Національний лісотехнічний університет України
(повна найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(освітньо-кваліфікаційний рівень)

на тему: **«Створення вебсистеми для залучення волонтерів до міжнародних волонтерських проєктів на основі ASP.NET та Angular»**

Виконала: студентка IV курсу, групи КН-41

Спеціальності 122 – "Комп'ютерні науки"

(шифр і назва напряму підготовки, спеціальності)

Наливайко Д. І.

(прізвище та ініціали)

Керівник: Яцишин С. І.

(прізвище та ініціали)

Керівник: Волинець Є. О

(прізвище та ініціали)

Рецензент: Олуд Л.О.

(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Борецька І. Б.
"10" червня 2025 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ

Наливайко Діані Іванівній

(прізвище, ім'я, по батькові)

1. Тема роботи: "Створення вебсистеми для залучення волонтерів до міжнародних волонтерських проєктів на основі ASP.NET та Angular "
Керівник роботи Яцишин С.І., К.Т.Н., доцент, Волинець Є.О. Асистент кафедри ІПЗ
Затверджені наказом вищого навчального закладу від 15 листопада 2024 року №С-882
2. Термін подання студентом роботи 10 червня 2025 р.
3.Вихідні дані до роботи "Створення вебсистеми для залучення волонтерів до міжнародних волонтерських проєктів на основі ASP.NET та Angular "
4.Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ.
Розділ 1. Опис предметної області.
Розділ 2. Інформаційне та математичне забезпечення.
Розділ 3. Програмно-технологічне забезпечення.
Висновки. Список використаних літературних джерел.
5. Перелік графічного матеріалу: презентація *для доповіді.*
6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних даних	19.11-31.12.2024	виконано
2	Розділ 1. Стан проблемної області	01.01-10.03.2025	виконано
3	Розділ 2. Інформаційне та математичне забезпечення	11.03-20.03.2025	виконано
4	Розділ 3. Програмне та технічне забезпечення	21.03-30.04.2025	виконано
5	Оформлення дипломної роботи	01.05-31.05.2025	виконано
6	Підготовка до захисту дипломної роботи, оформлення презентації	01.06-13.06.2025	виконано

Студентка _____ *Наливайко Д. І.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Яцишин С. І.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Волинець Є. О.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 69 сторінок пояснювальної записки, 16 зображення, 11 джерел, 1 додаток.

В дипломній роботі спроектовано і реалізовано вебсистему відповідно до технічного завдання. Вона розроблена засобами мови програмування C# з використанням ASP.NET для бекенд частин та Angular для клієнтської частини. Тут було реалізовано прикладний програмний інтерфейс REST API, який забезпечує швидку взаємодію між клієнтом і сервером. Клієнтський інтерфейс забезпечує сучасну платформу для потенційних волонтерів та організацій. Розроблена система дозволяє ефективно координувати волонтерську діяльність, подачі заявок та взаємодії між волонтерами і організаціями.

Ключові слова: ASP.NET, Angular, C#, REST API, волонтерство, міжнародні проєкти, вебсистема, автентифікація, авторизація, PrimeNg, EF.

ABSTRACT

The diploma project contains 69 pages of explanatory report, 16 images, 11 sources, and 1 appendix.

In this thesis, a web system was designed and implemented in accordance with the terms of reference. It was developed in C# programming language using ASP.NET for the backend parts and Angular for the client side. The REST API application program interface was implemented here, which provides fast interaction between the client and the server. The client interface provides a modern platform for potential volunteers and organizations. The developed system allows for effective coordination of volunteer activities, applying and interacting between volunteers and organizations.

Keywords: ASP.NET, Angular, C#, REST API, volunteering, international projects, web system, authentication, authorization, PrimeNg, EF.

ТЕХНІЧНЕ ЗАВДАННЯ

В дипломній роботі потрібно впровадити програмне рішення, що допоможе залучати волонтерів до міжнародних волонтерських проєктів. В рамках роботи потрібно:

- проаналізувати літературу, нормативні документи та подібні існуючі рішення у сфері волонтерських платформ;
- виявити та застосувати оптимальних архітектурних підходів до побудови сучасної вебсистеми;
- розробити серверну частину системи за допомогою фреймворку ASP.NET (C #);
- розробити клієнтську сторону веб-додатку за допомогою Angular;
- забезпечити аутентифікацію користувача та авторизацію з розмежуванням прав доступу (волонтер/організація/адмін).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface.

REST — Representational State Transfer.

SPA — Single Page Application.

ASP.NET — фреймворк для розробки веб-додатків на платформі .NET.

Angular — фреймворк для створення клієнтських веб-застосунків на мові TypeScript.

C# — об'єктно-орієнтована мова програмування, для .NET.

JWT — JSON Web Token.

GitHub — платформа для спільної роботи з частинами коду, яка підтримує Git.

CRUD — Create, Read, Update, Delete.

EF — Entity Framework.

ООП — об’єктно-орієнтоване програмування.

IDE — Integrated Development Environment.

WebStorm — IDE від компанії JetBrains.

ВСТУП

Суспільство живе в період численних гуманітарних викликів, що потребують швидкої та чисельної підтримки громадян. Волонтерський рух зараз потребує більше уваги та підтримки ніж коли-небудь. Воєнні конфлікти, природні та техногенні катастрофи напружують ситуацію в світі, тому роль волонтерства є важливою в суспільстві. У зв'язку з цим є потреба в створенні платформи для координації та підтримки для покращення міжнародних волонтерських ініціатив. Ця дипломна робота створена для розробки продукту, що зможе покращити залученість волонтерів до проектів.

Актуальність дипломної роботи. В умовах повномасштабної війни в Україні піднялася потреба в швидкому залученні волонтерів до гуманітарної, соціальної та логістичної підтримки як всередині держави, так і за кордоном. Управління волонтерської діяльності часто ускладнюється через відсутність зручних цифрових інструментів для пошуку можливостей, подачі заявок та обміну інформацією між добровольцями та організаціями. Створення платформи для додавання волонтерів до міжнародних проектів є актуальним та неодмінним кроком до цифровізації волонтерської діяльності, зміцнення міжнародної комунікації та ефективного використання людських ресурсів у важкий час для країни.

Метою проекту є розробка платформи, що забезпечить набір бажаючих для волонтерських проектів, яка надасть сучасний та зрозумілий у використанні веб-інтерфейс, легку взаємодію між користувачами та організаціями і безпечну обробку персональних даних.

Об'єктом дослідження у даній роботі є процес створення застосунку для цифрової взаємодії для координації волонтерських ініціатив, за підтримки закордонних організацій.

Предметом дослідження є структурування для побудови програмного рішення для вебсистеми, що допоможе полегшити управління для волонтерських ініціатив використовуючи новітні технології, такі як ASP.NET, Angular, REST API тощо.

Завдання дипломної роботи:

- проаналізувати технічні відомості для цієї теми, ретельно переглянути продукти, які вже існують в цьому напрямку;
- сконструювати архітектурну частину;
- розробити серверну частину за допомогою фреймворку ASP.NET (C #);
- розробити інтерфейс для взаємодії користувачів використовуючи Angular;
- провести тестування створеного продукту, та покращити його функціонал, якщо потрібно;

Результатом даної роботи має бути проект, що полегшить зв’язок небайдужих та організацій, що займаються підтримкою населення. Значимість цього проєкту є великою, оскільки він допоможе швидше реагувати на проблеми людей та надавати потрібну підтримку кожному, полегшить комунікацію між бажаючими та забезпечить швидке наймання волонтерів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Вимоги для волонтерської вебсистеми у сучасному суспільстві

У теперішній час роль волонтерства є надзвичайно важливою. Кожного дня на долю людей випадає багато випробувань, тому така допомога дає можливість змінювати життя людей та робити світ кращим. Волонтери - це люди, які надають допомогу за власним бажанням, вони не отримують за це винагороду. Їхня роль у сучасному суспільстві є неоціненною.

Переглядаючи сайти для волонтерства, я зіштовхнулася з проблемою: в українському просторі є дуже мала кількість таких платформ, а ті, що вдалось знайти, - незручні у користуванні, з застарілим і обмеженим функціоналом.

Тому було прийняти рішення створити веб-систему, яка могла б пришвидшити пошук кандидатів, полегшила роботу з організаціями через зручне фільтрування, а також забезпечила їхню перевірку за допомогою відгуків.

Основний функціонал, на мою думку, мав би включати наступне:

З боку організації:

- зручний функціонал для заповнення форм з інформацією про себе;
- пришвидшений пошук кандидатів для волонтерських робіт завдяки відгукам та рейтингам від попередніх організацій;
- можливість завантаження фотографій для привернення уваги з боку користувачів, що може позитивно впливати на рейтинг;
- структуроване фільтрування кандидатів за віком, містом, сферою діяльності, рейтингом тощо.

З боку волонтера:

- зручний функціонал для заповнення власного профілю;
- сучасна система фільтрації організацій за реальними відгуками: залишати відгуки можуть лише ті волонтери, яких було прийнято на волонтерську пропозицію;

- можливість залиши відгук про організацію після досвіду співпраці;
- не боятися шахрайських організацій, оскільки розроблений функціонал додав контроль з боку адміністраторів із можливістю приховувати або видаляти підозрілі організації.

1.2 Порівняння функціоналу існуючих аналогів

1. Volunteering Ukraine:

- Volunteering Ukraine платформа, де користувачі можуть шукати актуальні проєкти за різними критеріями, наприклад тип допомоги, організація, країна;
- Всі оголошення в Volunteering Ukraine містять інформацію про дату, умови та опис потреб;
- У Volunteering Ukraine волонтери можуть створитиакаунти та подавати заявки на участь у проектах;
- Тут є можливість фільтрування по містах та напрямках діяльності;
- На сайті є детальна інформація для новачків, а саме: як долучитися, які ризики, які речі необхідно взяти з собою;

2. Volunteer World:

- На цій платформі користувачі можуть знаходити цікаві програми за тривалістю, сферою діяльності, ціною. Вони можуть вказувати свій рівень досвіду, і платформа підбере найкращий варіант.
- Усі програми мають персональні сторінки з детальною інформацією, де розміщенні фото та відео з місць проектів;
- Є можливість фільтрувати відгуки та оцінки по кожній організації;
- Забезпечений функціонал для оплати внесків за участь;
- Волонтери можуть оцінювати досвід, який вони отримали після проекту на сторінці організації;

- Додаток використовує цей сайт для формування рейтингу програми, і це допомагає іншим при виборі наступної програми;

3. International Volunteer:

- Платформа надає можливість зареєструватися приблизно на 300 програм у 50 країнах світу;
- Усі користувачі можуть обирати різні локації в залежності від їхнього розташування;
- У кожної програми є своя сторінка з детальною інформацією, де вказано: умови проживання, вартість, інформацію про супровід та розпорядок дня, усюди додано фотографії та відео-відгуки учасників, а також інструкції до участі;
- Також International Volunteer надає безкоштовний курс-підготовку для волонтерської подорожі, дає на вибір координатора, який буде підтримувати під час всієї поїздки;
- Надається функціонал для оформлення страхування, трансферу з аеропорту, турів та екскурсій;
- Кожен бажаючий повинен при створенні акаунту обрати програму та сплатити вступний внесок (від ~$299);
- Після завершення програми учасник отримує сертифікат;

4. UN Volunteers:

- Сайт пропонує три волонтерства: міжнародне, національне, онлайн-волонтерство;
- Має доступ до проектів у більше ніж 125 країнах світу;
- Користувач повинен пройти реєстрацію, де вказує освіту, досвід роботи, мову, вибір напрямку зацікавленості;
- Усі бажаючі після успішної реєстрації на проект отримують щомісячну стипендію, медичне страхування, проїзд;

- Координатори на цій платформі закріпленні за кожним учасником;
- Ця платформа є некомерційною, а більш професійною, де до кандидатів є високі вимоги;

5. Worldpackers:

- Worldpackers є платформою, де волонтери допомагають в обмін на безкоштовне житло та харчування;
- Головними напрямками є: підтримка в хостелах, екопроєктах, школах, фермах;
- Мають пропозиції в понад 135 країнах світу;
- Користувачі можуть шукати можливості за країною, типом проєкту, тривалістю. Є фільтри по рівню досвіду та відгуках;
- Щоб подати заявку, потрібно створити профіль і оформити платну підписку;
- Платформа надає страхування "WP Insurance", яке покриває непередбачувані ситуації;
- Волонтери можуть проходити онлайн-курси (WP Academy) і отримувати сертифікати, що підвищують шанси на участь у кращих проєктах;
- У профілі волонтера зберігається історія подорожей, відгуків і досягнень;

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання структури за допомогою Entity Framework

Entity Framework (скорочено EF) - це фреймворк, створений Microsoft, для того, щоб полегшити взаємодію між класами в .NET та таблицями в SQL. Використовуючи його, девелоперам не потрібно буде писати sql-запити, вони зможуть керувати таблицями користуючись класами. EF- це простий та досить надійний інструмент, що може надати легкий спосіб керування.

EF був створений для Visual Studio 2008 року, датою випуску можна вважати 11 серпня 2008 року. Після виходу в світ багато розробників відмовилися співпрацювати з ним через низку критичних відгуків, але з версії 4.1 були виправленні помилки, і вже більша кількість користувачів почала використовувати цей продукт.

Головними перевагами є:

- підтримка Windows, Linux та macOS;
- спрощення виконання операцій CRUD, а саме: створення, читання, оновлення та видалення завдяки підтримці баз даних;
- на EF можна писати міграції, що виконуються з консолі NuGet Package Manager;
- завдяки EF можна швидко створювати модель даних та взаємодіяти з нею без написання SQL-запитів;
- легкість керування змінами структури бази без ручного написання SQL;
- використання LINQ зменшує кількість помилок та покращує читабельність коду;
- здатність працювати як і з реляційними, так і нереляційними базами даних, а саме: SQLite , PostgreSQL, Azure Table Storage, SQL Server, SQL Server Compact та інші;
- дозволяє підтримувати складні зв’язки: надає можливість працювати з відношеннями one-to-many, many-to-many, one-to-one;

Важливою частиною бази даних є побудова логічних зв’язків. Entity Framework дозволяє створювати різні типи зв'язків між сутностями. Як згадувалося раніше, в EF реалізовано 3 зв’язки: one-to-many, many-to-many, one-to-one.

Зв’язок one-to-one - це зв’язок, що створюється для поєднання тільки 1 запису до 1 запису в іншій таблиці. У кожної людини можуть бути лише 1 права, так і в 1 прав може бути лише 1 власник. Цей тип зв’язку корисно використовувати, щоб зменшити дублювання даних та розділити інформацію на окремі таблиці.

Це можна розглянути на одному прикладі: людина та її права на водіння.

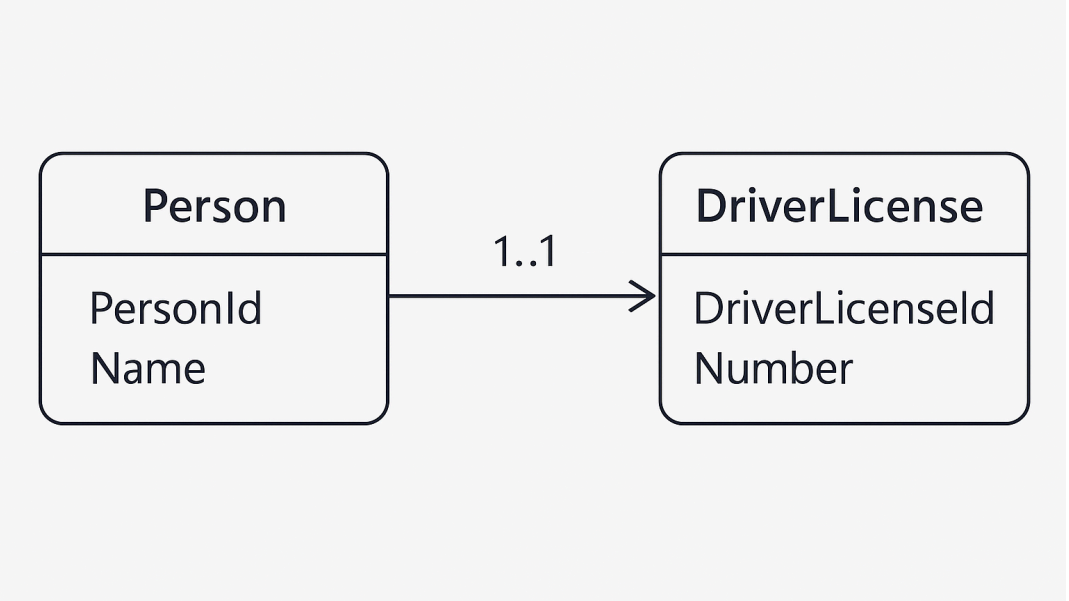

Рисунок 2.1 – Схема зв’язку one-to-one

На Рис. 2.1 ми можемо побачити 2 сутності: Person (людина) та DriverLicense (Водійські права). Цей зв’язок демонструє, що одна людина може мати лише одні водійські права, та кожні водійські права належать лише одній людині. Щоб реалізувати це в базі даних потрібно в таблиці Person

встановити зв’язок за допомогою спільного поля PersonId як зовнішній ключ в DriverLicense.

