Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, спеціаліста, магістра)

Студента Япринцева Ярослава Миколайовича
(ПІБ)

академічної групи 126-20-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)

на тему Розробка серверного застосунку інформаційної системи "Каса взаємодопомоги"
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікавційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри

____інформаційних технологій____
___та комп'ютерної інженерії____
(повна назва)

_____ __Гнатушенко В.В._____
(підпис) (прізвище, ініціали)

«_____»_____2024_року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____бакалавр_____
(бакалавра, спеціаліста, магістра

студенту _____Япринцеву Я.М._____ **академічної групи** ____126-20-1____
(прізвище та ініціали) (шифр)

спеціальності _____126 « Інформаційні системи та технології »_____

за освітньою-професійною програмою _____

_____«Інформаційні системи та технології»_____

на тему _____Розробка серверного застосунку інформаційної системи _____

_____"Каса взаємодопомоги"_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 р. № 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Ознайомлення з матеріалами сайтів, літературою та виконання пошуку технологій і формування відповідних рішень	01.02.2024 – 30.03.2024
Розділ 2	Реалізація проектних рішень	01.04.2024 – 20.06.2024

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____01.02.2024 р._____

Дата подання до екзаменаційної комісії ____02.07.2024 р.____

Прийнято до виконання _____ _____Япринцев Я.М._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 88 сторінки, 7 рисунків, 13 таблиць, 3 додатка, 27 джерел.

Об’єкт розробки: серверний застосунок інформаційної системи «Каса взаємодопомоги».

Мета кваліфікаційної роботи: розробка програмного забезпечення серверного застосунку інформаційної системи «Каса взаємодопомоги».

У першому розділі звертається увага на актуальність проблеми, проводиться аналіз існуючих аналогів серверних застосунків для інформаційних систем, і визначаються мета та задачі кваліфікаційної роботи.

Другий розділ присвячений проектуванню серверного застосунку інформаційної системи «Каса взаємодопомоги», де наведено опис його архітектури, алгоритму роботи, та функціональності. Крім того, розглядаються технології, які будуть використовуватись для реалізації проекту.

У наступному розділі детально описується реалізація застосунку. Виконується розробка самого вебзастосунку серверної частини «Каси взаємодопомоги» з використанням інструментів фреймворків Flask та Peewee, і мови програмування Python. Також описується порядок взаємодії з графічним інтерфейсом користувача.

Результатом проведеної роботи є створений серверний застосунок інформаційної системи «Каса взаємодопомоги». Цей вебзастосунок дозволяє обслуговувати запити від мобільних застосунків учасників благодійних програм, та від адміністративного сайту інфомаційної системи фонду взаємодопомоги.

Ключові слова: СЕРВЕР, КАСА ВЗАЄМОДОПОМОГИ, ВЕБЗАСТОСУНОК, ФРЕЙМВОРК, REST, FLASK, PEEWEE, PYTHON.

ABSTRACT

Explanatory note: 88 pages, 7 fig., 13 tab, 3 appendices, 27 sources.

The object of development: server application of the information system "Mutual Assistance Fund".

The purpose of the qualification work: development of software for server application of the information system "Mutual Assistance Fund".

The first section draws attention to the relevance of the problem, analyzes existing analogs of server applications for information systems, and defines the purpose and tasks of the qualification work.

The second section is devoted to the design of the server application of the information system "Mutual Assistance Fund", which describes its architecture, work algorithm, and functionality. In addition, the technologies that will be used for the implementation of the project are considered.

The next section describes the implementation of the application in detail. The development of the web application of the server part of the "Mutual Assistance Fund" using the tools of the Flask and Peewee frameworks and the Python programming language is in progress. The order of interaction with the graphical user interface is also described.

The result of the work is the created server application of the information system "Mutual Assistance Fund". This web application allows you to serve requests from mobile applications of charity program participants, and from the administrative site of the information system of the mutual aid fund.

Keywords: SERVER, MUTUAL ASSISTANCE FUND, WEB APPLICATION, FRAMEWORK, REST, FLASK, PEEWEE, PYTHON.

ЗМІСТ

ВСТУП

Взаємодопомога — це організаційна модель, у якій відбувається добровільний спільний обмін ресурсами та послугами для загальної користі між членами громади для подолання соціальних, економічних і політичних бар'єрів на шляху задоволення спільних потреб. Це може включати фізичні ресурси, як-от їжа, одяг чи ліки, а також послуги, як-от програми сніданку чи навчання. Ці групи часто створюються для щоденних потреб їхніх громад, але групи взаємодопомоги також можна знайти під час надання допомоги, наприклад, у разі стихійних лих або пандемій, як-от COVID-19.

У рамках даної кваліфікаційної роботи було поставлено завдання розробити серверний застосунок інформаційної системи «Каса взаємодопомоги». «Каса взаємодопомоги» є інформаційною системою, яка автоматизує процес функціонування руху взаємодопомоги.

Цей проект має на меті розробити зручний та ефективний механізм кооперації віддалених підсистем, що входять до складу «Каси взаємодопомоги». З його допомогою буде здійснюватися авторизація користувачів, обробка платежів, завантаження файлів на сервер системи, та обмін інформацією з базою даних.

У процесі розробки серверного застосунку інформаційної системи буде використана сучасна вебтехнологія, що дозволить створити потужний та функціональний інструмент для забезпечення функціонування каси взаємодопомоги. Також будуть застосовані методи авторизації, зокрема, через інтеграцію з «Дією», щоб забезпечити розділення доступу до персональних даних користувачів.

Основною метою цієї розробки є створення потужного та надійного інструменту, який забезпечить злагоджену та безперебійну спільну роботу адміністративного сайту фонду, мобільних застосунків його учасників, та сховища даних.

Розроблений вебзастосунок буде враховувати сучасні вимоги до надійності та функціональності, що забезпечить швиидку та коректну роботу інформаційної системи. Додатково будуть застосовані методи валідації та перевірки введених даних, щоб уникнути помилок та забезпечити коректність інформації, яку надсилає користувач.

Очікується, що розроблений серверний застосунок інформаційної системи «Каса взаємодопомоги» покращить інтегрованість та автономність процесу функціонування руху солідарної фінансової підтримки, сприяючи розвитку благодійних ініціатив членів громади.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Термін «взаємодопомога» був популяризований філософом-анархістом Пітером Кропоткіним у його збірці есе «Взаємодопомога: фактор еволюції», де стверджувалося, що співпраця, а не конкуренція, була рушійним механізмом еволюції через біологічний мутуалізм [1, 2]. Кропоткін стверджував, що взаємодопомога має прагматичні переваги для виживання людей і тварин і сприяла природному відбору, і що взаємодопомога, можливо, є такою ж стародавньою, як і людська культура [2]. Це визнання широкого характеру та індивідуальної користі взаємодопомоги суперечило теоріям соціального дарвінізму, які наголошували на індивідуальній конкуренції та виживанні найпристосованіших, а також проти ідей лібералів, таких як Жан-Жак Руссо, які вважали, що співпраця є мотивованою всесвітньою любов'ю [3].

Учасники взаємодопомоги працюють разом, щоб визначити стратегії та ресурси, щоб задовольнити потреби один одного, такі як їжа, житло, медична допомога та допомога при стихійних лихах, одночасно організовуючи себе проти системи, яка спочатку створила дефіцит [4].

Як правило, групи взаємодопомоги організовані членами, очолюються її членами, та відкриті для всіх. Вони часто мають неієрархічні, небюрократичні структури, де члени контролюють усі ресурси. Вони мають егалітарний характер і розроблені для підтримки демократії участі, рівності статусу членів, спільного керівництва та прийняття рішень на основі консенсусу.[5]

За визначенням радикального активіста та письменника Діна Спейда та дослідженого в його курсі Чиказького університету «Взаємодопомога квір та транс для виживання та мобілізації», взаємодопомога відрізняється від благодійності [6]. Радикальний активіст, вчений із питань соціального забезпечення та соціальний працівник Бенджамін Шепард визначає

взаємодопомогу як явище, при якому «люди, які дають, що можуть, і отримують те, що їм потрібно». [7] Проекти взаємодопомоги часто критикують модель благодійності та можуть використовувати девіз «солідарність, а не благодійність», щоб відрізнити себе від благодійних організацій.

Проблеми взаємодопомоги:

1. Відсутність технічних експертів, фінансування та легітимізації з боку громадськості [8].
2. Відсутність персоналу на повний робочий день може обмежити обсяг роботи, яку можна виконати, особливо роботу, яку потрібно виконувати в традиційний робочий час.
3. Неформальний статус може позбавити права на державні субсидії та податкові пільги.
4. Розвиток концентрованих соціальних ієрархій може призвести до дискримінації та відходу від принципів взаємодопомоги [9].
5. Вигорання тих, хто може допомогти підтримувати проекти взаємодопомоги.
6. Учасники виснажують ресурси швидше, ніж вони поповнюються.

Вирішити перелічені проблеми можуть допомогти сучасні інформаційні системи, шляхом автоматизації процесів управління фондом взаємодопомоги, спрощення та прискорення обробки звернень учасників та оформлення вхідних та вихідних платежів. Розроблений застосунок повинен надавати потужний та ефективний функціонал для всіх осіб, які залучені до каси взаємодопомоги, як для учасників ініціативи, так і для персоналу, що здійснює підтримку життєдіальності проекту та обслуговує користувачів. Серверний застосунок інформаційної системи, що розробляється у даній кваліфікаційній роботі, спрямований на обробку запитів від мобільних застосунків учасників благодійних програм, та від адміністративного сайту інфомаційної системи фонду, і призначений спеціально для конкретної організації "Каса взаємодопомоги".

1.2 Аналіз аналогів серверних застосунків інформаційних систем

Перед початком проектування застосунку необхідно провести пошук і аналіз існуючих аналогічних програмних продуктів, що належать до категорії серверних застосунків інформаційних систем, що містять у складі як мобільні, так і вебзастосунки. У результаті було виявлено, що подібні функції та призначеня мають системи MbaaS.

MbaaS (Mobile Backend as a Service – Мобільний Бекенд як Сервіс) – сервіси та хостинг для розробників мобільних програм, що виконують рутинні операції. До них відноситься зв'язок програмного забезпечення (ПЗ) з бекенд-сервером, управління користувачами, відправлення push-повідомлень, інтеграція з популярними соціальними мережами і так далі. Додатковими послугами тут можуть бути засоби розробки, адміністрування та налаштування програм.

Ключова перевага MBaaS у тому, що бекенд не доведеться розробляти з нуля. Сервіси автоматично керують API (Application Program Interface), забезпечують цілісність інформації у базах даних (БД) та підтримують безперебійну роботу додатків на різних пристроях. З'явилися такі рішення порівняно недавно, але багато експертів зі сфери інформаційних технологій (IT) вже вважають, що за ними майбутнє мобільної розробки.

Нижче наведений перелік найпопулярніших у 2024 році постачальників і інструментів MbaaS [10] з акцентом на їхні ключові функції та типи проектів, для яких ці інструменти підходять.

1. Firebase [11]

Firebase — одна з найпопулярніших платформ безсерверної розробки для Android, iOS і веб-додатків. Він пропонує широкий спектр інструментів і послуг, які спрощують розробку серверної частини для мобільних і веб-додатків.

Ключові особливості:

- база даних NoSQL у реальному часі для зберігання та синхронізації даних;
- безпечна система автентифікації, яка підтримує електронну пошту/пароль, вхід у соціальні мережі (Google, Facebook) і спеціальні методи автентифікації;
- хмарне сховище Firebase для вмісту, створеного користувачами, наприклад зображень, відео та файлів;
- Firebase Cloud Messaging для надсилання сповіщень, повідомлень або оновлень мобільним і веб-клієнтам;
- Firebase Crashlytics для відстеження, звітування та визначення пріоритетів збоїв і проблем програм на різних платформах.

Firebase складається з багатьох різних серверів, які можна використовувати незалежно. Якщо повна функціональність платформи не потрібна, можна використовувати лише деякі функції, не вибираючи повний пакет.

Для складних великомасштабних програм можливості запитів Firebase можуть бути обмеженими. Цінова модель Firebase робить його дорогим варіантом для таких проектів, як роздрібні програми, які передбачають великий обсяг транзакцій бази даних.

Підходить для: малих і середніх проектів

2. AWS Amplify [12]

AWS Amplify — це універсальна колекція бібліотек, компонентів інтерфейсу користувача та інтерфейсу командного рядка (CLI). Amplify віддають перевагу командам HyperTrack, Orangetheory і Branch.

Ключові особливості:

- вбудоване керування користувачами;
- фреймворк для надсилання push-повідомлень мобільним і веб-клієнтам;
- кілька варіантів зберігання даних;

- підтримка можливостей AI/ML, таких як розпізнавання зображень, інтерпретація тексту та транскрипція мовлення в додатках iOS і Android.

AWS Amplify є досить складним і потребує швидкого навчання. Це може стати перешкодою для компаній, які прагнуть скоротити час виходу на ринок.

Підходить для: великомасштабних проектів із суворими вимогами до якості та безпеки, а не для стартапів.

3. Heroku [13]

Heroku — це хмарна платформа, яка надає широкий спектр послуг для розміщення, розгортання та масштабування веб-додатків і мобільних серверів. Хоча традиційно не класифікується як платформа MBaaS, вона пропонує багато функцій і послуг, які можна використовувати для створення та розміщення серверної частини для вашого мобільного додатка.

Ключові особливості:

- серверний код і розгортання API з різними мовами програмування
- Інтегровані постачальники баз даних, такі як Heroku Postgres і Heroku Redis для кешування;
- безпечна автентифікація за допомогою сторонніх постачальників ідентифікаційної інформації;
- автоматичне масштабування та балансування навантаження для обробки різних рівнів трафіку.

Heroku зручний, підтримує кілька мов програмування та пропонує параметри масштабованості. Однак витрати можуть збільшуватися, особливо з численними додатками та службами баз даних.

Підходить для: малих і середніх проектів або MVP.

4. Backendless [14]

Backendless — одна з найкращих платформ MBaaS, яка конкурує з провідними гравцями на ринку. Він може похвалитися солідною клієнтською базою, включаючи підприємства малого та середнього бізнесу та компанії зі списку Fortune 500, такі як Accenture, Dell і Capgemini. Backendless підтримує

широкий спектр клієнтських мов, включаючи Java, Swift, Objective-C, JavaScript, Flutter і .NET.

Його набір функцій включає:

- безпечну автентифікацію, реєстрацію та керування користувачами;
- зберігання геопросторових даних і запити;
- обмін повідомленнями та чат у режимі реального часу;
- хмарне зберігання файлів і керування ними;
- масштабовану доставку push-повідомлень для залучення користувачів.

Backendless також пропонує спеціалізовані версії для великих підприємств, такі як Backendless Pro і Managed Backendless. На даний момент остання версія Backendless (7.0.4.8) також містить конструктор інтерфейсу користувача, розробку на кількох фреймворках і API транзакцій.

Підходить для: широкого спектру проектів, від невеликих стартапів до великих підприємств; особливо корисний для програм, які вимагають геолокації та обміну повідомленнями в реальному часі.

5. Progress Kinvey [15]

Progress Kinvey — це корпоративне рішення MBaaS, призначене для оптимізації розробки та розгортання iOS, Android, веб-додатків і прогресивних веб-додатків. Його використовують такі компанії, як Schneider Electric і VMware.

Ключові особливості:

- push-повідомлення;
- аутентифікація та керування користувачами;
- управління даними Інтернету речей;
- спеціалізовані мікросервіси для таких галузей, як охорона здоров’я, страхування та виробництво.

Progress Kinvey вирізняється своїми моделями диференційованого ціноутворення, що робить його економічно ефективним для індивідуальних розробників, малого та середнього бізнесу та великих корпорацій. Розширена технічна документація та велика база знань забезпечують доступність для розробників із різним рівнем досвіду.

Підходить для: підприємств різного розміру, яким потрібні такі функції, як push-сповіщення, керування даними Інтернету речей і галузеві мікросервіси.

6. Back4App [16]

Back4App — це платформа MBaaS, яка пропонує SDK для Swift, Android Studio, JavaScript, React Native, GraphQL та інших поширених середовищ розробки. Back4App значною мірою покладається на Parse — одну з найпопулярніших платформ для бекенд-розробки.

Набір інструментів мобільної серверної розробки Back4App включає:

- push-повідомлення;
- автоматичні електронні листи;
- вхід через соціальні мережі;
- базу даних електронних таблиць для легкого створення, оновлення та синхронізації даних;
- спеціалізований інструмент CLI для легкого переходу від Parse до Back4App.

Back4App дозволяє вибирати між спільним і спеціальним хостингом програмного забезпечення.

Підходить для: різних проектів мобільних додатків, від MVP до додатків корпоративного рівня

7. Kumulos [17]

Kumulos — це надійна платформа, відома для керування продуктивністю мобільних додатків.

Він пропонує:

- звіти про збої та діагностику;

- оптимізацію магазину додатків;
- SDK для основних середовищ.

Kumulos має високу масштабованість із простим інтерфейсом із функцією перетягування та можливістю збирати та використовувати власні дані для оптимізації продуктивності, створення персоналізованого досвіду користувача та вдосконалення програми. Він також пропонує спеціалізовану інфраструктуру MBaaS для підприємств із особливими вимогами, такими як виділені сервери, спеціальні заходи безпеки, параметри масштабованості та користувальницькі інтеграції API.

Підходить для: підприємств, яким потрібне надійне керування продуктивністю додатків і аналітика, а також можливості розробки на стороні мобільного сервера.

Незважаючи на очевидні переваги, MBaaS – готові продукти, що знижує гнучкість розробки. Щоб послуги приносили користь, треба чітко уявляти, як домогтися від їхньої роботи максимальної ефективності. Часто простіше орендувати виділений сервер, налаштувати під конкретний проект і не залежати від сторонніх постачальників послуг. Більшість розробників популярних програм так і вчиняють. Тому актуальною задачею є розробка серверного застосунку інформаційної системи «Каса взаємодопомоги», який би забезпечував потужний та функціональний інструмент для кооперації віддалених підсистем, що входять до складу «Каси взаємодопомоги».

1.3 Мета та задачі серверного застосунку інформаційної системи «Каса взаємодопомоги»

Мета даної кваліфікаційної роботи полягає у розробці серверного застосунку інформаційної системи «Каса взаємодопомоги». Цей застосунок забезпечить злагоджену та безперебійну спільну роботу адміністративного сайту фонду, мобільних застосунків його учасників, та сховища даних. Він

надасть можливість здійснювати авторизація користувачів, обробку запитів, завантаження файлів на сервер системи.

Також, враховуючи основну мету цієї програми, серверний застосунок «Каси взаємодопомоги» буде використовуватись для інтеграції платіжних систем та керування платежами учасників фонду.

Розроблений застосунок має забезпечувати виконання наступних функцій:

- відправка SMS-повідомлень для авторизації на сервері мобільного застосунку з використанням API зовнішнього сервісу повідомлень;
- авторизація на сервері мобільного застосунку та на сервері адміністративного вебзастосунку з використанням механізму JWT;
- робота з профілем учасника благодійних програм на сервері системи (створення, перегляд, редагування, підтвердження, блокування, перегляд запитів, перегляд персональних файлів, перегляд виплат);
- обробка запиту персональних даних на сервері "Дія";
- обробка завантаження та доступу до файлів на сервері системи;
- обробка інформації про благодійні програми на сервері системи;
- обробка інформації про платежі на сервері системи (обробка інформації від банку про надходження коштів, перегляд реєстру платежів, внесення інформації про платежі вручну співробітниками фонду);
- робота з інформацією про користувачів адміністративної частини сайту на сервері системи (створення та редагування облікових записів, зміна паролів, блокування тощо).

