Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для підтримки волонтерської діяльності. Серверна частина
(тема)

Виконав:
здобувач 4 року навчання,
групи ПЗПІ-21-4

Вадим СОРОКІН
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник проф. кафедри ПІ Володимир БОНДАРЄВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Сорокіну Вадиму Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для підтримки волонтерської діяльності. Серверна частина

Затверджена наказом по університету №397Ст від 19.05.2025

2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2025

3. Вихідні дані до роботи Розробити програмну систему для підтримки волонтерської діяльності, а саме такий елемент системи: Серверна частина.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2025	*виконано*
2	Створення специфікації ПЗ	16.04.2025	*виконано*
3	Проектування ПЗ	24.04.2025	*виконано*
4	Розробка ПЗ	23.05.2025	*виконано*
5	Тестування ПЗ	31.05.2025	*виконано*
6	Оформлення пояснювальної записки	01.06.2025	*виконано*
7	Підготовка презентації та доповіді	01.06.2025	*виконано*
8	Попередній захист	04.06.2025	*виконано*
9	Нормоконтроль, рецензування	08.06.2025	*виконано*
10	Здача роботи у електронний архів	12.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	12.06.2025	*виконано*

Дата видачі завдання 09 квітня 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. кафедри ПІ Володимир БОНДАРЄВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 50 стор., 22 рис., 9 джерел.

DOCKER, FAST API, JWT, POSTGRESQL, PYTHON, REST API, ВОЛОНТЕРСТВО, ЗБОРИ

Об'єкт розробки – серверна частина (back-end) програмної системи для автоматизації процесів збору пожертв, обліку закупівель та звітності в межах волонтерської ініціативи. Ця частина включає інтеграцію з базою даних, реалізацію бізнес-логіки для обробки запитів, забезпечення захисту даних та взаємодії з клієнтською частиною.

Мета розробки – створити надійну та ефективну серверну платформу, яка дозволяє автоматизувати основні процеси обліку фінансових надходжень, управління закупівлями, веденням звітів, а також забезпечить безпечну обробку даних і зручну інтеграцію з фронтенд-застосунками або іншими сервісами.

У результаті реалізовано серверну частину, що підтримує ключові функції системи для обліку пожертв, ведення закупівель, звітності та взаємодії з користувачами через HTTP API.

ABSTRACT

DOCKER, FAST API, JWT, POSTGRESQL, PYTHON, REST API, VOLUNTEERING, FEES

The object of development is the back-end of a software system for automating the processes of collecting donations, accounting for purchases and reporting within a volunteer initiative. This part includes integration with the database, implementation of business logic for processing requests, ensuring data protection and interaction with the client side.

The goal of the development is to create a reliable and efficient server platform that automates the main processes of accounting for financial receipts, procurement management, and reporting, as well as provides secure data processing and convenient integration with front-end applications or other services.

The result of the development is a modular, scalable, and secure server part that provides all the necessary functions for accounting for donations, procurement, reporting, and user interaction via HTTP API.

ЗМІСТ

ВСТУП

Сучасне суспільство постійно стикається з викликами ефективної організації взаємодії між різними учасниками соціальних ініціатив. Зростаюча роль волонтерських і благодійних проєктів вимагає впровадження цифрових рішень, які сприяють координації учасників, обліку ресурсів та формуванню звітності забезпечення прозорості процесів. У цьому контексті постає необхідність у створенні програмних систем, здатних автоматизувати ключові аспекти діяльності таких ініціатив.

Одним з основних аспектів успішної роботи волонтерських проєктів є ефективне управління фінансовими потоками, закупівлями та звітністю. Однак наявні системи фінансового обліку часто не відповідають специфіці волонтерських ініціатив, зокрема в умовах України. Вони можуть бути занадто складними, дорогими або не адаптованими до локальних потреб.

Актуальність даної роботи полягає в розробці програмної системи для автоматизації збору коштів, обліку витрат і закупівель, а також формування звітності для волонтерських організацій. Метою цієї роботи є створення надійної та ефективної серверної частини системи, яка дозволяє організувати процеси зборів, контролю фінансових транзакцій, управління закупівлями та забезпечує прозорість для донорів і волонтерів. Розробка такої системи дозволить оптимізувати процеси управління, забезпечити контрольоване управління фондами.

Для досягнення поставленої мети визначені наступні завдання: аналіз існуючих рішень, розробка архітектури бекенд-частини з використанням сучасних технологій, створення бази даних для зберігання інформації, забезпечення безпеки даних, автоматизація формування звітів та інтеграція з зовнішніми сервісами (банківськими системами, платіжними платформами, логістичними сервісами). У результаті створення такої системи з’явиться можливість значно зменшити адміністративні витрати, покращити управління фондами та закупівлями і, в кінцевому підсумку, підвищити ефективність волонтерської роботи.

Галузь застосування результатів роботи охоплює не лише волонтерські ініціативи в Україні, а й потенційно інші гуманітарні місії та донорські програми. Завдяки такій системі, організації зможуть покращити управління своїми фондами, забезпечити високий рівень прозорості та підзвітності і значно підвищити ефективність своєї роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної галузі

У відповідь на масштабні суспільні виклики останніх років волонтерський рух зазнав безпрецедентного зростання. Тисячі ініціатив, як локальних, так і міжнародних, спрямували свої зусилля на підтримку Збройних Сил України, евакуацію мирного населення, медичну допомогу, гуманітарне забезпечення та інші критичні потреби. У таких умовах особливої важливості набуває прозора, ефективна та захищена система для управління зборами коштів і звітністю щодо закупівель.

Одна з ключових проблем — недовіра до нових волонтерських ініціатив. Через відсутність або недостатню публічну звітність потенційні донори часто вагаються підтримувати проєкти, навіть якщо вони щиро допомагають. Це знижує ефективність зборів та потенційну кількість пожертв.

Серед проблем також можна назвати децентралізованість та неорганізованість через що більшу увагу зазвичай отримують медійні персони, та навпаки, безпосередньо військові або відомі у вузьких колах волонтери часто можуть не закривати збори. Окрім цього, у більшості випадків волонтери ведуть облік вручну (через Google Таблиці, Excel, нотатки, соцмережі), що ускладнює облік коштів, планування закупівель, звітність перед донорами та проведення аудит фінансових операцій що також впливає на довіру до волонтера та дозволяє знайти можливість для спекуляцій як власне волонтерам, так і їх конкурентам або опонентам, що намагаються підірвати їхню діяльність.

Як приклад системи що вже існує можна навести Volonter by Prom [1]. Проєкт створений для об'єднання українських фондів та волонтерів, яким не вистачає коштів, та всіх небайдужих дати можливість допомогти їм. Вони надають функціонал для збору коштів та подальшої звітності.

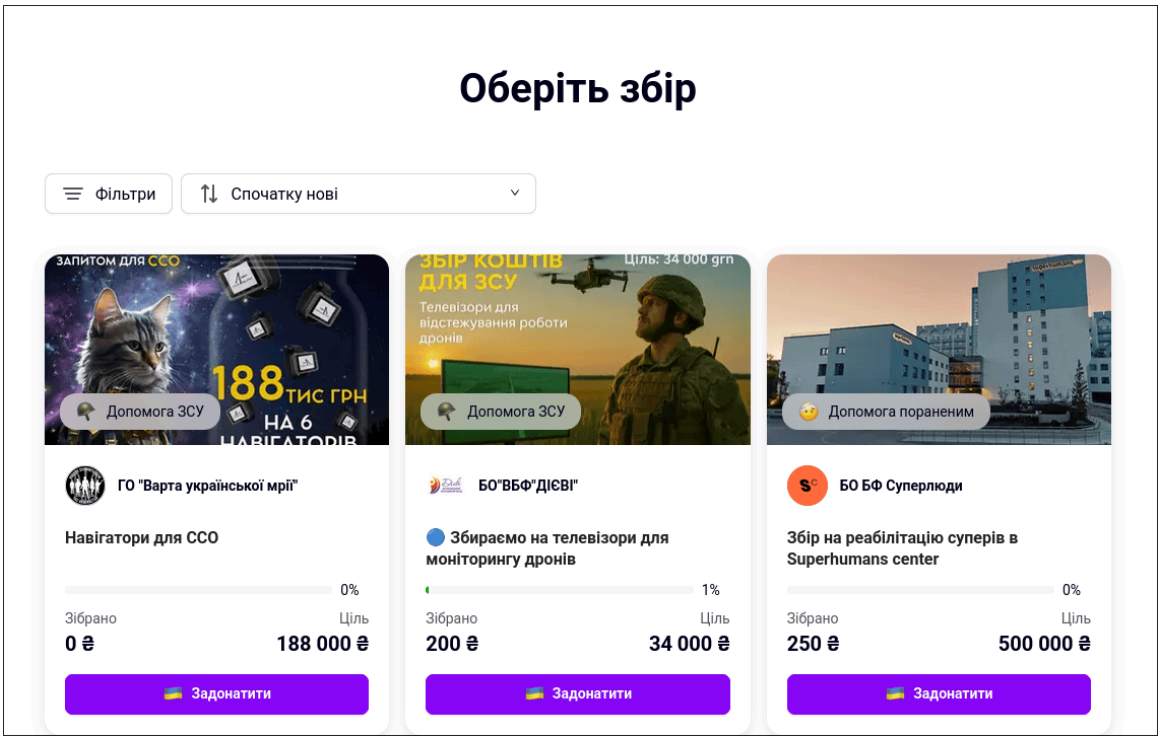

Рисунок 1.1 – Екран всіх зборів[1]

На екрані всіх зборів ми можемо почати список в зручному форматі плиток які надають основну інформацію про кожен збір та можливість задонатити на нього що зображено на рисунку 1.1.

