Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти − магістр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Інформаційна система управління розподілом гуманітарної допомоги з розробкою інтерактивного помічника користувача»

Виконав студент гр. КН-23-м. _____ Лещенко Є.В.

Керівник _____ Рубан С. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н.Рубан С.А.*

_____

« *5* » *липня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-23м *Лещенку Євгенію Валерійовичу*

1. Тема кваліфікаційної роботи: *«Інформаційна система управління розподілом гуманітарної допомоги з розробкою інтерактивного помічника користувача»*

затверджено наказом по університету № *594с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 77с., додатки, презентація у Microsoft PowerPoint (28 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 к.т.н.доц. Рубан С.А.

_____

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.24*
2	*Розділ 1 ОБГРУНТУВАННЯ ІДЕЇ ПРОДУКТУ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ*	*15.07.24*
3	*Розділ 2 РЕАЛІЗАЦІЯ ТЕХНІЧНОЇ СТОРОНИ ПРОЕКТУ*	*15.08.24*
4	*Розділ 3 ПРЕДСТАВЛЕННЯ ПРОГРАМНОГО РІШЕННЯ*	*15.09.24*
5	*Висновки*	*15.10.24*
6	*Оформлення кваліфікаційної роботи*	*20.11.24*
7	*Підготовка презентації та графічного матеріалу*	*28.11.24*
8	*Підготовка доповіді до захисту*	*01.12.24*

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ /Рубан С.А. /

7. Запевнення: *Я, Лещенко Євгеній Валерійович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ / Лещенко Є. В./

ANNOTATION

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program «Computer science» in specialty 122 – «Computer science». – Kryvyi Rih National University, Kryvyi Rih, 2024.

The work consists of an introduction, three chapters, conclusions, a list of used literature from 31 positions. The total volume of the work is 77 pages, of which the main content of the work is set out on 70 pages, includes 36 figures.

The relevance of the work is due to the importance of simplifying communication between funds and people.

The idea of the project is to simplify the administration of the database as much as possible and create the possibility of communication. It will no longer be necessary to leaf through a large number of papers to find a number, or make a large number of calls to invite people to issue humanitarian aid. All this can be done on the website. Volunteered offers a convenient tool for database administration to humanitarian funds, and the ability to sign up for the issuance of humanitarian aid and communicate with these funds to forced temporary migrants.

The aim of the project is to create a website for transparency and facilitation of registration for humanitarian aid, database administration capabilities and record regulation capabilities

The object of the research is information systems and the .net platform.

The subject of the research is the use of the c# programming language to create an information system.

Keywords:

INFORMATION SYSTEM, ASP.NET CORE, ANGULAR, TELEGRAM BOT, .NET, C#, HUMANITARIAN FUNDS, TYPESCRIPT.

АНОТАЦІЯ

Лещенко Є.В. Інформаційна система обліку та видачі гуманітарної допомоги

Кваліфікаційна робота на здобуття ступеня вищої освіти – магістр, за спеціальністю 122 Комп'ютерні науки. Криворізький національний університет, Кривий Ріг, 2024.

Робота складається зі вступу, трьох розділів, висновків, переліку використаної літератури з 31 позиції. Загальний обсяг роботи становить 77 сторінок, з яких основний зміст роботи викладено на 70 сторінках, включає 36 рисунків.

Актуальність роботи зумовлена важливістю спрощення комунікації між фондами та людьми.

Ідея проекту полягає в тому, щоб максимально спростити адміністрування бази даних переселенців та створити можливість комунікації. Більше не буде потрібно листати велику кількість паперів, щоб знайти номер, або робити велику кількість дзвінків, щоб запросити людей на видачу гуманітарної допомоги. Все це можна зробити на сайті. Volunteered пропонує зручний інструмент для адміністрації баз даних гуманітарним фондам, та можливість записуватись на видачу гуманітарної допомоги та комунікувати з цими фондами вимушено тимчасовим переселенцям.

Метою проекту є створення веб-сайту для прозорості та полегшення запису на гуманітарну допомогу, можливості адміністрування бази даних та можливості регламентування записів

Об'єктом дослідження є інформаційні системи та платформа .net.

Предметом дослідження є застосування мови програмування c# для створення інформаційної системи.

Ключові слова:

ІНФОРМАЦІЙНА СИСТЕМА, ASP.NET CORE, ANGULAR, TELEGRAM BOT, .NET, C#, ГУМАНІТАРНІ ФОНДИ, TYPESCRIPT.

ЗМІСТ

ВСТУП

В сучасних умовах України, коли через широкомасштабну агресію російської федерації значна кількість людей змушена покинути свої домівки, гостро постала необхідність в удосконаленні системи надання допомоги тимчасово переміщеним особам. Гуманітарні фонди та організації не були готові до такого напливу нужденних і не мали заздалегідь продуманих інструментів для обліку, управління процесами розподілу допомоги та зв’язку з людьми. Відсутність централізованої системи контролю й координування гуманітарної допомоги ускладнює процес надання підтримки тим, хто її потребує.

Ефективним рішенням цієї проблеми є розробка веб-сайту для керування фондами в межах організації, а також створення Telegram-бота для взаємодії з відвідувачами. Така платформа дозволить фондам оперативно адмініструвати процеси, забезпечить доступ до актуальної інформації, а також допоможе забезпечити чіткий облік видачі допомоги.

Розроблений сайт стане єдиною точкою доступу для координації роботи фондів: користувачі зможуть переглядати інформацію про фонди та розташування, записуватись на отримання допомоги, а також зручно комунікувати з волонтерами. Telegram-бот, у свою чергу, стане додатковим каналом зв’язку з відвідувачами, надаючи можливість запису на допомогу, перегляду інформації та отримання актуальних новин.

З боку фондів ця система задовольнить такі потреби, як: централізоване управління базою даних про переміщених осіб, можливість виконувати CRUD-операції з даними, розміщення контактів фонду, формування списків осіб для роздачі допомоги, а також перегляд історії видачі допомоги кожному з відвідувачів.

Мета даного проєкту – розробити веб-додаток та інтеграцію Telegram-бота для оптимізації процесів роботи гуманітарних фондів, використовуючи набуті під час навчання навички.

Для досягнення цієї мети необхідно виконати такі завдання:

- Розробити функціональні вимоги та дизайн сайту для управління фондами та Telegram-бота для комунікації з відвідувачами.
- Реалізувати веб-додаток та Telegram-бот відповідно до вимог і дизайну.
- Провести тестування функціональності, продуктивності та безпеки платформи.
- Забезпечити належний захист персональних даних.
- Виконати тестування та вдосконалити функціонал на основі отриманих результатів.

Цей проєкт надасть необхідні інструменти для автоматизації та спрощення процесів гуманітарної допомоги, сприяючи злагодженій роботі фондів у сучасних умовах.

РОЗДІЛ 1

ОБГРУНТУВАННЯ ІДЕЇ ПРОДУКТУ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

1.1 Опис потреби, ідеї та мети проекту

З огляду на постійний потік внутрішньо переміщених осіб, постала необхідність ефективної організації безперервної видачі гуманітарної допомоги. Виникає питання щодо вибору програмного середовища для обліку допомоги та реєстрації людей, які її потребують, а також забезпечення постійного зв'язку з ними.

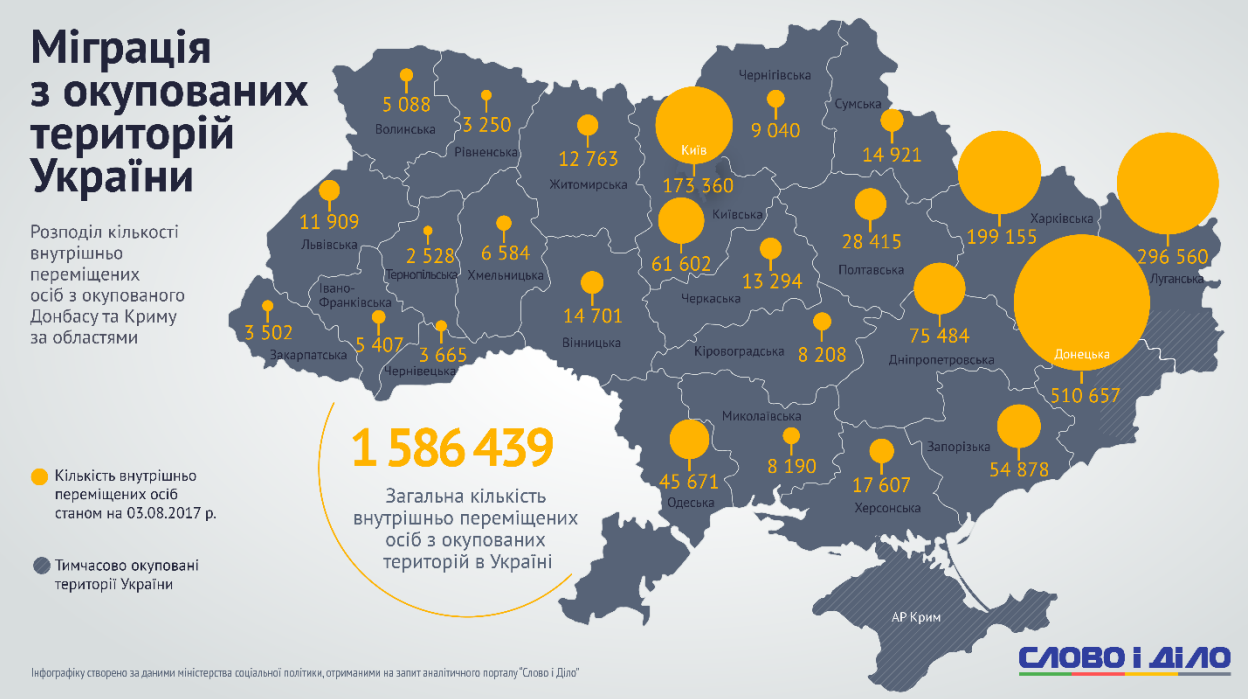

Рисунок 1.1 - Карта міграції з окупованих територій

Багато гуманітарних фондів усе ще покладаються на великий обсяг паперової документації, що призводить до накопичення великої кількості зайвої інформації, яку важко ефективно використовувати. Частина фондів веде облік у середовищі Excel, але цей інструмент не задовольняє всі функціональні вимоги для систематизації допомоги. Крім того, постає питання комунікації з тими, хто потребує підтримки: невеликим командам волонтерів доводиться

особисто телефонувати кожній сім'ї, запрошуючи їх отримати допомогу. Такий підхід є прийнятним для невеликої кількості отримувачів, але в разі значного потоку людей це стає неефективним і займає багато часу.

Мета проекту полягає в оптимізації управління базою даних внутрішньо переміщених осіб і спрощенні комунікації. Більше не потрібно переглядати сотні паперів або здійснювати масові дзвінки для координації видачі гуманітарної допомоги. Все це можна організувати через спеціальний веб-сайт і телеграм-бот. Проект пропонує фондам інструменти для зручного адміністрування бази даних, а також можливість переселенцям записуватись на допомогу та спілкуватися з фондами. Технології веб-панелі керування та телеграм-бота забезпечать простий і зручний доступ до інформації та оптимізують процеси адміністрування.

Проект спрямований на створення веб-сайту, для керування ресурсами гуманітарного фонду та чат-ботом для комунікації з відвідувачами.

1.2 Опис результату

Насамперед можна розділити очікуваний функціонал для двох категорій: гуманітарні фонди та вимушено тимчасові переселенці.

Для вимушено тимчасових переселенців будуть доступні такі функції:

- Реєстрація на сайті
- Перегляд гуманітарних фондів доступних у місті
- Запис на допомогу до обраного гуманітарного фонду
- Перегляд записів на отримання гуманітарної фонду
- Перегляд минулих записів

Для гуманітарних фондів:

- Можливість формування власної бази даних переселенців
- Можливість адміністрування бази даних: перегляд, додавання нових користувачів, видалення користувачів, запис на дату

- Можливість оголосити запис на дату з лімітом в встановлену кількість осіб
- Можливість комунікації з клієнтами через веб-сайт
- Можливість комунікації з клієнтами через пошту
- Можливість переглядати дату отримання гуманітарної допомоги людиною в останній раз

В результаті має вийти зручний сайт зі зрозумілим інтерфейсом та дизайном для користування як і для людей, потребуючих допомоги, так і для гуманітарних фондів.

1.3 Технічне забезпечення проекту

1.3.1 Фреймворк ангуляр

Angular - це сучасний фреймворк для розробки веб-додатків, розроблений компанією Google. Він дозволяє створювати масштабовані та продуктивні веб-застосунки з інтерактивним інтерфейсом. Angular використовує компонентну архітектуру, яка забезпечує логічну структуру коду та дає змогу розбити додаток на незалежні, перевикористовувані компоненти. Завдяки цьому значно полегшується підтримка та розширення проекту, а також спрощується робота команди розробників.

Фреймворк побудований на основі TypeScript - статично типізованої мови, що дозволяє краще контролювати типи змінних і знижує ризик помилок. Angular також підтримує потужні інструменти для роботи з формами, HTTP-запитами, маршрутизацією та обробкою стану додатка. Завдяки вбудованим інструментам для роботи з анімаціями та можливості підключення зовнішніх бібліотек Angular підходить для створення як простих, так і складних корпоративних додатків.

Angular забезпечує відмінну інтеграцію з інструментами тестування, такими як Jasmine і Karma, що дозволяє легко впроваджувати юніт-тести та енд-то-енд тестування. Його двонаправлене прив’язування даних дозволяє

автоматично синхронізувати інтерфейс користувача з моделями даних, що знижує кількість ручної роботи та помилок. Завдяки використанню CLI-інструменту Angular, розробники можуть швидко створювати нові проєкти, керувати залежностями та автоматизувати завдання розробки, такі як збірка, тестування та деплой.

Angular активно використовується для створення складних веб-додатків завдяки своїй високій продуктивності, широким можливостям налаштування та підтримці з боку великої спільноти розробників.

1.3.2 Платформа .Net

.NET - це потужна платформа для розробки програмного забезпечення, яка була створена Microsoft і використовується для побудови різноманітних застосунків: від веб-сайтів і десктопних програм до мобільних додатків і хмарних сервісів. .NET підтримує декілька мов програмування, зокрема C#, F#, і VB.NET, що дозволяє розробникам обирати зручну для себе мову та працювати в єдиному середовищі.

Сучасна платформа .NET (зокрема, .NET Core та .NET 5/6/7/8) є кросплатформеною, що дозволяє розгортати додатки на Windows, Linux і macOS. Однією з ключових особливостей .NET є його продуктивність і здатність забезпечувати високу швидкість роботи навіть для вимогливих до ресурсів систем. Розробники можуть використовувати .NET для створення веб-додатків із допомогою ASP.NET Core, яка надає можливості для створення динамічних веб-сайтів та API, а також підтримує MVC-архітектуру та шаблони REST.

.NET має багатий набір бібліотек і фреймворків, що покривають широкий спектр потреб, від роботи з базами даних (Entity Framework Core) до побудови користувацьких інтерфейсів (WPF та WinForms для десктопних додатків). Завдяки розвиненій інфраструктурі та вбудованим інструментам, таким як Microsoft Visual Studio, .NET сприяє ефективному написанню,

тестуванню та налагодженню коду. Крім того, платформа підтримує зручну інтеграцію з хмарними сервісами, такими як Azure, що дозволяє швидко масштабувати додатки в залежності від потреб бізнесу.

Також .NET надає зручні інструменти для автоматизації CI/CD-процесів, що робить її ідеальною для DevOps-практик. Завдяки великій спільноті, регулярним оновленням і підтримці з боку Microsoft, .NET є однією з найпопулярніших і стабільних платформ для побудови сучасних, продуктивних і надійних додатків.

Entity Framework Core (EF Core) - це об’єктно-реляційний маппер (ORM) для .NET, який дозволяє розробникам працювати з базами даних за допомогою об’єктно-орієнтованого підходу. EF Core є кросплатформеним інструментом, що підтримує різні бази даних, такі як Microsoft SQL Server, PostgreSQL, SQLite, MySQL та інші. Це дозволяє використовувати об’єктно-орієнтовану модель для доступу до даних замість написання складних SQL-запитів вручну.

EF Core дає можливість працювати з базою даних за допомогою концепцій, таких як Code-First, Database-First і Model-First. Це означає, що можна створювати і змінювати структуру бази даних, використовуючи класи C# без необхідності вручну коригувати схему бази даних. Бібліотека також підтримує автоматичне оновлення бази даних через механізм міграцій, що дозволяє синхронізувати зміни в моделі з базою даних.

EF Core дає можливість виконувати запити за допомогою LINQ (Language Integrated Query), що дозволяє писати чистий і зрозумілий код для доступу до даних. Завдяки трекінгу змін, EF Core автоматично відстежує внесені зміни в об’єктах і синхронізує їх із базою даних. Підтримка механізмів лінійного завантаження (Lazy Loading) дозволяє завантажувати пов’язані дані лише за потребою, що може покращити продуктивність. EF Core також надає функції жорсткої типізації, що знижує ймовірність помилок під час роботи з базою даних. EF Core підтримує міграції, що дозволяють легко керувати змінами в схемі бази даних та підтримувати її в актуальному стані.

1.3.3 Telegram Bot API

API Telegram Bot дозволяє розробникам створювати ботів, які можуть взаємодіяти з користувачами в Telegram. Для створення бота на платформі .NET зручно використовувати бібліотеку Telegram.Bot, яка забезпечує доступ до всіх основних функцій Telegram Bot API, включаючи обробку повідомлень, оновлень і команд.

Спочатку необхідно створити бота в Telegram за допомогою бота BotFather, який видасть унікальний токен для підключення. Потім у .NET-проєкті підключається бібліотека Telegram.Bot для інтеграції з API Telegram. За допомогою клієнта TelegramBotClient, створюється підключення до бота з використанням токена. Далі налаштовується обробка оновлень від Telegram, яка включає отримання повідомлень, обробку команд та надсилання відповідей користувачам.