Серверний застосунок «Каси взаємодопомоги» має бути розроблений відповідно до сучасних вимог надійності та функціональності. Програмне забезпечення повинно підтримувати виконання таких дій:

- обмін REST запитами з адміністративним вебзастосунком та мобільними клієнтами;
- авторизація користувачів;

- відправки СМС повідомлення на мобільний номер учасника;
- завантаження графічних файлів з мобільних пристроїв учасників;
- імпорт інформації з серверів державного підприємства “Дія”;
- використанням спільного сховища файлів та спільної бази даних для адміністративної частини системи, та мобільних застосунків користувачів.

Для досягнення поставленої мети і відповідно до вимог, необхідно виконати наступні завдання:

– здійснити експертний аналіз предметної області;
– вибрати необхідні інструменти для створення застосунку;
– розробити артітектуру програмного продукту;
– розробити відповідний функціонал застосунку;
– провести тестування застосунку.

Щоб забезпечити ефективну реалізацію проекту та виконання всіх вимог, необхідно мати детальне технічне завдання. Це є необхідною умовою для успішної реалізації проекту та досягнення поставлених цілей.

1.4 Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було розглянуто поняття фонду взаємодопомоги, а також вивчено особливості та складові серверних платформ та технологій, які можуть виступати у якості серверного застосунку інформаційної системи фонду. Було проведено аналіз семи інструментів MBaaS.

В результаті огляду цих платформ були виявлені наступні недоліки:

– обмежений обсяг транзакцій з базою даних;
– невелика гнучкість розробки;
– необхідність специфічних знань та навиків для налаштування та обслуговування;
– висока ціна.

Вищезазначені недоліки обумовлюють необхідність розробки серверного застосунку інформаційної системи «Каса взаємодопомоги», який би забезпечував потужний та функціональний інструмент для спільної роботи адміністративного сайту фонду, мобільних застосунків його учасників, та сховища даних.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВЕРНОГО ЗАСТОСУНКУ ІНФОРМАЦІЙНОЇ СИСТЕМИ «КАСА ВЗАЄМОДОПОМОГИ»

1.1 Функціонал та алгоритми функціонування застосунку

Зважаючи на те, що учасники благодійного фонду взаємодопомоги мусять мати можливість оперативного керування власною участю у благодійних програмах, а також повинні мати можливість подати запити на благодійну допомогу у зручний для себе спосіб, керівництвом благодійного фонду було прийнято рішення, що основним способом доступу учасників до інформаційної системи благодійного фонду повинен бути мобільний застосунок. Його функціонал має забезпечити весь спектр взаємодії між учасником та фондом.

Також, керівництвом фонду було прийнято рішення, що з боку фонду адміністрування його діяльності мусить відбуватися з адміністративного сайту, який уповноважені працівники фонду зможуть використовувати зі стаціонарних комп'ютерів, що належать фонду. Тобто фактично характер та тип програмного забезпечення, що мають використовувати учасники та персонал фонду, були визначені замовником самостійно, виходячи з власних міркувань і це стало вхідною вимогою до архітектури розроблюваної системи.

Враховуючи той факт, що функціонал доступний учасникам та адміністративному персоналу перетинається лише частково, і здебільшого пов’язаний з використанням спільного сховища файлів та спільної бази даних, в ході розробки було прийнято рішення розділити серверні застосунки, що обслуговуватимуть мобільні застосунки клієнтів та адміністративний сайт. Це дає можливість відокремити ці серверні частини та розмістити їх на різних апаратних чи віртуальних платформах під різними доменними іменами. Таким чином можна частково обезпечити кожну з цих частин серверного застосунку від наслідків взламу іншої. Крім того, так можна регулювати навантаження на відповідні сервери. Враховуючи, що переважна більшість сучасних серверних

застосунків збудовані за технологією REST, що продиктовано її перевагами зокрема легкістю розробки та впровадження, широкою підтримкою мовами програмування, масштабованістю тощо, обидві складові серверної частини було вирішено реалізовувати за допомогою технології REST, мови програмування Python та фреймворку Flask. Вибір мови програмування та фреймворку реалізації обумовлений широким розповсюдженням й, відповідно, невеликою кількістю відомих помилок, а також наявністю докладної документації та широкої спільноти розробників, яка підтримує інформаційні ресурси з матеріалами щодо використання цих продуктів.

Авторизація для мобільної частини системи виконується шляхом відправки СМС повідомлення на мобільний номер клієнта. Такий вибір методу авторизації був продиктований зворотньою сумісністю з наявним у фонда сайтом, який наразі не є частиною нової інформаційної системи, але мусить мати можливість брати з неї інформацію.

Враховуючи, що користувач повинен мати можливість надавати персональну інформацію для обробки співробітниками фонду, було прийнято рішення реалізувати два різні способи подання такої інформації в систему, а саме завантаження світлин з мобільних пристроїв учасників програм фонду , а також запит відповідної інформації з серверів державного підприємства “Дія”.

Подання запитів учасниками програм на отримання благодійної допомоги передбачає надання інформації про витрати, які потребують відшкодування. Файли світлин з інформацією про витрати мають бути збережені для подальшої обробки співробітниками фонду. Таким чином, частина сервера, що обслуговує мобільний застосунок, повинна мати доступ до файлового сховища де зберігаються відповідні дані. Інформація про сплату учасниками членських внесків надходить в систему з банківського сервера. Таким чином, частина сервера, що обслуговує мобільні застосунки, повинна забезпечити надходження відповідних даних про оплати.

Використання адміністративного сайту передбачає можливість роботи обмеженої кількості користувачів з невеликої кількості кінцевих пристроїв,

тому в якості засобу авторизації було використано механізм простого парольного захисту з подальшою видачею JWT токену доступу. Співробітники фонду мусять мати можливість переглядати файли надані учасниками програм, тому частина сервера що обслуговує адміністративний сайт повинна була мати можливість доступу до файлового сховища, яке використовує сервер мобільної частини.

Також обидві частини використовують спільну базу даних, яка функціонує під управлінням MySQL сервера, при цьому доступ до бази даних із застосунку Flask було вирішено реалізувати за допомогою фреймворку роботи з базами даних Peewee. Вибір цього фреймворку був обумовлений об’єктним стилем роботи з реляційними базами даних, а також простотою інтеграції та швидкістю використання.

2.2 Вибір технологій та мов програмування для створення програмного продукту

Для створення застосунку були обрані фреймворки Flask та Peewee, і мова програмування Python, СУБД MySQL та стандарт токена доступу JWT.

Flask [18] — це вебфреймворк, який дозволяє розробникам швидко й легко створювати легкі вебзастосунки за допомогою бібліотек Flask. Його розробив Армін Роначер, лідер Міжнародної групи любителів Python (POCCO). В основному він базується на наборі інструментів WSGI та системі шаблонів Jinja2.

Flask — це легка структура вебзастосунків WSGI. Він розроблений, щоб зробити початок роботи швидким і легким, з можливістю масштабування до складних програм. Він починався як проста обгортка навколо Werkzeug і Jinja і став одним із найпопулярніших фреймворків вебзастосунків Python.

Flask пропонує рішення, але не накладає жодних залежностей або компонування проекту. Розробник сам вибирає інструменти та бібліотеки, які

він хоче використовувати. Спільнота також пропонує численні розширення, які спрощують додавання нових функцій.

Peewee [19] — це проста і маленька система об'єктно-реляційного відображення (ORM). Цей фреймворк містить кілька концепцій, що робить його легким для вивчення та інтуїтивно зрозумілим у використанні. Переваги Peewee:

- Підтримка мов python 2.7+ і 3.4+
- Підтрмка СУБД sqlite, mysql, mariadb, postgresql і cockroachdb
- Наявність великої кількості розширень

Вихідний код Peewee розміщений на GitHub.

JWT [20], або веб-токен JSON, є відкритим стандартом (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Через відносно малий розмір JWT можна надіслати через URL-адресу, через параметр POST або всередині заголовка HTTP, і він швидко передається. JWT містить всю необхідну інформацію про сутність, щоб уникнути повторного запиту до бази даних. Одержувачу JWT також не потрібно викликати сервер для перевірки маркера.

Використання JWT має переваги порівняно з простими веб-токенами (SWT) і маркерами SAML:

Більш компактний: JSON є менш детальним, ніж XML, тому, коли він закодований, JWT менший, ніж маркер SAML. Це робить JWT хорошим вибором для передачі в середовищах HTML і HTTP.

Більш безпечний: JWT можуть використовувати для підпису пару відкритих/приватних ключів у формі сертифіката X.509. JWT також може бути симетрично підписаний спільним секретом за допомогою алгоритму HMAC. І хоча токени SAML можуть використовувати пари відкритих/приватних ключів, як-от JWT, підписати XML за допомогою цифрового підпису XML без введення незрозумілих прогалин у безпеці дуже складно порівняно з простотою підпису JSON.

Більш поширений: аналізатори JSON поширені в більшості мов програмування, оскільки вони відображаються безпосередньо на об’єкти. І навпаки, XML не має природного відображення документа в об’єкт. Це полегшує роботу з JWT, порівняно із твердженнями SAML.

Легший в обробці: JWT використовується в масштабі інтернету. Це означає, що його легше обробляти на пристроях користувачів, особливо мобільних.

2.3 Архітектура серверного застосунку «Каси взаємодопомоги»

Архітектура серверного застосунку «Каси взаємодопомоги» основана на паттерні BFF (Backend for Frontend pattern) [21]. BFF — це бекенд, який використовується певним зовнішнім додатком. Оскільки API кінцевих точок можуть мати кілька клієнтів із різними запитами, BFF може надати специфічний для клієнта серверний посередник і діяти як проксі, який пересилає та об’єднує кілька запитів до різних API служб [21]. Схема побудови паттерна показана на рис. 2.1.

Замість однієї точки входу він вводить кілька шлюзів. Завдяки цьому можна мати спеціалізований API, який відповідає потребам кожного клієнта (мобільний, Інтернет, комп’ютер, голосовий помічник тощо), і усунути багато проблем, спричинених збереженням усього в одному місці. Таким чином, можна використовувати acces_tokens для авторизації доступу до всіх API, але клієнти використовуватимуть сеансові файли cookie або токени у випадку мобільних пристроїв, а BFF буде проксі-сервером цих сторін [22].

BFF може зберігати токени в пам'яті та використовувати сеанс для керування ними, або зберігати маркери в зашифрованих файлах cookie однієї сторінки лише для HTTP [22]. BFF приховує непотрібні або конфіденційні дані перед передачею їх у зовнішній інтерфейс програми, тому ключі та маркери для сторонніх служб можна зберігати та використовувати з BFF.

Також BFF дозволяє надсилати відформатовані дані на зовнішній інтерфейс і через це може мінімізувати їх логіку. Крім того, використання BFF

надає можливості для покращення продуктивності та хорошої оптимізації для мобільних пристроїв.

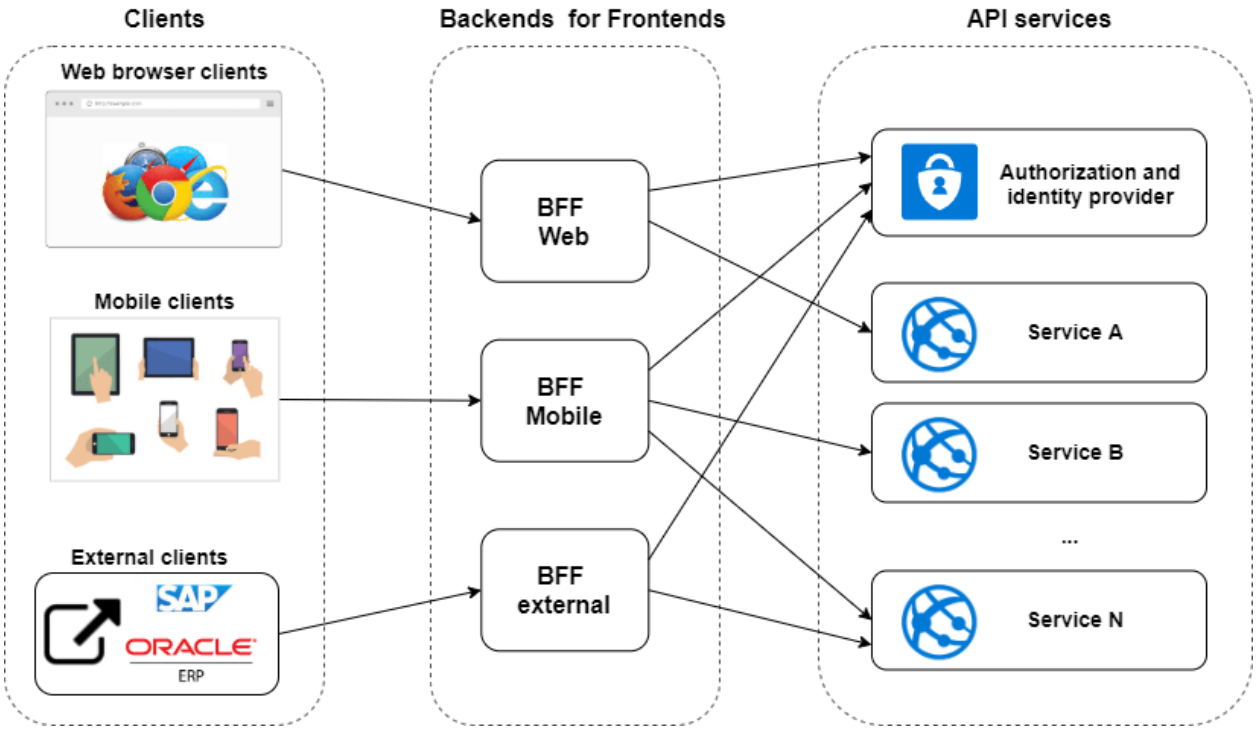

Рис. 2.1. Схема побудови застосунку за паттерном BFF [22]

В якості сервера баз даних було обрано MySQL сервер. Його вибір обумовлений тим, що за замовчуванням фреймворк Peewee підтримує SQLite, MySQL, MariaDB та Postgres. З цих варіантів MySQL забезпечує достатню високу продуктивність та споживає відносно небагато ресурсів обчислювальної системи. MariaDB в цьому варіанті є різновидом MySQL, тобто залежно від того який з цих серверів доступний у дереві пакетів операційної системи де буде розгорнуто програмний комплекс, таким чином базу даних можна переносити на MariaDB без змін.

При моделюванні бази даних, було виділено наступні сутності:

– Тип події - в системі можуть відбуватися події додавання нових об’єктів, редагування об’єктів, блокування об’єктів, зміна статусу об’єктів тощо;

– Подія - в системі має відбуватися реєстрація подій, які виконує будь-який користувач. Це має бути запобіжником від помилок та зловживань;

– Учасники - сутність учасника програми мусить містити базову інформацію про підписника програм благодійного фонду, таку як: ПІБ, ІПН, номер сертифіката учасника у благодійному фонді, дату народження, паспортні дані, а також додаткову інформацію для правильного відображення стану його облікового запису;

– Статус запиту - розрізняють чотири стани запитів учасників фонду: відкритий, закритий, очікує на реакцію користувача, очікує на реакцію співробітника фонду;

– Запити - ця сутність має містити інформацію про запит учасника програм благодійного фонду на благодійну допомогу, зокрема: опис, діагноз та суму призначену фондом у якості відшкодування за цим запитом;

– Файли - ця сутність дозволяє зберігати інформацію про файли завантажувані учасниками програм під час підтвердження їхнього облікового запису, а також у якості доданків до запитів;

– Типи ідентичності - для підтвердження облікових записів учасників програм, може бути використана різна персональна інформація така як: ідентифікаційний податковий номер, ID-карта, паперовий паспорт тощо. Ця сутність описує всі різновиди персональних даних, які можуть бути надані для підтвердження особистості учасника програм благодійного фонду;

– Ідентичність учасника - ця сутність описує надані учасниками програм персональні дані у вигляді файлів;

– Користувачі - адміністрування інформаційної системи збоку благодійного фонду здійснює група користувачів (адміністратори). Ця сутність відбиває інформацію про цих адміністраторів;

– Програми - ця сутність містить довідникову інформацію про програми благодійного фонду, зокрема про назву програми, її вартість на місяць, посилання на файл з умовами програми;

– Програми учасників - ця сутність описує конкретні програми на які в поточний момент часу підписані учасники;

– Типи платежів - благодійний фонд розрізняє три типи платежів: поповнення особового рахунку, виплата за запитом та часткова виплата за запитом. Ця сутність є довідником типів підтримуваних типів платежів;

– Платежі - ця сутність містить інформацію про платежі що надходили від учасників програм або були списані на користь учасників програм;

– Повідомлення запитів - в тому разі, якщо запит потребує уточнення, адміністратор благодійного фонду та користувач, можуть обмінюватися повідомленнями стосовно цього запиту. Ця сутність відбиває повідомлення стосовно запиту;

– Запит на вхід - під час авторизації учасників програм з мобільного застосунку, їм відправляється код підтвердження для входу. Ця сутність зберігає інформацію про запит на вхід в систему.

Відповідно до результатів моделювання було створено базу даних, фізична структура якої наведена на рисунку 2.2.