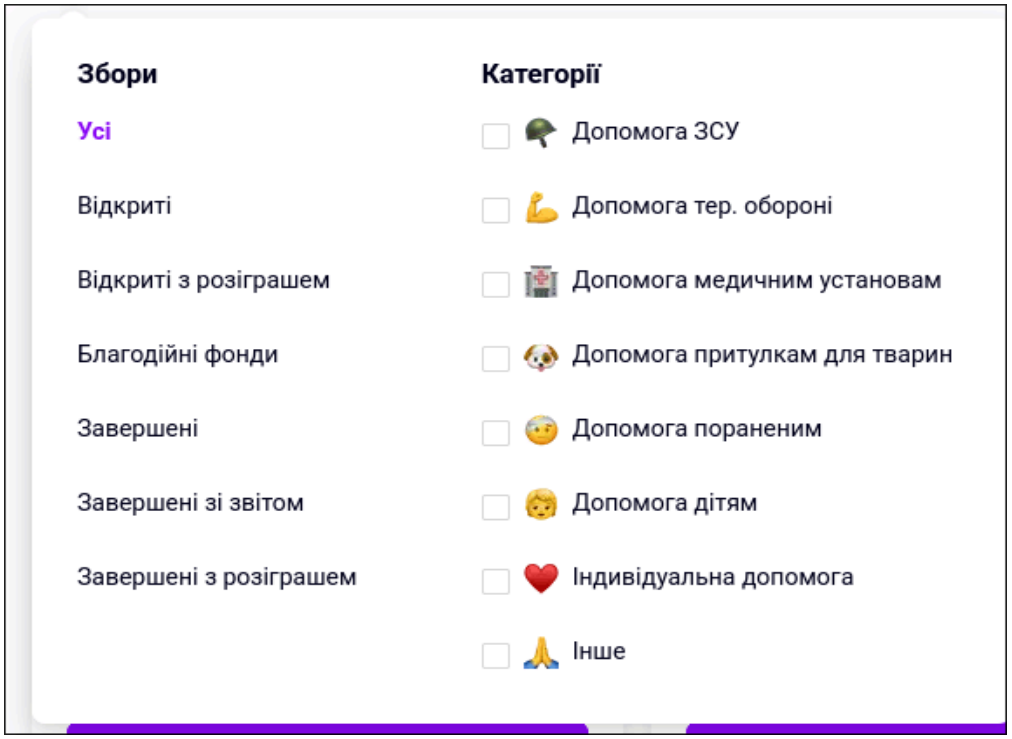

Рисунок 1.2 – Можливості фільтрування[1]

Також сайт надає гнучкі можливості для фільтрації та пошуку зборів, що покращує досвід користувача що зображено на рисунку 1.2.

Також на сторінці можна подати заявку на реєстрацію фонду що зображено на рисунку 1.3. Багато ініціатив працюють неофіційно або через партнерські фонди, тому важливо мати систему, яка враховує специфіку документального оформлення закупівель та може генерувати звіти відповідно до вимог чинного законодавства України.

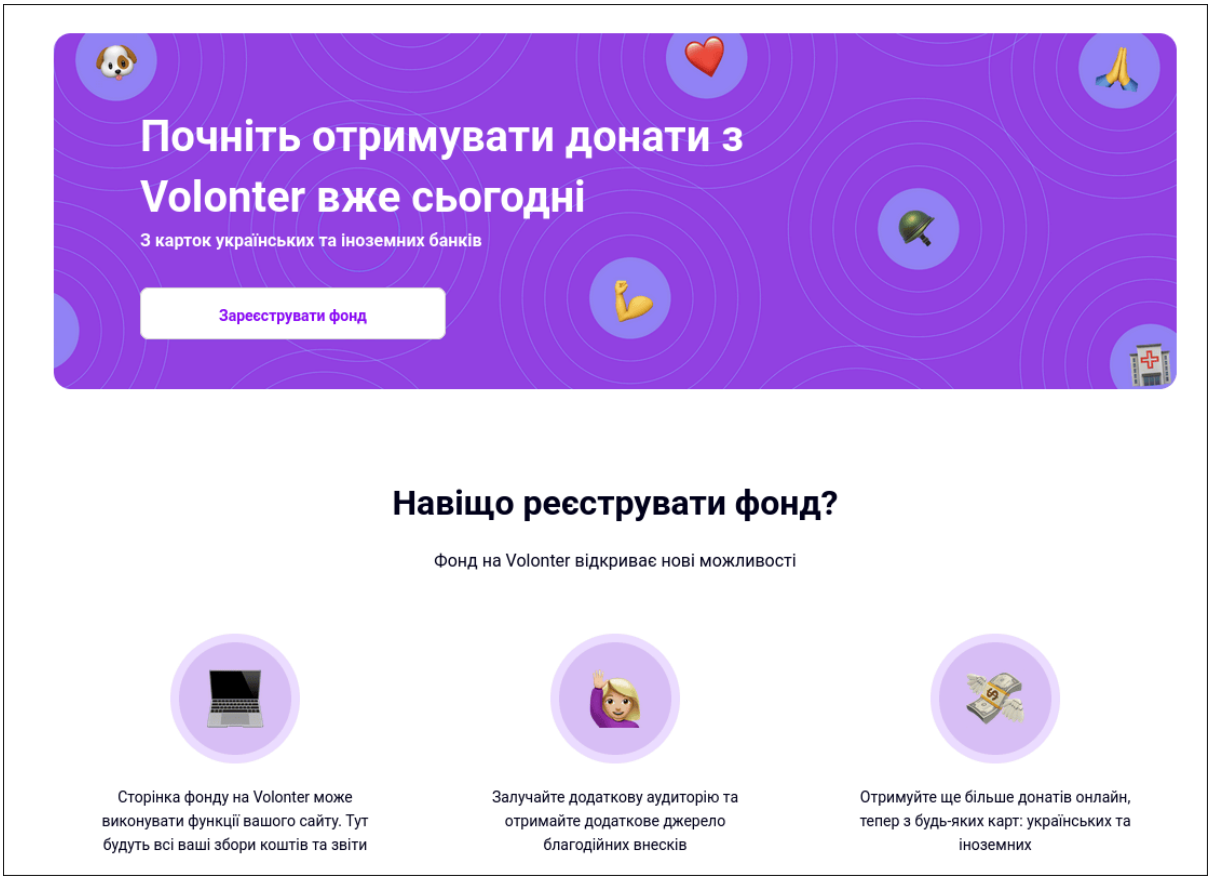

Рисунок 1.3 – Реєстрація фонду[1]

В умовах війни важливо не тільки залучати кошти, а й ефективно ними управляти, щоб підвищити довіру до волонтерського руху та зменшити корупційні ризики. Потреба у технологічному рішенні, зокрема бекенд-системі як частини повної системи, зумовлена:

- зростанням обсягів донатів (включно з міжнародними);

- необхідністю агрегувати великі обсяги даних (кошти, закупівлі, постачальники, запити);
- потребою в API для мобільних/веб-застосунків, телеграм-ботів, CRM-систем;
- інтеграцією з банківськими сервісами (Monobank, Приват24, LiqPay, Wayforpay тощо);
- необхідністю звітності у форматі, що можна легко верифікувати і поширити.

Back-end є критично важливою частиною такої системи, оскільки він забезпечує централізоване зберігання даних про транзакції, що дозволяє уникати маніпуляцій в даних та надавати можливість перевіряти волонтерів. Управління ролями (донор, волонтер, адміністратор) дозволяє чітко виокремити потреби кожного з учасників системи та адаптували функціонал під вимоги кожного. До прикладу донору треба бачити які збори наразі актуальні, волонтеру переглядати успіх його зборів, а військовим (або ж проксі, що купують необхідні товари) потрібна можливість переглядати чи вже хтось почав збирати на потрібні їм речі та комунікувати з волонтерами щодо просування тощо. REST API для взаємодії з фронтендом або іншими платформами та інтеграція з зовнішніми API (банки, платіжні системи, поштові сервіси) дозволяє покращити користувацький досвід та спросити організацію зборів, сконцентруватися на поширені серед людей та оптимізації самих закупівель, а не на оформленні звітності, постійному звітуванні військовим та донатерам. Взаємодія з API сервісу також дозволяє адаптувати збори у зручному форматі, до прикладу розробити віджети для стрімерів аби якісніше проводити збори.

1.2 Виявлення та вирішення проблем

Аудит, логування та безпека є фундаментальними складовими backend-системи для волонтерської платформи, адже йдеться про обробку фінансових транзакцій та персональних даних, які потребують максимальної прозорості та захисту. Завдяки ретельному логуванню можна не лише

відстежувати всі дії користувачів, а й оперативно виявляти помилки або потенційні зловживання, що суттєво підвищує рівень довіри до системи. У свою чергу, інтеграція автоматичного формування звітів у форматах PDF, Excel або CSV дозволяє не лише забезпечити швидке та зручне інформування донорів щодо цільового використання коштів, а й значно полегшує внутрішній контроль за обігом ресурсів, знижуючи навантаження на волонтерів, які раніше змушені були вручну готувати аналогічну документацію. Також надзвичайно важливу роль відіграє система сповіщень, яка повідомляє користувачів про надходження донатів, зміну статусу закупівель чи інші значущі події — це дозволяє зберігати прозорість комунікації, своєчасно реагувати на зміни та підтримувати мотивацію як серед волонтерів, так і серед донорів.