Для безперервної роботи бота рекомендується запускати його на сервері або хмарному середовищі. .NET дозволяє легко інтегрувати бота з іншими сервісами, що розширює його можливості, наприклад, додавати інтеграцію з базами даних, обробляти запити до зовнішніх API або здійснювати складні бізнес-операції. Завдяки потужності та гнучкості .NET, можна створити ефективного та надійного Telegram бота з широким функціоналом.

1.3.4 PostgresSQL

PostgreSQL - це потужна об’єктно-реляційна система управління базами даних з відкритим вихідним кодом, відома своєю надійністю, продуктивністю та відповідністю стандартам SQL. Вона підтримує широкий набір типів даних, включаючи числові, текстові, JSON, геопросторові та користувацькі типи, що робить її універсальною для зберігання різноманітних даних.

PostgreSQL забезпечує розширені функції, такі як транзакції з ACID-гарантіями, потужний механізм обробки запитів, контроль цілісності даних і підтримку паралельного виконання запитів. Вона також підтримує механізми

розширення - можна додавати нові типи даних, індекси, функції та мовні розширення. Завдяки цьому PostgreSQL є гнучкою базою для потреб сучасних застосунків.

Для роботи з великими обсягами даних PostgreSQL пропонує різноманітні типи індексів, а також можливості для шардінгу та реплікації. Вона підтримує як синхронну, так і асинхронну реплікацію, що дозволяє створювати резервні копії та забезпечувати високу доступність системи. Крім того, PostgreSQL має високий рівень безпеки завдяки функціям контролю доступу на основі ролей та можливості шифрування даних.

Завдяки своїй гнучкості, стабільності та розширюваності, PostgreSQL є популярним вибором для підприємств і розробників, яким потрібна надійна, відкрита та функціонально багата система управління базами даних для застосунків будь-якої складності.

1.3.5 Менеджер контролю версій Git

Git - це система контролю версій, яка дозволяє розробникам відслідковувати зміни в коді та співпрацювати над проектами. Вона була створена Лінусом Торвальдсом у 2005 році і з того часу стала найпопулярнішою системою контролю версій у світі розробки програмного забезпечення. Git дозволяє зберігати історію змін, працювати з різними гілками проекту та ефективно керувати внесками кількох розробників у великих і малих проектах.

Git є дистрибутивною системою, що означає, що кожен користувач має повну копію репозиторію з усією історією змін, і не залежить від централізованого серверу. Кожен розробник може працювати локально, вносячи зміни, створюючи нові гілки та зливаючи їх в основну гілку (наприклад, master або main), а потім синхронізувати свою роботу з іншими користувачами через віддалені репозиторії, такі як GitHub або GitLab.

Основною перевагою Git є його гнучкість і потужні інструменти для роботи з гілками, які дозволяють легко відокремлювати різні напрямки роботи над проектом, виправляти помилки та керувати версіями коду. З Git можна легко створювати нові гілки, переключатися між ними, зливати їх і навіть скасувати зміни без втрати даних.

Git також підтримує механізм мерджів, що дозволяє кільком розробникам працювати над однією частиною коду одночасно, а потім автоматично або вручну об’єднувати ці зміни. Для великих проектів з кількома учасниками Git є незамінним інструментом, що дозволяє підтримувати порядок, відслідковувати зміни та швидко реагувати на будь-які проблеми.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ ТЕХНІЧНОЇ СТОРОНИ ПРОЕКТУ

2.1 Описання архітектури бекенду

Проект було вирішено писати - бекенд мовою програмування c# на платформі .NET за допомогою фреймворка ASP.Net Core. За основу було взято стандартну архітектуру REST, БД – Postgres, фронтенд – angular. Було закладено архітектуру за допомогою якої можна легко розширити проект та відстежити помилки.

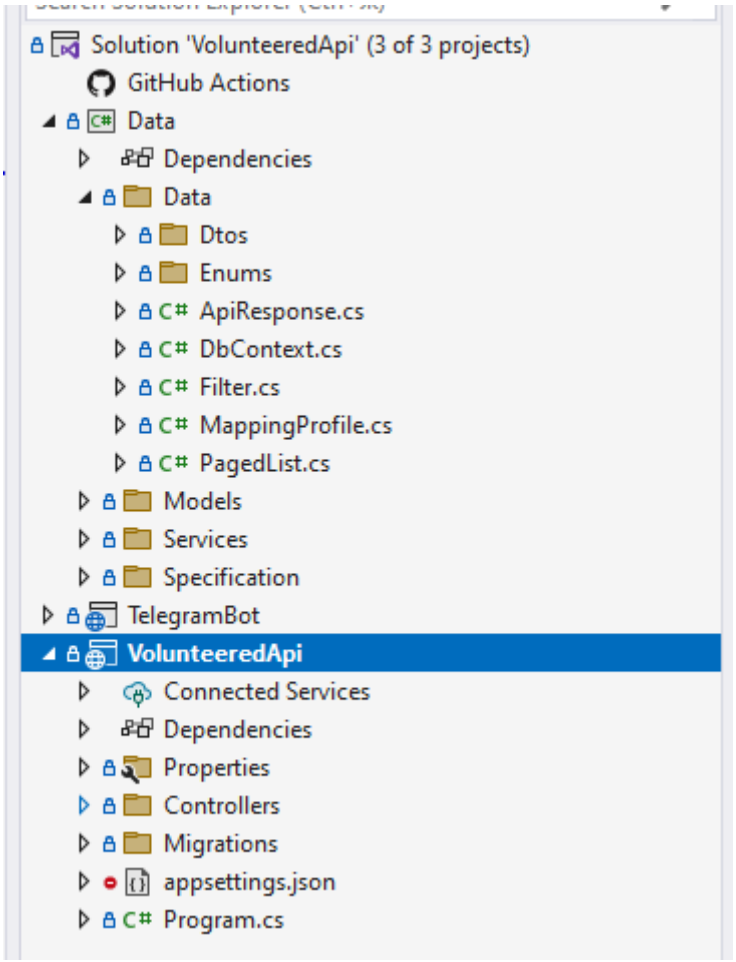

Рисунок 2.1 – Побудова файлової структури проекту

Файлова структура була логічна розділена на три проекти: Data, TelegramBot, VolunteeredApi.

2.1.1 Описання архітектури бази даних

Проект було логічно розділено на біблеотеку класів Data. Моделі бібліотеки класів Data будуть використовуватись як в телеграм боті, так і в веб-додатку. Бібліотека класів дата містить такі теки як: Dtos – моделі для комунікації з фронтом, Базові класи – ApiResponse – базовий клас для відповіді фронту з API. DbContext – базовий клас для ORM Efcore, клас Filter – базовий клас для фільтрації. MappingProfile – файл налаштування автомапера. PagedList – клас для відтворення пагінації на фронті. Міститься тека Migrations – для міграцій. Models – в якій описані всі моделі, які використовуються в проекті. Services – тека яка містить в собі реалізацію всіх інтерфейсів та сервісів, які прокидуються за допомогою DI. Проект VolunteeredApi складається з Controllers – класи контролерів для API. Migrations – зберігаються всі міграції. Program.cs – містить IoC-контейнер та є входом в програму.

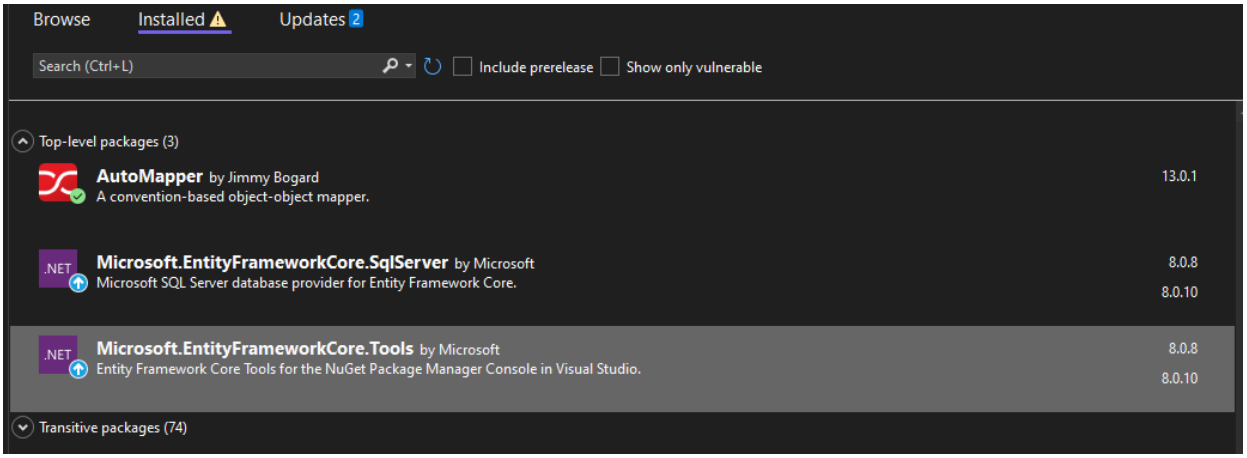

Рисунок 2.2 - Встановлені Nu-get пакети

Automapper - бібліотека, яка дозволяє автоматизувати мапінг (відображення) між об’єктами різних типів у .NET. AutoMapper зменшує кількість коду, необхідного для копіювання даних між різними моделями, і є особливо корисним для перетворення об'єктів бізнес-логіки у DTO та навпаки.

Microsoft.EntityFrameworkCore.SqlServer (версія 8.0.8) - цей пакет є провайдером бази даних SQL Server для Entity Framework Core. Він дозволяє .NET-додаткам підключатися до бази даних SQL Server і працювати з нею,

використовуючи ORM (Object-Relational Mapping) для створення запитів LINQ, виконання операцій CRUD і керування схемою бази даних.

Microsoft.EntityFrameworkCore.Tools (версія 8.0.8) - пакет містить інструменти для Entity Framework Core, які інтегруються з консоллю менеджера пакетів NuGet у Visual Studio. Він надає команди для міграцій, управління схемою бази даних і допоміжних операцій, що полегшує роботу з базами даних під час розробки.

Було додано такі моделі, як:

BaseModel – базова модель від якої будуть спадкуватись всі інші моделі, містить такі поля як:

- public Guid Id { get; set; } – ідентифікатор об’єкта
- public DateTime DateCreated { get; set; } – дата створення
- public DateTime DateUpdated { get; set; } – дата оновлення
- public bool IsDeleted { get; set; } – чи видаленний об’єкт
- Act – модель для опису актів списання:
- public DateTime Date { get; set; } – дата акту
- public string Number { get; set; } – номер акту
- public List<Nomenclature> Nomenclatures { get; set; } – список списаної номеклатури

Appointment – модель для опису видач гуманітарної допомоги

- public DateTime Date { get; set; } – дата
- public string Number { get; set; } – номер зустрічі
- public List<User>? Users { get; set; } – користувачі, які призначені на видачу
- public Guid? FundId { get; set; } – ідентифікатор фонду в якому відбувається видача гуманітарної допомоги
- public Fund? Fund { get; set; } – об’єкт фонду
- Card – Додаткова інформація про користувача
- public string OrganisationCard { get; set; } – карточка організації

Fund – модель для опису фонду, містить такі поля як:

- public string ShortName { get; set; } – коротка назва фонду
- public string FullName { get; set; } – повна назва
- public string Address { get; set; } – адреса фонду
- public string Phone { get; set; } – контактний номер
- public List<User> Users { get; set; } – користувачі, які прив’язані до фонду
- Invoice – для опису прибуткових накладних
- public DateTime Date { get; set; } – дата прибуткової накладної
- public string Number { get; set; } - номер
- public List<Nomenclature> Nomenclatures { get; set; } – список номенклатури, переміщеної на склад;

Menu – модель для опису меню

- public int Level{ get; set; } – рівень меню
- public int ParentId{ get; set; } – батьківський елемент меню
- public bool HasChildren{ get; set; } – чи має елемент меню дочірній елемент;

Nomenclature – модель для опису номенклатури

- public int Number { get; set; } - номер
- public string ShortName { get; set; } – назва номенклатури
- public string Measure { get; set; } – одиниця виміру

Organisation – модель для опису організації

- public string Name { get; set; } – назва
- public List<User> Users {get; set;} – користувачі, які прив’язані до організації
- public List<Settings> Settings { get; set; } – налаштування організації
- Protocol – Протокол видачі гуманітарної допомоги
- public string Number { get; set; } - номер
- public List<User> Users { get; set; } - користувачі
- public Guid AppointmentId { get; set; } – ідентифікатор видачі
- public Appointment Appointment { get; set; } – об’єкт видачі

Settings – модель для опису налаштуваня організації

- public string Name { get; set; } – назва налаштування
- public string Value { get; set; } – значення налаштування
- public Guid? OrganisationId { get; set; } – ідентифікатор організації
- public Organisation? Organisation { get; set; } – об’єкт організації

User – модель для опису користувача

- private string? _referralLink; - для опису реферального посилання, наприклад "https://t.me/volunteered_bot?start={CardNumber}"
- public string Email { get; set; }- пошта користувача
- public string? Password { get; set; } - пароль
- public string? Token { get; set; } - токен
- public long? TelegramId { get; set; } – ідентифікатор чату в телеграмі
- public string? Name { get; set; } – ім’я
- public string? Address { get; set; } - адреса
- public string? CardNumber { get; set; } – номер довідки
- public string? PhoneNumber { get; set; } – номер телефону
- public Guid? OrganisationId { get; set; } – ідентифікатор організації до якої прив’язаний користувач
- public Organisation? Organisation { get; set; } – об’єкт організації до якої прив’язаний користувач
- public Guid? FundId { get; set; } – ідентифікатор фонду
- public Fund? Fund { get; set; } – об’єкт фонду
- public List<Appointment>? Appointments { get; set; } – видачі гуманітарної допомоги, в яких приймав участь користувач

Також було додано допоможні моделі, такі як представлення на клієнті для кожної моделі.

Клас ApiResponse<T> - для відповіді клієнту, містить такі поля як:

- public T Data { get; set; } – дані які повертаються клієнту
- public Paging? Paging { get; set; } - пагінація
- public string Error { get; set; } – помилки, якщо немає, то повертається null

Клас Filter – для фільтрації з клієнта

- public int Page { get; set; } – номер сторінки яка потрібна клієнту
- public int PageSize { get; set; } – об’єм даних
- public string? Criteria { get; set; } – фільтр за критерієм
- public string? Sort { get; set; } – сортування

Клас Paging – для пагінації з клієнта

Клас Paging призначений для реалізації логіки пагінації, яка дозволяє працювати з даними, розбитими на сторінки. Цей клас надає основну інформацію про поточну сторінку, розмір сторінки, загальну кількість елементів та методи для визначення наявності попередньої або наступної сторінки.

Поля та властивості класу

Властивість Page зберігає номер поточної сторінки.

Це значення встановлюється через конструктор і є лише для читання (get-only), що означає, що після ініціалізації об’єкт Paging не дозволяє змінювати номер сторінки.

Важливість цієї властивості полягає в тому, що вона вказує, яку частину даних слід відобразити або обробити.

Властивість PageSize зберігає кількість елементів, які потрібно відобразити на кожній сторінці.

Це значення також встановлюється через конструктор і є лише для читання.

Наявність фіксованого PageSize дозволяє контролювати обсяг даних, що повертаються за раз, і запобігає надмірному навантаженню на систему при роботі з великими обсягами інформації.

Властивість TotalCount зберігає загальну кількість елементів у колекції. Це значення також передається в конструкторі та є лише для читання.

Воно дозволяє обчислити загальну кількість сторінок і дізнатися, чи є ще дані для відображення на наступних сторінках.

HasNextPage (bool) - це обчислювана властивість (get-only), яка повертає логічне значення (true або false), вказуючи, чи існує наступна сторінка.

Логіка реалізації: якщо добуток Page на PageSize менший за TotalCount, це означає, що є ще дані для відображення, і властивість повертає true.

Це особливо корисно для інтерфейсів користувача, які повинні відображати кнопку для переходу на наступну сторінку або індикатор завершення списку даних.

HasPreviousPage (bool) - це обчислювана властивість, яка повертає true, якщо існує попередня сторінка, інакше - false.

Логіка реалізації: якщо PageSize більше 1, значить, існує принаймні одна попередня сторінка, і властивість повертає true.

Це корисно для випадків, коли потрібно відобразити кнопку для повернення на попередню сторінку, допомагаючи користувачам легко переходити між сторінками.

```
public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
{

}

public DbSet<Fund> Funds { get; set; }

public DbSet<Appointment> Appointments { get; set; }

public DbSet<Card> Cards { get; set; }

public DbSet<Menu> Menus { get; set; }

public DbSet<Nomenclature> Nomenclatures { get; set; }

public DbSet<Organisation> Organisations { get; set; }

public DbSet<Protocol> Protocol { get; set; }

public DbSet<Role> Roles { get; set; }

public DbSet<Settings> Settings { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Act> Acts { get; set; }

public DbSet<Invoice> Invoices { get; set; }

public DbSet<AppointmentNomenclature> AppointmentNomenclatures { get; set; }

public DbSet<InvoiceProduct> InvoiceProducts { get; set; }

public DbSet<ActProduct> ActProducts { get; set; }

public DbSet<UserAppointment> UserAppointments { get; set; }



protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Settings>().HasKey(s => s.Id);
    modelBuilder.Entity<Settings>().Property(s => s.Name);
    modelBuilder.Entity<Settings>().Property(s => s.Value);
    modelBuilder.Entity<Settings>().Property(s => s.DateCreated);
    modelBuilder.Entity<Settings>().Property(s => s.DateUpdated);

    modelBuilder.Entity<Settings>().HasOne(s => s.Organisation).WithMany(o => o.Settings);
```

Рисунок 2.3 – Опис моделей бази даних в .NET

2.1.2 Використання патерну «Специфікатор»

На бекенді використовуються патерн специфікатор для побудови фільтрів.

