == "paying" && amount != "") {
                    self.topBarEthereum.text = "\(amount) ETH"
                }

                self.pay4Msgs()
                self.updateMessages()

                break
            case .failure(let error):
                print(error.localizedDescription)
        }
    }
}

func pay4Msgs() {
    let chatType = self.selectedChat.chatType
    let fromUserId = self.selectedChat.fromUser
    if (chatType == "paying" && fromUserId == userId) {
        print("paying chat")

        var totalAmount = 0.0
        var lastMsgId = ""
```

```
            var lastUserId = ""
            for msg in self.msgs {
                if (msg.userId != userId && !msg.isRead) {
                    print("msg.msg: \(msg.msg)")
                    msg.isRead = true
                    totalAmount += Double(msg.msg.count) * self.CHARACTER_PRICE
                    lastMsgId = msg.msgId
                    lastUserId = msg.userId
                }
            }
            // pay for others messages
            sendMicrotx(toUser: lastUserId, amount: totalAmount, msgId: lastMsgId)
        } else {
            self.readMsgs()
        }
    }

    func readMsgs() {
        print("read messages")
        AF.request("http://localhost:8080/chat/readMessages",
                   method: .post,
                   parameters: ["token": jwt, "chatId": self.selectedChat.chatId],
                   encoder: JSONParameterEncoder.default).responseJSON { status in
            print(status)
        }
    }

    func sendMessage(message: String) {
        AF.request("http://localhost:8080/chat/message",
                   method: .post,
                   parameters: ["token": jwt, "chatId": self.selectedChat.chatId,
"message": message],
                   encoder: JSONParameterEncoder.default).responseJSON { status in
            print(status)
            self.loadChats()
        }
    }

    func sendMicrotx(toUser: String, amount: Double, msgId: String) {
        print("sending microtx...")
        // Get private key from keychain
        let saveSuccessful: Bool =
KeychainWrapper.standard.set("0x2ac32c0d5a30db63270d6178c5b03a338d61de25d973732654bd65
738af14171", forKey: "prKey")
        print("Saving prKey to keychain: \(saveSuccessful)")
        guard let prKey: String = KeychainWrapper.standard.string(forKey: "prKey")
else { return }
```

```
        print(prKey)
        print("message id: \(msgId)")

        AF.request("http://localhost:8080/bc/signTransferByUserId",
                   method: .post,
                   parameters: [
                       "token": jwt,
                       "msgId": msgId,
                       "toUserId": toUser,
                       "amount": String(amount * 1000000000000000000),
                       "prKey": prKey
                   ],
                   encoder: JSONParameterEncoder.default).responseJSON { response in
            switch response.result {
                case .success(let data):
                    print("the mtx is sent")
                    let json = JSON(data)
                    let rawTx = json["rawTx"].stringValue
                    let totalAmountWei = json["totalAmount"].intValue
                    let totalAmountEth: Double = Double(totalAmountWei) /
1000000000000000000.0
                    print("rawTx: \(rawTx)")
                    print("totalAmountEth: " + String(format: "%.5f", totalAmountEth))
                    self.topBarEthereum.text = String(format: "%.5f", totalAmountEth)
+ " ETH"
                    self.readMsgs()
                    break
                case .failure(let error):
                    print(error.localizedDescription)
            }
        }
    }

    func isChatSelected(status: Bool) {
        if (status) {
            sendMessageView.isHidden = false
            topBarVideoCall.isHidden = false
            topBarAudioCall.isHidden = false
            topBarEthereum.isHidden = false
            msgsTableView.isHidden = false
            noChatsSelectedView.isHidden = true
        } else {
            sendMessageView.isHidden = true
            topBarVideoCall.isHidden = true
            topBarAudioCall.isHidden = true
            topBarEthereum.isHidden = true
            msgsTableView.isHidden = true
```

```
            noChatsSelectedView.isHidden = false
        }
    }

}
```

Services/SignUpHelper.swift

```
import UIKit
import CryptoKit
import Loaf
import Alamofire
import SwiftyJSON
import SwiftKeychainWrapper

extension SignUpViewController: UINavigationControllerDelegate,
UIImagePickerControllerDelegate, UITextFieldDelegate {

    func setupInputFields() {
        emailLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        firstNameLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        middleNameLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        lastNameLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        passLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        repeatPassLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)
        birthDateLabel.roundCorners(corners: [.topLeft, .bottomLeft], radius: 10)

        let translucentWhite = UIColor.init(displayP3Red: 255/255, green: 255/255,
blue: 255/255, alpha: 0.5)
        emailInputField.attributedPlaceholder = NSAttributedString(string:
"john@mail.com", attributes: [NSAttributedString.Key.foregroundColor:
translucentWhite])
        firstNameInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("john", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])
        middleNameInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("james", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])
        lastNameInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("doe", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])
        passInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("password", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])
        repeatPassInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("password", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])
```

```
        birthDateInputField.attributedPlaceholder = NSAttributedString(string:
NSLocalizedString("mm/dd/yyyy", comment: ""), attributes:
[NSAttributedString.Key.foregroundColor: translucentWhite])

        emailInputField.delegate = self
        passInputField.delegate = self
        repeatPassInputField.delegate = self
        emailInputField.addTarget(self, action:
#selector(SignUpViewController.removeGlowing(_:)), for: .editingChanged)
        passInputField.addTarget(self, action:
#selector(SignUpViewController.removeGlowing(_:)), for: .editingChanged)
        repeatPassInputField.addTarget(self, action:
#selector(SignUpViewController.removeGlowing(_:)), for: .editingChanged)

        invalidInputData(view: emailGlowingView)
        invalidInputData(view: passGlowingView)
        invalidInputData(view: repeatPassGlowingView)
        emailGlowingView.isHidden = true
        passGlowingView.isHidden = true
        repeatPassGlowingView.isHidden = true

        userImage.layer.borderWidth = 5
        userImage.layer.masksToBounds = false
        userImage.layer.borderColor = UIColor.white.cgColor
        userImage.layer.cornerRadius = userImage.frame.height / 2
        userImage.clipsToBounds = true
    }

    @objc func removeGlowing(_ textField: UITextField) {
        switch (textField) {
            case emailInputField:
                emailGlowingView.isHidden = true
                break
            case passInputField:
                passGlowingView.isHidden = true
                break
            case repeatPassInputField:
                repeatPassGlowingView.isHidden = true
                break
            default:
                break
        }
    }

    func textFieldDidBeginEditing(_ textField: UITextField) {
        removeGlowing(textField)
    }
```

```
    func invalidInputData(view: UIView) {
        view.roundCorners(corners: [.topRight, .bottomRight], radius: 10)
        view.addInnerShadow(onSide: UIView.innerShadowSide.all, shadowColor:
UIColor.red, shadowSize: 8, shadowOpacity: 2)
    }

    func checkInputData() -> Bool {
        let email = emailInputField.text ?? ""
        let pass = passInputField.text ?? ""
        let repeatPass = repeatPassInputField.text ?? ""

        if (email == "") {
            emailGlowingView.isHidden = false
            Loaf(NSLocalizedString("emptyEmail", comment: ""), state: .error, sender:
self).show()
            return false
        } else if (pass == "") {
            passGlowingView.isHidden = false
            Loaf(NSLocalizedString("emptyPass", comment: ""), state: .error, sender:
self).show()
            return false
        } else if (repeatPass == "") {
            repeatPassGlowingView.isHidden = false
            Loaf(NSLocalizedString("emptyRepeatPass", comment: ""), state: .error,
sender: self).show()
            return false
        } else if (pass != repeatPass) {
            repeatPassGlowingView.isHidden = false
            Loaf(NSLocalizedString("notSameRepeatPass", comment: ""), state: .error,
sender: self).show()
            return false
        }

        emailGlowingView.isHidden = true
        passGlowingView.isHidden = true
        repeatPassGlowingView.isHidden = true

        let rangeEmail = NSRange(location: 0, length: email.utf16.count)
        let regexEmail = try! NSRegularExpression(pattern: "^\\w+@[a-zA-Z_]+?\\.[a-zA-
Z]{2,3}$")
        let rangePass = NSRange(location: 0, length: pass.utf16.count)
        let regexPass = try! NSRegularExpression(pattern: "^(?=.*[a-z])(?=.*[A-
Z])(?=.*[0-9])(?=.*[!@#\\$%\\^&\\*])(?=.{8,})")

        if regexEmail.matches(in: email, options: [], range: rangeEmail).count == 0 {
```

```
            Loaf(NSLocalizedString("incorrectEmailFormat", comment: ""), state:
.error, sender: self).show()
            emailGlowingView.isHidden = false
            return false
        } else if regexPass.matches(in: pass, options: [], range: rangePass).count ==
0 {
            passGlowingView.isHidden = false
            Loaf(NSLocalizedString("weakPass", comment: ""), state: .error, sender:
self).show()
            return false
        }

        return true
    }

    func chooseImageFromDevice() {
       imagePicker.allowsEditing = false
       imagePicker.sourceType = .photoLibrary

       present(imagePicker, animated: true, completion: nil)
    }

    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
        if let pickedImage = info[UIImagePickerController.InfoKey.originalImage] as?
UIImage {
            userImage.contentMode = .scaleAspectFill
            userImage.image = pickedImage
            uploadUserPhotoButton.isHidden = true
        }

        dismiss(animated: true, completion: nil)
    }

    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
        dismiss(animated: true, completion: nil)
    }

    func registerUser() {
        guard let data = passInputField.text?.data(using: .utf8) else { return }
        let passHash = SHA512.hash(data: data).hexStr

        var avatarBase64 = ""
        if !(userImage.image?.isEqualToImage(image: UIImage(named: "upload-photo")!)
?? true) {
            avatarBase64 = userImage.image?.jpegData(compressionQuality:
1)?.base64EncodedString() ?? ""
```

