ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для підтримки молодіжних ініціатив громадської організації “Лідерська ініціатива”»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим ТАРАСЮК
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Максим ТАРАСЮК

Керівник: Ірина ЗАМРІЙ
д.т.н., професор

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Тарасюку Максиму Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для підтримки молодіжних ініціатив громадської організації “Лідерська ініціатива”»
керівник кваліфікаційної роботи д.т.н., професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки веб-застосунків, опис роботи веб-додатків, документація фреймворків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд застосунку для підтримки молодіжних ініціатив та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації застосунку для підтримки молодіжних ініціатив.
3. Проєктування архітектури застосунку для підтримки молодіжних ініціатив.
4. Програмна реалізація та тестування застосунку для підтримки

молодіжних ініціатив.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма розгортання клієнт-серверної моделі.
6. Діаграма структури клієнтської частини.
7. Діаграма розгортання.
8. Діаграма послідовності.
9. Діаграма класів.
10. Екранні форми.
11. Демонстрація роботи застосунку.
12. Апробація результатів дослідження.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	3.03-06.03.2025	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2025	
3	Огляд та аналіз застосунку	14.03-15.03.2025	
4	Огляд та аналіз засобів реалізації застосунку	16.03-20.03.2025	
5	Проєктування архітектури застосунку	21.03-03.04.2025	
6	Програмна реалізація та тестування застосунку	04.04-27.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	28.04-04.05.2025	
8	Розробка демонстраційних матеріалів	05.05-13.05.2025	
9	Попередній захист роботи	15.05-31.05.2025	

Здобувач вищої освіти _____ Максим ТАРАСЮК
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 1 табл., 19 рис., 20 джерел.

Мета роботи – підтримка управління молодіжними ініціативами громадської огранізації.

Об'єкт дослідження – процес управління інформацією та взаємодії в рамках лідерських ініціатив.

Предмет дослідження – програмне забезпечення для управління даними громадської організації.

Короткий зміст роботи: У роботі проаналізовано предметну галузь та методи реалізації веб-додатку для управління лідерськими ініціативами. Проведено огляд потреб цільової аудиторії (волонтери, студенти, менеджери) та сформульовано вимоги до функціональності, зокрема: відображення інформації про команду, напрями діяльності, новини, обробка заявок на співпрацю та вступ до організації, інтеграція з Telegram через вебхуки. Розроблено web-застосунок із використанням ASP.NET Core MVC для серверної частини, Razor і JavaScript (з бібліотеками Slick Carousel і Font Awesome) для клієнтської частини, Entity Framework Core і SQLite для зберігання даних. Реалізовано основні функціональні можливості: відображення каруселей зображень, обробка форм, асинхронна взаємодія з Telegram через вебхуки, обробка помилок через Middleware. Проведено ручне тестування функціональності, зокрема коректність відображення сторінок, роботи каруселей і обробки вебхуків. Для тестування локального сервера використано ngrok.

Сферою використання додатку є організація діяльності громадських ініціатив, зокрема координація команд, обробка заявок і комунікація через Telegram.

КЛЮЧОВІ СЛОВА: ASP.NET CORE MVC, C#, ENTITY FRAMEWORK CORE, SQLITE, JAVASCRIPT, SLICK CAROUSEL, WEBHOOK, TELEGRAM, WEB-ЗАСТОСУНОК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment (наприклад, Visual Studio, яка використовується для розробки)

API – Application Programming Interface (наприклад, API для вебхуків Telegram)

MVC – Model-View-Controller (архітектура, на якій побудовано проєкт)

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

ORM – Object-Relational Mapping (реалізовано через Entity Framework Core)

EF Core – Entity Framework Core (інструмент для роботи з базою даних SQLite)

DI – Dependency Injection (використовується в ASP.NET Core для ін’єкції залежностей)

JSON – JavaScript Object Notation (формат даних, який може використовуватися для передачі даних між сервером і клієнтом)

Razor – Razor-синтаксис (використовується в .cshtml файлах для створення динамічних сторінок)

SQLite – Легка реляційна база даних (використовується для зберігання даних у проєкті)

Slick – Slick Carousel (бібліотека для створення каруселей)

FA – Font Awesome (бібліотека для іконок, що використовується в дизайні інтерфейсу)

Webhook – Вебхук (механізм для інтеграції з Telegram через TelegramBotService)

ВСТУП

Актуальність: громадські ініціативи та лідерські проєкти відіграють важливу роль у сучасному суспільстві, сприяючи розвитку спільнот, залученню молоді до соціальних змін і формуванню активної громадянської позиції. Зі зростанням цифрових технологій з'являються нові можливості для організації таких ініціатив через веб-платформи, які забезпечують ефективне управління інформацією, координацію команд і взаємодію з учасниками. Веб-додатки дозволяють централізувати дані про напрями діяльності, команду, новини та заявки, а також інтегрувати зовнішні сервіси, такі як Telegram, для оперативної комунікації. Це робить процес управління ініціативами більш прозорим, зручним і доступним для широкої аудиторії, включаючи волонтерів, студентів і менеджерів.

web-застосунок виступає як універсальний інструмент, що об'єднує всі необхідні функції для координації лідерських ініціатив в одному місці. На платформі можуть бути представлені дані про напрями діяльності, членів команди, новини, а також форми для подання заявок на співпрацю чи вступ до організації. Інтеграція з Telegram через вебхуки дозволяє швидко обробляти запити користувачів, а використання сучасних бібліотек, таких як Slick Carousel, додає інтерактивності й привабливості інтерфейсу. Завдяки цьому організація діяльності стає більш структурованою, гнучкою та ефективною.

Об'єктом дослідження є процес управління інформацією та взаємодії в рамках лідерських ініціатив.

Предметом дослідження є програмне забезпечення для управління даними громадської організації.

Метою роботи є підтримка управління молодіжними ініціативами громадської огранізації.

Методи дослідження: першим кроком було проаналізовано потреби цільової аудиторії (волонтери, студенти, менеджери) для формулювання функціональних і нефункціональних вимог до програмного забезпечення. Далі проведено огляд стеку

технологій, що відповідають вимогам: обрано Visual Studio як IDE, C# і JavaScript як мови програмування, ASP.NET Core MVC як веб-фреймворк, Entity Framework Core і SQLite для роботи з базою даних, Slick Carousel і Font Awesome для клієнтської частини, ngrok для тестування вебхуків. Дослідження реалізації проводилося під час розробки, включаючи створення моделей, контролерів, Middleware, інтеграцію з Telegram і ручне тестування функціональності.

Наукова новизна роботи визначається наступними функціональними складовими: інтеграція вебхуків для асинхронної взаємодії з Telegram через локальний сервер із використанням ngrok; кастомний Middleware для обробки помилок із логуванням і перенаправленням; використання Slick Carousel для створення інтерактивних каруселей зображень членів команди.

Практична значущість результатів полягає в можливості використання розробленого веб-додатку для ефективного управління лідерськими ініціативами, координації команд, обробки заявок і комунікації через Telegram, що може бути застосовано громадськими організаціями на різних рівнях.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ КООРДИНАЦІЇ ЛІДЕРСЬКИХ ІНІЦІАТИВ

1.1 **Web-застосунок для координації лідерських ініціатив**

Web-застосунок для координації лідерських ініціатив — це спеціалізована онлайн-платформа, яка забезпечує централізоване управління інформацією, взаємодію з учасниками та організацію діяльності громадських проєктів. Такі веб-додатки допомагають користувачам (волонтерам, студентам, менеджерам) отримувати доступ до даних про напрями діяльності, команду, новини, а також подавати заявки на співпрацю чи вступ до організації. Основна мета такого веб-додатку полягає у створенні зручного та ефективного інструменту для координації ініціатив, що дозволяє автоматизувати процеси управління та забезпечити оперативну комунікацію через інтеграцію з зовнішніми сервісами, такими як Telegram.

Основні компоненти веб-додатку для координації лідерських ініціатив включають:

- інформаційні розділи, а саме, структуровані сторінки з інформацією про напрями діяльності, членів команди, новини та організаційні деталі;
- форми для заявок, тобто інструменти для подання заявок на співпрацю чи вступ до організації з валідацією введених даних;
- інтерактивні елементи, а саме, каруселі зображень (для відображення команди), створені за допомогою бібліотек, таких як slick carousel;
- інтеграція з месенджерами, а саме, використання вебхуків для взаємодії з telegram, що дозволяє обробляти запити користувачів у реальному часі;
- система обробки помилок, до прикладу**,** механізми для перехоплення та логування помилок із відображенням зрозумілого інтерфейсу для користувача.

Основні цілі веб-додатку для координації лідерських ініціатив:

- забезпечення доступу до актуальної інформації про організацію;
- автоматизація обробки заявок на співпрацю та вступ;
- поліпшення комунікації з учасниками через telegram;
- підвищення залученості аудиторії через інтерактивні елементи;

- забезпечення прозорості діяльності організації.

1.2 **Специфіка координації лідерських ініціатив**

Координація лідерських ініціатив має низку особливостей, які відрізняють її від інших видів діяльності. Розглянемо їх детальніше.

Організаційні аспекти:

- розподіл ролей, а саме, необхідність чіткого визначення ролей членів команди (наприклад, менеджери напрямів) для ефективного управління;
- різноманітність напрямів, до прикладу, ініціативи можуть охоплювати різні сфери (а саме, "лідер.junior", "лідер.student", "лідер.volunteer"), що вимагає гнучкої структури даних;
- обробка заявок, а саме, велика кількість заявок на співпрацю чи вступ потребує автоматизації та валідації даних.

Комунікаційні аспекти:

- оперативність, тобто важливість швидкої взаємодії з учасниками через месенджери, такі як telegram;
- прозорість, а саме, необхідність надавати відкритий доступ до інформації про діяльність організації;
- залученість, а саме, потреба в інтерактивних елементах (до прикладу, каруселі зображень) для привернення уваги аудиторії;

Технічні аспекти:

- інтеграція з зовнішніми сервісами, тобто використання вебхуків для зв’язку з telegram;
- обробка помилок, а саме, важливість надійної системи для перехоплення та логування помилок, щоб уникнути збоїв.

1.3 **Цільова аудиторія**

Web-застосунок "Лідерська Ініціатива" орієнтований на широку аудиторію, яка зацікавлена в громадських ініціативах і лідерстві. До основних категорій користувачів належать:

- студенти, а саме, молодь, яка хоче долучитися до ініціатив, таких як "лідер.student", для розвитку лідерських якостей;

- волонтери, а саме, особи, зацікавлені в напрямках таких як "лідер.volunteer", для участі в громадських проєктах;

- менеджери та організатори, а саме, керівники ініціатив, які координують діяльність команд і обробляють заявки;

- партнери та організації, тобто представники організацій, які хочуть співпрацювати через подання заявок на партнерство.

1.4 **Дослідження існуючих аналогів**

На ринку існує кілька платформ для управління громадськими ініціативами. Розглянемо три популярні аналоги: VolunteerMatch, Idealist і Mobilize.

1.4.1 **VolunteerMatch**

VolunteerMatch — це онлайн-платформа, яка з’єднує волонтерів із громадськими організаціями. Вона вирішує проблему пошуку можливостей для волонтерства, надаючи користувачам зручний спосіб знайти проєкти, що відповідають їхнім інтересам.

Основні функції VolunteerMatch включають:

-пошук можливостей, зокрема, інструменти для фільтрації проєктів за локацією, інтересами та типом діяльності;

-профілі організацій, тобто, детальна інформація про організації, їхні цілі та потреби для волонтерів;

-подання заявок, наприклад, функціонал для реєстрації та участі в проєктах через платформу;

-календар подій, зокрема, система відстеження майбутніх заходів, доступних для волонтерів;

-мобільний доступ, тобто, можливість використання платформи через веб-версію та мобільні додатки для ios і android.