Патерн Специфікатор (Specification Pattern) - це поведінковий патерн, який використовується для побудови складних логічних умов або критеріїв, за якими об'єкти перевіряються на відповідність певним вимогам. Цей патерн часто застосовується в системах, де необхідно реалізувати динамічні запити або фільтрацію об'єктів на основі змінних умов, без жорсткого кодування цих умов у класах.

Основна мета патерну Специфікатор - інкапсулювати логіку, що визначає, чи відповідає об'єкт певним критеріям. Замість того, щоб розміщувати цю логіку безпосередньо у класі, який потребує перевірки (або дублювати її в різних частинах програми), специфікації дозволяють винести її у спеціалізовані об'єкти. Це спрощує підтримку, розширення та тестування коду.

Фільтрація об'єктів - специфікатор дозволяє легко фільтрувати колекції об'єктів. Наприклад, якщо є список продуктів, то можна легко створити специфікації для вибору продуктів певної категорії або цінового діапазону.
Валідатори - Специфікатор може використовуватись як шаблон для перевірки бізнес-правил. Наприклад, в системі управління користувачами можна визначити правила для перевірки прав доступу або дозволів.

Об'єднання умов - Специфікатор зручний для побудови складних умов через комбінування специфікацій. Наприклад, можна створити специфікацію, яка буде поєднувати інші за допомогою логічних операторів "І", "АБО" або "НЕ". Це дає можливість будувати складні фільтри з простих умов.
Інкапсуляція бізнес-логіки - Патерн дозволяє відокремити бізнес-логіку, яка визначає правила перевірки, від класів, які повинні їх виконувати. Це забезпечує низьку зв'язаність коду і спрощує його розширюваність.

Переваги патерну Специфікатор:

Гнучкість - Можна легко змінювати або розширювати умови фільтрації або перевірки, додаючи нові специфікації.

Комбінування умов - Специфікатори можуть об'єднуватися через логічні оператори, що дозволяє створювати складні умови з простих.

Модульність - Логіка перевірки винесена у власні класи, що полегшує їх тестування та обслуговування.

Перевикористовуваність - Специфікації можна використовувати у різних частинах програми, зменшуючи дублювання коду.

Недоліки патерну Специфікатор

Складність при великій кількості специфікацій - Якщо існує багато специфікацій, код може стати важким для розуміння та підтримки.

Перевантаження - У випадках, коли умови досить прості, використання специфікаторів може бути надмірним, додаючи зайві рівні абстракції.

Виконання запитів у базі даних - Якщо специфікації використовуються для динамічних запитів у базу даних, їхня неправильна реалізація може призвести до низької продуктивності.

Комбінування специфікацій

Однією з ключових особливостей патерну Специфікатор є можливість комбінування специфікацій для створення складніших умов. Найпоширеніші типи комбінованих специфікацій:

- І (AND) - Комбінує дві або більше специфікацій таким чином, щоб об'єкт відповідав усім умовам.
- АБО (OR) - Комбінує специфікації так, щоб об'єкт відповідав хоча б одній з умов.
- НЕ (NOT) - Перевіряє, що об'єкт не відповідає певній специфікації.

Уявімо, що у нас є система для управління продуктами. Специфікатор може допомогти створити критерії для фільтрації продуктів, наприклад, вибір продуктів у певній категорії, з певним діапазоном цін чи характеристиками.

Це дозволяє користувачам будувати складні фільтри (наприклад, "Продукти в категорії Електроніка з ціною від 100 до 500 доларів") і зберігати ці фільтри у вигляді специфікацій.

Патерн Специфікатор - це потужний інструмент для побудови гнучких умов перевірки і фільтрації об'єктів. Він забезпечує високу гнучкість та розширюваність коду, дозволяє створювати динамічні запити та умови, які можна легко змінювати, комбінувати або повторно використовувати. Такий підхід особливо корисний у великих системах, де бізнес-логіка постійно змінюється або де потрібна складна фільтрація даних.

Реалізація патерну специфікатор.

Створення інтерфейсу ISpecification<T> з методами:

- Expression<Func<T, object>> OrderBy {get;} – експрешн для сортування
- Expression<Func<T, bool>>? Criteria { get; } - експрешн для фільтрації
- Expression<Func<T, object>> OrderByDescending {get;} – експрешн для сортування по спаданню

Створення класу BaseSpecification<T>(Expression<Func<T, bool>> criteria) : ISpecification<T>

```
20 references
public class BaseSpecification<T>(Expression<Func<T, bool>> criteria) : ISpecification<T>
{
    6 references
    public Expression<Func<T, bool>> Criteria => criteria;

    4 references
    public Expression<Func<T, object>> OrderBy {get; private set;}

    4 references
    public Expression<Func<T, object>> OrderByDescending { get; private set; }

    20 references
    protected void AddOrderBy(Expression<Func<T, object>> orderByExpression)
    {
        OrderBy = orderByExpression;
    }

    10 references
    protected void AddOrderByDescending(Expression<Func<T, object>> orderByDescExpression)
    {
        OrderByDescending = orderByDescExpression;
    }
}
```

Рисунок 2.4 – Базовий клас специфікатору

Створення SpecificationEvaluator<T> where T: BaseModel, для формування query в відповідь

```
1 reference
public class SpecificationEvalutor<T> where T : BaseModel
{
    1 reference
    public static IList<T> GetQuery(Filter filter, IQueryable<T> query, ISpecification<T> spec)
    {
        var result = new List<T>();
        if (spec.Criteria != null)
        {
            query = query.Where(spec.Criteria);
        }

        if (spec.OrderBy != null)
        {
            query = query.OrderBy(spec.OrderBy);
        }

        if (spec.OrderByDescending != null)
        {
            query = query.OrderByDescending(spec.OrderByDescending);
        }
        result = [.. query.Skip((filter.Page - 1) * filter.PageSize).Take(filter.PageSize)];

        return result;

    }
}
```

Рисунок 2.5 – Клас для отримання відфільтрованих даних за специфікатором

Створення конкретних специфаторів:

Було створено ActSpecificator – для фільтрації та пагінації актів списання:

```
public class ActSpecification : BaseSpecification<Act>
{

    public ActSpecification(string? date, string? sort) : base(x => (string.IsNullOrWhiteSpace(date)) || x.Date.Equals(date))
    {
        switch (sort)
        {
            case "dateAsc":
                AddOrderBy(x => x.Date);
                break;
            case "dateDesc":
                AddOrderByDescending(x => x.Date);
                break;
            default:
                AddOrderBy(x => x.Date);
                break;
        }
    }
}
```

Рисунок 2.6 – Специфікаторів для актів списання

В конструктор специфікатора передається дата, та сортування. Формується дерево виразів. Було додано конкретні специфікатори для кожної моделі бази даних, для фільтрації та сортуванням за необхідними полями та з необхідним сортуванням, надалі міститься опис конкретних специфікації які спадкуються від базового специфікатор і реалізують принципи об’єктно-орієнтованого програмування та SOLID

Було створено AppointmentSpecificator – для фільтрації та пагінації призначень.

Було створено FundSpecificator – для фільтрації та пагінації актів списання:

```
using VolunteeredApi.Models;
using VolunteeredApi.Specification;

namespace VolunteeredApi.Data.Specification
{
    1 reference
    public class FundSpecification : BaseSpecification<Fund>
    {
        0 references
        public FundSpecification(string? name, string? sort) : base(x => (string.IsNullOrWhiteSpace(name)) || x.ShortName.Contains(name))
        {
            switch (sort)
            {
                case "nameAsc":
                    AddOrderBy(x => x.ShortName);
                    break;
                case "nameDesc":
                    AddOrderByDescending(x => x.ShortName);
                    break;
                default:
                    AddOrderBy(x => x.ShortName);
                    break;
            }
        }
    }
}
```

Рисунок 2.7 – Специфікація FundSpecification

Було створено InvoiceSpecificator – для фільтрації та пагінації актів списання:

```
namespace VolunteeredApi.Specification
{

    public class InvoiceSpecification : BaseSpecification<Invoice>
    {

        public InvoiceSpecification(string? number, string? sort)
            : base(x => (string.IsNullOrWhiteSpace(number))
            || x.Number.Contains(number))
        {
            switch (sort)
            {
                case "dateAsc":
                    AddOrderBy(x => x.Date);
                    break;
                case "dateDesc":
                    AddOrderByDescending(x => x.Date);
                    break;
                default:
                    AddOrderBy(x => x.Date);
                    break;
            }
        }
    }
}
```

Рисунок 2.8 – Специфікація InvoiceSpecification

Було створено NomenclatureSpecificator – для фільтрації та пагінації актів списання:

```
public class NomenclatureSpecification : BaseSpecification<Nomenclature>
{

    public NomenclatureSpecification(string? invoiceId, string? sort) : base(x => (
    invoiceId == null || x.InvoiceId.ToString() == invoiceId))
    {
        switch (sort)
        {
            case "nameAsc":
                AddOrderBy(x => x.ShortName);
                break;
            case "nameDesc":
                AddOrderByDescending(x => x.ShortName);
                break;
            default:
                AddOrderBy(x => x.ShortName);
                break;
        }
    }
}
```

Рисунок 2.9 - Специфікація NomenclatureSpeficiation

Було створено OrganisationSpecificator – для фільтрації та пагінації актів списання.

```
public class OrganisationSpecification : BaseSpecification<Organisation>
{

    public OrganisationSpecification(string? name, string? sort) : base(x => (string.IsNullOrWhiteSpace(name)) || x.Name.Contains(name))
    {
        switch (sort)
        {
            case "nameAsc":
                AddOrderBy(x => x.Name);
                break;
            case "nameDesc":
                AddOrderByDescending(x => x.Name);
                break;
            default:
                AddOrderBy(x => x.Name);
                break;
        }
    }
}
```

Рисунок 2.10 – Специфікація OrganisationSpecification

Було створено SettingsSpecificator – для фільтрації та пагінації актів списання:

```
public class SettingsSpecification : BaseSpecification<Settings>
{

    public SettingsSpecification(string? name, string? sort) : base(x => (string.IsNullOrWhiteSpace(name)) || x.Name.Contains(name))

    {
        switch (sort)
        {
            case "nameAsc":
                AddOrderBy(x => x.Name);
                break;
            case "nameDesc":
                AddOrderByDescending(x => x.Name);
                break;
            default:
                AddOrderBy(x => x.Name);
                break;
        }

    }
}
```

Рисунок 2.11 – Специфікація SettingsSpecification

Було створено UserSpecificator – для фільтрації та пагінації актів списання:

```
public class UserSpecification : BaseSpecification<User>
{
    public UserSpecification(string? appointmentId, string? sort) : base(x => appointmentId == null || (x.Appointments.Any(a => a.Id.ToString() == appointmentId) && x.UserAppointments.Any(x=> !x.Was
    {
        switch (sort)
        {
            case "nameAsc":
                AddOrderBy(x => x.Name);
                break;
            case "nameDesc":
                AddOrderByDescending(x => x.Name);
                break;
            default:
                AddOrderBy(x => x.Name);
                break;
        }
    }
}
```

Рисунок 2.12 – Специфікація UserSpecification

Отже, було реалізовано конкретний специфікатор для кожної моделі. Кожна специфікація містить в собі поля та правила за якими можна фільтрувати та сортувати об’єкти та масив об’єктів. Було обрано патерн «Специфікатор», так як він є оптимальним рішення для обробки даних.

2.1.3 Реалізація сервісів та інтерфейсів

Було створено базовий інтерфейс IBaseService<T, D> where T : BaseModel

Task<IEnumerable<D>> ReadDataByFilter(Filter filter, ISpecification<T> spec); - метод інтерфейс для отримання даних за фільтром та специфікацією

Paging GetPaging(Filter filter, Expression<Func<T, bool>> spec) – метод інтерфейса для отримання пагінації

Task<IEnumerable<T>> Create(IEnumerable<D> dtos) – метод інтерфейса для створення об’єкта

Task<bool> Update(IEnumerable<D> dtos); - метод інтерфейса для оновлення об’єкта

Task<bool> Delete(IEnumerable<Guid> ids); - метод інтерфейса для видалення об’єкта

Task<D> GetById(Guid id); - метод інтерфейса для отримання елемента за id

Було реалізовано базовий сервіс від якого будуть відспадковані майже всі сервіси додатку.

public class BaseService<T, D>(ApplicationDbContext context, IMapper mapper) : IBaseService<T, D> where T : BaseModel – де T: це модель об’єкта, D – представлення, яке буде передаватись на клієнт;

Містить в собі реалізацію методів інтерфейсу IBaseService<T>:

public async virtual Task<IEnumerable<T>> Create(IEnumerable<D> dtos) – метод для створення масиву об’єктів в базі даних, спочатку відбувається трансформація об’єктів з IEnuberable<T> в IEnumerable<D>, потім кожному елементу масива ініціалізується зміна DateCreated – дата створення та Id – ідентифікатор об’єкта, що є базовими полями в класі BaseModel, метод може бути перевизначений в подальшому;

public async Task<bool> Delete(IEnumerable<Guid> ids) – метод видалення в який передаються масив ідентифікаторів об’єктів для видалення.

Спочатку знаходиться колекція елементів з бази даних за ідентифікатором, якщо вона не порожня, відбувається видалення елементів за допомогою RemoveRange(), оновлюється контекст бази даних;

public virtual async Task<IEnumerable<D>> ReadDataByFilter(Filter filter, ISpecification<T> spec) – метод для отримання колекції елементів за фільтром, в параметри метода передається об’єкт фільтра та специфікація, яка буде застосована. Формується запит в базу даних, метод може бути перевизначеним;

public Paging GetPaging(Filter filter, Expression<Func<T, bool>> spec) – метод для отримання пагінації колекції об’єктів, відбувається підрахунок повернених елементів клієнту, та підрахунок існуючих елементів в бд за фільтром.

В параметри метода передається фільтр та специфікація;

public async virtual Task<bool> Update(IEnumerable<D> dtos) – метод для оновлення колекції елементів в базі даних. В параметри метода передається колекція дто об’єктів, які необхідно оновити. Відбувається трансформація з типу D в тип T, і дані оновлюється, метод може бути перевизначений;

private async Task<IList<T>> ApplySpecification(Filter filter, ISpecification<T> spec) – приватний метод, який повертає відфільтрований масив об’єктів
public async Task<D> GetById(Guid id) – метод для отримання об’єкту за ідентифікатором;

2.1.4 Реалізація контролерів

Далі було створено базовий generic-контролер:

public class BaseController<T, D>(IBaseService<T, D> service, IHttpContextAccessor contextAccessor) where T : BaseModel – контролер містить в собі такі атрибути як:

- [ApiController] – для описання того, що це контролер
- [Authorize] – користуватись базовим контролером можуть тільки авторизовані користувачі
- [Route("api/v1/[controller]")] – для визначення роуту контролера;

В конструкторі контролера передаються через DI такі параметри як, IBaseService<T, D> service – generic сервіс, який буде працювати з класом T та IHttpContextAccessor – для отримання даних про запит;
Контролер містить в собі такі методи, як:

public async Task<ApiResponse<object>> ReadByFilter([FromBody] Filter filter) – ендпоїнт, для отримання даних за фільтром. Містить атрибут [HttpPost("readByFilter")]. Повертає дані в вигляді ApiResponse<IEnuberable<D>>.

Спочатку відбувається визначення контролера на який прийшов запит, потім за допомогою рефлексії визначається тип специфікації яку необхідно створити, якщо специфікацію не знайдено, клієнту повертається помилка «Заданого специфікатора не існує», далі за допомогою класу Activator та методу CreateInstance() створюється об’єкт специфікації в конструктор якого передаються параметри, клас приводиться до типу ISpecification<T>. Далі знайдений специфікатор передається в метод базового сервісу

ReadDataByFilter(), в який передається фільтр та специфікація, отримуються колекція відфільтрованих об’єктів за фільтром та специфікацією, далі отримується пагінація за допомогою методу базового сервіса GetPaging(), повертається ApiResponse, в який ініціалізуються такі поля як Data<IEnumerable<T>>, Error = null, Paging;

public async Task<ApiResponse<IEnumerable<T>>> Create([FromBody] IEnumerable<D> dtos) – ендпоїнт для створення колекції об’єктів, містить атрибут [HttpPost("create")], відбувається створення об’єктів за допомогою методу базового сервісу Create() в який передається колекція dto, повертається відповідь у форматі ApiResponse<IEnumerable<T>>.

public async Task<ApiResponse<bool>> Update([FromBody] IEnumerable<D> dtos) – ендпоїнт для оновлення колекції об’єктів, містить атрибут [HttpPut("update")], відбувається оновлення об’єктів за допомогою методу базового сервісу Update() в який передається колекція dto, повертається відповідь у форматі ApiResponse<bool>.

public async Task<ApiResponse<bool>> Delete([FromBody] IEnumerable<Guid> ids) - ендпоїнт для видалення колекції об’єктів, містить атрибут [HttpDelete("Delete")]відбувається оновлення об’єктів за допомогою методу базового сервісу Delete() в який передається колекція ідентифікаторів, повертається відповідь у форматі ApiResponse<bool>.

Для кожної моделі бази даних було додано сервіс, інтерфейс та контролер, який спадкується від базових.

Далі було додано:

Для опису актів списання - ActController – контролер, ActService – сервіс, IActService – інтерфейс.

Для опису призначень - AppointmentController – контролер, AppointmentService – сервіс, IAppointmnetService – інтерфейс.

Для опису додаткової інформації про користувача - CardController – контролер, CardService – сервіс, ICardService – інтерфейс.