Поля таблиць у наведеній схемі мають наступне призначення:

Таблиця 2.1. – Поля таблиці Request_messages

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ повідомлення
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
message	TEXT	Текст повідомлення
request_id	INT	Зовнішній ключ запиту
member_id	INT	Зовнішній ключ учасника
user_id	INT	Зовнішній ключ користувача (адміністратора благодійного фонду)

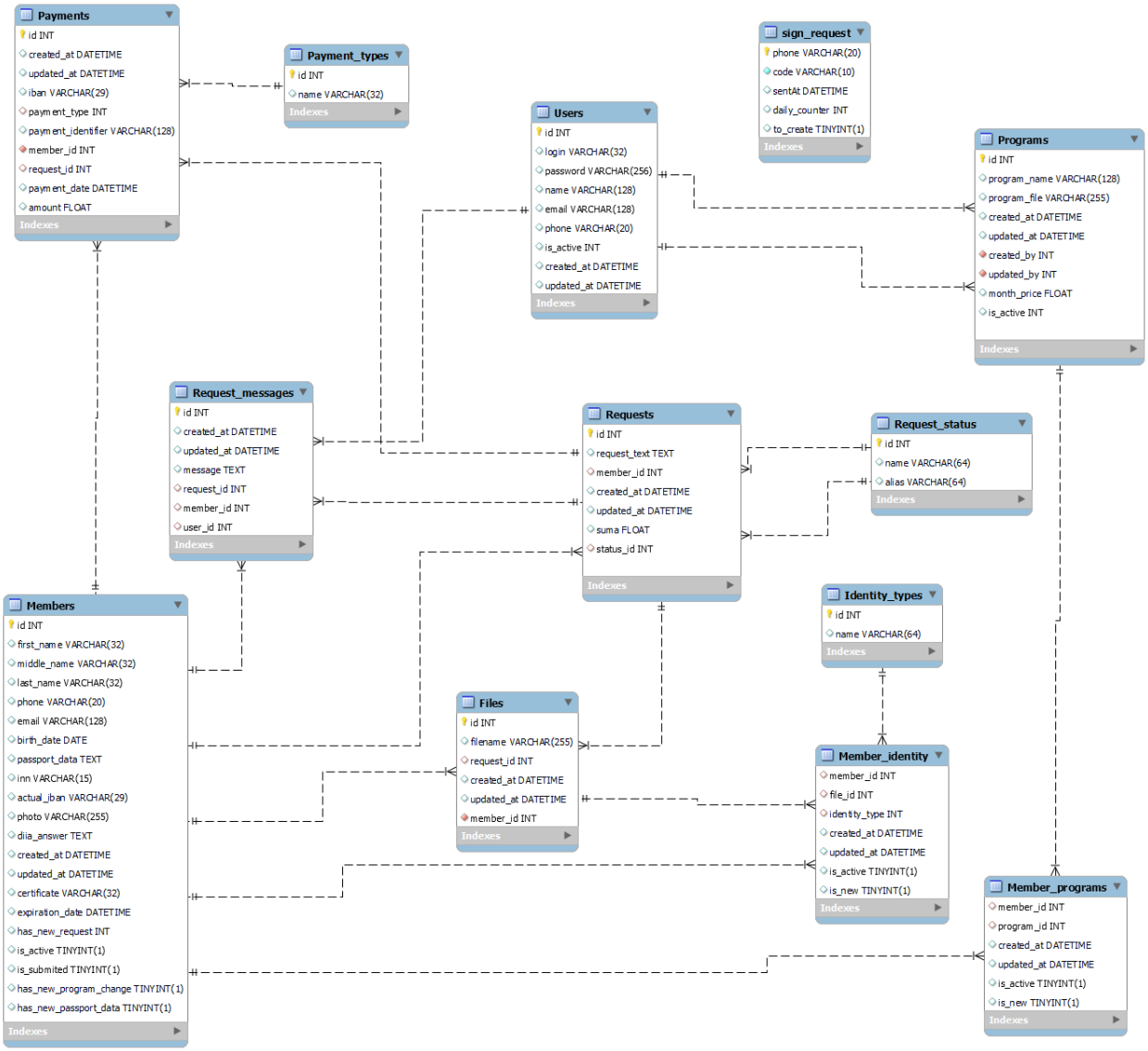

Рис. 2.2. Фізична структура бази даних

Таблиця 2.2. – Поля таблиці Members

Назва стовпчика	Тип даних	Призначення
id	INT	Первинний ключ учасника програм
first_name	VARCHAR(32)	Ім’я
middle_name	VARCHAR(32)	По-батькові
last_name	VARCHAR(32)	Прізвище
phone	VARCHAR(20)	Телефонний номер

Назва стовпчика	**Тип даних**	**Призначення**
email	VARCHAR(128)	Адреса електронної пошти
birth_date	DATE	Дата народження
passport_data	TEXT	Паспортні дані
inn	VARCHAR(15)	ІПН
actual_iban	VARCHAR(29)	Номер банківського рахунку
photo	VARCHAR(255)	Фото для відображення у мобільному застосунку
diia_answer	TEXT	У випадку заповнення персональних даних за допомогою сервісу “Дія”, повна відповідь від сервера “Дія”
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
certificate	VARCHAR(32)	Номер сертифіката учасника благодійних програм
expiration_date	DATETIME	Дата завершення участі у благодійних програмах (сплачено до)
has_new_request	INT	Прапорець наявності нових запитів від учасника
is_active	TINYINT(1)	Прапорець активації облікового запису учасника
is_submited	TINYINT(1)	Прапорець підтвердження наявності всіх необхідних для участі даних від учасника
has_new_program_change	TINYINT(1)	Прапорець зміни програми учасником
has_new_passport_data	TINYINT(1)	Прапорець зміни персональних файлів

Таблиця 2.3. – Поля таблиці Files

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ запису про файл
filename	VARCHAR (255)	Ім’я файлу
request_id	INT	Зовнішній ключ запиту
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата зміни запису
member_id	INT	Зовнішній ключ учасника

Таблиця 2.4. – Поля таблиці Users

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ користувача (адміністратора благодійного фонду)
login	VARCHAR (32)	Логін користувача
password	VARCHAR (256)	Пароль
name	VARCHAR (128)	Ім’я
email	VARCHAR (128)	Електрона адреса
phone	VARCHAR (20)	Телефонний номер
is_active	INT	Прапорець активації облікового запису
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата зміни запису

Таблиця 2.5. – Поля таблиці Request_status

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ статусу запиту
name	VARCHAR (64)	Назва статусу запиту
alias	VARCHAR (64)	Синонім імені статусу

Таблиця 2.6. – Поля таблиці Requests

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ запиту
request_text	TEXT	Текст запиту від учасника
member_id	INT	Зовнішній ключ учасника
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
suma	FLOAT	Затверджена фондом сума виплат за запитом
status_id	INT	Зовнішній ключ статусу запиту

Таблиця 2.7. – Поля таблиці Programs

Назва стовпчика	**Тип даних**	**Призначення**
id	INT	Первинний ключ програми
program_name	VARCHAR (128)	Назва програми
program_file	VARCHAR (255)	Повний URL файлу благодійної програми
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
created_by	INT	Зовнішній ключ користувача який створив запис
updated_by	INT	Зовнішній ключ користувача який оновив запис
month_price	FLOAT	Щомісячна плата за участь в програмі
is_active	INT	Прапорець активації програми

Таблиця 2.8. – Поля таблиці Member_program

Назва стовпчика	**Тип даних**	**Призначення**
member_int	INT	Зовнішній ключ учасника
program_id	INT	Зовнішній ключ програми
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
is_active	TINYINT (1)	Прапорець того, що ця програма є поточною для цього користувача

Назва стовпчика	Тип даних	Призначення
is_new	TINYINT (1)	Прапорець, що показує бажання учасника змінити програму

Таблиця 2.9. – Поля таблиці Payments

Назва стовпчика	Тип даних	Призначення
id	INT	Первинний ключ платежу
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
iban	VARCHAR (29)	Банківський рахунок
payment_type	INT	Тип платежу
payment_identifier	VARCHAR (128)	Номер замовлення за яким платіжний сервіс дозволяє ідентифікувати платіж
member_id	INT	Зовнішній ключ учасника
request_id	INT	Зовнішній ключ запиту
payment_date	DATETIME	Дата проведення платежу
amount	FLOAT	Розмір платежу

Таблиця 2.10. – Поля таблиці Payment_types

Назва стовпчика	Тип даних	Призначення
id	INT	Первинний ключ типу платежу
name	VARCHAR (32)	Назва типу платежу

Таблиця 2.11. – Поля таблиці Identity_types

Назва стовпчика	Тип даних	Призначення
id	INT	Первинний ключ типу ідентичності
name	VARCHAR (64)	Назва типу ідентичності

Таблиця 2.12. – Поля таблиці Member_identity

Назва стовпчика	**Тип даних**	**Призначення**
member_id	INT	Зовнішній ключ учасника
file_id	INT	Зовнішній ключ файлу
identity_type	INT	Зовнішній ключ тип ідентичності
created_at	DATETIME	Дата створення запису
updated_at	DATETIME	Дата оновлення запису
is_active	TINYINT (1)	Прапорець того, що цей файл ідентичності є поточним
is_new	TINYINT (1)	Прапорець того, що файл ідентичності щойно завантажений користувачем

Таблиця 2.13. – Поля таблиці sign_request

Назва стовпчика	**Тип даних**	**Призначення**
phone	VARCHAR (12)	Телефон для відправки СМС повідомлення
code	VARCHAR (6)	Код авторизації, який буде відправлено
sentAt	DATETIME	Час відправки
daily_counter	INT	Лічильник щоденної кількості спроб входу
to_create	TINYINT (1)	Прапорець створення нового запису учасника програм

2.4 Висновки до розділу 2

У другому розділі кваліфікаційної роботи було проведено проєктування серверного застосунку «Каси взаємодопомоги». Спочатку була розглянута структура та функціонал застосунку.

Був виконаний огляд мов сценаріїв та програмування, і прийнято рішення використовувати мову програмування Python. Також було розглянуто Python-сумісні фреймворки, і для створення програмного коду застосунку було обрано Flask. У якості основи для архутектури застосунку був обраний

паттерн BFF (Backend for Frontend). Для реалізації процедур авторизації було запропоновано використати веб-токен JWT, який є компактним і самодостатнім способом безпечної передачі інформації між компонентами інформаційної системи.

В якості сервера баз даних було обрано MySQL сервер, а для реалізації технології об’єктно-реляційного відображення – фреймворк Peewee. Було виконано моделювання бази даних, та складено детальний опис сутностей, таблиць та їх полів.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО КОМПОНЕНТУ СЕРВЕРНОГО ЗАСТОСУНКУ «КАСИ ВЗАЄМОДОПОМОГИ»

3.1 HTTP маршрутизація у застосунку

Згідно з функціональними вимогами, прописаними у п. 1.3 першого розділу кваліфіувційної роботи, та виходячи з архітектури системи, було прийнято рішення створити два окремі вебзастосунки, що призначенні для реалізації функціоналу обміну з мобільним застосунком системи та з адміністративним сайтом системи. Необхідність відокремити функціонал цих застосунків полягає в тому, що вони функціонуватимуть на різних віртуальних або апаратних платформах і матимуть тільки спільну базу даних та файлове сховище. В такому випадку, навантаження на сервер мобільного застосунку не впливатиме на роботу працівників благодійного фонду.

Розроблене програмне забезпечення функціонує за технологією REST, забезпечуючи JSON відповіді на запити до певних функцій системи з боку мобільного застосунку та адміністративного сайту системи. Для обслуговування запитів мобільного застосунку, було створено наступний план HTTP маршрутизації:

1. *URL:* /files/<uid>/<filename>

   Метод: GET

   Вхідні параметри: uid, filename

   Результат: Файловий потік

   Призначення: Запит файлу filename для користувача з ідентифікатором uid

2. *URL:* /.well-known/apple-app-site-association

   Метод: GET

   Вхідні параметри: -

   Результат: Повертає об’єкт, що містить динамічне посилання для користувачів Apple IOS задля переходу у сервіс “Дія”

Призначення: Для користуачів IOS повертає так званий deeplink - динамічне посилання для переходу із зовнішнього застосунку на певний екран поточного застосунку. У випадку розробленого програмного забезпечення це посилання повертає потік виконання на екран з повідомленням про результати запиту персональних даних з сервісу “Дія”

3. *URL:* /user/signrequest

Метод: POST

Вхідні параметри: phone: str, create: Optional[bool]

Результат: Результатом є об’єкт, де в полі status вказано результат виконання, а полі message наведено пояснення результату

Призначення: Користувач, запускаючи застосунок, мусить пройти аутентифікацію через код що відправляється йому у вигляді SMS. Якщо користувач вперше заходить у застосунок, поле create повинно мати значення true, щоб обліковий запис користувача було створено

4. *URL:* /user/signin

Метод: POST

Вхідні параметри: phone: str, code: str

Результат: Повертає електронний квиток доступу access_token в тому випадку, якщо користувача авторизовано або об’єкт зі статусом “помилка” та поясненням її причини

Призначення: В тому разі, якщо користувач вказав правильний авторизаційний код з SMS, сервер має повернути йому електронний квиток доступу access_token сформований за алгоритмом JWT

5. *URL:* /user/profile/create

Метод: POST

Вхідні параметри: first_name: str, middle_name: str, last_name: str, email: EmailStr, program_id: int, iban: str, id_1: Optional[FileStorage], id_2: Optional[FileStorage], inn: FileStorage, book_1: Optional[FileStorage], book_2: Optional[FileStorage], book_address: Optional[FileStorage]

Результат: Повертає об’єкт з повідомленням результату операції зі створенням облікового запису користувача

Призначення: Після реєстрації (першого входу у застосунок) користувач має повідомити персональну інформацію та завантажити персональні файли аби обробку співробітники фонду змогли підготувати угоду та інші обов’язкові фінансові документи до неї. Протягом виконання цього запиту, користувач заповнює інформаційні поля з відомостями про себе та завантажує файли персональних даних

6. *URL:* /user/profile/edit

Метод: POST

Вхідні параметри: email: EmailStr, phone: str, photo: Optional[FileStorage]

Результат: Повертає об’єкт з описом результату операції оновлення

Призначення: Користувач має сам слідкувати за актуальністю персональних файлів. У випадку зміни прізвища або імені, заміни паспорта, банківського рахунку тощо, користувач має повідомити сервер системи через виклик цієї функції

7. *URL:* /user/profile/wrong

Метод: POST

Вхідні параметри: first_name: str, middle_name: str, last_name: str, birth_date: str

Результат: Повертає об’єкт з результатом повідомлення про помилку

Призначення: Деякі поля профілю користувача мають бути відредаговані співробітниками фонду. Якщо користувач помічає помилку співробітника, то він може повідомити про це викликом цієї функції

8. *URL:* /user/profile/get

Метод: GET

Вхідні параметри: -

Результат: Повертає об’єкт з даними профіля поточного користувача або об’єкт з повідомленням про помилку

Призначення: Для відображення інформації для мобільного застосунку, запит за цим URL повертає всю необхідну інформацію

9. *URL:* /program/list

Метод: GET

Вхідні параметри: -

Результат: Повертає об’єкт зі списком інформації про поточні програми

Призначення: Для відображення інформації про активні програми, мобільний застосунок потребує список об’єктів із зазначенням назви програми, обсягу внесків на місяць та посиланням на файл з умовами програми

10. *URL:* /program/change

Метод: POST

Вхідні параметри: program_id: int

Результат: Повертає об’єкт з результатами зміни програми для поточного учасника або об’єкт з повідомленням про помилку

Призначення: Якщо користувач хоче перейти від участі в одній благодійній програмі до участі в іншій благодійній програмі, він може зробити відповідну заміну

11. *URL:* /user/charges/list

Метод: POST

Вхідні параметри: begin: str, end: str

Результат: Повертає об’єкт зі списком виплат за запитами учасника програм за обраний період часу або об’єкт з повідомленням про помилку

Призначення: Користувач мобільного застосунку може переглянути список виплат з боку благодійного фонду за запитами поданими користувачем

12. *URL:* /user/payments/list

Метод: GET

Вхідні параметри: -

Результат: Повертає об’єкт зі списом внесків користувача за участь у програмах благодійного фонду або об’єкт з повідомленням про помилку

Призначення: Користувач мобільного застосунку може переглянути інформацію про зроблені ним внески за участь у благодійних програмах

13. *URL:* /user/request/list

Метод: POST

Вхідні параметри: status: Optional[List[str]]

Результат: Повертає список запитів учасника програм благодійного фонду або об’єкт з повідомленням про помилку

Призначення: Для відображення у мобільному застосунку необхідно повертати список з об’єктами запитів користувача на благодійну допомогу

14. *URL:* /user/request/<id>

Метод: GET

Вхідні параметри: id

Результат: Повертає об’єкт запиту з ідентифікатором id або об’єкт з повідомленням про помилку

Призначення: Для перегляду деталей конкретного запиту необхідно викликати функцію за цим url передавши в ньому ідентифікатор бажаного запиту. Буде повернуто об’єкт з інформацією про запит та призначеною сумою виплати, якщо таке призначення відбулось

15. *URL:* /user/request/<id>/message

Метод: POST

Вхідні параметри: id, message: str

Результат: Повертає об’єкт з результатами відправки повідомлення за запитом

Призначення: Якщо запит потребує додаткового листування, функція за цим URL дозволяє відправити повідомлення від користувача мобільного застосунку співробітникам фонду

16. *URL:* /user/request/create

Метод: POST

Вхідні параметри: comment: str, files: Optional[List] #[FileStorage]]

Результат: Повертає об’єкт з результатами створення запиту або об’єкт з повідомленням про помилку

Призначення: Виклик функцій за цим URL створює новий запит від користувача мобільного застосунку

17. *URL:* /user/request/update

Метод: POST

Вхідні параметри: id: int, files: Optional[List]

Результат: Повертає об’єкт з результатами оновлення файлів доданих до запиту або об’єкт з повідомленням про помилку

Призначення: За умовами роботи фонду, кожен запит має містити файли з підтвердженням витрат які учасник програм фонду подає на відшкодування. У разі нестачі або поганої якості світлин певних документів, їх можна додати до запиту викликавши функцію за цим URL

18. *URL:* /user/payments\/liqpay

Метод: POST

Вхідні параметри: 'action': 'pay', 'amount': request.json["amount"], 'currency': 'UAH', 'description': 'Payment for order', 'order_id': order_id, 'version': '3', 'sandbox': 1, 'server_url': app.config["SERVER_URL"] + '/user/payments/liqpay-callback',

Результат: Повертає об’єкт з результатами запиту на поповнення особового рахунку або об’єкт з повідомленням про помилку

Призначення: Сервер мобільного застосунку потребує сформованого та підписаного електронним ключем системи запиту від сервера системи до платіжного сервісу LiqPay на поповнення особового рахунку учасника програм. Функція за цим URL формує такий об’єкт та повертає мобільному застосунку

19. *URL:* /user/payments/liqpay-callback

Метод: POST

Вхідні параметри: Об’єкт з результатами сплати внесків

Результат: Результат обробки інформації від сервісу LiqPay

Призначення: Сервіс LiqPay повертає результат обробки запиту на поповнення особового рахунку безпосередньо на сервер системи за цим URL

20. *URL:* /user/profile/diia

Метод: POST

Вхідні параметри: -

Результат: Результат об’єкт для запиту на сервер систем ”Дія”

Призначення: Під час створення профіля користувача, його персональні дані можуть бути завантажені з сервісу “Дія”. Для цього сервер системи формує запит на сервер “Дія”, викликаючи функцію за цім URL

21. *URL:* /user/profile/diia-callback

Метод: POST

Вхідні параметри: encodeData: str, files: Optional[List]

Результат: Результат зашифрований об’єкт з персональними даними користувача включно з файлами що містять аверс та реверс ID-карти та ідентифікаційний номер користувача

Призначення: Система “Дія” спрямовує персональні дані користувача безпосередньо на сервери системи благодійного фонду. Обробка відповіді від “Дія” виконується функцією за цим URL.