```
            print(avatarBase64)
        }

        AF.request("http://localhost:8080/auth/register",
                   method: .post,
                   parameters: [
                        "email": emailInputField.text ?? "",
                        "pass": passHash,
                        "firstName": firstNameInputField.text ?? "",
                        "middleName": middleNameInputField.text ?? "",
                        "lastName": lastNameInputField.text ?? "",
                        "birthDate": birthDateInputField.text ?? "",
                        "avatar": avatarBase64
                   ],
                   encoder: JSONParameterEncoder.default).responseJSON { response in
            switch response.result {
                case .success(let data):
                    print("\nSuccessfully registered")
                    let json = JSON(data)
                    print(json)
                    let status = json["status"].stringValue

                    if (status == "success") {
                        let vc = UIStoryboard.init(name: "Main", bundle:
Bundle.main).instantiateViewController(withIdentifier: "SignUp2Screen") as!
SignUp2ViewController
                        vc.modalPresentationStyle = .fullScreen

                        vc.userId = json["userId"].stringValue
                        vc.address = json["address"].stringValue
                        vc.token = json["token"].stringValue

                        // Save ethereum private key to keychain
                        let saveSuccessful: Bool =
KeychainWrapper.standard.set(json["prKey"].stringValue, forKey: "prKey")
                        print("Saving prKey to keychain: \(saveSuccessful)")

                        self.present(vc, animated: true, completion: nil)
                    } else {
                        Loaf(NSLocalizedString("signupError", comment: ""), state:
.error, sender: self).show()
                    }

                    break
                case .failure(let error):
                    print(error.localizedDescription)
            }
```

```
        }
    }
}
```

Services/SignUp2Helper.swift

```
import UIKit
import Alamofire
import SwiftyJSON
import Loaf

/**
 Second SignUp screen
 */
extension SignUp2ViewController: UICollectionViewDataSource, UICollectionViewDelegate,
UITextViewDelegate {

    // Number of views
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int {
        return keywords.count
    }

    // Populate views
    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "keyword",
for: indexPath) as! KeywordCell
        cell.keywordLabel.text = keywords[indexPath.row]
        cell.keywordLabel.padding = UIEdgeInsets(top: 5, left: 10, bottom: 5, right:
10)

        return cell
    }

    func setupInputFields() {
        descriptionTitleView.roundCorners(corners: [.topLeft, .topRight], radius: 10)
        ethereumAddrTitleView.roundCorners(corners: [.topLeft, .topRight], radius: 10)
        copyEthAddrButton.roundCorners(corners: [.bottomRight], radius: 10)
        qrCodeView.roundCorners(corners: [.allCorners], radius: 10)
        ethAddrTextField.text = address
        descriptionTextBox.text = descriptionText
        descriptionTextBox.textColor = translucentWhite
    }

    func textViewDidBeginEditing(_ textView: UITextView) {
        if descriptionTextBox.textColor == translucentWhite {
```

```
            descriptionTextBox.text = nil
            descriptionTextBox.textColor = UIColor.white
        }
    }

    func textViewDidEndEditing(_ textView: UITextView) {
        if descriptionTextBox.text.isEmpty {
            descriptionTextBox.text = descriptionText
            descriptionTextBox.textColor = translucentWhite
        }
    }

    func generateQRCode(from string: String) -> UIImage? {
        let data = string.data(using: String.Encoding.ascii)

        guard let filter = CIFilter(name: "CIQRCodeGenerator") else { return nil }
        filter.setValue(data, forKey: "inputMessage")

        // Scale the image
        let transform = CGAffineTransform(scaleX: 10, y: 10)
        guard let scaledQrImage = filter.outputImage?.transformed(by: transform) else
{ return nil }

        return UIImage(ciImage: scaledQrImage)
    }

    func finishUserSignUp() {
        AF.request("http://localhost:8080/auth/updateUserData",
                   method: .post,
                   parameters: [
                        "token": self.token,
                        "description": descriptionTextBox.text == descriptionText ? ""
: descriptionTextBox.text,
                        "keywords": "",
                   ],
                   encoder: JSONParameterEncoder.default).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let json = JSON(data)
                    let status = json["status"]

                    if (status == "success") {
                        let vc = UIStoryboard.init(name: "Main", bundle:
Bundle.main).instantiateViewController(withIdentifier: "MessagesScreen") as!
ChatViewController
                        vc.modalPresentationStyle = .fullScreen
                        jwt = self.token
```

```
                        self.present(vc, animated: true, completion: nil)
                    } else {
                        Loaf(NSLocalizedString("signup2Error", comment: ""), state:
.error, sender: self).show()
                    }

                    break
                case .failure(let error):
                    print(error.localizedDescription)
            }
        }
    }
}
```

Services/SettingsHelper.swift

```
import UIKit
import Alamofire
import SwiftyJSON

extension SettingsViewController: UICollectionViewDataSource,
UICollectionViewDelegate, UITextViewDelegate, UITableViewDataSource,
UITableViewDelegate {

    /**
     Txs table start
     */
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        return txs.count
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) ->
CGFloat {
        return 30
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        self.selectedTxIndex = indexPath.row
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let tx = txs[indexPath.row]
        let cell = tableView.dequeueReusableCell(withIdentifier: "TxCell") as! TxCell
        cell.setTx(tx: tx)
```

```
        if (tx.direction == "out") { cell.isUserInteractionEnabled = false }

        cell.backgroundColor = indexPath.row % 2 == 0 ? purple : UIColor.init(red: 1,
green: 1, blue: 1, alpha: 0.1)
        cell.selectionColor = white

        return cell
    }
    /**
     Txs table end
     */

    /**
     Keywords collection start
     */
    // Number of views
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int {
        return keywords.count
    }

    // Populate views
    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "keyword",
for: indexPath) as! KeywordCell
        cell.keywordLabel.text = keywords[indexPath.row]
        cell.keywordLabel.padding = UIEdgeInsets(top: 5, left: 10, bottom: 5, right:
10)

        return cell
    }
    /**
     Keywords collection end
     */

    func setupInputFields() {
        getUserProfile()

        self.txTableView.delegate = self
        self.txTableView.dataSource = self
        self.txTableView.backgroundColor = purple
        self.txTableView.addSubview(self.refreshControl)

        selectCorrectLang()
        loadUserBalance()
```

```
        headerDate.roundCorners(corners: [.topLeft], radius: 10)
        headerAmount.roundCorners(corners: [.topRight], radius: 10)
        avatar.cornerRadius = 60

        langSegmentedControl.setTitleTextAttributes([
            NSAttributedString.Key.foregroundColor: UIColor.white,
            NSAttributedString.Key.font: UIFont(name: "SF Pro Text", size: 15) ??
UIFont.systemFont(ofSize: 15)
        ], for: .normal)

langSegmentedControl.setTitleTextAttributes([NSAttributedString.Key.foregroundColor:
UIColor.black], for: .selected)

        keywordsCollectionView.backgroundColor = purple
        descriptionTitleView.roundCorners(corners: [.topLeft, .topRight], radius: 10)
        descriptionTextView.text = descriptionText
        descriptionTextView.textColor = translucentWhite
    }

    func textViewDidBeginEditing(_ textView: UITextView) {
        if descriptionTextView.textColor == translucentWhite &&
descriptionTextView.text == descriptionText {
            descriptionTextView.text = nil
            descriptionTextView.textColor = UIColor.white
        }
    }

    func textViewDidEndEditing(_ textView: UITextView) {
        if descriptionTextView.text.isEmpty {
            descriptionTextView.text = descriptionText
            descriptionTextView.textColor = translucentWhite
        }
    }

    func selectCorrectLang() {
        for i in 0..<languages.count {
            if languages[i][1] == currentLangCode {
                langSegmentedControl.selectedSegmentIndex = i
            }
        }
    }

    func getUserProfile() {
        AF.request("http://localhost:8080/auth/myProfile", parameters: ["token": jwt]
                ).responseJSON { response in
            switch response.result {
                case .success(let data):
```

```
                    let json = JSON(data)
                    let status = json["status"].stringValue
                    if (status == "success") {
                        // email, firstName, middleName, lastName, birthDate,
description, avatar
                        let imageBase64String = json["avatar"].stringValue
                        let imageData = Data(base64Encoded: imageBase64String)
                        self.avatar.image = UIImage(data: imageData ?? Data())
                        self.fullName.text = "\(json["firstName"].stringValue)
\(json["middleName"].stringValue) \(json["lastName"].stringValue)"

                        let bdArr =
json["birthDate"].stringValue.components(separatedBy: " ")
                        self.birthDate.text = "\(bdArr[2])
\(NSLocalizedString(self.months[Int(bdArr[1]) ?? 0], comment: "")) \(bdArr[0])"

                        let desc = json["description"].stringValue
                        if desc != "" {
                            self.descriptionTextView.text = desc
                            self.descriptionTextView.textColor = UIColor.white
                        }
                    } else {
                        NSLog("Can't load user's profile because of an internal server
error")
                    }
                case .failure(let error):
                    NSLog(error.localizedDescription)
            }
        }
    }

    func getUnpublishedTxs() {
        AF.request("http://localhost:8080/bc/transfers", parameters: ["token": jwt]
                ).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let json = JSON(data)
                    let status = json["status"].stringValue
                    if (status == "success") {
                        let txs = json["txs"]