Наступним розглянемо зв’язок one-to-many. Цей взаємозв’язок потрібний, коли ми хочемо пов’язати 1 запис з багатьма в іншій таблиці.

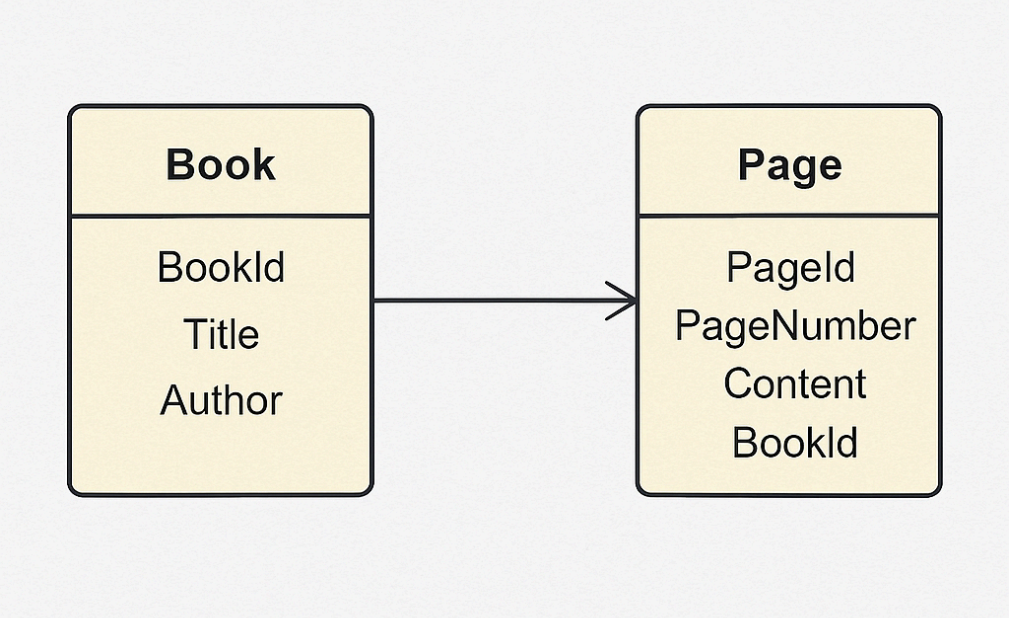

Рисунок 2.2 – Схема зв’язку one-to-many

На Рис. 2.2 відображено 2 сутності: Book (книга) та Page (сторінка).Такий зв’язок передбачає, що одна книга включає кілька сторінок, але кожна сторінка пов’язана лише з однією конкретною книгою. Він допомагає забезпечити логічну цілісність даних та ефективно створювати запити.

І останній зв’язок - many-to-many.

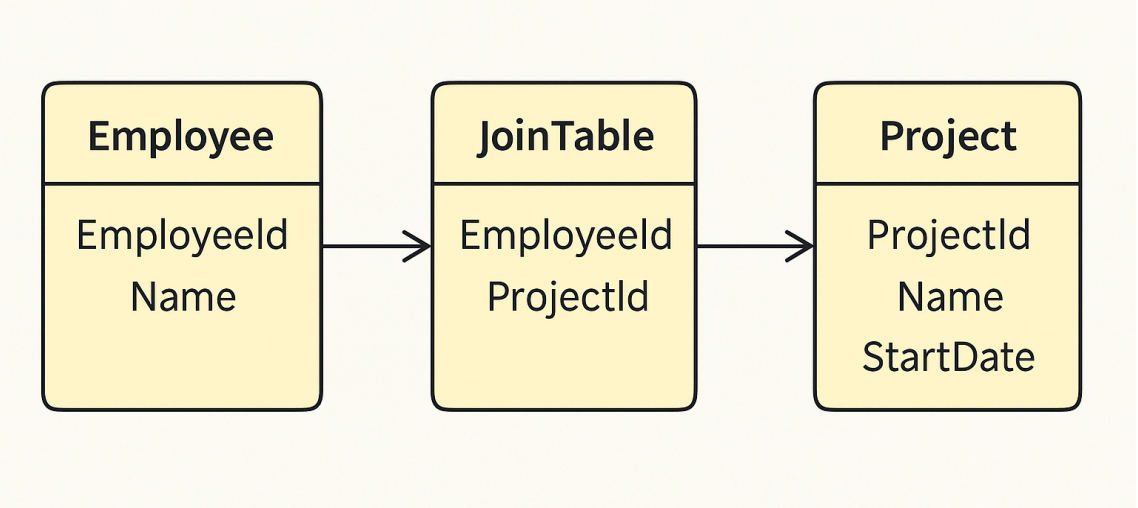

Рисунок 2.3 – Схема зв’язку many-to-many

На Рис. 2.3 зображено зв’язок, який показує, що один працівник може брати участь у кількох проєктах, і один проєкт може включати багато працівників. Цей зв’язок реалізовано через проміжну таблицю, яка містить два зовнішні ключі: EmployeeId та ProjectId. Many-to-many слід використовувати, коли кожен об’єкт однієї сутності може бути пов’язаний з багатьма об’єктами іншої сутності, і навпаки.

Також EF підтримує різні моделювання баз даних, що зображено на Рис. 2.4: Database First, Code First або Model First. Усі ці методи мають свої переваги, і це дозволяє розробникам обрати найбільш зручний варіант.

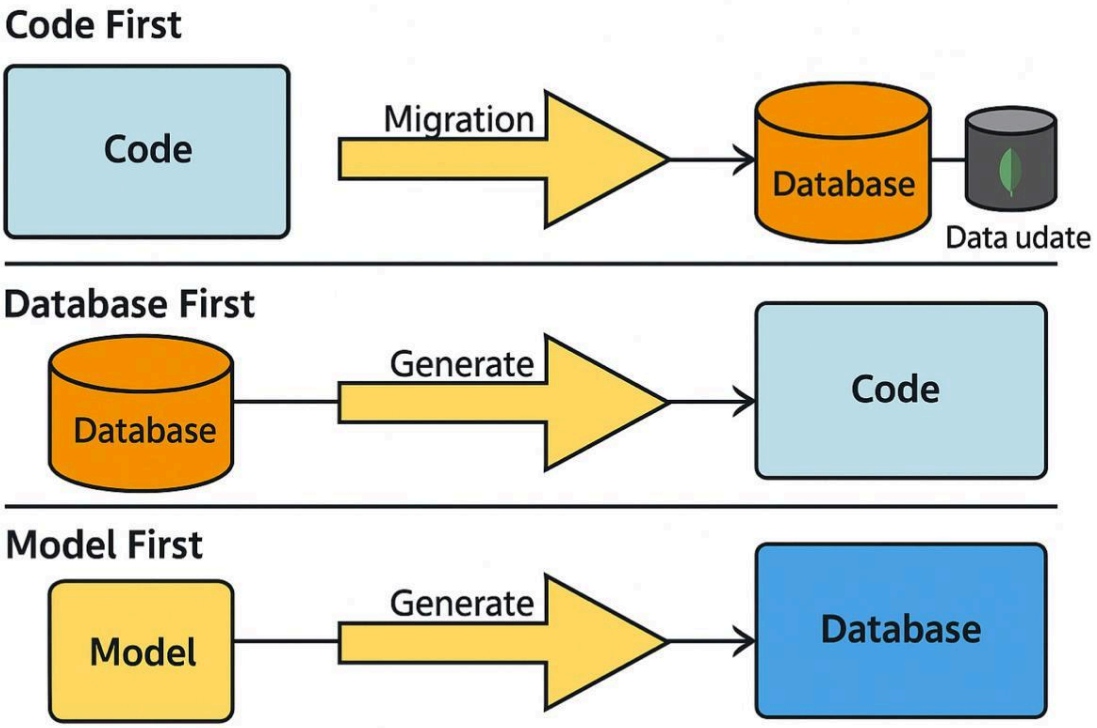

Рисунок 2.4 – Архітектурні підходи у розробці з використанням EF

Code First спочатку створює класи моделі в коді, а після цього база даних сама генерується на їх основі. Цей підхід дуже корисний на початкових стадіях проекту, коли база даних ще не створена. Структура бази даних створюється шляхом міграції. Це дає розробнику можливість повністю контролювати логіку, типи даних і зв'язками між сутностями, без необхідності вручну писати SQL-структури.

В Database First навпаки спершу створюється база даних, а далі вже на її основі генерується код моделі. Це корисно, коли база даних вже існує. Entity Framework створює потрібні класи на основі таблиць, спрощуючи інтеграцію нового додатку з існуючими даними.

Model First використовує графічну модель. Після цього код і база даних автоматично генеруються на основі цієї діаграми. Це є рішенням для для проектів, в яких вже існує візуальне відображення на початку розробки.

2.2 Typescript

У програмуванні, де проекти стають все складнішими та складнішими, і вимоги до безпеки збільшуються, TS забезпечує високу надійність та маштабованість. Typescript зменшує виникнення людських помилок завдяки строгій типізації, що допомагає компаніям збільшувати швидкість написання хорошого коду.

Typescript (TS) - це мова програмування, що створена Microsoft для масштабованих програмних продуктів. Це javascript з підтримкою строгої типізації, що полегшує знаходження помилок на етапі компіляції.

Ця мова програмування була запущена в 2012 році для великих компанії, щоб забезпечити можливість розробляти маштабовані додатки для великої кількості користувачів. Сьогодні Typescript активно використовується великою кількістю програмістів і це не дивно, оскільки вона надає багато переваг:

- швидке виявлення помилок на етапі компіляції: завдяки строгій типізації, компілятор TS перевіряє, чи правильно записані всі типи даних у всіх змінах, параметрах та поверненнях функцій. Тому це допомагає визначити потенційні помилки відразу. Від цього покращується ефективність та надійність;
- підтримка сучасних IDE: TS відмінно працює як і з Visual Studio Code, так і з WebStorm;
- код стає більш чистим та читабельним;
- забезпечує розуміння між командою, оскільки кожен програміст знає, який метод який тип повертає;
- підтримує ООП [9]: надає відомі парадигми для структуризації коду, а саме: модифікатори доступу, класи, інтерфейси, наслідування тощо.

Як і всі мови програмування TS також має і недоліки:

- на початку знайомства з цією мовою можуть виникнути труднощі, бо синтаксис може здаватися складним через типи, інтерфейси та інші поняття, які не зустрічаються у звичайному JavaScript;
- невеликий код може перерости у надмірний через наявність типів та додаткових конфігурацій;

TS перетворюється на JS за допомогою компілятора, що має назву tsc. Як це працює, коли програміст створює файл (.ts), компілятор зчитує цей код, перевіряє чи всі змінні відповідають типам і тоді перетворює його в JS. Процес трансформації відбувається так: видаляються типи, синтаксис, який ще не підтримується в JS, перетворюється на знайомий для JS.

Typescript використовується для фронтенд розробки:

- Angular, що є повністю побудований на TS;
- React часто використовує для типізації компонент;
- Vue (з v3+) офіційно підтримує TS;
- Svelte - підтримується при використанні плагінів;

Typescript використовується для бекенд розробки:

- NestJS - сучасний фреймворк, що повністю побудований на TS;
- Express - надає підтримку TS через типи з @types/express;
- Next.js - фреймворк для SSR, повністю підтримує TS на фронтендф та бекенді;

Typescript часто використовують у мобільній розробці:

- React Native;
- Ionic + Angular;
- NativeScript;

Typescript використовується для десктопної розробки:

- Electron - для створення кросплатформених застосунків;
- Tauri;

Typescript у своїх додатках використовують великі компанії: Microsoft, Google, Slack, Airbnb, GitHub, Asana, Uber, Netflix, Shopify. Так як однією з ключових особливостей TS є типізація, розглянемо її.

Типізація в TS є статичною, а це значить, що типи перевіряються до виконання програми. Typescript підтримує як вбудовані примітивні типи, так і складні користувацькі типи [4]. До примітивних типів можна віднести:

- string - рядок тексту, приклад оголошення коду: const username: string = ‘Diana’;
- number - числове значення, приклад оголошення коду: const age: number = 21;
- boolean - булеве значення, приклад оголошення коду: const isStudent: boolean = true;
- null, undefined - значення, які використовують для відсутності та невизначеності, приклад оголошення коду: const Adult: null | undefined = null;

  Масиви та кортежі:
- масив: const ages: number[] = [21, 22, 31];
- кортеж: const user: [string, number] = [‘Serge’, 50]. Кортеж має фіксовану кількість елементів і типи.

Оскільки TS підтримує ООП, то в мові програмування реалізована можливість розбиття на функціональні блоки, повторне використання та маштабованість додатків. Особливим механізмом для цього є інтерфейси. Вони слугують для опису структури об’єктів, де вказують на властивості або методи, що мають бути реалізованими. Головною відмінністю класів та інтерфейсів є те, що інтерфейси не компілюються у JavaScript і не мають впливу на виконання коду. Вони є лише структурою. Це дуже важливо, коли працюєш в команді, оскільки ми можемо знати, які дані повернуться з API.

Крім інтерфейсів, TS має підтримку класів, що є головний механізмом для створення об’єктів із властивостями, функціями, модифікаторами

доступу (private, protected, public). Інтерфейси потрібні для упорядкування структури даних додатку. Вони дають ясність, що саме повинне бути в об’єкті, які типи та поля. А також класи можуть реалізовувати інтерфеси: тобто інтерфейс показує, що повинне бути, а клас це реалізовує. Завдяки цьому ми може редагувати логіку та швидко розширити, якщо потрібно, при тому не ламаючи існуючої частини. Тому завдяки інтерфейсам і класам, код зрозумілий і зручний у підтримці.

2.3 Бази даних

Однією з ключових частин проекту є бази даних. Вони зберігають усю інформацію, що потрібна для продукту, наприклад інформація про користувачів, сутності з якими пов’язаний проект. Їх використовують для підтримки зберігання, доступу та аналізу даних. Вони керуються за допомогою Системи керування базами даних (СКБД). В програмуванні існує багато типів баз даних: вони відрізняються за типом вмісту та за організаційним підходом. В основному вони поділяються на реляційні, хмари, NoSQL, графові, ключ-значення.

Реляційні бази даних - це базуються на принципі зберігання даних у таблиці, де зв’язок сформовано за допомогою ключів, які поєднують таблиці. До переваг такого типу відносять:

- зручність для складних sql запитів;
- цілісність зберігання даних;
- робота з мовою SQL;
- підтримка транзакцій;
- уникнення дублювання даних;
- права доступу до таблиць, запитів;
- велика підтримка фреймворків;
- підходить для аналітичних завдань;

До недоліків такого типу відносять:

- не підходить для роботи з неструктурованою інформацією;

NoSQL - це бази даних, в яких дані зберігаються як документи (JSON/BSON). В цьому типі БД дані зберігаються без гнучкої структури та строгої схеми, як це було в реляційних БД. Часто використовують в веб-застосунках, де структура змінюється. До переваг такого типу відносять:

- гнучкість;
- високу швидкість;
- простоту масштабування;

До недоліків такого типу відносять:

- складність у перевірці зв’язків;
- складно керувати структурою;

Графові БД зберігають інформацію у вигляді вузлів та зв’язків. Така структура полегшує представлення складних взаємозв'язків і забезпечує підвищення ефективності при використанні. Головною відмінністю від реляційних БД є обробка інформації: зв’язки тут постають у вигляді ребер графа. До переваг такого типу відносять:

- швидкий пошук зв’язків між сутностями;
- ідеальний варіант для складних взаємозв’язків;
- гнучкість системи;
- мова запитів, адаптована

2.4 Архітектура клієнт-серверних застосунків

Архітектура програмного забезпечення є важливою складовою будь-якого ІТ-проєкту, оскільки визначає загальну структуру системи, принципи взаємодії її компонентів, рівень масштабованості, підтримуваності та надійності.

Одним із базових архітектурних рішень є монолітна архітектура, що передбачає об’єднання всієї бізнес-логіки, інтерфейсу користувача та доступу

до даних в одному застосунку. Такий підхід є досить простим у реалізації, однак ускладнює масштабування та супровід при зростанні проєкту.

