МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«_____»_____2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА

## (ДИПЛОМНИЙ ПРОЕКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

**Тема:** «Web-застосунок для менеджмента волонтерської діяльності»

**Виконавець:** студент групи УС-412Б Горбасенко Ілля Олегович

**Керівник:** д.т.н., професор Зіатдінов Юрій Кашафович

**Нормоконтролер:** ст. викл. Олександр ШЕВЧЕНКО

**Київ – 2023**

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«____»_____2023 р.

**ЗАВДАННЯ**

**на виконання кваліфікаційної роботи студента**

Горбасенка Іллі Олеговича

(прізвище, ім'я, по батькові)

**1. Тема роботи:** «Web-застосунок для менеджмента волонтерської діяльності» затверджена наказом ректора від 01.05.2023р. № 623/ст.

**2. Термін виконання роботи** з 15.05.2023 р. по 25.06.2023 р.

**3. Вихідні дані до роботи**: Двосторонній застосунок для менеджмента волонтерської діяльності надасть інформацію, як звичайним людям про існуючі фонди та організації з можливістю звернення, так і волонтерам про звернення людей до конкретного фонду, які потребують допомогу.

**4. Зміст пояснювальної записки**: аналіз проблематики розробки застосунку, дослідження конкурентного ринку, ретроспектива використаних технологій та алгоритмів, проектування архітектури веб-додатку, аналіз бізнес-процесів, написання програмного коду, тестування.

**5. Перелік обов'язкового графічного матеріалу**: діаграма життєвого циклу додатку, графічне зображення схеми бази даних, зображення результатів тестування.

**6. Календарний план-графік**

| № п/п | Етапи виконання дипломного проекту | Термін виконання етапів | Примітка |
|---|---|---|---|
| 1. | Проаналізувати літературу та джерела за темою дипломного проекту | 15.05.2023 – 17.05.2023 | |
| 2. | Розроблення та затвердження плану дипломного проекту | 18.05.2023 – 20.05.2023 | |
| 3. | Провести консультації з науковим керівником щодо створення першого розділу | 21.05.2023– 23.05.2023 | |
| 4. | Розробка розділу 1 | 24.05.2023– 02.06.2023 | |
| 5. | Розробка розділу 2 | 03.06.2023– 09.06.2023 | |
| 6. | Розробка розділу 3 | 10.06.2023 – 13.06.2023 | |
| 7. | Висновки та оформлення пояснювальної записки дипломного проекту | 14.06.2023 – 16.05.2023 | |
| 8. | Підписання необхідних документів у встановленому порядку | 17.05.2023 – 19.05.2023 | |

**7. Дата видачі завдання**: « 15 » травня 2023 р.

**Керівник дипломного проекту** Юрій ЗІАТДІНОВ

(підпис керівника) (П.І.Б.)

**Завдання прийняв до виконання** Ілля ГОРБАСЕНКО

(підпис студента) (П.І.Б.)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Web-застосунок для менеджмента волонтерської діяльності» складається зі вступу, трьох розділів, висновку, списку бібліографічних посилань містить 77 сторінок, 29 рисунків, 16 таблиць. Список бібліографічних посилань складається з 7 найменувань.

**Об'єкт дослідження**: проектування застосунку для волонтерів, благодійників та людей, які потребують допомогу

**Предмет дослідження**: Застосунок для менеджемнту волонтерської діяльності.

**Мета роботи:** навчитись робити аналітичні висновки проблематики розробки того чи іншого продукту, дослідити конкурентний ринок. Зробити аналіз використовуваних технологій та алгоритмів. Вдосконалити та застосувати знання з архітектурного проектування програмного забезпечення. Реалізувати додаток для спрощення та оптимізації благодійної діяльності, для покращення координації всередині корпоративних волонтерських організацій та між волонтерами і людьми, які потребують допомогу.

**Результат дослідження**: застосунок який допоможе волонтерам та благодійникам значно оптимізувати процес своєї діяльності, а також налагодити зручну комунікацію з людьми, які потребують допомогу.

ВОЛОНТЕРСЬКА ДІЯЛЬНІСТЬ, ДОПОМОГА, ЗАСТОСУНОК, JAVA, VAADIN, DESIGN PATTERNS, АРХІТЕКТУРНЕ ПРОЕКТУВАННЯ, БАЗА ДАНИХ

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

**API** - Application programming interface (інтерфейс прикладного програмування)

**API JSON** - JavaScript Object Notation (текстовий формат обміну даними)

**CSS** - Cascading Style Sheets (каскадні таблиці стилів)

**DI** - Dependency Injection (ін'єкція залежностей)

**GPS** - Global Position System (система глобального позиціонування)

**HQL** - Hibernate Query Language (Мова запитів Hibernate)

**HTML** - Hyper Text Markup Language (мова розмітки гіпертексту)

**HTTP** - Hypertext Transfer Protocol (протокол передачі гіпертексту)

**HTTPS** - Hypertext Transfer Protocol Secure (захищений протокол передачі гіпертексту)

**IoC** - Inversion of control (інверсія контролю)

**JPA** - Java Persistence

**ORM** - Object Relational Mapping (Реляційне співставлення об'єктів)

**POJO** - Plain Old Java Object (старий добрий Java об'єкт)

**REST** - Representational State Transfer (Передача репрезентативного стану)

**SQL** - Structured Query Language (Структурна мова запитів)

**СУБД** - система управління базами даних

**UI** - User Interface (користувацький інтерфейс)

**UML** - Unified Modeling Language (Мова графічного опису)

# ВСТУП

Волонтерська діяльність є невід'ємною складовою сучасного суспільства, яка забезпечує підтримку та розвиток різних соціальних, культурних та громадських ініціатив. Однак, організація та координація волонтерських проектів часто стикаються з викликами, пов'язаними з управлінням ресурсами, організацією комунікації та забезпеченням ефективності діяльності.

Метою моєї дипломної роботи є проектування та розробка застосунку для менеджменту волонтерської діяльності, який сприятиме покращенню організації та ефективності волонтерських проектів. Основна мета цього проекту - розробка інструменту, що дозволить керівникам волонтерських організацій ефективно керувати ресурсами, координувати дії волонтерів, спрощувати процес планування та моніторингу волонтерських заходів.

Актуальність цього проекту полягає в тому, що волонтерська діяльність є важливим інструментом соціального розвитку, проте її потенціал часто не використовується повністю через недостатню системність управління та координації. Застосунок, розроблений у моїй дипломній роботі, вирішує цю проблему, надаючи зручний та функціональний інструмент для управління волонтерськими проектами.

Подальші аспекти актуальності цього проекту полягають у можливості покращити ефективність волонтерської діяльності, сприяти більш гнучкому плануванню та розподілу завдань, полегшити взаємодію між керівниками та волонтерами, а також забезпечити звітність та моніторинг результативності проектів. Такий застосунок допоможе волонтерським організаціям оптимізувати використання своїх ресурсів, підвищити якість волонтерських послуг та посилити вплив на соціальну сферу.

Отже, цей проект пропонує інноваційний підхід до управління волонтерською діяльністю через розробку застосунку, який вирішує актуальні

проблеми у сфері організації та координації волонтерських проектів. Результати цього дослідження та розробки мають потенціал позитивно вплинути на розвиток волонтерського руху та сприяти його зростанню як значущої сили в суспільстві.

# РОЗДІЛ 1
# ДОСЛІДЖЕННЯ ПРОБЛЕМАТИКИ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ВОЛОНТЕРСЬКОЮ ДІЯЛЬНІСТЮ

## 1.1. Аналіз потреб та оптимізація волонтерської діяльності

Перш ніж розпочати розробку того, чи іншого програмного забезпечення перш за все потрібно зробити аналіз та прийти до повного розуміння об’єкту розробки, які бізнес-задачі буде виконувати продукт та чим дана ініціатива зможе допомогти бізнесу, соціуму тощо.

### 1.1.1. Визначення сутності та ролі волонтерської діяльності

Волонтерська діяльність відноситься до безкоштовних добровільних дій, які здійснюються особами без будь-яких винагород або матеріальних користей. Волонтери надають свій час, навички, ресурси та енергію для допомоги тим, хто її потребує або для покращення соціальної ситуації, навколишнього середовища чи громадського благополуччя.

Сутність волонтерської діяльності полягає в безкорисливій допомозі та підтримці інших людей або спільноти. Вона ґрунтується на принципах добровільності, самовідданості, взаємодопомоги і соціальної відповідальності.

Волонтерська діяльність може приймати різноманітні форми, включаючи роботу в благодійних організаціях, соціальній допомозі, охороні навколишнього середовища, підтримці освіти, культури, спорту тощо.

Роль волонтерської діяльності в суспільстві є надзвичайно важливою. Основні ролі волонтерів включають:

- Підтримка та допомога. Волонтери допомагають людям, які знаходяться у складних життєвих ситуаціях або потребують підтримки. Вони можуть надавати матеріальну допомогу, консультувати, проводити тренінги та навчальні заняття;
- Розвиток спільноти. Волонтери сприяють покращенню життя в спільноті, шляхом налагодження доброзичливих стосунків;
- Свідомий громадянин. Волонтерська діяльність сприяє розвитку свідомості та відповідальності громадян. Волонтери допомагають вирішувати соціальні, екологічні та економічні проблеми у своїй спільноті;
- Розвиток навичок. Волонтерство надає можливість розвивати навички та отримувати новий досвід. Волонтери вчаться комунікації, співпраці, організації заходів та керування проектами;
- Зміна світу. Волонтерська діяльність може мати позитивний вплив на світ. Волонтери працюють над проектами, спрямованими на покращення соціально-економічного прогресу.

### 1.1.2. Виявлення потреб та проблем, що виникають у волонтерській сфері

На основі власної бази знань волонтерської діяльності та опитування було проведено дослідження що головними потребами волонтерів є:

- Управління волонтерською базою даних. Громадські організації часто мають потребу в ефективному управлінні інформацією про волонтерів, таку як контактні дані, навички, доступність та досвід. Програмне забезпечення для управління волонтерською базою даних може допомогти автоматизувати процеси реєстрації, спілкування та координації волонтерів;

- Розкладання завдань і координація. Волонтерські проекти можуть вимагати розподілу завдань між волонтерами та їх планування. Програмне забезпечення для розкладування завдань може сприяти оптимізації процесу, дозволяючи організаторам проектів легко виділити завдання, призначити їх волонтерам та відстежувати прогрес виконання;
- Комунікація та спільне користування інформацією. Ефективна комунікація є ключовою у волонтерській діяльності. Програмне забезпечення для комунікації та спільного користування інформацією, таке як платформи спільної роботи або волонтерські портали, можуть полегшити обмін інформацією, спростити координацію і сприяти більш ефективній комунікації між волонтерами та організаторами проектів;
- Оцінка та звітність. Важливо мати механізми для оцінки продуктивності волонтерів та звітності про виконану роботу. Програмне забезпечення може допомогти створити систему моніторингу звітності та оцінювання.

### 1.1.3. Визначення основних викликів та вимог до ефективного управління волонтерською діяльністю

Перед розробкою будь-якого програмного забезпечення, необхідно визначати переліки вимог та викликів. В першу чергу вимоги допомагають зрозуміти, що саме потрібно досягти за допомогою програмного забезпечення та як це пов'язано з бізнес-цілями суб’єкту або організації. Вони служать посиланням між технічними можливостями і реальними потребами користувачів чи бізнесу. По-друге вимоги допомагають визначити, які функціональність, характеристики повинні бути включені в програмне забезпечення та визначити, які технології, людські ресурси, обладнання та інші ресурси будуть потрібні для реалізації проекту. Тому базуючись на дослідженні потреби (п.1.1.2.) зробимо заключення, що для ефективного управління діяльністю волонтерів необхідно:

- Автоматизація процесів. Розробка програмного забезпечення для управління волонтерською діяльністю передбачає автоматизацію рутинних завдань, таких як реєстрація волонтерів, розкладування завдань, збір даних та звітність. Це дозволяє зменшити ручну працю, забезпечити точність та ефективність процесів управління;
- Інтеграція засобів комунікації. Управління волонтерською діяльністю вимагає ефективної комунікації між волонтерами, організаторами, координаторами проектів та безпосередньо людьми які потребують допомоги. Розробка програмного забезпечення повинна передбачати можливості інтеграції засобів комунікації, таких як електронна пошта, спільні робочі простори або системи обміну повідомленнями, для полегшення обміну інформацією та спілкування;
- Безпека даних. Управління волонтерською діяльністю включає обробку конфіденційної інформації, такої як особисті дані волонтерів. Розробка програмного забезпечення повинна дотримуватись високих стандартів безпеки даних, забезпечувати захист і конфіденційність цих даних, а також дотримуватись відповідних норм і правил, таких як Загальний регламент про захист персональних даних;
- Гнучкість та масштабованість. Розробка програмного забезпечення для управління волонтерською діяльністю повинна бути гнучкою та масштабованою для подальшого вдосконалення.

## 1.2. Аналіз існуючих веб-застосунків для управління волонтерською діяльністю

Також важливим фактором підготовчих процесів до розробки програмного забезпечення є дослідження існуючих аналогів. По-перше це надасть можливість вивчити з найкращими практиками, які вже використовуються в схожих системах. Це може допомогти уникнути повторення помилок та знайти найбільш ефективне рішення. Також є можливість виявлення потенційних вдосконалень,

інновацій та отримати уявлення про конкурентне оточення та сферу, в якій буде працювати застосунок

### 1.2.1. Огляд наявних веб-платформ та додатків, призначених для керування волонтерами

В світі існують такі платформи:

- VolunteerMatch є однією з найбільших веб-платформ для знаходження та керування волонтерами. Вона забезпечує можливість організаціям публікувати вакансії для волонтерів, а також дозволяє волонтерам знаходити проекти, які відповідають їхнім інтересам та навичкам;
- Better Impact є веб-платформою, спеціально розробленою для управління волонтерами. Вона надає інструменти для реєстрації волонтерів, розкладування завдань, комунікації та звітності. Платформа також має можливість інтеграції з іншими системами та розширеними функціями звітності;
- Galaxy Digital пропонує веб-платформу, що дозволяє організаціям управляти волонтерськими проектами та волонтерами. Вона має інструменти для розкладування завдань, спілкування, збору даних та аналізу. Платформа також забезпечує можливість створення власного волонтерського порталу;
- GivePulse є веб-платформою, яка спрощує управління волонтерськими заходами та спільнотами. Вона дозволяє організаціям створювати події, керувати реєстрацією волонтерів, спілкуватися та відстежувати прогрес волонтерської діяльності;
- Track It Forward це платформа для управління добровольчою роботою та годинами внеску у спільноту. Вона надає організаціям та волонтерам зручні інструменти для відстеження, документування та аналізування добровольчих годин і внеску у спільноту.

### 1.2.2. Виявлення переваг та недоліків існуючих веб-застосунків

Визначимо переваги та недоліки розглянутих наявних веб-платформ та додатків, призначених для керування волонтерською діяльністю.

**VolunteerMatch**

Переваги:

- Велика база організацій та волонтерів, що допомагає знайти відповідних партнерів;
- Зручний пошук проектів за категоріями та регіонами;
- Інтуїтивний і простий у використанні інтерфейс.

Недоліки:

- Обмежені функціональні можливості для керування волонтерами поза платформою;
- Відсутність інтеграції з іншими системами управління волонтерами.

**Better Impact**

Переваги:

- Комплексні функції для управління волонтерами, включаючи реєстрацію, розкладування завдань та звітність;
- Можливість інтеграції з іншими системами та налаштування під потреби організації;
- Підтримка клієнтів та надання навчання та підтримки користувачам.

Недоліки:

- Високі витрати на використання платформи, особливо для менших організацій;
- Складність налаштування та впровадження платформи.

**Galaxy Digital**

Переваги:

- Розширені функціональні можливості, включаючи розкладування завдань, спілкування та аналітику;

- Інтеграція з іншими системами та можливість створення власного волонтерського порталу;
- Підтримка і навчання користувачів.

Недоліки:

- Вимагає певного часу та зусиль для оволодіння та використання платформи;
- Можливість перевищення потреб бюджету менших організацій.

**GivePulse**

Переваги:

- Зручний інтерфейс та легке використання, що дозволяє організаціям швидко створювати волонтерські події та залучати учасників;
- Можливість керувати реєстрацією волонтерів, спілкуватися з ними та відстежувати їхній прогрес волонтерської діяльності;
- Інтеграція з платіжними системами для збору пожертв та оплати волонтерських заходів.

Недоліки:

- Обмежені функціональні можливості у порівнянні з іншими волонтерськими платформами, особливо для великих організацій або проектів зі складнішою структурою;
- Відсутність деяких розширених функцій, таких як додаткові засоби аналітики або розкладання завдань.

**Track It Forward**

Переваги:

- Повний цикл керування волонтерами, включаючи реєстрацію, розкладання завдань, комунікацію та звітність;
- Зручний інтерфейс для волонтерів, що дозволяє легко відстежувати інформацію про свою діяльність та години волонтерства;
- Розширені функції звітності та аналітики для організацій, що допомагають в оцінці та вдосконаленні волонтерських програм.

Недоліки:

- Може вимагати певного часу та зусиль для оволодіння та використання всіх функцій платформи;
- Обмежена інтеграція з іншими системами, що може ускладнити використання платформи для деяких організацій.