Попри те, що практика збору коштів в Україні вже набула певної усталеної форми, і волонтери використовують різні інструменти для обліку та звітності — від бухгалтерських CRM до зв’язок на кшталт Notion з Google Sheets чи Open Collective — ці рішення не відповідають реальним умовам, у яких працюють українські волонтерські ініціативи. Вони не враховують специфіку військових закупівель, де важлива не лише фінансова точність, а й оперативність, довіра до волонтера та безпека учасників. Крім того, такі системи часто не мають українського інтерфейсу або підтримки, що створює додатковий бар’єр для користувачів. Значна частина з них є платними або потребують складного налаштування, яке вимагає технічної підготовки, на яку у волонтерів зазвичай просто немає ресурсів. До прикладу для використання notion без обмежень необхідно заплатити за plus план використання, а в google spreadsheets важко користуватися без розуміння формул. І головне — ці інструменти не надають можливості легко оформлювати публічні звіти у зрозумілому для донаторів вигляді: з історіями, фотографіями, чеками, які викликають довіру та дозволяють прозоро демонструвати, як саме були витрачені кошти.

Якщо казати за актуальність розробки, то створення open-source або публічно доступної бекенд-системи, яку кожна волонтерська ініціатива зможе адаптувати під власні потреби, має величезну суспільну цінність. Така система

дозволить підвищити довіру до волонтерських проєктів завдяки прозорості даних, спростить ведення рутинної звітності та звільнить час волонтерів для виконання справжньої мети — допомоги тим, хто її потребує. Наявність такого інструменту також зменшить бар’єр входу для нових ініціатив, які тільки починають працювати, і не мають ресурсів або експертизи для створення власного обліку. У довгостроковій перспективі подібна система може бути масштабована на інші країни чи сфери, зокрема гуманітарні місії, міжнародні донорські програми або локальні соціальні ініціативи, що відкриває можливості для розвитку українського досвіду цифрової мобілізації та його експорту як технологічного і суспільного рішення.

1.3 Постановка задачі

Метою цього проєкту є створення backend-частини інформаційної системи, яка дозволить волонтерам організовано вести облік зборів, закупівель і звітності, забезпечить захист даних, автоматизацію процесів та відкриту інтеграцію з іншими системами (наприклад, банківськими API, логістикою тощо). Система має підтримувати чітке розмежування ролей (донори, волонтери, адміністратори), забезпечувати аудит змін та прозору історію транзакцій, формувати звіти у зручному форматі, а також передбачати можливість масштабування під різні типи ініціатив або географічні регіони.

У межах цієї задачі необхідно:

- спроєктувати архітектуру бекенду;
- розробити REST API для взаємодії з інтерфейсами та зовнішніми сервісами;
- реалізувати механізми автентифікації, авторизації, логування, звітності;
- забезпечити безпеку зберігання фінансових даних та персональної інформації;
- створити механізм інтеграції з платіжними та банківськими сервісами;
- забезпечити відкритість і можливість адаптації системи іншими командами.

Ця система має стати фундаментом для цифрової інфраструктури волонтерських ініціатив, яка відповідатиме як на нагальні потреби сьогодення, так і закладатиме потенціал для довгострокового розвитку громадянського суспільства в Україні.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Мета проєкту

Програмна система, призначена для підтримки волонтерських ініціатив, зокрема управління зборами коштів та звітністю щодо закупівель, вимагає створення надійної, масштабованої та безпечної серверної частини. Основні користувачі системи — волонтери, координатори ініціатив, а також довірені особи, що відповідають за прийом допомоги, закупівлі та звітність. Програмне забезпечення повинно забезпечувати зручне керування заявками, збором коштів, витратами, закупленими товарами та прозору взаємодію з користувачами, що підтримують ініціативу.

Для реалізації серверної частини обрано технологію FastAPI, яка є сучасним фреймворком для створення високопродуктивних REST API. Вона дозволяє ефективно описувати типи даних, документувати інтерфейс системи, забезпечувати високу швидкість розробки та відповідність стандартам OpenAPI. Взаємодія між клієнтськими застосунками (мобільними чи веб) і сервером буде реалізована за допомогою REST API.

2.2 Припущення та залежності

У процесі створення веб-застосунку для підтримки волонтерських ініціатив важливо чітко визначити ключові припущення, на яких базується проєкт, а також виявити основні залежності, що можуть впливати на його функціональність та подальший розвиток. Це дозволяє формувати реалістичні очікування, краще планувати технічні рішення та забезпечити ефективне впровадження системи.

Одним з основних припущень є наявність у користувачів стабільного доступу до Інтернету. Передбачається, що як волонтери, так і отримувачі допомоги мають технічну можливість підключення до мережі та можуть використовувати веб-застосунок на різних пристроях: комп’ютерах, планшетах або смартфонах. Це критично важливо, адже функціонування системи передбачає

регулярний обмін даними з сервером та оперативне оновлення інформації. Незважаючи на можливі труднощі з якістю з’єднання в окремих регіонах, очікується, що більшість користувачів мають доступ до мережі.

Ще одне важливе припущення стосується рівня цифрової грамотності користувачів. Система розрахована на аудиторію, яка має базові навички роботи з веб-додатками та здатна самостійно взаємодіяти з інтерфейсом. Це дозволяє зосередитись на створенні зручного, інтуїтивно зрозумілого UI без необхідності проведення навчання для кожного нового користувача. Хоча деякі функції можуть вимагати короткого ознайомлення, загальний рівень цифрових навичок має бути достатнім для повноцінного користування платформою.

Розробка та експлуатація застосунку також пов’язані із зовнішніми залежностями. Зокрема, значну роль відіграють сторонні сервіси, які надають фінансову або інформаційну підтримку. Оскільки сам застосунок не проводить фінансових транзакцій, а лише отримує інформацію про платежі через API зовнішніх платформ, його стабільна робота значною мірою залежить від доступності та надійності цих сервісів. Збої, оновлення або зміни в умовах доступу можуть безпосередньо впливати на функціональність системи.

Ще однією суттєвою залежністю є використання сторонніх технологій для зберігання та обробки даних. Хоча застосунок не оперує чутливими персональними або фінансовими даними, важливо забезпечити збереження інформації про збори, потреби та звіти. Це вимагає впровадження надійних технічних рішень, які гарантують цілісність, доступність і захист даних від втрати або несанкціонованого доступу.

Окреме припущення стосується активної участі користувачів у розвитку системи. Очікується, що волонтери та організації будуть регулярно оновлювати дані, публікувати звіти та взаємодіяти між собою. Саме така залученість є запорукою ефективної роботи платформи. Відповідно, необхідною є постійна комунікація між розробниками та представниками спільноти для врахування їхніх потреб і зворотного зв’язку.

Фіксація ключових припущень і залежностей є критично важливою на етапі проєктування та розвитку системи. Вона дозволяє чітко окреслити межі її функціонування, визначити потенційні ризики та сформувати пріоритети для технічного розвитку. Такий підхід створює гнучке підґрунтя для адаптації веб-застосунку до реалій волонтерської діяльності та змін зовнішнього середовища.

2.3 Функціональні вимоги

Функціональність серверної частини має бути гнучко структурована згідно з ролями користувачів. Основні функціональні вимоги до системи:

- створення, редагування та перегляд зборів коштів;
- можливість додавати цілі та опис призначення збору;
- створення закупівель, що прив’язані до конкретного збору;
- можливість додавання чеків, фото, документів для підтвердження закупівель;
- формування звітів по кожному збору (сумарна сума, витрати, залишки, перелік придбаного);
- реєстрація волонтерів, їх ролей, прав доступу;
- підтвердження внесків (вручну або автоматично при інтеграції з платіжними системами);
- зберігання контактної інформації постачальників і благодійників;
- контроль змін (аудит) і логування дій користувачів.

Для зберігання інформації буде використано PostgreSQL як основну реляційну базу даних. Вона забезпечує стабільність, ACID-властивості, можливість складних запитів та високу продуктивність. Для роботи з ORM використовується SQLModel — бібліотека, що поєднує переваги SQLAlchemy та Pydantic.

Особливу увагу буде приділено питанням безпеки. Для цього була розроблена систему доступу в залежності від ролі користувача. Для аутентифікації

та авторизації буде реалізовано механізм на основі JWT (JSON Web Tokens). Передбачено розмежування прав доступу до функцій системи, а всі чутливі дані будуть передаватись через HTTPS із можливістю шифрування в базі.

Файли, пов’язані з підтвердженням витрат (фото чеків, документи), будуть зберігатись в зовнішньому файловому сховищі — наприклад, S3-сумісному бакеті, що дозволить масштабувати обсяг даних і зберігати їх окремо від основної БД.

У подальшому можлива інтеграція з платіжними сервісами (наприклад, LiqPay, WayForPay або Monobank), щоб автоматизувати фіксацію надходжень коштів. Крім того, може бути реалізована підтримка Telegram-бота або мобільного застосунку для полегшення доступу до інформації.

Документація API буде створена автоматично за допомогою OpenAPI/Swagger, що надає зручний інтерфейс для перегляду ендпоінтів, їх параметрів та прикладів відповідей. Це дозволить прискорити розробку клієнтських частин системи та забезпечити можливість інтеграції з іншими платформами.