Широко поширеним підходом у веб-розробці є клієнт-серверна архітектура, у якій клієнтська частина (наприклад, створена на Angular) відповідає за інтерфейс та взаємодію з користувачем, тоді як серверна частина (наприклад, .NET) обробляє запити, реалізує бізнес-логіку та забезпечує доступ до бази даних. Такий розподіл дозволяє досягти кращої масштабованості та спрощує підтримку окремих частин системи.

Ще одним ефективним підходом є трикаскадна архітектура (Three-layer architecture), що поділяє застосунок на три рівні: презентаційний (Presentation Layer), бізнес-логіки (Business Logic Layer) та доступу до даних (Data Access Layer). Це дозволяє чітко розмежовувати функціональність, підвищує модульність системи та полегшує її тестування.

У сучасних високонавантажених системах також активно застосовується мікросервісна архітектура, у межах якої кожна бізнес-функція реалізована у вигляді окремого сервісу. Такий підхід забезпечує високу гнучкість та можливість незалежної розробки і масштабування окремих компонентів, однак потребує складнішої інфраструктури для їхньої координації.

На фронтенді активно використовується компонентна архітектура, притаманна фреймворку Angular. У цьому підході логіка додатку розбивається на окремі багаторазово використовувані компоненти, що сприяє повторному використанню коду, спрощує тестування та підвищує підтримуваність.

Інші підходи, які можуть застосовуватись у певних ситуаціях, включають архітектуру на основі подій (event-driven), Serverless-архітектуру, а також шаблони проектування типу MVC та MVVM, які використовуються для кращої організації логіки та взаємодії між частинами застосунку.

Таким чином, вибір архітектурного підходу залежить від складності проєкту, обсягу функціональності, очікуваного навантаження та вимог до масштабованості й підтримки.

В своєму проекті я організувала класичну клієнт-серверну архітектуру, де клієнтська частина створена на Angular, а серверна частина — на .NET.

Архітектура проєкту побудована так, щоб забезпечити масштабованість, простоту підтримки та логічне розділення відповідальностей між компонентами.

Користувач взаємодіє з веб-додатком через Angular-інтерфейс. Усі дії (наприклад, авторизація, заповнення форм, перегляд даних) спрямовані на сервер за допомогою HTTP-запитів. Вони обробляються .NET API, який відповідає за бізнес-логіку, обробку запитів і взаємодію з базою даних через Entity Framework.

Серверна частина структурована на основі принципів розділення обов’язків (Separation of Concerns): контролери відповідають за прийом запитів, сервіси — за бізнес-логіку, а репозиторії — за доступ до БД. Такий підхід дозволяє легко тестувати окремі частини системи та розширювати функціонал без порушення загальної логіки.

Клієнтська частина також структурована на компоненти, сервіси та маршрути. Кожна сторінка реалізована у вигляді окремого компонента, що полегшує підтримку. Для запитів до API використовуються Angular-сервіси, які централізовано обробляють логіку передачі даних.

Усі дані зберігаються в реляційній базі даних, до якої звертається сервер через Entity Framework. Для зберігання сутностей використовуються класи-моделі, які одночасно служать і для валідації, і для формування таблиць у БД через Code First підхід.

2.5 Клієнтська частина, створена на основі Angular

Angular - це новітній фреймворк для розробки сайтів, який був використаний в моєму проекті. Його створила команда Google у 2010 році, на мові TypeScript. Angular підтримує розробку масштабованих, структурованих та зручних у підтримці вебзастосунків. Він надає великий набір готових інструментів і рішень, які значно прискорюють роботу.

Як і у всіх фреймворках, в Angular є свої переваги та недоліки. Почнемо з переваг. В Angular використовують TypeScript, що значно полегшує роботу з кодом, оскільки дозволяє виявляти помилки ще на етапі компіляції [4]. Це робить розробку безпечнішою та передбачуванішою.

Ще одна велика перевага — це компонентна архітектура. Завдяки їй можна легко розбивати інтерфейс на маленькі незалежні частини, які зручно перевикористовувати. Наприклад, якщо створити компонент кнопки, його можна вставити в будь-який інший компонент, де ця кнопка потрібна, не переписуючи код.

Також дуже корисною є можливість двостороннього зв’язування даних (two-way data binding). Це означає, що коли користувач щось вводить у формі, це автоматично оновлює дані в моделі, і навпаки — якщо модель змінюється, це одразу видно в інтерфейсі.

Angular має вбудовану маршрутизацію, що дозволяє легко створювати односторінкові застосунки (SPA) з багатьма сторінками, без потреби в перезавантаженні сайту [6]. Також він підтримує реактивне програмування з RxJS, що дуже зручно для роботи з потоками даних — наприклад, з вебсокетами або формами.

Варто згадати і про Angular CLI — це консольний інструмент, який дозволяє швидко генерувати компоненти, сервіси та інші частини проєкту [2]. Це значно пришвидшує розробку і забезпечує єдину структуру коду в команді.

Але, звісно, Angular має і свої недоліки. Один із головних — це високий поріг входу. Для новачка фреймворк може здатися складним через велику кількість нових понять: DI, RxJS, шаблони, модулі, сервіс-провайдери тощо.

Ще один момент — велика кількість шаблонного коду. Навіть для простого компонента потрібно створити кілька файлів: .ts, .html, .css, і це займає більше часу, ніж, скажімо, у React. Також RxJS, хоча й потужний, іноді буває складним у налагодженні, особливо якщо не розумієш, як працюють стріми [4].

Крім того, Angular потребує компіляції, і застосунок може бути важким для браузера, особливо якщо не налаштований lazy loading. А якщо не використовувати Angular Universal, SEO для сайтів на Angular буде складнішим, бо контент не видно пошуковим системам без серверного рендерингу.

Angular вирізняється серед інших фреймворків тим, що є повноцінним інструментом «з коробки». На відміну від, скажімо, React чи Vue, де багато функцій потрібно додатково налаштовувати або підключати сторонні бібліотеки, Angular одразу пропонує все необхідне для побудови масштабованого застосунку. Тут уже є вбудована маршрутизація, робота з формами, валідація, система сервісів, інжекція залежностей (DI) та багато іншого. Це робить Angular зручним вибором для команд, які хочуть зосередитися на логіці застосунку, а не на підключенні окремих модулів.

Можна визначити основні принципи Angular:

- Статична типізація

  Типи змінних, параметрів, функцій тощо вказуються явно або виводяться автоматично. Це дозволяє уникати багатьох помилок, які виникають у JavaScript під час виконання програми.
- Орієнтація на об'єктно-орієнтоване програмування (ООП)

Підтримка класів, інтерфейсів, абстракцій, наслідування, модифікаторів доступу (private, protected, public), дженеріків тощо [9].

- Підтримка нових стандартів JavaScript

  TypeScript дозволяє використовувати можливості ECMAScript наступних версій (ES6, ES7 і далі), навіть якщо середовище виконання їх не підтримує, компілюючи їх у backward-compatible JavaScript [4].

- Безперервна інтеграція з редакторами коду

  Наприклад, у Visual Studio Code TypeScript забезпечує автодоповнення, навігацію по коду, рефакторинг, підсвічування помилок у реальному часі та ін.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Структура та логічні зв’язки в базі даних

У проєкті використовується реляційна база даних, яка є логічною основою для зберігання, обробки та взаємодії з усіма основними даними системи. Реляційна модель бази даних є найпоширенішою серед сучасних інформаційних систем через її гнучкість, простоту у використанні та підтримку складних запитів. Всі сутності, що мають відношення до користувачів, організацій, вакансій, відгуків, формують повноцінну та взаємозв’язану структуру даних.

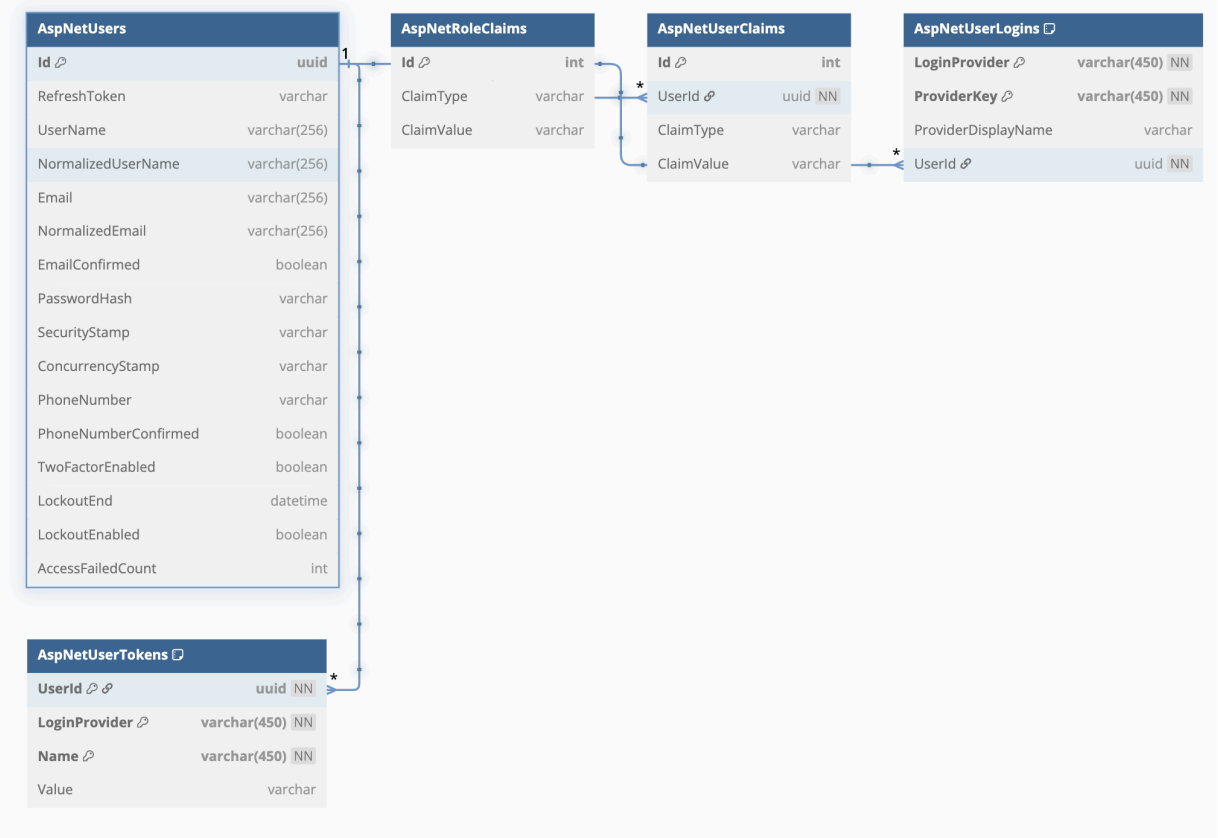

Рисунок 3.1 - Структура таблиць безпеки системи на основі ASP.NET Identity

На Рис. 3.1 зображено підсистему безпеки, що реалізується через стандартний функціонал ASP.NET Identity. Основною таблицею в цій ділянці є AspNetUsers, яка містить у собі інформацію про кожного зареєстрованого

користувача: логін, email, підтвердження пошти, хешований пароль, статус двофакторної автентифікації тощо. Всі інші таблиці — це допоміжні сутності, які зберігають додаткову інформацію:

- AspNetUserClaims — таблиця, яка дозволяє додавати claims (атрибути), що описують користувача (наприклад, права доступу або ролі);
- AspNetUserLogins — таблиця для реалізації входу через сторонні сервіси (OAuth-провайдери);
- AspNetUserTokens — використовується для зберігання токенів доступу та оновлення;
- AspNetRoleClaims — дозволяє прив’язати до певної ролі додаткові атрибути, що потім можуть застосовуватись до користувачів через систему ролей.

Ці зв’язки реалізують базову систему безпеки з ролями, правами доступу та автентифікацією, що дозволяє масштабувати платформу під потреби будь-якої організації.

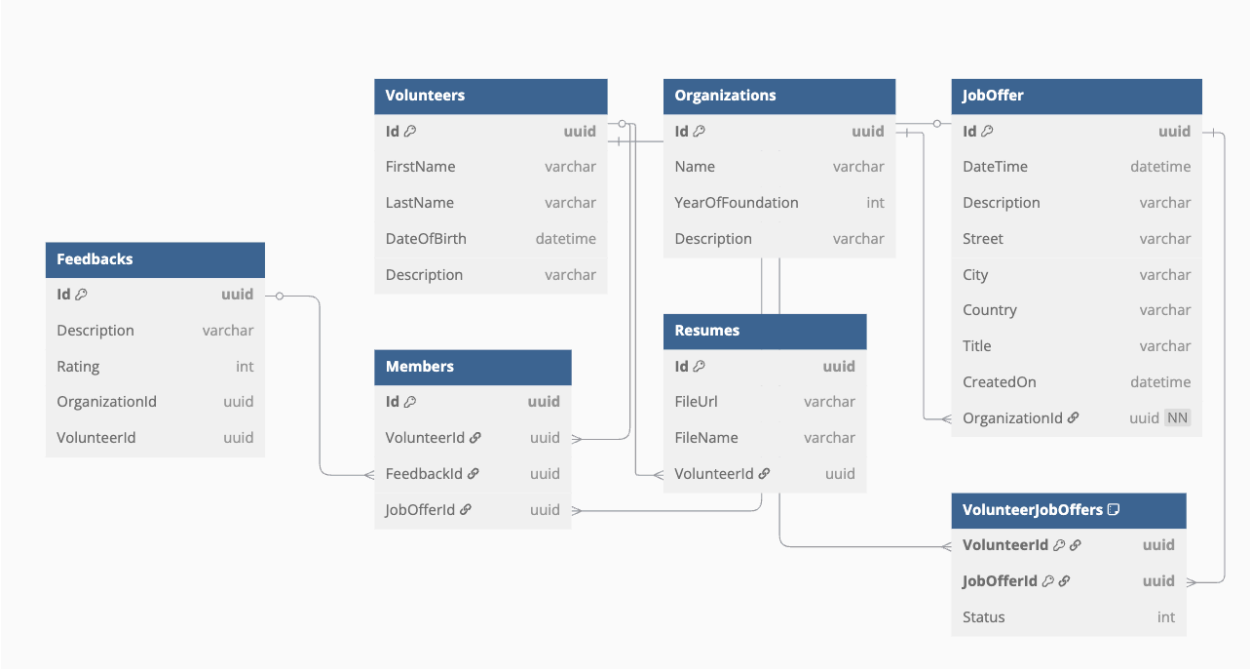

Рисунок 3.2 - Логічна структура основних сутностей волонтерської системи

На Рисунку 3.2 зображена основна предметна область — бізнес-логіка застосунку, яка відповідає за волонтерську взаємодію. Центральними таблицями тут є Volunteers та Organizations.

Таблиця Organizations зберігає базову інформацію про організацію, включаючи назву, опис та рік заснування. З організацією пов’язана таблиця JobOffer, яка представляє вакансії або пропозиції волонтерської роботи. Одна організація може мати багато таких пропозицій.

Таблиця Volunteers містить детальну інформацію про волонтерів — ім’я, прізвище, дату народження та опис. Волонтери можуть подаватися на вакансії, що реалізовано через таблицю VolunteerJobOffers, яка є проміжною сутністю для зв’язку багато до багатьох між Volunteers і JobOffer. Така структура дозволяє точно фіксувати статус участі волонтера в певній вакансії.

Таблиця Resumes забезпечує можливість завантаження волонтером резюме у вигляді файлу, що містить URL і назву файлу. Це дозволяє організаціям переглядати професійний бекграунд кандидатів.

Таблиця Feedbacks дозволяє волонтерам залишати відгуки про організації, вказуючи рейтинг і текстовий опис досвіду. У свою чергу, ці дані можуть бути використані для аналізу якості співпраці.

Центральною точкою фіксації результатів співпраці є таблиця Members, яка дозволяє об'єднувати волонтерів із вакансіями, доповнюючи це посиланням на відповідний Feedback. Завдяки цьому можна прослідкувати історію участі кожного волонтера в різних проєктах, включаючи відгуки та завершені задачі.

Уся структура спроектована за принципом нормалізації: усі зв’язки реалізовано через первинні та зовнішні ключі, що дозволяє уникати дублювання даних, підтримувати цілісність та полегшує масштабування системи. База даних адаптована як до безпечної роботи з користувачами, так і до виконання основної бізнес-логіки платформи для залучення волонтерів до міжнародних проєктів.