### 1.3. Поняття веб-застосунку

Веб-застосунок - це програмне забезпечення, яке працює через веб-браузер або мобільний пристрій та надає користувачам функціональні можливості через Інтернет. Веб додатки складаються з таких компонентів як клієнт(фронтенд), сервер(бекенд) та спосіб комунікації між компонентами у вигляді протоколів обміну даними(HTTP, HTTPS). Але в цілому компоненти можуть співіснувати один без одного. Клієнтський компонент (фронтенд) веб-застосунку використовує веб-технології, такі як HTML, CSS і JavaScript, для створення інтерфейсу та взаємодії з користувачем. В свою чергу серверний компонент можуть застосовувати серверні технології такі як високорівневі мови програмування (Java, C++, C#), бази даних(MySQL, Postgresql, Mongodb) тощо.

Веб-застосунки можуть мати різноманітний функціонал, включаючи створення, редагування та видалення даних, виконання обчислень, комунікацію з іншими користувачами, обробку платежів, управління контентом тощо. Вони можуть бути доступні як через веб-браузер на комп'ютері, так і на мобільних пристроях через спеціальні мобільні версії або мобільні додатки.

### 1.4. Переваги та недоліки веб-застосунків

Переваги веб-застосунків:

- Доступність. Веб-застосунки можна використовувати з будь-якого пристрою з підключенням до Інтернету та веб-браузером, що робить їх доступними для широкого кола користувачів;

- Кросплатформенність. Веб-застосунки працюють на різних операційних системах (Windows, macOS, Linux) без необхідності розробки окремих версій для кожної платформи;
- Оновлення. Оновлення та вдосконалення веб-застосунків можна робити централізовано на сервері, тому користувачам не потрібно встановлювати нові версії окремо;
- Низькі вимоги до ресурсів. Веб-застосунки не потребують значних обчислювальних потужностей на стороні користувача, оскільки більшість обчислень відбувається на сервері;
- Широкий доступ до інформації. Веб-застосунки можуть звертатися до великих обсягів даних та інформації, що доступна через Інтернет;

Недоліки веб-застосунків:

- Залежність від Інтернет-з'єднання. Веб-застосунки потребують постійного підключення до Інтернету для своєї роботи. У випадку відсутності зв'язку або повільного з'єднання користувачам може бути обмежений або недоступний доступ до функціоналу;
- Обмежені можливості взаємодії з операційною системою. Веб-застосункам часто не вдається повною мірою використовувати функції та ресурси операційної системи пристрою, такі як камера, мікрофон, GPS або специфічні API. Це може обмежити можливості деяких застосунків;
- Нижча продуктивність порівняно з нативними застосунками. У порівнянні з нативними додатками, розробленими спеціально для певної платформи, веб-застосунки можуть мати меншу продуктивність. Вони можуть бути повільнішими та менш ефективними у використанні ресурсів пристрою;
- Обмежена доступність офлайн. Веб-застосунки зазвичай потребують постійного з'єднання з Інтернетом, і функціонал може бути обмеженим або недоступним у випадку відсутності мережі. Це може бути незручним для користувачів, які потребують доступу до даних та функцій без з'єднання;

- Обмежені можливості доступу до пристроєвого обладнання. Веб-застосунки не мають повного контролю над пристроєвим обладнанням, таким як сканери, принтери або USB-пристрої. Це може ускладнити інтеграцію зовнішніх пристроїв та обмежити функціонал деяких застосунків.

### 1.5. Висновки до розділу 1

Розділ "Дослідження проблематики створення веб-застосунку для ефективного управління волонтерською діяльністю" розкриває різні аспекти та виклики, пов'язані з розробкою веб-застосунку для управління волонтерською діяльністю. Це дослідження виявило декілька ключових висновків, які можуть бути узагальнені і використані для подальшого розвитку теми.

В першу чергу, було виявлено, що волонтерська діяльність має свої особливості, що потребують спеціального підходу до менеджменту. Визначено, що головними потребами є координація волонтерів, виявлення та вирішення рутинних проблем у волонтерській сфері (реєстрація, облік, звітність тощо), а також забезпечення ефективного спілкування та зв'язку між волонтерами, організаціями та в першу чергу людьми, які потребують допомоги.

Дослідження також показало, що веб-застосунки можуть бути потужним інструментом для оптимізації процесу управління волонтерською діяльністю. Вони забезпечують зручний доступ до інформації про волонтерські можливості, дозволяють організувати та відстежувати роботу волонтерів, а також сприяють покращенню комунікації та обміну даними між волонтерами та організаціями.

Проте, дослідження також виявило певні недоліки веб-застосунків, зокрема залежність від Інтернет-з'єднання, обмежені можливості взаємодії з операційною системою та обмежена доступність офлайн. Ці аспекти потребують уваги та пошуку рішень для забезпечення найкращого використання веб-застосунків у волонтерів.

# РОЗДІЛ 2
# РЕТРОСПЕКТИВА ВИКОРИСТАНИХ ТЕХНОЛОГІЙ, ПІДХОДІВ, МЕТОДІВ ТА АЛГОРИТМІВ СТВОРЕННЯ WEB-ЗАСТОСУНКУ ДЛЯ МЕНЕДЖМЕНТУ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

## 2.1. Огляд архітектурної моделі для веб-застосунку

Після ознайомлення з предметом розробки, узгодження основного переліку вимог та визначення типу програмного забезпечення, в нашому випадку це веб-застосунок, наступним етапом розробки є розгляд доступних архітектурних моделей для данного типу ПЗ. На сьогоднішній день представлено декілька видів архітектурних моделей для веб-застосунку:

Монолітна архітектура - це традиційний підхід, в якому весь веб-застосунок розгортається як єдиний великий модуль. Усі компоненти, такі як серверна логіка, база даних, інтерфейс користувача і логіка бізнес-процесів, розташовуються в одному монолітному додатку.

Клієнт-серверна архітектура - ця архітектура коротко згадувалась в попередньому розділі, вона розділяє функції веб-застосунку між клієнтською (фронтенд) та серверною (бекенд) сторонами. Клієнтська частина зазвичай включає фронтенд-код, такий як HTML, CSS і JavaScript, в той час як серверна частина відповідає за бізнес-логіку, зберігання даних і взаємодію з базою даних.

Мікросервісна архітектура - в цій архітектурі веб-застосунок розбивається на невеликі автономні сервіси, які працюють разом і виконують конкретні функції. Кожен сервіс може бути розроблений та розгорнутий окремо, використовуючи мову програмування та технології, найкраще підходять для конкретної задачі.

Мікросервісна архітектура - в цій архітектурі веб-застосунок розбивається на невеликі автономні сервіси, які працюють разом і виконують конкретні функції. Кожен сервіс може бути розроблений та розгорнутий окремо, використовуючи мову програмування та технології, найкраще підходять для конкретної задачі.

Сервер без стану архітектура - у цій архітектурі сервер не зберігає інформацію про попередні запити користувача. Кожен запит обробляється незалежно від попередніх запитів, що дозволяє легко масштабувати застосунок

В нашому випадку буде застосовуватись монолітна архітектура на базі клієнт-серверної моделі поведінки. Такі змішані архітектурні рішення часто використовуються при застосування різноманітних бібліотек та фреймворків, що дозволяють вирішувати питання серверної обробки бізнес логіки, створення інтерфейсу та комунікації між ними в рамках однієї мови програмування. Більш детально ми розберемо саме клієнт-серверну, тому що за основу взятий саме паттерн поведінки клієнт-серверу. А моноліт тільки покриває випадок розташування модулів додатку. Модуль клієнтської частини(інтерфейсу) та модуль обробки бізнес-логіки(сервер, база даних) розташовані в одному проекті.

### 2.1.1. Клієнт-серверна архітектура

Клієнт-серверна архітектурна модель є однією з найпоширеніших моделей для побудови розподілених систем, включаючи веб-застосунки. У цій моделі функціональність системи розподілена між двома основними компонентами: клієнтом і сервером.

Клієнт - це кінцевий користувач або програма, яка звертається до сервера для отримання певних послуг або доступу до ресурсів. Клієнт може бути веб-браузером, мобільним додатком, настільною програмою або будь-яким іншим пристроєм або програмою, що взаємодіє з сервером.

Сервер - це центральна складова системи, яка надає послуги, обробляє

запити та забезпечує доступ до ресурсів. Веб-сервери, бази даних, файлові сервери та інші сервери можуть бути використані для реалізації функціональності серверної частини.

В клієнт-серверній моделі взаємодія між клієнтом і сервером відбувається через мережу, зазвичай за допомогою протоколу HTTP. Клієнт ініціює запит до сервера, передаючи необхідні дані, і сервер обробляє запит, генерує відповідь і надсилає її назад до клієнта. Модель клієнт-серверної архітектури наведений на рис. 2.1.

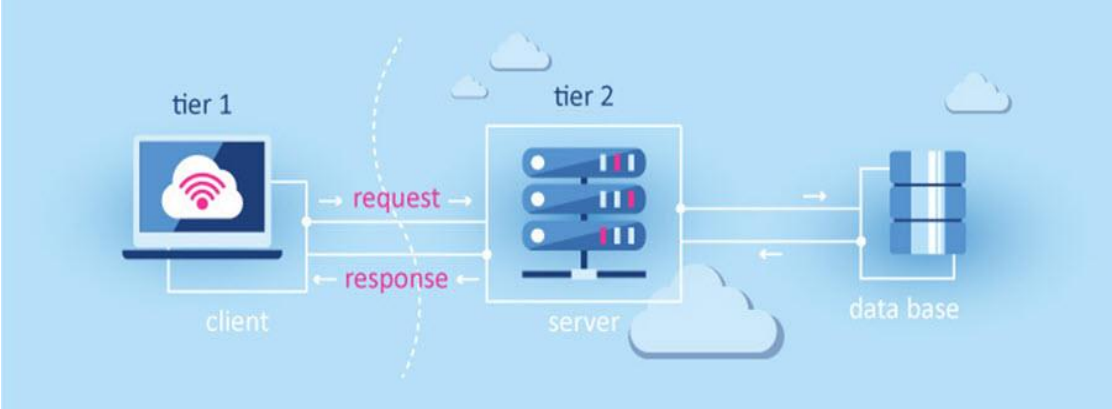


Рис. 2.1. Модель клієнт-серверної архітектури

Переваги клієнт-серверної архітектурної моделі:

- Розділення обов'язків. Клієнт і сервер виконують різні функції, що дозволяє розділити відповідальності між ними. Клієнт забезпечує взаємодію з користувачем та відображення інтерфейсу, тоді як сервер виконує обробку запитів, бізнес-логіку та зберігає дані. Це спрощує розробку, тестування і підтримку системи;
- Масштабованість. Клієнт-серверна модель дозволяє гнучко масштабувати систему. Можливе горизонтальне масштабування, де можна додавати додаткові сервери для розподілу навантаження, а також вертикальне масштабування, де можна збільшувати ресурси сервера для підвищення продуктивності;

- Більша безпека. Клієнт-серверна модель дозволяє забезпечити вищий рівень безпеки. Сервер може зберігати та керувати доступом до конфіденційних даних, а також встановлювати правила та обмеження доступу з боку клієнтів. Це робить систему менш вразливою для зловмисників.

Недоліки клієнт-серверної архітектурної моделі:

- Оверхед мережі. Комунікація між клієнтом і сервером відбувається через мережу, що може призвести до збільшення трафіку і затримок у передачі даних. Велика кількість запитів з боку клієнтів може перевантажити мережу і уповільнити відповідь системи;
- Одинична точка відмови. Сервер є центральною складовою системи, і якщо він відмовляє, це може призвести до недоступності всієї системи. У разі відмови сервера, клієнти можуть бути приречені на відсутність доступу до ресурсів і послуг;
- Складність масштабування. При зростанні обсягу користувачів і навантаження на систему можуть виникати складнощі з масштабуванням серверів. Додавання нових серверів та їх налагодження може бути складним процесом, особливо якщо система розподілена по різних локаціях;
- Залежність від мережевого з'єднання. Для взаємодії з сервером клієнтам необхідне активне мережеве з'єднання. Відсутність мережевого з'єднання або його нестабільність можуть призвести до втрати з'єднання з сервером і недоступності послуг;
- Складність розподіленої обробки даних. У деяких випадках, особливо при роботі з розподіленими даними, обробка та синхронізація даних між клієнтами і серверами можуть становити складність. Це може потребувати додаткової логіки та зусиль для забезпечення цілісності та консистентності даних.

## 2.2. Огляд фундаментальних використаних технологій

Після детального розгляду та вибору архітектурної моделі, визначення функціональних вимог на базі цього можна сформувати перелік технологій, в яких ми маємо певний рівень експертизи та вони будуть задовольняти визначенні функціональні потреби. В першу чергу ми обираємо мову програмування, яка дозволить реалізувати визначений функціонал. Далі до цього треба додати ті чи інші фреймворки, які допоможуть спростити, прискорити та оптимізувати розробку. Також потрібно визначитись з необхідністю використання бази даних, в нашому випадку це буде більш ніж доцільно. Також при підборі використовуваних технологій необхідно враховувати щоб вони між собою були сумісні, масштабовані з можливістю інтеграцій нових інструментів**.**

### 2.2.1. Java. Об’єктно орієнтована мова програмування

Java є об'єктно-орієнтованою мовою програмування, розробленою компанією Sun Microsystems (зараз є власністю Oracle Corporation). Вона була представлена у 1995 році і швидко стала однією з найпопулярніших мов програмування завдяки своїй простоті, надійності та переносимості.

Основні принципи об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція і абстракція, є ключовими складовими Java. Це дозволить нам створити додаток, орієнтований на об'єкти, що сприяє модульності, підтримці кодування, повторному використанню і підтримці розширюваності.

Основні переваги Java включають:

- Переносимість. Java є платформонезалежною мовою, що означає, що програми, написані на Java, можуть запускатися на різних операційних системах без потреби у перекомпіляції. Це робить Java привабливим вибором для розробки кросплатформеного програмного забезпечення;

- Безпека. Java використовує механізми безпеки, такі як сандбокс і контроль доступу, що дозволяють обмежувати дії програм та запобігати вразливостям. Це особливо важливо для розробки веб-додатків із запобіганням небезпечним операціям;
- Велика стандартна бібліотека. Java має широку стандартну бібліотеку, яка надає готові рішення для багатьох типових задач, таких як робота з рядками, мережеві операції, операції з базами даних, графічний інтерфейс і багато іншого. Це дозволяє розробникам прискорити процес розробки;

### 2.2.2. Spring Framework. Java платформа побудови веб-застосунків

Spring Framework - це відкрита платформа для розробки програмного забезпечення на мові Java. Вона надає розширення та інфраструктуру для ефективної розробки Java-додатків. Spring пропонує комплексний набір інструментів та бібліотек для різних аспектів програмування, включаючи управління залежностями, управління транзакціями, роботу з базами даних, веб-розробку та багато іншого, а також дозволяє розбити додаток на модулі, що сприяє покращенню організації та керуванню кодом. Модулі можуть бути взаємозамінними та можуть бути легко перевикористані в інших частинах програми.

Одним з основних функціональних принципів Spring є Інверсія управління (Inversion of Control, IoC). Spring використовує принцип IoC, що означає, що контроль над об'єктами та їхнім життєвим циклом передається контейнеру Spring. Це дозволяє зменшити залежності між компонентами та полегшує тестування та обслуговування додатків. Принцип Інверсії управління наведений на рис. 2.2.

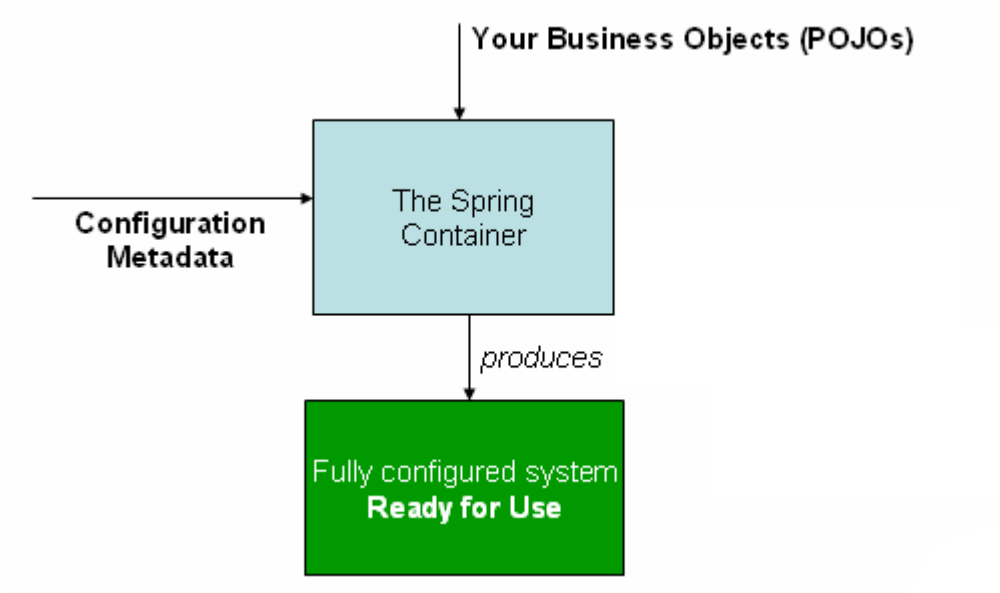


Рис. 2.2. Діаграми роботи принципу Інверсії управління

Наступною фундаментальною функціональною можливістю Spring - це Внедрення залежностей (Dependency Injection, DI) які дозволяють внедрювати залежності між компонентами без прямих посилань на них. Це полегшує розширення та зміну функціональності без зміни коду компонентів.