                        self.txs = []
                        for (_, tx) in txs {
                            let arr = tx["createdAt"].stringValue.split(separator: "
")
                            let date = "\(arr[0]) \(NSLocalizedString(String(arr[1]),
comment: "")) \(arr[2]) \(arr[3])"
```

```
                            self.txs.append(Tx(
                                id: tx["txId"].stringValue,
                                date: date,
                                userName: tx["fullName"].stringValue,
                                direction: tx["direction"].stringValue,
                                amount: tx["amount"].stringValue)
                            )
                        }
                        self.txTableView.reloadData()
                    } else {
                        NSLog("Can't load transactions because of an internal server
error")
                    }
                case .failure(let error):
                    NSLog(error.localizedDescription)
            }
        }
    }

    func loadUserBalance() {
        AF.request("http://localhost:8080/bc/balanceInContract", parameters: ["token":
jwt]).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let json = JSON(data)
                    let status = json["status"].stringValue
                    if (status == "success") {
                        let balance = json["balanceInEth"]

self.userBalance.setTitle(self.userBalance?.titleLabel?.text?.split(separator: "
").dropLast(2).joined(separator: " "), for: .normal)
                        guard let text = self.userBalance.titleLabel?.text else {
return }
                        self.userBalance.setTitle("\(text) \(balance) ETH", for:
.normal)
                    } else {
                        NSLog("Can't load user balance because of an internal server
error")
                    }
                case .failure(let error):
                    NSLog(error.localizedDescription)
            }
        }
    }

    func publishTransaction(tx: Tx) {
        print("publishing tx")
```

```
        AF.request("http://localhost:8080/bc/publishTransfer",
                   method: .post,
                   parameters: [
                       "token": jwt,
                       "txId": tx.id
                   ],
                   encoder: JSONParameterEncoder.default).responseJSON { response in
            switch response.result {
                case .success(let data):
                    let json = JSON(data)
                    let hash = json["txHash"].stringValue
                    print("The tx is now mining and it's hash is \(hash)")
                    self.getUnpublishedTxs()
                    break
                case .failure(let error):
                    print(error.localizedDescription)
            }
        }
    }
}
```

Extensions/ChatCell.swift

```
import UIKit

class ChatCell: UITableViewCell {

    @IBOutlet weak var avatarImageView: UIImageView!
    @IBOutlet weak var chatTypeImageView: UIImageView!
    @IBOutlet weak var chatTypeSelected: UIImageView!

    @IBOutlet weak var usernameLabel: UILabel!
    @IBOutlet weak var lastMsgTime: UILabel!
    @IBOutlet weak var lastMsgText: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
        avatarImageView.cornerRadius = 25
        lastMsgTime.text = NSLocalizedString("wasActiveAt", comment: "")
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
        if (selected) {
```

```
            usernameLabel.textColor = UIColor.white
            lastMsgText.textColor = UIColor.white
            lastMsgTime.textColor = UIColor.white
            chatTypeImageView.isHidden = true
            chatTypeSelected.isHidden = false
        } else {
            usernameLabel.textColor = UIColor.black
            lastMsgText.textColor = UIColor.black
            lastMsgTime.textColor = UIColor.black
            chatTypeImageView.isHidden = false
            chatTypeSelected.isHidden = true
        }
    }

    func setChat(chat: Chat) {
        avatarImageView.image = chat.avatar
        chatTypeImageView.image = chat.chatTypeImage
        chatTypeSelected.image = chat.chatTypeSelectedImage
        usernameLabel.text = chat.name
        lastMsgText.text = chat.lastMsgText
        lastMsgTime.text = chat.lastMsgTime
    }

}
```

Extensions/TxCell.swift

```
import UIKit

class TxCell: UITableViewCell {

    @IBOutlet weak var date: UILabel!
    @IBOutlet weak var userName: UILabel!
    @IBOutlet weak var txDirectionImg: UIImageView!
    @IBOutlet weak var amount: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        if (selected) {
            date.textColor = black
            userName.textColor = black
```

```
            amount.textColor = black
        } else {
            date.textColor = white
            userName.textColor = white
            amount.textColor = white
        }
    }

    func setTx(tx: Tx) {
        date.text = tx.date
        userName.text = tx.userName
        txDirectionImg.image = tx.direction == "in" ?  imageLiteral(resourceName: "in-
arrow") :  imageLiteral(resourceName: "out-arrow")
        amount.text = tx.amount
    }

}
```

Extensions/TableViewCell.swift

```
import Foundation
import UIKit

extension UITableViewCell {
    var selectionColor: UIColor {
        set {
            let view = UIView()
            view.backgroundColor = newValue
            self.selectedBackgroundView = view
        }
        get {
            return self.selectedBackgroundView?.backgroundColor ?? UIColor.clear
        }
    }
}
```

Extensions/KeywordCell.swift

```
import UIKit

class KeywordCell: UICollectionViewCell {

    @IBOutlet weak var keywordLabel: UILabel!

}
```

Extensions/MessageRightCell.swift

```
import UIKit

class MessageRightCell: UITableViewCell {

    @IBOutlet weak var time: UILabel!
    @IBOutlet weak var msg: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
        // Configure the view for the selected state
        msg.padding = UIEdgeInsets(top: 8, left: 15, bottom: 8, right: 15)
        msg.layer.masksToBounds = true
        msg.layer.cornerRadius = 10
    }

    func setMessage(message: Message) {
        msg.text = message.msg
        time.text = message.time
    }

}
```

Extensions/MessageLeftCell.swift

```
import UIKit

class MessageLeftCell: UITableViewCell {

    @IBOutlet weak var time: UILabel!
    @IBOutlet weak var msg: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
        // Configure the view for the selected state
        msg.padding = UIEdgeInsets(top: 8, left: 15, bottom: 8, right: 15)
```

```
        msg.layer.masksToBounds = true
        msg.layer.cornerRadius = 10
    }

    func setMessage(message: Message) {
        msg.text = message.msg
        time.text = message.time
    }

}
```

Extensions/UIView.ext.swift

```
import UIKit

/**
 Round corners
 */
extension UIView
{
    func roundCorners(corners:UIRectCorner, radius: CGFloat)
    {
        let maskLayer = CAShapeLayer()
        maskLayer.path = UIBezierPath(roundedRect: self.bounds, byRoundingCorners:
corners, cornerRadii: CGSize(width: radius, height: radius)).cgPath
        self.layer.mask = maskLayer
    }

    @IBInspectable
    var cornerRadius: CGFloat {
        get {
            return layer.cornerRadius
        }
        set {
            layer.cornerRadius = newValue
        }
    }
}

/**
 Inner shadow
 */
extension UIView
{
    // different inner shadow styles
    public enum innerShadowSide
    {
```

```
        case all, left, right, top, bottom, topAndLeft, topAndRight, bottomAndLeft,
bottomAndRight, exceptLeft, exceptRight, exceptTop, exceptBottom
    }

    // define function to add inner shadow
    public func addInnerShadow(onSide: innerShadowSide, shadowColor: UIColor,
shadowSize: CGFloat, cornerRadius: CGFloat = 0.0, shadowOpacity: Float)
    {
        // define and set a shaow layer
        let shadowLayer = CAShapeLayer()
        shadowLayer.frame = bounds
        shadowLayer.shadowColor = shadowColor.cgColor
        shadowLayer.shadowOffset = CGSize(width: 0.0, height: 0.0)
        shadowLayer.shadowOpacity = shadowOpacity
        shadowLayer.shadowRadius = shadowSize
        shadowLayer.fillRule = CAShapeLayerFillRule.evenOdd

        // define shadow path
        let shadowPath = CGMutablePath()

        // define outer rectangle to restrict drawing area
        let insetRect = bounds.insetBy(dx: -shadowSize * 2.0, dy: -shadowSize * 2.0)

        // define inner rectangle for mask
        let innerFrame: CGRect = { () -> CGRect in
            switch onSide
            {
                case .all:
                    return CGRect(x: 0.0, y: 0.0, width: frame.size.width, height:
frame.size.height)
                case .left:
                    return CGRect(x: 0.0, y: -shadowSize * 2.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height + shadowSize * 4.0)
                case .right:
                    return CGRect(x: -shadowSize * 2.0, y: -shadowSize * 2.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height + shadowSize * 4.0)
                case .top:
                    return CGRect(x: -shadowSize * 2.0, y: 0.0, width:
frame.size.width + shadowSize * 4.0, height: frame.size.height + shadowSize * 2.0)
                case.bottom:
                    return CGRect(x: -shadowSize * 2.0, y: -shadowSize * 2.0, width:
frame.size.width + shadowSize * 4.0, height: frame.size.height + shadowSize * 2.0)
                case .topAndLeft:
                    return CGRect(x: 0.0, y: 0.0, width: frame.size.width + shadowSize
* 2.0, height: frame.size.height + shadowSize * 2.0)
                case .topAndRight:
```

```
                    return CGRect(x: -shadowSize * 2.0, y: 0.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height + shadowSize * 2.0)
                case .bottomAndLeft:
                    return CGRect(x: 0.0, y: -shadowSize * 2.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height + shadowSize * 2.0)
                case .bottomAndRight:
                    return CGRect(x: -shadowSize * 2.0, y: -shadowSize * 2.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height + shadowSize * 2.0)
                case .exceptLeft:
                    return CGRect(x: -shadowSize * 2.0, y: 0.0, width:
frame.size.width + shadowSize * 2.0, height: frame.size.height)
                case .exceptRight:
                    return CGRect(x: 0.0, y: 0.0, width: frame.size.width + shadowSize
* 2.0, height: frame.size.height)
                case .exceptTop:
                    return CGRect(x: 0.0, y: -shadowSize * 2.0, width:
frame.size.width, height: frame.size.height + shadowSize * 2.0)
                case .exceptBottom:
                    return CGRect(x: 0.0, y: 0.0, width: frame.size.width, height:
frame.size.height + shadowSize * 2.0)
            }
        }()