Для обслуговування адміністративного сайту системи так само використовується вебзастосунок, побудований на базі REST, план HTTP маршрутизації якого наведено нижче:

1. *URL:* /files

Метод: POST

Вхідні параметри: user_id: Optional[int], filename: str

Результат: Файловий потік або об’єкт з повідомленням про помилку

Призначення: Цей маршрут викликає код який дозволяє приховати файлову структуру але завантажувати файли для перегляду

2. *URL:* /program/<filename>

Метод: GET

Вхідні параметри: filename

Результат: Файл програми або повідомлення про помилку 404 якщо такого файлу нема

Призначення: Цей маршрут викликає код для перегляду файлу конкретної програми благодійного фонду

3. *URL:* /signin

Метод: POST

Вхідні параметри: login: str, password: str

Результат: Об’єкт з електронним квитком доступу access_token або об’єкт з повідомленням про помилку

Призначення: Для авторизації користувачів адміністративного сайту благодійного фонду викликається метод за цим маршрутом

4. *URL:* /user; /user/<user_id>

Метод: GET

Вхідні параметри: user_id (необов'язковий)

Результат: За умови відсунтності параметра id повертається список усіх користувачів адміністративного сайта, за наявності параметра id повертається об’єкт користувача з ідентифікатором “id” або об’єкт з повідомленням про помилку

Призначення: Функція за цим маршрутом призначена для читання списку всіх користувачів або інформації про окремого користувача. У разі, якщо користувач незнайдений, виникає помилка 404

5. *URL:* /user

Метод: PUT

Вхідні параметри: email: EmailStr, is_active: bool, login: str, name: str, phone: Optional[str]

Результат: Об’єкт з повідомленням про успіх та даними створеного корисувача або об’єкт з повідомленням про помилку

Призначення: За цим маршрутом викликається код, що створює нового користувача

6. *URL:* /user

Метод: POST

Вхідні параметри: id: int, email: Optional[EmailStr], is_active: Optional[bool], login: Optional[str], name: Optional[str], phone: Optional[str]

Результат: Об’єкт з повідомленням про успіх та даними про внесені зміни в обліковий запис користувача або об’єкт з повідомленням про помилку

Призначення: За цим маршрутом викликається метод для зміни даних облікового запису користувача

7. *URL:* /user/<id>

Метод: DELETE

Вхідні параметри: id

Результат: Об’єкт з повідомленням про успіх або об’єкт з повідомленням про помилку

Призначення: За цим маршрутом викликається функція, що дозволяє фізично видалити обліковий запис користувача з бази даних

8. *URL:* /user/change_password

Метод: POST

Вхідні параметри: id: Optional[int], old_password: str, new_password: str

Результат: Об’єкт з повідомленням про успіх або об’єкт з повідомленням про помилку

Призначення: За цим маршрутом викликається функція зміни пароля користувача

9. *URL:* /members

Метод: POST

Вхідні параметри: filter: Dict

Результат: Список об’єктів учасників програм благодійного фонду або об’єкт з повідомленням про помилку

Призначення: Функція за цим маршрутом дозволяє отримати список учасників програм, що відповідають параметрам встановленого фільтру

10. *URL:* /member/generate_certificate

Метод: GET

Вхідні параметри: -

Результат: Об’єкт з номером сертифіката для нового користувача або об’єкт з повідомленням про помилку

Призначення: Ідентифікація користувачів у внутрішній документації благодійного фонду відбувається за номерами благодійних сертифікатів. Функція за цим маршрутом генерує новий номер благодійного сертифікату

11.*URL:* /member

Метод: POST

Вхідні параметри: id: int

Результат: Об’єкт з інформацією про обліковий запис учасника програм благодійного фонду або об’єкт з повідомленням про помилку

Призначення: Функція за цим маршрутом повертає деталі профіля учасника програм благодійного фонду, список завантажених ним персональних файлів, список його запитів, та список фінансових операцій щодо цього учасника

12.*URL:* /member

Метод: PUT

Вхідні параметри: id: int, first_name: Optional[str], middle_name: Optional[str], last_name: Optional[str], email: Optional[EmailStr], iban: Optional[str], birth_date: Optional[str], certificate: Optional[str], passport_data: Optional[str], inn: Optional[str], phone: Optional[str], is_active: Optional[bool], is_submited: Optional[bool]

Результат: Об’єкт з повідомленням про успіх зміни інформації про обліковий запис учасника програм або об’єкт з повідомленням про помилку

Призначення: Дозволяє змінити доступну для редагування адміністратором інформацію про обліковий запис учасника благодійних прогрм

13.*URL:* /programs

Метод: GET

Вхідні параметри: -

Результат: Повертає список об’єктів активних програм благодійного фонду або об’єкт з повідомленням про помилку

Призначення: Список за цим маршрутом повертає список актуальних благодійних програм благодійного фонду

14.*URL:* /program

Метод: POST

Вхідні параметри: id: int

Результат: Повертає об’єкт з інформацією про поточну програму або об’єкт з повідомленням про помилку

Призначення: Функція за цим маршрутом повертає інформацію про обрану програму

15.*URL:* /program

Метод: PUT

Вхідні параметри: id: Optional[int], month_price: Optional[float], program_file: Optional[str], new_program_file: Optional[FileStorage], program_name: Optional[str], is_active: Optional[bool]

Результат: Повертає об’єт з інформацією про створення чи зміну обраної програми або об’єкт з повідомленням про помилку

Призначення: Функція за цим маршрутом призначена для створення записів нових програм або зміни наявних

16.*URL:* /payment

Метод: PUT

Вхідні параметри: member: int, type: int, request: Optional[int], date: str, amount: float

Результат: Повертає об’єкт з інформацією про успіх, створення запису про новий платіж або об’єкт з повідомленням про помилку

Призначення: Загалом в системі платежі створюються автоматично завдяки використання зовнішнього платіжного сервісу та пов’язаного з ним коду. Але на той випадок, якщо надійде платіж з іншого джерела, яке не відстежується інформаційною системою, або у випадку виплати за запитом

учасника програм, необхідно викликати функцію за цим маршрутом, аби внести інформацію про цей платіж вручну

17.*URL:* /payments

Метод: POST

Вхідні параметри: type: Optional[List[int]], date_start: Optional[str], date_end: Optional[str], amount_start: Optional[float], amount_end: Optional[float], certificate: Optional[str], offset: int = 0, limit: int = 50

Результат: Повертає список об’єктів з інформацією про платежі або об’єкт з повідомленням про помилку

Призначення: Для перегляду списку платежів що відповідають параметрам фільтрації, треба викликати функцію за цим маршрутом

18.*URL:* /requests

Метод: POST

Вхідні параметри: query: Optional[str], date_start: Optional[str], date_end: Optional[str], status: List[str] = ['opened'], limit: int = 50, offset: int = 0

Результат: Повертає список об’єктів із запитами учасників програм або об’єкт з повідомленням про помилку

Призначення: Задля перегляду інформації про запити користувачів, що відповідають параметрам фільтрації, необхідно викликати функцію за цим маршрутом

19.*URL:* /request

Метод: PUT

Вхідні параметри: id: int, request_text: str, diagnos: str, suma: float, member: int

Результат: Повертає об’єкт з інформацією про успіх в оновленні запису про запит учасника програм або об’єкт з повідомленням про помилку

Призначення: Адміністратори благодійного фонду вручну призначають суму виплати за запитом, а також запитують додаткову інформацію учасника програм, якщо наданих даних недостатньо

20.*URL:* /request/message

Метод: POST

Вхідні параметри: request_id: int, message: str

Результат: Повертає об’єкт з інформацією про успіх про створення нового повідомлення або об’єкт з інформацією про помилку

Призначення: Для створення нового повідомлення при обробці певного запиту, від учасника програм, треба викликати функцію за цим маршрутом

21.*URL:* /request

Метод: POST

Вхідні параметри: request: int, member: Optional[int]

Результат: Повертає об’єкт з інформацією про поточний запит учасника програм або об’єкт з повідомленням про помилку

Призначення: Задля отримання інформації про певний запит учасника програм, включно із доданими ним файлами, листуванням та переліком виплат, необхідно викликати функцію за цим маршрутом.

3.2 Особливості програмної реалізації застосунку

Фреймворк Flask передбачає використання різних парадигм програмування, зокрема можна використовувати об’єктну парадигму але для виконання цієї роботи, було використано функціональне програмування, оскільки виклик кожного маршруту є автономним, а технологія побудови самого застосунку відповідає специфікації REST.

Фактично, частини серверного застосунку, які обслуговують підключення мобільних клієнтів та адміністративного сайту є окремими Python програмами, які здебільшого використовують одну і саму ідеологію обробки даних. По перше, там де це можливо, замість GET запитів, вживались POST та PUT запити, особливо це стосується тих частин коду, які може модифікувати кінцевий користувач. Обробка помилок у запитах POST та PUT, в такому випадку, є простішою, ніж спроба перехопити та виправити помилково чи зумисне викривлені URL GET запитів. По друге, перевірка

вхідних даних відбувалась за допомогою засобів бібліотеки Pyantic. Цей інструментар дозволяє не тільки перевіряти дані окремо від основної логіки обробки запитів, але й за необхідності вносити зміни у спосіб їхнього подання та виконувати трансформацію запитів як з типом контенту “aplication/json” так із типом контенту “application/x-ww-form-urlencoded” в об'єктний вигляд. Таке перетворення дає можливість однаково використовувати вхідні дані, незалежно від способу їхнього подання.

Розгляньмо наприклад клас, що перевіряє вхідні дані під час авторизації. Текст програми, що реалізує відповідний клас перевірки, наведено у лістингу 3.1

Лістинг 3.1. Програмний код класу NewSignin

```
class NewSignin(BaseModel):
    phone: str
    code: str

    @validator("phone")
    def correct_phone(cls, p):
        regex = re.compile("\+\d{12}")
        r = regex.match(p)
        if not r:
            raise ValueError("Phone is invalid")
        return p
```

У цьому прикладі вхідний параметр запиту, що містить телефон користувача мобільного застосунку, перевіряється на правильність формату. Загалом, така організація коду, що ґрунтується на класах збудованих на базі Pydantic, дозволяє перевірити наявність обов’язкових полів та особливості будь яких полів на правильність, а також виконати перетворення даних вхідного запиту як до початку перевірки, так і після нього.

3.2.1 Авторизація у мобільному застосунку

Логіка цієї операції передбачає, що користувач надсилає запит на вхід у систему, при цьому якщо це його перший вхід, вхідний запит мусить містити крім телефонного номера, ще прапорець “To create”. Якщо, вхідний запит правильний, користувачеві мобільного застосунку на вказаний номер телефону, має бути відправлено повідомлення з одноразовим паролем. Аби уникнути атак перебором, які можуть призводити до перенавантаження серверного застосунку та перевитрати коштів на відправку СМС повідомлень, у застосунку вводиться два обмеження: по перше, денний ліміт спроб входу. по друге, обмеження на час відправки наступного повідомлення (не раніше ніж n секунд після попереднього).

В тому разі, якщо одноразовий пароль з повідомлення вказаний правильно і решта умов також виконані, користувачеві повертається об’єкт, що містить електронний ключ доступу access_token, що формується за допомогою протоколу JWT.

Для реалізації безпосередньо відправки повідомлення, використовується сервіс Twilio. Цей сервіс має різні способи відправлення повідомлень але в цьому проекті використовується Python бібліотека Twilio, яка надає весь необхідний функціонал у вигляді конструкції Python. Код функції обгортки, для відправки СМС повідомлень, наведено у лістингу 3.2.

Лістинг 3.2. Програмний код функції send_sms

```
def send_sms(phone, code):
    account_sid = "ACbda4e78c1755ed162cc58844ca8b6f9f"
    auth_token = "ea01f5b6eb2b9a3fee84732de25f138a"
    client = Client(account_sid, auth_token)

    message = client.messages \
        .create(
            body = str(code),
            from_ = '+12513517573',
            to = phone
        )
```

Після отримання СМС з одноразовим паролем, користувач мобільного застосунку викликає функцію /user/signin. Після видачі JWT acess_tokes, его перевірка перед кожним викликом функцій, що потребують авторизованого доступу, виконуються за допомогою функцій декоратора, код якої наведено у лістингу 3.3.

Лістинг 3.3. Програмний код функції token_required

```
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        elif 'Authorization' in request.headers and
request.headers['Authorization'].strip().split(" ")[0] == "Bearer":
            token = request.headers['Authorization'].strip().split("
")[-1]
        if not token:
            return jsonify({'message' : 'Token is missing !!'}), 401
        try:
            data = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=['HS256'])
            current_user = Members.get(id = data['user_id'])
            kwargs.update({'current_user': current_user})
        except Exception as e:
            return jsonify({
                'message' : 'Token is invalid !!'
            }), 401
        return f( *args,**kwargs)

    return decorated
```

Схожим чином працює авторизація на адміністративному сайті за винятком того, що користувачі адміністративного сайту використовують сталі паролі. Узагальнена схема авторизації за допомогою JWT токена показана на рис. 3.1.

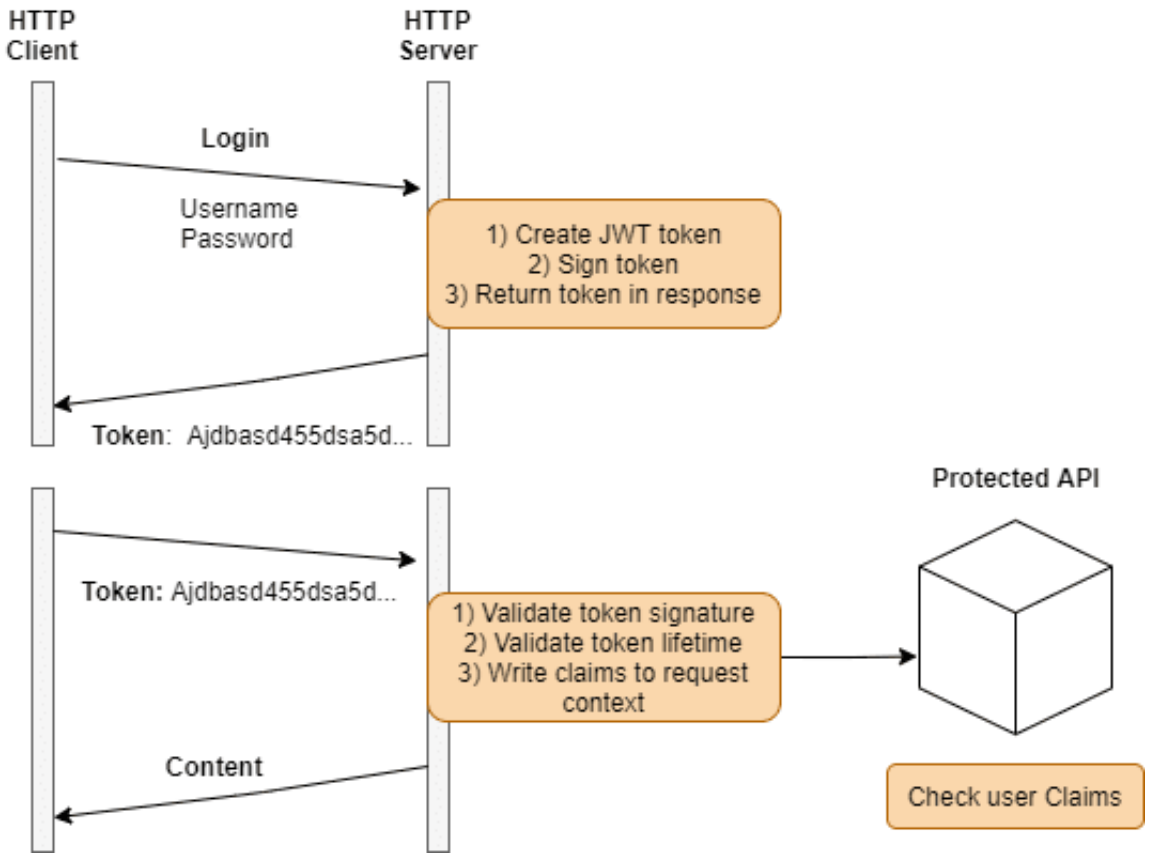

Рис. 3.1. Базовий приклад потоку токенів JWT

3.2.2 Оплата за допомогою сервіса LiqPay

Українські фінансові установи пропонують широкий спектр рішень з проведення онлайн платежів. Однак, платежі для комерційних установ коштують досить багато, як для благодійної установи. Тому з-поміж всіх гравців фінансового ринку був обраний сервіс LiqPay від ПриватБанку, оскільки він пропонує найвигідніші умови за наявності сервісу високої якості та цілодобової технічної підтримки.

Сервіс LiqPay пропонує проведення платежів з використанням технології REST API. Однак через те, що він приймає зашифровані дані для видачі платіжної сторінки, а ключ шифрування благодійного фонду є секретним і не підлягає передачі учасникам благодійних програм, було прийнято рішення створити функції які попередньо шифрують інформацію на сервері мобільного застосунку і повертають зашифровану послідовність

учаснику, який робить свій внесок у благодійну програму, в якій бере участь. Таким чином, відправивши зашифровану послідовність на сервер LiqPay, учасник отримує доступ до платіжної сторінки де може зробити перерахування. Узагальнена схема алгоритму цієї функції наведена на рисунку 3.2.

Програмний код цієї функції наведено у лістингу 3.4

Лістинг 3.4. Програмний код функції /user/payments/liqpay

```
@app.route("/user/payments/liqpay", methods=['POST'])
@token_required
def liqpay(**kwargs):
    current_user = kwargs["current_user"]
    liqpay = LiqPay(app.config["LIQPAY_PUBLIC_KEY"], app.config["LIQPAY_PRIVATE_KEY"])
    stamp = datetime.timestamp(datetime.now())
    order_id = str(stamp).replace(".", "") + str(current_user.id)

    print(request.url_root)

    params = {
        'action': 'pay',
        'amount': request.json["amount"],
        'currency': 'UAH',
        'description': 'Payment for order',
        'order_id': order_id,
        'version': '3',
        'sandbox': 1,
        'server_url': app.config["SERVER_URL"] + '/user/payments/liqpay-callback',
    }
    signature = liqpay.cnb_signature(params)
    data = liqpay.cnb_data(params)
    pid = PaymentTypes.get(name='Поповнення')
    Payments.create(payment_identifier=order_id, amount=float(request.json["amount"]), member=current_user.id, payment_type=pid, created_at=datetime.now(), updated_at=datetime.now())
    return jsonify({ 'signature': signature, 'data': data }), 200
```

Рис. 3.2. Узагальнена схема роботи функції /user/payments/liqpay

Друга API функція, яка реалізована у застосунку, відстежує відповідь від сервера LiqPay та подовжує участь того, хто зробив внесок на певну кількість діб. Узагальнена схема її роботи наведена на рисунку 3.3.

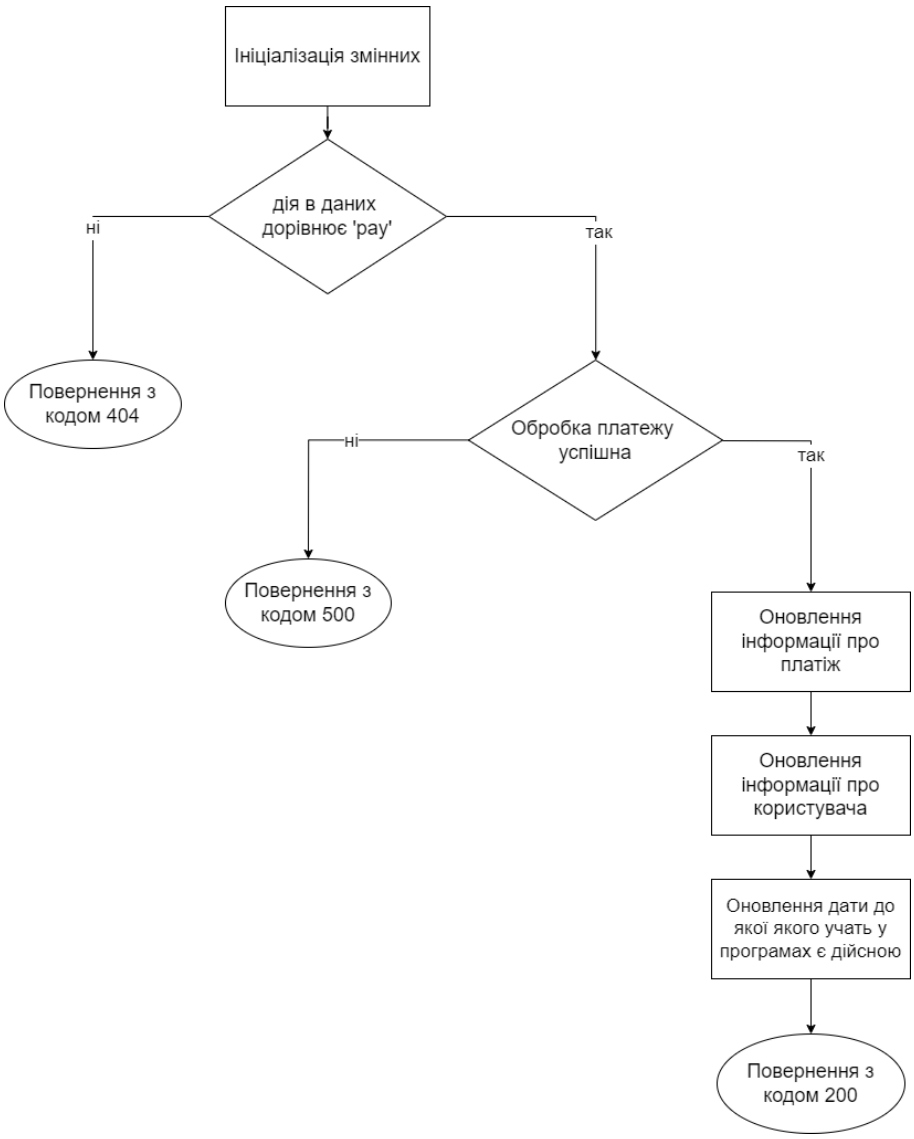

Рис. 3.3. Узагальнена схема роботи функції /user/payments/liqpay-callback

Програмний код цієї функції наведено у лістингу 3.5.