3.2 Архітектура файлової структури застосунку

Клієнтська частина застосунку реалізована з використанням Angular - сучасного фреймворку, що дозволяє будувати масштабовані SPA-додатки [6]. Структура проєкту розроблена згідно з принципами модульності, розділення відповідальностей та повторного використання компонентів. Всі функціональні блоки чітко організовані по папках: core містить ключову бізнес-логіку, shared — спільні компоненти й сервіси, а pages — окремі сторінки, що відповідають за маршрутизацію та відображення інтерфейсу. Такий підхід забезпечує легкість у розширенні функціональності, підтримці коду та залученні нових розробників до команди.

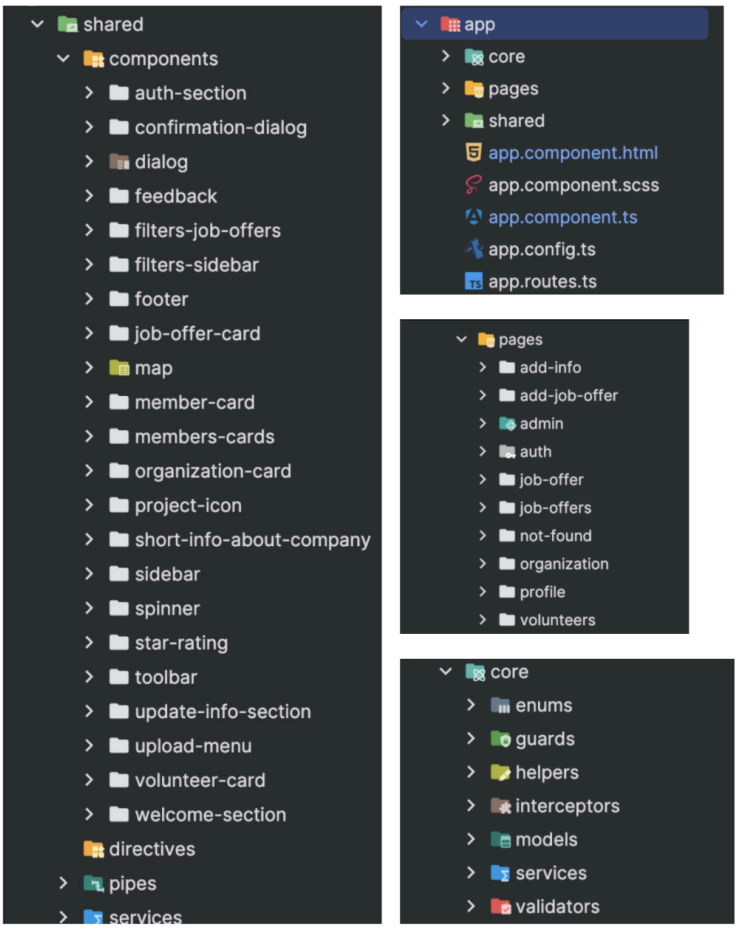

Рисунок 3.3 - Структура Angular-проєкту на клієнтській частині

На рисунку 3.3 показано дерево основних папок проєкту, яке складається з таких основних частин:

Папка app — це коренева папка клієнтського коду, яка включає головні конфігураційні файли, такі як app.component.ts, app.routes.ts, app.config.ts, а також основний HTML-шаблон і стилі. Цей рівень відповідає за стартову ініціалізацію Angular-застосунку.

1. Папка core — містить основні частини проєкту, що використовуються по всьому застосунку:

- enums — перелічення типів для роботи зі статусами, ролями тощо;
- guards — логіка доступу до сторінок (наприклад, AuthGuard);
- helpers — допоміжні функції та утиліти;
- interceptors — перехоплювачі HTTP-запитів (наприклад, для додавання токенів авторизації);
- models — інтерфейси й типи, що описують структуру даних (DTO);
- services — сервіси, які взаємодіють із backend API через HTTP;
- validators — логіка перевірки даних на рівні форм (наприклад, перевірка email чи пароля).

2. Папка pages — вміщує логіку для відображення основних сторінок користувача:

- add-info, add-job-offer, organization, volunteers — окремі сторінки;
- auth — сторінки реєстрації та входу;
- job-offer, job-offers — сторінки перегляду вакансій;
- profile — профіль користувача;
- admin — окрема область для адміністратора;
- not-found — сторінка 404, якщо маршрут не знайдено.

3. Папка shared — модуль спільного використання, містить багаторазові компоненти, що використовуються в кількох місцях застосунку:

- components — підпапка, в якій знаходяться дрібніші компоненти: auth-section, feedback, filters-job-offers, footer, map, member-card, sidebar, star-rating тощо;
- directives — кастомні Angular-директиви;

- pipes — пайпи для трансформації даних у шаблонах;
- services — сервіси, які надають спільну логіку, наприклад, роботу з toast-повідомленнями або локальним сховищем.

Завдяки такій структурі код фронтенд проєкту залишається добре організованим, масштабованим і зрозумілим як для поточних розробників, так і для нових членів команди. Всі функціональні одиниці чітко відокремлені відповідно до принципів модульності та розділення відповідальностей, що суттєво спрощує розширення та підтримку системи.

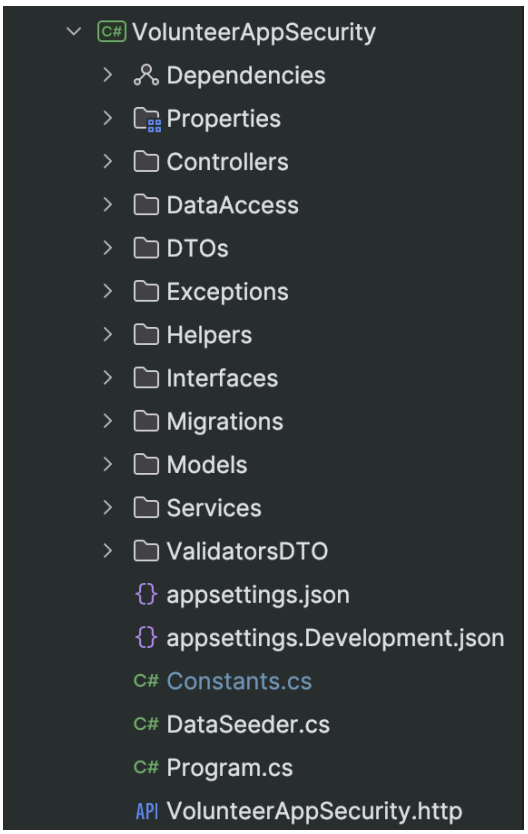

Рисунок 3.4 - Архітектурна структура бекенд-застосунку

Серверна частина проєкту реалізована з використанням ASP.NET Core. Уся логіка зосереджена в одному проєкті — VolunteerAppSecurity, структура якого побудована за принципами розділення відповідальностей та чистої архітектури. Папка Controllers містить контролери, які відповідають за обробку HTTP-запитів та взаємодіють із сервісами. У Services реалізована бізнес-логіка, а Interfaces містить відповідні контракти до сервісів. Папка

Models включає сутності, що відображають структуру таблиць у базі даних, тоді як DTOs — об'єкти передачі даних, що використовуються для комунікації між фронтендом і бекендом.

У DataAccess зберігаються класи, що відповідають за роботу з контекстом бази даних через Entity Framework Core. Migrations містить автоматично згенеровані файли для оновлення схеми бази даних. У ValidatorsDTO зберігаються класи для валідації вхідних даних на основі FluentValidation. Окремо виділені Helpers — для зберігання допоміжних функцій, Exceptions — для обробки помилок, та конфігураційні файли appsettings.json і appsettings.Development.json, які містять налаштування оточення та підключення до бази даних.

Також присутній файл DataSeeder.cs, який використовується для первинного наповнення бази, та Program.cs, що є точкою входу до застосунку. Уся ця структура дозволяє ефективно організувати серверну логіку, спростити тестування, розширення та підтримку проєкту.

3.3 Серверна логіка та архітектура бекенду

Серверна частина застосунку реалізована на основі технології .NET, яка забезпечує надійний інструментарій для побудови REST API з чіткою структурою та підтримкою принципів розділення відповідальностей.

У роботі використано шаблон чистої архітектури, де кожен модуль відповідає за свою частину логіки. Це дозволяє легко підтримувати, масштабувати та розширювати застосунок у майбутньому.

Одним з ключових компонентів бекенду є модуль авторизації та автентифікації, який реалізує логіку реєстрації, логіну, підтвердження пошти, оновлення токенів та видалення користувачів.

Одним із важливих контролерів у модулі авторизації є UserController, який обробляє основні запити, пов’язані з управлінням користувачами. У ньому реалізовано такі ендпоінти:

- login - для автентифікації користувача на основі email і пароля. Після перевірки даних генерується JWT-токен для подальшої авторизації.
- register - створення нового користувача з валідацією даних. Якщо створення успішне, система повертає статус 201.
- verification - підтвердження email за допомогою коду, що надсилається користувачу після реєстрації.
- token - оновлення access-токена на основі refresh-токена.
- users - видалення користувача за ідентифікатором (лише для авторизованих).

Уся бізнес-логіка винесена у сервіси (IUserService, ITokenGenerator), що дозволяє легко тестувати систему, замінювати реалізації та дотримуватись принципу інверсії залежностей (SOLID). Сервіси перевіряють наявність користувача в базі, валідність пароля, генерують токени доступу та оновлення, обробляють помилки.

Для роботи з користувачами використовується стандартна модель безпеки ASP.NET Identity. Дані зберігаються в таблицях AspNetUsers, AspNetRoles, AspNetUserRoles, що автоматично створюються через міграції Entity Framework. Це забезпечує підтримку ролей, прав доступу та зберігання токенів оновлення.

Таким чином, безпековий модуль побудовано згідно з рекомендаціями Microsoft: з використанням ASP.NET Identity, розділенням обов’язків між контролерами та сервісами, а також підтримкою безпечного оновлення токенів. Це забезпечує надійну автентифікацію, авторизацію та зручну масштабованість при розширенні системи.

Окрім модуля безпеки, серверна частина системи включає низку функціональних контролерів, які відповідають за ключову бізнес-логіку застосунку. Усі ці контролери реалізовані відповідно до принципів чистої архітектури - контролери відповідають за прийом HTTP-запитів, сервіси - за бізнес-логіку, а DTO - за передачу даних. Це дозволяє досягти високої

гнучкості та підтримуваності коду.

Один із таких контролерів - JobOfferController, який обробляє взаємодію організацій із вакансіями. У цьому контролері реалізовано:

- create - створення нової пропозиції роботи. Контролер приймає DTO, отримує ідентифікатор організації з токена авторизації і передає дані до сервісу.
- update, delete - оновлення або видалення вакансії. Усі операції супроводжуються перевіркою валідності моделі та обробкою виключень.
- get-job-offer, get-job-offers - отримання однієї або списку вакансій з фільтрацією та пагінацією.
- get-cities, get-countries, get-streets, get-count-job-offers - допоміжні методи для отримання довідкової інформації, необхідної на етапах створення або пошуку вакансій.

OrganizationController реалізує логіку роботи з організаціями. Організація може:

- create - створити профіль,
- update - оновити свої дані,
- get - переглянути свій профіль або профілі інших організацій,
- delete - видалити обліковий запис, при цьому, окремим HTTP-запитом виконується запит до зовнішнього сервісу для видалення пов’язаного користувача.

Контролер FeedbackController дозволяє волонтерам залишати відгуки про організації після завершення співпраці. Метод AddFeedback зчитує ID волонтера з JWT-токена і передає відгук у сервіс, де перевіряється його валідність і зберігається у базі даних. Водночас GetListFeedbacks дозволяє організаціям та адміністраторам переглядати список з усіма залишеними відгуками.

RequestController обробляє заявки волонтерів на вакансії. Реалізовані такі основні методи:

- add - волонтер подає заявку на вакансію. Ідентифікатор вакансії передається у запиті, а ID волонтера зчитується з JWT.
- get-job-offers-requests, get-volunteers-requests - повертають список заявок, поданих волонтером або отриманих організацією, з фільтрацією.
- confirm, cancel - організація може підтвердити або скасувати заявку певного волонтера.
- get-status - перевірка статусу поданої заявки.

3.4 Фронтенд-реалізація користувацького інтерфейсу

Коли користувач уперше відкриває вебзастосунок, йому відображається форма аутентифікації, яка дозволяє або увійти до системи, або створити новий обліковий запис.

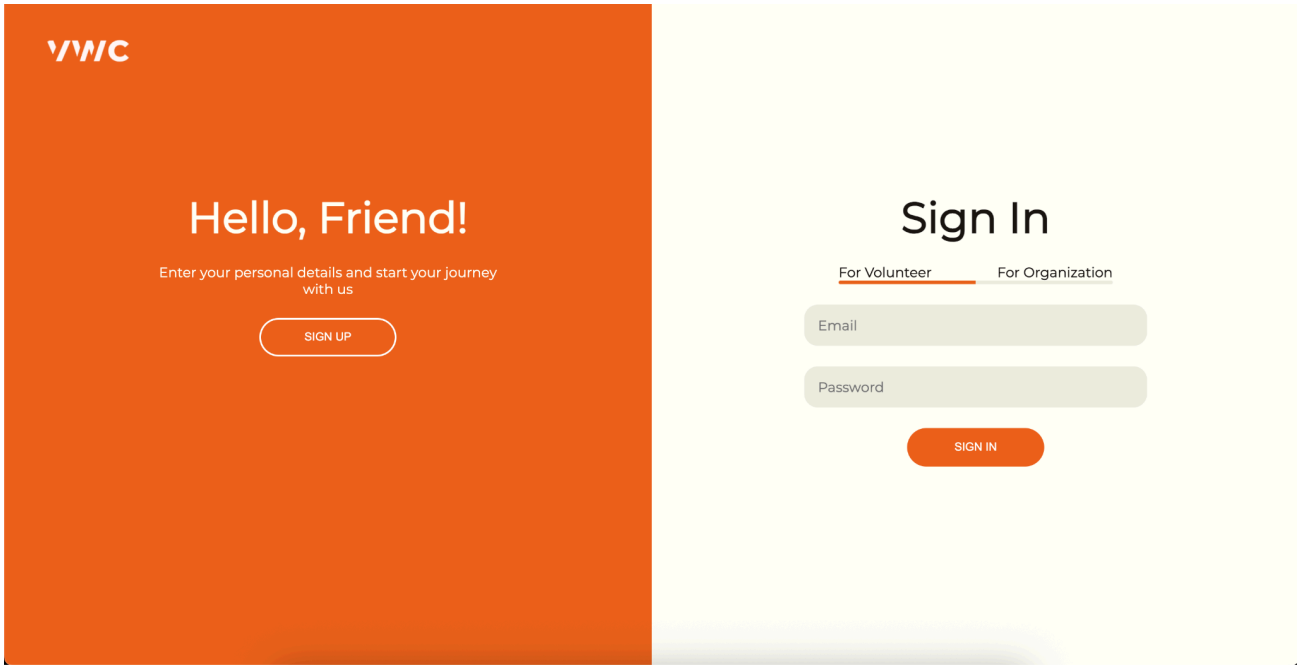

Рисунок 3.5 - Форма для входу

На рисунку 3.5 представлено зовнішній вигляд форми входу та реєстрації. Якщо користувач ще не зареєстрований, він може натиснути

кнопку “SIGN UP”, після чого відображається форма реєстрації, зображена на Рис. 3.6.