Для опису гуманітарних фондів - FundController – контролер, FundService – сервіс, IFundService – інтерфейс.

Для опису прибуткових накладних - InvoiceController – контролер, InvoiceService – сервіс, IInvoiceService – інтерфейс.

Для опису меню - MenuController – контролер, MenuService – сервіс, IMenuService – інтерфейс.

Для опису номенклатури - NomenclatureController – контролер, NomenclatureService – сервіс, INomenclatureService – інтерфейс.

Для опису організації - OrganisationController – контролер, OrganisationService – сервіс, IOrganisationService – інтерфейс.

Для опису налаштувань організації - SettingsController – контролер, SettingsService – сервіс, ISettingsService – інтерфейс.

Для опису користувача - UserController – контролер, UserService – сервіс, IUserService – інтерфейс.

```
builder.Services.AddScoped<IFundService, FundService>();
builder.Services.AddScoped<IAppointmentService, AppointmentService>();
builder.Services.AddScoped<ICardService, CardService>();
builder.Services.AddScoped<IMenuService, MenuService>();
builder.Services.AddScoped<INomenclatureService, NomenclatureService>();
builder.Services.AddScoped<IOrganisationService, OrganisationService>();
builder.Services.AddScoped<ISettingsService, SettingsService>();
builder.Services.AddScoped<IAuthService, AuthService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IActService, ActService>();
builder.Services.AddScoped<IInvoiceService, InvoiceService>();
builder.Services.AddScoped<IInvoiceProductService, InvoiceProductService>();
builder.Services.AddScoped<IActProductService, ActProductService>();
```

Рисунок 2.13 – Загальний вигляд інтерфейсів та сервісів у Startup

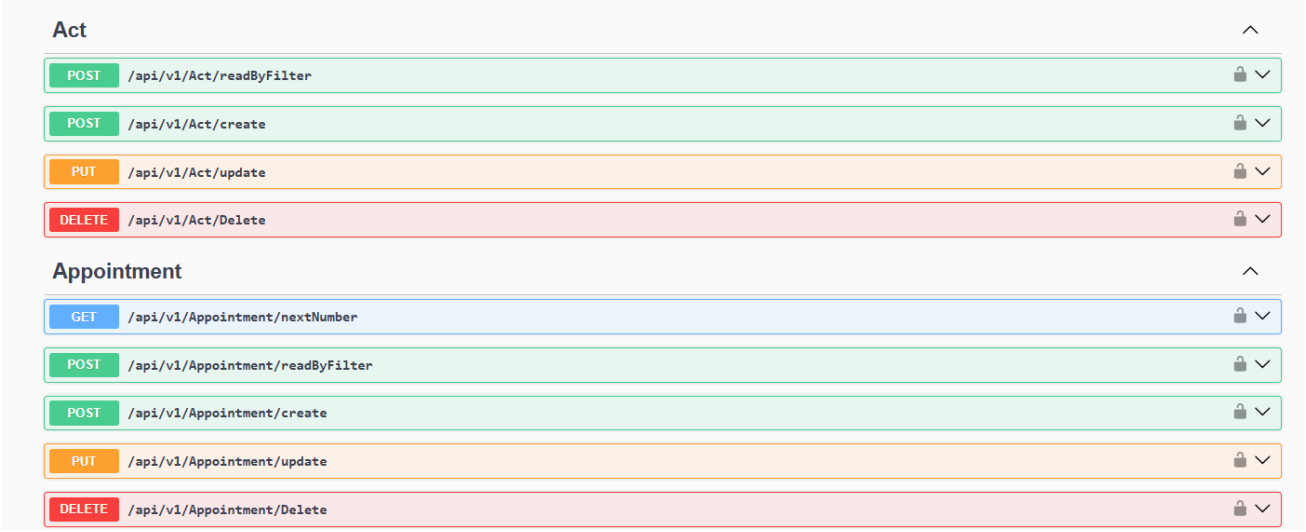

Рисунок 2.14 – Swagger API

2.1.5 Авторизація та автентифікація

Було вирішено зробити авторизацію за допомогою JWT-токену.

JWT (JSON Web Token) - це компактний, URL-безпечний спосіб передачі затверджених даних між сторонами, використовуючи токен у форматі JSON. У контексті авторизації в .NET, JWT використовується для забезпечення безпечної передачі аутентифікаційних даних, що дає змогу клієнту отримати доступ до захищених ресурсів без необхідності зберігати облікові дані. Після успішної аутентифікації сервер генерує JWT, який містить закодовану інформацію про користувача, наприклад, його ідентифікатор та ролі.

Токен складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature). Заголовок визначає тип токену (JWT) та алгоритм шифрування. Корисне навантаження містить закодовані дані про користувача у вигляді клейм (claims) - наприклад, ідентифікатор користувача, ролі або інші параметри, необхідні для авторизації. Підпис генерується на основі заголовка та корисного навантаження за допомогою секретного ключа або приватного ключа у випадку використання асиметричного алгоритму шифрування. Підпис гарантує, що токен не був змінений після створення.

У .NET JWT токени використовуються переважно з middleware для авторизації, яке перевіряє присутність і валідність токену в кожному запиті до захищеного ресурсу.

Впровадження авторизації за допомогою JWT-токену на сервер.

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = configuration["JwtSettings:Issuer"],
            ValidAudience = configuration["JwtSettings:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JwtSettings:SecretKey"]))
        };
    });
```

Рисунок 2.15 – Додання конфігурації JWT-токену у Startup

Було додано сервіс для авторизації AuthService та інтерфейс IAuthService, з методами Login та Register. Конструктор сервісу приймає IConfiguration, ApplicationDbContext. Реалізація методу Register:

public async Task<bool> Register(string email, string password) – метод приймає такі параметри як: пошта та пароль. Перевіряється чи за даною поштою вже не існує користувача, якщо користувача було знайдено, то повертається помилка «Користувач з поштою {email} вже зарєстрований». Якщо користувача не знайдено, формується хеш паролю за допомогою SHA-256.

Створюється базова організація користувача під назвою «Нова організація» в якій буде працювати користувач, потім до цієї організації будуть додаватись фонди, які будуть прив’язуватись користувачі в рамках цієї організації. Створюється об’єкт організації в базі даних, до користувача додається організація, користувач записується в базу даних, оновлюється контекст, якщо дані успішно зберігаються в базі даних, повертається true;
public async Task<string> Login(string email, string password, bool isRememberMe = false) – метод авторизації, приймає в себе параметри: пошта, пароль, запам’ятати мене. Спочатку перевіряється чи існує користувач в системі, якщо користувача не знайдено, повертається помилка «Дана пошта не зареєстрована».

Далі визначається хеш паролю, якщо він співпадає з тим хешом який записаний в базі даних, формується токен, в клейми якого записуються: пошта користувача, організація до якої прив’язаний користувач.
Якщо обраний прапор «Запам’ятати мене», токен формується на місяць, якщо ні, токен формується на день. У відповідь повертається токен.

Далі було створено AuthController, який спадкується від класу ControllerBase, містить атрибут [Route("api/v1/[controller]")]. Конструктор приймає сервіс авторизації AuthService. Контролер містить такі ендпоїнти, як: public async Task<ApiResponse<bool>> Register([FromBody] LoginDto model) – містить атрибут [HttpPost("register")], як параметр приймає LoginDto. Далі

виконується метод сервісу авторизації Register і повертається результат виконання

public async Task<object> Login([FromBody] LoginDto model)) – містить атрибут [HttpPost("login")], як параметр приймає LoginDto. Далі виконується метод сервісу авторизації Login і повертається токен авторизації.

2.1.6 Додання бібліотеки «Automapper»

AutoMapper - це бібліотека об'єктно-об'єктного мапінгу в .NET, яка автоматизує процес перетворення об’єктів між різними типами. Вона допомагає значно скоротити код, що потрібно для копіювання даних з однієї моделі до іншої, особливо якщо моделі мають схожу структуру.

AutoMapper дозволяє налаштувати мапінг між типами за допомогою профілів (Profiles), де описується, як властивості одного об'єкта зіставляються з властивостями іншого. Це корисно в тих випадках, коли моделі бази даних і бізнес-логіки або DTO (Data Transfer Objects) мають різну структуру, але використовуються для одних і тих самих даних. AutoMapper дозволяє налаштовувати правила, які керують мапінгом - наприклад, ігнорувати деякі властивості, налаштовувати спеціальні перетворення для властивостей, використовувати кастомні функції тощо.

Після налаштування, AutoMapper виконує мапінг автоматично, зчитуючи конфігурацію і відображаючи властивості об’єктів. Це особливо корисно для скорочення кількості повторюваного коду, зменшення помилок і підтримання більш чистої та зрозумілої структури коду в проєктах .NET.

Було додано автомапер в конфігурацію сервісів, з конфігурацією MappingProfile : Profile.

MappingProfile містить наступну конфігурацію:

- CreateMap<Fund, FundDto>().ReverseMap();
- CreateMap<Appointment, AppointmentDto>().ReverseMap();
- CreateMap<Card, CardDto>().ReverseMap();

- CreateMap<Nomenclature, NomenclatureDto>().ReverseMap();
- CreateMap<Organisation, OrganisationDto>().ReverseMap();
- CreateMap<Settings, SettingsDto>().ReverseMap();
- CreateMap<User, UserDto>().ReverseMap();
- CreateMap<Invoice, InvoiceDto>().ReverseMap();
- CreateMap<Act, ActDto>().ReverseMap();

2.2 Описання архітектуру фронтенду

2.2.1 Бібліотека стилів «PrimeNG»

PrimeNG - це популярна бібліотека компонентів для побудови інтерфейсів користувача у веб-додатках, що працюють на Angular. Вона забезпечує розробників широким набором готових елементів, які полегшують створення складних та багатофункціональних інтерфейсів. Бібліотека надає компоненти для різноманітних потреб, включаючи таблиці, форми, діалогові вікна, календарі, графіки, меню, індикатори прогресу, сповіщення, випадаючі списки і ще багато іншого. Завдяки PrimeNG розробники можуть легко додавати ці інтерфейсні елементи до своїх додатків без необхідності створювати їх з нуля, що дозволяє значно прискорити процес розробки.

Одна з ключових переваг PrimeNG полягає в тому, що її компоненти вже мають сучасний і привабливий вигляд за замовчуванням, а також є стилізованими за допомогою CSS і SCSS. Це дозволяє дизайнерам та розробникам кастомізувати зовнішній вигляд елементів відповідно до стилю їхнього додатка, без зайвих зусиль. PrimeNG підтримує різноманітні теми, завдяки чому можна легко змінювати стиль усього інтерфейсу, обираючи потрібну кольорову схему чи настроюючи її під конкретний бренд або вимоги.

Крім того, бібліотека адаптивна, що дозволяє компонентам автоматично підлаштовуватися під різні розміри екранів, зокрема мобільні телефони та планшети, забезпечуючи при цьому зручність користування на будь-якому пристрої.

PrimeNG побудована на Angular, що дозволяє їй органічно інтегруватися у структуру Angular-додатків. Вона використовує Angular модулі, директиви та сервіси, завдяки чому всі її компоненти можуть працювати в рамках реактивних форм, підтримувати двостороннє зв’язування даних і з легкістю взаємодіяти з іншими Angular-інструментами. Це робить її привабливим вибором для розробників, які використовують Angular і бажають повністю використовувати його можливості для створення складних та взаємодіючих елементів.

PrimeNG також має добре розроблену документацію, яка включає докладні описи та приклади використання кожного компонента. Це значно полегшує початок роботи з бібліотекою для нових користувачів, а також допомагає швидко знаходити відповіді на питання для тих, хто вже інтегрував її у свої проєкти. Велика та активна спільнота розробників, що використовує PrimeNG, додає ще одну перевагу, оскільки в мережі легко знайти поради, керівництва та готові рішення для поширених задач, що можуть виникнути під час роботи з бібліотекою.

Завдяки готовим та добре оптимізованим компонентам PrimeNG значно полегшує процес розробки сучасних інтерфейсів. Розробники можуть сфокусуватися на бізнес-логіці додатка, тоді як PrimeNG забезпечує всі необхідні елементи інтерфейсу з багатим функціоналом. Такий підхід дозволяє скоротити час розробки та покращити якість інтерфейсу, створюючи враження цілісного та професійного продукту.

2.2.2 Опис загальних налаштувань проекту

Структура тек логічно розбита на directives – в якій містяться директиви, http-services – в якій містяться сервіси який працюють безпосередньо з запитами та httpClient, models – опис моделей, так як використовується typescript. services -–сервіси для бізнес-логіки, shared – директорія в якій зберігаються загальні компоненти, такі як: об’єкт фільтрві, інтерсептори,

меню, нав-бар, компонент таблиці, директорія views – в якій зберігаються компоненти представлення. Архітектура була побудована за принципом використання вже існуючих компонентів та класів які будуть застосовані на всіх сторінки.

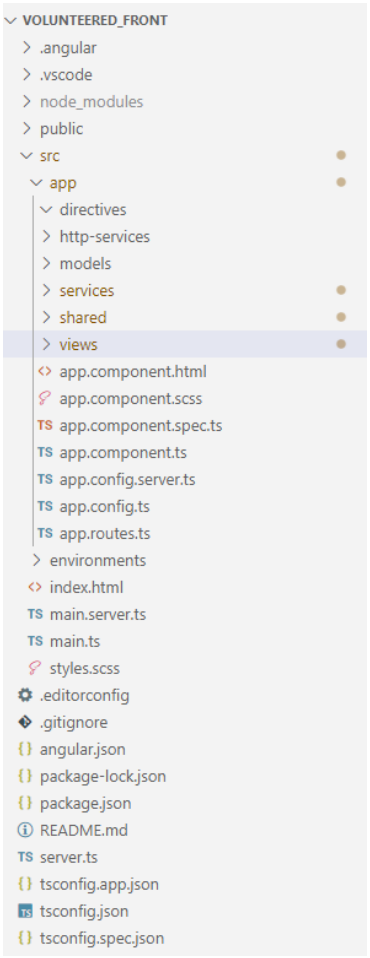

Рисунок 2.16 – Структура файлової системи фронтеду

```
export const appConfig: ApplicationConfig = {
  providers: [provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes), provideClientHydration(), provideAnimations(),
    provideHttpClient(withInterceptorsFromDi(), withFetch()),
     {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }]
};
```

Рисунок 2.17 – Конфігурація app.config.ts

В app.config.ts було додано налаштування та інтерсептори:

- provideZoneChangeDetection({ eventCoalescing: true }) - це провайдер, який налаштовує зміну зон (Zone.js) у додатку. Опція eventCoalescing: true об’єднує події, що надходять у рамках одного кадру браузера, що може зменшити кількість зайвих змін у зоні та підвищити продуктивність.

- provideRouter(routes) - цей провайдер відповідає за маршрутизацію в додатку, використовуючи масив маршрутів routes. Він забезпечує підтримку внутрішньої навігації та управління сторінками всередині додатка, керуючи маршрутами, які користувачі можуть відвідувати.
- provideClientHydration() - провайдер гідратації, який забезпечує підтримку відновлення стану на стороні клієнта. Це корисно для додатків, які використовують серверний рендеринг (SSR), дозволяючи клієнтській частині повторно використовувати дані, згенеровані на сервері, для зменшення часу завантаження.
- provideAnimations() - цей провайдер додає підтримку анімацій Angular. Він дозволяє використовувати різні ефекти анімацій для покращення взаємодії з користувачем, наприклад, переходи між компонентами або елементи з плавними змінами.
- provideHttpClient(withInterceptorsFromDi(), withFetch()) - налаштування HTTP-клієнта, яке додає підтримку перехоплювачів (withInterceptorsFromDi()) для обробки HTTP-запитів, таких як аутентифікація, та підтримку fetch() API (withFetch()). Ці параметри дозволяють інтегруватися з HTTP-запитами, додаючи до них додаткові можливості, наприклад, авторизацію, обробку помилок, і підтримку інтерфейсу fetch() для HTTP-запитів.

```
export const routes: Routes = [
  {
    path: 'app',
    canActivate: [AuthGuard],
    loadComponent: () =>
      import('../app/views/app-view/app-view.component').then(
        (m) => m.AppViewComponent
      ),
    children: [
      { path: '', redirectTo: 'organisation', pathMatch: 'full' },

      {
        path: 'settings',
        loadComponent: () =>
          import('../app/views/settings/settings.component').then(
            (m) => m.SettingsComponent
          ),
      },
      {
```

Рисунок 2.18 – Описання routes для app.routing.ts

Було створено налаштування роутів, компоненти яких будуть підгружатись за принципом lazy-load.

Також був створений клас Auth.Guard, який активується кожен раз, як користувач намагається зайти по шляху app/ - якщо користувач не має токену авторизації Bearer, користувач перенаправляється на сторінку авторизації.

Було створено загальний компонент app-view.component.ts для відображення чіткої структури елементів ДОМ:

```
<div class="layout-container">
    <app-nav-bar></app-nav-bar>
    <div class="content-area">
        <app-menu></app-menu>
        <div style="width: 100%;">
            <router-outlet></router-outlet>
        </div>
    </div>
</div>
```

Рисунок 2.19 – Загальна html структура компонентів

Зверху завжди буде nav-bar, зліва меню, а по центру компонент необхідний для відображення.

Також було додано компонент авторизації login.component.ts та компонент реєстрації registration.component.ts.

2.2.3 Опис структури запитів на API

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private auth: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    let authReq = req.clone();
    if(this.auth.isLoggedIn()){
        const authToken = localStorage.getItem('authUser')!;

        authReq = req.clone({
          headers: req.headers.set('Authorization', `Bearer ${authToken}`)
        });
    }

    return next.handle(authReq);
  }
}
```

Рисунок 2.20 – Перехоплювач для авторизації

Перехоплювач HTTP_INTERCEPTORS – в кожен запит в хедери буде записувати токен для авторизації.