Таким чином, система повинна відповідати вимогам масштабованості, прозорості, безпеки та бути готовою до розширення з урахуванням потреб різних ініціатив.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПЗ

3.1 UML проєктування ПЗ

У процесі проектування системи волонтерських зборів було розроблено діаграми прецедентів, що відображають взаємодію різних категорій користувачів із системою. Діаграми охоплюють функціонал для адміністраторів, неавторизованих користувачів, волонтерів та отримувачів допомоги.

Use case діаграма представлена на рисунку 3.1.

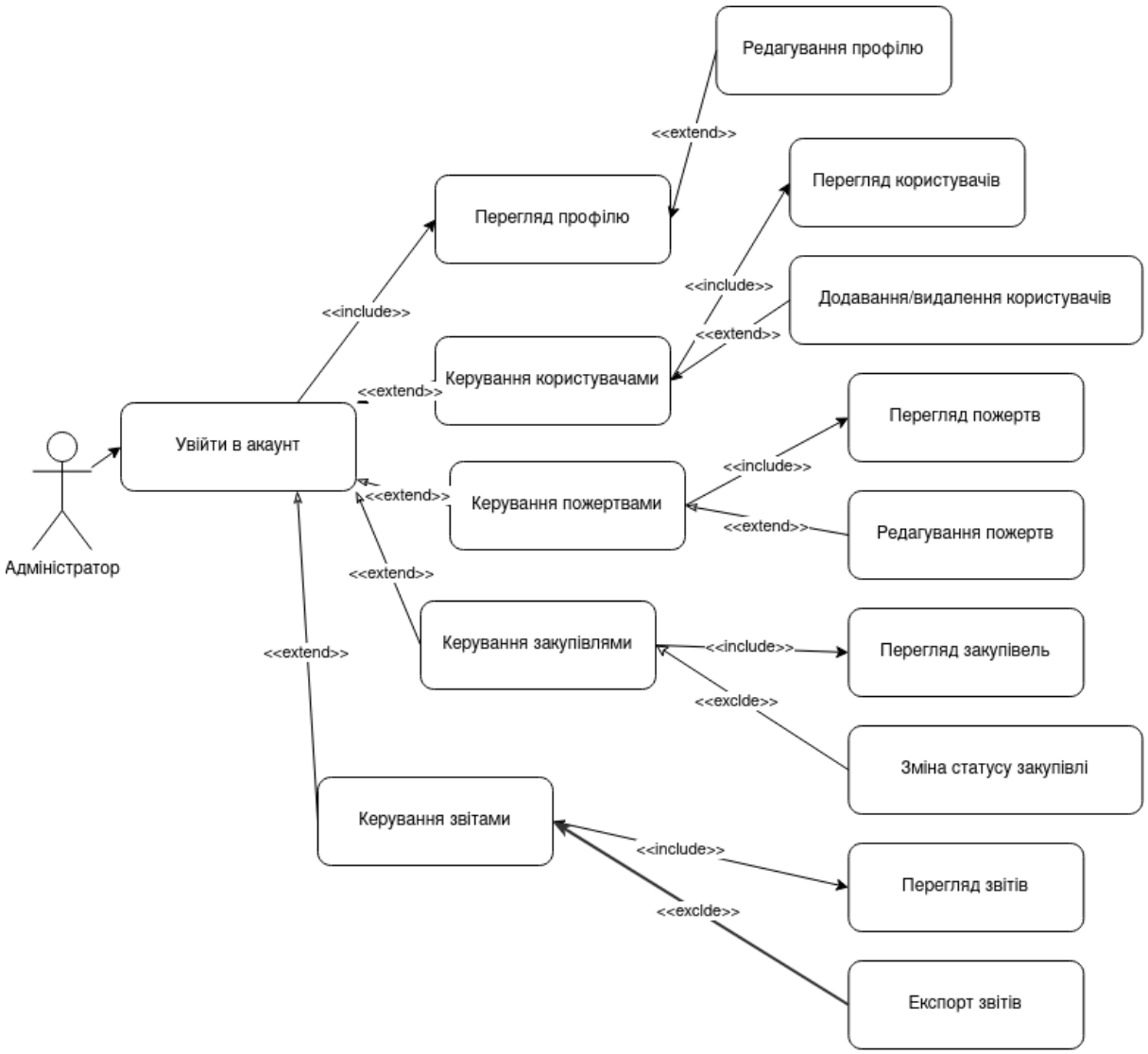

Рисунок 3.1 – Use Case діаграма для ролі Адміністратора(рисунок виконаний самостійно)

Адміністратор системи має розширені права доступу, що включають:

- управління користувацькими профілями (перегляд, редагування, додавання та видалення);
- контроль над зборами (перегляд усіх пожертв, редагування інформації);
- керування процесом закупівель (зміна статусів, моніторинг);
- роботу зі звітами (перегляд, експорт у різних форматах).

Для інших категорій користувачів (Рисунок 3.2) передбачено чимало кейсів.

По-перше, це неавторизовані користувачі можуть здійснювати пошук зборів, переглядати інформацію про активні кампанії та проходити авторизацію для отримання додаткових функцій.

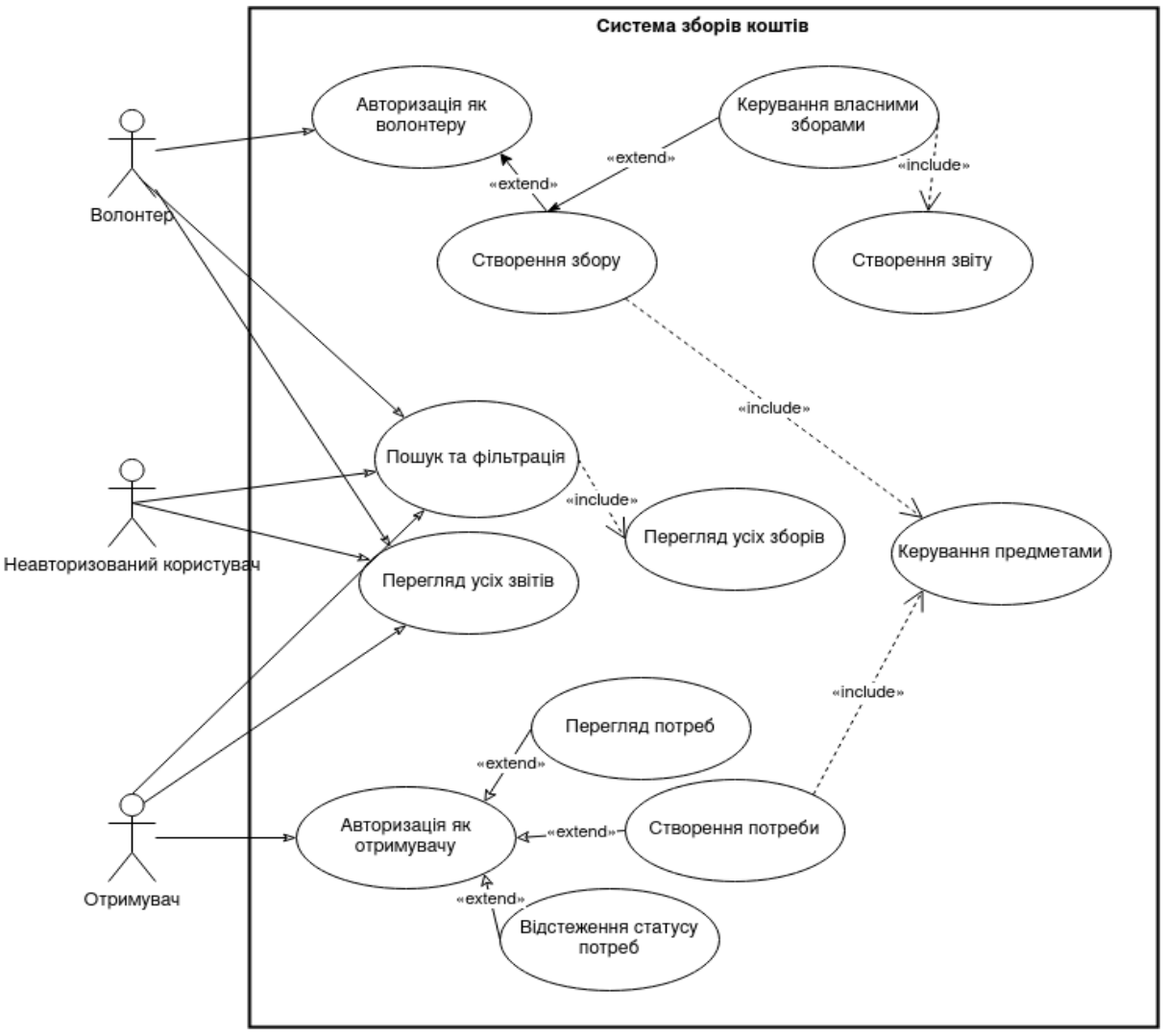

Рисунок 3.2 – Use Case діаграма для ролі користувачів що не є Адміністраторами (рисунок виконаний самостійно)

По-друге, волонтери після авторизації отримують доступ до створення та управління власними зборами, включаючи формування списків потреб, моніторинг статусу зборів та перегляд інших активних кампаній.

Отримувачі допомоги мають доступ до інформації про призначені їм збори, можуть відстежувати стан задоволення потреб та ознайомлюватися зі звітністю.