Даний фреймворк наділений можливістю аспектно-орієнтованого програмування, що дозволяє вивести загальну функціональність, таку як логування, транзакції та безпека, в окремі аспекти. Це спрощує підтримку додатків з розподіленими вимогами та дозволяє уникнути повторення коду.

Окрім нативних інструментів, даний фреймворк складає в собі ще безліч інструментів для підтримки різноманітних сторонніх технологій. До таких інструментів відносяться:

- Spring Data JPA - надає простий і зручний спосіб взаємодії з реляційними базами даних за допомогою об'єктно-реляційного відображення (ORM). JPA (Java Persistence API) є стандартом, який визначає способи роботи з об'єктами в контексті реляційної бази даних. Spring Data JPA надає реалізацію цього стандарту для роботи з базами даних в Spring-додатках;
- Spring Security - забезпечує механізми аутентифікації (перевірки ідентичності користувача) та авторизації (контролю доступу до ресурсів) веб-

додатків на основі різних видів ідентифікаційних даних, таких як логіни/паролі, токени, сертифікати тощо, надає гнучкі конфігураційні можливості, які дозволяють легко налаштувати правила доступу до ресурсів на основі ролей, прав, або власних критеріїв безпеки. Він підтримує різні механізми аутентифікації, такі як форма входу, HTTP Basic або Digest аутентифікація, аутентифікація на основі токенів, OAuth і багато інших;

- Spring MVC - надає шаблонний підхід до розробки веб-додатків на основі патерну проектування Model-View-Controller. У Spring MVC модель відповідає за бізнес-логіку додатку та управління даними. Представлення (View) відображає дані моделі користувачу та приймає від нього вхідні дані. Контролер (Controller) обробляє вхідні запити від користувача, взаємодіє з моделлю для отримання або оновлення даних та передає ці дані відповідному представленню для відображення результатів користувачу.

Важливим фактом є те, що Spring надає простий та зрозумілий API для розробки додатків на мові Java. Його можна використовувати як для невеликих проектів, так і для великих і складних додатків. Його можна легко розширювати та налаштовувати для відповідності потребам конкретного додатку. У загальному, Spring Framework - це потужний та гнучкий інструмент для розробки додатків на мові Java, який надає розробникам багато функціональних можливостей та спрощує розробку, тестування та обслуговування додатків.

### 2.2.3. Vaadin Framework. Java фреймворк для побудови інтерактивних веб-інтерфейсів

Vaadin - так як і Spring є відкритим програмним забезпеченням та фреймворком для розробки веб-додатків з використанням мови Java. Він надає засоби для створення багаторівневих додатків користувацького інтерфейсу, які повністю працюють з боку сервера, тобто весь код виконується на сервері, а не в браузері клієнта. Життевий цикл Vaadin наведений на рис. 2.3.

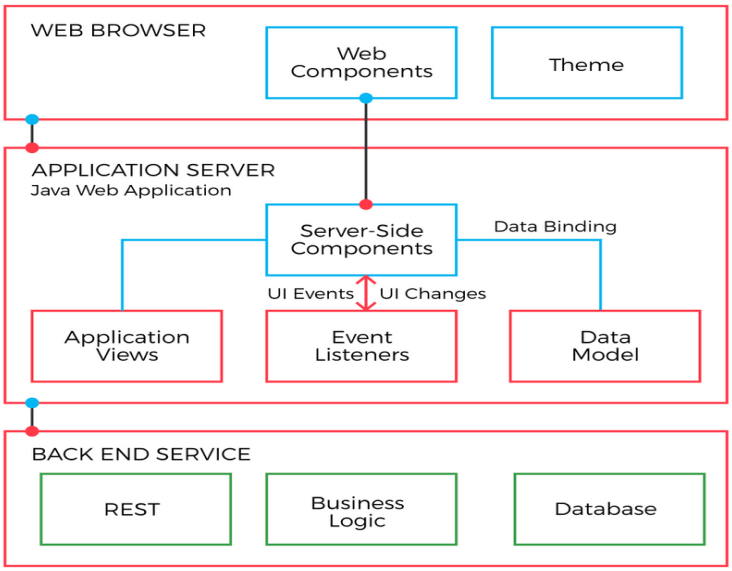


Рис. 2.3. Життєвий цикл роботи Vaadin

Vaadin побудований з використанням модульної архітектури. Він складається з різних модулів, які можуть бути використані за потреби. Це дозволяє розробникам використовувати лише ті функціональність та компоненти, які їм необхідні, що сприяє ефективному використанню ресурсів та зменшенню обсягу коду.

До основних функціональних можливостей Vaadin входить те, що він базується на мові Java, що дозволяє розробникам використовувати всю потужність та екосистему Java для створення веб-додатків. Ми можемо використовувати Java-фреймворки та бібліотеки для розширення можливостей своїх додатків. Vaadin надає багато готових веб-компонентів, таких як кнопки, тексти, таблиці, форми тощо. Вони включають в себе як клієнтський код (JavaScript та HTML), так і серверний код (Java). Є можливість легко налаштовувати та розширювати ці компоненти за допомогою Java та найголовніше, що Vaadin автоматично генерує та оновлює відповідний клієнтський код для взаємодії з компонентами у браузері. Усі компоненти фреймворку можуть спокійно кастомізуватись під себе.

Vaadin також вирішує проблеми адаптивного дизайну, що дозволяє створювати додатки, які працюють і виглядають добре на різних пристроях і розмірах екранів. Правила адаптації також можна використовувати або нативні,

або підлаштовувати під себе.

Фреймворк наділений високою продуктивністю, так як пропонує механізми кешування та оптимізації, що сприяють швидкому відгуку додатків. Він використовує мінімум обміну даними між клієнтом та сервером, що зменшує навантаження на мережу та покращує продуктивність.

Також важливим аспектом розширюваності є можливість інтеграції, Vaadin може бути легко інтегрований з іншими технологіями та фреймворками Java. Він підтримує інтеграцію з серверними технологіями, такими як Java EE та Spring, а також з фронтенд фреймворками, такими як React або Angular.

### 2.2.4. Hibernate ORM Framework. Java фреймворк роботи з базами даних

ORM (Object-Relational Mapping) - це технологія програмування, яка дозволяє взаємодіяти з базами даних, моделюючи дані бази даних у вигляді об'єктів в об'єктно-орієнтованому програмуванні.

Традиційні бази даних використовують реляційну модель, де дані зберігаються у вигляді таблиць зі структурованими рядками та колонками. З іншого боку, об'єктно-орієнтоване програмування оперує об'єктами, які мають свої властивості та методи.

ORM служить посередником між об'єктно-орієнтованим світом та реляційною базою даних, автоматично перетворюючи дані між ними. За допомогою ORM, розробники можуть використовувати об'єктно-орієнтований підхід до роботи з базою даних, ігноруючи складність SQL-запитів та взаємодії з таблицями та стовпцями.

ORM-фреймворки, такі як Hibernate (який був згаданий раніше), дозволяють розробникам визначати взаємозв'язки між об'єктами, анотувати класи та поля для відображення на таблиці бази даних і автоматично генерувати SQL-запити для взаємодії з базою даних. ORM також надає можливість виконувати операції над даними, такі як збереження, оновлення, видалення та

вибірка, використовуючи об'єктно-орієнтований синтаксис, замість написання складних SQL-запитів. Це спрощує розробку та підтримку програмного забезпечення, оскільки дозволяє розробникам фокусуватись на бізнес-логіці додатку, а не на деталях роботи з базою даних.

Hibernate є одним з найпопулярніших фреймворків для об'єктно-реляційного відображення (ORM) у світі Java. Він надає зручний спосіб взаємодії з базами даних, дозволяючи розробникам працювати з об'єктами Java, а не з SQL-запитами безпосередньо. Архітектурна модель Hibernate наведена на рис. 2.4.

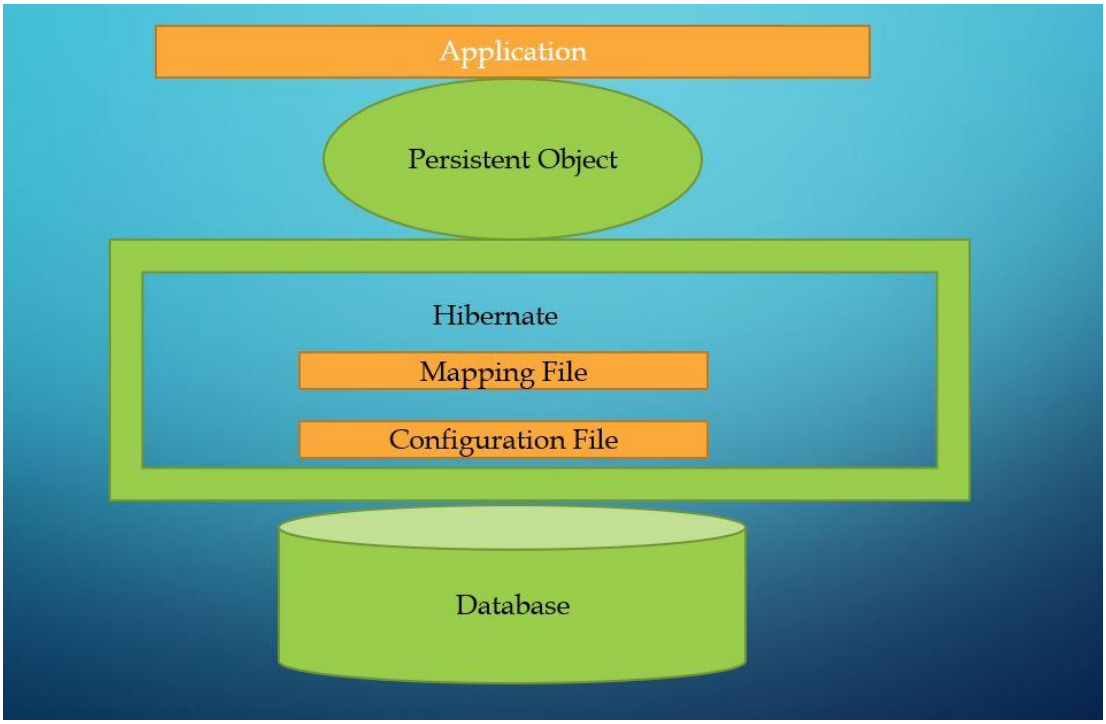


Рис. 2.4. Архітектура Hibernate

Функціональні можливості Hibernate ORM Framework:

- Об'єктно-реляційне відображення - Hibernate дозволяє відображати об'єкти Java безпосередньо на таблиці бази даних. Він автоматично вирішує проблеми відповідності між об'єктною моделлю та схемою бази даних, забезпечуючи зручний спосіб доступу до даних;
- Підтримка різних баз даних - Hibernate підтримує багато різних баз даних, включаючи MySQL, PostgreSQL, Oracle, Microsoft SQL Server та інші. Це

дозволяє розробникам використовувати Hibernate в різних проектах, незалежно від використовуваної бази даних;

- Мова запитів Hibernate Query Language (HQL) - Hibernate надає власну мову запитів, відому як HQL, яка є об'єктно-орієнтованою та незалежною від бази даних. HQL дозволяє виразно виразити запити до бази даних, використовуючи об'єктну модель додатку;
- Підтримка транзакцій та кешування - Hibernate забезпечує механізми для керування транзакціями та кешуванням даних. Це дозволяє забезпечити цілісність даних та поліпшити продуктивність додатків шляхом ефективного використання кешування;
- Лінива ініціалізація - Hibernate підтримує ліниву ініціалізацію. Лінива ініціалізація є концепцією в програмуванні, в якій об'єкти або дані завантажуються або ініціалізуються лише тоді, коли вони фактично потрібні для виконання певної операції. Це означає, що об'єкт або дані не завантажуються зразу при завантаженні або створенні об'єкту, а відкладаються до моменту, коли вони потрібні, що досить сильно оптимізує процес виконання запитів у базу даних.

### 2.2.5. Postgres SQL. База даних.

База даних - є організованою колекцією даних, які зберігаються та управляються комп'ютерною системою. Вона є основним компонентом інформаційної системи, який дозволяє зберігати, організовувати та керувати даними.

База даних може включати різноманітні типи інформації, такі як текстові дані, числа, зображення, звуки тощо. Вона дозволяє структурувати дані за допомогою таблиць, полів та зв'язків між ними, що дозволяє ефективно організовувати та здійснювати доступ до інформації.

Після того, як бази даних набули великого спросу в користування була розроблена система управління базами даних, а саме СУБД. Вона була створена

з метою полегшення та покращення роботи з великими обсягами даних. Основна ідея полягає у тому, щоб забезпечити ефективне зберігання, організацію, керування та доступ до даних.

СУБД (Система управління базами даних) - це програмне забезпечення, яке дозволяє створювати, керувати та оптимізувати бази даних. Вона забезпечує інтерфейс між користувачами, додатками та самою базою даних, дозволяючи виконувати операції збереження, оновлення та отримання даних. До списку можливостей СУБД також входить створення структури бази даних, включаючи таблиці, поля, зв'язки та інші об'єкти. Вона також забезпечує можливість оновлювати цю структуру, додавати або видаляти таблиці та модифікувати поля. СУБД надає мову запитів, яка дозволяє вибирати, фільтрувати та сортувати дані в базі даних. Найпоширенішою мовою запитів є SQL (Structured Query Language). Дана система підтримує концепцію транзакцій, що дозволяє групувати операції над даними у логічні одиниці. Вона забезпечує атомарність, цілісність, стійкість та консистентність даних під час виконання транзакцій.

PostgreSQL - це потужна та розширювана система керування базами даних, яка підтримує реляційну модель даних. Вона була розроблена як відкрите програмне забезпечення і має широкий набір функцій та можливостей, що робить її однією з найпопулярніших СУБД на сьогоднішній день.

Окрім вище перелічених стандартних можливостей СУБД, PostgreSQL має в собі декілька особливостей.

PostgreSQL дозволяє розширювати функціональність за допомогою власних розширень, включаючи власні типи даних, функції, оператори та інші об'єкти бази даних. Надає різні механізми реплікації для створення резервних копій та розподіленого доступу до даних. Він також підтримує кластеризацію для розподіленої обробки даних та збільшення продуктивності. Має підтримку для роботи з географічними та геологічними даними, зокрема, засоби для обробки географічних запитів та розширення, такі як PostGIS. PostgreSQL надає можливості повнотекстового пошуку, що дозволяє виконувати ефективний пошук текстових даних за допомогою повнотекстових індексів та запитів. Має

вбудовану підтримку для роботи з даними у форматі JSON. Ви можете зберігати, опрацьовувати та запитувати дані JSON безпосередньо в базі даних. І однією з основних особливостей PostgreSQL є наявність власної мови програмування під назвою PL/pgSQL. PL/pgSQL є блок-структурованою мовою програмування, яка розширює стандартний SQL в PostgreSQL. Вона надає можливість визначати функції, процедури, тригери та інші об'єкти бази даних.

В результаті вищевказаного аналізу, можна зробити висновок що в цілому, PostgreSQL є потужною та надійною системою керування базами даних з багатьма розширеними можливостями та гнучкістю. Він є популярним вибором для проектів різного розміру та складності.

### 2.3. Висновки до розділу 2

На основі розглянутих вище технологій, підходів, методів та алгоритмів, можна зробити висновок, що найбільш доцільним буде використання мови програмування Java, по-перше через достатній рівень експертизи, по-друге через її простоту, надійність та наповненість. Java дозволить нам застосувати принципи об’єктно-орієнтованого програмування, що сприятиме легкому масштабуванню, розширюванності, потворному використанню коду та модульності. Для полегшення та прискорення розробки допоможуть такі обрані фреймворки та бібліотеки як Spring Framework, Vaadin Framework та Hibernate Framework. Spring дозволить нам налагодити екосистему модульних компонентів та налаштувати без труднощів між ними залежності. Деякі з компонентів Spring Framework та Hibernate Framework надасть можливість працювати з базою даних, її таблицями як з Java об’єктами, классами та без особливих зусиль виконувати запити в базу даних використовуючи вже вбудовану в Hibernate об’єктно-орієнтовану мову запитів HQL. Vaadin Framework допоможе в реалізації користувацького інтерфейсу без жодного використання веб-технологій та мов клієнтського інтерфейсу такі як HTML, CSS, JavaScript. В якості сховища для даних, буде використана одна з

найпопулярніших СУБД - це Postgresql. Вона надасть можливість створювати, керувати та оптимізувати нашу базу даних, забезпечить інтерфейс між користувачами, додатком та самою базою даних, дозволивши виконувати операції збереження, оновлення та отримання даних.