        // add outer and inner rectangle to shadow path
        shadowPath.addRect(insetRect)
        shadowPath.addRect(innerFrame)

        // set shadow path as show layer's
        shadowLayer.path = shadowPath

        // add shadow layer as a sublayer
        layer.addSublayer(shadowLayer)

        // hide outside drawing area
        clipsToBounds = true
    }
}
```

Extensions/UIViewController.swift

```
import UIKit
import SwiftyJSON
import Loaf

extension UIViewController {

    func navigateToScreen(screenName: String, storyboardName: String = "Main") {
```

```
        // Navigate to the Messages screen
        let storyBoard = UIStoryboard(name: "Main", bundle: nil)
        let vc = storyBoard.instantiateViewController(withIdentifier: screenName)
        vc.modalPresentationStyle = .fullScreen
        self.present(vc, animated: true, completion: nil)
    }

    func setStatusBarBackgroundColor(color: UIColor) {
        if #available(iOS 13.0, *) {
            let app = UIApplication.shared
            let statusBarHeight: CGFloat = app.statusBarFrame.size.height

            let statusbarView = UIView()
            statusbarView.backgroundColor = color
            statusbarView.tintColor = UIColor.white
            view.addSubview(statusbarView)

            statusbarView.translatesAutoresizingMaskIntoConstraints = false
            statusbarView.heightAnchor
                .constraint(equalToConstant: statusBarHeight).isActive = true
            statusbarView.widthAnchor
                .constraint(equalTo: view.widthAnchor, multiplier: 1.0).isActive =
true
            statusbarView.topAnchor
                .constraint(equalTo: view.topAnchor).isActive = true
            statusbarView.centerXAnchor
                .constraint(equalTo: view.centerXAnchor).isActive = true

        } else {
            let statusBar = UIApplication.shared.value(forKeyPath:
"statusBarWindow.statusBar") as? UIView
            statusBar?.backgroundColor = color
        }
    }

    func listenForGlobalSocketMsgs() {
        socket.socket.on("tx-confirmed") { data, ack in
            print("event tx-confirmed")
            let event = JSON(data)[0]

            print(event["userId"].stringValue)
            print(event["txHash"].stringValue)

            if event["userId"].stringValue == userId {
                Loaf(NSLocalizedString("txConfirmed", comment: ""), state: .success,
location: .top, sender: self).show()
            }
```

```
        }
    }

}
```

Extensions/Utils.swift

```
import Foundation
import CryptoKit

extension Digest {
    var bytes: [UInt8] { Array(makeIterator()) }
    var data: Data { Data(bytes) }

    var hexStr: String {
        bytes.map { String(format: "%02X", $0) }.joined().lowercased()
    }
}
```

Extensions/UILabel.ext.swift

```
import Foundation
import UIKit

extension UILabel {
    private struct AssociatedKeys {
        static var padding = UIEdgeInsets()
    }

    public var padding: UIEdgeInsets? {
        get {
            return objc_getAssociatedObject(self, &AssociatedKeys.padding) as?
UIEdgeInsets
        }
        set {
            if let newValue = newValue {
                objc_setAssociatedObject(self, &AssociatedKeys.padding, newValue as
UIEdgeInsets?, objc_AssociationPolicy.OBJC_ASSOCIATION_RETAIN_NONATOMIC)
            }
        }
    }

    override open func draw(_ rect: CGRect) {
        if let insets = padding {
            self.drawText(in: rect.inset(by: insets))
        } else {
```

```
            self.drawText(in: rect)
        }
    }

    override open var intrinsicContentSize: CGSize {
        guard let text = self.text else { return super.intrinsicContentSize }

        var contentSize = super.intrinsicContentSize
        var textWidth: CGFloat = frame.size.width
        var insetsHeight: CGFloat = 0.0
        var insetsWidth: CGFloat = 0.0

        if let insets = padding {
            insetsWidth += insets.left + insets.right
            insetsHeight += insets.top + insets.bottom
            textWidth -= insetsWidth
        }

        let newSize = text.boundingRect(with: CGSize(width: textWidth, height:
CGFloat.greatestFiniteMagnitude),
                                        options:
NSStringDrawingOptions.usesLineFragmentOrigin,
                                        attributes: [NSAttributedString.Key.font:
self.font ?? UIFont()], context: nil)

        contentSize.height = ceil(newSize.size.height) + insetsHeight
        contentSize.width = ceil(newSize.size.width) + insetsWidth

        return contentSize
    }
}
```

Extensions/Microfutures.swift

```
import Foundation

public enum Result<T> {
    case success(T)
    case failure(Error)
}

public struct Future<T> {
    public typealias ResultType = Result<T>

    private let operation: ( @escaping (ResultType) -> ()) -> ()

    public init(result: ResultType) {
```

```
            self.init(operation: { completion in
                completion(result)
            })
        }

        public init(value: T) {
            self.init(result: .success(value))
        }

        public init(error: Error) {
            self.init(result: .failure(error))
        }

        public init(operation: @escaping ( @escaping (ResultType) -> ()) -> ()) {
            self.operation = operation
        }

        fileprivate func then(_ completion: @escaping (ResultType) -> ()) {
            self.operation() { result in
                completion(result)
            }
        }

        public func subscribe(onNext: @escaping (T) -> Void = { _ in }, onError: @escaping
    (Error) -> Void = { _ in }) {
            self.then { result in
                switch result {
                case .success(let value): onNext(value)
                case .failure(let error): onError(error)
                }
            }
        }
    }

    extension Future {
        public func map<U>(_ f: @escaping (T) throws -> U) -> Future<U> {
            return Future<U>(operation: { completion in
                self.then { result in
                    switch result {

                    case .success(let resultValue):
                        do {
                            let transformedValue = try f(resultValue)
                            completion(Result.success(transformedValue))
                        } catch let error {
                            completion(Result.failure(error))
                        }
```

```
                case .failure(let errorBox):
                    completion(Result.failure(errorBox))

                }
            }
        })
    }

    public func flatMap<U>(_ f: @escaping (T) -> Future<U>) -> Future<U> {
        return Future<U>(operation: { completion in
            self.then { firstFutureResult in
                switch firstFutureResult {
                case .success(let value): f(value).then(completion)
                case .failure(let error): completion(Result.failure(error))
                }
            }
        })
    }
}
```

Extensions/KeywordStyle.swift

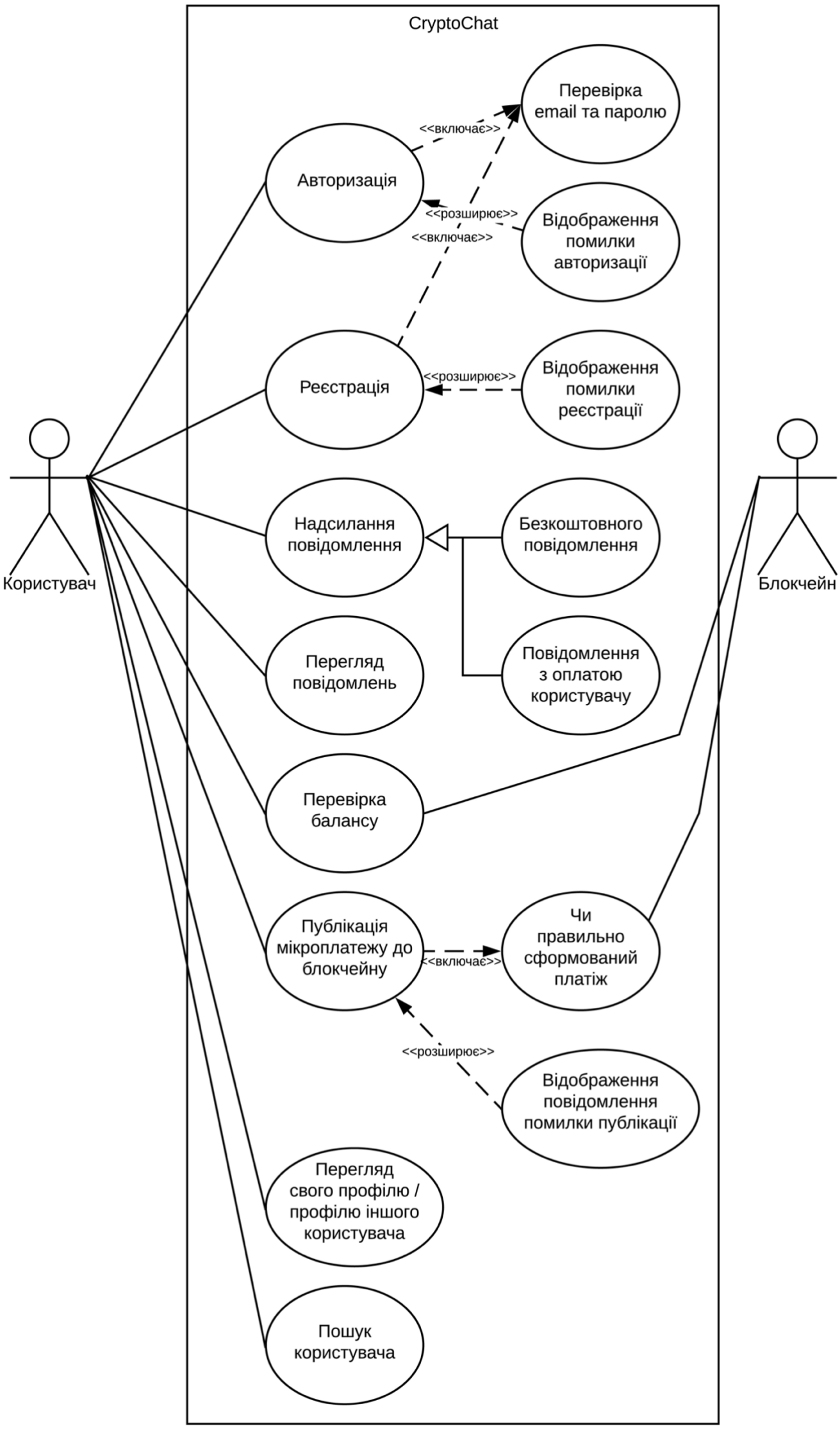
```
import UIKit

class KeywordStyle: UIView {

    //initWithFrame to init view from code
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    //initWithCode to init view from xib or storyboard
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    //common func to init our view
    private func setupView() {
        backgroundColor = UIColor(red: 122, green: 140, blue: 255, alpha: 0)

        cornerRadius = 15
        layer.borderColor = UIColor.white.cgColor
        layer.borderWidth = 1
```