Лістинг 3.5. Програмний код функції /user/payments/liqpay-callback

```
@app.route("/user/payments/liqpay-callback", methods=['POST'])
def liqpay_callback():
    liqpay = LiqPay(app.config["LIQPAY_PUBLIC_KEY"],
app.config["LIQPAY_PRIVATE_KEY"])
    data = liqpay.decode_data_from_str(request.form["data"].encode())
    print(data)
    if data['action'] == 'pay':
        try:
            payment = Payments.get(Payments.payment_identifier ==
data['order_id'])
            x = str(datetime.now().timestamp()).index(".")
```

```
                    c = float(str(data["end_date"])[:x] + "." +
str(data["end_date"])[x:])
                    payment.payment_date = datetime.fromtimestamp(c)
                    payment.updated_at = datetime.now()
                    payment.save()
                    member = Members.get(id=payment.member)
                    program = MemberPrograms.get(MemberPrograms.member ==
payment.member, MemberPrograms.is_active == True)
                    days = 30 * payment.amount // program.program.month_price
                    if member.expiration_date is None or (member.expiration_date
is not None and datetime.now() > member.expiration_date):
                        member.expiration_date =  datetime.now() +
timedelta(days = days)
                    else:
                        member.expiration_date += timedelta(days = days)
                    member.save()
                    return jsonify({ 'status': 'success', 'message': 'Платіж
успішно підтверджено', 'data': data }), 200
                except Exception as e:
                    return jsonify({ 'status': 'error', 'message': 'Помилка' }),
500
            return jsonify({ 'status': 'error', 'message': 'Платіж не знайдено'
}), 404
```

3.2.3 Інтеграція сервісу “Дія”

В той момент, коли користувач вперше завантажив мобільний застосунок і починає роботу з інформаційною системою, однією з умов участі у благодійних програмах є надання персональних даних, включно зі світлинами документів, що ідентифікують особу. Ці документи можуть бути завантажені вручну, але завдяки державному сервісу “Дія”, який створений для обробки персональних даних та надання адміністративних послуг, є можливість автоматично завантажити інформацію потрібну для створення облікового запису учасника благодійних програм. Зокрема можуть бути завантажені файли, що містять світлини ідентифікаційних карт, кодів ІПН тощо. Запит до сервісу Дія може виконуватися винятково з мобільного пристрою, але інформація про запит має бути зашифрована закритим ключем

з пари клієнтських ключів/сертифікатів, що видаються державними центрами сертифікації ключів. Тому, задля підтримки сервісу Дія, серверна частина підтримує дві функції: перша, підписує запит на отримання персональних даних та документів учасників програм. Результати її роботи надаються мобільному застосунку, задля виклику функцій сервісу Дія. Узагальнена схема цього алгоритму наведена на рисунку 3.4.

Рис. 3.4. Узагальнена схема роботи функції /user/profile/diia

Програмний код цієї функції наведено у лістингу 3.6.

Лістинг 3.6. Програмний код функції /user/profile/diia

```
@app.route("/user/profile/diia", methods=["POST"])
@token_required
```

```
      def diia(**kwargs):
          current_user = kwargs["current_user"]
          session = prequests.Session()
          session.auth = (app.config["DIIA_USER"],
app.config["DIIA_PASSWORD"])
          response =
session.get(f"https://{app.config['DIIA_URL']}/api/v1/auth/acquirer/likcasa_t
est_token_24")
          print("test_token {0}".format(response.status_code))
          answer = json.loads(response.content.decode())
          print(answer)
          headers = {'accept': 'application/json', 'Content-Type':
'application/json', "Authorization": "Bearer "+answer["token"]}
          payload = json.dumps({"name": "Назва
послуги","returnLink":"https://test3.kozach.pp.ua/diia/", "scopes":
{"sharing": ["internal-passport", "taxpayer-card"]}})
          response =
prequests.post(f"https://{app.config['DIIA_URL']}/api/v1/acquirers/branch/412
fcb082dbd37c3f328851f3aeb302b72820a1893729763b25934feb0e1eb0f7c47b44c24c6787d
2086d92c30cb2826c358722f818e69f873b8897b61e6d858/offer", data=payload,
headers=headers)
          print("offer {0}".format(response.status_code))
          answer=response.json()
          print(answer)
          payload = json.dumps({"offerId": answer["_id"], "requestId":
datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")})
          response =
prequests.post(f"https://{app.config['DIIA_URL']}/api/v2/acquirers/branch/412
fcb082dbd37c3f328851f3aeb302b72820a1893729763b25934feb0e1eb0f7c47b44c24c6787d
2086d92c30cb2826c358722f818e69f873b8897b61e6d858/offer-request/dynamic",
data=payload, headers=headers)
          print("dynamic {0}".format(response.status_code))
          answer=response.json()
          print(answer)
          url = answer['deeplink']
          index = url.rfind("/")
          diia_uuid = url[index+1:]
          Members.update(diia_answer=diia_uuid).where(Members.id ==
current_user.id).execute()
          return jsonify(answer), 200
```

Друга функція є функцією зворотного виклику для сервісу Дія, яка приймає інформацію та файли у зашифрованому вигляді на сервер інформаційної системи благодійного фонду та записує отриману інформацію у відповідну базу даних та сховище файлів. Узагальнена схема цього алгоритму наведена на рисунку 3.5.

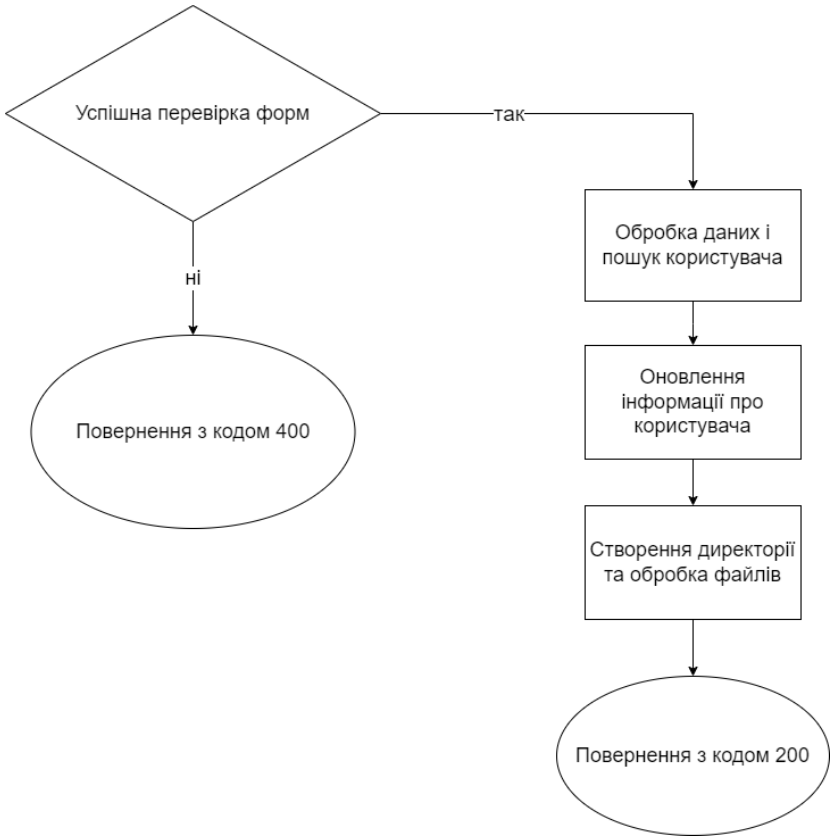

Рис. 3.5. Узагальнена схема роботи функції /user/profile/diia-callback

Програмний код цієї функції наведено у лістингу 3.7.

Лістинг 3.7. Програмний код функції /user/profile/diia-callback

```
@app.route("/user/profile/diia-callback", methods=["POST"])
def diia_callback():
    try:
        values = FormDataDiiaCallback(**request.form, **request.files)
    except pydantic.error_wrappers.ValidationError as e:
        return jsonify({ 'success': False }), 400
    d = datetime.utcnow()
    dl = d.astimezone(pytz.timezone('Europe/Kiev'))
```

```
          data = eu_develop(app.config["PRIVATE_KEY_FILE_PATH"],
app.config["PRIVATE_KEY_PASSWORD"], values.encodeData,
app.config["CERTFILE_PATH"])
          diia_uuid = re.findall(r'[A-Za-z0-9]{8}\-[A-Za-z0-9]{4}\-[A-Za-z0-
9]{4}\-[A-Za-z0-9]{4}\-[A-Za-z0-9]{12}', data[0].decode())
          diia_answer = json.loads(data[0].decode())
          member = Members.get(diia_answer=diia_uuid[0])
          member.diia_answer = data[0].decode()
          member.inn = diia_answer["data"]["internal-
passport"][0]["taxpayerNumber"]
          member.passport_data ="""ID-картка: {0}
                                Видана: {1}
                                Місце проживання: {2}""".format(
                                    diia_answer["data"]["internal-
passport"][0]["docNumber"],
                                    diia_answer["data"]["internal-
passport"][0]["issueDate"],
                                    diia_answer["data"]["internal-
passport"][0]["residenceUA"])
          member.last_name = diia_answer["data"]["internal-
passport"][0]["lastNameUA"]
          member.first_name = diia_answer["data"]["internal-
passport"][0]["firstNameUA"]
          member.middle_name = diia_answer["data"]["internal-
passport"][0]["middleNameUA"]
          member.birth_date =
datetime.strptime(diia_answer["data"]["internal-passport"][0]["birthday"],
"%d.%m.%Y")
          member.updated_at = datetime.now()
          member.save()
          print(member)
          userdir = os.path.join(app.config['UPLOAD_FOLDER'], str(member.id))
          if not os.path.exists(userdir):
              os.mkdir(userdir)
          for f in values.files:
              data = eu_develop(app.config["PRIVATE_KEY_FILE_PATH"],
app.config["PRIVATE_KEY_PASSWORD"], f.read(), app.config["CERTFILE_PATH"])
              if data:
                  i = f.filename.find(".pdf")
                  new_filename = f.filename[:i]+".pdf"
                  f1 = open(os.path.join(userdir, new_filename), "wb")
                  f1.write(data[0])
```

```
                    f1.close()
                    newfile = Files.create(created_at=datetime.now(),
filename=new_filename, member=member.id, updated_at=datetime.now())
                    if f.filename == diia_answer["data"]["internal-
passport"][0]["fileName"]:
                        it = IdentityTypes.get(name="ID-картка, аверс")
                        MemberIdentity.create(created_at=datetime.now(),
file=newfile.id, identity_type=it.id, is_new=True, member=member.id,
updated_at=datetime.now())
                    elif f.filename == diia_answer["data"]["taxpayer-
card"][0]["fileName"]:
                        it = IdentityTypes.get(name="Ідентифікаційний код")
                        MemberIdentity.create(created_at=datetime.now(),
file=newfile.id, identity_type=it.id, is_new=True, member=member.id,
updated_at=datetime.now())
          return jsonify({'success': True }), 200
```

3.3 Висновки до розділу 3

Було досліджено архітектуру застосунку, включаючи його відокремлені складові, зміст і принцип роботи. Також була проаналізована схема HTTP маршрутизації, та процедури відправки повідомлень, авторизації, та здійснення платежів. Описано структуру програмного компоненту, включаючи всі функції та класи, з яких він складається, з детальним описом кожного з них.

Також було розроблено опис використання цього проекту, включаючи його компоненти та запити. Було створено керівництво користувача, яке дозволяє користувачам ознайомитися з функціональністю серверного застосунку інформаційної системи «Каса взаємодопомоги» та отримати обізнаність щодо його використання та взаємодії з ним. Керівництво надає детальні вказівки щодо використання можливостей застосунку та розуміння технологій його роботи.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи "Розробка серверного застосунку інформаційної системи «Каса взаємодопомоги»" було успішно реалізовано ряд завдань та досягнуто мету роботи.

Застосунок написаний мовою Python з використанням фреймворку Flask для організації Web API застосунку, та фреймворку Peewee для організації доступу до бази даних. Серверна частина складається з двох застосунків, один з яких обслуговує запити від мобільних застосунків, якими користуються учасники благодійних програм. Друга частина обслуговує запити від адміністративного сайту системи, якою користуються учасники благодійного фонду.

Розроблене програмне забезпечення дозволяє організувати взаємодію із зовнішніми сервісами LiqPay для підтримки оплат учасниками своїх внесків, та сервісу Дія задля отримання персональних даних про учасників програм. Також використовується сервіс Twilio для відправки СМС повідомлень з одноразовими паролями для входу в систему.

Було проведено тестування функціоналу застосунку вбудованими засобами тестування Flask. Результати тестування були успішними, що підтверджує його стабільну та ефективну роботу.

Документація, включаючи опис архітектури та функціональності застосунку, була ретельно підготовлена. Це допоможе користувачам швидко ознайомитися з програмним компонентом, та ефективно використовувати його.

Загальним висновком є те, що розроблений серверний застосунок «Каси взаємодопомоги» відповідає поставленим вимогам та має потенціал для подальшого розвитку. У проєкті застосовані сучасні технології програмування і відповідні методи та шаблони побудови архітектури бекенд застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kropotkin P., Robinson V. Mutual Aid – A Factor of Evolution: With an Excerpt from Comrade Kropotkin by Victor Robinson. Read Books Limited, 2020.

2. Mutual aid [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Mutual_aid_(organization_theory)

3. Bertram C. Jean Jacques Rousseau, in Zalta, Edward N. (ed.), The Stanford Encyclopedia of Philosophy, 2020.

4. Katie H. From Mutual Aid To Dual Power: How Do We Build A New World In The Shell Of The Old? Plan C, 2020.

5. Turner F. J. Canadian encyclopedia of social work. Waterloo, Ont.: Wilfrid Laurier University Press, 2005. pp. 337–8.

6. Spade D. Solidarity Not Charity: Mutual Aid for Mobilization and Survival. Social Text, 2020. 38 (1). 131–151.

7. Shepard B. Community practice as social activism: from direct action to direct services. CA: Thousand Oaks, 2015. p. 166.

8. Zola. I. K. The problems and prospects of mutual aid groups. Rehabilitation Psychology, 1972. 19 (4): 180–183.

9. Izlar J. Radical social welfare and anti-authoritarian mutual aid. Critical and Radical Social Work. 2019. 7 (3), pp. 349–366.

10. Top 7 Mobile Backend as a Service (MBaaS) Platforms and tools. [Електронний ресурс] – Режим доступу: https://medium.com/@Apriorit/top-7-mobile-backend-as-a-service-mbaas-platforms-and-tools-ac008ac6504e

11. Firebase [Електронний ресурс] – Режим доступу: https://firebase.google.com/

12. AWS [Електронний ресурс] – Режим доступу: https://aws.amazon.com/amplify/?nc1=h_ls

13. Heroku [Електронний ресурс] – Режим доступу: https://www.heroku.com/

14. Backendless [Електронний ресурс] – Режим доступу: https://backendless.com/

15. Kinvey [Електронний ресурс] – Режим доступу: https://devcenter.kinvey.com/rest/guides/core-overview

16. Back4app [Електронний ресурс] – Режим доступу: https://www.back4app.com/

17. Kumulos [Електронний ресурс] – Режим доступу: https://www.kumulos.com/videos/mbaas-video/

18. Flask [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/3.0.x/

19. Peewee [Електронний ресурс] – Режим доступу: https://docs.peewee-orm.com/en/latest/

20. JSON Web Tokens. [Електронний ресурс] – Режим доступу: https://auth0.com/docs/secure/tokens/json-web-tokens

21. Szczepanik M. Backend for frontend (BFF) pattern— why do you need to know it? 2021. [Електронний ресурс] – Режим доступу: https://medium.com/mobilepeople/backend-for-frontend-pattern-why-you-need-to-know-it-46f94ce420b0

22. Kundrat D. Web App Security, Understanding the Meaning of the BFF Pattern. 2021. [Електронний ресурс] – Режим доступу: https://dev.to/damikun/web-app-security-understanding-the-meaning-of-the-bff-pattern-i85

23. Lutz M. Learning Python. O'Reilly Media, 2021. 1200 p.

24. Васильєв О. Програмування в PYTHON. Теорія і практика. Ліра-К, 2023.

25. Nichter D. Efficient MySQL Performance: Best Practices and Techniques. 1st Ed. O'Reilly Media, 2021. 275 p.

26. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення: ДСТУ 3008-95. – Чинний від 1996–01–01. – К.: Держстандарт України, 1996. – 39 с.

27. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги: ДСТУ 3582-97. – Чинний від 1998–07–01. – К.: Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

ДОДАТОК А. ЛІСТИНГ

Фрагмент admin-app/app.py

```
import os
from flask import Flask, request, jsonify
from flask_peewee.db import Database
import jwt
from functools import wraps
import os
from twilio.rest import Client
import json

app = Flask(__name__)
app.secret_key = "Pjr34vd1t6KqMfV51t6Vd"
app.config["SERVER_URL"] = "https://test4.kozach.pp.ua:15001"
app.config['UPLOAD_FOLDER'] = os.path.join(os.path.dirname(__file__), 'uploaddir')
app.config['MAX_CONTENT_LENGTH'] = 102400000
app.config['DAILY_LIMIT'] = 100
MYSQL_HOST = os.environ.get('MYSQL_HOST', 'localhost')
MYSQL_PORT = int(os.environ.get('MYSQL_PORT', 3306))
app.config['DATABASE'] = {
        'host': MYSQL_HOST,
        'port': MYSQL_PORT,
        'name': 'lkasa',
        'engine': 'peewee.MySQLDatabase',
        'user': 'adminkasa',
        'passwd': 'adminkasa'
    }
app.config["LIQPAY_PUBLIC_KEY"] = "sandbox_i99942863669"
app.config["LIQPAY_PRIVATE_KEY"] = "sandbox_8KnQcvQo4eG4jHARanzpboy2ECJYxJKro4R53Tug"
app.config["ALLOWED_EXTENSIONS"] = {'jpg', 'png', 'pdf', 'txt'}

app.config["DIIA_URL"] = "api2s.diia.gov.ua"
app.config["DIIA_USER"] = "acquirer_559"
app.config["DIIA_PASSWORD"] = "likcasa_test_token_24"
app.config["CONCORD_URL"] = "https://pay.concord.ua/api/"
app.config["CONCORD_MERCHANT_ID"] = "Q6hbx6m6N19vz4jpGVAbi_sIls-"
app.config["CONCORD_PRIVATE_KEY"] = "73Gpc4kt2oM02PG3268f9Yj3W31ci24Pm32rM06qThvvDuKG2xtobayXY68CR4J8ohcDNTi70z6934rpr25NbDpvVg6U075yxr0AN82x7qrv0n85p9kXgy6kg8roWq70"
app.config["PRIVATE_KEY_FILE_PATH"] = os.path.join(os.path.dirname(os.path.abspath(__file__)), "certificates/Key-6.dat")
app.config["PRIVATE_KEY_PASSWORD"] = "тщшыущге"
app.config["CERTFILE_PATH"] = "/data/certificates/Diia_2022.cer"
db = Database(app)
```

Фрагмент admin-app/functions.py

```
from flask import Flask, request, jsonify
from flask_peewee.db import Database
from random import randint
import jwt
import uuid
from functools import wraps
import os
from twilio.rest import Client
from models import *
from app import app, db
from datetime import *
import re
import base64

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        elif 'Authorization' in request.headers and request.headers['Authorization'].strip().split(" ")[0] == "Bearer":
            token = request.headers['Authorization'].strip().split(" ")[-1]
```

```
        if not token:
            return jsonify({'message' : 'Token is missing !!'}), 401
        try:
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
            current_user = Users.get(id = data['user_id'])
            kwargs.update({'current_user': current_user})
        except Exception as e:
            return jsonify({
                'message' : 'Token is invalid !!'
            }), 401
        return f( *args,**kwargs)

    return decorated

def generate_random_password(length=10):
    alphabet = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789_!@#$%^&*"
    password = ""
    for _ in range(length):
        password += alphabet[randint(0,len(alphabet)-1)]
    return password

def send_sms(phone, code):
    account_sid = "ACc180ae14934eccf4b7a75cb014df01d8"
    auth_token = "8715cc681170110246f0bf6862f9d29e"
    client = Client(account_sid, auth_token)

    message = client.messages \
        .create(
            body = str(code),
            from_ = '+13602275325',
            to = phone
        )


def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in app.config["ALLOWED_EXTENSIONS"]

def send_error_response(message, code):
    pass

def cleanhtml(raw_html):
    CLEANR = re.compile('<.*?>')
    cleantext = re.sub(CLEANR, '', raw_html)
    return cleantext

def get_date(date, format):
    d = None
    if type(date) == datetime:
        d = date
    elif type(date) == str:
        d = datetime.strptime(date, "%Y-%m-%d %H:%M:%S")
    return d.strftime(format)

def get_auth_code():
    alphabet = 'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789'
    code = ""
    for i in range(10):
        code += alphabet[randint(0, len(alphabet)-1)]
    return code

def change_filename(data, key):
    if hasattr(data, key) and getattr(data, key) is not None:
        i = getattr(data, key).filename.split(".")
        getattr(data, key).filename = str(uuid.uuid4()) + "." + i[-1]

def convert_date(date_str, formats):
    _date = None
    for format in formats:
        try:
            _date = datetime.strptime(date_str, format)
        except:
            _date = None
    return _date
```

Фрагмент admin-app/main.py

```
import os
import base64
import operator
from functools import reduce
from hashlib import sha256
from datetime import *
```

```
from random import randint
import urllib.request
from app import app, db
from functions import *
from time import sleep
from flask import Flask, request, redirect, jsonify, url_for, send_from_directory
from werkzeug.utils import secure_filename
import jwt, re, pytz, uuid
from models import *
from urllib.parse import urlparse
from playhouse.shortcuts import model_to_dict, dict_to_model
from flask_pydantic import validate
from validate import *
import pydantic
import json
from typing import Optional

@app.after_request
def add_cors_headers(response):
    response.headers["Access-Control-Allow-Origin"] = "*"
    response.headers["Access-Control-Allow-Methods"] ="GET, POST, PUT, DELETE, OPTIONS"
    response.headers["Access-Control-Allow-Headers"] = "Content-Type, Content-Disposition, Authorization"
    return response

@app.route("/files", methods=["POST"])
@token_required
@validate()
def downloader(body: FormGetFile, *args, **kwargs):
    current_user = kwargs["current_user"]
    print(body)
    if body.user_id:
        return send_from_directory(os.path.join(app.config["UPLOAD_FOLDER"], str(body.user_id)), body.filename, as_attachment=False)
    return jsonify({ 'status': 'error', 'message': 'Файл не знайдено' }), 404

@app.route("/program/<filename>", methods=["GET"])
def download_program(filename, **kwargs):
    if os.path.exists(os.path.join(app.config["UPLOAD_FOLDER"], filename)):
        return send_from_directory(app.config["UPLOAD_FOLDER"], filename, as_attachment=False)
    return jsonify({ 'status': 'error', 'message': 'Файл не знайдено' }), 404

@app.route("/signin", methods=['POST'])
@validate()
def signin(body: FormSignin):
    try:
        db_user = Users.get(login=body.login, password=sha256(body.password.encode()).hexdigest())
        token = jwt.encode({
            'user_id': db_user.id,
            'name': db_user.name,
            'email': db_user.email,
            'exp' : datetime.utcnow() + timedelta(minutes = 5) #43200)
        }, app.config['SECRET_KEY']) #, algorithms=['HS256'])
        return jsonify({ 'access_token': token, 'user': {'name': db_user.name, 'login': db_user.login, 'email': db_user.email, 'is_active': db_user.is_active, 'updated_at': get_date(db_user.updated_at, "%d.%m.%Y") }}), 200
    except:
        return jsonify({}), 404

@app.route("/user", defaults={'user_id': None}, methods=['GET'])
@app.route("/user/<user_id>", methods=['GET'])
@token_required
def user_get(user_id, **kwargs):
    current_user = kwargs["current_user"]
    if current_user.login != 'admin':
        return jsonify({'status': 'error', 'message': 'Ця операція дозволена тільки головному адміністратору'}), 403
    if user_id:
        users = Users.select().where(Users.id == int(user_id)).execute()
    else:
        users = Users.select().execute()
    return jsonify({'data': [model_to_dict(m) for m in users]}), 200
```

```
@app.route("/user", methods=['PUT'])
@token_required
@validate()
def user_create(body: FormUserCreate, **kwargs):
    current_user = kwargs["current_user"]
    try:
        user = Users.select().where(Users.login == body.login).get()
        return jsonify({'status': 'error', 'message': 'user already exist'}), 422
    except:
        pass
    try:
        pwd = generate_random_password()
        pwd_hash = sha256(pwd.encode()).hexdigest()
        Users.create(login=body.login, password=pwd_hash, email=body.email, phone=body.phone, is_active=body.is_active, name=body.name, created_at=datetime.now(), updated_at=datetime.now())
        return jsonify({'status': 'success', 'message': 'user created successfully'}), 200
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@app.route("/user", methods=['POST'])
@token_required
@validate()
def user_update(body: FormUserUpdate, **kwargs):
    current_user = kwargs["current_user"]
    user = None
    try:
        user = Users.get(id=body.id)
    except Users.DoesNotExist:
        return jsonify({'status': 'error', 'message': 'user does not exist'}), 422
    try:
        if body.email:
            user.email = body.email
        if body.phone:
            user.phone = body.phone
        if body.is_active:
            user.is_active = body.is_active
        if body.name:
            user.name = body.name
        user.updated_at = datetime.now()
        user.save()
        return jsonify({'status': 'success', 'message': 'user updated successfully', 'data': model_to_dict(user)}), 200
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@app.route("/user/<id>", methods=['DELETE'])
@token_required
def user_delete(id, **kwargs):
    current_user = kwargs["current_user"]
    if id:
        try:
            user = Users.select().where(Users.id == id).get()
            user.delete_instance()
            return jsonify({'status': 'success', 'message': 'User deleted successfully'}), 200
        except Exception as e:
            return jsonify({'status': 'error', 'message': str(e)}), 500
    return jsonify({'status': 'error', 'message': 'No user to delete'}), 400

@app.route("/user/change_password", methods=['POST'])
@validate()
@token_required
def change_password(body: FormDataUserPasswordChange, **kwargs):
    current_user = kwargs["current_user"]
    db_user = None
    user_id = None
    try:
        db_user = Users.get(id=current_user.id)
    except:
        return jsonify({ 'status': 'error', 'meaage': 'Помилковий користувач' }), 422
    if body.id:
        if db_user.login != 'admin' and body.id != current_user.id:
            return jsonify({ 'status': 'error', 'message': 'Вам ця операція не дозволена'}), 403
        user_id = body.id
    else:
        user_id = current_user.id
    try:
        pwd = sha256(body.old_password.encode()).hexdigest()
```

```
        user = Users.select().where(Users.id==user_id, Users.password==pwd).get()
        user.password = sha256(body.new_password.encode()).hexdigest()
        user.save()
        return jsonify({'status': 'success', 'message': 'Пароль змінено'}), 200
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@app.route("/members", methods=['POST'])
@token_required
@validate()
def member_list(body: FilterForm, **kwargs):
    current_user = kwargs["current_user"]
    if body.filter["query"] == "":
        members = Members.select().where((Members.is_submited == False) | (Members.has_new_program_change == True) | (Members.has_new_passport_data == True) | (Members.has_new_request > 0)).execute()
    else:
        conditions = []
        for e in body.filter["query"].split():
            if body.filter["fields"]["passport"]:
                conditions.append((fn.LOWER(Members.passport_data).contains(e)))
            if body.filter["fields"]["name"]:
                conditions.append((fn.LOWER(Members.first_name).contains(e)))
                conditions.append((fn.LOWER(Members.middle_name).contains(e)))
                conditions.append((fn.LOWER(Members.last_name).contains(e)))
            if body.filter["fields"]["inn"]:
                conditions.append((fn.LOWER(Members.inn).contains(e)))
            if body.filter["fields"]["iban"]:
                conditions.append((fn.LOWER(Members.actual_iban).contains(e)))
            if body.filter["fields"]["certificate"]:
                conditions.append((fn.LOWER(Members.certificate).contains(e)))
        where_expression = reduce(operator.or_, conditions)
        members = Members.select().where(where_expression).execute()
    return jsonify({'data': [model_to_dict(m) for m in members]}), 200

@app.route("/member/generate_certificate", methods=["GET"])
@token_required
def generate_certificate():
    certificate = -1
    while app.config["certificate"] >= certificate:
        timestamp = datetime.now().timestamp()
        certificate = int(str(timestamp).split(".")[0])
    return jsonify({ 'data': str(certificate) }), 200

@app.route("/member", methods=['POST'])
@token_required
@validate()
def member_get(body: ModelGet,  **kwargs):
    current_user = kwargs["current_user"]
    try:
        member = Members.get(id=body.id)
        programs = MemberPrograms.select().join(Programs).where(MemberPrograms.member == member.id).order_by(MemberPrograms.updated_at).execute()
        requests = [model_to_dict(r) for r in Requests.select().where(Requests.member == body.id).order_by(Requests.updated_at).execute()]
        print(requests)
        payments = Payments.select().join(PaymentTypes, on=(Payments.payment_type == PaymentTypes.id)).join(Requests, JOIN.LEFT_OUTER, on=(Requests.id == Payments.request)).where(Payments.member == member.id & (PaymentTypes.name == "Поповнення" | PaymentTypes.name == "Відшкодування" | PaymentTypes.name == "Часткове відшкодування")).order_by(Payments.payment_date).execute()
        idata = []
```

```
        actual_data = []
        new_data = []
        identities = MemberIdentity.select().join(IdentityTypes).join(Files, on=(MemberIdentity.file == Files.id)).where(MemberIdentity.member == member.id).order_by(MemberIdentity.identity_type).order_by(MemberIdentity.updated_at).execute()
        for m in identities:
            md = model_to_dict(m)
            md["is_active"] = True if m.is_active == 1 else False
            md["is_new"] = True if m.is_new == 1 else False
            idata.append(md)
            if m.is_new:
                new_data.append(md)
            if m.is_active:
                actual_data.append(md)
        mdict = model_to_dict(member)
        try:
            mdict["birth_date"] = member.birth_date.strftime("%Y-%m-%d")
        except:
            mdict["birth_date"] = None
        mdict["is_active"] = True if member.is_active == 1 else False
        pdata = []
        actual_program = None
        new_program = None
        for p in programs:
            mp = model_to_dict(p)
            mp["is_active"] = True if p.is_active == 1 else False
            mp["is_new"] = True if p.is_new == 1 else False
            if p.is_active:
                actual_program = mp
            if p.is_new:
                new_program = mp
        return jsonify({'member': mdict, 'actual_program': actual_program, 'new_program': new_program, 'programs': pdata, 'current_identities': actual_data, 'new_identities': new_data, 'identities': idata, "requests": requests, 'payments': [model_to_dict(p) for p in payments]}), 200
    except Exception as e:
        return jsonify({ 'status': 'error', 'message': 'Учасника не знайдено' }), 404

@app.route("/member", methods=['PUT'])
@token_required
@validate()
def member_update(body: FormMemberUpdate, **kwargs):
    try:
        member= Members.get(id=body.id)
        if body.first_name:
            member.first_name = body.first_name
        if body.middle_name:
            member.middle_name = body.middle_name
        if body.last_name:
            member.last_name = body.last_name
        if body.email:
            member.email = body.email
        if body.iban:
            member.iban = body.iban
        if body.birth_date:
            try:
                member.birth_date = datetime.strptime(body.birth_date, "%d.%m.%Y")
            except:
                member.birth_date = datetime.strptime(body.birth_date, "%Y-%m-%d")
        if body.certificate:
            member.certificate = body.certificate
        if body.passport_data:
            member.passport_data = body.passport_data
        if body.inn:
            member.inn = body.inn
        if body.phone:
            member.phone = body.phone
        if body.is_active:
            member.is_active = body.is_active
        if body.is_submited:
            member.is_submited = body.is_submited
        member.updated_at = datetime.now()
        if member.has_new_program_change:

            MemberPrograms.update(is_active=False).where(MemberPrograms.member == member.id).execute()
            MemberPrograms.update(is_active=True, is_new=False).where(MemberPrograms.member == member.id, MemberPrograms.is_new == True).execute()
            member.has_new_program_change = False
        if member.has_new_passport_data:
```

```
MemberIdentity.update(is_active=False).where(MemberIdentity.member == member.id, MemberIdentity.is_active == True).execute()
        MemberIdentity.update(is_active=True, is_new=False).where(MemberIdentity.member == member.id, MemberIdentity.is_new == True).execute()
        member.has_new_passport_data = False
      member.save()
      return jsonify({ 'status': 'success', 'message': 'Інформація про учасника успішно оновлена' }), 200
   except Exception as e:
      return jsonify({ 'status': 'error', 'message': 'Помилка оновлення інформації про учаника', 'data': str(e) }), 500



@app.route("/programs", methods=["GET"])
@token_required
def programs_get(**kwargs):
   programs = Programs.select().where(Programs.is_active == True).execute()
   return jsonify({ 'status': 'success', 'data': [model_to_dict(m) for m in programs] }), 200

@app.route("/program", methods=["POST"])
@token_required
@validate()
def program_get(body: ModelGet, **kwargs):
   current_user = kwargs["current_user"]
   try:
      program = Programs.get(id=body.id)
      return jsonify({ 'status': 'success', 'data': model_to_dict(program) }), 200
   except Exception as e:
      return jsonify({ 'status': 'error', 'message': 'Помилка отримання інформації про програми', 'data': str(e)}), 500

@app.route("/program",  methods=["PUT"])
@token_required
def program_create_update(**kwargs):
   current_user = kwargs["current_user"]
   try:
      body = ProgramData(**request.form, **request.files)
   except Exception as e:
      return jsonify({ 'status': 'error', 'message': 'Помилка перевірки правильності інформації про програму', 'data': str(e) }), 400
   try:
      if body.new_program_file:
         change_filename(body, "new_program_file")

body.new_program_file.save(os.path.join(app.config['UPLOAD_FOLDER'], body.new_program_file.filename))
      if body.new_program_file:
         filename = "{0}/program/{1}".format(app.config["SERVER_URL"], body.new_program_file.filename)
      if body.id:
         program = Programs.get(id=body.id)
         if body.month_price:
            program.month_price=body.month_price
         if body.program_name:
            program.program_name=body.program_name
         if body.new_program_file:
            program.program_file=filename
         if body.is_active:
            program.is_active=body.is_active
         program.updated_by=current_user.id
         program.updated_at=datetime.now()
         program.save()
         return jsonify({ 'status': 'success', 'message': 'Програму успішно оновлено', 'data': model_to_dict(program) }), 200
      else:
         program = Programs.create(created_at=datetime.now(), created_by=current_user.id, month_price=body.month_price, program_file=filename, program_name=body.program_name, updated_at=datetime.now(), updated_by=current_user.id, is_active=body.is_active)
         return jsonify({ 'status': 'success', 'message': 'Програма успішно створена', 'data': model_to_dict(program) }), 200
   except Exception as e:
      return jsonify({ 'status': 'eeror', 'message': 'Помилка створення/оновлення програми', 'data': str(e)}), 500


@app.route("/payment",  methods=["PUT"])
```

```
@token_required
@validate()
def payment_create(body: PaymentCreate, **kwargs):
    current_user = kwargs["current_user"]
    try:
        if hasattr(body, "request"):
            payment = Payments.create(amount=body.amount, created_at=datetime.now(), member=body.member, payment_date=body.date, payment_type=body.type, updated_at=datetime.now(), request=body.request)
        else:
            payment = Payments.create(amount=body.amount, created_at=datetime.now(), member=body.member, payment_date=body.date, payment_type=body.type, updated_at=datetime.now())
        return jsonify({ 'status': 'success', 'message': 'Інформацію про платіж додано', 'data': model_to_dict(payment) }), 200
    except Exception as e:
        return jsonify({ 'status': 'error', 'message': str(e)}), 500

@app.route("/payments", methods=["POST"])
@token_required
@validate()
def payments_get(body: PaymentsList, **kwargs):
    current_user = kwargs["current_user"]
    query = Payments.select().join(PaymentTypes).join(Members, on=(Members.id == Payments.member)).join(Requests, on=(Requests.id == Payments.request))
    conditions = []
    if hasattr(body, "certificate") and body.certificate is not None:

        conditions.append((Payments.member.certificate.contains(body.certificate)))
    if hasattr(body, "type") and body.type is not None and len(body.type) > 0:

        conditions.append((Payments.payment_type.id.in_(body.type)))
    if hasattr(body, "date_start") and hasattr(body, "date_end") and body.date_start is not None and body.date_end is not None and body.date_start != "" and body.date_end != None:
        date1 = convert_date(body.date_start, ["%d.%m.%Y", "%Y-%m-%d"])
        date2 = convert_date(body.date_start, ["%d.%m.%Y", "%Y-%m-%d"])

        conditions.append((Payments.updated_at.between(date1, date2)))
    elif hasattr(body, "date_start") and body.date_start is not None and body.date_start != "":
        date1 = convert_date(body.date_start, ["%d.%m.%Y", "%Y-%m-%d"])
        conditions.append((Payments.updated_at > date1))
    elif hasattr(body, "date_end") and body.date_end is not None and body.date_end != "":
        date2 = convert_date(body.date_start, ["%d.%m.%Y", "%Y-%m-%d"])
        conditions.append((Payments.updated_at < date2))
    if hasattr(body, "amount_start") and hasattr(body, "amount_end") and body.amount_start is not None and body.amount_end is not None:

        conditions.append((Payments.amount.between(body.amount_start, body.amount_end)))
    elif hasattr(body, "amount_start") and body.amount_start is not None:
        conditions.append((Payments.amount > body.amount_start))
    elif hasattr(body, "amount_end") and body.amount_end is not None:
        conditions.append((Payments.amount < body.amount_end))
    if len(conditions) > 0:
        where_expression = reduce(operator.or_, conditions)
        query = query.where(where_expression)
    query = query.order_by(Payments.updated_at.desc()).offset(body.offset).limit(body.limit)
    payments = query.execute()
    return jsonify({ 'status': "success", "data": [model_to_dict(p) for p in payments]}), 200

@app.route("/requests", methods=['POST'])
@token_required
@validate()
def requests_get(body: RequestList, **kwargs):
    current_user = kwargs["current_user"]
```

```
    requests = []
    try:
        sids = [m.id for m in RequestStatus.select().where(RequestStatus.alias.in_(body.status)).execute() ]
        conditions = [ (Requests.status.id.in_(sids)) ]
        if hasattr(body, 'date_start') and body.date_start and hasattr(body, 'date_end') and body.date_end:
            print("Step 1")
            date_start = datetime.strptime(body.date_start, "%d.%m.%Y")
            date_end = datetime.strptime(body.date_end, "%d.%m.%Y")
            conditions += [ (Requests.updated_at.between(date_start, date_end)) ]
        elif hasattr(body, 'date_start') and body.date_start:
            print("Step 2")
            date_start = datetime.strptime(body.date_start, "%d.%m.%Y")
            conditions += [ (Requests.updated_at > date_start) ]
        elif hasattr(body, 'date_end') and body.date_end:
            print("Step 3")
            date_end = datetime.strptime(body.date_end, "%d.%m.%Y")
            conditions += [ (Requests.updated_at < date_end) ]
        if hasattr(body, 'query') and body.query != "":
            conditions += [ ((Requests.member.last_name.contains(body.query)) | (Requests.member.first_name.contains(body.query)) | (Requests.member.middle_name.contains(body.query)) | (Requests.member.inn.contains(body.query)) | (Requests.member.certificate.contains(body.query))) ]

        where_expression = reduce(operator.or_, conditions)
        for m in Requests.select().join(RequestStatus).join(Members, on=(Requests.member == Members.id)).where(where_expression).execute():
#Order_by(Requests.updated_at).order_by(Requests.id).offset(body.offset).limit(body.limit).execute():
            mdict = model_to_dict(m)
            mdict["created_at"] = get_date(mdict["created_at"], "%d.%m.%Y")
            mdict["updated_at"] = get_date(mdict["updated_at"], "%d.%m.%Y")
            requests.append(mdict)
        return jsonify({ 'data': requests }), 200
    except Exception as e:
        return jsonify({}), 500


@app.route("/request", methods=["PUT"])
@token_required
@validate()
def request_update(body: RequestUpdate, **kwargs):
    current_user = kwargs["current_user"]
    try:
        _request = Requests.update(diagnos=body.diagnos, suma=body.suma, updated_at=datetime.now()).where(Requests.id == body.id).execute()
        return jsonify({ 'status': 'success', 'message': 'Запит успішно оновлено' }), 200
    except Exception as e:
        return jsonify({ 'status': 'error', 'message': str(e) }), 500



@app.route("/request/message", methods=["POST"])
@token_required
@validate()
def request_message_new(body:NewMessage, **kwargs):
    current_user = kwargs["current_user"]
    try:
        _request = Requests.get(id=body.request_id)
        message = RequestMessages.create(created_atdatetime.now(), user=current_user.id, message=body.message, request=_request.id, updated_at=datetime.now())
        status = RequestStatus.get(alias="awaiting_client")
        _request.status = status
        _request.save()
        return jsonify({ 'status': 'success', 'message': 'Повідомлення додано' }), 200
    except Requests.DoesNotExist as e:
        return jsonify({ 'status': 'error', 'message': 'Запит не знайдено' }), 404
    except Exception as e:
        return jsonify({ 'status': 'error', 'message': str(e) }), 500

@app.route("/request", methods=["POST"])
@token_required
@validate()
def request_get(body: RequestGet,  **kwargs):
```

```
    current_user = kwargs["current_user"]
    print(body)
    try:
        _request = model_to_dict(Requests.get(id=body.request))
        _request["created_at"] = get_date(_request["created_at"], "%d.%m.%y")
        _request["updated_at"] = get_date(_request["updated_at"], "%d.%m.%Y")
        _request["messages"] = [model_to_dict(m) for m in RequestMessages.select().where(Requests.id == _request.id).order_by(updated_at).execute()]
        request_files = Files.select().where(Files.request == body.request).execute()
        files = [ model_to_dict(m) for m in request_files]
        _request["files"] = files
        types = [model_to_dict(t) for t in PaymentTypes.select().where(PaymentTypes.name.in_(['Виплата', 'Часткова виплата'])).execute()]
        tids = [m["id"] for m in types]
        charges = [model_to_dict(c) for c in Payments.select().where((Payments.payment_type.in_(tids)) & (Payments.member == body.member) & (Payments.request == body.request)).execute()]
        _request["charges"] = charges
        _request["types"] = types
        return jsonify( { 'data': _request }), 200
    except pydantic.error_wrappers.ValidationError as e:
        return jsonify({'status': 'error', 'message': str(e) }), 400
    except IndexError as e:
        return jsonify({ 'status': 'error', 'message': 'Вказаний запит не знайдено'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message':  str(e) }), 500

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=15001)
```