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ МЕНЕДЖМЕНТУ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

## 3.1. Визначення переліку функціоналу та опис бізнес-процесів

Після дослідження проблематики реалізації даного продукту, аналізу потреб, існуючих застосунків даної предметної області, а також після визначення списку технологій, який буде використаний в розробці можна переходити до етапу самої розробки. На основі сформованої бази знань, перш за все необхідно сформувати перелік бізнес-процесів. Сам термін бізнес-процесу, регламентує себе, як послідовність взаємопов'язаних дій, які повторюються та мають логічну структуру, спрямованих на використання ресурсів підприємства з цілю обробки об'єкта та досягнення вимірних результатів. Основною метою формування переліку бізнес-процесів є:

- Порядок дій;
- З'ясувати виконавців або інструменти виконання того чи іншого процесу;
- Що буде результатом процесу;
- Надійність процесу;
- Чи можем процес бути модифікований або розширений у майбутньому.

Перш за все архітектуру наших процесів потрібно розділити на три умовні системно-модульні складові – корпоративна складова, клієнтська складова та загальна. Кожна з складових може нілічувати в собі певні об'єкти та підоб'єкти з переліком функціоналу пов'язаного з ними.

**Корпоративна складова** – це функціональна складова веб-застосунку для авторизованих користувачів, призначена для управління та координації волонтерської діяльності в корпоративному середовищі. Вона надає можливості організації або фонду контролювати, або спрощувати процеси пов'язані з волонтерами та клієнтами. Об`єкти корпоративної складової та перелік функціоналу з описом бізнес процесів:

- Об'єкт – Користувач. Перелік функціоналу:
  - Авторизація користувача. Авторизація перш за все необхідна для ідентифікації користувачів. Вона дозволяє встановити ідентичність користувача, переконатись, що він має відповідні дозволи на доступ до системи та встановити його рівень привілеїв. Також авторизація допоможе забезпечити безпеку даних та інформації, що зберігаються в системі та дозволить контролювати доступ користувачів до конфіденційної та приватної інформації, а також встановити права доступу в залежності від ролі користувача;
  - Реєстрація користувача. Реєстрація в свою чергу дозволить зберегти інформацію про користувача, таку як контактні дані, облікові дані і тд;
  - Підтвердження користувача через пошту – після реєстрації необхідна реалізація верифікації користувача через пошту. Це дасть гарант системі, що користувач який реєструється не є ботом та допоможе запобігти зловмисних хакерських та ddos-атак;
  - Авторизація через стороні сервіси(Google). Механізм, який дозволяє користувачам використовувати свої облікові записи Google для авторизації та входу в систему. Замість створення окремого облікового запису в системі, ми можемо надати користувачу можливість використати дані для входу в гугл для доступу до сервісу. Ми живемо в такий час, що майже у кожної людини від дитини до пенсіонера є акаунт такого сервісу як, Google. Також сервіс Google користується світовою довірою та надає високий рівень безпеки для своїх облікових записів, використовуючи різні механізми аутентифікації та захисту.

Даний функціонал вважається опціональним, проте створює певну зручність, коли не потрібно запам’ятовувати дані для входу в окремий сервіс, а використати вже існуючі;

- Відновлення облікових даних. Процес за допомогою якого, користувач може отримати доступ до свого обліокового запису, якщо він забув свої облікові дані. Ця процедура дозволяє змінити старі облікові дані на нові у випадку втрати та надасть можливість додаткового захисту від несанкціонованого доступу;
- Оновлення та перегляд персональних даних. В будь якій системі, дані облікових записів якої будуть десь використовуватись інформаційно, обов’язково повина бути можливість продивитись профіль зі своїми даними, та в разі потреби оновити їх. Це позбавить вас проблеми в разі якщо інформація про вас застаріла, або була допущена помилка – реєструватись заново.

❖ Об‘єкт - Фонд(волонтерська організація). Перелік функіоналу:

- Реєстрація фонду. Щоб звернутись до певного волонтерського фонду. Він, як мінімум, повинен існувати. Тому в системі повинна бути можливість зареєструвати фонд. Вже зареєстрований волонтер, повинен мати змогу створити свій власний фонд, додавши до нього назву, опис, категорії діяльності, контактну інформацію, адресу та за бажанням реквізити одні або більше. Далі, люди які не зареєстровані в системі зможуть побачити його та за потребою звернутись по допомогу, або за бажанням зробити благодійний внесок, за вказаними реквізитами в описі фонду;
- Оновлення та перегляд даних фонду. Цей функціонал допоможе редагувати дані, вже створеного фонду будь якому члену команди, позбавивши проблеми в разі якщо інформація про фонд застаріла, або була допущена помилка – реєструвати заново;
- Запрошення/заявки на участь у фонді. Рідко коли в волонтерських організаціях працює одна людина. Тому у людини, яка тільки

зареєструвалась повинна бути можливість або створити свій власний фонд, знайти за назвою існуючий в системі та надіслати до нього заявку на участь, або прийняти запрошення від фонду. В свою чергу члени команди того чи іншого фонду можуть запрошувати у команду зареєстрованих користувачів, які ще не є учасниками жодної з організацій або приймати заявки на участь від інших користувачів;

- Редагування команди волонтерської організації. Важливою складовою інфраструктури волонтерської організації є можливість внесення коректівів в уже існуючу команду Тобто, або запрошувати/приймати нових учасників, що було вже описано, або забирати з команди вже існуючих. Керівник фонду в разі потреби, повинен мати змогу видалити учасника команди, або сам учасник міг покинути організацію за бажанням. Це гарантує підтримання екосистеми організації на належному рівні.

- Об’єкт - Заявка на допомогу. Перелік функціоналу:
  - Отримання заявок на допомогу. В кожній вже зареєстрованій волонтерській організації обов’язково повинна бути можливість отримувати та переглядати заявки на допомогу. Тобто будь-який член команди, може перейти в пункт меню «Заявки» та продивитись їх. Кожна заявка має містити в собі ідентифікаційний номер, статус, дату створення, деталі та контактні дані. Весь цей перелік інформації надасть повний обсяг розуміння про потребу та вчасно та ефективно зкоординувати на неї;
  - Зміна статусів заявок та сповіщення клієнтів. Для ефективної комунікаці з клієнтами, необхідна можливість зміни статусів заявок. Та після зміни статусів автоматичнее емейл сповіщення, за контактними даними, які лишив клієнт. Тобто будь-який член команди може перейти в пункт меню «Заявки» та змінити статус .

**Клієнтська складова** – це складова, яка буде відповідати за комунікацію волонтерських організацій та людей, які потребують допомоги. Буде надавати

певну інформацію по існуючим та зареєстрованим фондам та полегшувати їх пошук та обмін інформацією. Об`єкти клієнтської складової та перелік функціоналу з описом бізнес процесів:

- Об’єкт – Фонд. Перелік функціоналу:
  - Пошук фонду за назвою або категорією діяльності. Будь який неавторизований користувач системи повинен мати змогу відшукати потрібний йому фонд за категорією діяльності або назвою, або отримати інформацію по всім зареєстрованим фондам для подальшої взаємодії.
- Об’єкт - Заявка. Перелік функціоналу:
  - Створення заявки. Будь який неавторизований користувач системи повинен мати змогу залишити заявку на допомогу в будь-який зареєстрований фонд, вказавши перелік необхідних даних, а саме контакти та детальний опис заявки. Це дасть можливість волонтерам певного фонду дізнатись про потребу, вчасно скоординувати та зв’язатись з користувачем. Після створення заявки – вона повинна з’явитись в кабінеті волонтера конкретного фонду, а самому користувачеві на пошту буде надісланий номер самої заявки та поточний статус;
  - Можливість перевірити статус заявки за номером. Кожен користувач системи, який раніше лишав заявку на допомогу певному фонду, повинен мати змогу перевірити статус заявки за номером, який був надісланий користувачеві на пошту після створення. Це позбавить користувачів потреби в зайвих дзвінках або особистих повідомлень, що являє собою оптимізуючий фактор.

**Загальна складова** – ця складова буде накопичувати в собі той перелік функціональності, який буде містити в собі вся система, а не конкретна умовна одиниця або модуль. В загальній складовій я виділив один об’єкт – мова. Тобто у користувачів обох складових(корпоративна/клієнтська) повинна бути можливість змінити мову з української на міжнародну англійську, у випадку,

якщо фонд буде реєструватись не в Україні. Даний принцип називається локалізація. Локалізація – це процес адаптації програмного забезпечення або інших продуктів до особливостей конкретної регіональної аудиторії, в тому числі і мови. Локалізація дозволяє залучити більш користувачів з різних країн, оскільки людям комфортніше використовувати продукт, який відповідає іхній мові та культурі, чим саме покращує користувацький досвід. Локалізація, дозволить продукту виходити на нові ринки, створюючи нову велику базу конкурентних гравців.

### 3.2. Графічне зображення бізнес-процесів (діаграма Use case)

Для полегшення сприйняття перенесемо перелік бізнес-процесів на діаграму, інструмент, який використовується для візуалізації певних концепцій, ідей, взаємозв'язків або процесів. Вони дозволяють зображувати складні концепції і відносини елементів в легкозрозумілій формі. Діаграми можуть мати різноманітні форми та стилі, в залежності від мети та проблеми, яку ми будемо вирішувати. В нашому випадку для зображення загальної картини та розуміння функціональності системи, найбільше підійдуть діаграми Use case.

Use case-діаграма - це тип діаграми в аналізі та проектуванні програмного забезпечення, який використовується для моделювання функціональності системи з точки зору користувачів. Вона відображає взаємодію між системою та її зовнішніми акторами (користувачами, іншими системами або зовнішніми процесами).

Основною метою use case-діаграми є визначення і опис функціональності системи з позиції її користувачів. Вона допомагає команді розробників і зацікавленим сторонам зрозуміти, як система повинна працювати, які дії можуть виконувати користувачі і які результати вони очікують.

Основні компоненти use case-діаграми включають:

- Актори: представляють ролі, які виконують користувачі системи або інші системи;

- Use case (сценарії використання): представляють окремі функціональні дії або задачі, які можуть виконувати актори у системі;
- Взаємодії між акторами та use case: відображають способи взаємодії акторів з системою та між собою.

Діаграма представлена на рис. 3.1.

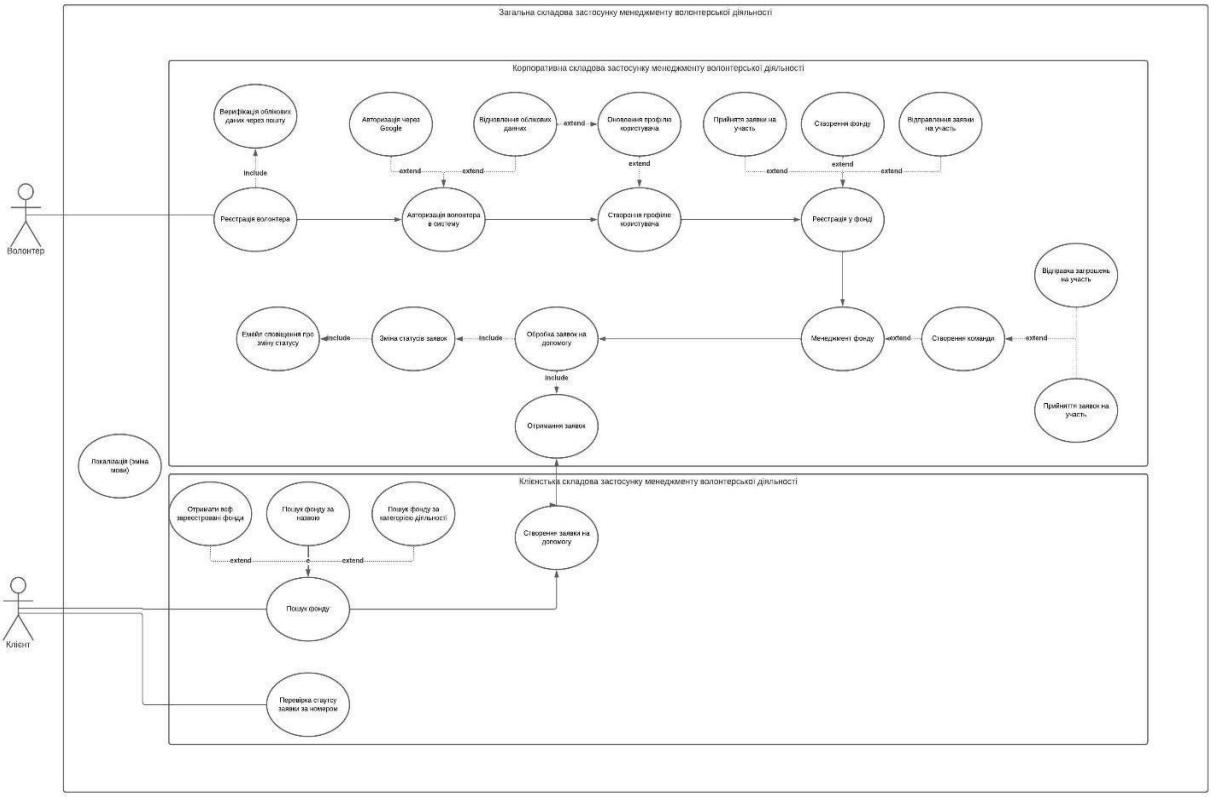


Рис. 3.1. Use case діаграма життєвого циклу застосунку

## 3.3. Проектування архітектури бази даних

Для того, щоб наш застосунок був динамічний, необхідно організувати зберігання даних. Для цього ми будемо використовувати реляційну базу даних PostgreSQL. Важливим аспектом є те, що проектувати архітектуру бази даних потрібно в першу чергу, тому що саме це буде мати каркасне значення в проекті. Незалежно який застосунок, кожен з них має шарову структуру. Де все починається з каркасу та вибудовується шар за шаром. І можна провести аналогію з будівництвом будинку. Спочатку возводиться каркас з різноманітних

металоконструкцій, далі на ці конструкцію насаджуються бетоні плити, далі навколо цього будується фасад. Жодний архітектурний проект не починається з фасаду. Тому, якщо каркас буде погано пропрацьований, то очевидно вся конструкція не витримає і будинок почне руйнуватись. За таким же принципом будуються додатки. Також проектування бази даних допомагає визначити, які дані потрібно зберігати для виконання функцій та задач застосунку. Це включає ідентифікацію сутностей, їх атрибутів та взаємозв'язків між ними. База даних взаємодіє з іншими компонентами застосунку, такими як інтерфейс користувача, логіка бізнес-процесів і зовнішні системи. Спроектована база даних може покращити продуктивність та ефективність застосунку. Це дозволяє оптимізувати запити до бази даних, швидше обробляти та отримувати дані. Крім того, вже на етапі проектування можна врахувати масштабованість, що дозволить легко розширювати базу даних у майбутньому. Проектування бази даних на ранніх етапах дозволяє виявити та виправити можливі проблеми ще до початку розробки. Це зменшує ризики помилок та необхідність вносити коригування у вже розроблений застосунок, що може бути витратним у часі й ресурсах. Правильною архітектурою вважається та, яка при масштабуванні не несе за собою високі ресурсні витрати і легко масштабується.

### 3.4. Визначення ключових сутностей та атрибутів

У контексті баз даних, сутність (також відома як таблиця) відображає конкретний об'єкт або поняття, яке ми бажаємо зберегти і управляти в базі даних. Сутність може представляти реальний об'єкт (наприклад, користувач, товар, замовлення) або абстрактний концепт (наприклад, студент, курс, категорія). Кожна сутність має набір атрибутів, які описують властивості або характеристики цієї сутності. Атрибути можуть мати різні типи даних, такі як рядки, числа, дата або булеві значення, що відображають характеристики сутності. Наприклад, для сутності "користувач" атрибути можуть включати "ім'я", "прізвище", "адресу електронної пошти" та "дату народження". Кожен

атрибут має ім'я, що його ідентифікує, і може мати деякі обмеження або правила щодо допустимих значень. Наприклад, атрибут "адреса електронної пошти" може мати обмеження, що вимагає, щоб значення було унікальним або мало певний формат електронної пошти. У базах даних сутності представляються у вигляді таблиць, де кожен стовпець відповідає атрибуту, а кожен рядок містить конкретні значення атрибутів для певного екземпляра сутності. Таким чином, таблиця "користувачі" може мати стовпці "ім'я", "прізвище", "адреса електронної пошти", а рядки таблиці міститимуть конкретні дані про окремих користувачів. Сутності та їх атрибути визначають структуру бази даних і допомагають зберігати та організовувати дані відповідно до вимог застосунку. Виділимо ключові сутності нашого застосунку.

User(Користувач) – ця сутність буде описувати та зберігати в собі всі дані про зареєстрованих користувачів. Атрибути представлені у табл. 3.1.

Таблиця 3.1

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| birth_date | Дата народження користувача | timestamp |
| email | Адреса поштової скриньки користувача | Varchar(255) |
| middle_name | По-батькові користувача | Varchar(255) |
| name | Ім’я користувача | Varchar(255) |
| phone | Мобільний номер телефону користувача | Varchar(255) |
| surname | Прізвище користувача | Varchar(255) |
| account_id | Посилання(Foreign key) на облікові дані користувача | bigint |

Продовження таблиці 3.1

| | | |
|---|---|---|
| fund_id | Посилання(Foreign key) на фонд в якому користувач бере участь | bigint |
| picture_id | Посилання(Foreign key) на метадані файлу аватару | bigint |

Account(Акаунт) – ця сутність буде описувати облікові та авторизаційні дані користувачів. Атрибути представлені у табл. 3.2.