2.2.4 Використання реактивних форм

В проекті використовуються реактивні форми.

Реактивні форми в Angular - це підхід до створення та управління формами, який дозволяє розробникам мати повний контроль над станом та валідацією форм. Цей підхід побудований на основі реактивного програмування, що забезпечує можливість керувати змінами даних у реальному часі та дозволяє чітко визначати, коли і як вони мають оновлюватися. Головними складовими реактивних форм є FormControl, FormGroup і FormArray.

FormControl відповідає за окреме поле вводу у формі, де відстежується його значення та стан (наприклад, чи є поле дійсним, чи було воно змінене тощо). FormGroup об’єднує кілька FormControl елементів у логічний блок, дозволяючи організувати форму у групи полів, що зручно для багаторівневих форм з різними наборами даних. FormArray дозволяє створювати масив контролів для динамічних форм, де кількість полів або груп може змінюватися, наприклад, у випадку додавання списків елементів.

Завдяки реактивним формам можна задавати валідаційні правила, як прості, так і кастомні, та виконувати валідацію на рівні групи або індивідуального поля. Цей підхід робить додатки на Angular більш гнучкими, дозволяючи відстежувати зміни в полях у реальному часі та виконувати додаткові дії на основі цих змін, наприклад, автоматичну перевірку введених даних або активацію/деактивацію елементів інтерфейсу.

Angular підтримує обробку подій, таких як відправка форми або фокусування на полі, що полегшує взаємодію з користувачем та надає можливість розробникам динамічно змінювати вигляд і поведінку форми. Реактивні форми вважаються більш передбачуваними та зручними для

складних додатків, оскільки всі дані щодо стану та валідації форми зберігаються в одному місці, а це спрощує обслуговування і модифікацію коду.

В компоненті авторизації заповнюється форма авторизації та дані передаються на сервер, в залежності від відповіді відбувається подальша поведінка. Якщо отримано відповідь зі статус кодом 200 та токен, токен записується в кеш та користувача перенаправляє на початкову сторінку – сторінку організації. Якщо отримано відповідь зі статус кодом 400, користувачу виводиться помилка, яка прийшла від серверу.

В компоненті реєстрації заповнюється форма, яка складається з трьох полей: пошта, пароль, повторення паролю. Якщо форма ліквідна, запит з цим даними відправляється на сервер за шляхом /api/v1/Auth/register, якщо від сервер отримується відповідь зі статус кодом 200, то користувач перенаправляється на сторінку авторизації, якщо приходить відповідь зі статус кодом 400, то користувачу висвічується відповідне повідомлення.

2.2.5 Побудова базових сервісів

Створення базового абстрактного http-сервісу

- Базовий сервіс містить такі поля як:
apiUrl – шлях до сервера
- ControllerName – назва контролеру до якого буде відбуватись запит
- HtttpClient – клас Http-клієнту

Конструктор приймає такі параметри як назва контролеру, в конструкторі відбувається інжект http-клієнту.

Сервіс має такі методи:

- loadData() – повертає дані отримані з серверу за вказаним фільтром, приймає такі параметри як: page – номер сторінки, pageSize – кількість необхідних елементів на сторінці, criteria – фільтр, sort – поле для сортування.

- Запит get направляється за шляхом`${this.apiUrl}/${this.controllerName}/readByFilter`;
- Create$() – метод для створення елементів. В параметри приймає масив елементів для створення. Запит post надсилається за шляхом `${this.apiUrl}/${this.controllerName}/create`;
- Update$() – метод для оновлення елементів. В параметри приймає масив елементів для створення. Запит put надсилається за шляхом `${this.apiUrl}/${this.controllerName}/update `;
- Delete$() – метод для видалення елементів. В параметри приймає масив елементів для створення. Запит delete надсилається за шляхом `${this.apiUrl}/${this.controllerName}/delete `;

Сервіс провайдиться в root.

Створення базового сервісу таблиці:

Базовий сервіс таблиці містить в собі такі поля як:

- data$: BehaviorSubject<any> = new BehaviorSubject<any>([]) – дані для відображення в таблиці
- count$: BehaviorSubject<any> = new BehaviorSubject<any>(0); - кількість даних отриманих від серверу
- loading$: BehaviorSubject<boolean> = new BehaviorSubject<any>(true); - чи знаходиться сервіс в стані завантаження даних
- page$: BehaviorSubject<any> = new BehaviorSubject<any>(1); номер сторінки в таблиці
- pageItems$: BehaviorSubject<any> = new BehaviorSubject<any>(5); - кількість відображених елементів в таблиці
- filter$: BehaviorSubject<any> = new BehaviorSubject<any>(null); - фільтр для фільтрації даних;
- sort: string = ''; - за яким полем відбувається сортування таблиці;
- fields: Field[] = []; - поля таблиці які будуть відображені в хедерах таблиці;
- service!: ABaseTableHttpService; - сервіс http для взаємодії з сервером.

- dialogService!: DialogService; - сервіс для відображення модалок
- createProfileForm!: FormGroup; - форма створення
- editProfileForm!: FormGroup; - форма редагування;
- messageService!: MessageService; - сервіс для відображення повідомлень користувачу
- confirmationService!: ConfirmationService; - сервіс для відображення підтвердження;

В конструктор сервісу приймаються такі параметри як: fields: Field[] – поля для відображення хедерів таблиці, service: ABaseTableHttpService – сервіс для взаємодії з сервером.

В конструкторі ініціалізується http-сервіс, відбувається завантаження першої сторінки таблиці за допомогою методу loadByFilter(), відбувається ініціалізація хедерів таблиці, провадження messageService, dialogService, confirmationService.

Також базовий сервіс для взаємодії з таблицею містить такі методи як:

- loadByFilter() – метод який повертає дані за фільтром та пагінацією, викликає метод http-сервісу loadData і передає в нього наступні параметри: this.page$.getValue(), this.pageItems$.getValue(), this.filter$.getValue(), this.sort. Якщо приходить успішна відповідь від серверу, тоді дані записуються в поле сервісу data та кількість даних за заданим фільтром в count. Якщо сервер повернув помилку виводиться повідомлення користувачеві.
- Create() – метод для виклику модальну форму з create-компонентом. Приймає в параметри component необхідний для відображення в модальній формі, та назву елемента, який буде створений.
- Викликається за допомогою dialogService метод open, в який передається create компонент, реактивна форма створення, та інші характеристики. Посилання на модальну форму залишається. Створюється підписка на закриття модальної форми, якщо на закриття модальної форми передається результат, тоді спрацьовує метод service.loadData(), який виконує запит на сервер, отримані дані від серверу записуються.

- Update() – метод для виклику модальну форму з edit-компонентом. Приймає в параметри component необхідний для відображення в модальній формі, та назву елемента, який буде створений, також додаткові дані які необхідно передати в модальну форму.

Викликається за допомогою dialogService метод open, в який передається create компонент, реактивна форма створення, та інші характеристики. Посилання на модальну форму залишається. Створюється підписка на закриття модальної форми, якщо на закриття модальної форми передається результат, тоді спрацьовує метод service.loadData(), який виконує запит на сервер, отримані дані від серверу записуються.

- Delete() – метод для виклику видалення обраних елементів, приймає в параметри name: назву обраного елемента, масив ідентифікаторів обраних елементів. Викликається метод confirm сервісу dialogService, в який передаються обрані параметри. Якщо користувач підтверджує видалення елементів, тоді викликається метод delete$ сервісу для взаємодії з сервером в який передається масив ідентифікаторів обраних елементів, якщо отримується успішна відповідь від серверу, тоді виводиться повідомлення про те, що обрані об’єкти успішно видалено, та відбувається перезавантаження елементів сторінки за фільтром.
- Create$() – метод для створення елементів, в параметри приймає масив даних, які необхідно створити, викликає метод create сервісу для взаємодії з сервером
- Update$() – метод для редагування елементів, в параметри приймає масив даних, які необхідно створити, викликає метод create сервісу для взаємодії з сервером.

2.2.6 Побудова компонента таблиці

В компонент таблиці було імпортовано такі модулі, як:

- TableModule - це модуль з PrimeNG, який надає інструменти для роботи з таблицями, включаючи сортування, фільтрацію, пагінацію та редагування рядків. Він дозволяє створювати складні, інтерактивні таблиці з підтримкою кастомізації для відображення даних.
- AsyncPipe - це вбудований Angular Pipe, що дозволяє автоматично підписуватися на об’єкти Observable або Promise та отримувати їхнє значення асинхронно. Він автоматично керує відпискою, що спрощує управління асинхронними даними.
- JsonPipe - це Angular Pipe, який перетворює об’єкти або масиви у формат JSON для легкого виведення в шаблонах. Він використовується для швидкого перегляду даних під час розробки або відображення структури об'єктів.
- PaginatorModule - це PrimeNG модуль, який забезпечує зручні елементи пагінації для розбиття даних на сторінки. Він підтримує налаштування кількості елементів на сторінку, навігаційні кнопки та інформацію про сторінки.
- CommonModule - це основний Angular модуль, який містить часто використовувані директиви, такі як ngIf, ngFor, та інші. Він надає базові інструменти, необхідні для побудови шаблонів і роботи з умовним та циклічним рендерингом.

Компонент таблиці містить наступні поля:

- data$: BehaviorSubject<any> = new BehaviorSubject<any>([]) – дані для відображення в таблиці
- count$: BehaviorSubject<any> = new BehaviorSubject<any>(0); - кількість даних отриманих від серверу
- loading$: BehaviorSubject<boolean> = new BehaviorSubject<any>(true); - чи знаходиться сервіс в стані завантаження даних
- page$: BehaviorSubject<any> = new BehaviorSubject<any>(1); номер сторінки в таблиці
- pageItems$: BehaviorSubject<any> = new BehaviorSubject<any>(5); - кількість відображених елементів в таблиці

- service: ABaseTableService - сервіс для роботи з таблицею
- toolbarTemplate: TemplateRef<any> - темплейт тулбару таблиці;
- datakey: string = «id» - назва поля за яким буде працювати селект даних таблиці
- usecheckbox: Boolean – чи використовувати чекбокси для таблиці
- selectionMode!: "single" | "multiple" – чи використовувати множинний селект елементів таблиці
- @Output() selectedData: EventEmitter<any> = new EventEmitter<any>() – передача обраних даних батьківському компоненту;
- При ініціалізації компоненту в методі ngOnInit() властивості data$, count$, loading$, page$, pageItems$, fields посилаються на відповідні поля табличного сервісу, також ініціалізується спосіб селекту елемнтів таблиці: множинний або по одному.

Табличний компонент містить в собі такі методи, як:

- onPageChange(event: any) – метод, який викликається при зміні сторінки, оновлюються властивості pageItems та page, викликається метод табличного сервісу loadDataByFilter(), для оновлення даних за пагінацією.
- OnRowSelect(event:any) – емітяться дані в батьківський компонент
- OnSort(event: any) – визначається порядок сортування та поле за яким сортування відбудеться, змінюється поле табличного сервісу sort, викликається метод loadByFilter для оновлення даних.

```
</div>
<p-table #table class="custom-table" [value]="service.data$ | async" [paginator]="true"
    [tableStyle]="{ 'min-width': '50rem' }" [first]="((service.page$ | async) - 1) * (service.pageItems$ | async)"
    [rows]="service.pageItems$ | async" [totalRecords]="service.count$ | async"
    currentPageReportTemplate="{first} до {last} з {totalRecords} рядків" [showCurrentPageReport]="true" [lazy]="true"
    [dataKey]="datakey" (onPage)="onPageChange($event)" [scrollable]="true" scrollHeight="400px"
    (onRowSelect)="onRowSelect($event)" [selectionMode]="usecheckbox ? null : 'single'" editMode="cell"
    (onSort)="OnSort($event)" emptyMessage="Дані не додано" [rowsPerPageOptions]="[5, 10, 25]">

    <ng-template pTemplate="header">
```

Рисунок 2.21 – Html-структура компоненту таблиці

Цей фрагмент коду представляє компонент таблиці p-table із бібліотеки PrimeNG, яка використовується для відображення даних з підтримкою пагінації, сортування та вибору рядків. Загальний контейнер

містить внутрішній компонент <ng-container>, що підключає шаблон toolbarTemplate з використанням Angular директиви [ngTemplateOutlet]. Цей шаблон може містити інструменти або дії, які потрібні в панелі управління таблицею, такі як кнопки для створення, оновлення чи видалення записів.
Таблиця p-table має клас custom-table, який дозволяє застосовувати стилі через SCSS. Дані для таблиці підключаються за допомогою service.data$, асинхронного потоку Observable, підписка на який здійснюється через Angular pipe async.

Таблиця використовує paginator, щоб реалізувати функціонал сторінок, де кількість записів на кожній сторінці визначається потоком service.pageItems$, також підключеним через async. Для коректного відображення першого запису на поточній сторінці використовується параметр [first], значення якого обчислюється як зміщення від початку: поточна сторінка (service.page$) мінус один, помножена на кількість елементів на сторінці (service.pageItems$). Загальна кількість записів у таблиці задається параметром [totalRecords], який також отримує значення через асинхронний потік service.count$.

Атрибут currentPageReportTemplate визначає шаблон повідомлення, яке відображає інформацію про поточну сторінку, наприклад, діапазон записів {first} до {last} з загальною кількістю {totalRecords}. Ця інформація відображається під таблицею завдяки параметру [showCurrentPageReport]="true". Функція завантаження даних таблиці реалізована в режимі лінивого завантаження ([lazy]="true"), що дозволяє завантажувати дані поступово по мірі необхідності, наприклад, при зміні сторінки.

Ключ dataKey встановлює унікальний ідентифікатор для кожного рядка, що допомагає керувати вибором даних. Подія (onPage) спрацьовує при зміні сторінки, що дозволяє виконувати необхідні дії, такі як завантаження даних для нової сторінки, через метод onPageChange($event). Таблиця має параметр [scrollable]="true", що дозволяє прокручування даних у таблиці з фіксованою

висотою scrollHeight="400px". При виборі рядка спрацьовує подія (onRowSelect), яка викликає функцію onRowSelect($event), обробляючи обране значення, а selectionMode контролює режим вибору (наприклад, одиночний чи множинний), визначений змінною selectionMode.

Шаблон header для таблиці містить заголовок, де для кожного поля масиву fields створюється відповідна колонка. Якщо поле дозволяє сортування (field.canSort), тоді колонка включає компонент <p-sortIcon field="code">, який активує сортування за відповідним полем. Для колонок, де сортування не потрібне, просто виводиться назва поля. У шаблоні body кожен рядок даних відображається через let-data, який представляє окремий об'єкт із масиву даних. Якщо змінна usecheckbox встановлена в true, для кожного рядка додається чекбокс <p-tableCheckbox [value]="data">, який дозволяє користувачеві обирати один або кілька рядків. Далі для кожного поля з fields формується комірка <td>, яка виводить значення відповідного поля data[field.value], що дозволяє гнучко відображати дані у таблиці відповідно до вмісту масиву fields.

Компонент nav-bar містить меню та інжектить сервіс AuthService, використовуються метод logout().

Компонент menu містить в собі масив меню, та роутер. При переходах по меню відбувається переходи за обраним шляхом.

Модель Field містить в собі такі властивості, як name: string, value: string, canSort: Boolean.

2.2.7 Реалізація сервісів для взаємодії з API

Було створено такі http-service, які спадкуються від ABaseTableHttpService:

- Act-http.service.ts – сервіс для взаємодії з сервером за актами списання;
- Appointment-http.service.ts - сервіс для взаємодії з сервером за назначеннями роздачі гуманітарної допомоги
- Auth-http.service.ts - сервіс для взаємодії з сервером для авторизаіції;

- Fund-http.service.ts - сервіс для взаємодії з сервером за фондами;
- Invoice-http.service.ts - сервіс для взаємодії з сервером за прибутковими накладними;
- Nomenclature-http.service.ts - сервіс для взаємодії з сервером за номенклатурою;
- Organization-http.service.ts - сервіс для взаємодії з сервером за організацією;
- Settings-http.service.ts - сервіс для взаємодії з сервером за налаштуваннями;
- User-http.service.ts - сервіс для взаємодії з сервером за користувачами;

Було створено базовий сервіс для роботи з модальними формами:

Цей код визначає абстрактний сервіс AModalBaseService, який є базовим сервісом для роботи з модальними вікнами в Angular із використанням PrimeNG компонентів. Він містить логіку для створення та оновлення об'єктів на основі даних, що вводяться в форму, а також забезпечує обробку повідомлень для користувача про успішність чи помилки операцій. Сервіс позначений декоратором @Injectable(), що означає, що він може бути використаний як інжектований сервіс в інших компонентах чи сервісах. У класі AModalBaseService використовується кілька основних залежностей, які інжектуються за допомогою функції inject, на відміну від традиційного інжектування через конструктор.

Це зокрема DialogService, що забезпечує роботу з діалоговими вікнами, DynamicDialogRef, який представляє посилання на активне діалогове вікно, DynamicDialogConfig для отримання конфігураційних даних модального вікна, а також MessageService, що дозволяє надсилати користувацькі повідомлення в інтерфейсі. Залежності інжектуються всередині конструктора, і значення властивостей класу ініціалізуються на основі інжекцій.

Діалогове вікно отримує дані через DynamicDialogConfig, зокрема profileForm та service. profileForm - це форма Angular FormGroup, що містить поля даних, необхідних для створення або оновлення об'єкта. service є екземпляром

ABaseTableService, який забезпечує методи для створення та оновлення об'єктів (наприклад, create$ і update$). Ці методи повертають Observables, які підписуються для виконання відповідних операцій.

Метод close дозволяє закривати модальне вікно, викликаючи this.ref.close().