Фрагмент admin-app/validate.py

```
from pydantic import Field, BaseModel, EmailStr, validator, root_validator
from typing import Optional, List, Dict
from datetime import *
import re
from functions import allowed_file
from werkzeug.datastructures import FileStorage

class FormDataUserPasswordChange(BaseModel):
    id: Optional[int]
    old_password: str
    new_password: str

class PaymentCreate(BaseModel):
    member: int
    type: int
    request: Optional[int]
    date: str
    amount: float

class PaymentsList(BaseModel):
    type: Optional[List[int]]
    date_start: Optional[str]
    date_end: Optional[str]
    amount_start: Optional[float]
    amount_end: Optional[float]
    certificate: Optional[str]
    offset: int = 0
    limit: int = 50

class NewMessage(BaseModel):
    request_id: int
    message: str

class FormGetFile(BaseModel):
    user_id: Optional[int]
    filename: str

class FilterForm(BaseModel):
    filter: Dict

class FormSignin(BaseModel):
    login: str
    password: str

class FormUserCreate(BaseModel):
    email: EmailStr
    is_active: bool
    login: str
    name: str
    phone: Optional[str]
```

```
class FormUserUpdate(BaseModel):
    id: int
    email: Optional[EmailStr]
    is_active: Optional[bool]
    login: Optional[str]
    name: Optional[str]
    phone: Optional[str]

class NewSignRequest(BaseModel):
    phone: str
    create: Optional[bool]

    @validator("phone")
    def correct_phone(cls, p):
        regex = re.compile("\+\d{12}")
        r = regex.match(p)
        if not r:
            raise ValueError("Phone is invalid")
        return p

class RequestList(BaseModel):
    query: Optional[str]
    date_start: Optional[str]
    date_end: Optional[str]
    status: List[str] = ['opened']
    limit: int = 50
    offset: int = 0

    @validator("status")
    def correct_status(cls, p):
        status_set = {'opened', 'closed', 'awaiting_client',
'awaiting_agent'}
        for s in p:
            if s not in status_set:
                raise ValueError(f"Unknow status '{s}'")
        return p

class ChargeListRequest(BaseModel):
    begin: str
    end: str

    @validator("begin")
    def correct_begin(cls, p):
        d = datetime.strptime(p, "%d.%m.%Y")
        return p

    @validator("end")
    def correct_end(cls, p):
        d = datetime.strptime(p, "%d.%m.%Y")
        return p

class NewSignin(BaseModel):
    phone: str
    code: str

    @validator("phone")
    def correct_phone(cls, p):
        regex = re.compile("\+\d{12}")
        r = regex.match(p)
        if not r:
            raise ValueError("Phone is invalid")
        return p

class ModelGet(BaseModel):
    id: int

class RequestGet(BaseModel):
    request: int
    member: Optional[int]

class FormMemberUpdate(BaseModel):
    id: int
    first_name: Optional[str]
    middle_name: Optional[str]
    last_name: Optional[str]
    email: Optional[EmailStr]
    iban: Optional[str]
    birth_date: Optional[str]
    certificate: Optional[str]
    passport_data: Optional[str]
    inn:  Optional[str]
    phone: Optional[str]
    is_active: Optional[bool]
    is_submited: Optional[bool]


class FormDataProfileEdit(BaseModel):
    email: EmailStr
    phone: str
    photo: Optional[FileStorage]

    @validator("phone")
```

```
    def correct_phone(cls, p):
        regex = re.compile("\+\d{12}")
        r = regex.match(p)
        if not r:
            raise ValueError("Phone is invalid")
        return p

    @validator("photo")
    def correct_photo(cls, p):
        if not allowed_file(p.filename):
            raise ValueError("Photo file name is empty or this
file type is not allowed")
        return p

    class Config:
        arbitrary_types_allowed = True


class FormDataWrongProfileData(BaseModel):
    first_name: str
    middle_name: str
    last_name: str
    birth_date: str

class ProgramData(BaseModel):
    id: Optional[int]
    month_price: Optional[float]
    program_file: Optional[str]
    new_program_file: Optional[FileStorage]
    program_name: Optional[str]
    is_active: Optional[bool]

    @validator("program_file")
    def correct_program_file(cls, p):

        return p

    class Config:
        arbitrary_types_allowed = True

class FormDataRequestCreate(BaseModel):
    comment: str
    files: Optional[List]  #[FileStorage]]

    @root_validator(pre=True)
    def make_files(cls, values):
        result = { 'files': [] }
        for k in values:
            if k != 'comment':
                result["files"].append(values[k])
            else:
                result[k] = values[k]
        return result

    class Config:
        arbitrary_types_allowed = True


class RequestUpdate(BaseModel):
    id: int
    request_text: str
    diagnos: str
    suma: float
    member: int

    class Config:
        arbitrary_types_allowed = True


class FormDataDiiaCallback(BaseModel):
    encodeData: str
    files: Optional[List]

    @root_validator(pre=True)
    def make_files(cls, values):
        result = { 'files': [] }
        for k in values:
            if k != 'encodeData':
                result["files"].append(values[k])
            else:
                try:
                    result[k] = values[k]
                except:
                    raise ValueError("Wrong id type")
        return result

    class Config:
        arbitrary_types_allowed = True
```

Фрагмент kasa-app/app.py

```
import os
```

```
from flask import Flask, request, jsonify
from flask_peewee.db import Database
import jwt
from functools import wraps
import os
from twilio.rest import Client

app = Flask(__name__)
app.secret_key = "Pjr34vd1t6KqMfV51t6Vd"
app.config["SERVER_URL"] ="https://test3.kozach.pp.ua"
app.config['UPLOAD_FOLDER'] =
os.path.join(os.path.dirname(__file__), 'uploaddir')
app.config['MAX_CONTENT_LENGTH'] = 102400000
app.config['DAILY_LIMIT'] = 100
MYSQL_HOST = os.environ.get('MYSQL_HOST',
'localhost')
MYSQL_PORT = int(os.environ.get('MYSQL_PORT',
3306))
app.config['DATABASE'] = {
        'host': MYSQL_HOST,
        'port': MYSQL_PORT,
        'name': 'lkasa',
        'engine': 'peewee.MySQLDatabase',
        'user': 'adminkasa',
        'passwd': 'adminkasa'
    }
app.config["LIQPAY_PUBLIC_KEY"] =
"sandbox_i99942863669"
app.config["LIQPAY_PRIVATE_KEY"] =
"sandbox_8KnQcvQo4eG4jHARanzpboy2ECJYxJKro4R5
3Tug"
app.config["ALLOWED_EXTENSIONS"] = {'jpg', 'png',
'pdf', 'txt'}

app.config["DIIA_URL"] = "api2s.diia.gov.ua"
app.config["DIIA_USER"] = "acquirer_559"
app.config["DIIA_PASSWORD"] =
"likcasa_test_token_24"
app.config["PRIVATE_KEY_FILE_PATH"] =
"/data/certificates/Key-6.dat"
app.config["PRIVATE_KEY_PASSWORD"] =
"тщшыущге"
app.config["CERTFILE_PATH"] =
"/data/certificates/Diia_2022.cer"
db = Database(app)
```