Таблиця 3.2

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| auth_type | Тип реєстрованого акаунту. Внутрішня реєстрація, або реєстрація через сторонні сервіси(Google) | Varchar(255) |
| password | Пароль користувача | Varchar(255) |
| username | Умовне та унікальне ім’я користувача в системі | Varchar(255) |
| status | Статус користувача в системі | Varchar(255) |
| password_recover_link_id | Посилання(Foreign key) на запис верифікації (пароль) | bigint |

Продовження таблиці 3.2

| | | |
|---|---|---|
| verification_link_id | Посилання(Foreign key) на запис верифікації (особа) | bigint |

AccountRole(Роль акаунта) – сутність буде описувати ролі користувачів в системі. Атрибути представлені у табл. 3.3.

Таблиця 3.3

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| account_id | Посилання(Foreign key) | bigint |
| role | Назва ролі користувача | Varchar(255) |

VerificationLink(Верифікаційне посилання) – сутність, яка описує данні про посилання на верифікацію, які будуть отримувати користувачі при реєстрації, або відновленні облікових даних. Атрибути представлені у табл. 3.4.

Таблиця 3.4

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| expiration_time | Срок придатності верифікаційного посилання | timestamp |
| token | Унікальний користувацький ідентифікатор посилання | Varchar(255) |

Продовження таблиці 3.4

| | | |
|---|---|---|
| verification_type | Тип верифікаційного посилання. Може бути відновлення облікових даних, або підтвердження особистості в системі | Varchar(255) |

FileMetaData(Мета дані файлу) – сутність, яка описує дані про файли, які будуть завантажуватись у файлову систему через додаток. Атрибути представлені у табл. 3.5.

Таблиця 3.5

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| code | Унікальний користувацький код файлу в системі | Varchar(255) |
| created_at | Дата та час створення запису | timestamp |
| file_extension | Розширення файлу | Varchar(255) |
| name | Оригінальна назва файлу | Varchar(255) |
| path | Абсолютний шлях до файлу в файловій системі | Varchar(255) |
| status | Статус файлу в системі | Varchar(255) |
| updated_at | Дата оновлення запису | timestamp |

Fund(Фонд) – сутність, яка описує всі дані про фонди, організації, благодійні утворення, які будуть реєструватись в системі та в подальшому оброблюватись. Атрибути представлені у табл. 3.6.

Таблиця 3.6

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| name | Назва фонду | Varchar(255) |
| description | Опис деталей фонду | text |
| phone | Корпоративний номер телефону фонду | Varchar(255) |
| email | Корпоративна адреса поштової скриньки фонду | Varchar(255) |
| address_id | Посилання(Foreign key) на запис в сутності Address. Фактична адреса фонду | bigint |
| created_by | Посилання(Foreign key) на запис в сутності User. Фактичний керівник фонду | bigint |

FundCategories(Категорії діяльності певного фонду) – ця сутність буде зберігати категорії діяльності по кожному фонду. Атрибути представлені у табл. 3.7.

Таблиця 3.7

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| fund_id | Посилання(Foreign key) на запис в таблиці Fund | bigint |
| category | Назва категорії | Varchar(255) |

FundRequisite(Реквізит фонду) – сутність, яка буде зберігати в собі реквізити фондів, які будуть реєструватись. Атрибути представлені у табл. 3.8.

Таблиця 3.8

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| recipient | Отримувач | Varchar(255) |
| bank | Назва банку | Varchar(255) |
| bank_code | Код банку | Varchar(255) |
| Iban | Міжнародний номер банківського рахунку | Varchar(255) |
| legal_address | Юридична адреса отримувача | Varchar(255) |
| payment_account | Номер рахунку | Varchar(255) |
| payment_link | Посилання на оплату | Varchar(255) |
| swift_code | SWIFT-код | Varchar(255) |
| fund_id | Посилання(Foreign key) на запис в таблиці Fund | bigint |

FundTeamRequest(Запит на участь до фонду) - сутність, яка буде зберігати в собі запити на участь у фондах, які будуть реєструватись. Ця сутність буде містити, як запити на участь від користувачів, так і запрошення від фондів до користувачів, розділяючи бізнес-логіку за атрибутом fund_request_type. Атрибути представлені у табл. 3.9.

Таблиця 3.9

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| fund_request_type | Тип запиту на участь. Це може бути або запрошення від фонду, або запит від користувача на участь | Varchar(255) |
| status | Статус запиту/запрошення. Може бути новий, прийнятий, відхилений. | Varchar(255) |
| user_id | Посилання(Foreign key) на сутність User | bigint |
| fund_id | Посилання(Foreign key) на сутність Fund | bigint |

FundHelpRequest(Заявка на допомогу фонду) – сутність, яка описує заявки про допомогу, які будуть надходити до фондів. Запити будуть надходити від користувачів, які не реєстровані в системі. Тому до цієї сутності була створена зв’язуюча сутність, яка на перший погляд дублює дані, проте через те, що користувачі не зареєстровані, за контекстом схеми ми не можемо на не зраеєстрованого користувача зберігати дані у таблиці користувачів. Також на цю

сутність будуть додані табличні індекси. Індекс – це маханізм, довзоляючий реляційним базам даних здійснювати запити за різноманітними, прискорючими пошуковами алгоритмами. Для цієї сутності вони необхідні через велику кількість запитів до неї, та різноманітних фільтрацій за атрибутами таблиці. Атрибути представлені у табл. 3.10.

Таблиця 3.10

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| number | Унікальний номер заявки на допомогу | Varchar(255) |
| status | Статус заявки. Може бути новий, прийнятий, відхилений, в обробці або завершений | Varchar(255) |
| created_at | Дата створення запису | timestamp |
| updated_at | Дата оновлення запису | timestamp |
| description | Деталі заявки | text |
| executor_id | Посилання(Foreign key) на сутність HelpRequestExecutor | bigint |
| fund_id | Посилання(Foreign key) на сутність Fund | bigint |

HelpRequestExecutor(Виконавець заявки на допомогу) – сутність, яка зберігає інформацію про виконавців запитів на допомогу до фондів. Це зв’язуюча сутність до сутності FundHelpRequest(Заявка на допомогу фонду), яка на перший погляд дублює дані, проте через те, що користувачі не зареєстровані, за

контекстом схеми ми не можемо на не зраеєстрованого користувача зберігати дані у таблиці користувачів.  На цю таблицю також будуть додані індекси. Атрибути представлені у табл. 3.11.

Таблиця 3.11

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| name | Ім’я виконавця | Varchar(255) |
| surname | Прізвище виконавця | Varchar(255) |
| middle_name | По-батькові виконавця | Varchar(255) |
| organization_name | За необхідністю назва організаціях від котрої здійснюється запит | Varchar(255) |
| phone | Контактний номер телефону виконавця | Varchar(255) |
| email | Адреса поштової скриньки виконавця | Varchar(255) |

MessageEntry(Повідомлення) – сутність, яка зберігає інформацію про відправленні повідомлення або сповіщення, через емейл, смс, тощо. Повідомлення можуть надсилатись, як одному так і декільком користувачам, тому таблиця повідомлень та таблиця адресатів була поділена на дві таблиці, що відповідає принципам нормалізації. Також таблиця буде містити в собі технічні атрибути такі як статус відправки та звіт з відправки. Тому що повідомлення відправляються через сторонній сервіс, а він не завжди може бути доступний, тому доцільно було додати інформаційні, технічіні атрибути. Атрибути

представлені у табл. 3.12.

Таблиця 3.12

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| created_at | Дата та час створення запису | timestamp |
| message | Повідомлення | Varchar(255) |
| report_message | Звіт причини у випадку, якщо повідомлення не було доставлено | Varchar(255) |
| send_attemt | Кількість спроб відправлень повідомлення системою | integer |
| status | Статус повідомлення. Може бути новий, успішний, не успішний, в обробці, не коректний запит | Varchar(255) |
| subject | Предмет повідомлення або тематика | Varchar(255) |
| type | Тип повідомлення. Може бути на пошту, на пошту з додатком або смс | Varchar(255) |

MessageEntryAcounts – ця сутність зберігає адреси (номер телефону, або емейл), куди було відправлене конкретне повідомлення. Атрибути представлені

у табл. 3.13.

Таблиця 3.13

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| message_entry_id | Посилання(Foreign key) на сутність MessageEntry | bigint |
| account | Адреса повідомлення. Може бути адреса поштової скриньки або номер телефону | Varchar(255) |

Address(Адреса) – ця сутність зберігає фактичні адреси. Атрибути представлені у табл. 3.14.

Таблиця 3.14

| Назва атрибуту | Опис | Тип даних |
|---|---|---|
| id | Унікальний ідентифікатор, що однозначно ідентифікує кожен запис у таблиці бази даних | bigint |
| country | Назва країни | Varchar(255) |
| region | Назва регіону | Varchar(255) |
| city | Назва міста | Varchar(255) |
| street | Назва вулиці | Varchar(255) |
| house | Номер будинку | Varchar(255) |
| corpus | Номер корпусу | Varchar(255) |

Продовження таблиці 3.14

| | | |
|---|---|---|
| office | Номер офісу | Varchar(255) |
| post_index | Поштовий індекс | Varchar(255) |

### 3.5. Визначення зв’язків між сутностями та графічне представлення схеми (E-R diagram)

Реляційні зв'язки в базі даних визначають залежності та зв'язки між різними таблицями. Вони використовуються для моделювання і встановлення зв'язків між даними, що дозволяє ефективно організувати та керувати інформацією в базі даних.

У реляційній моделі бази даних існує кілька типів зв'язків:

- Один до одного (One-to-One): Кожен запис в одній таблиці відповідає точно одному запису в іншій таблиці. Цей тип зв'язку використовується, коли між двома сутностями існує пряма взаємозалежність;
- Один до багатьох (One-to-Many): Кожен запис в одній таблиці може мати багато відповідних записів у іншій таблиці, але кожен запис у другій таблиці може бути пов'язаний лише з одним записом першої таблиці. Цей тип зв'язку використовується, коли одна сутність має відношення до кількох екземплярів іншої сутності;
- Багато до багатьох (Many-to-Many): Кожен запис в одній таблиці може бути пов'язаний з багатьма записами в іншій таблиці, і навпаки. Цей тип зв'язку використовується, коли кілька екземплярів однієї сутності можуть бути пов'язані з кількома екземплярами іншої сутності. Для моделювання такого типу зв'язку часто використовуються проміжні таблиці (таблиці-посередники).

Реляційні зв'язки в базі даних дозволяють організовувати дані в структурований і зв'язаний спосіб. Вони дозволяють зберігати дані у відповідному форматі, спрощують розподіл даних між різними таблицями і забезпечують цілісність даних шляхом забезпечення правильних залежностей між записами. Крім того, реляційні зв'язки дозволяють виконувати запити до

бази даних, що включають дані з різних таблиць, шляхом злиття (join) таблиць за встановленими зв'язками. Це дозволяє отримувати пов'язану інформацію з різних джерел та забезпечує ефективне використання даних у базі даних. Дуже важливо розуміти, що на базі зв’язків будується бізнес-логіка додатку, тому зв’язками можна контролювати розвиток розробки та робити певні обмеження. Наприклад звичайно більш доцільно та логічно було б зробити, щоб зв’язок між користувачем і фондом був багато-до-багатьох. Це б означало, що волонтери могли б брати участь не тільки в одному фонді, якщо казати мовою бізнес-процесів. Проте це і на рівень ускладнює розробку та підвищує об’єм роботи, часовитрати. Тому було прийняте рішення зробити обмеження на рівні зв’язків. Визначимо зв’язки між сутностями відносно нашої предметної області у вигляді таблиці. Зв’язки представлені у табл. 3.15.

Таблиця 3.15

| Сутність 1 | Сутність 2 | Тип зв’язку | Додатковий опис |
|---|---|---|---|
| User | Account | OneToOne | Одному користувачу може відповідати тільки один запис обілікових даних |
| Account | AccountRole | OneToMany | Акаунт може містити декілька ролей |
| Account | VerificationLink | OneToOne | В одного акаунта може бути тільки одне верифікаційне посилання |
| User | FileMetaData | OneToOne | Користувачу може відповідати тільки один аватар |
| User | FundTeamRequest | OneToMany | Користувач може робити багато запитів на участь у фондах і також мати багато запрошень. В свою чергу запит може належати тільки одному користувачу і фонду |

Продовження таблиці 3.15

| | | | |
|---|---|---|---|
| Fund | User | OneToMany | Фонд може мати в команді багато користувачів. В свою чергу користувач може приймати участь тільки в одному фонді |

Зв’язки сутностей клієнстької частини були винесені у окрему таблицю зв’язків, щоб розділити контекст. Їх можна продивитись у табл. 3.16.

Таблиця 3.16

| | | | |
|---|---|---|---|
| Fund | FundTeamRequest | OneToMany | Фонд може робити багато запрошень користувачам і також багато запитів від користувачів. В свою чергу запрошення/запит може належати тільки одному користувачу і тільки одному фонду |
| Fund | FundRequisite | OneToMany | Фонд може мати багато реквізитів. В свою чергу реквізит може належати тільки одному фонду, через свою унікальність |
| Fund | FundHelpRequest | OneToMany | Фонд може мати багато запитів на допомогу |
| Fund | Address | OneToOne | Фонду може відповідати тільки одна адреса |
| Fund | FundCategory | OneToOne | Фонду можуть відповідати декілька категорій |
| FundHelpRequest | HelpRequestExecutor | OneToOne | Запиту на допомогу може відповідати тільки один виконавець |

Продовження таблиці 3.16

| | | | |
|---|---|---|---|
| MessageEntry | MessageEntryAccount | OneToMany | Одне повідомлення може відноситись до декількох адресатів |

Для більш детального розуміння сформованої схеми бази даних, її можна візуалізувати графічно за допомогою ER-діаграм. ER-діаграма (Entity-Relationship diagram) - це графічне зображення, що використовується для моделювання і представлення структури бази даних. Вона використовується для візуалізації сутностей (таблиць), їх атрибутів і зв'язків між ними. ER-діаграми дозволяють легко розуміти структуру бази даних та залежності між її складовими частинами. Основні елементи ER-діаграми:

Сутності (Entities): Вони представляють окремі таблиці або сутності в базі даних. Наприклад, сутність "Користувач" відповідає таблиці "Користувачі".

Атрибути (Attributes): Вони визначають властивості або характеристики сутностей. Кожна сутність може мати свій набір атрибутів. Наприклад, для сутності "Користувач" атрибути можуть включати "ім'я", "прізвище" і "адресу електронної пошти".

Зв'язки (Relationships): Вони визначають зв'язки та залежності між сутностями. Наприклад, зв'язок "Має" між сутностями "Користувач" і "Замовлення" вказує на те, що кожен користувач може мати багато замовлень. Зображення діаграми зв’язків можна побачити на рис. 3.2.

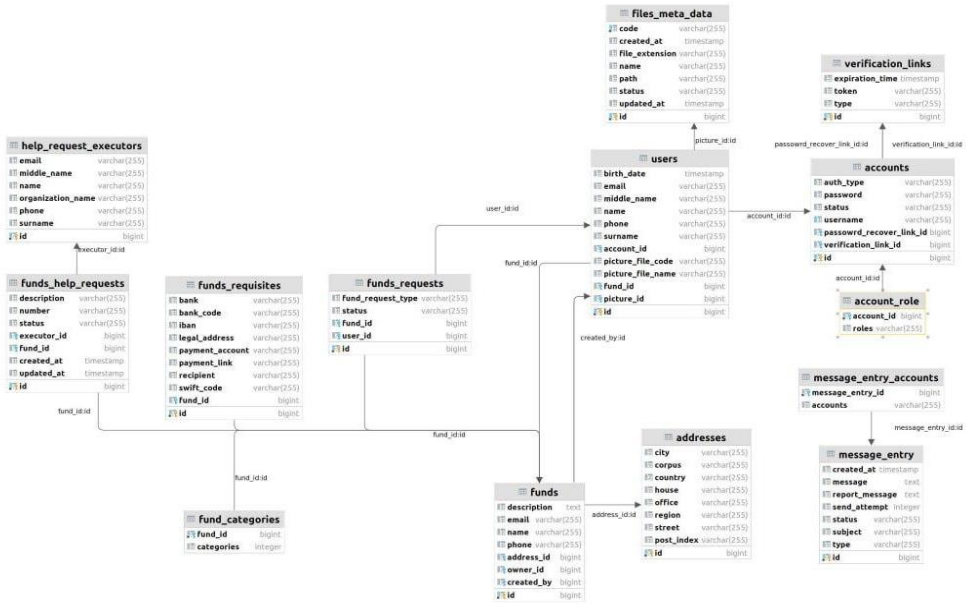


Рис. 3.2. ER діаграма схеми бази даних

## 3.6. Побудова архітектури проекту та розподіл на шари

Після створення схеми бази даних доцільно перейти до формування шарів архітектури проекту. Найбільш ефективним та структурованим способом є розподіл проекта на цільові пакети з технічної точки зору, в кожному цільовому технічному пакеті додатковий розподіл на пакети бізнес-процесів. Таким чином це дасть нам можливість глобально бачити технічну картину вже всередині з певною деталізацією. Цей метод краще зарекомендував себе, чим ділити спочатку на бізнес процеси, а потім всередині розділяти на технічні аспекти. Тому що ієрархія структури проекта повинна рухатись від більш глобального, абстрактного до специфічного. І в більшості випадків бізнес-процес є специфічніший, ніж технічний аспект. Тому в нашому випадку було прийнято рішеня зробити монолітний проект з домішками патерну клієнт-серверної архітектури. Для більш детального розуміння і back-end і front-end бути писатись однією мовою, в рамках одного проекту і навіть компілюватись вони будуть одночасно. Але незважаючи на це, бібліотека Vaadin, яка допоможе нам розробити подібного формату застосунок, в будь-якому випадку в глибині свого вихідного коду перетворює Java код на html розмітку, де за обробку подій і

внутрішні REST запити відповідає Java Script. Тобто формально і клієнт і сервер знаходяться в одному контексті, що повністю відповідає моноліту, але все одно клієнт надсилає запити формату клієнт-сервер, хоч і внутрішні. Так я пояснюю термін архітектура монолітного характеру з домішками клієнт-серверного прототипу. Якщо переходити до нашого проекту, то він буде поділений на три частини, три пакети – backend, ui, common. Пакет backend буде відповідати за виконання основної бізнес-логіки, обробку даних, збереження в базу даних, локалізацію та конфігурацію. Пакет ui – буде містити в собі логіку клієнтської частини, інтерфейсу, валідацію вхідних даних, та надсилання їх на обробку до модулю back-end. Пакет common – цей пакет буде містити в собі спільні дані, такі як константи, класи представлення сутностей в базі даних, різноманітні маппери та класи передачі даних. Директорія resources відповідає за статичні данні проекту та місце зберігання глобальних конфігурацій проекту. В ній у нас будуть зберігатись файли локалізації та файл налаштування додатку. Структуру проекту можна роглянути на рис. 3.3.