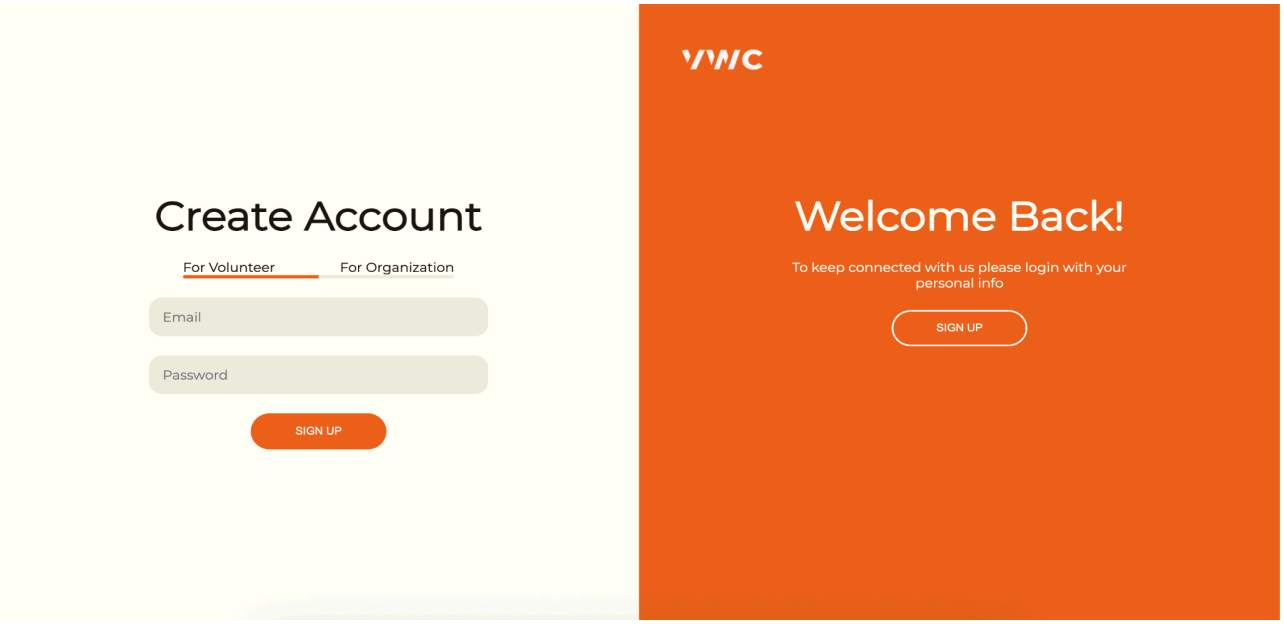

Рисунок 3.6 - Форма реєстрації нового користувача

Інтерфейс реалізовано у вигляді двох вкладок: для волонтерів і для організацій. Це дозволяє одразу розділити логіку взаємодії відповідно до ролі користувача в системі. У формі реєстрації користувач повинен ввести адресу електронної пошти та пароль, після чого система автоматично надсилає листа з посиланням для підтвердження пошти. Після підтвердження пошти користувач переходить до форми введення додаткових персональних даних, які залежать від обраного типу облікового запису. На Рис. 3.7 зображено форму заповнення інформації для організації, а на Рис. 3.8 – для волонтера.

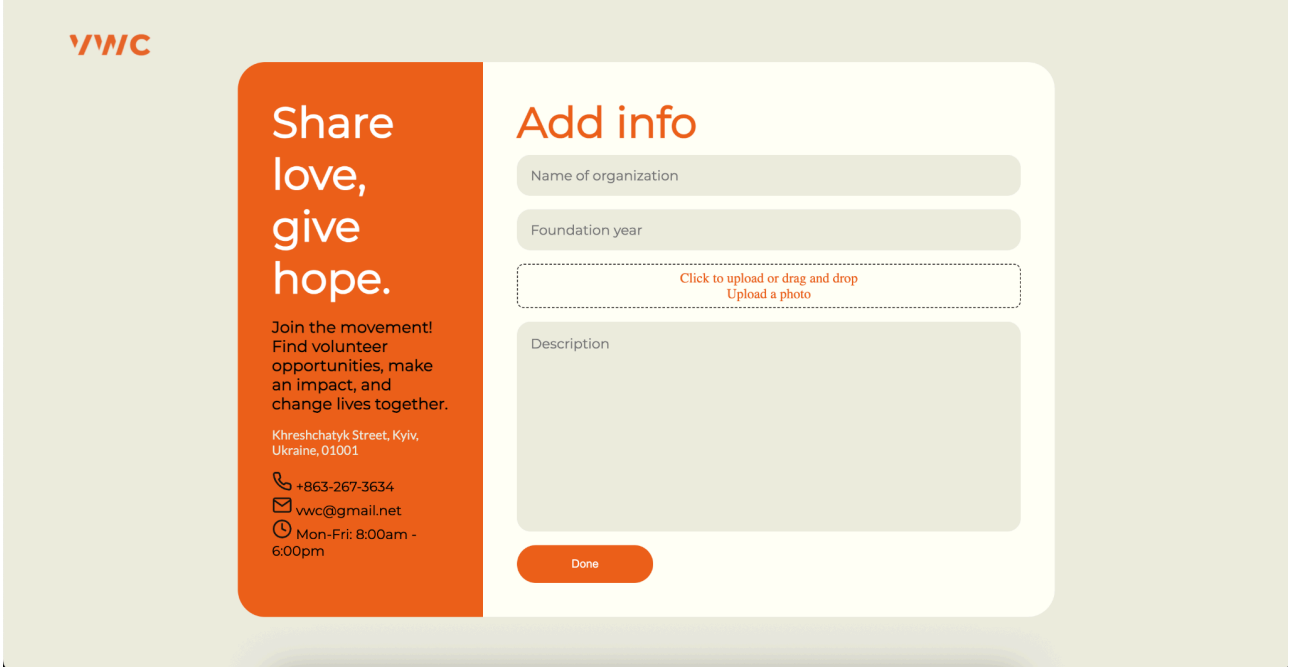

Рисунок 3.7 - Форма введення інформації про організацію

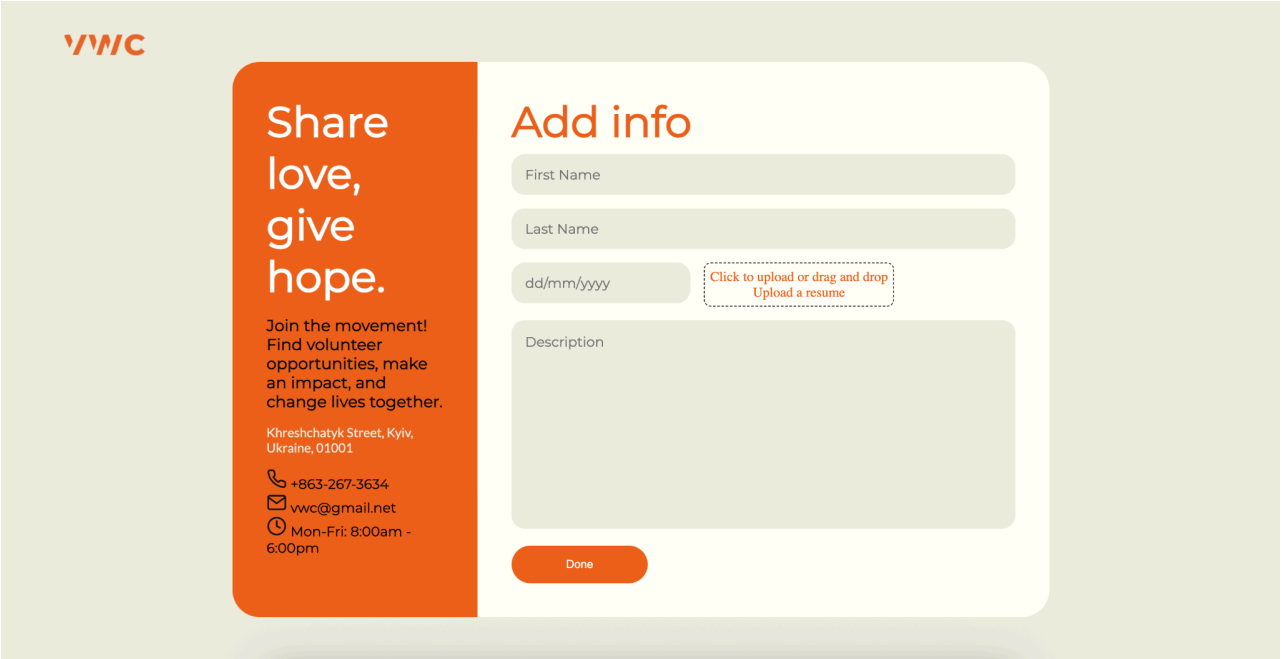

Рисунок 3.8 - Форма введення інформації про волонтера

Після успішного входу до системи, користувач автоматично перенаправляється на інтерфейс, відповідний до його ролі. У випадку, якщо роль користувача — волонтер, система відкриває сторінку з актуальними офферами від організацій.

На цій сторінці реалізовано функціонал пошуку за назвою вакансії, а також пагінацію, що дозволяє ефективно переглядати пропозиції навіть при великій кількості записів. Візуальне представлення цієї сторінки наведено на Рис. 3.9.

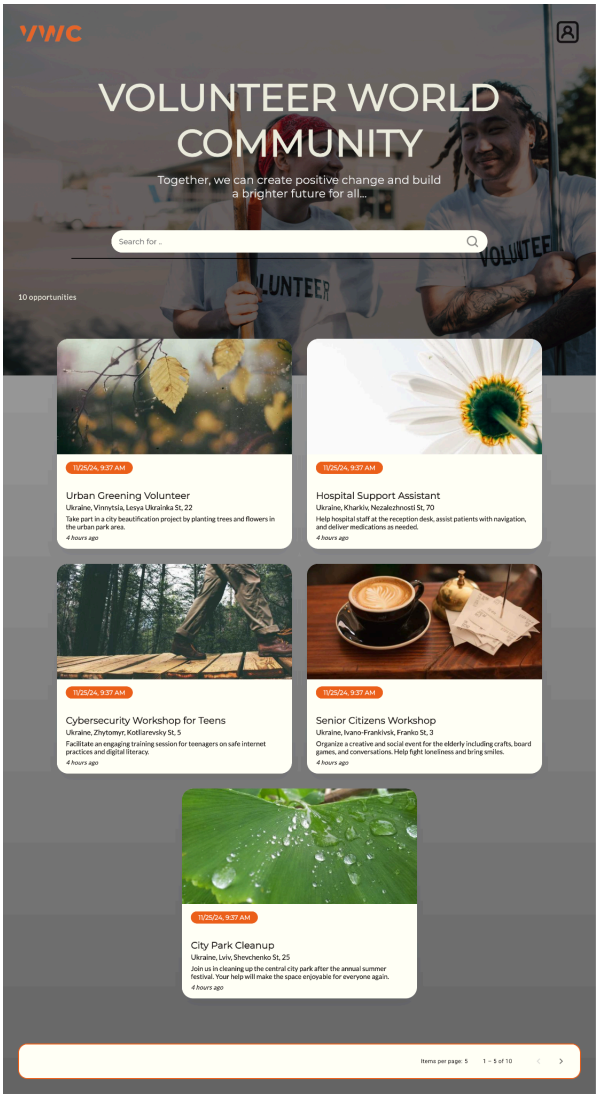

Рисунок 3.9 - Сторінка з вакансіями від організацій для волонтера

Користувач має можливість натиснути на конкретний оффер, щоб переглянути детальну інформацію про нього. Такий підхід дозволяє прийняти зважене рішення щодо участі в проєкті. Інтерфейс перегляду одного конкретного офферу зображено на Рис. 3.10.

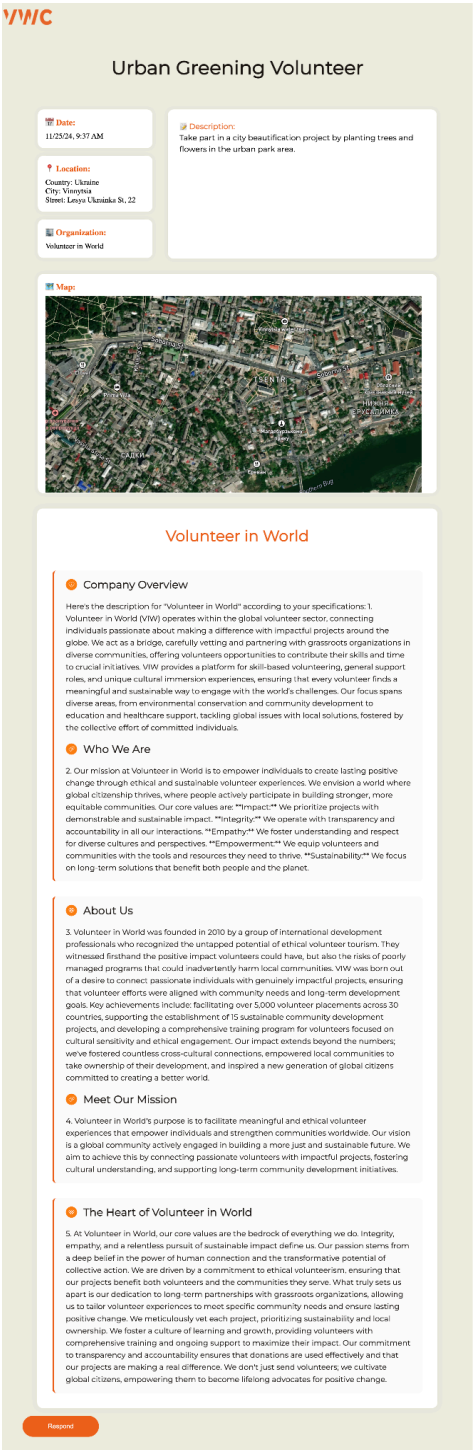

Рисунок 3.10 – Детальна інформація про обраний оффер

У разі, якщо волонтера зацікавила пропозиція, він може надіслати запит на участь у цьому оффері безпосередньо зі сторінки детального перегляду. Таким чином, забезпечується інтуїтивно зрозумілий та зручний механізм взаємодії між волонтерами та організаціями через систему.

Також волонтер має можливість перейти до особистого профілю, що зображено на Рис. 3.11, де доступні наступні функції: редагування персональних даних, видалення акаунта, вихід із системи, а також перегляд власних запитів на участь у волонтерських проєктах із зазначеним статусом підтвердження.

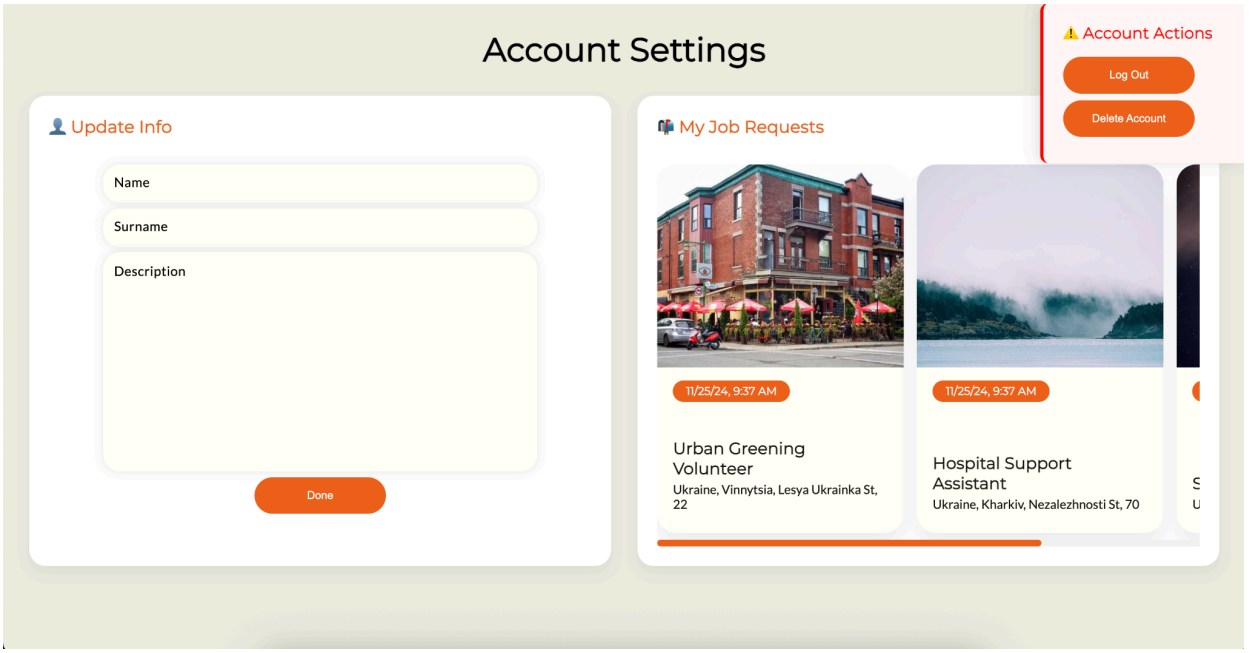

Рисунок 3.11 – Сторінка особистого профілю користувача

3.5 Інтеграція сторонніх бібліотек та фреймворків

У межах розширення функціональних можливостей клієнтської частини системи було реалізовано інтеграцію з AI-сервісом Google Gemini. Для цього було використано офіційний SDK від Google — @google/genai, який дозволяє надсилати запити до моделей штучного інтелекту Gemini прямо з Angular-застосунку.