Фрагмент kasa-app/main.py

```
import os
import base64
from datetime import *
from random import randint
import urllib.request
from app import app, db
from time import sleep
from functions import *
from flask import Flask, request, redirect, jsonify, url_for,
send_from_directory
from werkzeug.utils import secure_filename
import jwt, re, pytz, uuid
from models import *
from playhouse.shortcuts import model_to_dict
from urllib.parse import urlparse
from liqpay3 import LiqPay
from playhouse.shortcuts import model_to_dict,
dict_to_model
from flask_pydantic import validate
from validate import *
import pydantic
import re
import json
import requests as prequests
import hmac
import hashlib
from EUSignCP import *
from typing import Optional

def eu_develop(privKeyPath, privKeyPassword,
customerCrypto, senderCertPath):
    try:
        EULoad()
        pIface = EUGetInterface()
        pIface.Initialize()
        if not pIface.IsPrivateKeyReaded():
            pInfo = {}
            pIface.ReadPrivateKeyFile(privKeyPath,
privKeyPassword, pInfo)
        with open(senderCertPath, "br") as f:
            senderCert = f.read()
        customerCrypto = base64.b64decode(customerCrypto)
        ppbData = []
        pSenderInfo = {}
```

```
        pIface.DevelopDataEx(None, customerCrypto, len(customerCrypto),
            senderCert, len(senderCert), ppbData, pSenderInfo)
        signedData = ppbData[0]
        pSignInfo = {}
        pIface.VerifyDataInternal(None, signedData, len(signedData),
            ppbData, pSignInfo)
        return ppbData
    except Exception as e:
        print(e)
        return None

@app.route('/files/<uid>/<filename>')
@token_required
def downloader(uid, filename, *args, **kwargs):
    return send_from_directory(os.path.join(app.config["UPLOAD_FOLDER"], uid), filename, as_attachment=False)

@app.route('/.well-known/apple-app-site-association')
def get_apple_json():
    return jsonify({ "applinks": {
        "details": [{
            "appIDs": [ "5JTBMP57KF.ua.org.lKassa" ],
            "components": [{"/": "/diia/*" }]
        }]
        }
    }), 200

@app.route('/user/signrequest', methods=['POST'])
@validate()
def sign_request_post(body: NewSignRequest):

    db_sign_request = None
    db_user = None
    try:
        db_user = Members.get(phone=body.phone)
        if body.create:
            return jsonify({'message': 'User already exist'}), 422
    except Members.DoesNotExist:
        if not body.create:
            return jsonify({'message': 'User does not exist'}), 422
    code = 1111
    try:
        db_sign_request = SignRequest.get(phone=body.phone)
        if db_sign_request and db_sign_request.daily_counter >= app.config["DAILY_LIMIT"] and (datetime.now() - db_sign_request.sent_at).days < 1:
            return jsonify({ 'message': 'Access denied' }), 401

        elif db_sign_request and db_sign_request.daily_counter >= app.config["DAILY_LIMIT"] and (datetime.now() - db_sign_request.sent_at).days >= 1:
            print("Variant 3")
            db_sign_request.delete_instance()
            raise Exception("sign request deleted")
        db_sign_request.code = code
        db_sign_request.sent_at = datetime.now()
        db_sign_request.to_create = body.create
        db_sign_request.daily_counter += 1
        db_sign_request.save()
    except Exception as e:
        new_sign_request = SignRequest.create(phone=body.phone, code=code, sent_at=datetime.now(), to_create=body.create, daily_counter=1)
    return jsonify({ 'message': 'Message was sent.' }), 200

@app.route('/user/signin', methods=['POST'])
@validate()
def signin_post(body: NewSignin):

    db_sign_request = None
    db_user = None
    user_exception = None
    usermeta_exception = None
    try:
        db_sign_request = SignRequest.get(phone=body.phone, code=body.code)
        if ((datetime.now() - db_sign_request.sent_at).seconds // 60) > 3:
            raise SignRequest.DoesNotExist("code exceded")
        db_user = Members.get(phone=body.phone)
    except SignRequest.DoesNotExist:
        return jsonify({ 'message': 'Access denied for this user' }), 401
    except Members.DoesNotExist:
        user_exception = True
```

```
    if user_exception is not None:
        if db_sign_request and not db_sign_request.to_create:
            return jsonify({ 'message': 'User does not exist' }),
404
        else:
            try:
                db_user = Members.create(phone=body.phone,
created_at=datetime.now(), updated_at=datetime.now())
            except Exception as e:
                print(e)
                return jsonify({ 'message': 'Can not create user.' }),
500
    token = jwt.encode({
        'user_id': db_user.id,
        'name': "{0} {1}".format(db_user.first_name,
db_user.last_name),
        'email': db_user.email,
        'exp' : datetime.utcnow() + timedelta(minutes = 43200)
    }, app.config['SECRET_KEY']) #, algorithms=['HS256'])
    return jsonify({ 'access_token': token}), 200

@app.route('/user/profile/create', methods=['POST'])
@token_required
def user_profile_post(**kwargs):

    current_user = kwargs["current_user"]
    try:
        values = FormDataProfileCreate(**request.form,
**request.files)
    except pydantic.error_wrappers.ValidationError as e:
        msg = str(e).split("\n")
        return jsonify({ "message": msg[-1].strip() }), 400
    change_filename(values, "inn")
    change_filename(values, "id_1")
    change_filename(values, "id_2")
    change_filename(values, "book_1")
    change_filename(values, "book_2")
    change_filename(values, "book_address")
    try:
        innfile = Files.create(filename=values.inn.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
        if values.book_1 and values.book_2 and
values.book_address:
            book1file =
Files.create(filename=values.book_1.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
            book2file =
Files.create(filename=values.book_2.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
            bookaddress =
Files.create(filename=values.book_address.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
        if values.id_1 and values.id_2:
            id1file =
Files.create(filename=values.id_1.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
            id2file =
Files.create(filename=values.id_2.filename,
created_at=datetime.now(), updated_at=datetime.now(),
member_id=current_user.id)
        userdir =
os.path.join(app.config['UPLOAD_FOLDER'],
str(current_user.id))
        if not os.path.exists(userdir):
            os.mkdir(userdir)
        if values.inn:
            values.inn.save(os.path.join(userdir,
values.inn.filename))
            MemberIdentity.create(file=innfile.id,
identity_type=6, created_at=datetime.now(),
updated_at=datetime.now(), member=current_user.id,
is_new=True)
        if values.id_1:
            values.id_1.save(os.path.join(userdir,
values.id_1.filename))
            MemberIdentity.create(file=id1file.id,
identity_type=1, created_at=datetime.now(),
updated_at=datetime.now(), member=current_user.id,
is_new=True)
        if values.id_2:
            values.id_2.save(os.path.join(userdir,
values.id_2.filename))
            MemberIdentity.create(file=id2file.id,
identity_type=2, created_at=datetime.now(),
updated_at=datetime.now(), member=current_user.id,
is_new=True)
        if values.book_1:
```

```
            values.book_1.save(os.path.join(userdir,
values.book_1.filename))
            MemberIdentity.create(file=book1file.id,
identity_type=3,                created_at=datetime.now(),
updated_at=datetime.now(),          member=current_user.id,
is_new=True)
        if values.book_2:
            values.book_2.save(os.path.join(userdir,
values.book_2.filename))
            MemberIdentity.create(file=book2file.id,
identity_type=4,                created_at=datetime.now(),
updated_at=datetime.now(),          member=current_user.id,
is_new=True)
        if values.book_address:
            values.book_address.save(os.path.join(userdir,
values.book_address.filename))
            MemberIdentity.create(file=bookaddress.id,
identity_type=5,                created_at=datetime.now(),
updated_at=datetime.now(),          member=current_user.id,
is_new=True)
        user = Members.update(first_name=values.first_name,
middle_name=values.middle_name,
last_name=values.last_name,          email=values.email,
actual_iban=values.iban,                is_active=False,
is_submited=False,          has_new_passport_data=True,
has_new_program_change=True,
updated_at=datetime.now()).where(Members.id==current_
user.id).execute()
        MemberPrograms.create(member=current_user.id,
program=values.program_id,    created_at=datetime.now(),
updated_at=datetime.now(), is_new=True)
        return jsonify({'message': 'profile successfuly
created'}), 200
    except Exception as e:
        print(e)
        return jsonify({'message': 'Profile creation error'}), 500

@app.route('/user/profile/edit', methods=['POST'])
@token_required
def user_update(**kwargs):

    current_user = kwargs["current_user"]
    try:
        values      =      FormDataProfileEdit(**request.form,
**request.files)
    except Exception as e:
        msg = str(e).split("\n")
        return jsonify({ "message": msg[-1].strip() }), 400

    try:
        db_user = Members.get(phone=values.phone)
        if db_user.id != current_user.id:
            return jsonify({'message': 'Can not assign this phone
number'}), 422
        if values.photo:
            change_filename(values, "photo")
            userdir                                          =
os.path.join(app.config['UPLOAD_FOLDER'],
str(current_user.id))
            if not os.path.exists(userdir):
                os.mkdir(userdir)
            values.photo.save(os.path.join(userdir,
values.photo.filename))
            Members.update(phone=values.phone,
email=values.email,          updated_at=datetime.now(),
photo=values.photo.filename).where(id                ==
current_user.id).execute()
        else:
            Members.update(phone=values.phone,
email=values.email,   updated_at=datetime.now()).where(id
== current_user.id).execute()
        return      jsonify({'message':      'Profile      successfuly
updated'}), 200
    except Exception as e:
        return jsonify({'message': 'Profile update error'}), 500

@app.route("/user/profile/wrong", methods=['POST'])
@token_required
def profile_wrong(**kwargs):

    current_user = kwargs["current_user"]
    try:
        values = FormDataWrongProfileData(**request.form,
**request.files)
    except Exception as e:
        msg = str(e).split("\n")
        return jsonify({ "message": msg[-1].strip() }), 400
    try:

ParticipantRequest.create(first_name=values.first_name,
middle_name=values.middle_name,
last_name=values.last_name,  birth_date=values.birth_date,
```

```
user_id=current_user.id, created_at=datetime.now(), request_type=2)
        return jsonify({'message': 'Request successfuly sent'}), 200
    except:
        return jsonify({'message': 'Request failed'}), 500

@app.route('/user/profile/get')
@token_required
def user_get(*args, **kwargs):

    current_user = kwargs["current_user"]
    parts = urlparse(request.base_url)
    host = parts.scheme + "://" + parts.netloc
    profile = {}
    try:
        db_user = Members.get(id=current_user.id)
        if not db_user.is_submited:
            return jsonify({'message': 'Your profile is not approved yet'}), 403
        profile = model_to_dict(db_user)
        profile["birth_date"] = get_date(profile["birth_date"], "%d.%m.%Y") if profile["birth_date"] else None
        if profile["photo"] is not None and profile["photo"] != "":
            profile["photo"] = host + "/files/{0}/{1}".format(str(current_user.id), profile["photo"])
        else:
            profile["photo"] = ""
        profile["expiration_date"] = get_date(profile["expiration_date"], "%d.%m.%Y") if profile["expiration_date"] else None
        m = MemberPrograms.select().join(Programs).where(MemberPrograms.member == current_user.id, MemberPrograms.is_active == True).get()
        profile["program"] = {'program_id': m.program.id, 'program_name': m.program.program_name, 'program_file': m.program.program_file, 'month_price': m.program.month_price }
        return jsonify(profile), 200
    except Exception as e:
        print(e)
        return jsonify({'message': 'Error reading profile'}), 500

@app.route('/program/list')
def program():
    programs = Programs.select().where(Programs.is_active == 1).execute()
    result = [{'program_id': p.id, 'program_name': p.program_name, 'program_file': p.program_file, 'month_price': p.month_price } for p in programs]
    return jsonify(result), 200

@app.route("/program/change", methods=['POST'])
@token_required
def program_change(**kwargs):

    current_user = kwargs["current_user"]
    try:
        values = FormDataProgramChange(**request.form, **request.files)
        try:
            mp = MemberPrograms.select().where(MemberPrograms.member == current_user.id, MemberPrograms.is_new == True).get()
            mp.delete_instance()
        except:
            pass
        pm = MemberPrograms.create(created_at=datetime.now(), member=current_user.id, program=values.program_id, is_active=False, is_new=True, updated_at=datetime.now())
        return jsonify({'status': 'status', 'message': 'Request for program chage successfuly sent', 'data': model_to_dict(pm) }), 200
    except pydantic.error_wrappers.ValidationError as e:
        msg = str(e).split("\n")
        return jsonify({ 'status': 'error', 'message': msg[-1].strip() }), 400
    except:
        return jsonify({'status': 'error', 'message': 'Request failed'}), 500

@app.route("/user/charges/list", methods=['POST'])
@token_required
def payments_get(**kwargs):
    current_user = kwargs["current_user"]
    results = []
    body = request.json
```

```
    if body is None:
        body = {}
    pts = []
    for p in PaymentTypes.select().where(PaymentTypes.name.in_(["Відшкодування", "Часткове відшкодування"])).execute():
        pts.append(p.id)
    try:
        body = ChargeListRequest(**body)
        begin = datetime.strptime(body.begin, "%d.%m.%Y")
        end = datetime.strptime(body.end, "%d.%m.%Y")
        vutratu = Payments.select().where(Payments.member == current_user.id, Payments.payment_type.in_(pts), Payments.date.between(begin, end)).execute()
    except:
        vutratu = Payments.select().where(Payments.member == current_user.id, Payments.payment_type.in_(pts)).execute()
    for pm in vutratu:
        results.append({
            "date": get_date(pm.payment_date, "%d.%m.%Y"),
            "sum": str(pm.suma),
            "diagnosis": pm.request.diagnos,
            "request_id": pm.request.id,
            "type": pm.payment_type.name
        })
    return jsonify(results), 200

@app.route("/user/payments/list")
@token_required
def charges_get(**kwargs):

    current_user = kwargs["current_user"]
    results = []
    pts = []
    for p in PaymentTypes.select().where(PaymentTypes.name.in_(["Поповнення"])).execute():
        pts.append(p.id)
    payments = Payments.select().where(Payments.member == current_user.id, Payments.payment_type.in_(pts)).execute()
    for pm in payments:
        results.append({
            "date": get_date(pm.payment_date, "%d.%m.%Y"),
            "sum": str(pm.suma),
            "payed_to": None
        })
    return jsonify(results), 200

@app.route("/user/request/list", methods=['POST'])
@token_required
def request_list(**kwargs):
    current_user = kwargs["current_user"]
    body = request.json
    try:
        body = RequestListRequest(**body)
    except:
        body = None
    requests = []
    try:
        if body and body.status:
            rs_ids = [rs.id for rs in RequestStatus.select().where(RequestStatus.alias.in_(body.status)).execute()]
            print("!!! ", rs_ids)
            requests = [{ "id": r.id, "status": r.status.alias, "comment": r.request_text, "date": get_date(r.created_at, "%d.%m.%Y"), "date_modified": get_date(r.updated_at, "%d.%m.%Y"), "sum": r.suma, "diagnos": r.diagnos if r.diagnos else "", "files": [] } for r in Requests.select().where(Requests.member == current_user.id, Requests.status.in_(rs_ids)).execute()]
        else:
            print("$$$$")
            requests = [{ "id": r.id, "status": r.status.alias, "comment": r.request_text, "date": get_date(r.created_at, "%d.%m.%Y"), "date_modified": get_date(r.updated_at, "%d.%m.%Y"), "sum": r.suma, "diagnos": r.diagnos, "files": [] } for r in Requests.select().where(Requests.member == current_user.id).execute()]
        print(requests)
        r_ids = [r["id"] for r in requests]
        requests_files = Files.select().where(Files.request.in_(r_ids)).execute()
        for rf in requests_files:
            _request = next(r for r in requests if r["id"] == rf.request.id)
            print(_request)
            m = model_to_dict(rf)
```

```
            m["url"] = "{0}/files/{1}/{2}".format(app.config["SERVER_URL"], current_user.id, rf.filename)
            _request["files"].append(m)
        return jsonify(requests), 200
    except Exception as e:
        print(e)
        return jsonify({}), 500


@app.route("/user/request/<id>")
@token_required
def request_get(id, **kwargs):
    current_user = kwargs["current_user"]
    try:
        print("Step 1")
        request_files = Files.select().join(Requests).where(Files.member == current_user.id, Files.request == id).execute()
        files = []
        for r in request_files:
            print("Step 2")
            m = model_to_dict(r)
            m["url"] = "{0}/files/{1}/{2}".format(app.config["SERVER_URL"],current_user.id, r.filename)
            files.append(m)
            print("Step 3")
        print(files)
        _request = files[0]["request"]
        messages = [model_to_dict(m) for m in RequestMessages.select().join(Members, JOIN.LEFT_OUTER, on=(RequestMessages.member == Members.id)).join(Requests, on=(RequestMessages.request == Requests.id)).join(Users, JOIN.LEFT_OUTER, on=(RequestMessages.user == Users.id)).where(Requests.id == _request["id"]).order_by(RequestMessages.updated_at).execute()]
        print(messages)
        request_dict = {
            "id": _request["id"],
            "status": _request["status"]["alias"],
            "comment": _request["request_text"],
            "date": get_date(_request["created_at"], "%d.%m.%Y"),
            "date_modified": get_date(_request["updated_at"], "%d.%m.%Y"),
            "sum": _request["suma"],
            "diagnos": _request["diagnos"] if _request["diagnos"] else "",
            "files": files,
            "messages": messages
        }
        return jsonify(request_dict), 200
    except pydantic.error_wrappers.ValidationError as e:
        print(e)
        return jsonify({'status': 'error', 'message': str(e) }), 400
    except IndexError as e:
        print(e)
        jsonify({ 'status': 'error', 'message': 'Вказаний запит не знайдено'}), 404
    except Exception as e:
        print(e)
        return jsonify({'status': 'error', 'message':  str(e) }), 500


@app.route("/user/request/<id>/message", methods=['POST'])
@token_required
@validate()
def request_message_new(id, body:NewMessage, **kwargs):
    current_user = kwargs["current_user"]
    try:
        _request = Requests.get(id=id)
        message = RequestMessages.create(created_at=datetime.now(), member=current_user.id, message=body.message, request=_request.id, updated_at=datetime.now())
        status = RequestStatus.get(alias="awaiting_agent")
        _request.status = status
        _request.save()
        return jsonify({ 'status': 'success', 'message': 'Повідомлення додано' }), 200
    except Requests.DoesNotExist as e:
        print(e)
        return jsonify({ 'status': 'error', 'message': 'Запит не знайдено' }), 404
    except Exception as e:
        print(e)
        return jsonify({ 'status': 'error', 'message': str(e) }), 500
```

```
@app.route("/user/request/create", methods=['POST'])
@token_required
def request_create(**kwargs):

    current_user = kwargs["current_user"]
    d = datetime.utcnow()
    dl = d.astimezone(pytz.timezone('Europe/Kiev'))
    try:
        values = FormDataRequestCreate(**request.form, **request.files)
        opened = RequestStatus.get(alias="opened")
        _request = Requests.create(created_at=datetime.now(), member=current_user.id, request_text=values.comment, status=opened.id, suma=0, updated_at=datetime.now())
        if hasattr(values, 'files'):
            for f in values.files:
                try:
                    userdir = os.path.join(app.config['UPLOAD_FOLDER'], str(current_user.id))
                    if not os.path.exists(userdir):
                        os.mkdir(userdir)
                    saved_name = "{0}_{1}".format(str(int(dl.timestamp())), f.filename)
                    f.save(os.path.join(userdir, saved_name))
                    Files.create(created_at=datetime.now(), filename=saved_name, member=current_user.id, request=_request.id, updated_at=datetime.now())
                except Exception as e:
                    pass
        return request_get(_request.id, **kwargs)
    except pydantic.error_wrappers.ValidationError as e:
        return jsonify ({'error': 'Comment is absent' }), 400
    except Exception as e:
        print(e)
        return jsonify({'error': 'Error while creating request'}), 500

@app.route("/user/request/update", methods=['POST'])
@token_required
def request_update(**kwargs):

    current_user = kwargs["current_user"]
    d = datetime.utcnow()
    dl = d.astimezone(pytz.timezone('Europe/Kiev'))
    try:
        values = FormDataRequestUpdate(**request.form, **request.files)
        _request = Requests.get(id=values.id)
        if hasattr(values, 'files'):
            for f in values.files:
                try:
                    userdir = os.path.join(app.config['UPLOAD_FOLDER'], str(current_user.id))
                    if not os.path.exists(userdir):
                        os.mkdir(userdir)

                    saved_name = "{0}_{1}".format(str(int(dl.timestamp())), f.filename)
                    f.save(os.path.join(userdir, saved_name))
                    Files.create(created_at=datetime.now(), filename=saved_name, member=current_user.id, request=_request.id, updated_at=datetime.now())
                except Exception as e:
                    pass
        return request_get(_request.id, **kwargs)
    except pydantic.error_wrappers.ValidationError as e:
        print(e)
        return jsonify({'error': 'Request id not valid' }), 400
    except Exception as e:
        print(e)
        return jsonify({'error': 'Error while creating request'}), 500

@app.route("/user/payments\/liqpay", methods=['POST'])
@token_required
def liqpay(**kwargs):
    current_user = kwargs["current_user"]
    liqpay = LiqPay(app.config["LIQPAY_PUBLIC_KEY"], app.config["LIQPAY_PRIVATE_KEY"])
    stamp = datetime.timestamp(datetime.now())
    order_id = str(stamp).replace(".", "") + str(current_user.id)

    print(request.url_root)

    params = {
        'action': 'pay',
        'amount': request.json["amount"],
        'currency': 'UAH',
        'description': 'Payment for order',
```

```
        'order_id': order_id,
        'version': '3',
        'sandbox': 1,
        'server_url': app.config["SERVER_URL"] + '/user/payments/liqpay-callback',
    }
    print(params)
    signature = liqpay.cnb_signature(params)
    data = liqpay.cnb_data(params)
    pid = PaymentTypes.get(name='Поповнення')
    Payments.create(payment_identifier=order_id, amount=float(request.json["amount"]), member=current_user.id, payment_type=pid, created_at=datetime.now(), updated_at=datetime.now())
    return jsonify({ 'signature': signature, 'data': data }), 200

@app.route("/user/payments/liqpay-callback", methods=['POST'])
def liqpay_callback():
    liqpay = LiqPay(app.config["LIQPAY_PUBLIC_KEY"], app.config["LIQPAY_PRIVATE_KEY"])
    data = liqpay.decode_data_from_str(request.form["data"].encode())
    print(data)
    if data['action'] == 'pay':
        Payments.update(payment_date=data[""], updated_at=datetime.now()).where(Payments.payment_identifier == data['order_id']).execute()
        return jsonify({ 'status': 'success', 'message': 'Платіж успішно підтверджено', 'data': data }), 200
    return jsonify({ 'status': 'error', 'message': 'Платіж не знайдено' }), 404

@app.route("/user/profile/diia", methods=["POST"])
@token_required
def diia(**kwargs):
    current_user = kwargs["current_user"]
    session = prequests.Session()
    session.auth = (app.config["DIIA_USER"], app.config["DIIA_PASSWORD"])
    response = session.get(f"https://{app.config['DIIA_URL']}/api/v1/auth/acquirer/likcasa_test_token_24")
    print("test_token {0}".format(response.status_code))
    answer = json.loads(response.content.decode())
    print(answer)
    headers = {'accept': 'application/json', 'Content-Type': 'application/json', "Authorization": "Bearer "+answer["token"]}
    payload = json.dumps({"name": "Назва послуги","returnLink":"https://test3.kozach.pp.ua/diia/", "scopes": {"sharing": ["internal-passport", "taxpayer-card"]}})
    response = prequests.post(f"https://{app.config['DIIA_URL']}/api/v1/acquirers/branch/412fcb082dbd37c3f328851f3aeb302b72820a1893729763b25934feb0e1eb0f7c47b44c24c6787d2086d92c30cb2826c358722f818e69f873b8897b61e6d858/offer", data=payload, headers=headers)
    print("offer {0}".format(response.status_code))
    answer=response.json()
    print(answer)
    payload = json.dumps({"offerId": answer["_id"], "requestId": datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")})
    response = prequests.post(f"https://{app.config['DIIA_URL']}/api/v2/acquirers/branch/412fcb082dbd37c3f328851f3aeb302b72820a1893729763b25934feb0e1eb0f7c47b44c24c6787d2086d92c30cb2826c358722f818e69f873b8897b61e6d858/offer-request/dynamic", data=payload, headers=headers)
    print("dynamic {0}".format(response.status_code))
    answer=response.json()
    print(answer)
    url = answer['deeplink']
    index = url.rfind("/")
    diia_uuid = url[index+1:]

Members.update(diia_answer=diia_uuid).where(Members.id == current_user.id).execute()
    return jsonify(answer), 200

@app.route("/user/profile/diia-callback", methods=["POST"])
def diia_callback():
    try:
        values = FormDataDiiaCallback(**request.form, **request.files)
    except pydantic.error_wrappers.ValidationError as e:
        return jsonify({ 'success': False }), 400
    d = datetime.utcnow()
```

```
    dl = d.astimezone(pytz.timezone('Europe/Kiev'))
    data = eu_develop(app.config["PRIVATE_KEY_FILE_PATH"], app.config["PRIVATE_KEY_PASSWORD"], values.encodeData, app.config["CERTFILE_PATH"])
    diia_uuid = re.findall(r'[A-Za-z0-9]{8}\-[A-Za-z0-9]{4}\-[A-Za-z0-9]{4}\-[A-Za-z0-9]{4}\-[A-Za-z0-9]{12}', data[0].decode())
    diia_answer = json.loads(data[0].decode())
    print("Step 0")
    member = Members.get(diia_answer=diia_uuid[0])
    print("Step 1")
    member.diia_answer = data[0].decode()
    print("Step 2")
    member.inn = diia_answer["data"]["internal-passport"][0]["taxpayerNumber"]
    print("Step 3")
    member.passport_data ="""ID-картка: {0}
                Видана: {1}
                Місце проживання: {2}""".format(
                    diia_answer["data"]["internal-passport"][0]["docNumber"],
                    diia_answer["data"]["internal-passport"][0]["issueDate"],
                    diia_answer["data"]["internal-passport"][0]["residenceUA"])
    print("Step 4")
    member.last_name = diia_answer["data"]["internal-passport"][0]["lastNameUA"]
    print("Step 5")
    member.first_name = diia_answer["data"]["internal-passport"][0]["firstNameUA"]
    print("Step 6")
    member.middle_name = diia_answer["data"]["internal-passport"][0]["middleNameUA"]
    print("Step 7")
    member.birth_date = datetime.strptime(diia_answer["data"]["internal-passport"][0]["birthday"], "%d.%m.%Y")
    print("Step 8")
    member.updated_at = datetime.now()
    print("Step 9")
    member.save()
    print(member)
    print("Step 10")
    userdir = os.path.join(app.config['UPLOAD_FOLDER'], str(member.id))
    if not os.path.exists(userdir):
        os.mkdir(userdir)
    for f in values.files:
        data = eu_develop(app.config["PRIVATE_KEY_FILE_PATH"], app.config["PRIVATE_KEY_PASSWORD"], f.read(), app.config["CERTFILE_PATH"])
        if data:
            i = f.filename.find(".pdf")
            new_filename = f.filename[:i]+".pdf"
            f1 = open(os.path.join(userdir, new_filename), "wb")
            f1.write(data[0])
            f1.close()
            newfile = Files.create(created_at=datetime.now(), filename=new_filename, member=member.id, updated_at=datetime.now())
            if f.filename == diia_answer["data"]["internal-passport"][0]["fileName"]:
                it = IdentityTypes.get(name="ID-картка, аверс")

MemberIdentity.create(created_at=datetime.now(), file=newfile.id, identity_type=it.id, is_new=True, member=member.id, updated_at=datetime.now())
            elif f.filename == diia_answer["data"]["taxpayer-card"][0]["fileName"]:
                it = IdentityTypes.get(name="Ідентифікаційний код")

MemberIdentity.create(created_at=datetime.now(), file=newfile.id, identity_type=it.id, is_new=True, member=member.id, updated_at=datetime.now())
    return jsonify({'success': True }), 200

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=15000)
```