```
    }
}
```

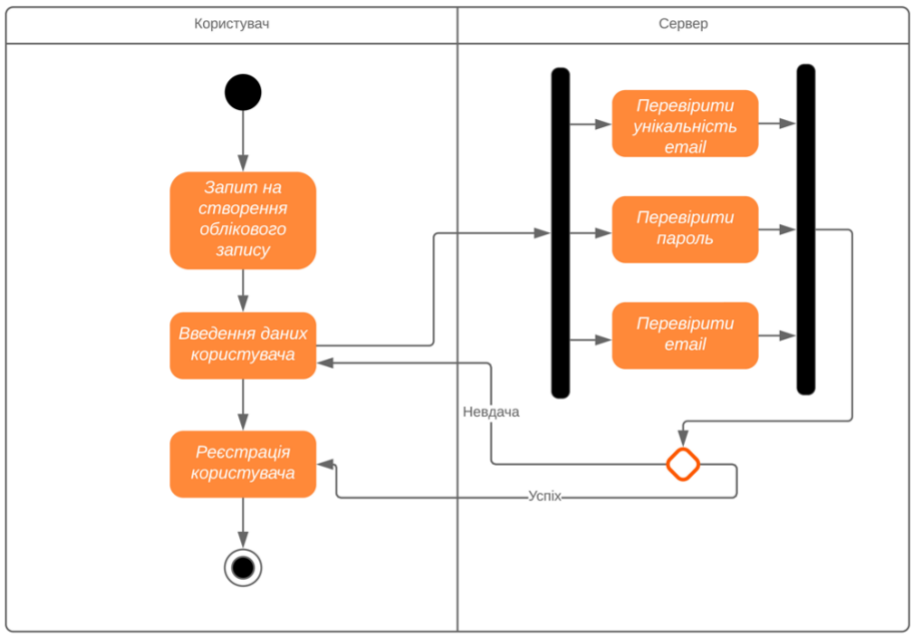
Extensions/UIImage.ext.swift

```
import UIKit

extension UIImage {

    func isEqualToImage(image: UIImage) -> Bool {
        let data1: NSData = self.pngData()! as NSData
        let data2: NSData = image.pngData()! as NSData
        return data1.isEqual(data2)
    }