Метод create відповідає за створення нового об'єкта на основі даних, що вводяться у формі profileForm. Спочатку перевіряється, чи є форма валідною. Якщо форма дійсна, викликається метод this.service.create$, який передає значення полів форми через getRawValue() у вигляді масиву. Після виклику create$ відбувається підписка на Observable, що повертається методом, щоб виконати запит на створення нового об'єкта на сервері. При успішному виконанні серверного запиту метод закриває діалогове вікно та додає повідомлення користувачеві про успішне створення об'єкта. У разі помилки виводиться відповідне повідомлення із текстом, отриманим із помилки сервера. Якщо форма не є валідною, користувач отримує попередження через messageService про те, що вона не заповнена належним чином.

Метод update має подібну логіку, але призначений для оновлення існуючого об'єкта. Якщо форма є валідною, метод this.service.update$ викликається з даними форми. Як і у випадку з create, підписка на Observable дозволяє обробляти успішне виконання запиту або помилку. При успішному оновленні відбувається закриття діалогового вікна та показ повідомлення про успішне завершення операції, а у випадку помилки відображається повідомлення з інформацією про помилку, що була отримана з відповіді сервера.

Таким чином, AModalBaseService є основою для побудови модальних діалогів, що забезпечують створення або редагування даних, інтегрованих з PrimeNG компонентами для відображення повідомлень і роботи з діалоговими вікнами. Він також використовує реактивний підхід для обробки серверних відповідей, забезпечуючи асинхронність і обробку помилок у взаємодії з користувачем.

Було створено також такі сервіси, які спадкуються від базового табличного сервісу:

ActService, який містить такі поля, як:

- Дата – date,
- Номер – number

AppointmentService – Сервіс для роботи із записами про призначення, що також наслідується від ABaseTableService та використовує AppointmentHttpService для HTTP-запитів. Таблиця містить поля: Дата (date) з можливістю сортування, Фонд (fundName) та Користувач (userName). Містить додатковий метод getNextNumber() для отримання наступного номеру призначення.

Ось детальний опис для сервісу AuthService:

AuthService – Сервіс, який забезпечує функціонал аутентифікації в Angular-додатку та позначений декоратором @Injectable з опцією { providedIn: 'root' }, що робить його доступним у всьому додатку. AuthService працює з AuthHttpService для виконання HTTP-запитів та Router для навігації між сторінками.

- Метод signup(data: any) викликає відповідний метод HTTP-сервісу AuthHttpService для реєстрації нового користувача. Метод login(data: any) відправляє дані авторизації (логін і пароль) на сервер, використовуючи AuthHttpService. Обидва методи повертають Observables, які можна підписати для обробки відповідей від сервера.
- Метод logout() видаляє дані про користувача з локального сховища (localStorage), що здійснює вихід з системи, та перенаправляє користувача на сторінку login, використовуючи метод navigateByUrl з Router. Метод isLoggedIn() перевіряє, чи є користувач авторизованим, шляхом перевірки наявності запису authUser у локальному сховищі.

FundService – це сервіс для роботи з даними про фонди, який наслідує ABaseTableService і використовує FundHttpService для обробки HTTP-запитів. Він визначає конфігурацію для таблиці з такими полями:

- Назва (shortName) – поле, що містить назву фонду і підтримує сортування (canSort: true).
- Одиниця виміру (measure) – поле, яке показує одиницю виміру фонду, не підтримує сортування.
- Кількість (number) – поле, що показує кількість одиниць фонду, не підтримує сортування.

Цей сервіс передає зазначені параметри в конструктор ABaseTableService разом із httpService, щоб налаштувати базові функції таблиці для управління фондами в додатку.

InvoiceService – це сервіс для роботи з накладними, який наслідує від базового класу ABaseTableService і використовує InvoiceHttpService для здійснення HTTP-запитів. Сервіс налаштовує таблицю з двома полями:

- Дата (date) – поле для відображення дати рахунку, яке підтримує сортування (властивість canSort: true).
- Номер (number) – поле для відображення номера рахунку, без можливості сортування.

Конструктор сервісу передає масив полів таблиці та інстанс InvoiceHttpService в конструктор базового класу ABaseTableService, що дозволяє сервісу успадковувати базову логіку для роботи з таблицею та HTTP-запитами.

NomenclatureService – це сервіс для роботи з номенклатурними даними, який наслідує від базового класу ABaseTableService і використовує NomenclatureHttpService для виконання HTTP-запитів. Сервіс конфігурує таблицю для відображення таких полів:

- Назва (shortName) – поле для відображення короткої назви номенклатури, з можливістю сортування (властивість canSort: true).
- Одиниця виміру (measure) – поле для відображення одиниці виміру номенклатури, без можливості сортування.
- Кількість (number) – поле для відображення кількості одиниць номенклатури, також без підтримки сортування.

Конструктор цього сервісу передає масив полів та інстанс NomenclatureHttpService в конструктор базового класу ABaseTableService, що дозволяє сервісу успадковувати основні методи для роботи з таблицями та виконання HTTP-запитів для отримання, створення, редагування або видалення записів.

OrganisationService – це сервіс для роботи з організаціями, який наслідує від базового класу ABaseTableService і використовує OrganisationHttpService для здійснення HTTP-запитів. Сервіс конфігурує таблицю, яка має тільки одне поле:

Назва (name) – поле для відображення назви організації, яке підтримує сортування (властивість canSort: true).

Конструктор цього сервісу передає масив полів та реалізацію OrganisationHttpService в конструктор базового класу ABaseTableService, що дозволяє використовувати базову логіку для роботи з таблицею, а також взаємодіяти з HTTP-сервісами для отримання, оновлення або видалення записів про організації.

SettingsService – це сервіс для роботи з налаштуваннями в додатку, який наслідує від базового класу ABaseTableService і використовує SettingsHttpService для виконання HTTP-запитів. Сервіс налаштовує таблицю з двома полями:

- Назва (name) – поле для відображення назви налаштування, яке підтримує сортування (властивість canSort: true).
- Значення (value) – поле для відображення значення налаштування, без можливості сортування.

Конструктор сервісу передає масив полів таблиці та інстанс SettingsHttpService в конструктор базового класу ABaseTableService, що дозволяє сервісу успадковувати функціональність для роботи з таблицею та виконання HTTP-запитів для отримання, оновлення або видалення налаштувань.

UserService – це сервіс для роботи з даними користувачів, який наслідує від базового класу ABaseTableService і використовує UserHttpService для виконання HTTP-запитів. Сервіс налаштовує таблицю з п’ятьма полями:

- Пошта (email) – поле для відображення електронної пошти користувача.
- ПІБ (name) – поле для відображення повного імені користувача, яке підтримує сортування (властивість canSort: true).
- Адреса (address) – поле для відображення адреси користувача.
- Номер (cardNumber) – поле для відображення номера картки користувача.
- Реферальне посилання (referralLink) – поле для відображення реферального посилання користувача.

Конструктор сервісу передає масив полів таблиці та інстанс UserHttpService в конструктор базового класу ABaseTableService, що дозволяє сервіту успадковувати логіку для роботи з таблицею та взаємодії з HTTP-сервісами для отримання, створення, оновлення або видалення записів користувачів.

Для кожного компонента було створено, компонент створення, компонент редагування та відповідні форми.

2.2.8 Реалізація компонентів сторінки

Було створено компонент для списання актів:

ActCreateComponent – це компонент для створення актів, який наслідує від абстрактного сервісу AModalBaseService і використовує інші сервіси для управління номенклатурою та виконанням створення акта. Він має кілька важливих властивостей та методів, що забезпечують функціональність компонента.

Компонент імпортує різні модулі, такі як ReactiveFormsModule для роботи з реактивними формами, TableComponent для відображення таблиць, StepperModule для реалізації кроків, ButtonModule для використання кнопок,

та CalendarModule для додавання календаря в інтерфейс. Він також використовує компонент NomenclatureComponent для роботи з номенклатурними даними, а NomenclatureService надається як провайдер для доступу до сервісу номенклатури.

Компонент визначає кілька важливих змінних, включаючи:

- selectedProduct – змінна для зберігання вибраного продукту.
- id – ідентифікатор створеного акта, який буде отриманий після успішного створення акта.
- stepper – змінна, яка посилається на елемент Stepper для управління переходами між кроками форми.
- productToAdd – змінна для додавання продукту.
- productsToCreate – масив продуктів, які потрібно додати.

Метод createAct викликається при створенні нового акта. Він робить HTTP-запит через this.service.create$ з даними форми. Після успішного створення акта, він отримує ID створеного акта та викликає методи сервісу номенклатури для фільтрації та завантаження даних, оновлюючи список номенклатури на основі ID акта. Потім він переходить на наступний крок у формі за допомогою методу stepper.nextCallback, оновлюючи компонент, викликаючи cdr.detectChanges() для правильного оновлення UI.

Він також обробляє помилки, якщо вони виникають під час створення акта, і виводить їх у консоль.

Для оновлення даних акту списання:

ActEditComponent - це компонент для редагування існуючого акта, який наслідує від абстрактного класу AModalBaseService. Він надає функціональність для редагування акта та інтеграції з сервісами, такими як NomenclatureService для управління номенклатурними даними, і InvoiceService для роботи з рахунками.

Компонент використовує кілька модулів з бібліотеки PrimeNG, таких як ButtonModule для кнопок, CalendarModule для календаря, та StepperModule для кроків в інтерфейсі користувача. Також він імпортує ReactiveFormsModule для

роботи з реактивними формами та компонент TableComponent для відображення таблиць.

У конструкторі компонента ініціалізується сервіс NomenclatureService для завантаження та фільтрації номенклатури, використовуючи значення з форми profileForm. Через метод next() сервісу передається значення ідентифікатора акта (який зберігається в полі форми з назвою id), що дозволяє фільтрувати номенклатуру відповідно до цього ідентифікатора. Потім викликається метод loadByFilter(), щоб завантажити відповідні дані на основі фільтру.

Метод ngOnInit є частиною життєвого циклу компонента і зараз не містить специфічної логіки, але його можна використовувати для ініціалізації додаткових даних або налаштувань.

Завдяки цьому компоненту користувач може редагувати існуючий акт, а також переглядати та змінювати асоційовані з ним номенклатурні дані.

Були створені також форми:

Цей код визначає дві функції, getCreateProfileForm та getEditProfileForm, які створюють реактивні форми в Angular для створення та редагування профілю.

Функція getCreateProfileForm створює форму для створення нового профілю. Вона повертає екземпляр FormGroup, який складається з трьох полів:

- date: Це поле для введення дати, яке обов'язкове для заповнення. Для цього використовується FormControl з валідатором Validators.required, що вимагає, щоб поле не було порожнім.
- number: Поле для введення номера профілю. Воно також обов'язкове, тому застосовано валідатор Validators.required.
- nomenclatures: Поле для введення списку номенклатур, яке також є обов'язковим для заповнення. Тут використовується FormControl з початковим значенням порожнього масиву і валідатором Validators.required.
- Функція getEditProfileForm створює форму для редагування існуючого профілю. Вона повертає також FormGroup, але з деякими відмінностями:

- id: Це поле для збереження ідентифікатора профілю, яке не є обов'язковим, оскільки воно необхідне лише для редагування вже існуючого профілю.
- date та number: Поля для введення дати та номера профілю, як і у першій формі, але в цій формі вони можуть бути змінені (знову ж таки, обов'язкове заповнення).
- nomenclatures: Це поле для введення списку номенклатур, але в цьому випадку воно не є обов'язковим, що дозволяє редагувати існуючі дані без необхідності змінювати цей список.

Обидві функції використовують Angular FormControl для створення контролів форми, і FormGroup для групування цих контролів в єдиний об'єкт. Валідація на рівні полів забезпечується через Validators.required для обов'язкових полів, що гарантує, що користувач не зможе залишити їх порожніми при відправці форми.

Інші компоненти були за схожим принципом, тільки з іншими полями.

2.3 Описання архітектури телеграм бота

2.3.1 Створення та налаштування проекту

Було створено проект Asp.net core .net 8.0, який посилається на вищеописані бібліотеки класів Data. Телеграм бот буде використовувати однакову базу даних, сервіси та моделі, що і використовував API.

Налаштування Startup:

var builder = WebApplication.CreateBuilder(args) - створює новий об'єкт builder, який використовується для налаштування сервісів, середовища виконання та інших аспектів веб-додатку.

Реєстрація служб у контейнері залежностей:

builder.Services.AddControllers() - додає необхідні сервіси для обробки HTTP-запитів через контролери (реєстрація служб для API).

builder.Services.AddDbContext<ApplicationDbContext>(options => {...}) - реєструє службу контексту бази даних для використання Entity Framework Core. Використовується SQL Server як база даних, а рядок підключення береться з конфігурації додатку (зчитується з налаштувань конфігураційного файлу).

options.UseSqlServer(connectionString, b => b.MigrationsAssembly("VolunteeredApi")) - вказує, що міграції будуть виконуватися з відповідної збірки (в даному випадку, збірка "VolunteeredApi").

Налаштування AutoMapper:

builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()) - додає AutoMapper для мапінгу об'єктів між різними типами даних (DTO). Використовується для автоматичного відображення між моделями та DTO в додатку. Усі асамблеї, що завантажуються в поточному домені, перевіряються на наявність профілів для мапінгу.

Реєстрація Telegram бота та сервісу користувача:

builder.Services.AddSingleton<TelegramBot>() - реєструє бота як синглтон, що означає, що один екземпляр цього бота буде використовуватися протягом усього життєвого циклу програми.

builder.Services.AddScoped<IUserService, UserService>() - реєструє сервіс користувачів, де IUserService - інтерфейс, а UserService - реалізація цього інтерфейсу, як Scoped, тобто кожен запит отримає новий екземпляр цього сервісу.

Побудова програми:

var app = builder.Build() - створює саму програму на основі попередньо налаштованих служб і конфігурацій.

Ініціалізація Telegram бота:

app.Services.GetRequiredService<TelegramBot>().GetBot().Wait() - отримує екземпляр Telegram бота з контейнера залежностей і викликає його метод GetBot(), щоб ініціалізувати та запустити бота. Використовується метод Wait(), щоб заблокувати виконання на цьому етапі, поки бот не буде готовий.

Налаштування Middleware:

- app.UseAuthorization() - додає послугу для авторизації в додатку, хоча в коді конкретно сама авторизація не конфігурується.
- app.MapControllers() - налаштовує маршрути для контролерів, що дозволяє API обробляти HTTP-запити.

Запуск додатку:

app.Run() - запускає додаток, починаючи обробку вхідних запитів.

Цей код налаштовує веб-додаток, який працює з базою даних через Entity Framework, взаємодіє з Telegram-ботом і забезпечує відповідні сервіси для обробки запитів та логіки користувачів.

Цей код представляє собою контролер для Telegram бота в додатку, який обробляє повідомлення та команду з боку користувача. Ось детальний опис кожної частини коду::

using Telegram.Bot - імпортується бібліотека для роботи з Telegram API.

using VolunteeredApi.Services.Interfaces - імпортуються сервіси для роботи з користувачами.

using Newtonsoft.Json - для десеріалізації оновлень, отриманих від Telegram.

using System.Text - для побудови текстових повідомлень.

В конструкторі контролера відбувається ін'єкція залежностей: IServiceProvider для отримання бот-клієнта через TelegramBot та IUserService для роботи з користувачами.

2.3.2 Ініціалізація бота

private TelegramBotClient _botClient - створюється екземпляр TelegramBotClient за допомогою сервісу TelegramBot, який повертає налаштований бот. Бот використовується для відправки повідомлень в чат.

3. Обробка HTTP запиту:

Контролер містить один метод Update, який обробляє запити POST на URL api/message/update. Він приймає об'єкт update у вигляді JSON і десеріалізує його в об'єкт типу Update з Telegram API

[FromBody] object update - отримує оновлення (update) у форматі JSON з тіла запиту.

Update? upd = JsonConvert.DeserializeObject<Update>(update.ToString()) - десеріалізує отримане оновлення у тип Update.

Якщо повідомлення не містить чату або callback-запиту (що зазвичай означає, що це не повідомлення або callback), метод відразу повертає успішний статус 200 (OK).

Якщо текст повідомлення містить команду /start, бот виконує наступне:

Отримує параметр після /start (це номер картки користувача).

Викликає метод userService.GetUserByCardNumber(parameter), щоб знайти користувача за номером картки.

Якщо користувач не знайдений, бот відправляє повідомлення з проханням повторно ввести номер картки.

Якщо користувач знайдений, його Telegram ID зберігається в базі даних, а також бот вітає користувача та повідомляє про успішну реєстрацію в гуманітарному фонді.

Якщо повідомлення містить команду /getMyAppointments, бот виконує наступне:

Отримує користувача за ChatId за допомогою userService.GetUserByChatId(upd.Message.Chat.Id).

Якщо користувач знайдений, бот отримує список його роздач за допомогою userService.GetUserAppointments(user).

Для кожного запису в роздачах будується список і відправляється користувачу у вигляді текстового повідомлення.

Наприкінці метод повертає Ok(), що вказує на успішне оброблення запиту.

Цей контролер відповідає за обробку оновлень (повідомлень і команд) від Telegram бота. Ключові функції включають:

Реєстрація користувачів за номером картки через команду /start.

Виведення списку призначених роздач для користувачів через команду /getMyAppointments.

Взаємодія з базою даних через IUserService для збереження та отримання даних користувачів.

Цей код дозволяє додатку взаємодіяти з користувачами через Telegram, приймати команди та працювати з даними користувачів для надання їм актуальної інформації.