Сервіс OpenAiService, реалізований у структурі Angular, забезпечує зручний інтерфейс для взаємодії з моделлю gemini-2.0-flash. Для автентифікації використовується API-ключ, що зберігається у конфігураційному файлі середовища (environment.ts). Запити до моделі

виконуються асинхронно, а отримані відповіді обробляються та передаються у фронтенд.

```
export class OpenAiService {
  private apiKey: string = environment.openaiApiKey;
  private ai: GoogleGenAI = new GoogleGenAI({ apiKey: this.apiKey });

  Show usages
  public async getAIResponse(prompt: string) {
    try {
      const response: GenerateContentResponse =
        await this.ai.models.generateContent({
          model: 'gemini-2.0-flash',
          contents: [{ role: 'user', parts: [{ text: prompt }] }],
        });

      const content : string  = response.candidates?.[0]?.content?.parts[0]?.text || '';

      return content;
    } catch (error) {
      return 'Error fetching overview';
    }
  }
}
```

Рисунок 3.5 - Клас OpenAiService для інтеграції з моделлю ШІ

Завдяки такій інтеграції, що зображена на Рис. 3.5, користувачі можуть отримувати відповіді, згенеровані ШІ, наприклад, для автоматичного формування описів або узагальнень на основі введених даних.

Крім інтеграції з ШІ, у застосунку реалізовано використання Mapbox GL JS - сучасної JavaScript-бібліотеки для роботи з інтерактивними картами.

Компонент MapComponent дозволяє відображати карту із супутниковим виглядом, а також динамічно позиціонувати маркер на основі адреси, що складається з міста та вулиці. Для цього використовується Mapbox Geocoding API, який виконує пошук координат за текстовим запитом.

Ініціалізація карти здійснюється при завантаженні компонента, а при зміні значення міста чи вулиці карта автоматично переміщується у відповідне місце. Також додається маркер для візуального позначення координат на мапі.

ВИСНОВКИ

У даній роботі була розглянута, спроєктована та реалізована веборієнтована система для управління волонтерськими ініціативами, що включає в себе як серверну, так і клієнтську частини.

Програмне рішення для серверної частини було створено за допомогою об’єктно-орієнтованої мови C# на базі фреймворку ASP.NET Core, з використанням Entity Framework Core для взаємодії з реляційною базою даних Microsoft SQL Server. Клієнтська частина була реалізована засобами Angular як SPA-додаток, з використанням бібліотеки PrimeNG для побудови зручного інтерфейсу користувача.

Сформовано основну задачу системи — ефективна взаємодія волонтерів та організацій, а також описано ключовий функціонал, серед якого: реєстрація/авторизація користувачів, додавання волонтерських пропозицій, подання заявок на участь, перегляд і фільтрація заявок, система оцінювання та зворотного зв'язку, редагування профілю, управління організаціями тощо.

У даній роботі використовувалися сторонні сервіси та бібліотеки. Зокрема, інтегровано Google Gemini AI через офіційний SDK @google/genai, що дозволило реалізувати функцію автоматичного створення опису організацій. Також реалізовано візуалізацію геолокації організації за допомогою Mapbox GL JS.

Таким чином, розроблена система відповідає поставленим цілям та вимогам, є функціонально завершеною та може бути використана як реальне рішення для пошуку волонтерських можливостей і комунікації між організаціями та волонтерами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adam Freeman, Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages, Apress, 2022, 1080 сторінок.
2. Deborah Kurata, Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies, Packt Publishing, 2021, 400 сторінок.
3. Julie Lerman, Programming Entity Framework: Code First, O'Reilly Media, 2011, 193 сторінки.
4. Rasmus Lerdorf, Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker, O’Reilly Media, 2021, 350 сторінок.
5. Steve Krug, Don't Make Me Think: A Common Sense Approach to Web Usability, New Riders, 2014, 216 сторінок.
6. Alan Dennis, Barbara Haley Wixom, Roberta M. Roth, Systems Analysis and Design, Wiley, 2015, 576 сторінок.
7. Adam Boduch, Angular 13 by Example: Build modern web apps using Angular 13 and TypeScript by working on 10 different projects, Packt Publishing, 2022, 500 сторінок.
8. Scott Klein, Pro SQL Server Relational Database Design and Implementation, Apress, 2020, 700 сторінок.
9. Eric Freeman, Elisabeth Robson, Head First Design Patterns: A Brain-Friendly Guide, O'Reilly Media, 2020, 694 сторінки.
10. Mapbox documentation. [Електронний ресурс]. URL:https://docs.mapbox.com/mapbox-gl-js-guides/
11. Gemini API Documentation. [Електронний ресурс]. URL: https://ai.google.dev/gemini