}
```

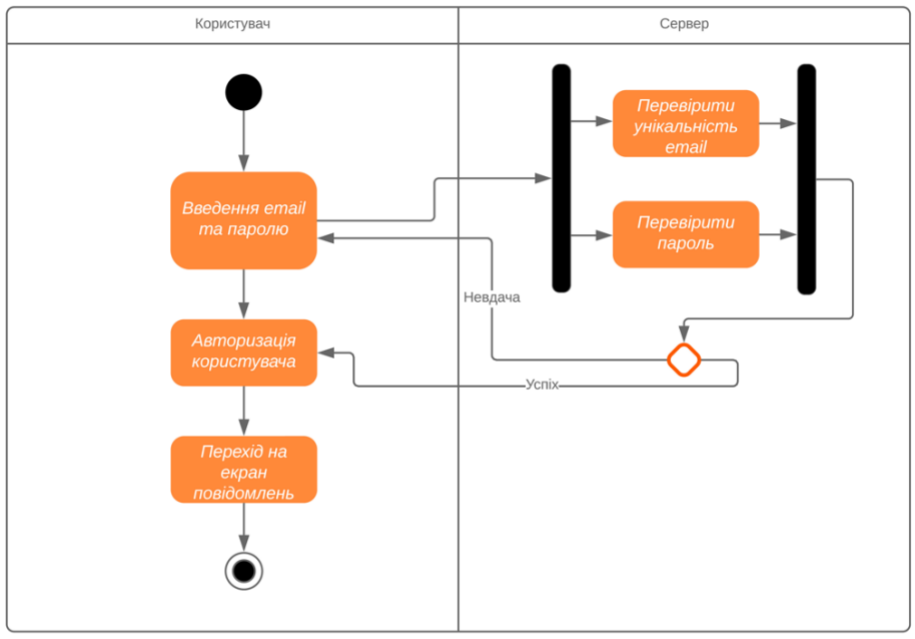
КПІ.ІП-6116.045490.08.СС



					КПІ.ІП-6116.045490.08.СС		
					Схема структурна варіантів використань	Літ.	Маса / Масш.
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кушка М.О.				Аркуш 1	Аркушів 1
Перевірив		Ліщук К.І.					
Т. Контр.							
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61	
Н. Контр.		Ліщук К.І.					
Затверд.							

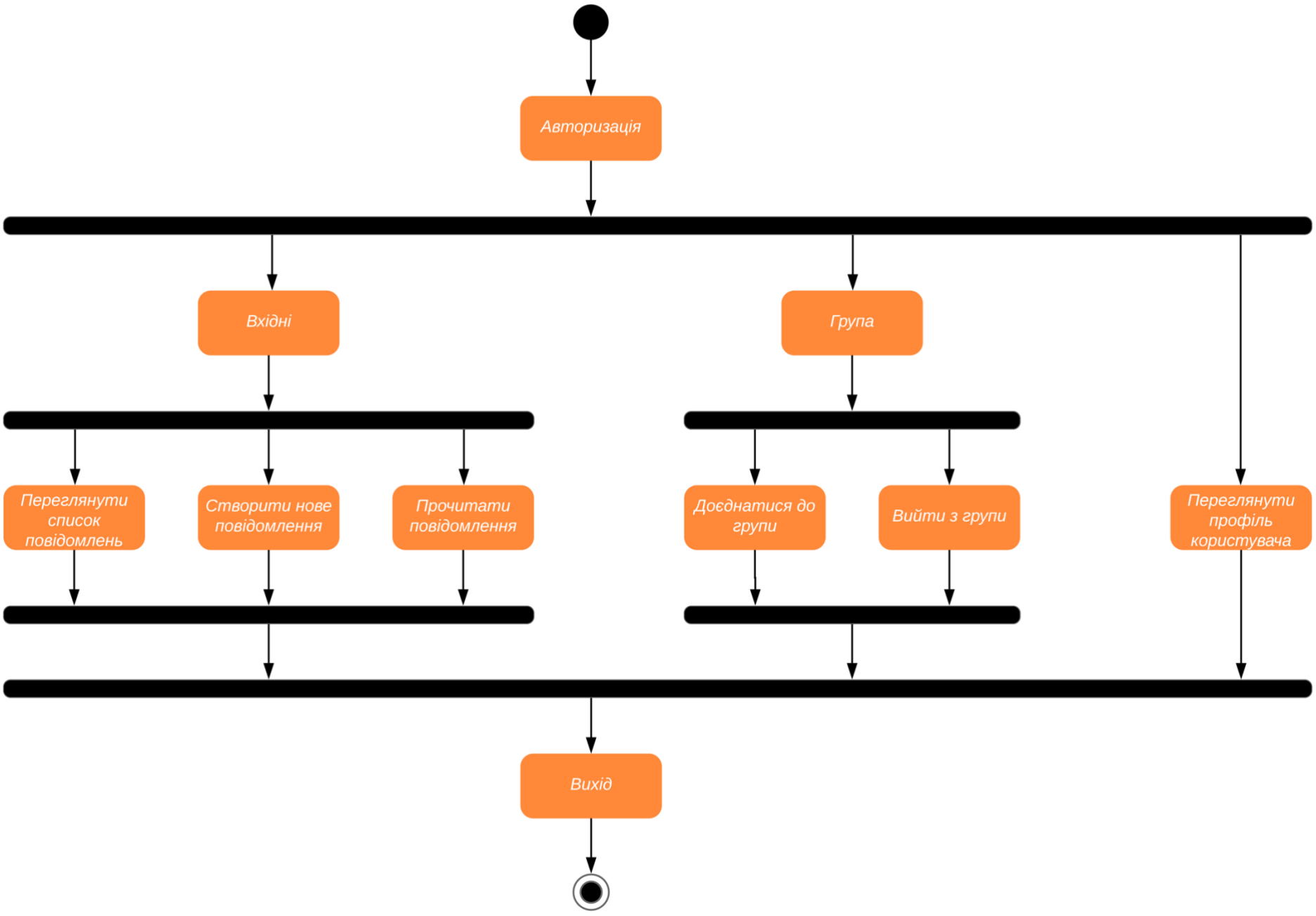


Реєстрація

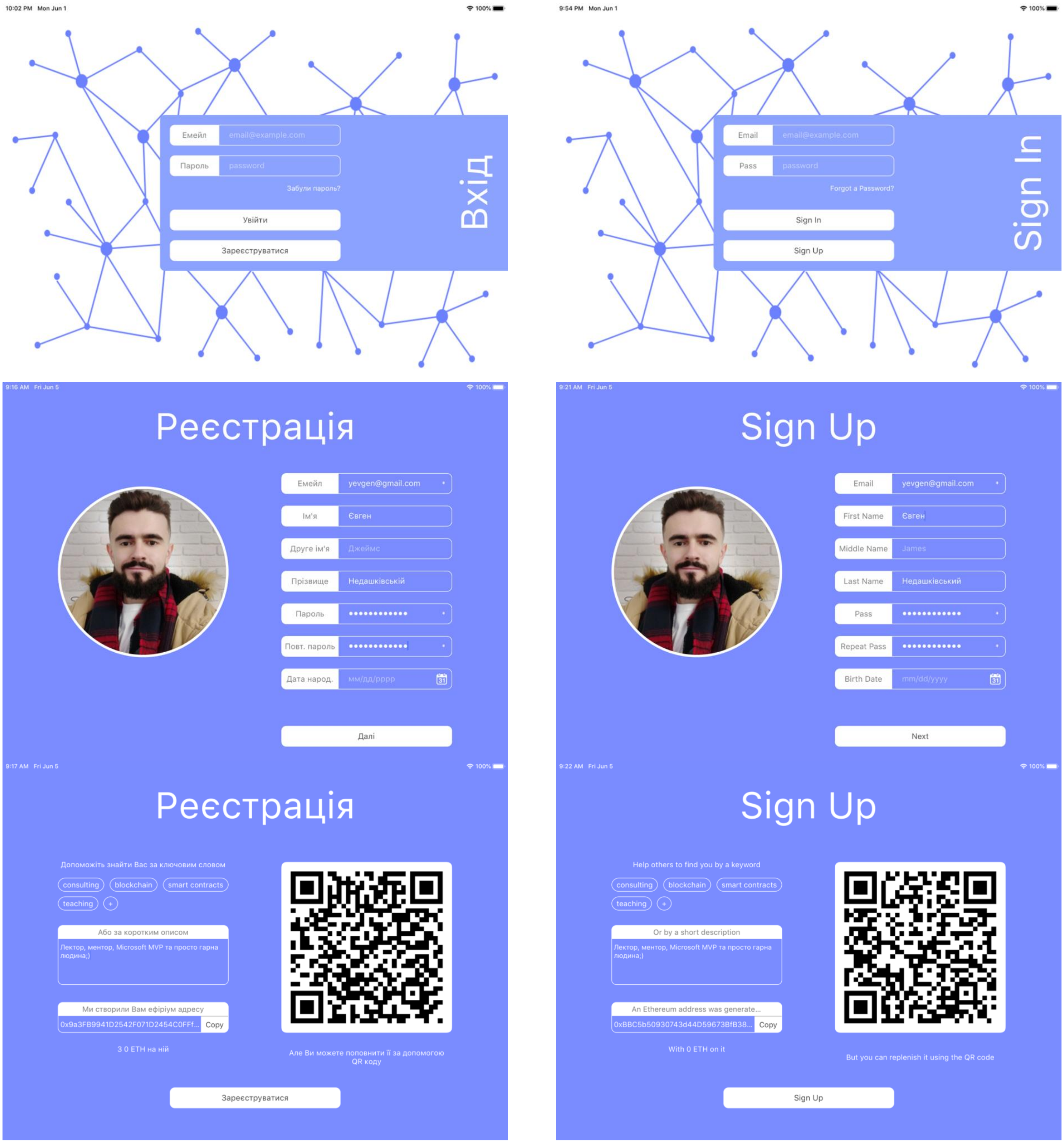


Авторизація

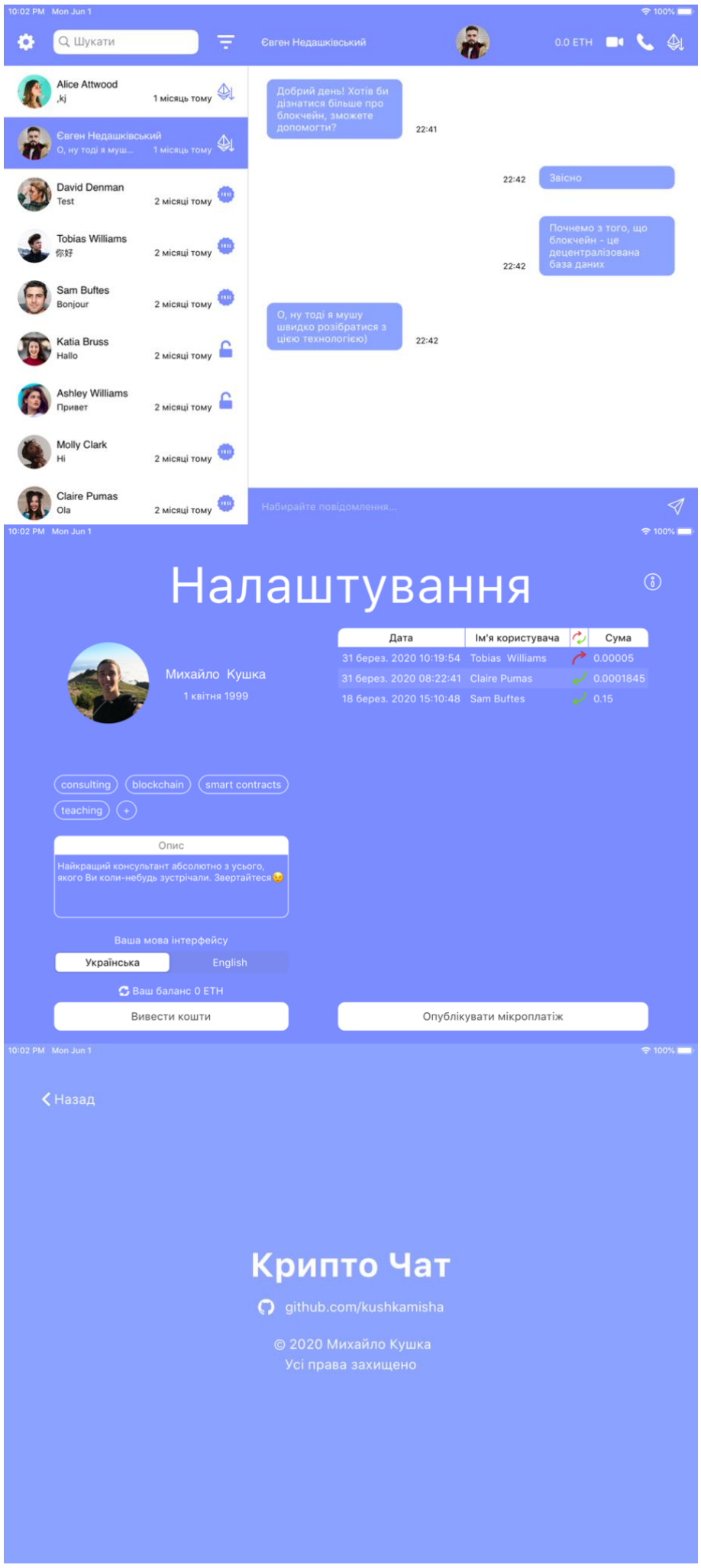
Чат

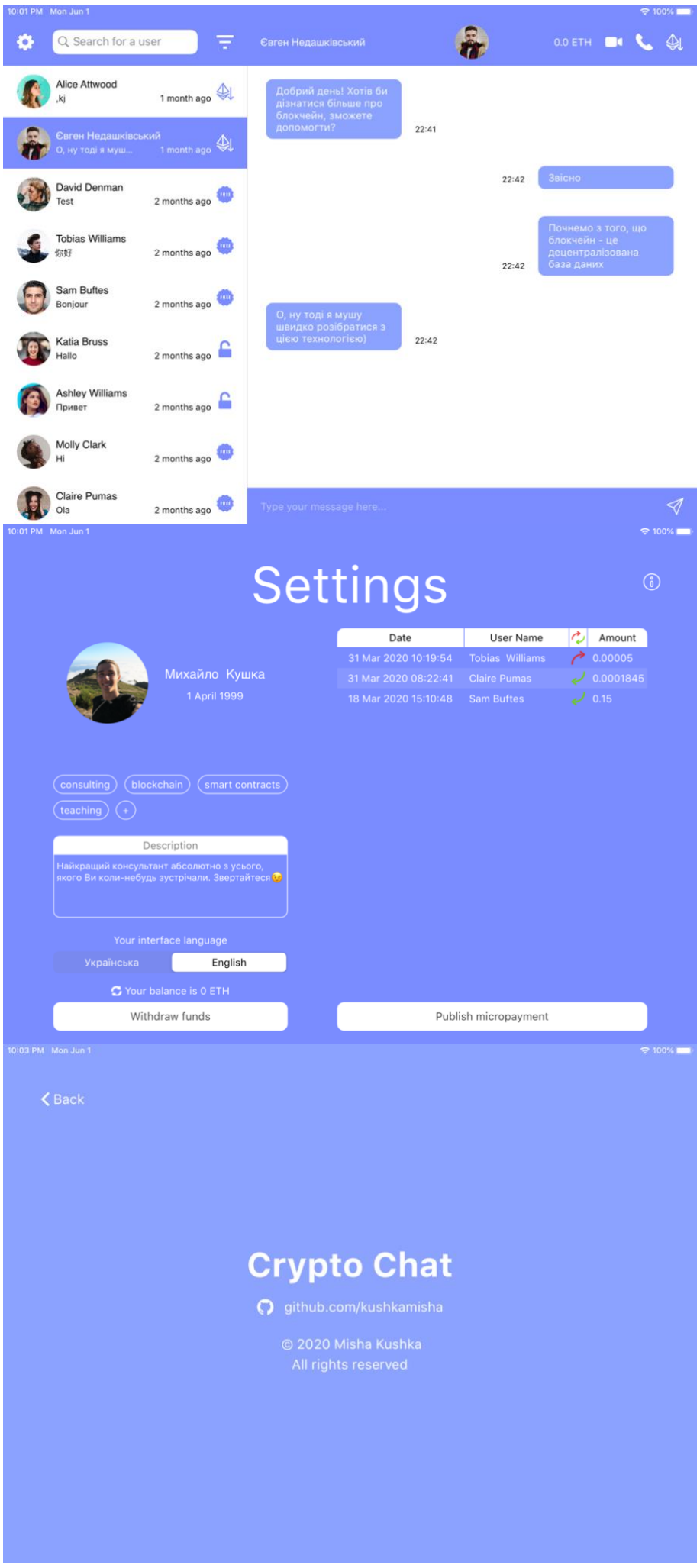


					КПІ.ІП-6116.045490.09.СС			
Зм.	Арк.	№ докум.	Підпис	Дата	Схема структурна діяльності	Літ.	Маса	Масш.
Розробив		Кушка М.О.						
Перевірив		Ліщук К.І.				Аркуш 1	Аркушів 1	
Т. Контр.								
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		
Н. Контр.		Ліщук К.І.						
Затверд.								



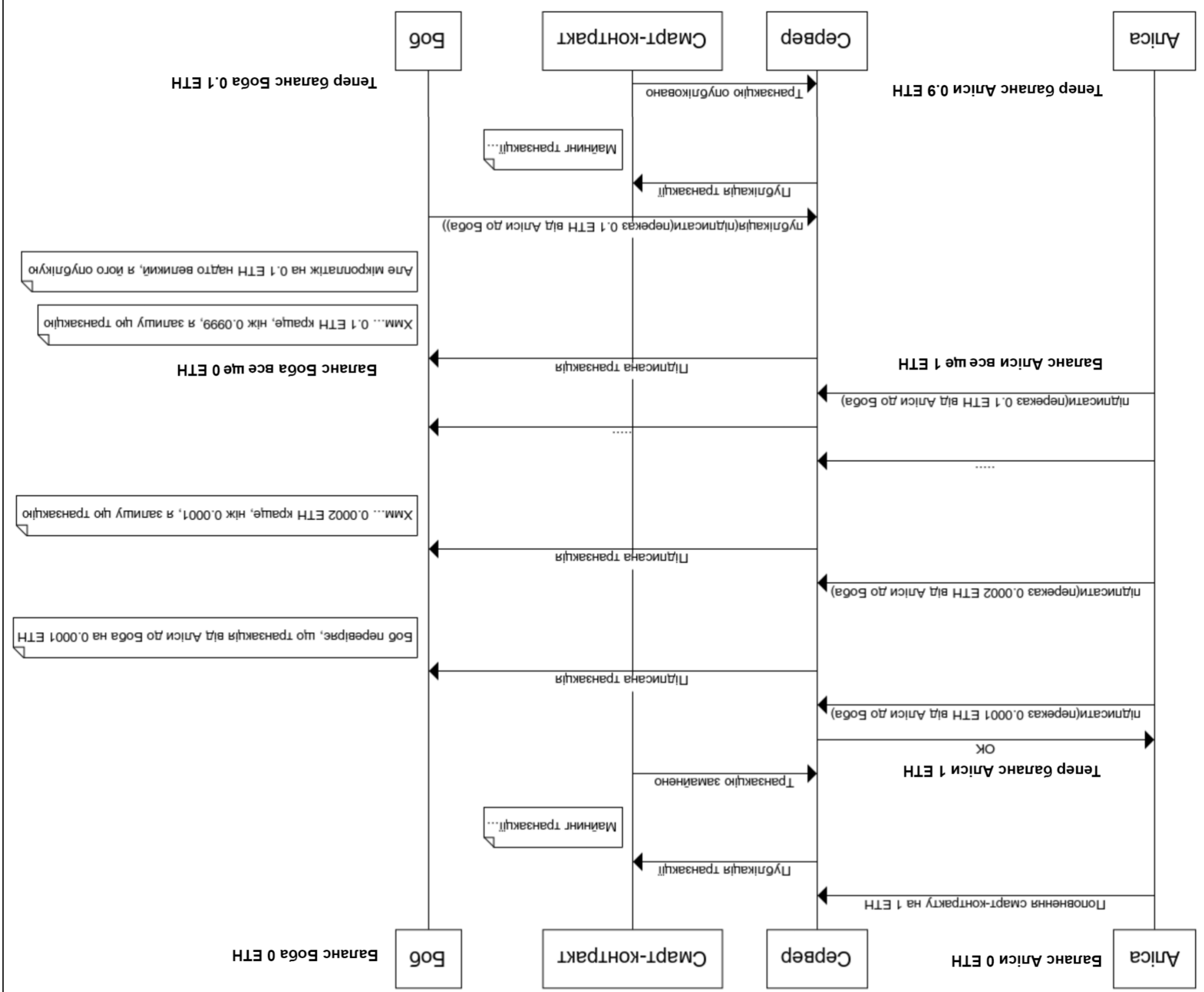
					КПІ.ІП-6116.045490.10.КЕ			
					Креслення вигляду екранних форм	Літ.	Маса	Масш.
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Кушка М.О.						