2.3.3 Архітектура бота

Обробка повідомлень, які надсилає користувач, та обробка повідомлень реалізовано за допомогою шаблону програмування «Команда». Наш клас містить багато класів, які спадкуються від базового – BaseCommand. Клас BaseCommand містить в собі загальний опис команди, а саме такі поля як Name та метод ExecuteAsync – який призначений для обробку запиту клієнта. В контролері відбувається пошук потрібного сервісу (команди) за полем Name та контекстом виконання останньої команди. Кожна команда виконання записується в базу даних для відстежування на якому етапі меню знаходиться користувач. Далі виконується реалізація конкретного сервісу або «команди» - методу ExecuteAsync().

РОЗДІЛ 3

ПРЕДСТАВЛЕННЯ ПРОГРАМНОГО РІШЕННЯ

3.1. Сторінка авторизації та реєстрації

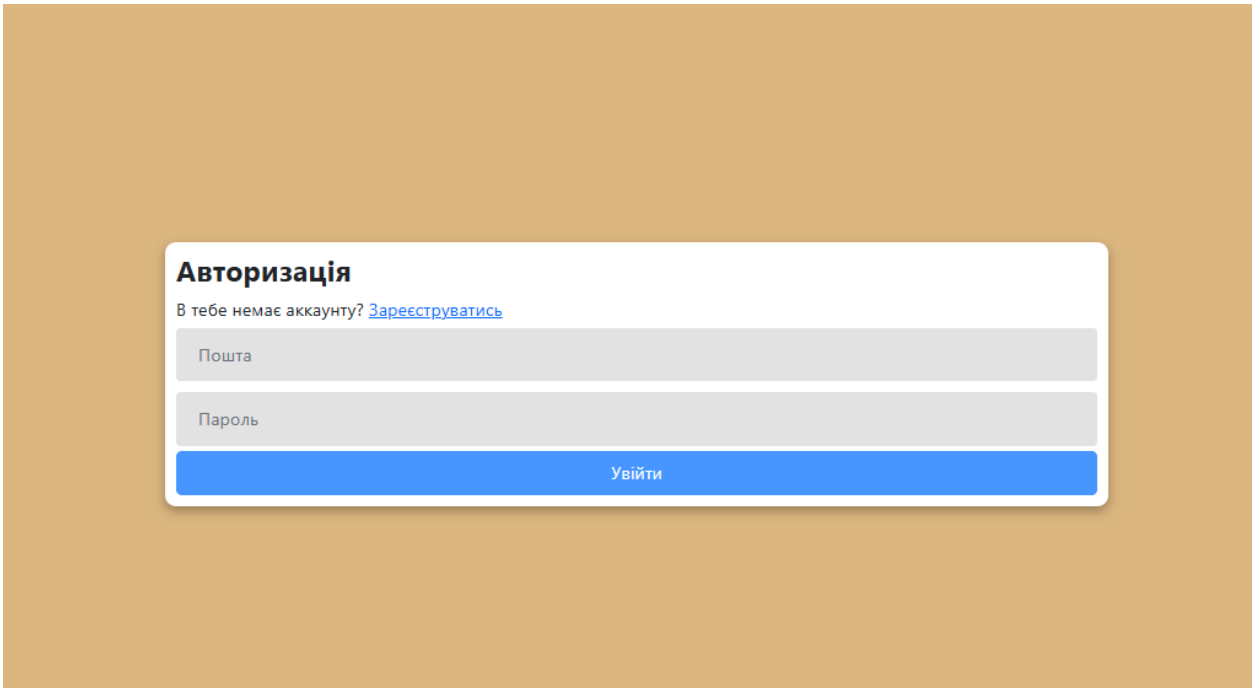

Рисунок 3.1 – Сторінка авторизації

Користувач має можливість авторизувати на сайті.

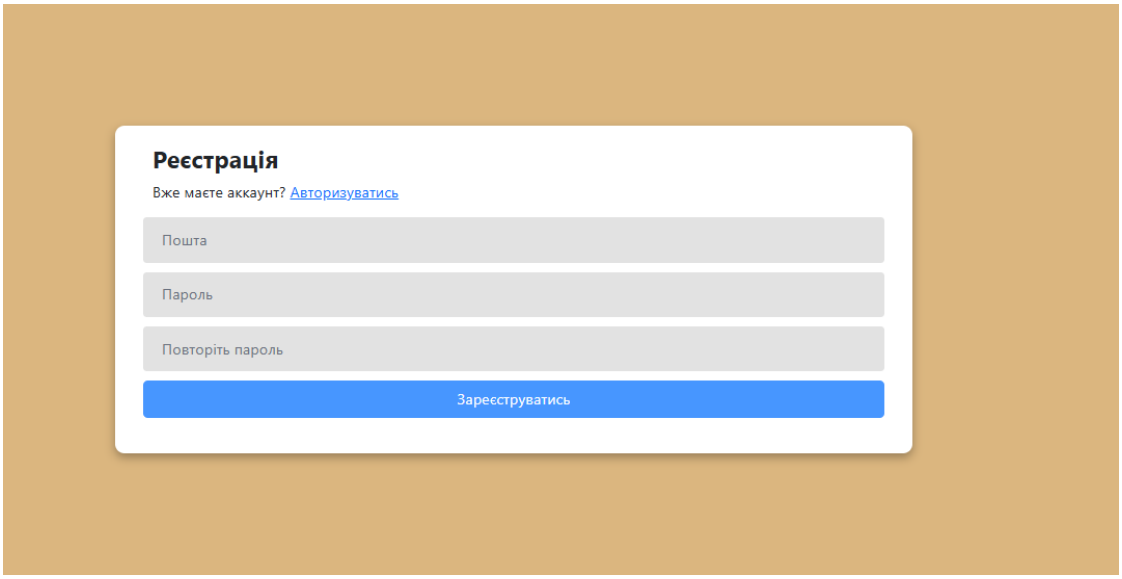

Рисунок 3.2 – Сторінка реєстрації

Користувач має можливість зареєструватись на сайті. При реєстрації також створюється організація в яку попадає користувач.

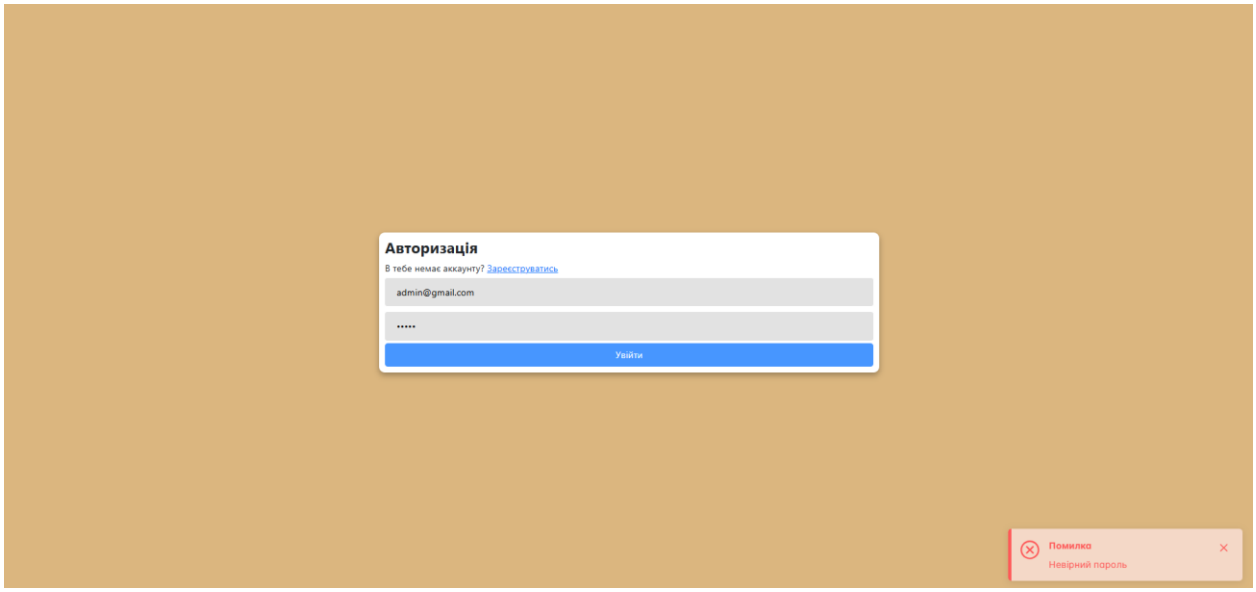

Рисунок 3.3 – Приклад повідомлення про помилку

Також на форму авторизації та реєстрації включено валідацію. Якщо поля порожні або пошта не відповідає заданому формату регулярної функції, то висвічується відповідне повідомлення.

3.2 Робота з веб-панеллю

Веб-панель складається з 8-ми пунктів меню які описані на беке і отримуються фронтом, легко адмініструються за допомогою бази даних, користувачу доступні такі пункти меню, як:

- Організація – Інформація про організацію – для керування інформацією про поточну організацію;
- Організація – Налаштування – для керування налаштуваннями організації;
- Організація – Аналітика – перегляд статистики;
- Користувачі – Записи - адміністрування видач гуманітарної допомоги;
- Користувачі – Список користувачів

- Фонд – Керування фондами
- Склад – Номенклатура
- Склад – Прибуткові накладні
- Склад – Акти списання

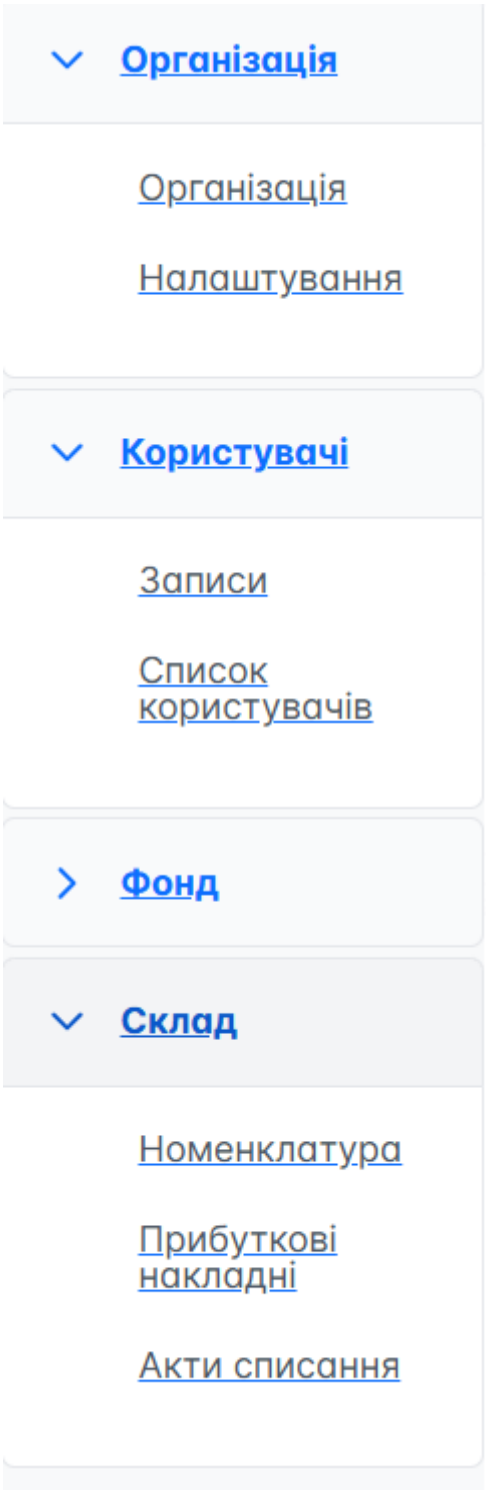

Рисунок 3.4 – Загальний вигляд меню користувача

Після авторизації користувач потрапляє на сторінку користувачів.

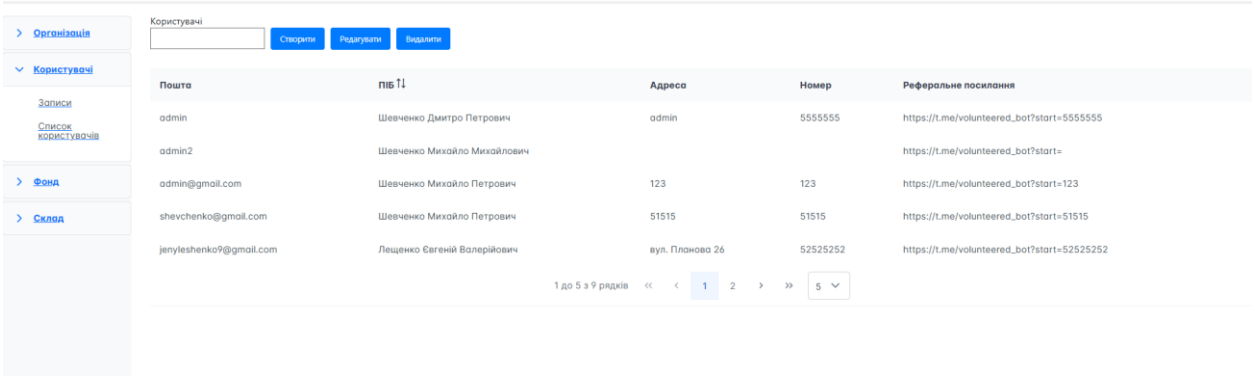

Рисунок 3.5 – Вигляд сторінки користувачів

На сторінці користувачів можна побачити усіх користувачів, які зареєстровані в організації. Тобто можна отримати список усіх тимчасово переміщених осіб, які отримують допомогу в рамках заданої організації в будь-якому фонді

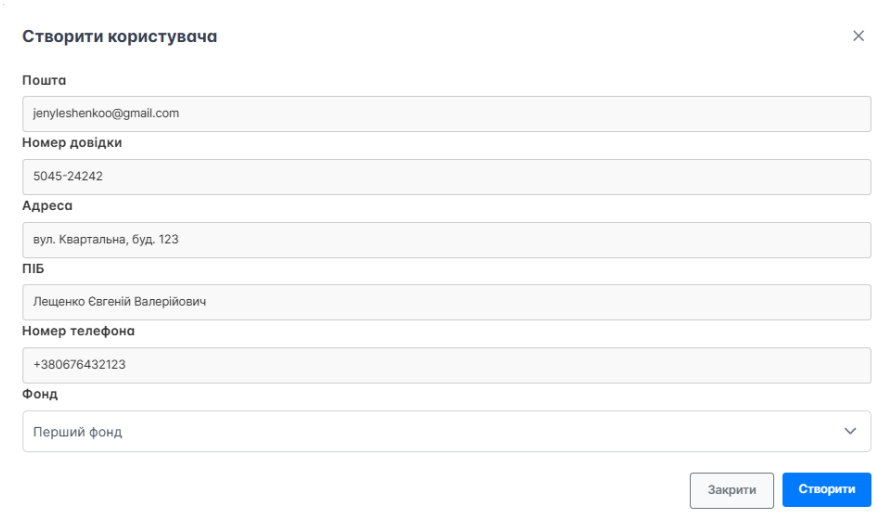

Рисунок 3.6 – Форма створення користувача

Користувач також має можливість додати інших користувач аби в подальшому записати на видачу гуманітарної допомоги. Користувач прив’язується до фонду в рамках організації в якому він зможе отримати гуманітарну допомогу. Важливо розуміти, що видача гуманітарної допомоги відбувається в рамках конкретного фонду, але побачити дані при видачу можна в рамках організації, це забезпечує прозорість та зручність для контролю над гуманітарною допомогою

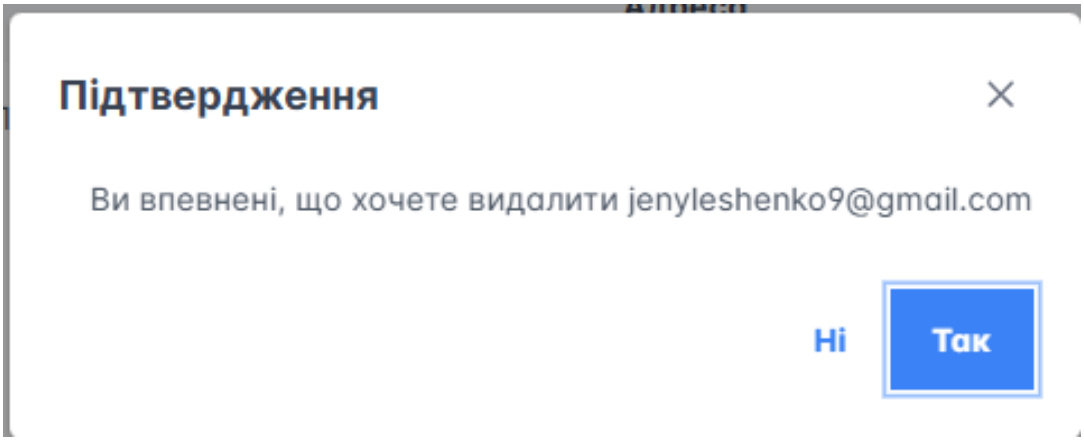

Рисунок 3.7 – Видалення користувача

Користувач має можливість видалити з організації задля того щоб ВПО, якого видалили не брав участі в видачах гуманітарної допомоги. Архітектурою закладено, що жодна інформація не видаляється з бази даних, а просто помічається на видалення задля збереження історії дій користувача.