Переваги VolunteerMatch включають:

-широкий вибір проєктів, зокрема, велика база організацій і різноманітних можливостей для волонтерів;

-зручний пошук, наприклад, наявність фільтрів, які дозволяють швидко знайти відповідні проєкти;

-доступність, тобто, підтримка платформи як у веб-версії, так і в мобільних додатках;

Недоліки VolunteerMatch включають:

-обмежена інтерактивність, зокрема, відсутність таких елементів, як каруселі зображень чи інтерактивні форми для користувачів;

-відсутність інтеграції з месенджерами, наприклад, неможливість прямого зв’язку через telegram для швидкої комунікації;

-складність для малих організацій, тобто, труднощі для менших ініціатив у конкуренції за увагу волонтерів на платформі;

Сторінки додатку Idealist наведено на рисунку 1.1.

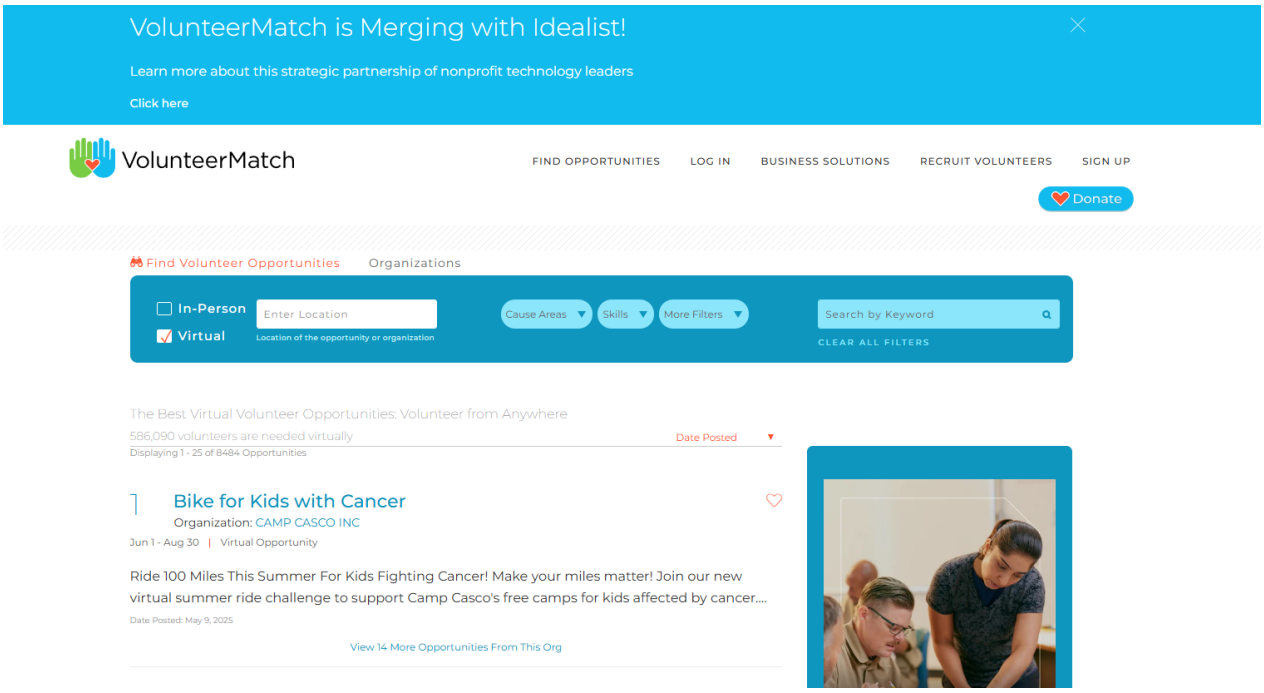


Рис. 1.1 Сторінки пошуку роботи Idealist

1.4.2 **Idealist**

Idealist — це платформа для пошуку волонтерських можливостей, стажувань і роботи в некомерційних організаціях. Вона зосереджена на підтримці соціальних ініціатив і залученні активістів.

Основні функції Idealist включають:

-пошук можливостей, зокрема, інструменти для фільтрації за типом діяльності, локацією та навичками користувача;

-інформація про організації, тобто, детальні профілі з описом місії та проєктів, які реалізує організація;

-форум спільноти, наприклад, платформа для спілкування та обміну досвідом між активістами;

-події та навчання, зокрема, інформація про тренінги, заходи та освітні можливості для користувачів;

Переваги Idealist включають;

-глибокий опис організацій, тобто, надання користувачам повної інформації про ініціативи та їхню діяльність;

-спільнота, зокрема, форум, який сприяє обміну досвідом між учасниками платформи;

-різноманітність можливостей, наприклад, наявність не лише волонтерських проєктів, а й вакансій та стажувань.

Недоліки Idealist включають:

- складність для початківців, зокрема, перевантажений інтерфейс, який може бути незручним для нових користувачів;

- відсутність інтерактивних елементів, наприклад, брак каруселей чи інших візуальних інструментів для покращення досвіду;

- обмежена інтеграція, тобто, відсутність зв’язку з месенджерами, такими як telegram, для швидкої комунікації;

Екранні форми додатку Idealist наведено на рисунку 1.2.

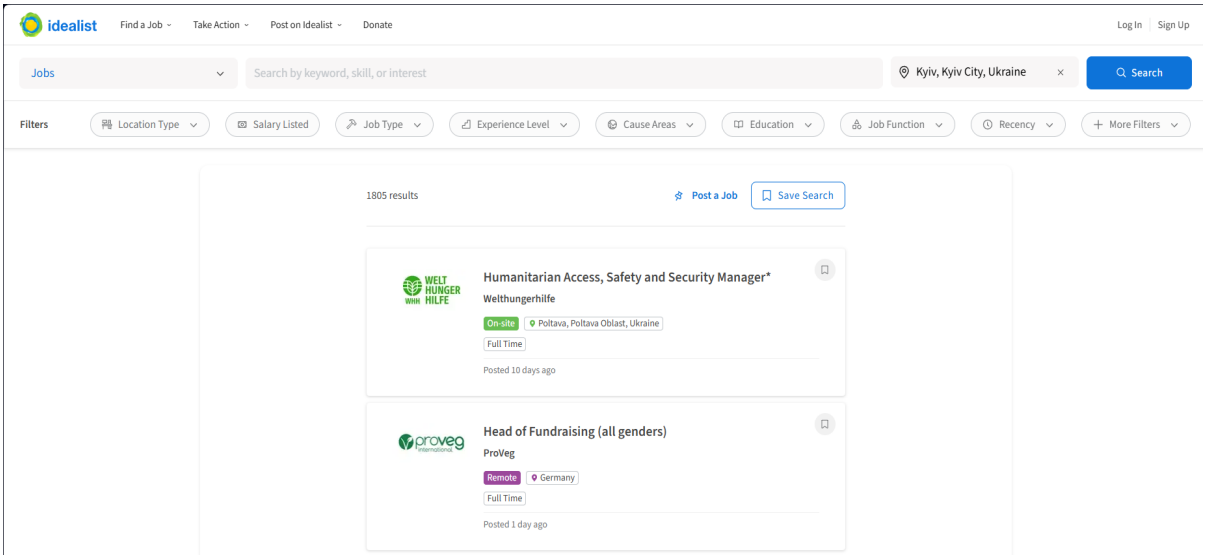


Рис. 1.2 Сторінки пошуку роботи Idealist

1.4.3 **Mobilize**

Mobilize — це платформа для організації волонтерських кампаній і заходів. Вона допомагає громадським організаціям залучати учасників і координувати їхню діяльність.

Основні функції Mobilize включають:

-організація подій, зокрема, інструменти для створення та управління подіями для волонтерів;

-реєстрація учасників, тобто, форми для запису на заходи з можливістю подання заявок;

-сповіщення, наприклад, функціонал для надсилання нагадувань учасникам через email або sms;

-аналітика, зокрема, система відстеження залученості учасників для оцінки активності.

Переваги Mobilize включають:

-зручність для організаторів, тобто, наявність інструментів для ефективного управління подіями;

-сповіщення, зокрема, автоматизовані повідомлення, які підвищують залученість учасників;

-аналітика, наприклад, можливість оцінити ефективність кампаній завдяки даним про активність.

Недоліки Mobilize включають:

-обмежена інформація, зокрема, менший обсяг даних про саму організацію порівняно з платформою idealist;

-відсутність інтерактивності, наприклад, брак візуальних елементів, таких як каруселі для покращення досвіду;

-обмежена інтеграція, тобто, відсутність зв’язку з месенджерами, такими як telegram, для швидкої комунікації.

Приклади наведено на рисунку 1.3 та 1.4.

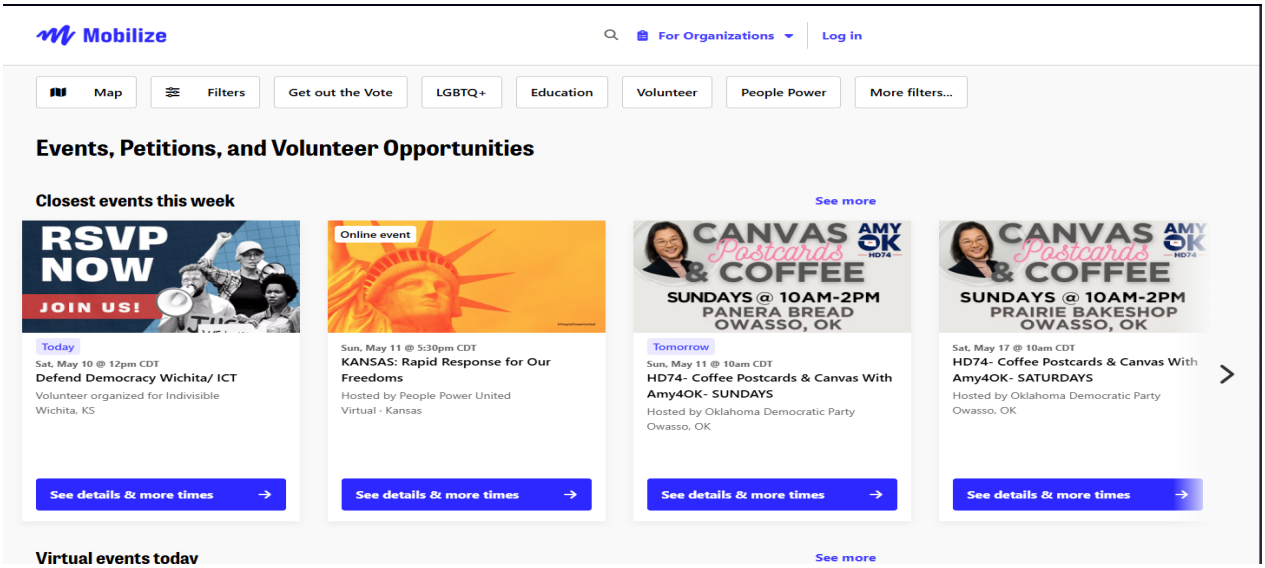


Рис. 1.3 Приклад головної сторінки Mobilize

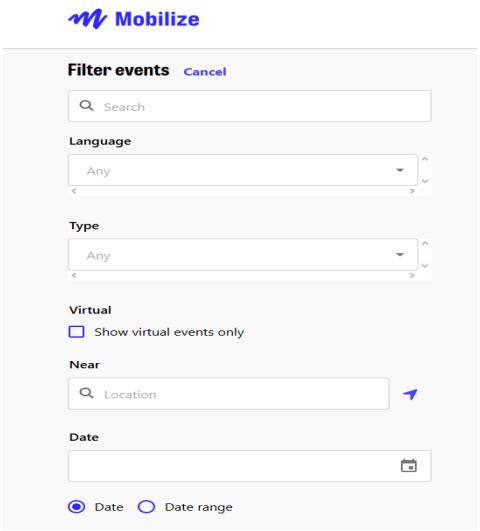


Рис 1.4 Приклад фільтрів подій Mobilize

1.5 **Порівняльний аналіз застосунків для громадських ініціатив**

В таблиці 1.1 зведено результати аналізу існуючих застосунків.