Перевірив		Ліщук К.І.						
Т. Контр.						Аркуш 1	Аркушів 2	
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		
Н. Контр.		Ліщук К.І.						
Затверд.								

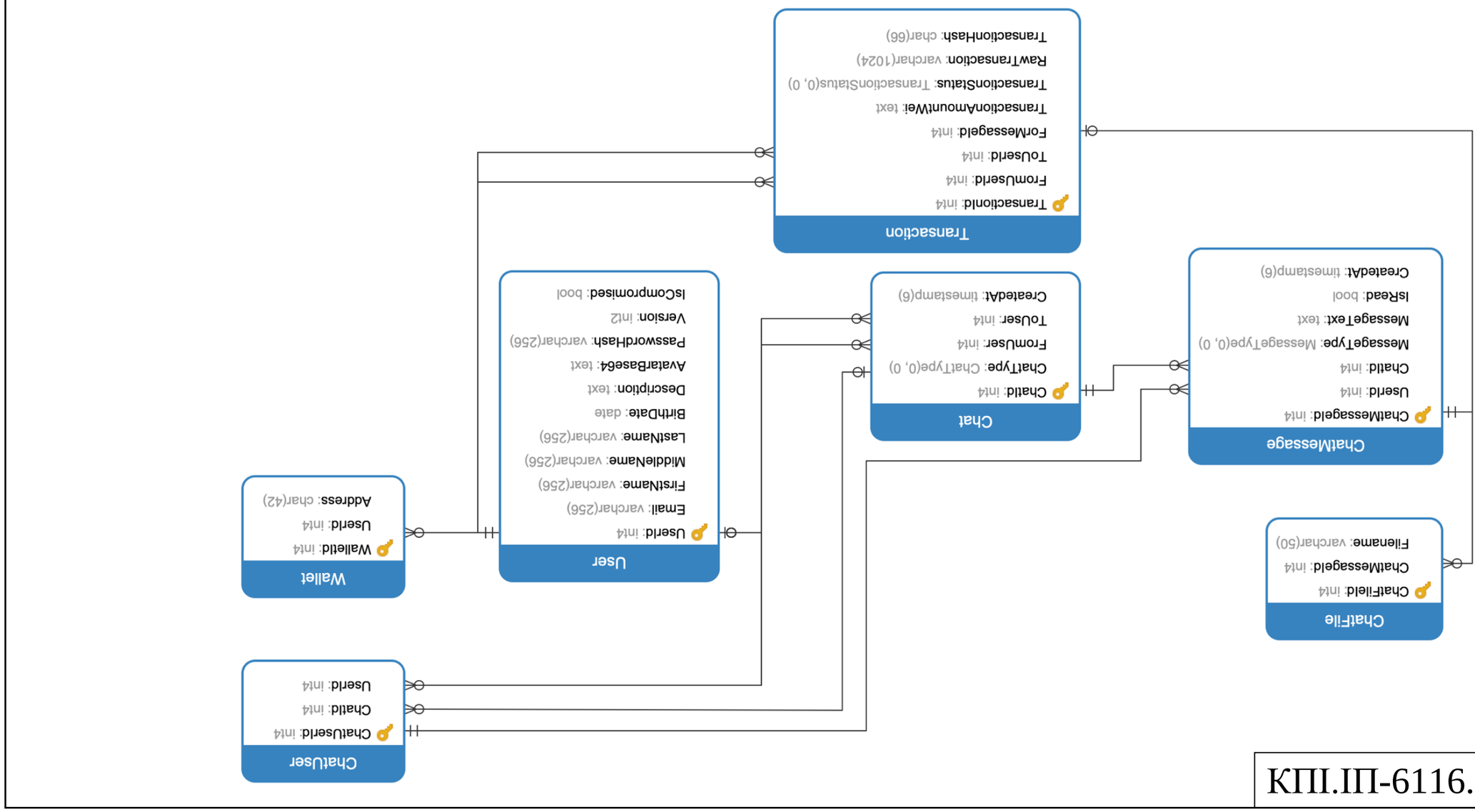




					КПІ.ІП-6116.045490.10.КЕ		
					Креслення вигляду екранних форм	Літ.	Маса / Масш.
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кушка М.О.				Аркуш 2	Аркушів 2
Перевірив		Ліщук К.І.					
Т. Контр.							
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61	
Н. Контр.		Ліщук К.І.					
Затверд.							

						КПІ.ІП-6116.045490.11.СС			
Зм.	Арк.	№ докум.	Підпис	Дата	Схема структура послідовності	Літ.	Маса	Масш.	
Розробив		Кушка М.О.							
Перевірив		Лішук К.І.							
Т. Контр.						Аркуш 1		Аркушів 1	
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61			
Н. Контр.		Лішук К.І.							
Затверд.									

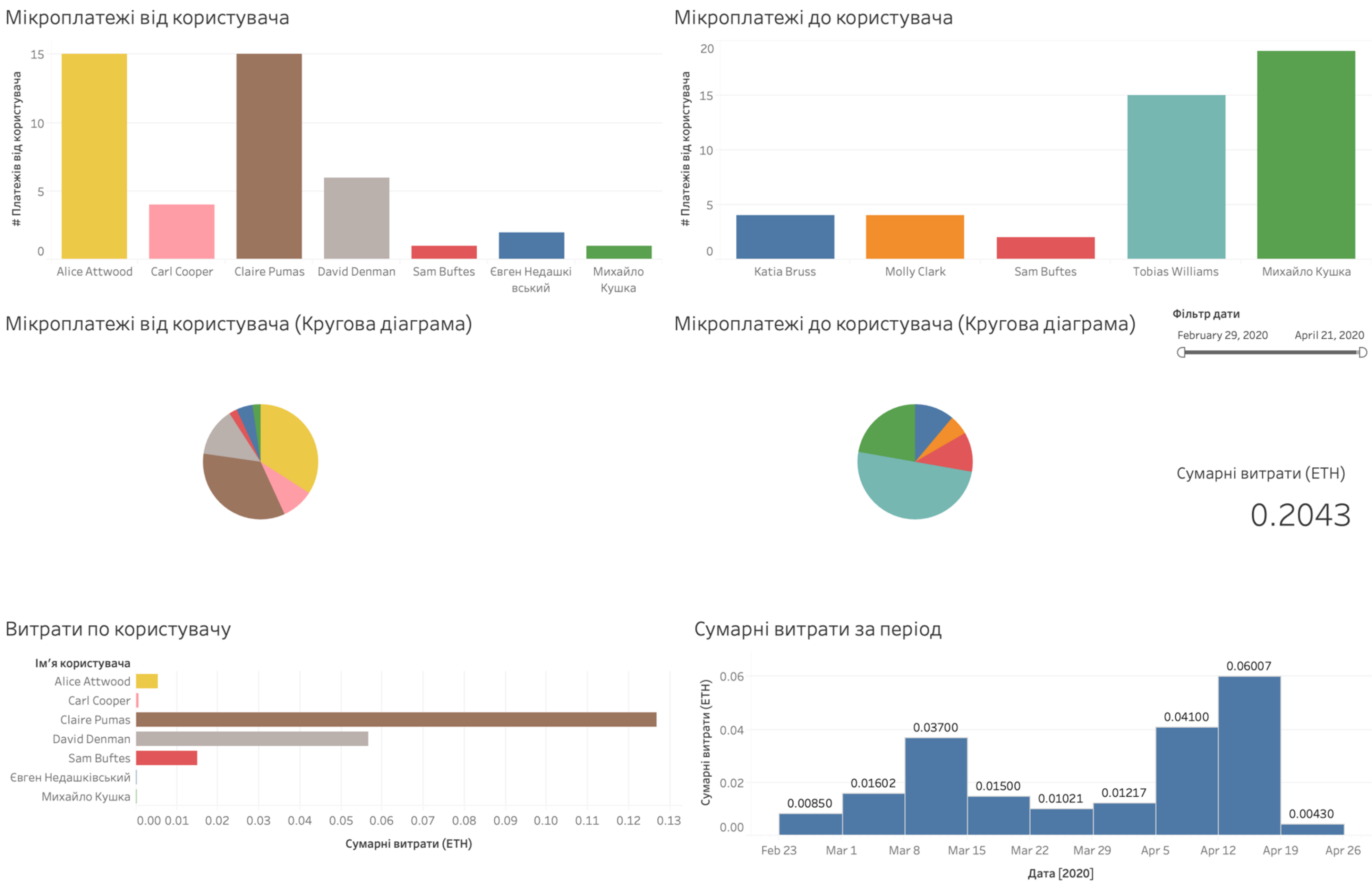




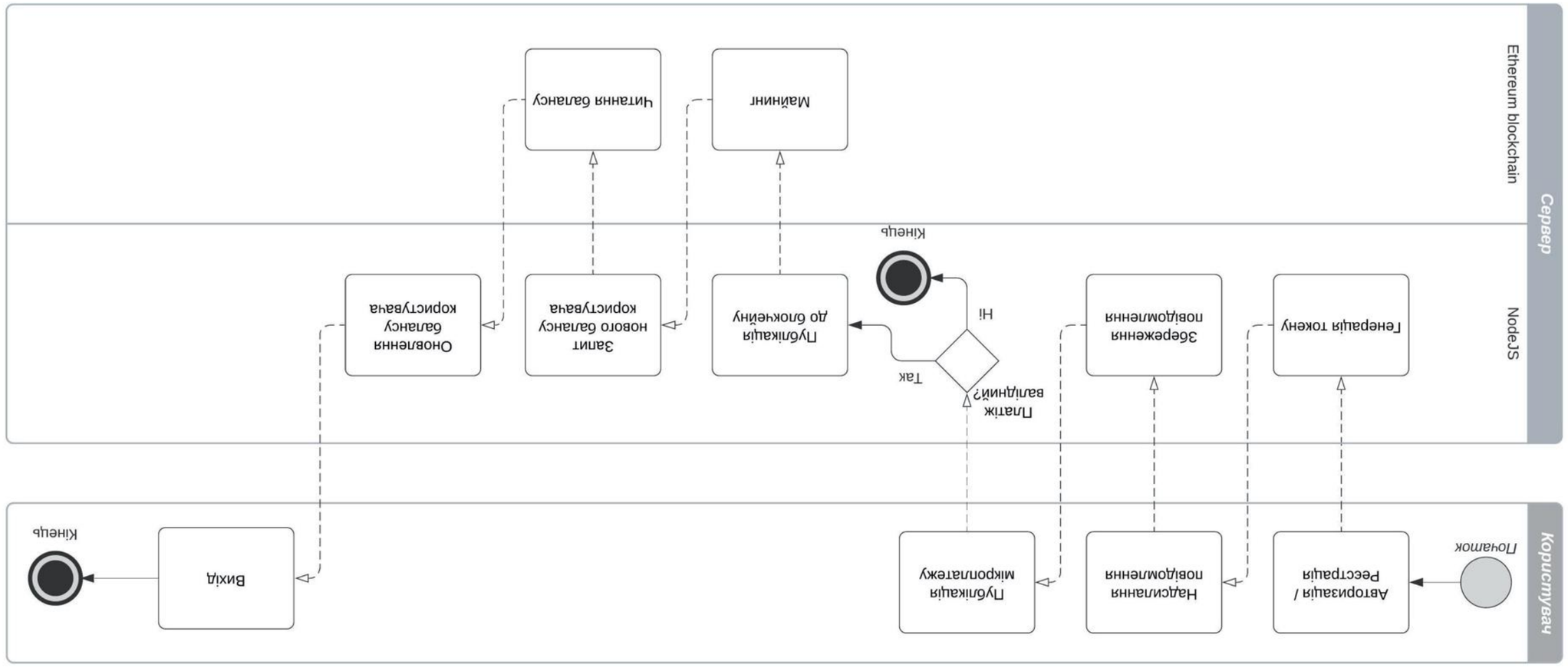
КПІ.ІП-6116.045490.12.СБД

								КПІ.ІП-6116.045490.12.СБД		
Змн.	Арк.	№ докум.	Підпис	Дата						
Розробив		Кушка М.О.						Схема бази даних	Літ.	Маса	Масш.
Перевірив		Лищук К.І.									
Т. Контр.											
Керівник		Недашківський Є. А.							Аркуш 1	Аркушів 1	
Н. Контр.		Лищук К.І.						Меседжер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61	
Затверд.										

КПІ.ІП-6116.045490.13.КЗ



Зм.	Арк.	№ докум.	Підпис	Дата	КПІ.ІП-6116.045490.13.КЗ
Розробив		Кушка М.О.			Креслення вигляду звітних форм
Перевірив		Ліщук К.І.			Літ. / Маса / Масш.
Т. Контр.					Аркуш 1 / Аркушів 1
Керівник		Недашківський Є. А.			Месенджер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій
Н. Контр.		Ліщук К.І.			КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61
Затверд.					



										КПІ.ІП-6116.045490.14.СС
Зм.	Арк.	№ докум.	Підпис	Дата						
Розробив		Кушка М.О.			Схема структура бізнес процесу			Літ.	Маса	Масш.
Перевірив		Лищук К.І.								
Т. Контр.										
Керівник		Недашківський Є. А.						Аркуш 1	Аркушів 1	
Н. Контр.		Лищук К.І.			Меседжер для надання консультаційних послуг на основі блокчейн-технології та мікротранзакцій			КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		
Затверд.										