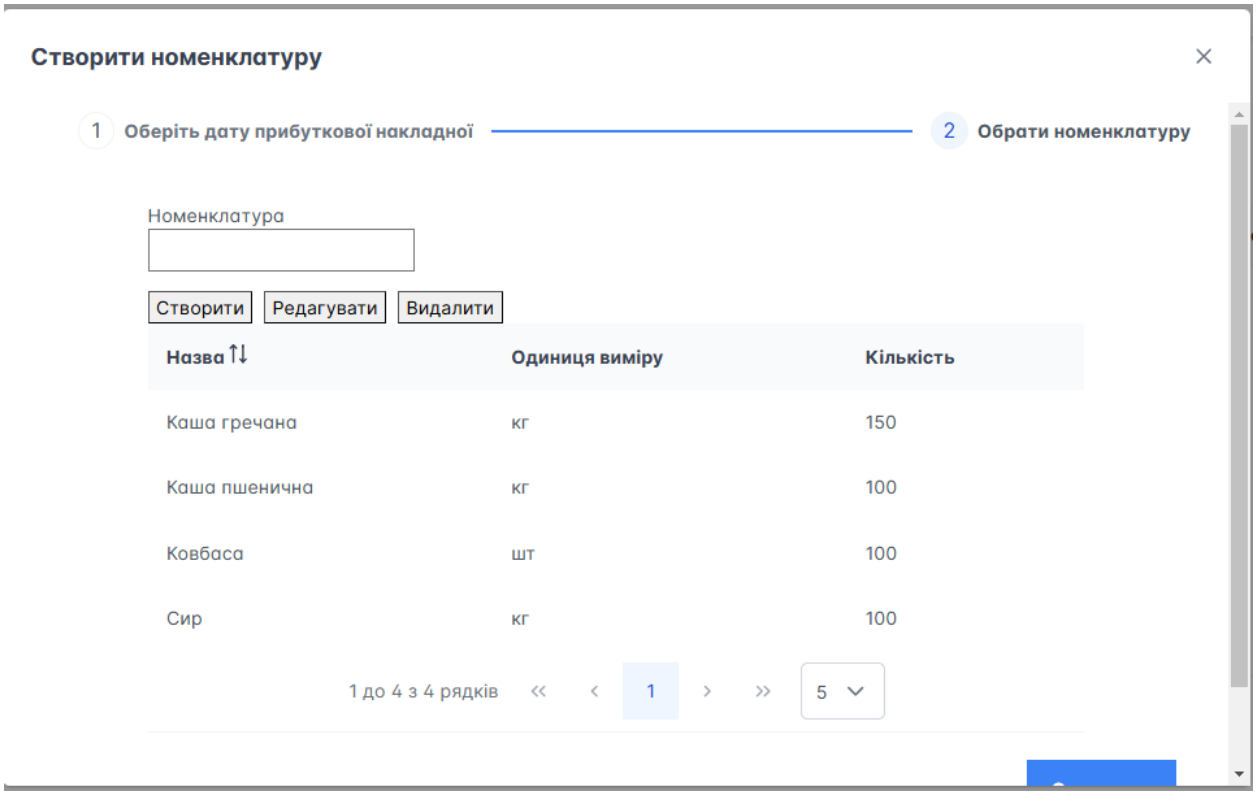

Рисунок 3.8 – Створення прибуткової накладної

При отримані продуктів на склад користувач має створити прибуткову накладну з номенклатурою, яку було отримано. Спочатку створюється

накладна, другим пунктом прив’язуються товари для цієї накладн

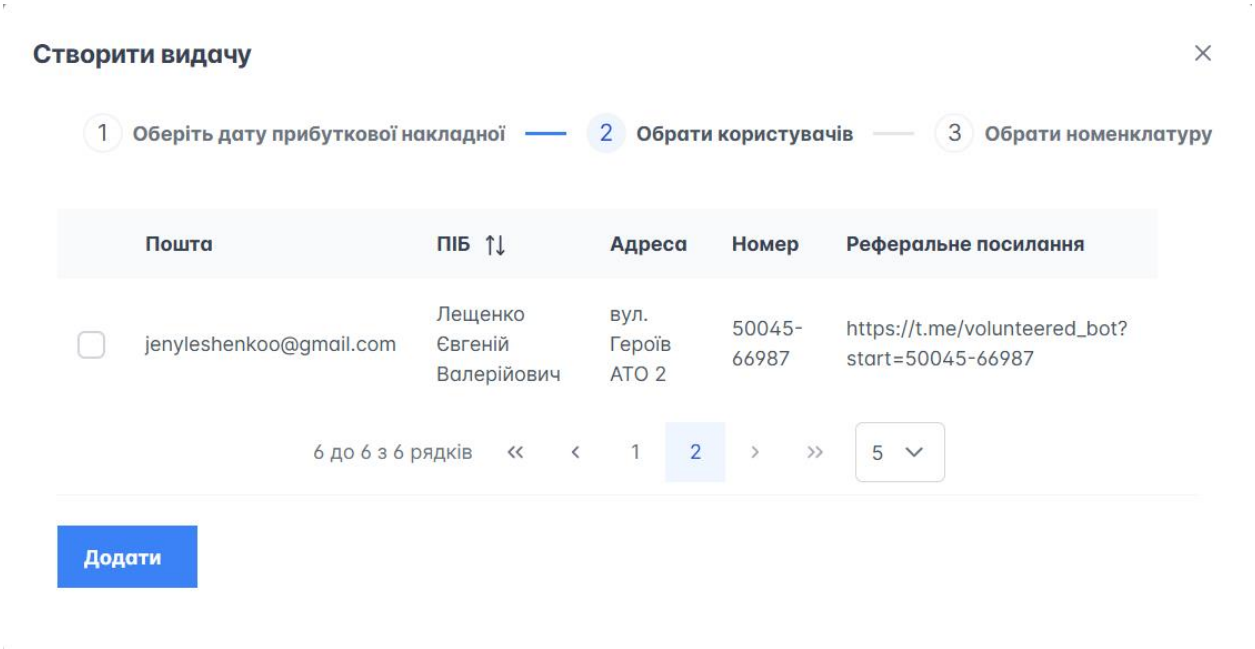

Рисунок 3.9 – Форма створення видачі гуманітарної допомоги

Спочатку створюються сама видача, номер, обирається дата та фонд в якому вона буде відбуватись. Потім обираються користувачі яким буде видана гуманітарна допомога. Останнім етапом обирається номенклатура на користувача яка буде видана. При створені видачі гуманітарної допомоги обрана кількість номенклатури резервується та розраховується на залишках номенклатури.

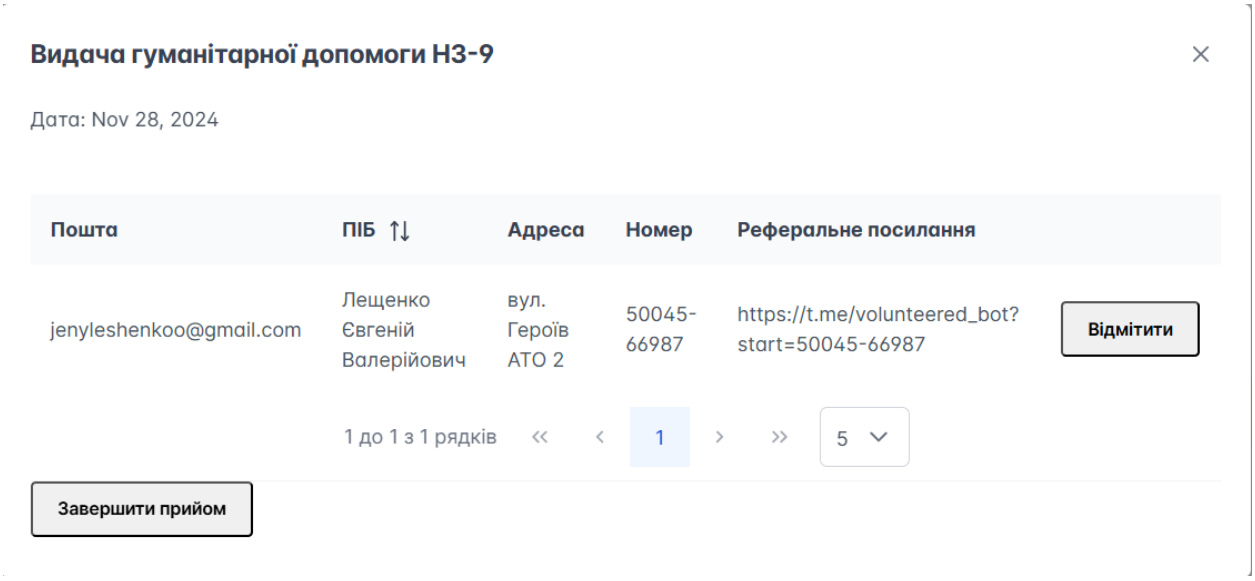

Рисунок 3.10 – Процес видачі гуманітарної допомоги

Під час видачі гуманітарної допомоги користувач має можливість відмітити тих, кому була видана гуманітарна допомога. Після того, як користувач був відмічений, оновлюється таблиця з користувачами, які ще не були відмічені наприкінці видачі користувач має натиснути «Завершити прийом» після чого він отримує статистику видачі гуманітарної допомоги.

Рисунок 3.11 – Звіт видачі гуманітарної допомоги

Після цього залишки списуються з залишків номенклатури за тих користувачів які отримали гуманітарну допомоги, за тих користувачів які не були присутні на видачі гуманітарної допомоги – залишки повертаються на склад.

3.3. Робота з Telegram ботом

Користувач, який хоче отримати гуманітарну допомогу, має можливість приєднатись за реферальним посиланням від гуманітарного фонду, тоді він матиме можливість отримувати сповіщення при гуманітарну допомогу від цього гуманітарного фонду.

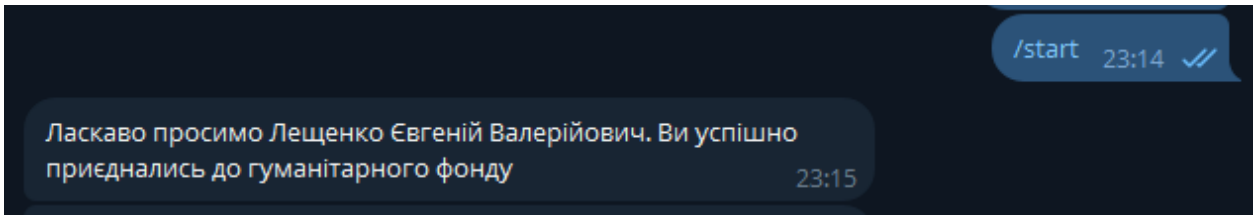

Рисунок 3.12 – Приєднання за реферальним посиланням

Після приєднання за реферальним посиланням, користувач потрапляє в головне меню, яке складається з таких пунктів як:

- Мої роздачі – історія видачі гуманітарної допомоги користувача;
- Мій фонд – інформація про гуманітарний фонд в якому знаходиться користувач;
- Записатись на наступну роздачу – можливість переглянути список наступних роздач гуманітарного фонду на які можливо записатись, можливість переглянути номенклатуру, яка буде видана;
- Технічна підтримка – технічна підтримка фонду боту;

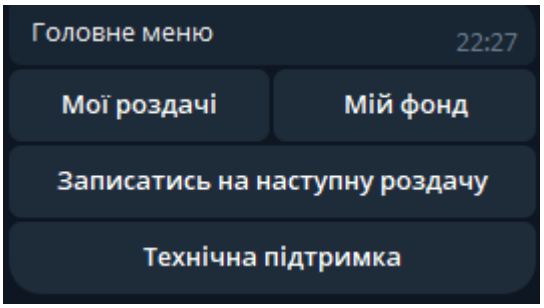

Рисунок 3.11 – Вигляд головного меню телеграм бота

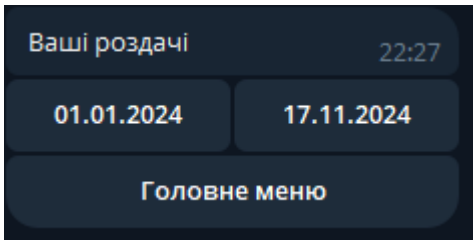

Рисунок 3.12 – Перехід в пункт меню «Мої роздачі»

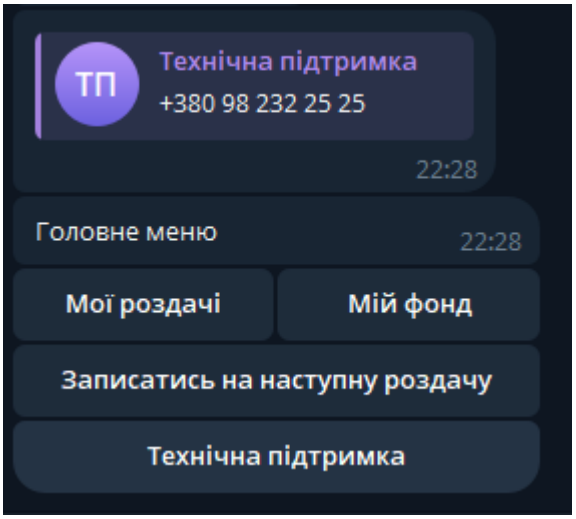

Рисунок 3.13 – Перехід в пункт меню «Технічна підтримка»

Отже був створений бот, який синхронізується з даними, які надійшли в API і які містяться в базі даних. Цей бот був створений для допомоги

користувачеві в інформації про фонд та видачі гуманітарної допомоги, також для перегляду історії видачі гуманітарної допомоги і можливості записатись на наступну.

ВИСНОВКИ

Дипломний проект "Інформаційна система управління розподілом гуманітарної допомоги з розробкою інтерактивного помічника користувача" спрямований на розробку онлайн-платформи для ефективного управління гуманітарною допомогою в Україні. Веб-сайт, створений за допомогою технологій .NET та Angular, надавати допомогу через зручний та інтуїтивно зрозумілий інтерфейс.

Система забезпечує безпечну взаємодію між користувачами та фондом, використовуючи сучасні методи захисту персональних даних, зокрема шифрування передачі даних, аутентифікацію та валідацію користувачів. Веб-додаток інтегровано з Telegram-ботом для зручної комунікації та взаємодії з користувачами, що дозволяє легко отримувати інформацію про видачу гуманітарної допомоги.

У процесі розробки було проведено тестування функціональності сайту, перевірено його стабільність та продуктивність. Виявлені проблеми були усунуті, що значно підвищило якість системи. Крім того, були проведені аудити безпеки, що підтвердило надійність платформи від атак та зловживань.

Результати дипломного проекту демонструють успішне виконання поставлених цілей. Реалізована система забезпечує ефективний та безпечний механізм управління фондом гуманітарної допомоги, а інтеграція з Telegram-ботом дозволяє зручніше взаємодіяти з користувачами. Подальше вдосконалення проекту сприятиме його широкому використанню та позитивному впливу на суспільство.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Слово і Діло. Переміщення внутрішньо переміщених осіб. URL: https://www.slovoidilo.ua/
2. Осадчук Володимир. Розробка системи ранжування авторів за публікаціями та цитуваннями (бекенд, ASP.NET Core). Робота створює базу даних авторів з їхніми публікаціями та API для ранжування авторів за певними критеріями. Використано ASP.NET Core та Ms SQL Server.
3. Документація по Angular. URL: https://angular.dev/
4. Телеграм бот FAQ. Документація Telegram bot API. URL: https://telegram.org/faq
5. Асинхронність у C#. Правила та рекомендації. URL: https://dou.ua/lenta/articles/asynchronous-programming/
6. Робота HTTP протоколу у взаємодії з ASP.NET Core API. URL: https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient
7. Довідник Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/
8. Іванова, О. (2022). Розробка інформаційної системи для обліку та видачі гуманітарної допомоги. Науковий вісник Національного університету "Львівська політехніка".
9. Петренко, І. (2021). Огляд фреймворка ASP.NET для розробки веб-додатків. Інформаційні технології та комп'ютерна інженерія.
10. Сидоренко, В. (2023). Тестування стратегій для проектів на ASP.NET. Журнал програмної інженерії.
11. Ковальчук, М. (2022). Інформаційна система для управління гуманітарною допомогою: підвищення відповідальності. Вісник науки і технологій.
12. Дубовий, В. (2021). ASP.NET: Комплексний посібник з розробки веб-додатків. Київ: Видавництво "Софтпрес".

13. Прокопенко, Н. (2022). Оцінка ефективності інформаційної системи для управління гуманітарною допомогою. Журнал оцінки технологій.
14. Андрієнко, Ю. (2022). Роль інформаційних систем у управлінні ланцюгом постачання гуманітарної допомоги. Міжнародний журнал логістики.
15. Марченко, О. (2022). Оцінка задоволеності користувачів інформаційною системою для видачі гуманітарної допомоги. Журнал користувацького досвіду, 6(3), 175-192.
16. Smith, J. (2022). Design and Implementation of an Information System for Humanitarian Aid Distribution. Journal of Information Systems.
17. Johnson, A. (2021). An Overview of ASP.NET MVC Framework for Web Application Development. International Journal of Web Development.
18. Brown, K. (2023). Testing Strategies for ASP.NET MVC Projects. Software Testing Journal.
19. Thompson, M. (2022). Enhancing Accountability in Humanitarian Assistance through Information Systems. Journal of Humanitarian Technology.
20. Davis, R. (2021). ASP.NET MVC: A Comprehensive Guide to Building Web Applications. New York, NY: Springer.
21. Clark, L. (2022). Evaluating the Effectiveness of an Information System for Humanitarian Aid Management. Journal of Information Technology Evaluation.
22. Wilson, S. (2021). Agile Development Practices in ASP.NET MVC Projects. Software Development Quarterly.
23. Adams, E. (2022). The Role of Information Systems in Humanitarian Aid Supply Chain Management. International Journal of Logistics Management.
24. Martinez, G. (2021). Building Secure Web Applications with ASP.NET MVC. Journal of Web Security.
25. Roberts, P. (2022). An Evaluation of User Satisfaction with an Information System for Humanitarian Aid Distribution. Journal of User Experience.
26. Turner, B. (2021). ASP.NET MVC: Best Practices for Scalable Web Application Development. Software Engineering Journal.

27. Тронь В. В., Маринич І. А. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 174 - Автоматизація, комп’ютерно-інтегровані технології та робототехніка”. Кривий Ріг: Видавничий центр КНУ, 2022. 50 с.
28. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с.(Інформація та документація).
29. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с.(Інформація та документація).
30. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с.(Інформація та документація).
31. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).