Таблиця 1.1

Порівняння існуючих програм-аналогів

Показник	**VolunteerMatch**	**Idealist**	**Mobilize**	**LeadershipInitiative**
Інформація про організацію	Базові профілі	Детальні профілі	Обмежена інформація	Детальні профілі, включаючи напрями діяльності, новини, реквізити
Заявки та форми	Форма реєстрації	Вакансії, події	Події	Форми для приєднання та співпраці з валідацією даних
Інтерактивність	Відсутня	Відсутня	Відсутня	Асинхронне оновлення даних, інтерактивні форми
Інтеграція з месенджерами	Відсутня	Відсутня	Відсутня	Інтеграція з Telegram-ботом для обробки запитів
Комунікація	Через email	Форум спільноти	Email/SMS	Через Telegram-бот для оперативного інформування
Доступність	Web, iOS, Android	Web	Web, iOS, Android	Web (з можливістю розширення)
Аналітика	Відсутня	Відсутня	Відстеження залученості	Відсутня (з можливістю додавання в майбутньому)
Цільова аудиторія	Волонтери	Волонтери, активісти	Організатори, волонтери	Волонтери, активісти, потенційні партнери
Особливості	Широкий вибір проєктів	Глибокі профілі, форум	Управління подіями, сповіщення	Інтеграція з Telegram, автоматизація заявок, динамічне інформування
Недоліки	Обмежена інтерактивність	Складний інтерфейс	Обмежена інформація	Обмежена доступність (лише Web), відсутність аналітики

1.6 **Визначчення вимог до застосунку**

Проаналізувавши специфіку координації лідерських ініціатив, цільову аудиторію та аналоги, було визначено наступні вимоги до веб-додатку "Лідерська Ініціатива":

1. Інформаційні сторінки: Відображення даних про напрями діяльності, членів команди, новини та організаційні деталі.
2. Форми для заявок: Можливість подання заявок на співпрацю та вступ до організації з валідацією даних (до прикладу, формат телефону).
3. Інтерактивний інтерфейс: Використання каруселей (Slick Carousel) для відображення команди та іконок (Font Awesome) для дизайну.
4. Інтеграція з Telegram: Обробка запитів через вебхуки з використанням ngrok для тестування.
5. Обробка помилок: Наявність Middleware для перехоплення помилок, логування та відображення сторінки помилки.
6. Швидкий і адаптивний інтерфейс: Завантаження сторінок менш ніж за 2 секунди, адаптація до різних розмірів екранів.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 **Середовище розробки**

Інтегроване середовище розробки (IDE) – це комплексне програмне забезпечення, яке надає програмістам набір інструментів для створення, тестування та налагодження програмного коду. Такі середовища зазвичай включають текстовий редактор із підсвіткою синтаксису, засоби компіляції та побудови проєктів, вбудований дебагер, а також інтеграцію з системами контролю версій. Це дозволяє розробникам ефективно працювати над проєктами в єдиному інтерфейсі.

Visual Studio – це потужне інтегроване середовище розробки, створене компанією Microsoft, яке широко використовується для розробки веб-додатків, настільних програм, мобільних додатків та інших типів програмного забезпечення. Для реалізації проєкту "Лідерська Ініціатива" було обрано Visual Studio через його підтримку платформи ASP.NET Core, яка є основою даного веб-додатку.

Visual Studio пропонує інтуїтивний інтерфейс із розширеним набором інструментів, таких як IntelliSense (система автодоповнення коду), вбудований дебагер, підтримка Git для контролю версій, а також засоби для роботи з базами даних і тестування. Ці можливості значно спрощують процес написання, налагодження та оптимізації коду.

Ключовою перевагою Visual Studio для даного проєкту є його підтримка C#, мови програмування, яка використовується в ASP.NET Core. Середовище надає вбудовані шаблони для створення контролерів, моделей і представлень, що прискорює розробку веб-додатків. Крім того, Visual Studio інтегрується з системою збірки .NET, дозволяючи легко компілювати та розгортати проєкт.

Visual Studio також підтримує розширення, які можна додавати для розширення функціональності, скажімо, для роботи з SQLite (використовуваною в проєкті базою даних) або для налаштування специфічних інструментів розробки. Це робить його гнучким інструментом, придатним як для новачків, так і для досвідчених розробників.

2.2 **Мови програмування**

Мова програмування – це формалізована система команд, яка використовується для створення програмного забезпечення шляхом визначення алгоритмів і логіки роботи додатків. Вона дозволяє розробникам створювати інструкції, які комп’ютер може виконувати, забезпечуючи ефективну взаємодію між програмістом і машиною. У проєкті "Лідерська Ініціатива" використано кілька мов програмування для реалізації як серверної, так і клієнтської частин веб-додатку.

2.2.1 **C#**

C# – це сучасна, об’єктно-орієнтована мова програмування, розроблена компанією Microsoft у 2000 році. Вона є основною мовою для створення додатків на платформі .NET, включаючи ASP.NET Core MVC, який використовується в даному проєкті. C# поєднує в собі простоту синтаксису, подібного до C і C++, із сучасними можливостями, такими як автоматичне управління пам’яттю (garbage collection) і строга типізація.

Однією з ключових переваг C# є її підтримка об’єктно-орієнтованого програмування (ООП), що дозволяє створювати модульний, повторно використовуваний код. У проєкті "Лідерська Ініціатива" C# використовується для написання серверної логіки, включаючи контролери, моделі та обробку запитів у рамках ASP.NET Core MVC. До прикладу, контролери, такі як HomeController і ErrorController, написані на C#, відповідають за обробку HTTP-запитів і повернення відповідних представлень.

C# також забезпечує асинхронне програмування через ключові слова async і await, що було використано в проєкті для обробки асинхронних операцій, таких як запити до бази даних через Entity Framework Core. Це підвищує продуктивність веб-додатку, дозволяючи ефективно обробляти велику кількість одночасних запитів.

2.2.2 **JavaScript**

JavaScript є високорівневою, інтерпретованою мовою програмування, яка відіграє ключову роль у створенні інтерактивних і динамічних веб-сторінок. Вона була розроблена Бренданом Айком у 1995 році в компанії Netscape як інструмент для додавання інтерактивності до веб-сторінок, спочатку під назвою LiveScript. Згодом, завдяки стандартизації через ECMAScript (перша версія ES1 була випущена в 1997 році), JavaScript став однією з трьох основних технологій веб-розробки поряд із HTML (структура сторінок) і CSS (стилізація). Сучасні версії мови, такі як ES6 (2015) і наступні, додали підтримку модулів, асинхронного програмування (а саме, async/await) і сучасних API, таких як Fetch API для роботи з сервером. JavaScript виконується в браузерах через рушії (наприклад, V8 у Chrome), що дозволяє розробникам створювати складні клієнтські додатки без необхідності встановлення додаткового програмного забезпечення.

JavaScript вирізняється своєю гнучкістю та універсальністю. Вона підтримує об’єктно-орієнтоване програмування (з прототипами), функціональне програмування (з функціями як об’єктами першого класу) і процедурний стиль кодування.

Основні можливості включають:

- обробка подій, а саме, реакція на дії користувача, такі як кліки, натискання клавіш або прокручування сторінки;
- валідація форм, тобто перевірка введених даних перед відправленням на сервер;
- асинхронні запити, наприклад, отримання даних із сервера без перезавантаження сторінки (технологія ajax або використання fetch api) ;
- маніпуляція dom, а саме, зміна структури та стилів html-елементів у реальному часі;
- анімації та ефекти, до прикладу, створення плавних переходів або слайдерів, таких як карусель;

у проєкті "лідерська ініціатива" javascript активно використовується для реалізації клієнтської логіки, що значно підвищує інтерактивність і зручність

використання веб-додатку. Він інтегрується з HTML і CSS у файлах представлень (.cshtml), які створюються за допомогою Razor у рамках ASP.NET Core MVC, дозволяючи комбінувати серверний C#-код із клієнтськими скриптами. Ця інтеграція дозволяє створювати гнучкий і адаптивний користувацький інтерфейс, адаптований до потреб молодіжної аудиторії, яка очікує швидкої реакції та інтерактивних елементів.

Роль JavaScript у проєкті:

У контексті "Лідерської Ініціативи" JavaScript є невід'ємною частиною клієнтської логіки, яка забезпечує інтерактивність і зручність для користувачів.

Наприклад:

- обробка подій, тобто кліки по кнопках "співпрацювати" чи "волонтерити" запускають дії, такі як відкриття форм чи переходи на нові сторінки;
- адаптивність, до прикладу, карусель slick carousel адаптується до розміру екрана, що важливо для молодіжної аудиторії, яка часто використовує мобільні пристрої;
- асинхронність, а саме, ajax-запити дозволяють оновлювати дані (до прикладу, дані про новини) без перезавантаження, що підвищує швидкість роботи сайту;
- візуальна привабливість, до прикладу, ефекти наведення чи анімації додають сучасного вигляду, що відповідає бренду організації.

JavaScript інтегрується з Razor через вбудований код у .cshtml-файли або зовнішні скрипти, підключені через

Це дозволяє централізовано управляти залежностями для всіх сторінок.

Переваги та виклики:

- переваги, а саме, гнучкість у реалізації інтерактивних елементів, швидке оновлення інтерфейсу, широка підтримка браузерів;
- виклики, тобто, необхідність забезпечення кросбраузерної сумісності (а саме, різна поведінка jquery у старих версіях ie) і залежність від правильного підключення зовнішніх бібліотек, що може спричинити помилки, якщо cdn недоступний.

Тестування JavaScript:

Тестування JavaScript у проєкті проводилося вручну, наприклад, перевірка каруселі на різних розмірах екранів і коректність роботи модальних вікон. У майбутньому можна розглядати автоматизоване тестування з використанням фреймворків, таких як Jest, для перевірки асинхронних функцій.

Таким чином, JavaScript у "Лідерській Ініціативі" забезпечує інтерактивність, адаптивність і сучасний дизайн, роблячи web-застосунок привабливим і зручним для користувачів, що сприяє реалізації цілей організації з розвитку молодіжного потенціалу.

2.2.3 **Razor**

Razor – це спеціальний синтаксис, який використовується у файлах із розширенням .cshtml для створення динамічних веб-сторінок у рамках ASP.NET Core MVC. Razor не є окремою мовою програмування, а виступає як механізм, що поєднує елементи мови програмування C# із HTML-розміткою. Це дозволяє розробникам вбудовувати серверний код безпосередньо в HTML, забезпечуючи динамічну генерацію веб-сторінок на основі даних, отриманих із сервера, або логіки, визначеної в коді. Razor працює на сервері під час обробки запиту користувача, генеруючи HTML, який потім надсилається клієнту (браузеру) для відображення.

Основна перевага Razor полягає в його здатності забезпечувати плавний перехід між HTML і C#. А саме, за допомогою символу @ можна почати вираз C#, який буде виконаний на сервері, а результат – вставлений у HTML-розмітку. Razor підтримує широкий спектр конструкцій C#, таких як цикли (@for, @foreach), умови (@if), виведення змінних (@Model) і навіть виклик методів. Це дозволяє створювати сторінки, що адаптуються до даних і логіки програми без необхідності писати складний код для обробки шаблонів.

У проєкті "Лідерська Ініціатива", який є веб-додатком на основі ASP.NET Core MVC, Razor використовується для створення всіх представлень (Views), таких як Index.cshtml, Error.cshtml, Support.cshtml, Directions.cshtml, News.cshtml, _Layout.cshtml та інших. Ці файли відповідають за відображення сторінок веб-

сайту, таких як головна сторінка, сторінка з інформацією про напрями діяльності, команду, новини, реквізити організації, а також сторінка з обробкою помилок. Razor дозволяє ефективно поєднувати статичну HTML-розмітку з динамічними даними, отриманими з моделі, контролера або бази даних.