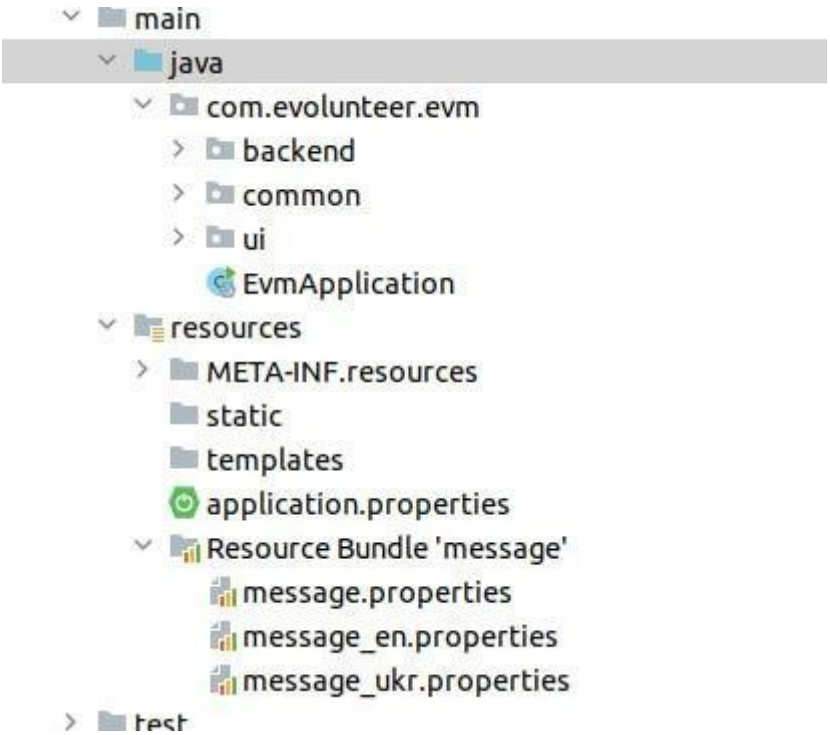


Рис.3.3. Структура проекту

Далі розіб’ємо пакет backend на технічні аспекти, а вже кожен технічний аспект на специфічний бізнес-процес компонент. Перш за все створимо пакет config, він буде зберігати усі необхідні глобальні класи налаштувань. Далі необхідно додати пакет repository, цей пакет буде містити в собі інтерфейси, які будуть наслідувати Spring Data класи та забезпечувати зв’язок з базою даних, завдяки налаштуванням, які будуть вказані в конфігураціях. А вже всередині пакету repository можна розбити на специфічні бізнес-процеси, такі як обробка користувачів, фондів, файлів і тд. Також необхідно створити окремий пакет з безпековими налаштуваннями і назвати його security. І останній пакет, який буде створенний і буде також додатково поділений на під-пакети бізнес-процесів – це service. В ньому будуть міститись класи-обробники цільової бізнес-логіки. Структуру пакету backend можна розглянути на рис. 3.4.


Рис. 3.4. Структура пакету backend

Пакет common буде містити в собі domain класи, мапери та утилітарні класи. Domain класи – це класи, які відображають реальні об’єкти або концепції в домені додатка. Мапери – це класи, які використовуються для перетворення

даних між різними об'єктами або структурами даних. Ці класи допомагають забезпечити взаємодію між об'єктно-орієнтованим світом та іншими джерелами даних, такими як реляційні бази даних, веб-сервіси або файли. Утилітарні класи - це класи, які містять набір статичних методів, які зазвичай використовуються для виконання загальних функцій, що не пов'язані з конкретною об'єктно-орієнтованою логікою. Утилітарні класи можуть включати методи для виконання різних операцій, таких як математичні обчислення, маніпуляції зі строками, робота з датами і часом, введення-виведення даних, обробка файлів і багато інших. Структуру пакету common можна розглянути на рис. 3.5.

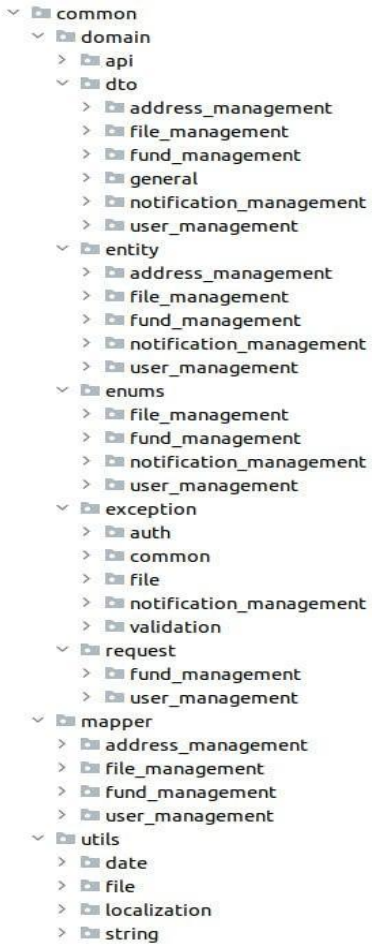


Рис. 3.5. Структура пакету Common

Останній пакет ui розміщує в собі обробники подій та компоненти клієнтського інтерфейсу такі як сторінки(view), макети(layout), діалогові вікна(dialog), кнопки(button), контейнери(div) тощо. Один з найкращих

принципів в програмуванні будь-якою мовою – це перевикористання коду. Тому цей пакет можна додатково розбити на компоненти general, які будуть перевикористовуватись та більш специфічні компоненти додатку app. А також виділити окремий пакет утилітарних класів, які будуть стосуватись конкретно користувацького інтерфейсу. Структуру пакету common можна розглянути на рис. 3.6.

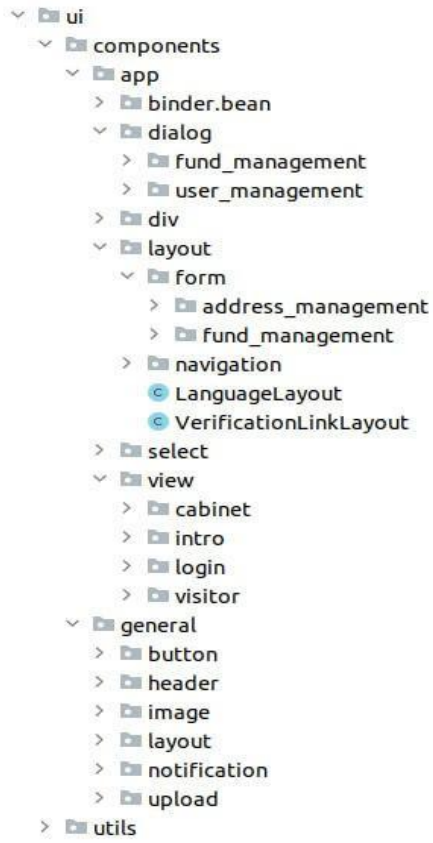


Рис. 3.6. Структура пакету UI

### 3.7. Програмна реалізація застосунку

Розглянемо реалізацію базового функціоналу, а саме стартової сторінки, де у користувача буде відображатись текст привітання та дві кнопки вибору волонтер чи звичайний відвідувач. Кнопка волонтер буде направляти на сторінку авторизації, а кнопка звичайний відвідувач направляти на сторінку з пошуком фондів, з можливістю звернення. На малюнку видно, що в конструкторі класу IntroductionView створюється загальний контейнер Div, ньому призначається

бажаний стиль, вже локалізований вище текст. Далі створюється макет кнопок та також додається у контейнер обгортку. І далі цей контейнер додається на батьківську сторінку. Програмна реалізація стартової сторінки зображена на рис. 3.7.

```
@Route(RouteUtils.INTRO_ROUTE)
@PageTitle("Introduction | E-Volunteer")
public class IntroductionView extends VerticalLayout {

    ± illia *
    public IntroductionView(MessageSource messageSource) {

        setSizeFull();
        setAlignItems(Alignment.CENTER);
        setJustifyContentMode(JustifyContentMode.CENTER);

        final String introductionText = messageSource.getMessage(LocalizationUtils.UI.IntroductionView.INTRODUCTION_TEXT,
                args: null, LocalizationUtils.getLocale());
        final String simpleClientButtonText = messageSource.getMessage(LocalizationUtils.UI.IntroductionView.SIMPLE_CLIENT_BUTTON_TEXT,
                args: null, LocalizationUtils.getLocale());
        final String volunteerClientButtonText = messageSource.getMessage(LocalizationUtils.UI.IntroductionView.VOLUNTEER_BUTTON_TEXT,
                args: null, LocalizationUtils.getLocale());

        final Div introductionDiv = new Div();
        introductionDiv.setWidth("100%");
        introductionDiv.getStyle().set("text-align", "center");
        introductionDiv.setText(introductionText);


        final Button volunteerButton = new Button(volunteerClientButtonText, buttonClickEvent -> UI.getCurrent().navigate(RouteUtils.LOGIN_ROUTE));
        final Button simpleClientButton = new Button(simpleClientButtonText, buttonClickEvent -> UI.getCurrent().navigate(RouteUtils.VISITOR_ROUTE));

        final HorizontalLayout buttonLayout = new HorizontalLayout(volunteerButton, simpleClientButton);
        buttonLayout.setWidth("100%");
        buttonLayout.setJustifyContentMode(JustifyContentMode.CENTER);

        introductionDiv.add(buttonLayout);
        add(new LanguageLayout(messageSource), introductionDiv);
    }
}
```

Рис. 3.7. Компонент стартової сторінки

Далі можна розглянути повний процес реєстрації користувача. На сторінці авторизації користувач має натиснути кнопку в мене немає облікового запису, йому відкриється діалогове вікно з формою реєстрації. Після заповнення форми на натиснення кнопки підтвердити, дані повинні зберегтись в базу даних та створитись новий обліковий запис за яким потім можна буде авторизуватись в систему. На рис. 3.8 видно, що в батьківському компоненті(діалогове вікно) ми створюємо поля для вводу, в яких також визначаємо правила валідації та кнопку підтвердити, яка буде безпосередньо делегувати процес реєстрації далі по сервісам. Слід зазначити, що компонент реєстрації створений в пакеті ui, де далі в процесі делегації передасть обов’язок виконання логіки реєстрації та

збереження в базу даних сервісу розміщеному вже в пакеті backend.

```
        .bind(CreateUserRequest::getEmail, CreateUserRequest::setEmail);

final DatePicker birthDateField = new DatePicker(birthDateFieldText);
birthDateField.setRequired(true);
birthDateField.setRequiredIndicatorVisible(true);
createUserRequestBinder.forField(birthDateField)
        .withValidator(Objects::nonNull, birthDateValidationText)
        .bind(CreateUserRequest::getBirthDate, CreateUserRequest::setBirthDate);

final DatePicker.DatePickerI18n singleFormatI18n = new DatePicker.DatePickerI18n();
singleFormatI18n.setDateFormat(DateUtils.DEFAULT_DATE_FORMAT);
birthDateField.setI18n(singleFormatI18n);

final TextField usernameField = new TextField(usernameFieldText);
usernameField.setRequired(true);
usernameField.setRequiredIndicatorVisible(true);
createUserRequestBinder.forField(usernameField)
        .withValidator(username -> !StringUtils.isBlank(username), usernameValidationText)
        .bind(CreateUserRequest::getUsername, CreateUserRequest::setUsername);

final PasswordField passwordField = new PasswordField(passwordFieldText);
passwordField.setRequired(true);
passwordField.setRequiredIndicatorVisible(true);
createUserRequestBinder.forField(passwordField)
        .withValidator(new RegexpValidator(passwordValidationText, ValidationUtils.PASSWORD_REGEX))
        .bind(CreateUserRequest::getPassword, CreateUserRequest::setPassword);

confirmButton = new ConfirmButton(messageSource, locale, this.registration());

createUserRequestBinder.setBean(createUserRequest);

final FormLayout registrationForm = new FormLayout();

registrationForm.add(registrationFormHeader, nameField, surnameField, middleNameField,
        phoneField, emailField, birthDateField, credentialsHeader, usernameField, passwordField);
registrationForm.setResponsiveSteps(new FormLayout.ResponsiveStep( minWidth: "200px",  columns: 1));

add(registrationForm);
this.getFooter().add(confirmButton, new CancelButton(messageSource, locale, buttonClickEvent -> this.close()));
}
```

Рис. 3.8. Компонент діалогового вікна реєстрації користувача

Далі ми безпосередньо розглянемо функцію реєстрації визначеної в цьому ж компоненті. В ній ми валідуємо дані, які були введені в поля. І якщо введена інформація валідна то викликаємо метод реєстрації в сервісі користувачів(UserService), який також додатково провалідує дані та збереже їх у базу даних. В іншому ж випадку ми виведемо всі помилки валідації користувачу на екран на коригування та не будемо здійснювати делегування процесу реєстрації UserService-у. Функцію збереження форми користувача можна побачити на рис. 3.9.

```
private ComponentEventListener<ClickEvent<Button>> registration() {
    return buttonClickEvent -> {
        if (createUserRequestBinder.writeBeanIfValid(createUserRequest)) {
            final String successfullyRegistration = messageSource.getMessage(SUCCESS_REGISTRATION_TEXT,  args: null, locale);
            final String accountAlreadyExistValidationText = messageSource.getMessage(VALIDATION_ACCOUNT_ALREADY_EXIST_BY_USERNAME_ERROR,
                    new String[]{createUserRequest.getUsername()}, locale);

            final Optional<AccountDto> optionalAccountDto = accountService.getAccountByUsername(createUserRequest.getUsername());
            if (optionalAccountDto.isPresent() && !optionalAccountDto.get().getStatus().isExpired()) {
                NotificationFactory.error(accountAlreadyExistValidationText).open();
            } else  {
                userService.registerInternalUser(createUserRequest);
                this.removeAll();
                this.setWidth("500px");
                this.setHeight("300px");


                final VerticalLayout verticalLayout = new VerticalLayout();
                verticalLayout.setSizeFull();
                verticalLayout.setAlignItems(FlexComponent.Alignment.CENTER);
                verticalLayout.setJustifyContentMode(FlexComponent.JustifyContentMode.CENTER);

                final Icon icon = new Icon(VaadinIcon.ENVELOPE_O);
                final Span message = new Span(successfullyRegistration);
                verticalLayout.add(message, icon);
                add(verticalLayout);
                this.getFooter().remove(confirmButton);
            }
        }
    };
}
```

Рис. 3.9. Функція збереження форми реєстрації користувача

Останньою ланкою реєстрації є отримання цільовим сервісом даних та збереження іх в базу. На прикладі ми можемо побачити реалізацію методу збереження користувацьких даних в базу даних в UserService. Відмічаємо, що спочатку дані додатково валідуються, далі для користувача створюється обліковий запис окремим сервісом AccountService, після цього вхідний об’єкт перетворюється на об’єкт сутності користувача маппером та за допомогою репозиторія зберігається в базу даних. Програмний код функції збереження даних користувача в базу можна розглянути на рис. 3.10.

```java
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;
    private final AccountService accountService;
    private final AccountMapper accountMapper;
    private final UserMapper userMapper;
    private final FundMapper fundMapper;
    private final FileMetaDataMapper fileMetaDataMapper;
    private final FileService fileService;

    1 usage  ± illia *
    @Transactional
    @Override
    public BaseUserDto registerInternalUser(final CreateUserRequest registrationRequest) {
        if (Objects.isNull(registrationRequest)) {
            throw new ValidationException("Unable to create user. Registration request is null!");
        }
        ValidationUtils.validate(registrationRequest);
        final AccountDto createdAccount = accountService.createInternalAccount(
                new CreateAccountRequest(registrationRequest.getUsername(), registrationRequest.getPassword(), registrationRequest.getEmail()));

        final User mappedUser = userMapper.mapRegistrationRequestToUser(registrationRequest);
        mappedUser.setAccountDetails(accountMapper.mapAccountDtoToAccount(createdAccount));
        return userMapper.mapUserToUserDto(userRepository.save(mappedUser));
    }
```