Система реалізує чіткий розподіл прав доступу:

- адміністратори мають повний контроль;
- волонтери керують призначеними зборами;
- отримувачі мають обмежений доступ до відповідної інформації;
- гості системи можуть ознайомлюватися з публічними даними.

Така структура забезпечує ефективну взаємодію всіх учасників процесу зборів при збереженні безпеки даних і чіткому розмежуванні обов'язків.

3.2 Проєктування архітектури ПЗ

Для розробки серверної частини системи волонтерських зборів було обрано трирівневу архітектуру, яка є оптимальним рішенням для створення масштабованих та гнучких веб-додатків. Вибір цієї архітектури обумовлений її перевагами у контексті даного проекту. Така архітектура є простотою для розробки та підтримки бо окрема розробка кожного рівня дозволяє паралелізувати процес розробки та відокремити помилки в одній від ефекту на інші. Також така архітектура гнучка до змін, модифікації на одному рівні не вимагають змін на інших. Важливим залишається фактор масштабованість яка надає можливість масштабувати окремі компоненти системи та безпека що дає чітке розмежування рівнів та зменшує поверхню атак.

Сама архітектура системи включає три основні рівні:

Рівень доступу до даних який взаємодіє з базою даних. Для реалізації було використано паттерн "Repository" для абстракції операцій з БД, також інтеграція з postgresql через … що дає можливість асинхронного доступу до бази та збільшення продуктивності як з боку оптимізованості БД, так і з боку взаємодії з

нею через паралельну обробку. Також моделі даних на основі Pydantic для забезпечують типізації та валідації даних для уникнення помилок та некоректного трактування. Цей рівень інкапсулює всю логіку роботи з персистентними даними, надаючи зручний інтерфейс для бізнес-логіки.

Рівень бізнес-логіки є ядром системи, де відбувається обробка всіх ключових процесів. На цьому рівні реалізовано основні сценарії роботи, такі як управління зборами, де кожна компанія має свій життєвий цикл від створення до завершення, включаючи моніторинг статусу та автоматичні сповіщення. Система рейтингів волонтерів працює на основі оцінок та відгуків, що забезпечує прозорість та мотивацію для учасників.

Важливим компонентом є механізми валідації, які перевіряють коректність вхідних даних перед обробкою, запобігаючи помилкам та потенційним загрозам безпеці. Бізнес-правила системи реалізовані у вигляді окремих модулів, що дозволяє легко адаптувати їх до змінних вимог. Інтеграція з зовнішніми сервісами, такими як платіжні системи для збору коштів та email-розсилки для сповіщень, забезпечує комплексний підхід до управління волонтерською діяльністю. Усі ці компоненти реалізовані як незалежні сервіси, що значно спрощує їх тестування, супровід та майбутнє вдосконалення.

Архітектура API організована за модульним принципом, де кожен функціональний компонент (управління зборами, вимогами, волонтерами тощо) має свою чітко визначену структуру, зображено на рисунку 3.3.

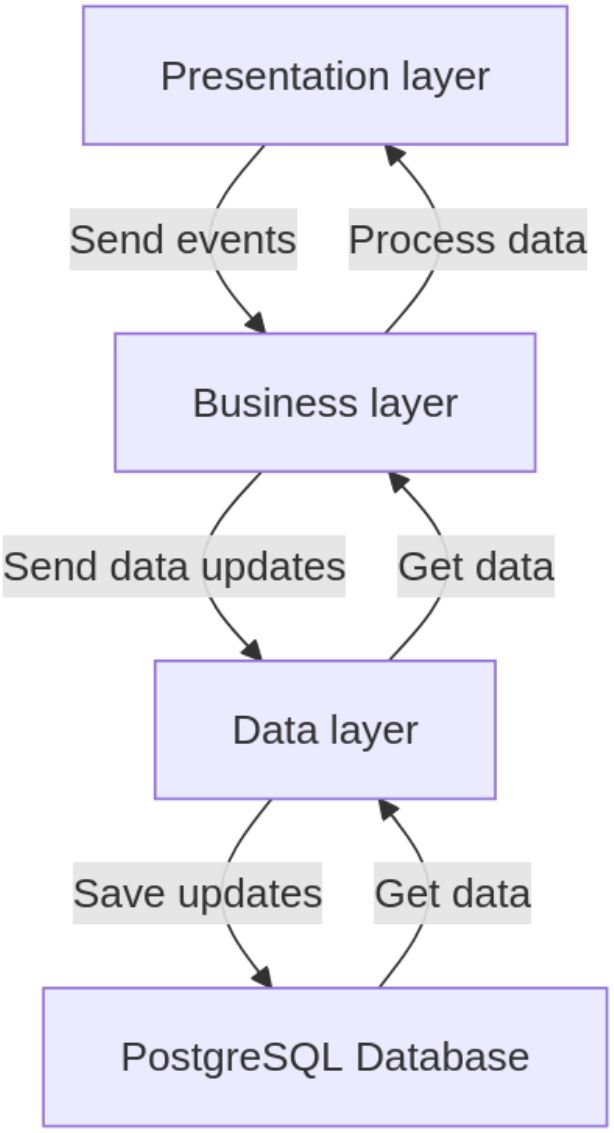

Рисунок 3.3 – Схематична архітектура застосунку (рисунок виконаний самостійно)

Презентаційний рівень відповідає за надання API для взаємодії з клієнтською частиною. Для його реалізації обрано фреймворк FastAPI, який дозволяє швидко створювати ефективні RESTful endpoints з автоматичною генерацією документації у форматі OpenAPI. Система безпеки побудована на JWT-аутентифікації, що забезпечує захист даних та контроль доступу.

Особливу увагу приділено обробці файлів, зокрема завантаженню зображень для зборів, що реалізовано з дотриманням принципів безпеки та

продуктивності. RESTful підхід до проектування API забезпечує стандартизовану структуру ресурсів, використання HTTP-методів відповідно до їх семантики, підтримку кешування для підвищення продуктивності та легку інтеграцію з різними клієнтськими рішеннями.

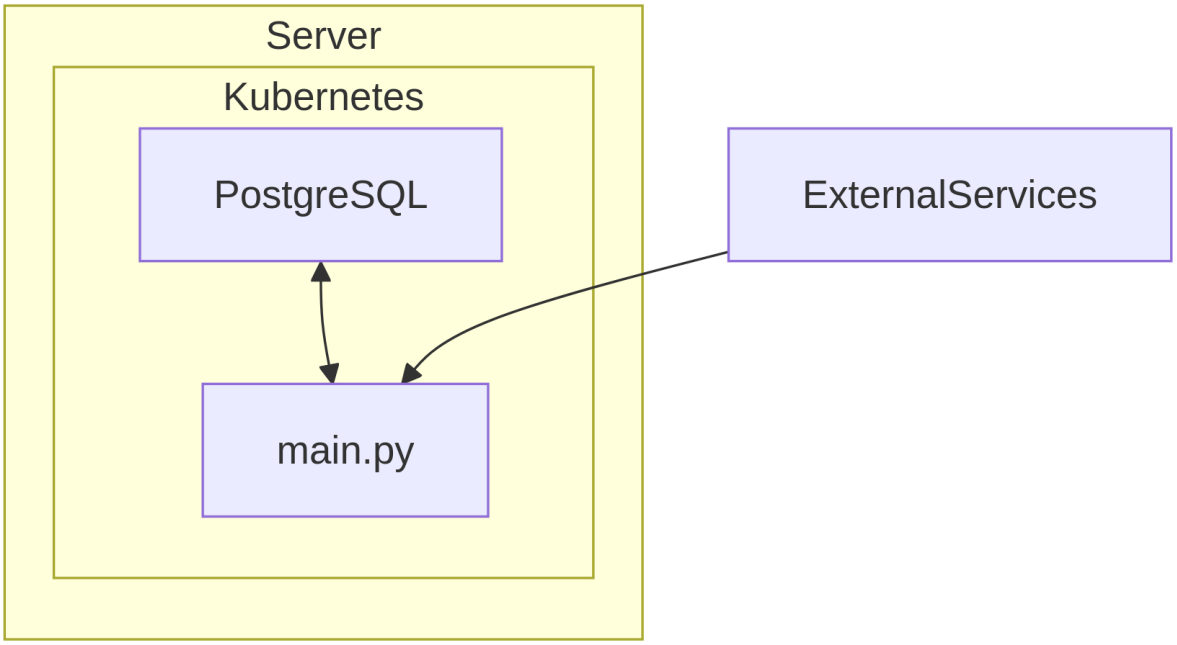

Рисунок 3.4 – Діаграма розгортання системи (рисунок виконаний самостійно)

Система розгорнута на хмарній платформі з використанням Kubernetes, що надає низку переваг: можливість швидкого масштабування ресурсів у відповідь на зростаюче навантаження, високу доступність сервісу за рахунок розподіленої архітектури, спрощені процеси розгортання нових версій без простою, а також автоматизацію CI/CD pipelines для безперервного інтегрування та доставки оновлень.

Архітектура системи має потенціал для подальшого розширення: перехід на мікросервісну архітектуру дозволить ще більше підвищити гнучкість та масштабованість, додавання черг повідомлень (RabbitMQ/Kafka) забезпечить надійну обробку асинхронних подій, інтеграція додаткових зовнішніх сервісів

розширить функціональність, а підтримка gRPC дозволить оптимізувати внутрішню комунікацію між компонентами системи.