До прикладу, файл _Layout.cshtm, відіграє ключову роль у створенні єдиного шаблону для всіх сторінок веб-додатку. Цей файл є основним макетом (Layout), який визначає спільну структуру та дизайн усіх представлень (Views), таких як Index.cshtml, Error.cshtml, Directions.cshtml тощо. Використання _Layout.cshtml дозволяє централізовано керувати HTML-розміткою, стилізацією та поведінкою сторінок, уникаючи дублювання коду та спрощуючи підтримку проєкту.

Файл _Layout.cshtml зазвичай містить базову HTML-розмітку, яка включає заголовок (<head>), навігаційну панель, основний вміст сторінки та футер. Основна ідея полягає в тому, щоб визначити спільні елементи інтерфейсу, які повторюються на кожній сторінці, а динамічний вміст окремих представлень вставляти в спеціально виділені секції. Для цього використовується Razor-синтаксис із секціями, такими як @RenderBody() і @RenderSection(), які дозволяють інтегрувати унікальний вміст із дочірніх сторінок.

Основні елементи файлу _Layout.cshtml:

1. Заголовок (<head>) , який:

- визначає метадані сторінки (до прикладу, кодування charset і адаптивність через viewport) ;
- динамічний заголовок сторінки задається через @viewdata["title"], який дочірні сторінки можуть перевизначити;
- підключаються зовнішні стилі (а саме, bootstrap через cdn) і локальні css-файли (наприклад, site.css).

2. Навігаційна панель (<nav>), що:

- містить меню з посиланнями на основні розділи сайту ("хто ми", "напрямки", "підтримати", "новини", "співпрацювати");
- використовує компоненти bootstrap для адаптивного дизайну, який розгортається на мобільних пристроях.

3. Основний вміст (<main>), що:

- використовує @renderbody(), щоб вставити вміст із дочірнього файлу .cshtml (таких як, index.cshtml) ;
- дозволяє кожній сторінці додавати свій унікальний вміст у межах загального макета.

4. Футер (<footer>), який:

- містить копірайт і додаткову інформацію (до прикладу, рік і назву проєкту) ;
- використовує bootstrap для стилізації.

5. Скрипти, які:

- підключають JavaScript-файли (а саме, Bootstrap JS через CDN).

Також можна сказати про @RenderSection("Scripts", required: false), що дозволяє дочірнім сторінкам додавати власні скрипти, якщо потрібно (до прикладу, для валідації форм).

2.3 **Веб-фреймворки**

Веб-фреймворк – це програмний каркас, призначений для спрощення розробки веб-додатків шляхом надання стандартних бібліотек, інструментів і шаблонів. У проєкті "Лідерська Ініціатива" використано наступний веб-фреймворк.

ASP.NET Core MVC є потужним веб-фреймворком, розробленим компанією Microsoft, який входить до складу платформи ASP.NET Core – сучасної, кросплатформної екосистеми для створення веб-додатків, API та мікросервісів. Фреймворк базується на моделі-вид-презентатор (Model-View-Controller, MVC), що забезпечує чіткий поділ відповідальності між компонентами: моделі обробляють дані, контролери керують логікою, а представлення генерують інтерфейс користувача. ASP.NET Core MVC є еволюцією класичного ASP.NET MVC, представленого в 2009 році, і був перероблений у 2016 році разом із випуском ASP.NET Core 1.0 для підвищення продуктивності, модульності та підтримки операційних систем, таких як Windows, macOS і Linux. У проєкті "Лідерська Ініціатива" цей фреймворк було обрано завдяки його гнучкості, високій продуктивності, широкій інтеграції з сучасними технологіями (наприклад, Entity

Framework Core і Telegram Bot API) та підтримці розширених можливостей розробки, що ідеально відповідає потребам динамічного веб-додатку для підтримки молодіжних ініціатив.

Архітектура ASP.NET Core MVC

Архітектура MVC у ASP.NET Core забезпечує структурований підхід до розробки веб-додатків, що полегшує підтримку та масштабування коду. Кожен компонент відіграє специфічну роль:

- моделі (model), які представляють дані та логіку бізнес-об’єктів. у "лідерській ініціативі" моделі, такі як direction, teammember і news, визначені в папці /models і використовуються для зберігання інформації про напрями діяльності, команду та новини;

- контролери (controller) які відповідають за обробку http-запитів, взаємодію з моделями і повернення відповідних представлень. у проєкті реалізовані контролери, такі як homecontroller і errorcontroller, які керують основними сторінками. до прикладу, метод index у homecontroller отримує дані з бази даних і передає їх у представлення;

- контролер використовує ін’єкцію залежностей для доступу до контексту бази даних, що є частиною сучасного підходу asp.net core до управління сервісами;

- представлення (view)які генерують html-вивід на основі даних із моделей. у проєкті представлення реалізуються через файли .cshtml із використанням razor-синтаксису.

Ключові можливості asp.net core mvc.

Asp.net core mvc надає ряд потужних інструментів, які були задіяні в "лідерській ініціативі":

- маршрутизація, а саме, вбудована система маршрутизації дозволяє визначати url-адреси для сторінок. у program.cs налаштована стандартна маршрутизація;

- middleware який є гнучким механізмом і дозволяє додавати обробку запитів на різних етапах.

- авторизація та безпека, хоча в поточній версії проєкту авторизація не реалізована в повному обсязі, asp.net core mvc підтримує вбудовані засоби для захисту маршрутів, які можуть бути додані в майбутньому для обмеження доступу до адміністративних сторінок.

Роль ASP.NET Core MVC у проєкті.

У "Лідерській Ініціативі" ASP.NET Core MVC слугує основою для реалізації серверної та клієнтської логіки. Контролери обробляють запити до сторінок (скажімо, перегляд новин чи подання заявок), моделі забезпечують структуру даних для бази даних, а представлення генерують інтерактивний інтерфейс. Інтеграція з вебхуками дозволяє повідомляти користувачів через Telegram, що є важливим для залучення молодіжної аудиторії. Асинхронна обробка і middleware гарантують стабільність і швидкість роботи, а підтримка адаптивного дизайну через Razor і JavaScript робить web-застосунок зручним для використання на різних пристроях.

Таким чином, ASP.NET Core MVC забезпечує надійну, масштабовану і сучасну платформу для "Лідерської Ініціативи", сприяючи реалізації її цілей із розвитку молодіжного потенціалу через технологічний і інтерактивний web-застосунок.

2.4 **Інструменти роботи з базами даних**

Інструменти роботи з базами даних дозволяють розробникам ефективно зберігати, отримувати та управляти даними у веб-додатках. У проєкті "Лідерська Ініціатива" використано наступні технології для роботи з базою даних.

2.4.1 **Entity Framework Core**

Entity Framework Core (EF Core) – це легковаговий, відкритий об’єктно-реляційний мапер (ORM), розроблений Microsoft для платформи .NET. EF Core використовується в проєкті для спрощення взаємодії з базою даних SQLite, дозволяючи мапувати об’єкти C# на таблиці бази даних.

EF Core забезпечує автоматичне створення міграцій (migration), що дозволяє оновлювати структуру бази даних на основі змін у моделях. У проєкті це

застосовувалося для створення таблиць, таких як TeamMembers і Directions, із відповідними моделями. Завдяки асинхронним методам (таких як ToListAsync), EF Core підвищує продуктивність при отриманні даних, що важливо для веб-додатків із динамічним вмістом.

Крім того, EF Core підтримує гнучкі конфігурації, дозволяючи налаштовувати відображення між об’єктами та таблицями через Fluent API або атрибути, що забезпечує кращий контроль над схемою бази даних. У проєкті це було використано для додавання індексів та унікальних обмежень, як приклад, для поля PhoneNumber у таблиці JoinOrganizationRequests, що оптимізує запити та запобігає дублюванню даних. Також EF Core інтегрується з Dependency Injection у ASP.NET Core, що полегшує управління контекстом бази даних (AppDbContext) та його ін’єкцію в контролери, роблячи код більш модульним і тестопридатним. Ця інтеграція сприяла ефективній реалізації асинхронних операцій, таких як збереження нових запитів на співпрацю в реальному часі.

2.4.2 **SQLite**

SQLite є легкою, вбудованою реляційною системою управління базами даних (СУБД), яка вирізняється відсутністю необхідності в окремому серверному процесі, що робить її ідеальним вибором для невеликих і середніх проєктів. Розроблена у 2000 році Двейном Річардом Хіпсом, SQLite стала однією з найпоширеніших вбудованих СУБД завдяки своїй простоті, компактності та широкій підтримці платформ, включаючи Windows, macOS, Linux, iOS і Android. У проєкті "Лідерська Ініціатива" SQLite була обрана як основна СУБД завдяки своїй легкості інтеграції, мінімальним вимогам до ресурсів і достатній функціональності для зберігання різноманітних даних, таких як інформація про команду (таких як імена, посади, фотографії), напрями діяльності (таких як назви, описи, статуси), новини, запити на співпрацю та організаційні реквізити. Ця технологія дозволяє ефективно підтримувати динамічний вміст веб-додатку, не ускладнюючи інфраструктуру, що є важливим для молодіжного проєкту з обмеженим бюджетом і масштабом.

Архітектура та особливості SQLite

SQLite працює у файловому режимі, де вся база даних зберігається в одному файлі (скажімо, app.db), що спрощує її перенесення, резервне копіювання та розгортання. На відміну від клієнт-серверних СУБД, таких як PostgreSQL чи Microsoft SQL Server, SQLite не потребує запущеного серверного процесу, що зменшує накладні витрати на адміністрування. Вона підтримує стандарт SQL (з деякими обмеженнями, наприклад, відсутність повної підтримки JOIN для великих обсягів даних) і забезпечує швидкий доступ до даних завдяки вбудованим індексам і транзакційній системі. Основні особливості включають:

-компактність, а саме, розмір бібліотеки sqlite становить менше 500 кб, що дозволяє її використовувати навіть на пристроях із обмеженими ресурсами;

-транзакційна цілісність, тобто, підтримка acid (атомарність, узгодженість, ізоляція, довговічність) гарантує надійність даних при одночасному доступі;

-простота налаштування, а саме, необхідно лише вказати шлях до файлу бази даних у конфігурації, що робить її доступною для швидкого старту;

-кросплатформність, тобто, сумісність із різними платформами забезпечує гнучкість розгортання проєкту.

У "Лідерській Ініціативі" база даних SQLite зберігається в корені проєкту як файл app.db, а доступ до неї налаштовано через рядок підключення в appsettings.json.

SQLite інтегрується з ASP.NET Core через Entity Framework Core (EF Core) – об’єктно-реляційний мапер (ORM), який спрощує взаємодію між об’єктами C# і таблицями бази даних. Ця інтеграція дозволяє розробникам виконувати операції CRUD (Create, Read, Update, Delete) без необхідності ручного написання складних SQL-запитів.

2.5 **Інші інструменти та бібліотеки**

Для реалізації додаткових функцій і тестування проєкту "Лідерська Ініціатива" використано низку допоміжних інструментів і бібліотек.

2.5.1 **JavaScript-бібліотеки**

Для створення інтерактивного інтерфейсу у проєкті використано JavaScript-бібліотеки:

Slick Carousel – бібліотека для створення каруселей з фотографіями. Вона підключається через CDN (наприклад, <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick-theme.min.css" />), що дозволяє легко реалізувати слайдери зображень на сторінках проєкту.

Font Awesome – бібліотека іконок, підключена через CDN (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css" />), яка використовується для додавання стилізованих іконок до інтерфейсу.