ДОДАТКИ

```
namespace VolunteerAppSecurity.Controllers
{
    [Route("api/users")]
    [ApiController]
    public class UserController : ControllerBase
    {
        readonly IUserService _userService;
        readonly ITokenGenerator _tokenGenerator;

        public UserController(IUserService userService, ITokenGenerator tokenGenerator)
        {
            _userService = userService;
            _tokenGenerator = tokenGenerator;
        }
        [HttpPost("login")]
        [AllowAnonymous]
        public async Task<IActionResult> Login([FromBody] LoginDTO userLoginDTO)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);

            var user = await _userService.UserExist(userLoginDTO.Email);
            if (!user) return NotFound("No user exists");

            var checkPassword = await _userService.CheckPassword(userLoginDTO.Email,
userLoginDTO.Password);

            if (!checkPassword)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status400BadRequest,
                    Title = "Incorrect password!",
                    Detail = "Incorrect password!"
                };
            }
            var token = await _userService.GenerateTokens(userLoginDTO.Email);

            if (token == null)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "Error occured
while creating user on server");
            }

            return Ok(token);
        }
        [HttpPost("register")]
        [AllowAnonymous]
        public async Task<IActionResult> Registration([FromBody] RegisterDTO userRegisterDTO)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);

            var createdUser = await _userService.CreateUser(userRegisterDTO);

            if (createdUser != null)
                return Created("/api/users", createdUser);
            else
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "Server error",
                    Detail = "Error occured while creating user on server"
                };
        }
        [HttpGet("verification")]
```

```
        public async Task<IActionResult> VerificationEmail([FromQuery] string userId,
[FromQuery] string code)
        {
            if (userId == null || code == null)
                return BadRequest(new AuthResponse()
                {
                    Errors = new List<string>() { "Invalid email confirmation url" },
                    Result = false
                });
            var user = await _userService.GetUserById(userId);
            if (user == null)
            {
                return BadRequest(new AuthResponse()
                {
                    Errors = new List<string>()
                    {
                        "Invalid email parameter"
                    },
                    Result = false
                });
            }
            code = code.Replace(' ', '+');
            if (await _userService.VerifyEmail(user, code))
            {
                return Redirect("http://localhost:4200/signin");
            }
            return Redirect("http://localhost:4200/signup");
        }
        [AllowAnonymous]
        [HttpPost("token")]
        public async Task<IActionResult> RefreshToken([FromBody] RefreshTokenDTO
refreshTokenDTO)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            var result = await _tokenGenerator.RefreshAccessToken(refreshTokenDTO.AccessToken,
                refreshTokenDTO.RefreshToken);
            if (result == null) return StatusCode(StatusCodes.Status400BadRequest);
            return Ok(result);
        }
        [Authorize]
        [HttpDelete]
        public async Task<IActionResult> DeleteUser([FromQuery] Guid userId)
        {
            var deletedUser = await _userService.DeleteUserById(userId);

            if (deletedUser)
            { return Ok();}
            else
            {throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "Server error",
                    Detail = "Error occured while creating user on server"
                };
            }
        }
    }
}
namespace VolunteerAppSecurity.DataAccess
{
    public class SecurityDBContext : IdentityDbContext<User, IdentityRole<Guid>, Guid>
    {
        public SecurityDBContext(DbContextOptions options) : base(options){}
        protected SecurityDBContext(){}
    }
}
namespace VolunteerAppSecurity.Exceptions
```

```
{
    public class ExceptionResponseFilter : IExceptionFilter
    {
        private readonly ILogger<ExceptionResponseFilter> _logger;
        public ExceptionResponseFilter(ILogger<ExceptionResponseFilter> logger)
        {
            _logger = logger;
        }
        public void OnException(ExceptionContext context)
        {
            var statusCode = context.Exception switch
            {
                ApiException apiException => apiException.StatusCode,
                ArgumentNullException _ => StatusCodes.Status422UnprocessableEntity,
                ValidationException _ => StatusCodes.Status422UnprocessableEntity,
                AuthenticationException _ => StatusCodes.Status401Unauthorized,
                _ => StatusCodes.Status500InternalServerError
            };
            var title = context.Exception switch
            {
                ApiException apiException => apiException.Title,
                ArgumentNullException _ => "Ignored or malformed argument",
                ValidationException _ => "Validation error",
                AuthenticationException _ => "Authentication issue",
                _ => "Internal error"
            };
            var message = context.Exception switch
            {
                ApiException apiException => apiException.Detail,
                ArgumentNullException _ => context.Exception.Message,
                ValidationException _ => context.Exception.Message,
                AuthenticationException _ => "Action is permitted because of authentication
reasons",
                _ => "Internal error occured. Please try a little bit later"
            };
            var problemDetails = new ProblemDetails
            {
                Title = title,
                Status = statusCode,
                Detail = message
            };
            if (problemDetails.Status >= StatusCodes.Status500InternalServerError)
            {
                _logger.LogError(context.Exception, "Critical error handled");
            }
            else if (problemDetails.Status >= StatusCodes.Status400BadRequest)
            {
                _logger.LogError(context.Exception, "Request error handled");
            }
            var response = BuildResponse(problemDetails);
            context.HttpContext.Response.StatusCode = response.StatusCode ??
StatusCodes.Status500InternalServerError;
            context.Result = response;
            context.ExceptionHandled = true;
        }
        private static ObjectResult BuildResponse(ProblemDetails problem) =>
           new ObjectResult(problem)
           {
               StatusCode = problem.Status ?? StatusCodes.Status500InternalServerError,
               ContentTypes = new MediaTypeCollection { "application/problem+json" }
           };
    }
}
namespace VolunteerAppSecurity.Services
{
    public class TokenGenerator : ITokenGenerator
    {
```

```
        private const int RefreshTokenSize = 32;
        private readonly UserManager<User> _userManager;
        private readonly AuthenticationSetup _authSettings;
        public TokenGenerator(UserManager<User> userManager, IOptions<AuthenticationSetup>
authSettings)
        {
            _userManager = userManager;
            _authSettings = authSettings.Value;
        }
        public async Task<string> GenerateAccessToken(User user)
        {
            var handler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_authSettings.SecretKey);
            var roles = await _userManager.GetRolesAsync(user);
            ClaimsIdentity identity = new ClaimsIdentity(new[] {
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
                new Claim(ClaimTypes.Role, roles.FirstOrDefault(ClaimTypes.Role)),
                new Claim(ClaimTypes.Name, user.UserName)
            });
            var securityToken = handler.CreateToken(new SecurityTokenDescriptor
            {
                Issuer = _authSettings.Issuer,
                Audience = _authSettings.Audience,
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature),
                Subject = identity,
                Expires = DateTime.Now.AddMinutes(_authSettings.AccessTokenExpirationMinutes)
            });
            return handler.WriteToken(securityToken);
        }
        public async Task<AuthenticationResponse> GenerateTokens(User user)
        {
            user.RefreshToken = GenerateRefreshToken(user);
            var result = await _userManager.UpdateAsync(user);
            if (!result.Succeeded)
            {
                throw new Exception("Unable to create refresh token");
            }
            return new AuthenticationResponse
            {
                AccessToken = await GenerateAccessToken(user),
                RefreshToken = user.RefreshToken,
            };
        }
        public string GenerateRefreshToken(User user)
        {
            var randomNumber = new byte[RefreshTokenSize];
            using (var rng = RandomNumberGenerator.Create())
            {
                rng.GetBytes(randomNumber);
                return Convert.ToBase64String(randomNumber);
            }
        }
        public async Task<AuthenticationResponse> RefreshAccessToken(string accessToken,
string refreshToken)
        {
            var principal = GetPrincipalFromExpiredToken(accessToken);
            var user = await _userManager.FindByNameAsync(principal.Identity.Name);
            if (user == null || user.RefreshToken != refreshToken)
            {
                throw new SecurityTokenException("Invalid refresh token");
            }
            user.RefreshToken = GenerateRefreshToken(user);
            var result = await _userManager.UpdateAsync(user);
            if (!result.Succeeded)
            {
```

```
                throw new Exception("Unable to create refresh token");
            }
            return new AuthenticationResponse
            {
                AccessToken = await GenerateAccessToken(user),
                RefreshToken = user.RefreshToken
            };
        }
        private ClaimsPrincipal GetPrincipalFromExpiredToken(string token)
        {
            var tokenValidationParameters = new TokenValidationParameters
            {
                ValidateAudience = false,
                ValidateIssuer = false,
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = _authSettings.GetSecurityKey(),
                ValidateLifetime = false
            };
            var tokenHandler = new JwtSecurityTokenHandler();
            var principal = tokenHandler.ValidateToken(token, tokenValidationParameters, out
_);
            return principal;
        }
    }
}
namespace VolunteerAppSecurity.Services
{
    public class UserService : IUserService
    {
        readonly UserManager<User> _userManager;
        readonly RoleManager<IdentityRole<Guid>> _roleManager;
        private readonly SecurityDBContext _securityDBContext;
        private readonly ITokenGenerator _tokenGenerator;
        public UserService(UserManager<User> userManager, RoleManager<IdentityRole<Guid>>
roleManager, SecurityDBContext securityDBContext, ITokenGenerator tokenGenerator)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _securityDBContext = securityDBContext;
            _tokenGenerator = tokenGenerator;
        }
        public Task<string> CallBackUrl(User user, string code)
        {
            var ngrok = Constants.ngrok;
            var callbackUrl = ngrok + "/api/users/verification" +
$"?userId={user.Id}&code={code}";

            return Task.FromResult(callbackUrl);
        }
        public Task<bool> UserExist(string email)
           => _securityDBContext.Users.AnyAsync(u => u.Email == email);
        public async Task<UserDTO> CreateUser(RegisterDTO registerDTO)
        {
            var checkByEmail = await UserExist(registerDTO.Email);
            if (checkByEmail)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status400BadRequest,
                    Title = "User exist",
                    Detail = "User with this email already exists"
                };
            }
            var user = new User()
            {
                UserName = registerDTO.Email,
                Email = registerDTO.Email
```

```
            };
            var createUserResult = await _userManager.CreateAsync(user, registerDTO.Password);
        }
        public async Task<UserDTO> GetUserById(string id)
        {
            var user = await _userManager.FindByIdAsync(id);
            return new UserDTO()
            {
                Id = user.Id,
                Email = user.Email
            };
        }
        public async Task<UserDTO> GetUserByEmail(string email)
        {
            var user = await _securityDBContext.Users.FirstOrDefaultAsync(i => i.Email ==
email);
            return new UserDTO()
            {
                Id = user.Id,
                Email = user.Email,
            };
        }
        private async Task<bool> SendEmail(User user)
        {
            var emailConfirmationToken = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
            var emailConfirmationUrl = await CallBackUrl(user, emailConfirmationToken);
            try
            {
                var email = new MimeMessage();
                email.From.Add(MailboxAddress.Parse(Constants.SenderEmail));
                email.To.Add(MailboxAddress.Parse(user.Email));
                email.Subject = "Email verification";
                email.Body = new TextPart(TextFormat.Html)
                {
                    Text = EmailTemplateGenerator.GenerateEmailTemplate(emailConfirmationUrl)
                };
                using var client = new SmtpClient();
                string mySmptServerAddres = "smtp.gmail.com";
                int mySmptPort = 587;
                await client.ConnectAsync(mySmptServerAddres, mySmptPort,
SecureSocketOptions.StartTls);
                await client.AuthenticateAsync(Constants.SenderEmail, Constants.Password);
                await client.SendAsync(email);
                await client.DisconnectAsync(true);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
        public async Task<bool> VerifyEmail(UserDTO user, string token)
        {
            var userVerify = await _userManager.FindByEmailAsync(user.Email);
            var result = await _userManager.ConfirmEmailAsync(userVerify, token);

            return result.Succeeded;
        }
        private async Task<string> AddUserRoleAsync(User user, Role roleName)
        {
            var isAddedUserRole = await _userManager.AddToRoleAsync(user,
roleName.ToString());

            if (isAddedUserRole.Succeeded)
            {
                var roles = await _userManager.GetRolesAsync(user);
```

```
                return roles.First();
            }
            else
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "Error adding role",
                    Detail = "Role doesn't added"
                };
            }
        }
        public async Task<bool> DeleteUserById(Guid id)
        {
            string userId = id.ToString();
            var foundUser = await _userManager.FindByIdAsync(userId);

            if (foundUser == null)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status404NotFound,
                    Title = "User not found",
                    Detail = "User not found"
                };
            }
            var deletionResult = await _userManager.DeleteAsync(foundUser);
            return deletionResult.Succeeded;
        }
        public async Task<AuthenticationResponse> GenerateTokens(string email)
        {
            var user = await _userManager.FindByEmailAsync(email);
            return await _tokenGenerator.GenerateTokens(user);
        }
        public async Task<bool> CheckPassword(string email, string password)
        {
            var userByEmail = await _userManager.FindByEmailAsync(email);
            return await _userManager.CheckPasswordAsync(userByEmail, password);
        }
    }
}
namespace VolunteerAppSecurity
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            var builder = WebApplication.CreateBuilder(args);
            builder.Services.AddControllers(options =>
            {
                options.Filters.Add(typeof(ExceptionResponseFilter));
            });
            builder.Services.AddValidatorsFromAssemblyContaining<EmailDTOValidator>();
            builder.Services.AddValidatorsFromAssemblyContaining<LoginDTOValidator>();
            builder.Services.AddValidatorsFromAssemblyContaining<RegisterDTOValidator>();
            builder.Services.AddEndpointsApiExplorer();
            builder.Services.AddSwaggerGen();

            builder.Services.AddDbContext<SecurityDBContext>(options =>
            {
                options.UseSqlServer(builder.Configuration.GetConnectionString("ConStr"));
            });
            builder.Services.AddScoped<IUserService, UserService>();
            builder.Services.AddScoped<ITokenGenerator, TokenGenerator>();
            builder.Services.AddIdentity<User, IdentityRole<Guid>>()
                .AddEntityFrameworkStores<SecurityDBContext>()
                .AddDefaultTokenProviders();
```

```
            builder.Services.AddCors(options =>
            {
                options.AddPolicy("AllowMyOrigins", policy =>
                {
                    policy
                    .AllowAnyOrigin()
                    .AllowAnyHeader()
                    .AllowAnyMethod();
                });
            });

builder.Services.Configure<AuthenticationSetup>(builder.Configuration.GetSection("JwtOptions")
);
            builder.Services.AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            })
                .AddJwtBearer(options =>
                {
                    options.SaveToken = true;
                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateIssuer = true,
                        ValidIssuer = builder.Configuration["JwtOptions:Issuer"],

                        ValidateAudience = true,
                        ValidAudience = builder.Configuration["JwtOptions:Audience"],

                        ValidateLifetime = true,

                        ValidateIssuerSigningKey = true,
                        ClockSkew = TimeSpan.Zero,
                        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(builder.Configuration["JwtOptions:SecretKey"])),
                    };
                });
            builder.Host.UseSerilog((context, configuration) =>
                configuration.ReadFrom.Configuration(context.Configuration));
            var app = builder.Build();
            if (args.Length == 1 && args[0].ToLower() == "seeddata")
                SeedData(app);

            void SeedData(IHost app)
            {
                var scopedFactory = app.Services.GetService<IServiceScopeFactory>();

                using (var scope = scopedFactory.CreateScope())
                {
                    var service = scope.ServiceProvider.GetService<DataSeeder>();
                    service.SeedAdmins();
                }
            }
            if (app.Environment.IsDevelopment())
            {
                app.UseSwagger();
                app.UseSwaggerUI();
            }
            else
            {
                app.UseExceptionHandler("/Error");
            }
            app.UseRouting();
            app.UseCors("AllowMyOrigins");
            app.UseSerilogRequestLogging();
            app.UseHttpsRedirection();
```

```
            app.UseAuthentication();
            app.UseAuthorization();
            app.MapControllers();
            using (var scope = app.Services.CreateScope())
            {
                var roleManager =
scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole<Guid>>>();

                var roles = new[] { "Admin", "Volunteer", "Organization" };
                foreach (var role in roles)
                {
                    if (!await roleManager.RoleExistsAsync(role))
                        await roleManager.CreateAsync(new IdentityRole<Guid>(role));
                }
            }
            app.Run();
        }
    }
}
namespace Volunteer.BL.Helper.Exceptions
{
    public class ExceptionResponseFilter : IExceptionFilter
    {
        private readonly ILogger<ExceptionResponseFilter> _logger;
        public ExceptionResponseFilter(ILogger<ExceptionResponseFilter> logger)
        {
            _logger = logger;
}
namespace Volunteer.BL.Interfaces
{
    public interface IJobOfferService
    {
        Task CreateJobOffer(Guid id, JobOfferCreateDTO jobOfferDTO);
        Task<PagedPesponse<List<JobOfferDTO>>> GetAllJobOffers(
            PaginationFilter paginationFilter,
            CancellationToken cancellationToken = default);
        Task<int> GetJobOfferCount(CancellationToken cancellationToken = default);
        Task<ShowJobOfferDTO> GetJobOfferInfo(Guid id);
        Task<JobOffer> GetGobOfferById(Guid id);
        Task UpdateJobOffer(Guid id, JobOfferUpdateDTO jobOfferDTO);
        Task<bool> DeleteJobOffer(Guid id);
        Task<List<IdName>> GetCities(
            CancellationToken cancellationToken = default);
        Task<List<IdName>> GetCountries(
            CancellationToken cancellationToken = default);
        Task<List<IdName>> GetStreets(
            CancellationToken cancellationToken = default);
    }
namespace Volunteer.BL.Services;
public class FeedbackService : IFeedbackService
{
    private readonly VolunteerDBContext _dbContext;
    public FeedbackService(VolunteerDBContext dbContext)
    {
        _dbContext = dbContext;
    }
    public async Task<bool> AddFeedback(
        Guid volunteerId,
        FeedbackDTO feedbackDTO,
        CancellationToken cancellationToken = default)
    {
        var feedback = new Feedback
        {
            Id = Guid.NewGuid(),
            OrganizationId = feedbackDTO.OrganizationId,
            VolunteerId = volunteerId,
            Description = feedbackDTO.Comment,
```

```
            Rating = feedbackDTO.Rating
        };
        await _dbContext.Feedbacks.AddAsync(feedback, cancellationToken);
        return await SaveChangesAsync();
    }
    public async Task<PagedPesponse<List<FeedbackPaginationDTO>>> GetFeedbacks(
        PaginationFilter filter,
        CancellationToken cancellationToken = default)
    {
        var query = _dbContext.Feedbacks
            .Include(t => t.Member)
            .ThenInclude(t => t.Volunteer)
            .Where(t => t.OrganizationId == filter.OrganizationId);
        var countRecords = await query
            .CountAsync(cancellationToken);
        query = query
            .Skip(filter.PageNumber * filter.PageSize)
            .Take(filter.PageSize);
        var result = await query
            .Select(p => new FeedbackPaginationDTO
            {
                Id = p.Id,
                Comment = p.Description,
                Rating = p.Rating
            })
            .ToListAsync(cancellationToken);
        return new PagedPesponse<List<FeedbackPaginationDTO>>(
            result,
            countRecords,
            filter.PageNumber, filter.PageSize);
    }
    private async Task<bool> SaveChangesAsync()
        => await _dbContext.SaveChangesAsync() > 0;
}
namespace Volunteer.BL.Services
{
    public class JobOfferService : IJobOfferService
    {
        private readonly VolunteerDBContext _dbContext;
        public JobOfferService(VolunteerDBContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task CreateJobOffer(Guid organizationId, JobOfferCreateDTO jobOfferDTO)
        {
            var jobOffer = new JobOffer
            {
                Id = Guid.NewGuid(),
                OrganizationId = organizationId,
                Title = jobOfferDTO.Title,
                Street = jobOfferDTO.Street,
                City = jobOfferDTO.City,
                Country = jobOfferDTO.Country,
                Description = jobOfferDTO.Description,
                DateTime = jobOfferDTO.DateTime,
                CreatedOn = DateTime.UtcNow,
            };
            await _dbContext.JobOffer.AddAsync(jobOffer);
            await _dbContext.SaveChangesAsync();
        }
        public async Task UpdateJobOffer(Guid id, JobOfferUpdateDTO jobOfferDTO)
        {
            var jobOffer = await GetGobOfferById(id);
            if (jobOffer == null)
            {
                throw new ApiException()
                {
```

```
            StatusCode = StatusCodes.Status404NotFound,
            Title = "JobOffer doesn't exist",
            Detail = "No jobOffer on database"
        };
    }
    jobOffer.Title = jobOfferDTO.Title;
    jobOffer.Street = jobOfferDTO.Street;
    jobOffer.City = jobOfferDTO.City;
    jobOffer.Country = jobOfferDTO.Country;
    jobOffer.Description = jobOfferDTO.Description;
    jobOffer.DateTime = jobOfferDTO.DateTime;
    _dbContext.JobOffer.Update(jobOffer);
    await _dbContext.SaveChangesAsync();
}
public async Task<bool> DeleteJobOffer(Guid id)
{
    var jobOffer = await GetGobOfferById(id);

    if (jobOffer == null)
    {
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status404NotFound,
            Title = "JobOffer doesn't exist",
            Detail = "No jobOffer on database"
        };
    }
    _dbContext.JobOffer.Remove(jobOffer);
    return await SaveChangesAsync();
}
public async Task<PagedPesponse<List<JobOfferDTO>>> GetAllJobOffers(
    PaginationFilter filter,
    CancellationToken cancellationToken)
{
    IQueryable<JobOffer> query = _dbContext.JobOffer;
    if (filter.OrganizationId != null)
    {
        query = query.Where(o => o.OrganizationId == filter.OrganizationId);
    }
    if (!string.IsNullOrEmpty(filter.SearchCriteria))
    {
        query = query.Where(s => s.Title.Contains(filter.SearchCriteria));
    }
    if (filter.SortDirection.HasValue)
    {
        switch (filter.SortColumn)
        {
            case "CreatedOn":
                query = filter.SortDirection.Value == SortOrder.Asc
                    ? query.OrderBy(jo => jo.CreatedOn).ThenBy(jo => jo.Id)
                    : query.OrderByDescending(jo => jo.CreatedOn).ThenBy(jo => jo.Id);
                break;
            case "DateTime":
                query = filter.SortDirection.Value == SortOrder.Asc