Такий підхід до архітектури поєднує в собі надійність, продуктивність та гнучкість, забезпечуючи стабільну роботу системи навіть при значному зростанні навантаження, спрощуючи процеси підтримки та надаючи широкі можливості для адаптації до нових вимог та вдосконалень в майбутньому.

3.3 Проєктування структури зберігання даних

В якості основного рішення для зберігання даних системи волонтерських зборів було обрано PostgreSQL – сучасну об'єктно-реляційну систему управління базами даних. Даний вибір обумовлений комплексом технічних переваг, які забезпечують високу продуктивність, надійність та безпеку системи [6].

PostgreSQL демонструє високу ефективність обробки запитів навіть при значних навантаженнях, що досягається за рахунок оптимізованого виконання SQL-запитів та підтримки складних агрегацій даних. Система забезпечує повноцінну підтримку принципів ACID, гарантуючи атомарність операцій, контроль цілісності даних, правильну ізоляцію транзакцій та стійкість результатів після виконання операцій.

Важливим аспектом є вбудовані механізми безпеки, що включають шифрування даних, деталізовану систему розмежування доступу на основі ролей (RBAC), підтримку захищених SSL-з'єднань та можливість аудиту всіх операцій з даними. Масштабованість рішення дозволяє як вертикально збільшувати потужності сервера, так і організовувати горизонтальне масштабування через технології шардингу.

Для підвищення ефективності роботи було реалізовано асинхронний інтерфейс взаємодії з базою даних через asyncpg, що значно підвищує продуктивність системи при обробці асинхронних запитів. Ключові таблиці оптимізовані за допомогою спеціальних індексів, а для часто використовуваних даних застосовано механізми кешування.

Архітектура бази даних враховує можливість подальшого масштабування та розширення функціоналу. Інтеграція процедурних мов, зокрема PL/pgSQL, відкриває можливості для реалізації складних бізнес-логічних операцій безпосередньо на рівні бази даних.

Під час аналізу потреб системи, було розроблено логічну модель сутностей системи, що зображена на рисунку 3.5.

Розроблена логічна модель системи волонтерських зборів включає наступні ключові сутності:

Волонтери (Volunteer) - Профілі користувачів, які організовують збори. Містять контактні дані (телефон), ім’я, прізвище, вік, статус доступності, а також належать до певної категорії діяльності (через зв’язок зі Специфікаціями). Пов’язані з відповідним обліковим записом користувача.

Отримувачі (Recipient) - Інформація про осіб або організації, які отримують допомогу через систему. Містять контактні дані, пов’язані з обліковим записом користувача.

Збори (Fund) - Описують кампанії зі збору коштів або матеріальних цінностей. Включають назву, опис, статус (Active, Completed, Cancelled), посилання на звіт (Report), волонтера-організатора (Volunteer) та додаткові медіа (картинки, URL).

Потреби (Requirement) - Перелік конкретних потреб у межах зборів з інформацією про пріоритетність, терміни виконання, опис, а також отримувача, для якого ці потреби сформовані. Кожна потреба пов’язана зі збором (Fund) та отримувачем (Recipient).

Предмети (Item) - Конкретні позиції, які необхідні для задоволення потреб. Містять назву, кількість, категорію, а також інформацію про те, які збори їх резервують.

Звіти (Report) - Інформація про результати зборів, що включає оцінку (рейтинг) та фінальний висновок. Кожен звіт пов’язаний з одним збором.

Специфікації (Specific) - Додаткові категорії або напрями діяльності для волонтерів, які визначають їх спеціалізацію.

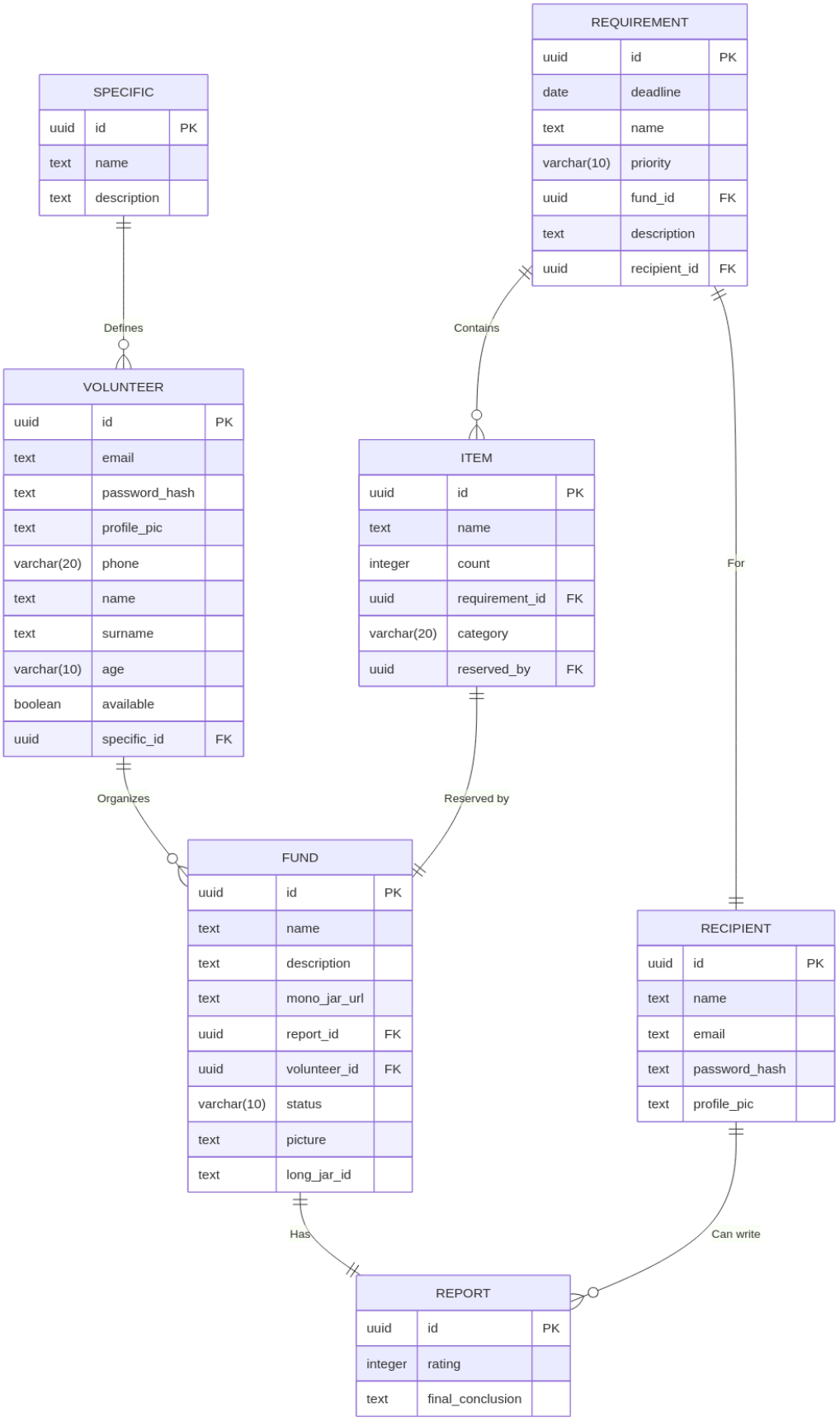
SPECIFIC
uuid id PK
text name
text description
Defines
VOLUNTEER
uuid id PK
text email
text password_hash
text profile_pic
varchar(20) phone
text name
text surname
varchar(10) age
boolean available
uuid specific_id FK
Organizes
REQUIREMENT
uuid id PK
date deadline
text name
varchar(10) priority
uuid fund_id FK
text description
uuid recipient_id FK
Contains
For
ITEM
uuid id PK
text name
integer count
uuid requirement_id FK
varchar(20) category
uuid reserved_by FK
Reserved by
FUND
uuid id PK
text name
text description
text mono_jar_url
uuid report_id FK
uuid volunteer_id FK
varchar(10) status
text picture
text long_jar_id
Has
RECIPIENT
uuid id PK
text name
text email
text password_hash
text profile_pic
Can write
REPORT
uuid id PK
integer rating
text final_conclusion

Рисунок 3.5 – ER діаграма системи (рисунок виконаний самостійно)

Така логічна модель дозволяє ефективно керувати всіма аспектами волонтерської діяльності - від організації зборів та визначення потреб до розподілу допомоги та аналізу результатів. Модель забезпечує цілісність даних, легкість їх обробки та можливість швидкого пошуку необхідної інформації.

Розроблена структура є оптимальною для реалізації всіх бізнес-процесів системи, забезпечуючи при цьому гнучкість для майбутніх змін та розширення функціоналу. Вона слугує надійною основою для побудови ефективної системи управління волонтерськими зборами, що відповідає сучасним вимогам до продуктивності, безпеки та зручності використання.

3.4 Приклади найцікавіших алгоритмів та методів

У межах системи автоматизації процесів розподілу завдань, є ключовим алгоритм оптимізації розподілу потреб між волонтерами.

Алгоритм реалізує механізм оптимального призначення потреб між волонтерами, ґрунтуючись на принципі мінімізації витрат. Основою є матриця вартостей, де кожен елемент відображає умовну "ціну" призначення певної потреби конкретному волонтеру. Чим менше значення у клітинці — тим вигіднішим є таке призначення. На етапі ініціалізації формується матриця розміром N×M, де N — кількість волонтерів, а M — кількість актуальних потреб. Всі значення спершу встановлюються на нуль.