Ці бібліотеки інтегруються з HTML і CSS у файлах .cshtml, забезпечуючи візуальну привабливість і зручність для користувачів.

2.5.2 **Webhook**

Webhook – це механізм, який дозволяє зовнішнім сервісам (таких як Telegram) надсилати HTTP-запити до вашого додатку у реальному часі. У проєкті "Лідерська Ініціатива" webhook використовується для обробки повідомлень від Telegram-бота. Реалізація включає асинхронну обробку запитів у Middleware, що дозволяє інтегрувати зовнішні дані в web-застосунок.

Реалізація webhook у проєкті включає створення спеціального контролера, такого як TelegramWebhookController, який обробляє вхідні POST-запити від Telegram API. Цей контролер використовує атрибут [HttpPost] і маршрут [Route("api/telegram")] для прийому оновлень у форматі JSON, переданих через об’єкт Update бібліотеки Telegram.Bot. Асинхронна обробка запитів у Middleware дозволяє додатку ефективно реагувати на велику кількість одночасних оновлень, що є важливим для масштабованих систем. Як приклад, метод Post у контролері асинхронно викликає сервіс TelegramBotService для обробки отриманих даних, що підвищує продуктивність і дозволяє уникнути блокування основного потоку виконання.

Для реалізації webhook у проєкті було використано наступні кроки:

1. Налаштування URL-адреси webhook через метод SetWebhook Telegram Bot API з використанням інструменту ngrok для експонування локального сервера назовні, що забезпечило безпечний доступ до вашого додатку під час розробки.

2. Інтеграція з AppDbContext для збереження даних із Telegram (як приклад, стану користувача UserState) у базі даних SQLite, що дозволяє відстежувати прогрес користувачів у взаємодії з ботом.

3. Обробка помилок із логуванням у консоль (а саме, Console.WriteLine), що допомагає діагностувати проблеми, такі як некоректні оновлення чи з'єднання.

4. Використання асинхронних методів (async Task<IActionResult>) для забезпечення неblокуючої обробки запитів, що особливо важливо при високому навантаженні, до прикладу, під час масових повідомлень від користувачів.

Ця реалізація також включає валідацію вхідних даних (перевірка на null у Update) та повернення відповідних HTTP-статусів (таких як Ok() або BadRequest()), що відповідає стандартам REST API. Інтеграція webhook із Middleware у ASP.NET Core дозволяє централізовано обробляти вхідні запити перед передачею їх контролерам, забезпечуючи додатковий рівень безпеки та логування. Таким чином, webhook у проєкті не лише полегшує взаємодію з Telegram, але й слугує основою для майбутнього розширення функціональності, а саме, додавання нових команд чи інтеграції з іншими зовнішніми сервісами.

2.5.3 **Ngrok**

Ngrok – це інструмент, який створює тимчасовий публічний URL для тестування локального сервера. У проєкті ngrok використовувався для налаштування доступу до вебхука Telegram, дозволяючи приймати запити від зовнішнього сервісу на локальному середовищі розробки. Це полегшує тестування без необхідності розгортання на віддаленому сервері.

Реалізація ngrok у проєкті включала кілька ключових кроків:

1. Встановлення та запуск: Після встановлення ngrok (через завантаження з офіційного сайту чи npm), інструмент запускався командою ngrok http 5001 у командному рядку, де 5001 — порт локального сервера ASP.NET Core. Це

генерувало унікальний публічний URL (до прикладу, https://abcd1234.ngrok.io), який можна було використати для зв’язку з зовнішнім світом.

2. Налаштування вебхука: За допомогою Telegram Bot API метод SetWebhook був викликаний із сгенерованим URL (а саме, https://abcd1234.ngrok.io/api/telegram), щоб перенаправити оновлення від Telegram до TelegramWebhookController. Це дозволило тестувати обробку повідомлень і команд бота в реальному часі.

3. Безпека та логування: Ngrok підтримує шифрування (HTTPS), що забезпечує безпечну передачу даних, а також надає детальні логи в консолі, які допомагали відстежувати вхідні запити, помилки чи затримки, що було корисним для діагностики проблем із вебхуком.

4. Динамічність: Оскільки URL ngrok є тимчасовим і змінюється при кожному новому запуску, у проєкті потрібно було регулярно оновлювати вебхук у Telegram API, що вимагало автоматизації цього процесу (скажімо, через скрипт) для зручності.

Цей підхід значно спростив тестування інтеграції з Telegram, дозволяючи розробнику перевіряти асинхронну обробку оновлень (а саме, через HandleUpdate у TelegramBotService) та взаємодію з базою даних SQLite (до прикладу, збереження стану користувача UserState) без необхідності розгортання на хмарному сервері, як-от Azure чи AWS. Ngrok також підтримує додаткові функції, такі як інспекція трафіку (за допомогою веб-інтерфейсу на http://localhost:4040), що дозволяло аналізувати надіслані запити та відповіді, допомагаючи виявляти некоректну поведінку додатку. Завдяки цій гнучкості ngrok став незамінним інструментом для імітації реальних умов роботи вебхука під час розробки та відлагодження.

3 ПРОЄКТУВАННЯ

3.1 **Проєктування архітектури web-застосунку**

Web-застосунок "Лідерська Ініціатива" реалізований за моделлю клієнт-сервер, що забезпечує розподіл обробки даних між серверною та клієнтською частинами. Така архітектура сприяє підвищенню масштабовності, ефективності та зручності розробки та обслуговування системи.

Клієнтська частина (фронтенд) відповідає за взаємодію з користувачем через веб-інтерфейс, створений за допомогою HTML, CSS і JavaScript. Використання бібліотек, таких як Slick Carousel для каруселей зображень і Font Awesome для іконок, додає інтерактивності та візуальної привабливості. Фронтенд відправляє HTTP-запити до сервера, зокрема через асинхронні виклики, і отримує відповіді, які динамічно відображаються на сторінках завдяки інтеграції з серверною логікою через Razor-синтаксис.

Серверна частина (бекенд), побудована на основі ASP.NET Core MVC, реалізує основну бізнес-логіку додатку. Вона обробляє запити від клієнта, взаємодіє з базою даних SQLite через Entity Framework Core для зберігання та вилучення даних (до прикладу, про команду та напрями), і повертає відповіді у вигляді HTML-сторінок або JSON. Сервер також підтримує обробку вебхуків, зокрема для інтеграції з Telegram, що дозволяє приймати зовнішні запити в реальному часі.

Для тестування локального сервера під час розробки використовувався ngrok, який створював тимчасовий публічний URL для доступу до вебхука. Це забезпечило зручність налаштування та відлагодження інтеграції з Telegram без розгортання на віддаленому сервері. На рисунку 3.1 наведено діаграму розгортання клієнт-серверної моделі

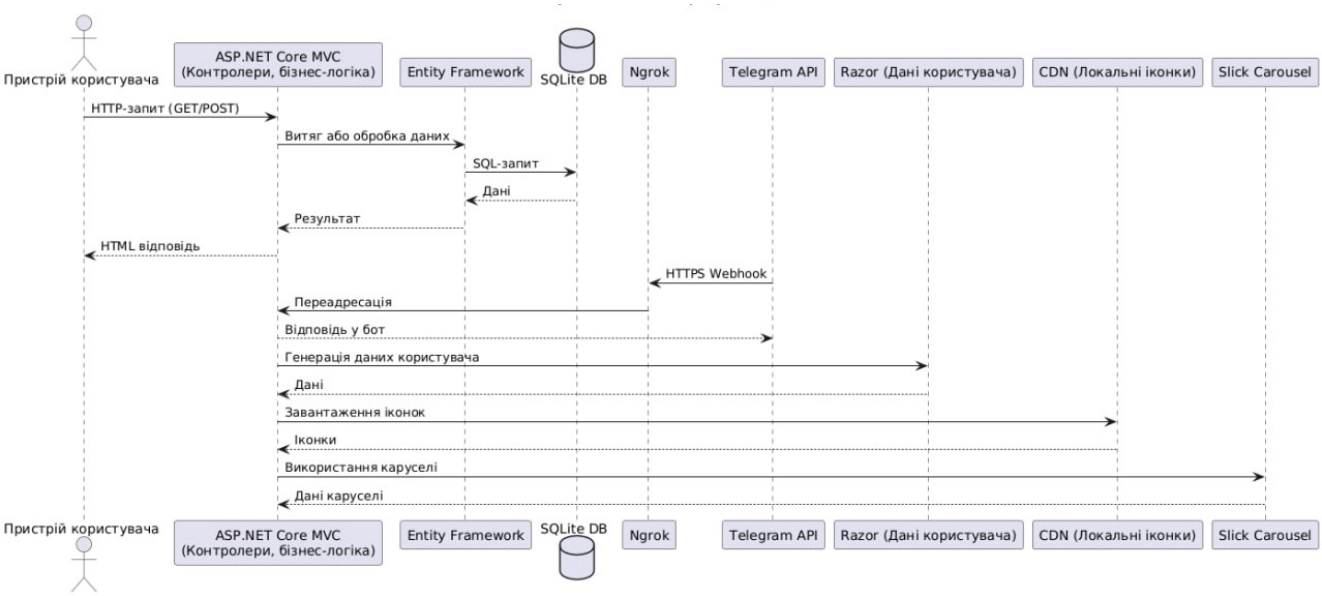


Рис. 3.1 Діаграма розгортання клієнт-серверної моделі

Діаграма показує два основні компоненти:

Клієнт, а саме, браузер користувача, який відображає сторінки (наприклад, карусель із Slick Carousel) і відправляє запити через HTTP.

Сервер, тобто, ASP.NET Core MVC, що обробляє запити, взаємодіє з SQLite через EF Core і підтримує вебхуки через ngrok.

Стрілки вказують на двосторонню взаємодію (запити від клієнта до сервера та відповіді назад), а також на зовнішній зв’язок із Telegram через вебхук.

3.2 **Архітектура клієнтської частини**

Клієнтська частина веб-додатку "Лідерська Ініціатива" відповідає за візуалізацію даних і взаємодію з користувачем через браузер. Вона побудована на основі HTML, CSS і JavaScript, інтегрованих у файли .cshtml через синтаксис Razor. Така архітектура забезпечує динамічне відображення вмісту та інтерактивність, що є ключовими аспектами сучасних веб-додатків.

Основу клієнтської частини становлять представлення (views), створені за допомогою Razor-синтаксису в файлах .cshtml. Ці файли містять HTML-розмітку, стильову інформацію (CSS) і серверний код C#, який генерує динамічний вміст на основі даних із моделі. До прикладу, файл Error.cshtml використовує модель ErrorViewModel для відображення повідомлень про помилки, а файли, такі як

Index.cshtml, можуть включати дані про команду чи напрями, отримані з бази даних.

Для додавання інтерактивності та візуальних елементів використовуються JavaScript і зовнішні бібліотеки:

Slick Carousel – бібліотека, підключена через CDN (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick-theme.min.css" />), яка реалізує каруселі зображень. Вона дозволяє створювати слайдери для відображення фотографій, таких як, у розділі про команду, забезпечуючи плавну анімацію та навігацію.

Font Awesome – бібліотека іконок (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css" />), яка додає стилізовані іконки до інтерфейсу, до прикладу, для кнопок чи декоративних елементів.

Структура клієнтської частини базується на модульному підході, де кожен .cshtml файл відповідає конкретному представленню (таких як головна сторінка, сторінка помилки, сторінка новин). JavaScript-код, якщо використовується локально, може бути розміщений у окремих .js файлах або вбудований у .cshtml для обробки подій (до прикладу, кліків по кнопках чи оновлення вмісту). Ця організація дозволяє легко підтримувати та розширювати інтерфейс.