                    ? query.OrderBy(jo => jo.DateTime).ThenBy(jo => jo.Id)
                    : query.OrderByDescending(jo => jo.DateTime).ThenBy(jo => jo.Id);
                break;
            default:
                query = query.OrderBy(t => t.Id);
                break;
        }
    }
    else
    {
        query = query.OrderBy(t => t.Id);
    }
    var countRecords = await query.CountAsync(cancellationToken);
```

```
            query = query
                .Skip(filter.PageNumber * filter.PageSize)
                .Take(filter.PageSize);
            var result = await query
                .Select(jo => new JobOfferDTO
                {
                    Id = jo.Id,
                    Title = jo.Title,
                    Street = jo.Street,
                    City = jo.City,
                    Country = jo.Country,
                    DateTime = jo.DateTime,
                    Description = jo.Description,
                    CreatedOn = jo.CreatedOn,
                })
                .ToListAsync(cancellationToken);
            return new PagedPesponse<List<JobOfferDTO>>(
                result,
                countRecords,
                filter.PageNumber,
                filter.PageSize);
        }
        public async Task<int> GetJobOfferCount(CancellationToken cancellationToken = default)
        {
            IQueryable<JobOffer> query = _dbContext.JobOffer;
            var countRecords = await query.CountAsync(cancellationToken);
            return countRecords;
        }
        public async Task<ShowJobOfferDTO> GetJobOfferInfo(Guid id)
        {
            var jobOffer = await _dbContext.JobOffer.FirstOrDefaultAsync(i => i.Id == id);
            var organization = await _dbContext.Organizations
                .FirstOrDefaultAsync(o => o.Id == jobOffer.OrganizationId);
            if (jobOffer == null)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status404NotFound,
                    Title = "JobOffer doesn't exist",
                    Detail = "No jobOffer on database"
                };
            }
            return new ShowJobOfferDTO
            {
                Id = id,
                DateTime = jobOffer.DateTime,
                Description = jobOffer.Description,
                City = jobOffer.City,
                Country = jobOffer.Country,
                Street = jobOffer.Street,
                Title = jobOffer.Title,
                OrganizationId = organization.Id,
                OrganizationName = organization.Name
            };
        }
        public async Task<JobOffer> GetGobOfferById(Guid id)
        {
            return await _dbContext.JobOffer.FirstOrDefaultAsync(i => i.Id == id);
        }
        public async Task<List<IdName>> GetCountries(CancellationToken cancellationToken =
default)
        {
            var countries = await _dbContext.JobOffer
                .GroupBy(jo => jo.Country)
                .Select(g => new
                {
                    Country = g.Key,
```

```
                    Count = g.Count()
                })
                .OrderByDescending(g => g.Count)
                .Take(5)
                .ToListAsync(cancellationToken);

            var countryList = countries.Select( c => new IdName
            {
                Id = Guid.NewGuid(),
                Name = c.Country
            }).ToList();

            return countryList;
        }
        private async Task<bool> SaveChangesAsync()
            => await _dbContext.SaveChangesAsync() > 0;
    }
}
    public async Task<PagedPesponse<List<JobOfferRequestDTO>>> GetJobOffersRequests(
        Guid volunteerId,
        PaginationFilter filter,
        CancellationToken cancellationToken)
    {
        var jobOffersByVolunteerQuery = _dbContext.VolunteerJobOffers
            .Include(t => t.JobOffer)
           .Where(t => t.VolunteerId == volunteerId);
        var countRecords = await jobOffersByVolunteerQuery
            .CountAsync(cancellationToken);
        jobOffersByVolunteerQuery = jobOffersByVolunteerQuery
            .Skip(filter.PageNumber * filter.PageSize)
            .Take(filter.PageSize);
        var result = await jobOffersByVolunteerQuery
            .Select(p => new JobOfferRequestDTO
            {
                Id = p.JobOfferId,
                Title = p.JobOffer.Title,
                Street = p.JobOffer.Street,
                City = p.JobOffer.City,
                Country = p.JobOffer.Country,
                DateTime = p.JobOffer.DateTime,
                Status = p.Status
            })
            .ToListAsync(cancellationToken);
        return new PagedPesponse<List<JobOfferRequestDTO>>(
            result,
            countRecords,
            filter.PageNumber, filter.PageSize);
    }
    public async Task<StatusRequest> GetStatus(Guid volunteerId, Guid offerId)
    {
        var status = await _dbContext.VolunteerJobOffers
            .Where(t => t.VolunteerId == volunteerId && t.JobOfferId == offerId).FirstAsync();
        if (status == null)
        {
            throw new ApiException()
            {
                StatusCode = StatusCodes.Status404NotFound,
                Title = "No record in database",
                Detail = "There is no record in the database with such ids"
            };
        }
        return status.Status;
    }
    private async Task<bool> SaveChangesAsync()
        => await _dbContext.SaveChangesAsync() > 0;
        public static string GetStaticContentDirectory()
        {
```

```
            var result = "C:\\Users\\Diana\\Exoft\\ResumeUploads";
            if (!Directory.Exists(result))
            {
                Directory.CreateDirectory(result);
            }
            return result;
        }
        public static string GetFilePath(string fileName)
        {
            var _GetStaticContentDirectory = GetStaticContentDirectory();
            return Path.Combine(_GetStaticContentDirectory, fileName);
        }
        public async Task<bool> SaveChangesAsync()
           => await _dbContext.SaveChangesAsync() > 0;
    }
}
namespace Volunteer.BL.Services
{
    public class StatusHistoryService : IStatusHistoryService
    {
        private readonly VolunteerDBContext _dbContext;
        public StatusHistoryService(VolunteerDBContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task<PagedPesponse<List<OrganizationDTO>>> GetAllOrganizations(
            PaginationFilter filter,
            CancellationToken cancellationToken)
        {
            IQueryable<Organization> query = _dbContext.Organizations;
            query = filter.SortColumn switch
            {
                "Id" when filter.SortDirection == SortOrder.Asc => query
                    .OrderBy(p => p.Id),
                "Id" => query.OrderByDescending(p => p.Id),
                _ => query
            };
            var countRecords = await query
              .CountAsync(cancellationToken);
            query = query
                .Skip(filter.PageNumber * filter.PageSize)
                .Take(filter.PageSize);
            var result = await query
                .Select(p => new OrganizationDTO
                {
                    Id = p.Id,
                    Name = p.Name,
                    YearOfFoundation = p.YearOfFoundation
                })
                .ToListAsync(cancellationToken);
            return new PagedPesponse<List<OrganizationDTO>>(
                result,
                countRecords,
                filter.PageNumber, filter.PageSize);
        }
    }
}
namespace Volunteer.BL.Services
{
    public class VolunteerService : IVolunteerService
    {
        private readonly VolunteerDBContext _dbContext;

        public VolunteerService(VolunteerDBContext dbContext)
        {
            _dbContext = dbContext;
        }
```

```
public async Task AddVolunteer(
    Guid id,
    VolunteerCreateDTO volunteerDTO,
    CancellationToken cancellationToken = default)
{
    var volunteerExists = await GetVolunteerById(id);
    if(volunteerExists != null)
    {
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status400BadRequest,
            Title = "Volunteer already exists",
            Detail = "A volunteer with this id already exists"
        };
    }

    var volunteer = new DAL.Models.Volunteer
    {
        Id = id,
        DateOfBirth = volunteerDTO.DateOfBirth,
        Description = volunteerDTO.Description,
        FirstName = volunteerDTO.FirstName,
        LastName  = volunteerDTO.LastName
    };

    await _dbContext.Volunteers.AddAsync(volunteer, cancellationToken);
    await _dbContext.SaveChangesAsync(cancellationToken);
}

public async Task<VolunteerDTO> UpdateVolunteer(
    Guid  id,
    VolunteerDTO volunteerDTO,
    CancellationToken cancellationToken = default)
{
    var volunteer = await GetVolunteerById(id);

    volunteer.FirstName = volunteerDTO.FirstName;
    volunteer.LastName = volunteerDTO.LastName;
    volunteer.Description = volunteerDTO.Description;

    _dbContext.Volunteers.Update(volunteer);

    await _dbContext.SaveChangesAsync(cancellationToken);

    return volunteerDTO;
}

public async Task<bool> DeleteVolunteer(Guid volunteerId)
{
    var volunteer = await GetVolunteerById(volunteerId);

    if (volunteer == null)
    {
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status404NotFound,
            Title = "No volunteer exists",
            Detail = "Volunteer does`n exist in the database"
        };
    }

    _dbContext.Volunteers.Remove(volunteer);

    return await SaveChangesAsync();
}
```

```
public async Task<VolunteerDTO> GetVolunteer(Guid id)
{
    var volunteer = await GetVolunteerById(id);
    if (volunteer == null)
    {
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status404NotFound,
            Title = "No volunteer exists",
            Detail = "Volunteer does`n exist in the database"
        };
    }
    return new VolunteerDTO
    {
        DateOfBirth = volunteer.DateOfBirth,
        FirstName = volunteer.FirstName,
        LastName = volunteer.LastName,
        Description = volunteer.Description
    };
}

public async Task<PagedPesponse<List<VolunteerPaginationDTO>>> GetVolunteers(
    PaginationFilter filter,
    CancellationToken cancellationToken)
{
    IQueryable<DAL.Models.Volunteer> query = _dbContext.Volunteers;
    if (filter.SearchCriteria != null)
    {
        query = query.Where(s =>
            s.LastName.Contains(filter.SearchCriteria) ||
            s.FirstName.Contains(filter.SearchCriteria)
        );
    }
    query = filter.SortColumn switch
    {
        "Id" when filter.SortDirection == SortOrder.Asc => query
            .OrderBy(p => p.Id),
        "Id" => query.OrderByDescending(p => p.Id),
        _ => query
    };

    var countRecords = await query
        .CountAsync(cancellationToken);

    query = query
        .Skip(filter.PageNumber * filter.PageSize)
        .Take(filter.PageSize);
    var result = await query
        .Select(p => new VolunteerPaginationDTO
        {
            Id = p.Id,
            DateOfBirth = p.DateOfBirth,
            FirstName = p.FirstName,
            LastName = p.LastName,
            Description = p.Description
        })
        .ToListAsync(cancellationToken);

    return new PagedPesponse<List<VolunteerPaginationDTO>>(
        result,
        countRecords,
        filter.PageNumber, filter.PageSize);
}
public async Task<bool> IsMember(Guid organizationId, Guid volunteerId)
{
    var isMember = await _dbContext.Members
```

```
                .AnyAsync(m => m.VolunteerId == volunteerId && m.JobOffer.OrganizationId ==
organizationId);
            if (isMember)
            {
                return true;
            }
            return false;
        }
        private async Task<DAL.Models.Volunteer> GetVolunteerById(Guid id)
        {
            return await _dbContext.Volunteers.FirstOrDefaultAsync(i => i.Id == id);
       }
        private async Task<bool> SaveChangesAsync()
            => await _dbContext.SaveChangesAsync() > 0;
    }
}
namespace Volunteer.DAL.DataAccess
{
    public class VolunteerDBContext : DbContext
    {
        public DbSet<Feedback> Feedbacks { get; set; }
        public DbSet<JobOffer> JobOffer { get; set; }
        public DbSet<Member> Members { get; set; }
        public DbSet<Organization> Organizations {get; set; }
        public DbSet<Resume> Resumes { get; set; }
        public DbSet<Models.Volunteer> Volunteers { get; set; }
        public DbSet<VolunteerJobOffer> VolunteerJobOffers { get; set; }
        public VolunteerDBContext(DbContextOptions<VolunteerDBContext> option) : base(option)
        {}
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {

optionsBuilder.UseSqlServer("Server=localhost;Database=VolunteerDB;TrustServerCertificate=True
");
            optionsBuilder.EnableSensitiveDataLogging();
            base.OnConfiguring(optionsBuilder);
        }
        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.ApplyConfiguration(new FeedbackConfiguration());
            builder.ApplyConfiguration(new JobOfferConfiguration());
            builder.ApplyConfiguration(new MemberConfiguration());
            builder.ApplyConfiguration(new OrganizationConfiguration());
            builder.ApplyConfiguration(new ResumeConfiguration());
            builder.ApplyConfiguration(new VolunteerConfiguration());
            builder.ApplyConfiguration(new VolunteerJobOfferConfiguration());
            base.OnModelCreating(builder);
        }
    }
namespace Volunteer.DAL.DataAccess.Configurations
{
    public class JobOfferConfiguration : IEntityTypeConfiguration<JobOffer>
    {
        public void Configure(EntityTypeBuilder<JobOffer> builder)
        {
            builder.HasMany(x => x.Members)
                .WithOne(x => x.JobOffer)
                .HasForeignKey(x => x.JobOfferId);

            builder.HasKey(x => x.Id);
        }
    }
}
namespace Volunteer.DAL.DataAccess.Configurations
{
    public class MemberConfiguration : IEntityTypeConfiguration<Member>
    {
```

```
        public void Configure(EntityTypeBuilder<Member> builder)
        {
            builder.HasOne(x => x.Feedback)
                .WithOne(x => x.Member)
                .HasForeignKey<Member>(x => x.FeedbackId);
            builder.HasKey(x => x.Id);
        }
    }
}
namespace Volunteer.DAL.DataAccess.Configurations
{
    public class OrganizationConfiguration : IEntityTypeConfiguration<Organization>
    {
        public void Configure(EntityTypeBuilder<Organization> builder)
        {
            builder.HasMany(x => x.JobOffers)
                 .WithOne(x => x.Organization)
                 .HasForeignKey(x => x.OrganizationId);

            builder.HasKey(x => x.Id);
        }
    }
}
namespace Volunteer.DAL.DataAccess.Configurations
{
    public class VolunteerConfiguration : IEntityTypeConfiguration<Models.Volunteer>
    {
        public void Configure(EntityTypeBuilder<Models.Volunteer> builder)
        {
            builder.HasMany(x => x.Members)
                .WithOne(x => x.Volunteer)
                .HasForeignKey(x => x.VolunteerId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.HasKey(x => x.Id);
        }
    }
}
namespace Volunteer.DAL.DataAccess.Configurations
{
    public class VolunteerJobOfferConfiguration : IEntityTypeConfiguration<VolunteerJobOffer>
    {
        public void Configure(EntityTypeBuilder<VolunteerJobOffer> builder)
        {
            builder.HasKey(x => new { x.VolunteerId, x.JobOfferId });

            builder.HasOne(x => x.Volunteer)
                .WithMany(x => x.VolunteerJobOffers)
                .HasForeignKey(x => x.VolunteerId);

            builder.HasOne(x => x.JobOffer)
                .WithMany(x => x.VolunteerJobOffers)
                .HasForeignKey(x => x.JobOfferId);
        }
    }
}
namespace Volunteer.WebAPI.Controllers;
[ApiController]
[Route("api/feedbacks")]
public class FeedbackController : ControllerBase
{
    private readonly ICurrentUserService currentUserService;
    private readonly IFeedbackService feedbackService;

    public FeedbackController(ICurrentUserService currentUserService, IFeedbackService
feedbackService)
    {
```

```
        this.currentUserService = currentUserService;
        this.feedbackService = feedbackService;
    }
    [Authorize(Roles = "Volunteer")]
    [HttpPost]
    public async Task<IActionResult> AddFeedback(FeedbackDTO feedbackDTO, CancellationToken
cancellationToken = default)
    {
        var volunteerId = currentUserService.GetIdFromClaims(HttpContext.User);
        var result = await feedbackService.AddFeedback(volunteerId, feedbackDTO,
cancellationToken);
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status500InternalServerError,
            Title = "Error retrieving data from the database",
            Detail = "Error retrieving data from the database"
        };
    }
    [Authorize(Roles = "Organization, Volunteer, Admin")]
    [HttpGet]
    public async Task<IActionResult> GetListFeedbacks([FromQuery] PaginationFilter filter,
CancellationToken cancellationToken = default)
    {
        var feedbacks = await feedbackService.GetFeedbacks(filter, cancellationToken);
        throw new ApiException()
        {
            StatusCode = StatusCodes.Status500InternalServerError,
            Title = "Error retrieving data from the database",
            Detail = "Error retrieving data from the database"
        };
    }
}
namespace Volunteer.WebAPI.Controllers
{
    [ApiController]
    [Route("api/job-offers")]
    public class JobOfferController : ControllerBase
    {
        private readonly IJobOfferService jobOfferService;
        private readonly ICurrentUserService currentUserService;

        public JobOfferController(IJobOfferService jobOfferService, ICurrentUserService
currentUserService)
        {
            this.jobOfferService = jobOfferService;
            this.currentUserService = currentUserService;
        }
        [Authorize(Roles = "Organization")]
        [HttpPost]
        public async Task<IActionResult> CreateJobOffer([FromBody] JobOfferCreateDTO
jobOfferDTO)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var organizationId = currentUserService.GetIdFromClaims(HttpContext.User);
            await jobOfferService.CreateJobOffer(organizationId, jobOfferDTO);

            return Ok();
        }
        [Authorize(Roles = "Organization")]
        [HttpPut("{id:Guid}")]
        public async Task<IActionResult> UpdateJobOffer([FromRoute] Guid id, [FromBody]
JobOfferUpdateDTO jobOfferInfoDTO)
        {
```

```
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            await jobOfferService.UpdateJobOffer(id, jobOfferInfoDTO);

            return Ok();
        }
        [Authorize(Roles = "Organization")]
        [HttpDelete("{id:Guid}")]
        public async Task<IActionResult> DeleteJobOffer([FromRoute] Guid id)
        {
            if (await jobOfferService.DeleteJobOffer(id))
            {
                return NoContent();
            }
            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Offer is not deleted",
                Detail = "Error occured while deleting offer on server"
            };
        }

        [Authorize(Roles = "Organization, Volunteer")]
        [HttpGet("{id:Guid}")]
        public async Task<IActionResult> GetJobOffer([FromRoute] Guid id)
        {
            var jobOffer = await jobOfferService.GetJobOfferInfo(id);

            return Ok(jobOffer);
        }
        [Authorize(Roles = "Volunteer, Organization, Admin")]
        [HttpPost("filter")]
        public async Task<IActionResult> GetJobOffers([FromBody] PaginationFilter filter,
CancellationToken cancellationToken = default)
        {
           var jobOffers = await jobOfferService.GetAllJobOffers(filter, cancellationToken);

            if (jobOffers != null)
            {
                return Ok(jobOffers);
          }
            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Error retrieving data from the database",
                Detail = "Error retrieving data from the database"
            };
        }

        [Authorize(Roles = "Volunteer, Organization, Admin")]
        [HttpGet("count")]
        public async Task<IActionResult> GetCountJobOffers(CancellationToken cancellationToken
= default)
        {
            var count = await jobOfferService.GetJobOfferCount(cancellationToken);

            return Ok(count);
        }
       [Authorize(Roles = "Volunteer, Organization, Admin")]
        [HttpGet("cities")]
        public async Task<IActionResult> GetCities(CancellationToken cancellationToken =
default)
        {
            var cities = await jobOfferService.GetCities(cancellationToken)
```

```
            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Error retrieving data from the database",
                Detail = "Error retrieving data from the database"
            };
        }

       [Authorize(Roles = "Volunteer, Organization, Admin")]
        [HttpGet("countries")]
        public async Task<IActionResult> GetCountries(CancellationToken cancellationToken =
default)
        {
            var countries = await jobOfferService.GetCountries(cancellationToken);

            if (countries != null)
            {
                return Ok(countries);
            }

            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Error retrieving data from the database",
                Detail = "Error retrieving data from the database"
            };
        }

        [Authorize(Roles = "Volunteer, Organization, Admin")]
        [HttpGet("streets")]
        public async Task<IActionResult> GetStreets(CancellationToken cancellationToken =
default)
        {
            var streets = await jobOfferService.GetStreets(cancellationToken);

            if (streets != null)
            {
                return Ok(streets);
            }

            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Error retrieving data from the database",
                Detail = "Error retrieving data from the database"
            };
        }
    }
}
namespace Volunteer.WebAPI.Controllers
{
    [ApiController]
    [Route("api/organization")]
    public class OrganizationController : ControllerBase
    {
        private readonly IOrganizationService organizationService;
        private readonly ICurrentUserService currentUserService;

        public OrganizationController(
            IOrganizationService organizationService,
            ICurrentUserService currentUserService)
        {
            this.organizationService = organizationService;
            this.currentUserService = currentUserService;
        }

        [HttpPost]
```

```
        [Authorize(Roles = "Organization")]
        public async Task<IActionResult> AddOrganization(OrganizationDTO organizationInfoDTO)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var id = currentUserService.GetIdFromClaims(HttpContext.User);

            var existingOrganization = await organizationService.AddOrganization(id,
organizationInfoDTO);

            if (existingOrganization != null)
            {
                return Ok(existingOrganization);
            }
            else
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "Could not add organization",
                    Detail = "Could not add organization"
                };
            }
        }

        [Authorize(Roles = "Organization")]
        [HttpDelete]
        public async Task<IActionResult> DeleteOrganization()
        {
            var id = currentUserService.GetIdFromClaims(HttpContext.User);
            var auth = Request.Headers.Authorization;
            using var client = new HttpClient();

            client.DefaultRequestHeaders.Add("Authorization", auth.ToString());

            var response = await
client.DeleteAsync($"{Constants.ngrok}/api/users?userId={id}");
            if (response.IsSuccessStatusCode)
            {
                if (await organizationService.DeleteOrganization(id))
                {
                    return NoContent();
                }
            }

            throw new ApiException()
            {
                StatusCode = StatusCodes.Status500InternalServerError,
                Title = "Organization is not deleted",
                Detail = "Error occured while deleting orhanization on server"
            };

        }

        [Authorize(Roles = "Organization")]
        [HttpPut]
        public async Task<IActionResult> UpdateOrganization([FromBody] OrganizationDTO
organizationDTO)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }
```

```
            var id = currentUserService.GetIdFromClaims(HttpContext.User);

            var organizationToUpdate = await organizationService.GetOrganizationById(id);

            if (organizationToUpdate == null)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status404NotFound,
                    Title = "Organization doesn't exist",
                    Detail = "No organization on database"
                };
            }
            var organization = await organizationService.UpdateOrganization(id,
organizationDTO);

            if (organization == null)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "Organization doesn't update",
                    Detail = "Failed to update organization"
                };
            }
            else
            {
                return Ok(organization);
            }
        }

        [Authorize(Roles = "Organization, Volunteer, Admin")]
        [HttpGet("{id:Guid}")]
        public async Task<IActionResult> GetOrganization([FromRoute] Guid id)
        {
            var organization = await organizationService.GetOrganizationInfo(id);

            if (organization == null)
            {
                throw new ApiException()
                {
                    StatusCode = StatusCodes.Status500InternalServerError,
                    Title = "An error occurred",
                    Detail = "An error occurred"
                };
            }
            return Ok(organization);
        }
    }
}
namespace Volunteer.WebAPI.Controllers;
[ApiController]
[Route("api/requests")]
public class RequestController : ControllerBase
{
    private readonly ICurrentUserService currentUserService;
    private readonly IRequestService requestService;

    public RequestController(ICurrentUserService currentUserService, IRequestService
requestService)
    {
        this.currentUserService = currentUserService;
        this.requestService = requestService;
    }
```