Рис. 3.10. Функція збереження даних користувача в базу даних

Додатково розглянемо реалізацію сторінки, де будуть розміщатись заявки на допомогу від людей у вигляді таблиці. Слід зазаначити, що ця сторінка вже закрита, і доступна тільки для авторизованих користувачів! В нашому батьківсьому компоненті, ми спочатку отримуємо поточного авторизованого користувача. Перевіряємо, якщо він не прив’язаний до якогось фонду – то нічого йому не відображаємо. Якщо прив’язаний то отримуємо з об’єкту цього користувача фонд, а з фонду отримуємо всі заявки на допомогу. Створюємо таблицю(Grid) типу заявки(FundHelpRequestDto). Далі формуємо колонки addColumn() на основі полів об’єкту FundHelpRequestDto десь напряму, а десь перевизначаємо і робимо колонку у вигляді компоненту. Кожній колонці назначаємо локалізований варіант заголовку. Наприклад колонка статус буде у вигляді випадаючого списку(Select), та при виборі іншого елементу випадаючого списку буде викликатись функція FundService – updateFundHelpRequestStatus(requestId, newStatus), яка буде оновлювати статус заявки та в якості аргументів буде приймати унікальний ідентифікатор заявки та нове значення статусу. Програмний код компоненту заявок на допомогу від користувачів можна розглянути на рис. 3.11.

```java
public ApplicationsView(MessageSource messageSource, FundService fundService, UserService userService) {
    this.locale = LocalizationUtils.getLocale();
    this.messageSource = messageSource;

    final BaseUserDto contextUser = userService.getContextUser();
    final FundDtoFull fund = fundService.getFundById(contextUser.getFund().getId());

    final String requestNumberHeader = messageSource.getMessage(GRID_REQUEST_NUMBER_HEADER_TEXT, args: null, locale);
    final String requestStatusHeader = messageSource.getMessage(GRID_REQUEST_STATUS_HEADER_TEXT, args: null, locale);
    final String requestCreatedAtHeader = messageSource.getMessage(GRID_REQUEST_CREATE_AT_HEADER_TEXT, args: null, locale);
    final String requestDetailsHeader = messageSource.getMessage(GRID_REQUEST_DETAILS_HEADER_TEXT, args: null, locale);
    final String requestExecutorHeader = messageSource.getMessage(GRID_REQUEST_EXECUTOR_HEADER_TEXT, args: null, locale);

    final ListDataProvider<FundHelpRequestDto> dataProvider = new ListDataProvider<>(fund.getHelpRequests());

    final Grid<FundHelpRequestDto> grid = new Grid<>(FundHelpRequestDto.class, autoCreateColumns: false);
    grid.addThemeVariants(GridVariant.LUMO_WRAP_CELL_CONTENT);
    grid.addColumn(FundHelpRequestDto::getNumber).setHeader(requestNumberHeader).setSortable(true);
    grid.addComponentColumn(fundHelpRequestDto -> {
        final Select<FundHelpRequestStatus> statuses = new Select<>();
        statuses.setItems(FundHelpRequestStatus.values());
        statuses.setItemLabelGenerator(status -> messageSource.getMessage(status.getLocalizedValue(), args: null, locale));
        statuses.setValue(fundHelpRequestDto.getStatus());
        statuses.addValueChangeListener(event -> {
            fundService.updateFundHelpRequestStatus(fundHelpRequestDto.getId(), event.getValue());
            UI.getCurrent().getPage().reload();
        });
        return statuses;
    }).setHeader(requestStatusHeader).setSortable(true);
    grid.addColumn(FundHelpRequestDto::getFormattedCreateAt).setHeader(requestCreatedAtHeader).setSortable(true);
    grid.addColumn(FundHelpRequestDto::getDescription).setHeader(requestDetailsHeader).setSortable(true);
    grid.addComponentColumn(fundHelpRequestDto -> this.executorLayout(fundHelpRequestDto.getExecutor()))
            .setHeader(requestExecutorHeader).setSortable(true);
    grid.setItems(dataProvider);

    this.add(grid);
}
```

Рис. 3.11. Компонент заявок на допомогу від користувачів

### 3.8. Тестування програмного застосунку

На стартовій сторінці ви маєте можливість обрати для себе чи ви волонтер, чи ви просто відвідувач який хоче зробити благодійний внесок або звернутись по допомогу. Стартову сторінку можна побачити на рис. 3.12.

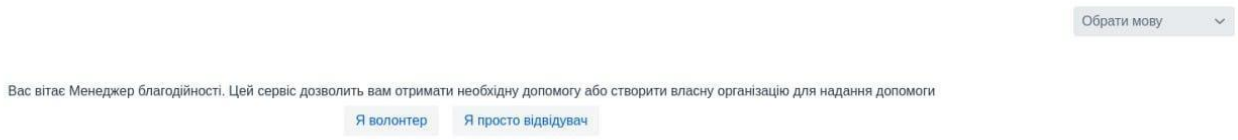


Рис. 3.12. Стартова сторінка додатку

Якщо ви натиснете на кнопку “Я просто відвідувач” ви потрапите на інформаційну сторінку зароеєстрованих фондів. Тут ви зможете відшукати фонд за категорією діяльності, ім’ям або просто отримати всі зареєстровані фонди. Зробити благодійний внесок за вказаними реквізитами, або звернутись по допомогу до певної організації. Сторінку простого відвідувача можна побачити на рис. 3.13.

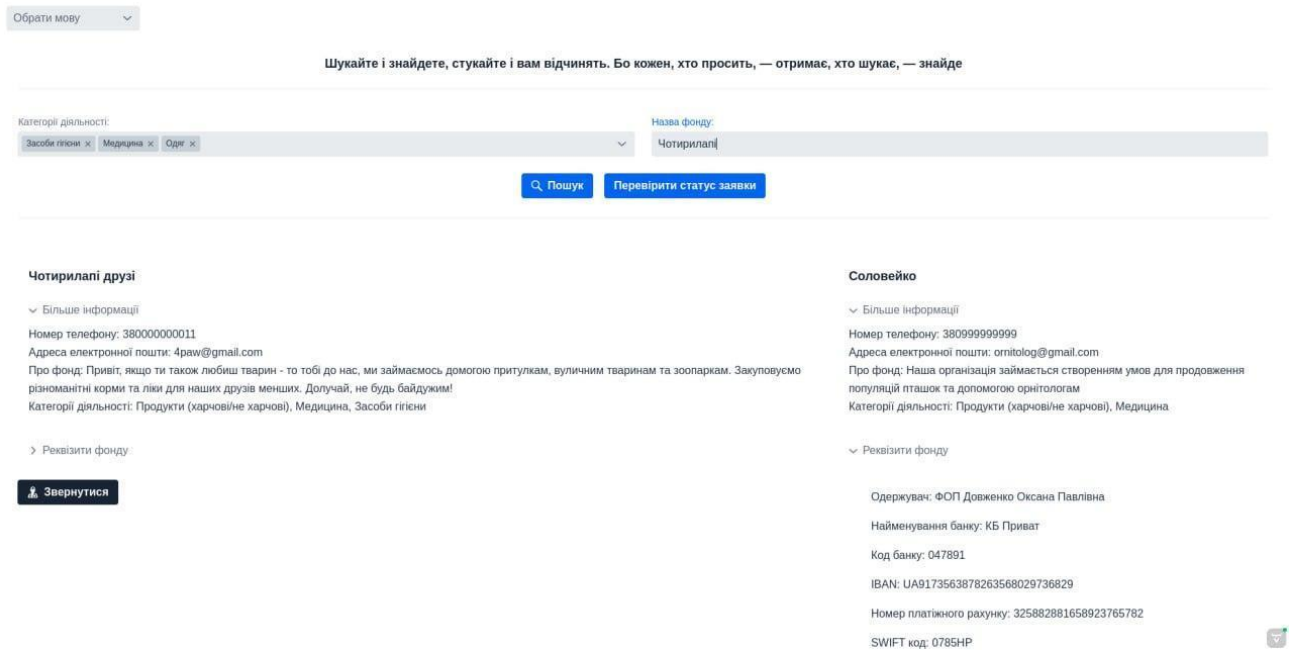


Рис. 3.13 Сторінка неавторизованого користувача “Я просто відвідувач”

По натисненю на кнопку звернутись – вам відкриється діалогове вікно, де потрібно буде заповнити всі необхідні дані. Якщо щось обов’язкове не буде заповнене – ви отримаєте візуально зрозуміло сповіщення що не було заповнено. Після натиснення кнопки “Надіслати”фонд отримає вашу заявку, а вам буде виданий унікальний номер вашої заявки. Далі ви будете отримувати сповіщення на пошту при кожній зміні статусу заявки, а також зможете перевірити статус заявки за виданим унікальним номером по кнопці “Перевірити статус заявки”. Взаємодію користувача з фондами можна побачити на рис. 3.14, рис. 3.15, рис. 3.16, рис. 3.17.


Рис. 3.14. Форма заявки на допомогу у випадку невалідних даних

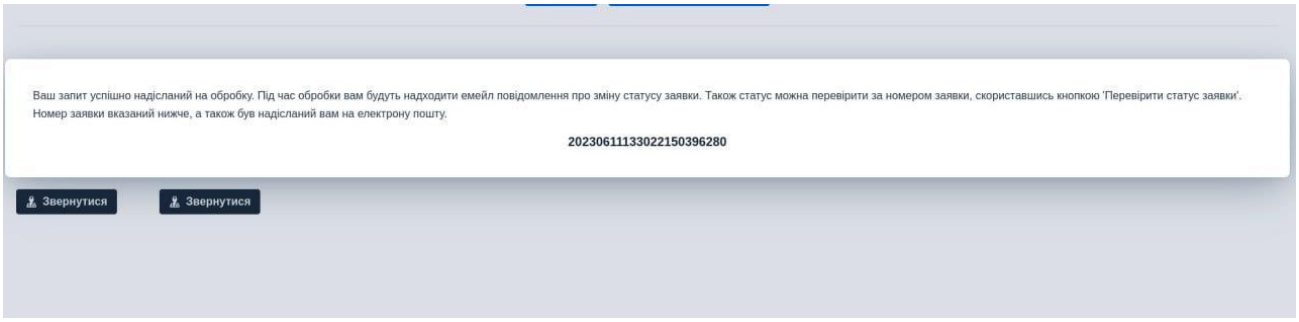


Рис. 3.15. Діалогове вікно у випадку успішної відправки заявки на допомогу

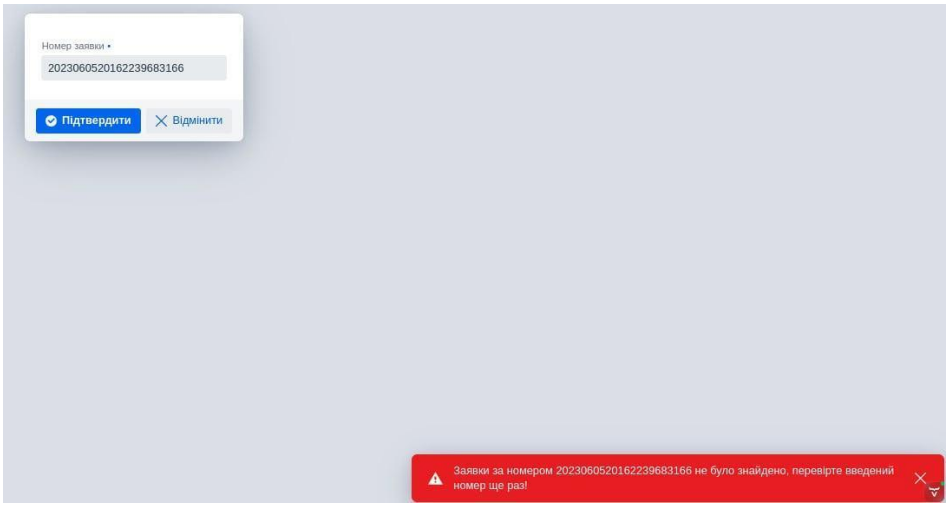


Рис. 3.16. Форма перевірки статусу заявки на допомогу за номером. Випадок, якщо заявки за номером не існує

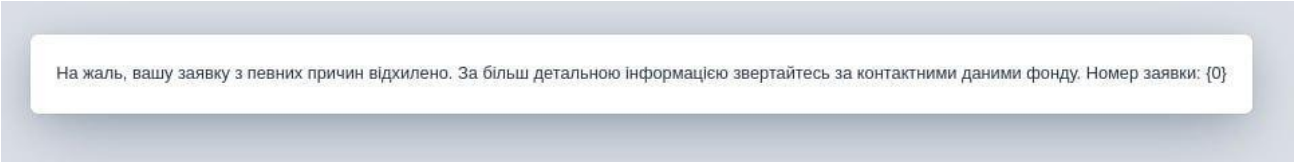


Рис. 3.17. Форма перевірки статусу заявки на допомогу за номером. Випадок успішної перевірки

Далі розглянемо випадок, якщо ми хочемо скористатись додатком в ролі волонтера. Якщо на стартовій сторінці ми натиснемо кнопку “Я волонтер” ми потрапимо на сторінку авторизації, де зможемо авторизуватись, авторизуватись через Google акаунт, зареєструватись, або змінити пароль. Сторінку авторизації волонтера можна розглянути на рис. 3.18.

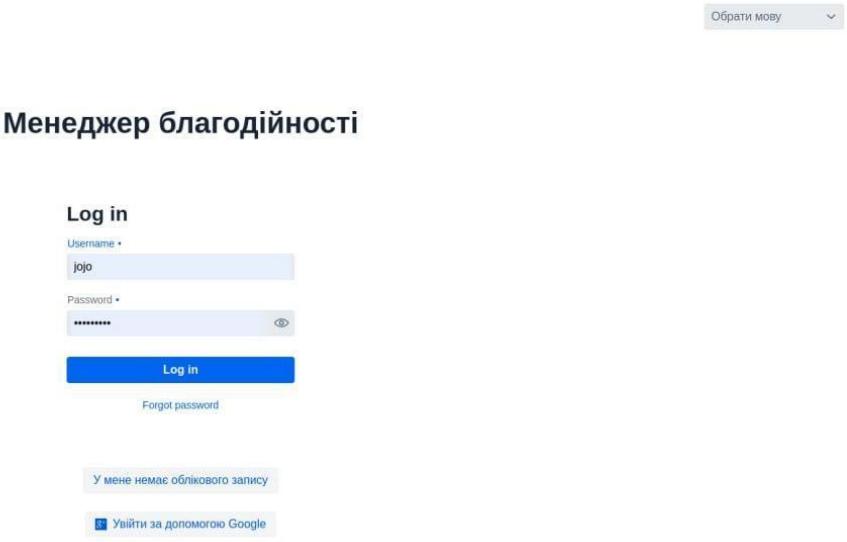


Рис. 3.18. Сторінка авторизації волонтера в систему. “Я волонтер”

Після успішної реєстрації та авторизації ми попадаємо в систему “Кабінет волонтера” у профіль користувача, де ми можемо оновити свої особисті або облікові дані та завантажити фото профілю. Сторінку профілю користувача можна розглянути на рис. 3.19.

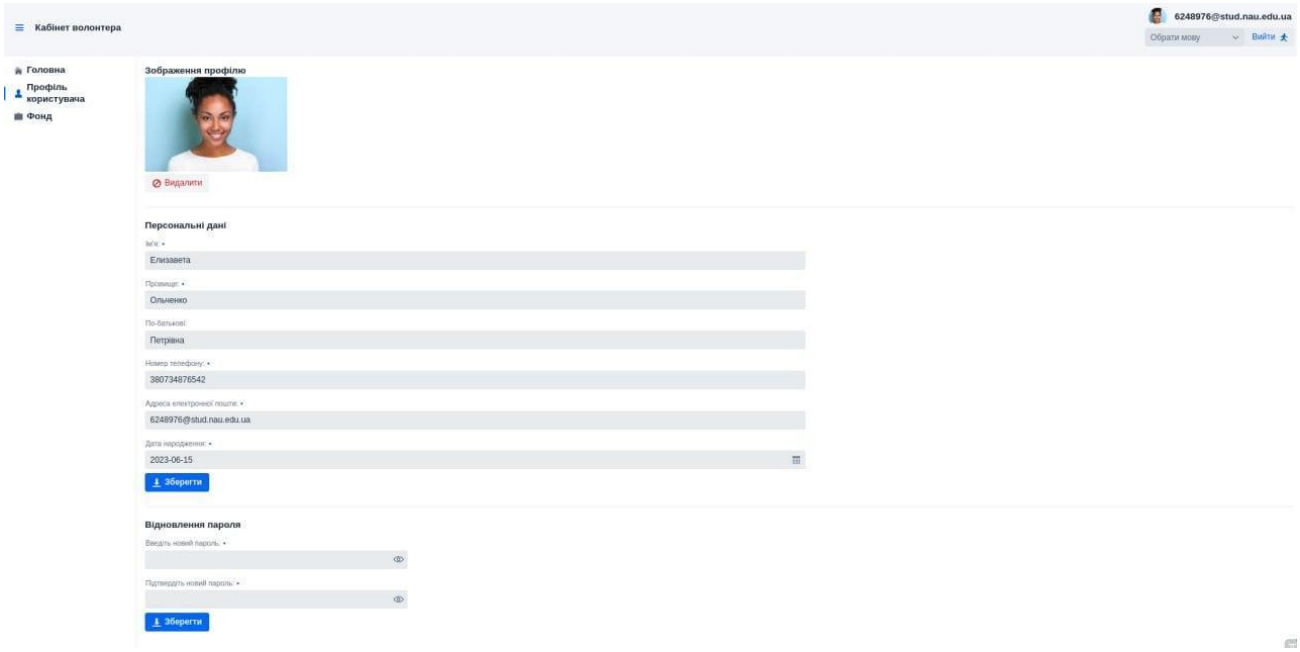


Рис. 3.19. Сторінка профілю користувача в кабінеті волонтера

У вкладці “Фонд” вам надана можливість створити власний фонд, надіслати заявку в уже існуючий або прийняти запрошення від одного з фондів. Всі заявки до фондів або запрошення будуть супроводжуватись емейл сповіщеннями. Сторінку вкладки фонду можна розглянути на рис. 3.20, рис. 3.21.