Перевагою такої архітектури є її інтеграція з серверною частиною через Razor, що дозволяє динамічно генерувати сторінки на основі даних із сервера. Використання зовнішніх бібліотек (Slick Carousel і Font Awesome) забезпечує швидке додавання функціональності без необхідності написання власного коду з нуля. Однак варто зазначити, що клієнтська логіка обмежена порівняно з повноцінними фронтенд-фреймворками (таких як Angular), оскільки основна обробка даних виконується на сервері. На рисунку 3.2 наведено діаграму структури клієнтської частини.

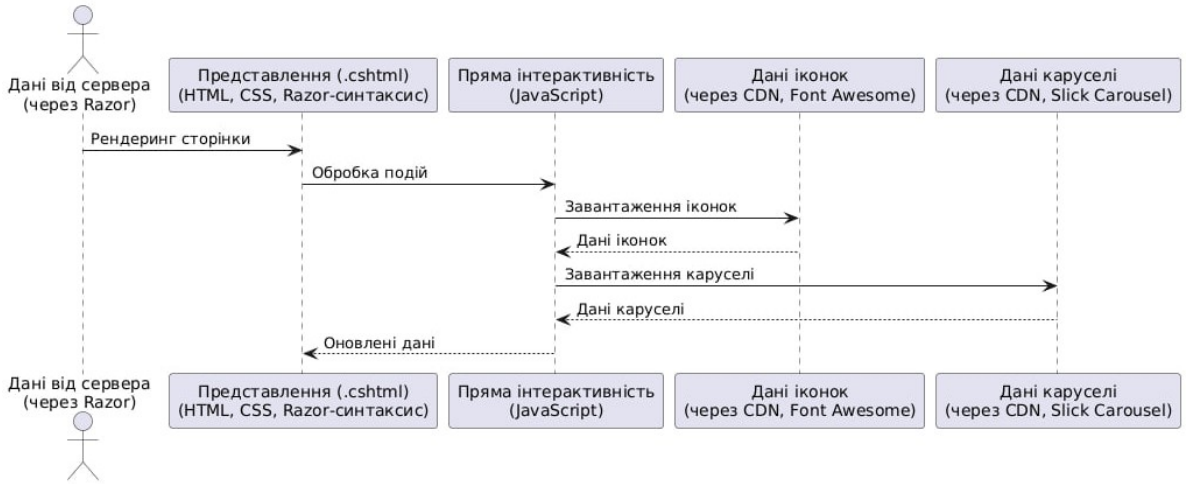


Рис. 3.2 Діаграма структури клієнтської частини

Діаграма відображає:

-представлення (.cshtml) – основний блок із html, css і razor-синтаксисом;

-javascript – шар для обробки подій і взаємодії;

-бібліотеки – slick carousel і font awesome, підключені через cdn, із стрілками до відповідних елементів (до прикладу, карусель до зображень).

Стрілки показують, як дані від сервера (через Razor) передаються до HTML, а JavaScript і бібліотеки додають інтерактивність.

3.3 **Архітектура серверної частини**

Серверна частина додатку "Лідерська Ініціатива" побудована на основі фреймворку ASP.NET Core MVC, який забезпечує чітку організацію коду за шаблоном Model-View-Controller (MVC). Ця архітектура дозволяє розділити логіку обробки даних, бізнес-логіку та генерацію представлень, що сприяє модульності, легкості підтримки та масштабованості проєкту.

ASP.NET Core MVC складається з кількох ключових компонентів, які взаємодіють між собою:

-моделі (models), які визначають структуру даних додатку та взаємодіють із базою даних через entity framework core. як приклад, у проєкті використовуються моделі, такі як teammembers і directions, які мапуються на таблиці в базі даних sqlite. ef core забезпечує абстракцію для виконання операцій crud (створення, читання, оновлення, видалення) без написання складних sql-запитів.

-контролери (controllers), які обробляють http-запити від клієнта, викликають відповідну бізнес-логіку та повертають результат у вигляді представлення або json-даних. у проєкті використано контролери, такі як homecontroller, який відповідає за відображення головної сторінки, newscontroller для роботи з новинами, та errorcontroller для обробки помилок. до прикладу, метод index у homecontroller отримує дані через ef core і передає їх у представлення;

-представлення (views), які генеруються на основі файлів .cshtml із використанням razor-синтаксису. вони отримують дані від контролерів через моделі та відображають їх у вигляді html-сторінок. у проєкті це включає такі сторінки, як index.cshtml для головної сторінки та error.cshtml для відображення повідомлень про помилки;

-middleware, які asp.net core використовує для обробки запитів на різних етапах їхнього життєвого циклу. у проєкті реалізовано кастомний errorhandlingmiddleware, який перехоплює виключення, логує їх у консоль для розробників і викликає errorcontroller для відображення сторінки помилки користувачу. middleware також використовується для перенаправлення запитів (до прикладу, із "/" на "/home");

-інтеграція з вебхуками, а саме, серверна частина яка підтримує обробку вебхуків для інтеграції з telegram. у program.cs ініціалізується telegrambotservice, який налаштовує вебхук для прийому повідомлень від telegram через endpoint /api/telegram. це дозволяє додатку взаємодіяти з користувачами через telegram-бот, отримуючи та обробляючи запити в реальному часі.

Архітектура ASP.NET Core MVC у проєкті побудована з урахуванням принципів модульності та повторного використання коду. Наприклад, маршрутизація, визначена в Program.cs, дозволяє легко налаштувати доступ до різних контролерів (до прикладу, home для головної сторінки, news для новин). Використання асинхронного програмування (з ключовими словами async і await) підвищує продуктивність, особливо під час роботи з базою даних через EF Core.

Перевагами такої архітектури є:

-модульність, що допомогає розділенню коду на моделі, контролери та представлення спрощує підтримку та розширення додатку;

-гнучкість, а саме, middleware, який дозволяє легко додавати нові функції, такі як обробка помилок чи інтеграція з вебхуками;

-продуктивність, а саме, асинхронна обробка запитів і ефективна взаємодія з базою даних через ef core забезпечують швидкий відгук додатку.

На рисунку 3.3 наведено діаграму послідовності.

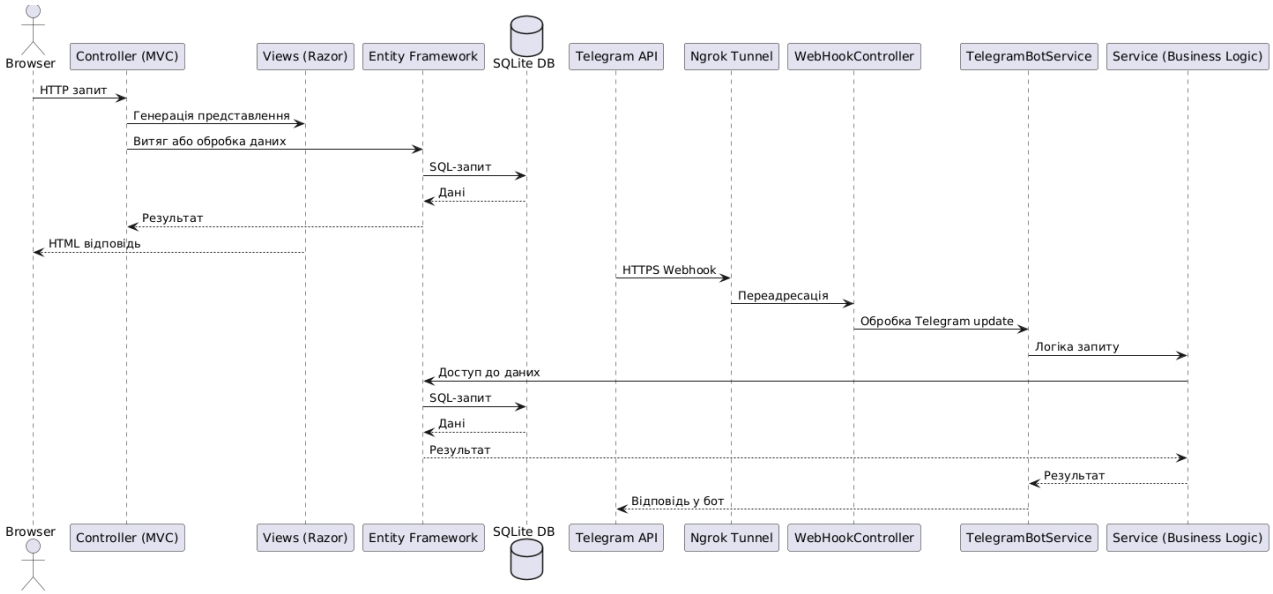


Рис. 3.3 Діаграма послідовності

Діаграма відображає:

- веб-інтерфейс (web-потік) ;

- інтеграція з telegram (telegram-потік).

3.3.1 **Веб-інтерфейс(Web-потік)**

Цей потік описує, як користувач взаємодіє з веб-додатком через браузер:

1. Browser — користувач ініціює HTTP-запит до веб-сайту.

2. MvcController (контролер) — отримує запит.

3. Надсилає запит до Entity Framework для взаємодії з базою даних (витяг чи оновлення інформації).

4. Передає дані до представлення (Views), яке формує HTML-сторінку.

5. Entity Framework (EF) — ORM, який формує SQL-запити на основі запитів контролера.

6. SQLite DB — база даних, яка обробляє SQL-запити й повертає результати.

7. Views (Razor) — отримує дані від контролера і створює HTML-сторінку.

8. Browser — отримує HTML-відповідь і показує її користувачеві.

Примітка: В даній архітектурі відсутній окремий сервісний шар (Business Logic), оскільки уся логіка реалізована безпосередньо у відповідних контролерах MVC.

3.3.2 **Telegram-потік (Telegram інтеграція)**

Цей потік описує, як система взаємодіє з користувачами Telegram через бота:

1. Telegram API — надсилає webhook-запит (оновлення повідомлень) через HTTPS.

2. Ngrok Tunnel — тунелює запит із зовнішнього Telegram-сервера до локального сервера розробника.

3. WebHookController — приймає вхідний webhook-запит і передає оновлення Telegram далі.

4. TelegramBotService — обробляє повідомлення користувача згідно з логікою бота.

5. TelegramBotService — формує відповідь користувачу.

6. Telegram API — отримує відповідь бота і надсилає її користувачу.

3.4 **Архітектура бази даних**

Архітектура бази даних проєкту "Лідерська Ініціатива" базується на реляційній моделі, реалізованій за допомогою SQLite – легкої вбудованої системи управління базами даних, яка ідеально підходить для невеликих і середніх проєктів. SQLite була обрана завдяки своїй простоті, компактності, відсутності необхідності в окремому сервері та достатній функціональності для зберігання даних про напрями, команду, новини, запити на співпрацю та організаційні деталі. Ця СУБД забезпечує швидкий доступ до даних, мінімальне використання ресурсів і легкість інтеграції в .NET-додатки, що робить її оптимальним вибором для веб-додатку з динамічним вмістом.

Доступ до бази даних здійснюється через Entity Framework Core (EF Core), який виконує роль об’єктно-реляційного мапера (ORM), спрощуючи взаємодію між

об’єктами C# і таблицями бази даних. EF Core дозволяє розробникам працювати з даними на рівні коду, використовуючи моделі (таких як, Direction, TeamMember, JoinOrganizationRequest), які автоматично відображаються на відповідні таблиці в SQLite. У проєкті EF Core забезпечує створення та управління структурою бази даних через механізм міграцій, що дозволяє автоматично генерувати SQL-запити для створення або оновлення таблиць при зміні моделей. До прикладу, додавання нового поля до моделі News автоматично відображається у вигляді нової колонки в таблиці News після застосування міграції.

3.4.1 **Структура таблиць**

База даних складається з кількох таблиць, які відповідають моделям, визначеним у проєкті. Ось їхній опис:

Directions – це таблиця зберігає інформацію про напрями діяльності. Вона містить поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

Name (строка, до 100 символів, обов’язкове) – назва напряму (як приклад, "Лідер.Junior").