Подальша обробка полягає в розрахунку індивідуальної вартості для кожної пари «волонтер–потреба». У цьому процесі враховується поточне навантаження на волонтера, його рейтинг, а також пріоритет самої потреби. Високо пріоритетні запити отримують більшу вагу, а волонтери з високими рейтингами вважаються більш придатними до виконання завдань. Щоб забезпечити стабільну роботу алгоритму, матриця вартостей нормалізується — всі значення масштабуються до інтервалу від нуля до одного.

Після цього застосовується модифікований угорський алгоритм [2], реалізований через функцію linear_sum_assignment з бібліотеки SciPy [3]. Цей метод дозволяє знайти такий розподіл пар «волонтер–потреба», який мінімізує сумарні витрати. Результати обробляються з урахуванням обмеження на максимальну кількість потреб, які може обробити один волонтер. У разі, якщо частину потреб не вдалося призначити на цьому етапі, алгоритм додатково розподіляє залишкові потреби між волонтерами з найменшим поточним навантаженням.

Реалізація побудована на використанні векторизованих операцій з бібліотеки NumPy [4], що дозволяє досягти високої продуктивності навіть за значної кількості вхідних даних. Таким чином, забезпечується не лише ефективне балансування навантаження, а й урахування якісних характеристик волонтерів і гнучке реагування на зміну вхідних умов.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Архітектура проекту

Архітектура серверної частини для організації волонтерської діяльності розроблена з урахуванням сучасних підходів до back-end розробки, що поєднує використання шарової моделі, масштабованість, використання контейнеризації. В основі роботи сервера бібліотека FastAPI та сервер uvicorn, що забезпечують просту та зрозумілу логіку обробки запитів та підтримки програмного забезпечення. Це дозволяє організувати оптимальним чином проєкт в залежності від складності та потреб.

Серед особливостей Kubernetes [5] є також гнучке масштабування та можливість створення кластерів серверів що може знадобитися при збільшенні попиту на проєкт.

Серед структури безпосередньо серверу варто виділити теку pkg, де реалізовані всі обробники, функції роботи з базою даних та додаткові утиліти для роботи з авторизацією.

Основна логіка обробки запитів розміщена в теці pkg/api, яка структурована відповідно до доменних областей платформи. Кожен файл у цій теці відповідає за окремий блок функціональності: fund.py — за логіку роботи зі зборами коштів, requirement.py — з потребами, volunteer.py — з профілем волонтера, а recipient.py — з даними отримувача допомоги. Такий підхід дозволяє чітко розмежувати зони відповідальності, що значно спрощує супровід коду та його масштабування. Кожен з цих модулів включає маршрути FastAPI, які об’єднують функції з базовою логікою, що полегшує підтримку REST-архітектури.

Моделі, з якими працює система створюються у файлі models.py, який містить оголошення Pydantic-схем. Вони забезпечують зв’язок між всіма структурами застосунку, що дозволяє реалізувати перевірку типів, обробку вхідних даних та формування відповідей у єдиному стилі. Робота безпосередньо з базою даних організована через окремий модуль database.py, що інкапсулює логіку створення сесій, підключення та ініціалізації таблиць.

Файл middleware.py відповідає за реалізацію проміжної обробки запитів, зокрема логування, що мають виконуватись до потрапляння запиту до основного ендпоінту. Це дозволяє легко масштабувати систему без зміни основної логіки роботи модулів.

Особливу увагу приділено також модулю utils.py, який містить допоміжні функції для генерації токенів. Завдяки цьому логіка залишається чистою, а повторюваний функціонал виноситься у спільне місце, що покращує читаність та повторне використання коду.

Файл optimizer.py закладений як розширення логіки оптимізації — розрахуноку пріоритетності або рекомендованих дій для волонтерів. Наразі цей компонент перебуває в стадії розробки.

Усі завантаження зображень або файлів зберігаються у теці uploads, яка також доступна через маршрути сервера для публічного або авторизованого перегляду. Це дозволяє забезпечити зберігання візуальних матеріалів — наприклад, фото чеків, підтверджень, або зображень зборів — без потреби підключення до зовнішніх сховищ.

Запуск сервера відбувається через точку входу bin/server.py, де налаштовано об'єднання всіх роутерів, підключення до бази даних, ініціалізація CORS та запуск через uvicorn. Такий підхід дозволяє централізовано керувати всіма параметрами старту застосунку та легко додавати додаткові функції при потребі.

Наявність Dockerfile та файлів залежностей (pyproject.toml, requirements.txt, uv.lock) дозволяє легко відтворити середовище у контейнері. Це, у свою чергу, дає змогу швидко розгортати застосунок у хмарі або на локальному сервері, зберігаючи повну відповідність конфігурації незалежно від ОС. Така підготовка є передумовою для майбутнього переходу до Kubernetes та хмарної інфраструктури з балансуванням навантаження та автоматичним масштабуванням.

Таким чином, архітектура бекенду демонструє чітке розділення відповідальностей, модульність і готовність до масштабування. Вона побудована на сучасному стеку технологій, що забезпечує високу гнучкість, безпеку, та зручність як для розробників, так і для користувачів.

Файлова структура програмного забезпечення є важливою складовою архітектури, адже саме вона визначає порядок і логіку розміщення всіх елементів системи. Оптимізована структура дозволяє максимально полегшити підтримку, масштабування та розвиток застосунку у майбутньому (див.рис. 4.1).

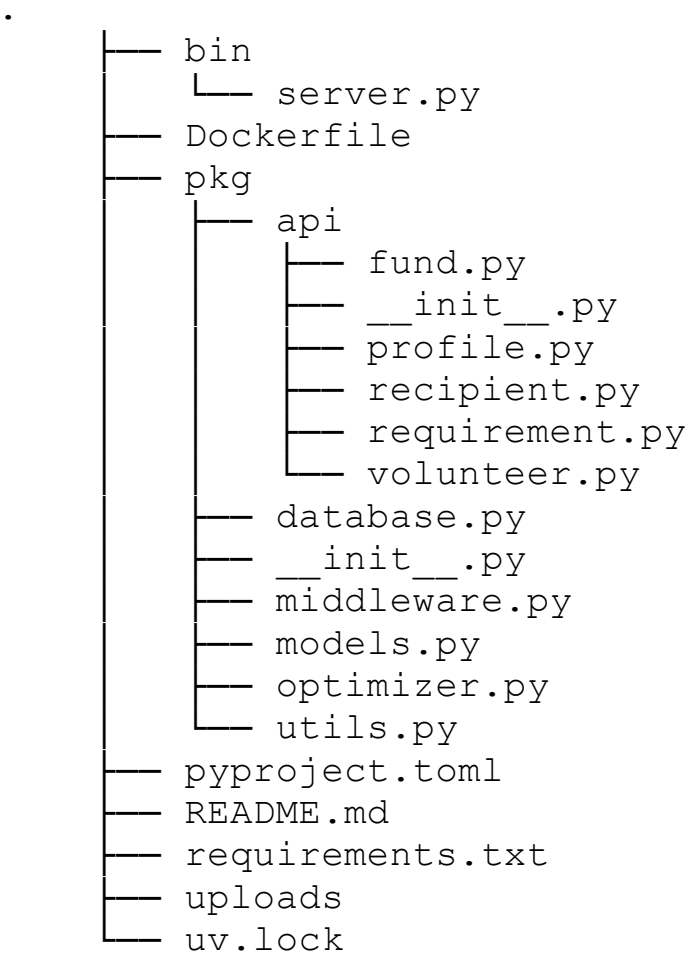

Рисунок 4.1 – Файлова структура проекту (рисунок виконаний самостійно)

4.2 Реалізація обробки запитів

Початок роботи користувача системи в більшості сценаріях роботи буде починати з авторизації. Через це необхідно якісно обробляти запити цього типу, надавати всю потрібну інформацію та забезпечити захист даних у разі спроби обходу захисту системи. Нижче наведений код що видає помилку у разі не

валидных вхідних даних, перевіряє роль, створює токен та надає вичерпну інформацію у разі підтвердження авторизації.

```
@profile_router.post("/login")
async def login_endpoint(user_data: UserLogin, req: Request) -> LoginResponse:
    db = req.app.state.db
    try:
        user = await user_login(db, user_data.email, user_data.password)
    except Exception as e:
        raise HTTPException(status_code=401, detail="Invalid credentials")

    role: RoleEnum
    match user:
        case Volunteer():
            role = RoleEnum.volunteer
        case Recipient():
            role = RoleEnum.recipient
    token = create_access_token(
        data={"sub": user.email, "id": user.id},
    )

    return LoginResponse(
        access_token=token,
        role=role,
        user=user,
    )
```

Як можна побачити, функція обробки не має в собі логіки взаємодії з базою даних або алгоритму перевірки вхідних даних, навпаки, викликає функції з відповідних модулей системи. Також функція повертає модель що забезпечує валідацію даних та документованість системи.