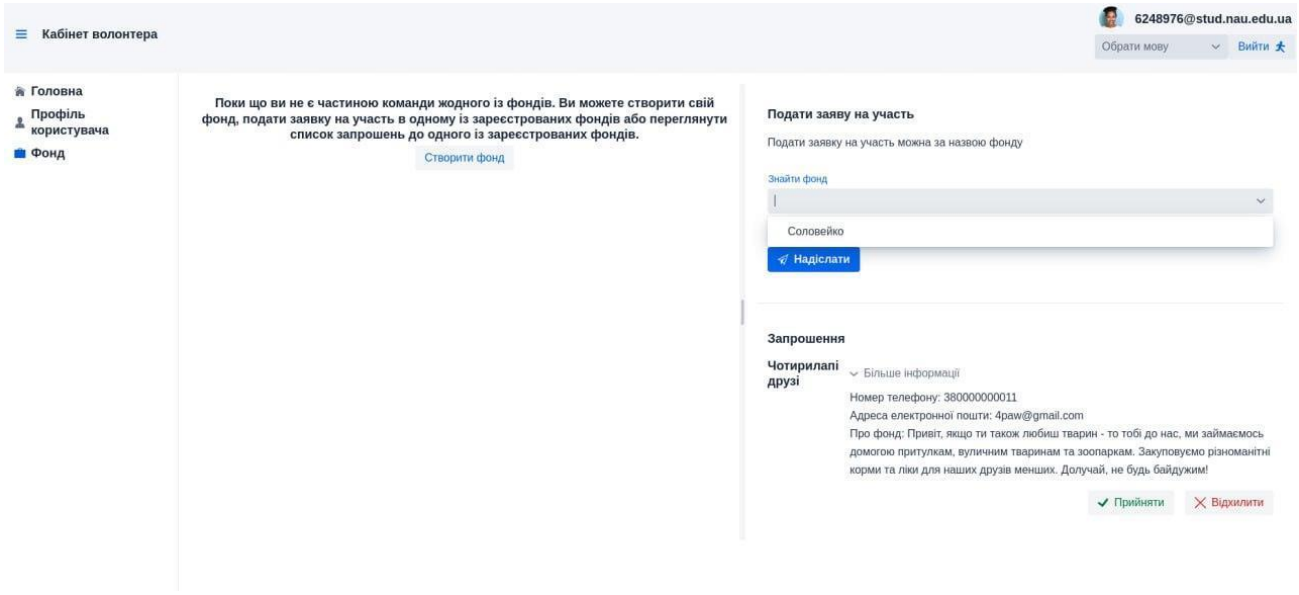


Рис. 3.20. Сторінка фонду в кабінеті волонтера у випадку якщо ви не є членом жодного з фондів

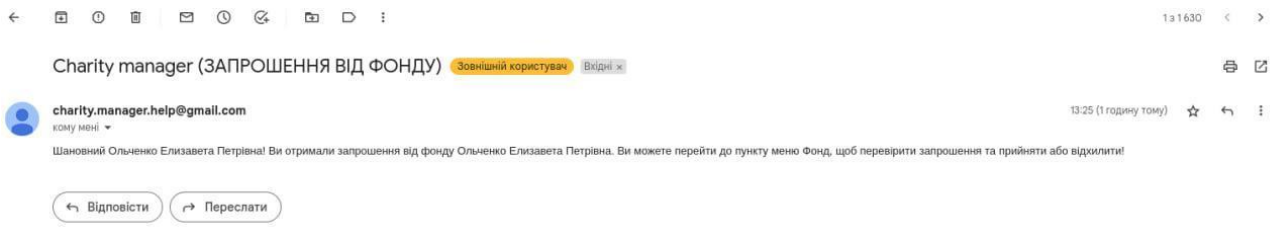


Рис. 3.21. Емейл сповіщення запрошення від фонду

Після того як ви зареєструєте свій фонд, або зареєструєтесь в одному з існуючих вам відкриються нові вкладки кабінету такі як “Команда”, “Заявки” та у вкладці “Фонд” ви побачите інформацію по фонду в якому зареєструвались. Профіль фонду можна розглянути на рис. 3.22.

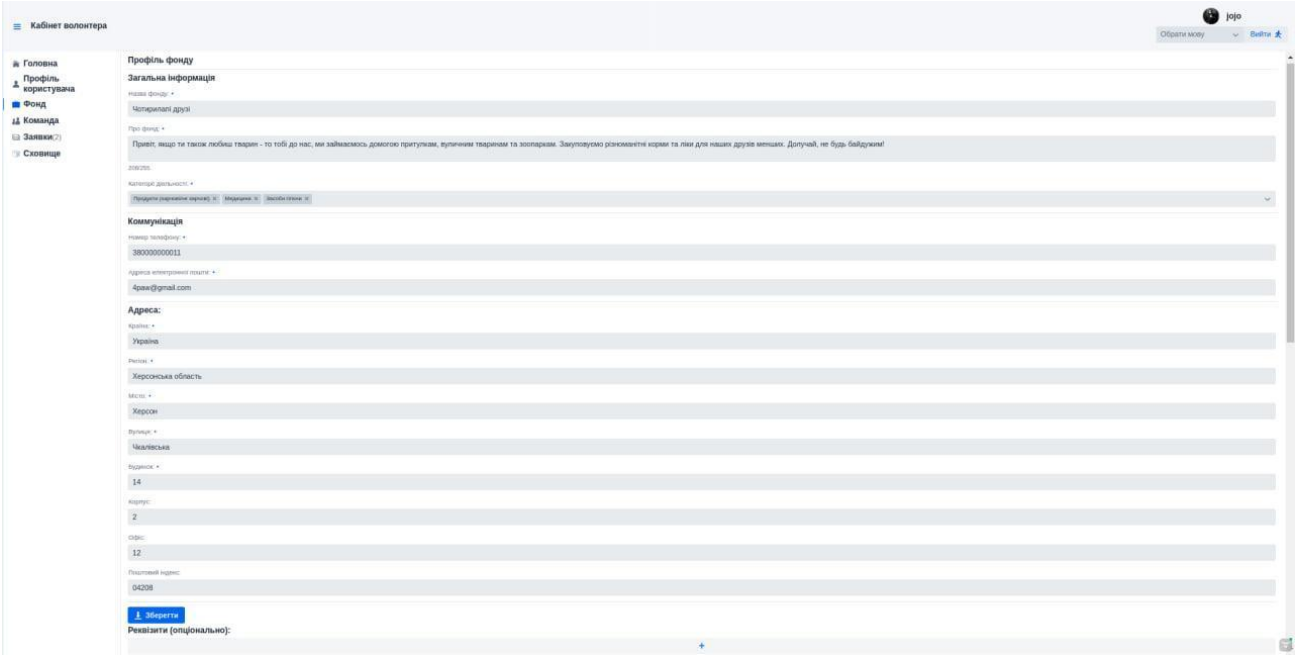


Рис. 3.22. Сторінка фонду в кабінеті волонтера у випадку якщо ви вже є членом одного з фондів

Після реєстрації в фонді необхідно взяти до уваги вкладку “Команда”, в ній якщо ви є власником фонду можете вносити корективи щодо команди. Видаляти поточних учасників та приймати заявки/запрошувати нових. А також переглядати інформацію по кожному члену команди. Сторінку команди фонду можна побачити на рис. 3.23.

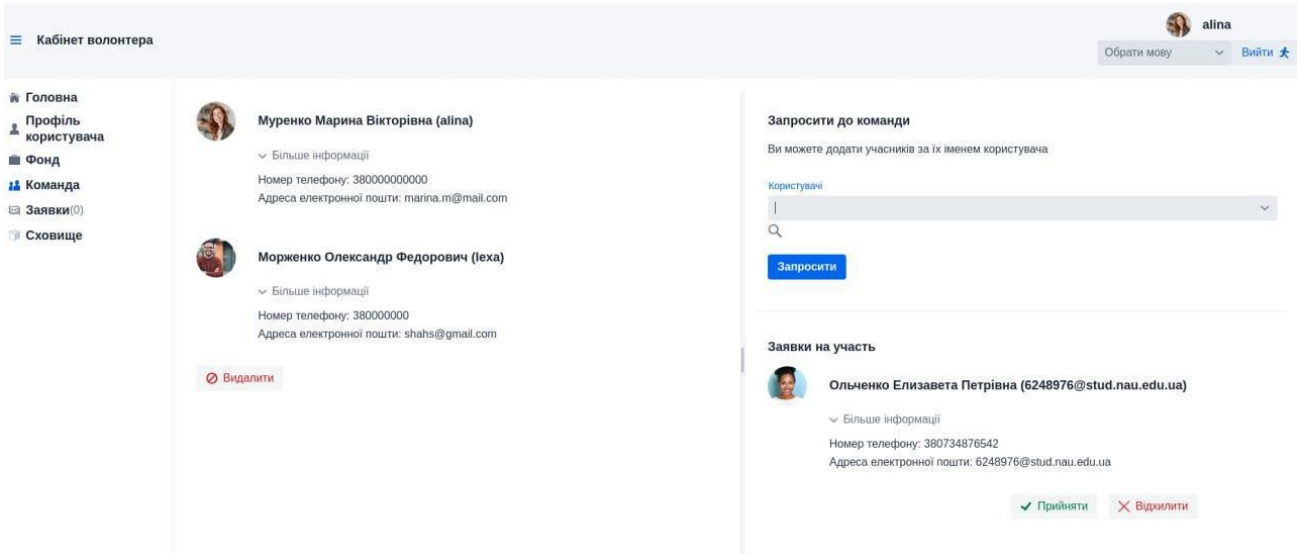


Рис. 3.23. Сторінка команди фонду

Біля вкладки “Заявки” буде відображатись кількість нових заявок. В самій вкладці ви зможете переглядати усі існуючі заявки по фонду та змінювати статус в будь-якій, сповіщуючи через емейл клієнта, який її лишив. Сторінку заявок від користувачів та зміни статусів можна розглянути на рис. 3.24, рис. 3.25.

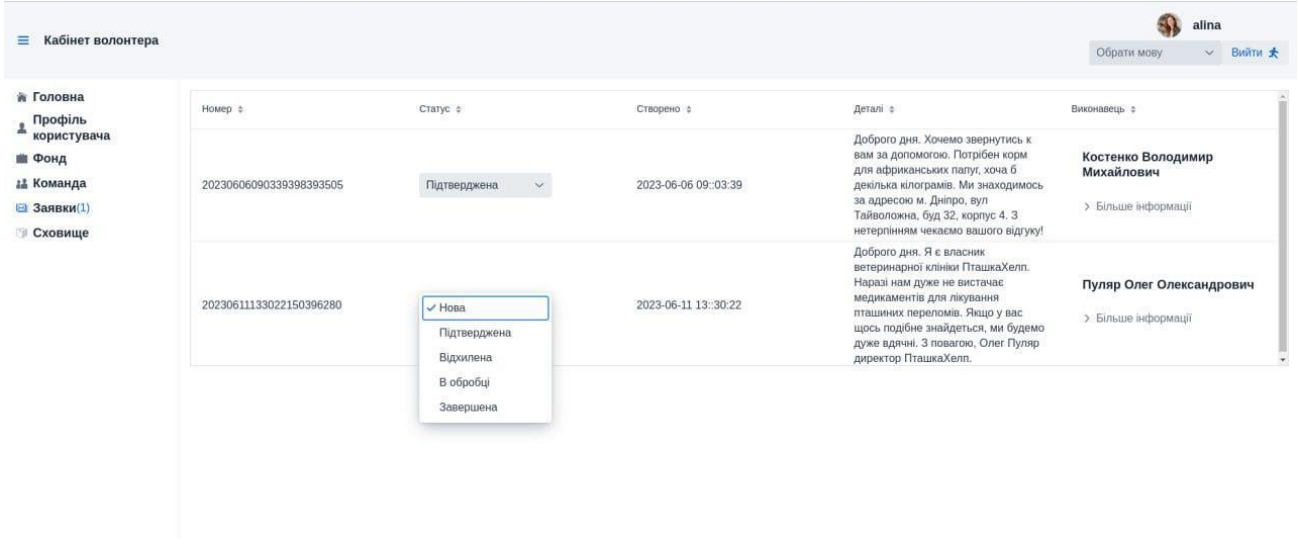


Рис. 3.24. Сторінка заявок фонду


Рис. 3.25. Емейл сповіщення про зміну статусу заявки на допомогу до фонду

## 3.8. Висновки до розділу 3

В цьому розділі був досліджений, проаналізований та визначений весь перелік функціоналу та бізнес-процесів з детальним описом кожного з них. Для більш детального розуміння життєвого циклу застосунку було наведено графічне зображення бізнес-процесів за допомогою діаграм Use-case. Пісял детального ознайомлення з усіма аспектами бізнес-логіки була сформована архітектура бази даних. Були визначені ключові сутності бази даних з їхніми атрибутами. Досліджені та назначені відповідні зв’язки між сутностями з детальним поясненням кожного. На основі сформованої бази знань був реалізований програмний код застосунку з аналізом деяких базових його компонентів. Після реалізації програмного коду було проведене детальне тестування більшої частини компонентів додатку, де на рисунках була представлена візуальна частина та інтерфейс.

# ВИСНОВКИ

Сьогодення в котрий раз доводить нам, що неабияк важливо в скрутну хвилину не залишатись осторонь та допомагати один одному. А також, кожен, хто просить, - отримає, хто шукає, - знайде. Цей принцип повинен супроводжувати вас все життя. І саме розробкою додатку менеджменту волонтерської діяльності ми зможемо наблизити кожну людину до цього кредо. В даному застосунку був використаний дуже нестандартний підхід до проектування веб-додатків. Симбіоз моноліту та клієнт-серверної архітектури. І на мою думку досить успішно. Було застосовано багато наук, новітніх технологій та архітектурних патернів побудови програмного забезпечення, такі як передпроектна аналітика проблематики продукту, аналіз вимог та визначення переліку функціоналу на основі потреб, проектування та візуалізація бізнес-процесів за допомогою діаграм та технології UML, проектування та графічне зображення баз даних, написання коду високорівневою мовою програмування, використовуючи великий перелік сучасних бібліотек і технологій, та масштабне тестування.

І саме у першому розділі ми провели детальний аналіз проблематики та вимог до продукту, зробили порівняльне дослідження конкурентного ринку та сформували висновки.

У другому розділі на основі аналітики потреб провели детальну ретроспективу алгоритмів та технологій, які будуть використовуватись в розробці.

Третій розділ містить в собі усі аспекти технічної розробки, а саме дослідження життєвого циклу додатку та усіх його бізнес процесів з різноманітними візуалізаціями та графічними зображеннями, написання коду та тестування.

Я вважаю, що саме цей додаток зможе зробити великий внесок у розвиток благодійної діяльності та суттєво спростити всі процеси, що стосуються цього,

та надасть суттєву оптимізацію для волонтерів та благодійників, а можливо комусь і врятує життя.

# СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Java». [Електронний ресурс]. – Режим доступу: https://www.oracle.com/cis/java/ (дата звернення 24.04.2023 р) — Назва з екрана.

2. «Spring Framework». [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-framework (дата звернення 27.04.2023 р) — Назва з екрана.

3. Vaadin Framework». [Електронний ресурс]. – Режим доступу: https://vaadin.com/docs/latest/guide/quick-start (дата звернення 01.05.2023 р) — Назва з екрана.

4. «Hibernate». [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/320542/ (дата звернення 05.05.2023 р) — Назва з екрана.

5. Харченко О. Г. Організація баз даних і знань: Лабораторний практикум. Модулі I, II./Уклад.: / О.Г.Харченко, І.Е.Райчев. – К.: НАУ, 2006 – 53 с.

6. Postgresql Database». [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення 08.05.2023 р) — Назва з екрана.

7. Design Patterns». [Електронний ресурс]. – Режим доступу: https://refactoring.guru/uk/design-patterns (дата звернення 10.05.2023 р) — Назва з екрана.