Description (строка, до 500 символів) – опис напряму.

Початкові дані включають три напрями: "Лідер.Junior", "Лідер.Student" і "Лідер.Volunteer".

TeamMembers – це таблиця містить дані про членів команди. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

FirstName (строка, до 50 символів, обов’язкове) – ім’я члена команди.

LastName (строка, до 50 символів, обов’язкове) – прізвище.

Description (строка, до 500 символів) – опис ролі чи внеску.

Початкові дані включають 14 членів команди з їхніми іменами, прізвищами та ролями (до прикладу, "Даніїл Буднік" – менеджер "Лідер.Volunteer").

News – це таблиця для зберігання новин. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

Title (строка, до 200 символів, обов’язкове) – заголовок новини.

Description (строка, до 500 символів, обов’язкове) – текст новини.

OrganizationDetails: Таблиця для деталей організації. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

Currency (строка, до 50 символів, обов’язкове) – валюта (наприклад, "UAH" або "USD").

Recipient (строка, до 200 символів, обов’язкове) – отримувач платежу.

Edrpou (строка, до 50 символів) – код ЄДРПОУ (може бути null).

BankName (строка, до 200 символів) – назва банку.

CorrespondentBankSwiftCode (строка, до 50 символів) – код SWIFT кореспондентського банку (може бути null).

AccountNumber (строка, до 50 символів) – номер рахунку.

PurposeOfPayment (строка, до 200 символів) – мета платежу.

Початкові дані включають два записи: один для UAH і один для USD із деталями для "ПриватБанку".

CooperationRequestEntity: Таблиця для заявок на співпрацю. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

Surname, FirstName (строка, до 50 символів, обов’язкове) – прізвище та ім’я заявника.

PhoneNumber (строка, до 20 символів, обов’язкове) – номер телефону.

TelegramHandle, Organization (строка, до 50/100 символів) – нік у Telegram і назва організації (може бути null).

CooperationFormats (список строк) – формати співпраці.

ProposalDescription (строка, до 500 символів, обов’язкове) – опис пропозиції.

RequiresFunding (логічне значення) – чи потрібне фінансування.

AdditionalInfo (строка, до 500 символів) – додаткова інформація (може бути null).

SubmissionDate (дата і час) – дата подачі заявки.

JoinOrganizationRequest: Таблиця для заявок на вступ до організації. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

LastName, FirstName (строка, до 100 символів, обов’язкове) – прізвище та ім’я.

PhoneNumber (строка, до 16 символів, обов’язкове) – номер телефону з валідацією формату ("+380631960172").

TelegramNickname, Location (строка, до 50/100 символів) – нік у Telegram і місце проживання (може бути null).

MembershipType, InterestedAreas, WeeklyTimeCommitment, PreviousExperience, PreferredEngagement (строка, до 20/500 символів, обов’язкове) – тип членства, напрями інтересів, часовий внесок, досвід і форма залучення.

PreviousExperienceDescription (строка, до 500 символів) – опис досвіду (може бути null).

AgreesToStatute, ConsentToDataProcessing (логічне значення, обов’язкове) – згода зі статутом і обробкою даних.

ApplicationDate (дата і час) – дата подачі заявки, ініціалізована як поточна.

UserState – це таблиця для стану користувача в процесі взаємодії з Telegram-ботом. Поля:

Id (ключ, ціле число) – унікальний ідентифікатор.

ChatId (ціле число) – ідентифікатор чату в Telegram.

Step (ціле число) – поточний етап діалогу.

Surname, FirstName, PhoneNumber (строка, обов’язкове) – дані користувача.

TelegramHandle, Organization, CooperationFormats, ProposalDescription, AdditionalInfo (строка, може бути null) – додаткові поля.

RequiresFunding (логічне значення, може бути null) – потреба у фінансуванні.

3.4.2 **Використання Entity Framework Core**

Entity Framework Core забезпечує доступ до бази даних через об’єктно-орієнтований підхід. Клас AppDbContext визначає набір DbSet, що відповідає кожній таблиці:

Directions, TeamMembers, News, OrganizationDetails, CooperationRequests, UserStates, JoinOrganizationRequests.

У методі OnModelCreating налаштовуються початкові дані (seed data) для таблиць Directions, TeamMembers і OrganizationDetails. До прикладу, таблиця Directions заповнюється трьома напрямами, а TeamMembers – 14 членами команди.

Це дозволяє запускати проєкт із попередньо підготовленими даними без ручного введення.

EF Core також підтримує асинхронні операції (до прикладу, ToListAsync), що підвищує продуктивність при отриманні даних із бази. Валідація полів (таких як Required, StringLength, регулярні вирази для PhoneNumber у JoinOrganizationRequest) виконується на рівні моделі, забезпечуючи цілісність даних.

3.5 **Відносини між таблицями**

У поточній реалізації явно не визначено зовнішні ключі чи відносини між таблицями (наприклад, зв’язок між TeamMembers і Directions). Однак можливі логічні відносини:

Один до багатьох: Один напрям (Direction) може бути пов’язаний із кількома членами команди (TeamMembers) через їхні ролі, хоча це не реалізовано в моделях.

Багато до багатьох: Запити на співпрацю (CooperationRequestEntity) можуть бути пов’язані з кількома напрямами, але це реалізовано через список CooperationFormats як текстовий рядок, а не як окрему таблицю зв’язків.

Для повноцінної реляційної моделі можна було б додати таблицю зв’язків (а саме, TeamMemberDirection), але в даному проєкті пріоритет віддано простоті через SQLite і EF Core.

3.6 **Переваги архітектури**

Цілісність даних, а саме, валідація на рівні моделей і початкові дані гарантують коректність введення, забезпечуючи високу якість даних у базі SQLite проєкту "Лідерська Ініціатива". Використання атрибутів валідації, таких як [Required], [StringLength] і [RegularExpression], у моделях (як приклад, JoinOrganizationRequest із перевіркою формату телефону) дозволяє перевіряти введені користувачем дані на стороні сервера перед їх збереженням у базі. Крім того, початкові дані, такі як зразки напрямків (Direction) чи членів команди (TeamMember), завантажуються через seed-методи в EF Core, що забезпечує

базовий набір коректних записів і полегшує тестування. Такий підхід мінімізує ризик дублювання чи некоректних даних, а також підтримує цілісність через унікальні обмеження та зв’язки між таблицями.

Простота, до прикладу, SQLite не потребує окремого серверного процесу, що значно полегшує розгортання та підтримку проєкту. На відміну від традиційних СУБД, таких як MySQL чи PostgreSQL, SQLite працює як файлова база даних, що інтегрується безпосередньо в web-застосунок, що робить його ідеальним для локального середовища розробки та невеликих веб-додатків. У проєкті це дозволяє швидко налаштувати базу даних на локальному комп’ютері чи сервері без складної конфігурації серверного програмного забезпечення, а також спрощує перенесення даних між середовищами (до прикладу, із розробки на продакшн) через єдиний файл бази (а саме, leadershipinitiative.db). Така архітектура знижує витрати на ресурси та спрощує адміністрування.

Гнучкість, а саме, EF Core дозволяє легко оновлювати схему бази даних через міграції, що забезпечує адаптивність архітектури проєкту до змін у вимогах. У процесі розробки "Лідерської Ініціативи" міграції використовувалися для додавання нових полів (таких як, AdditionalInfo у CooperationRequestEntity) чи таблиць (таких як UserState) без ручного написання SQL-запитів. Процес включає створення міграції командою dotnet ef migrations add <MigrationName> і її застосування через dotnet ef database update, що автоматизує оновлення схеми бази даних. Ця гнучкість дозволяє швидко реагувати на еволюцію функціональності, наприклад, додавання нових форм чи інтеграцію з Telegram, зберігаючи сумісність із існуючими даними та забезпечуючи плавний перехід між версіями бази.

На рисунку 3.4 наведено діаграму класів бази даних.

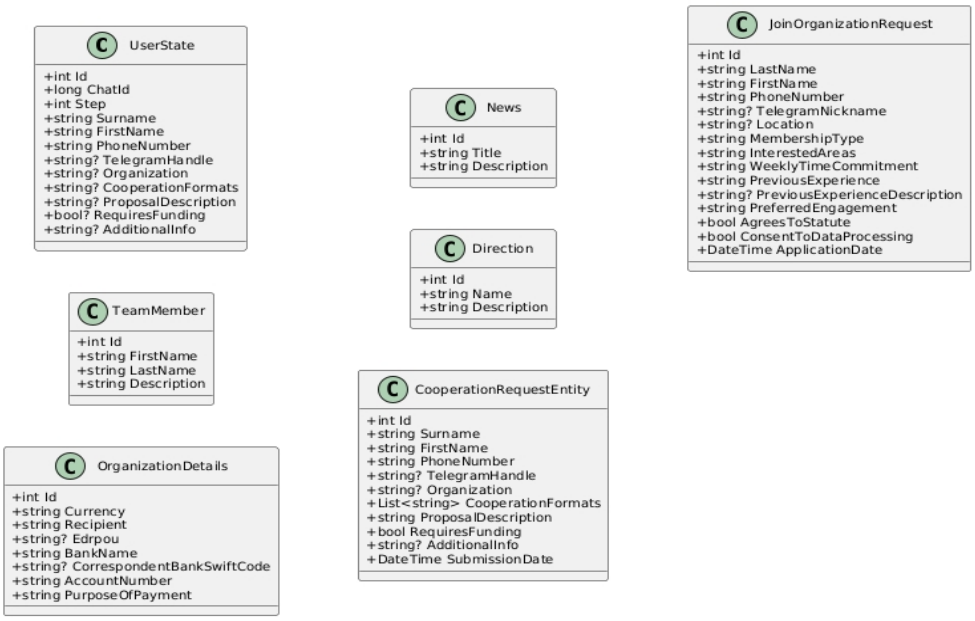


Рис. 3.4 Діаграма класів бази даних

Діаграма відображає:

1. Сім таблиць, таких як Directions, TeamMembers, News, OrganizationDetails, CooperationRequestEntity, JoinOrganizationRequest, UserState.

2. Кожна таблиця зображена як блок із полями (до прикладу, Id, Name для Directions).

3. Початкові дані позначені як окремі записи в таблицях.

Для зображення взаємодії користувача з системою створимо діаграму варіантів використання. На ній зліва зображується актор (в даному випадку – користувач) і система, яка відповідає на певні запроси користувача. На рисунку 3.4 наведено діаграму варіантів використання.

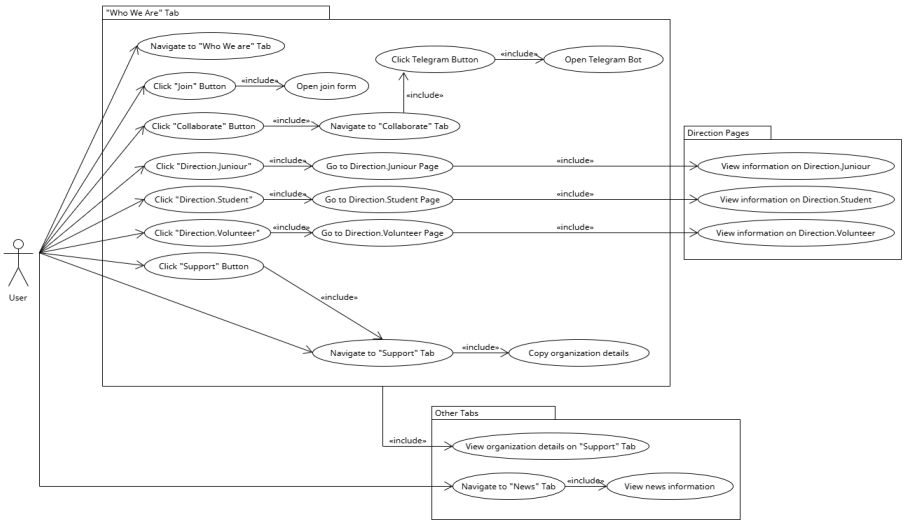


Рис. 3.5 Діаграма варіантів використання

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 **Дизайн інтерфейсу**

Дизайн інтерфейсу веб-додатку "Лідерська Ініціатива" реалізований через файли представлень .cshtml, створені в рамках ASP.NET Core MVC із використанням Razor-синтаксису. Цей підхід дозволяє комбінувати HTML, CSS і серверний код C# для створення динамічних сторінок. Основна структура дизайну визначена в папці Views, де кожен файл відповідає окремому екранному представленню (наприклад, Home/Index.cshtml, Error/Error.cshtml).