Якщо переходити детальніше у роботи з базою даних, то відбувається вона таким чином: отримання змінної db з глобального контекста застосунку, виклик функції з передачею йому бази даних та необхідних змінних та подальша обробка

результату, що також передбачає можливість помилки при неправильних вхідних даних. Сама функція звернення до бази даних має такий код:

```
async def user_login(db: Database, email: str, password: str) -> Volunteer
| Recipient:
    query = "SELECT * FROM Volunteer WHERE Email = :email"
       user = await db.connection.fetch_one(query=query, values={"email":
email})
    if user and verify_password(password, user["PasswordHash"]):
        return Volunteer(
            id=user["ID"],
            email=user["Email"],
            profile_pic=user["ProfilePic"],
            phone=user["Phone"],
            name=user["Name"],
            surname=user["Surname"],
            age=user["Age"],
            available=user["Available"],
        )
    query = "SELECT * FROM Recipient WHERE Email = :email"
       user = await db.connection.fetch_one(query=query, values={"email":
email})
    if user and verify_password(password, user["PasswordHash"]):
        return Recipient(
            id=user["ID"],
            name=user["Name"],
            email=user["Email"],
            profile_pic=user["ProfilePic"],
        )
    raise DatabaseException("Invalid email or password")
```

Даний код виконує пошук даних серед різних ролей та у разі незнаходження в обох випадках повертає помилку. Використання sql покращує гнучкість роботи а додавання змінних до запиту за допомогою спеціалізованої функції дозволяє убезпечити систему від sql ін’єкцій. Після отримання даних, вони передають до функції, що генерує токен для авторизованого взаємодії з системою:

```
def create_access_token(data: dict, expires_delta: Optional[timedelta] =
None):
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (
        expires_delta
        if expires_delta
        else timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Таким чином структура та код можуть бути легко розширення при подальшій розробці та зберегти легкість у підтримці та супроводженні коду.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Підхід до тестування та розробка тестових випадків

Забезпечення якості та надійності серверної частини системи координації волонтерської допомоги є критично важливим завданням. Тестування розглядалося як ключовий етап розробки, спрямований на перевірку відповідності реалізованого API та бізнес-логіки визначеним вимогам, а також на забезпечення стабільної та коректної роботи бекенду в різних сценаріях використання. Основна мета тестування полягала у виявленні та усуненні можливих дефектів, а також у підтвердженні коректності обробки даних і надійності серверних сервісів.

Процес тестування охоплював перевірку всіх ключових аспектів серверної логіки. Особливу увагу було приділено коректності обробки запитів та відповідей API: перевіряється правильність валідації вхідних даних, логіка бізнес-процесів (створення, редагування та видалення зборів, вимог, звітів тощо), а також коректність роботи системи аутентифікації та авторизації. Були розроблені інтеграційні сценарії, що імітують реальні послідовності дій користувачів, з метою перевірки наскрізної функціональності бекенду — від авторизації, створення та отримання даних зборів і вимог, до обробки звітів та управління профілями.

Крім функціональної коректності, тестування включає оцінку стійкості та продуктивності серверної частини при навантаженнях, а також правильність обробки помилок і виключень, зокрема коректне повернення відповідних кодів помилок і повідомлень клієнту.

Особливий акцент у тестуванні було зроблено на коректності взаємодії бекенду з базою даних: перевіряється цілісність даних, правильність виконання транзакцій та узгодженість стану системи після різних операцій.

5.2 Визначення плану впровадження

Для ефективного планування етапу впровадження онлайн-системи волонтерських зборів доцільно скласти план із розподілом етапів. Це дозволяє зменшити ймовірність виникнення помилок під час запуску. Нижче у таблиці 5.1 наведено етапи впровадження та відповідні дати.

Таблиця 5.1 – План впровадження онлайн-системи волонтерських зборів

Назва етапу	Дата початку	Дата завершення
Внутрішнє тестування	30.05.2025	02.06.2025
Підготовка до закритого тестування	03.06.2025	05.06.2025
Закрите тестування	06.06.2025	08.06.2025
Підготовка до виходу в «Production»	09.06.2025	11.06.2025
Запуск у «Production»	12.06.2025	12.06.2025

Таблиця 5.2 – Тест-кейс №1

Інформація про тест-кейс	
Ідентифікатор тесту:	Тест-кейс №1
Опис функції:	Створення збору коштів
Власник тесту:	Сорокін Вадим Сергійович
Дата створення:	01.06.2025
Мета тесту:	Перевірити можливість створення збору коштів зареєстрованим користувачем з роллю "волонтер”.
Передумова	

Продовження таблиці 5.2

№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити веб-інтерфейс тестування за допомогою swagger	Користувач має доступ до веб-інтерфейсу	Пройдено
2	Авторизуватись з роллю "волонтер"	Повертається інформацію про волонтера та токен авторизації	Пройдено
Створення збору коштів			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити форму надсилання запиту на створення збору у swagger	Відображається форма з усіма параметрами (назва, опис, сума, посилання на банку)	Пройдено
2	Надіслати запит з валідними даними (JSON body)	Отримано відповідь 201 Created, тіло містить ID нового збору	Пройдено
3	Залишити обов’язкове поле (наприклад, "title") порожнім	Отримано відповідь 422 Unprocessable Entity, описано помилку валідації	Пройдено
4	Надіслати некоректні типи (наприклад, amount="п’ятсот")	Отримано відповідь 422 з описом типізаційної помилки	Пройдено
5	Вказати дату завершення в минулому	Отримано відповідь 400 Bad Request	Пройдено
6	Надіслати зображення як multipart-запит через окремий endpoint	Отримано підтвердження завантаження	Пройдено
7	Перевірити отриманий запис через /funds/{id} (GET)	Дані відповідають надісланим; статус — “Активний”	Пройдено

Кінець таблиці 5.2

Перевірка отримання списку зборів коштів			
№	Опис випадку	Очікуваний результат	
1	Надіслати запит без параметрів на отримання зборів	Отримано 200 OK, список зборів у форматі JSON	Пройдено
2	Використати пошук за частиною назви (search=дрон)	У відповідь повертаються лише збори з відповідними ключовими словами в назві	Пройдено
3	Неавторизований запит (без токена)	Отримано 401 Unauthorized або публічний список (залежно від налаштування доступу)	Пройдено
4	Використати пошук за назви, якої не існує (search=asdf)	Отримано 200 OK, але порожній список	Пройдено
2	Використати пошук за частиною назви (search=дрон)	У відповідь повертаються лише збори з відповідними ключовими словами в назві	Пройдено
3	Неавторизований запит (без токена)	Отримано 401 Unauthorized або публічний список (залежно від налаштування доступу)	Пройдено
4	Використати пошук за назви, якої не існує (search=asdf)	Отримано 200 OK, але порожній список	Пройдено
Результати тестування			
Тестувальник: Сорокін В. С.	Дата прогону тесту: 01.06.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

ВИСНОВКИ

У межах проходження передатестаційної практики було реалізовано серверну частину програмної системи, призначеної для підтримки волонтерської діяльності, зокрема — збору коштів, створення запитів на допомогу та формування звітності щодо виконаних закупівель.

Розробка почалась із вивчення вже наявних платформ, що дозволило виявити основні проблеми в подібних рішеннях. Найбільш поширеними серед них виявились відсутність прозорої звітності, не структурована передача інформації між волонтерами та отримувачами, а також обмеженість у гнучкому керуванні процесами збору. Це дало змогу сформулювати цілісне уявлення про функціональність, яка має бути реалізована в системі. Вони охоплюють ключові сценарії взаємодії користувачів: реєстрацію та автентифікацію, створення і ведення зборів, надсилання запитів на допомогу, додавання звітів із медіафайлами та управління профілями.

У результаті було реалізовано можливість доступу до профілів волонтерів, перегляд зборів та їх поточного стану, можливість донатити через банку. Також з боку отримувачів переглядати та додавати потреби.

Отримане рішення дозволяє автоматизувати частину рутинних процесів, пов’язаних із координацією волонтерських ініціатив, та створює базу для майбутнього розширення системи. У перспективі така платформа може використовуватись як внутрішній інструмент для волонтерських організацій або як частина публічного цифрового сервісу з підтримкою багатьох ролей користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Платформа для благодійних фондів та зборів | Volonter [Електронний ресурс]. Volonter. URL: https://volonter.prom.ua/ (дата звернення: 01.06.2025).
2. Contributors to Wikimedia projects. Hungarian algorithm – Wikipedia [Електронний ресурс] // Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Hungarian_algorithm (дата звернення: 01.06.2025).
3. SciPy [Електронний ресурс]. URL: https://scipy.org/ (дата звернення: 01.06.2025).
4. NumPy [Електронний ресурс]. URL: https://numpy.org/ (дата звернення: 01.06.2025).
5. Kubernetes Documentation [Електронний ресурс]. Kubernetes. URL: https://kubernetes.io/docs/home/ (дата звернення: 01.06.2025).
6. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. O'Reilly Media, Incorporated, 2014. 238 p.
7. Бондарєв В. М., Черепанова Ю. Ю. Комп’ютерна симуляція термодинамічних процесів з навчальними цілями // Наукові праці Вінницького національного технічного університету. 2024. № 2. С. 6–16. DOI: https://doi.org/10.31649/2307-5376-2024-2-6-16.
8. Бондарєв В., Черепанова Ю. Комп’ютерна симуляція небесної механіки з навчальними цілями // Інформаційні системи та технології: матеріали 13-ї Міжнар. наук.-техн. конф. (Харків, 26–28 листоп. 2024 р.). Харків: ХНУРЕ, 2024. URL: https://ist-conf-nure.com.ua/.
9. Посилання на GitHub, де розташовані всі електронні матеріали до кваліфікаційної роботи [Електронний ресурс]. URL: https://github.com/NureSorokinVadym/2025_B_PI_PZPI-21-4_Sorokin_V_S