Процес дизайну включав:

-створення шаблонів, а саме, у файлах .cshtml визначено базову розмітку (до прикладу, у error.cshtml) з інтеграцією стилів css, підключених через wwwroot/css/site.css;

-додавання інтерактивності, тобто, використання javascript і бібліотек, таких як slick carousel (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick-theme.min.css" />) для каруселей зображень і font awesome (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css" />) для іконок, додає візуальної привабливості та функціональності;

-адаптивність, до прикладу, стилі в site.css і розміри елементів у .cshtml підлаштовуються під різні розміри екранів, забезпечуючи коректне відображення на десктопах і мобільних пристроях.

Дизайн орієнтований на простоту й інтуїтивність, з акцентом на відображення даних (а саме, команд і новин) і обробку форм (наприклад, заявок на співпрацю).

На рисунку 4.1 , 4.2 наведено приклад головної сторінки та екранна форма сторінки помилки.

Рис. 4.1 Головна сторінка

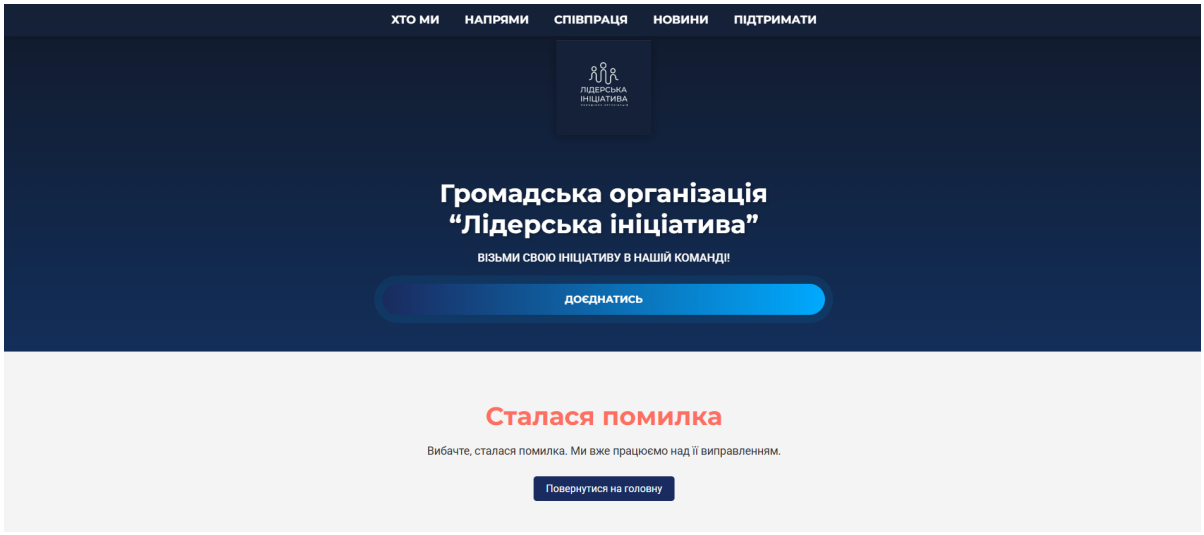


Рис. 4.2 Екранна форма сторінки помилки

4.2 **Реалізація клієнтської частини**

Реалізація клієнтської частини веб-додатку "Лідерська Ініціатива" базується на технології ASP.NET Core MVC із використанням Razor для рендерингу динамічних веб-сторінок. Цей підхід дозволяє ефективно поєднувати серверну логіку з HTML-розміткою, забезпечуючи зручне створення адаптивного і інтерактивного інтерфейсу для підтримки молодіжних ініціатив. Процес розробки клієнтської частини включає кілька ключових етапів, кожен із яких відіграє важливу роль у формуванні функціонального й естетично привабливого веб-

додатку.

4.2.1 **Ініціалізація проєкту**

Першим кроком стало створення проєкту в середовищі розробки Visual Studio, вибравши шаблон ASP.NET Core Web Application із архітектурою MVC (Model-View-Controller). Цей шаблон забезпечує готову структуру, яка полегшує організацію коду та інтеграцію різних компонентів. Проєкт ініціалізовано за допомогою команди dotnet new mvc у терміналі, що дозволило автоматично згенерувати базову конфігурацію, включаючи файли Startup.cs і Program.cs для конфігурації сервера та маршрутизації.

4.2.2 **Структура каталогів**

Структура каталогів, відображена в Solution Explorer, має чіткий поділ на ключові компоненти:

-/views/ – це каталог, що містить файли представлень (view), створені за допомогою razor. до прикладу, файли index.cshtml, error.cshtml, news.cshtml, directions.cshtml, support.cshtml, cooperation.cshtml відповідають за відображення відповідних сторінок веб-додатку. ці файли динамічно генерують html на основі даних, отриманих від контролерів.

- /wwwroot/ – це папка для статичних файлів, таких як стилі (css), скрипти (javascript), зображення та шрифти. наприклад, тут зберігаються файли site.css, які містять кастомні стилі, і зображення для каруселі чи логотипу "лідерської ініціативи";

- /models/ – це каталог із класами моделей даних, які представляють структуру даних, що передаються між контролерами та представленнями. до прикладу, клас teammember містить властивості, такі як name, position і photourl, а клас direction – title, description і status, що використовуються для відображення інформації про команду та напрями діяльності;

- /controllers/, який містить класи контролерів, які обробляють http-запити та повертають відповідні представлення. наприклад, homecontroller управляє головною сторінкою (index), а errorcontroller – сторінкою помилок.

Ця структура забезпечує модульність і легкість розширення проєкту, дозволяючи

додавати нові сторінки чи функціонал без значних змін у існуючому коді.

На рисунку 4.3, 4.4 зображено структуру проєкту.

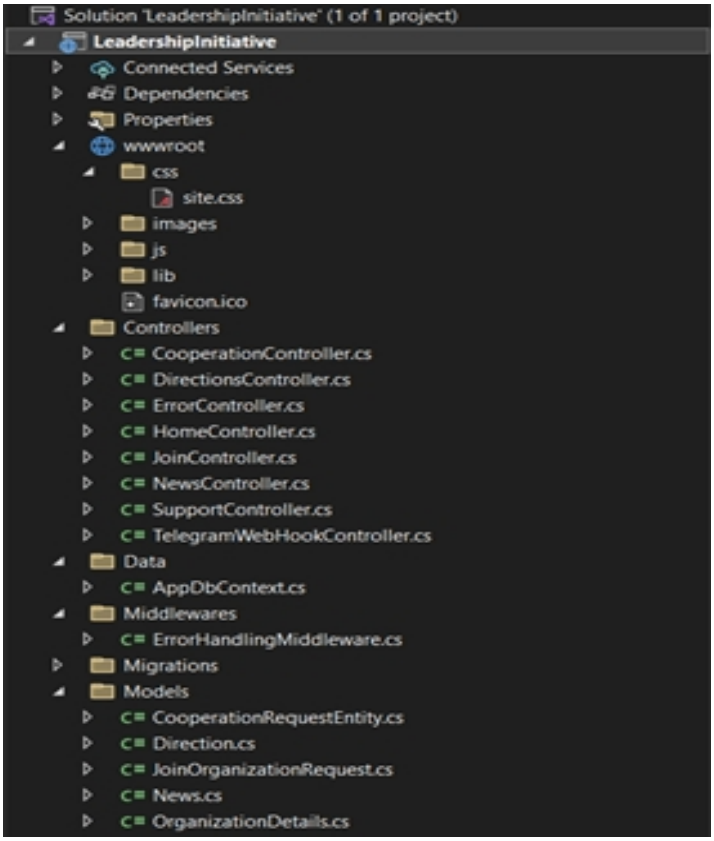


Рис. 4.3 Структура проєкту

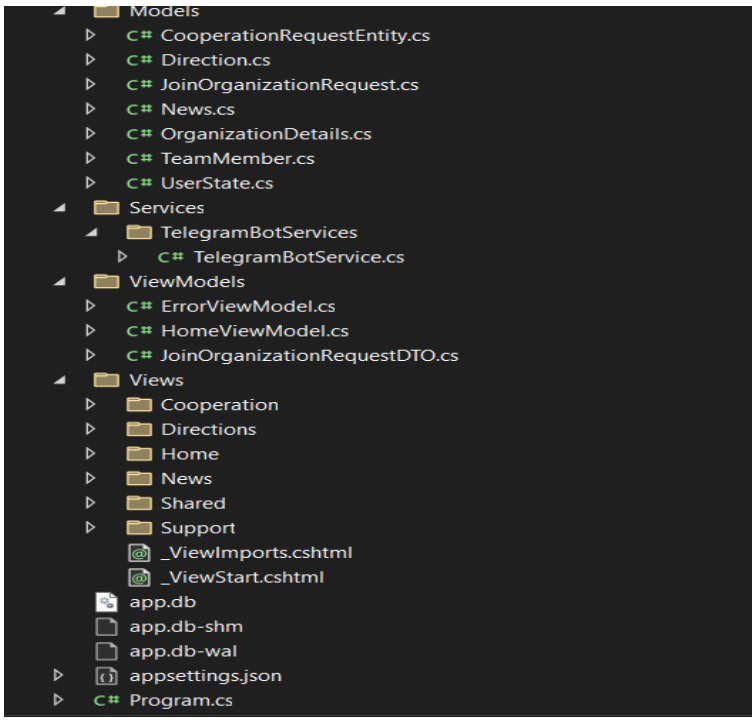


Рис. 4.4 Структура проєкту

4.2.3 **Створення представлень**

Розробка представлень у веб-додатку "Лідерська Ініціатива" здійснюється шляхом створення файлів із розширенням .cshtml, кожен із яких відповідає за відображення окремої сторінки або її частини. Ці файли використовують Razor-синтаксис, який дозволяє вбудовувати C#-код безпосередньо в HTML-розмітку, забезпечуючи динамічне відображення даних, отриманих із моделей через контролери. Razor полегшує створення інтерактивних і адаптивних сторінок, дозволяючи поєднувати статичний HTML із динамічними елементами, такими як цикли, умови та змінні. У проєкті "Лідерська Ініціатива" представлення відіграють ключову роль у формуванні користувацького інтерфейсу, забезпечуючи зручний доступ до інформації про організацію, її напрями, команду, а також можливості для взаємодії, таких як, подання заявок на співпрацю чи волонтерство.

4.2.4 **Додавання стилів і логіки**

Стилізація клієнтської частини виконується через CSS-файли, розміщені в /wwwroot/css/, а логіка додається за допомогою JavaScript, інтегрованого у .cshtml або підключеного як окремий файл.

Основні стилі, які визначені в wwwroot/css/site.css, де застосовуються адаптивні дизайни для забезпечення коректного відображення на різних пристроях. До прикладу:

-.error-container { max-width: 800px; margin: 0 auto; padding: 20px; text-align: center; background-color: #f8d7da; border-radius: 5px; } @media (max-width: 600px) { .error-container { padding: 10px; font-size: 14px; } }

Цей код забезпечує, що контейнер помилок має фіксовану максимальну ширину, центрований вирівнювання та адаптивність для екранів шириною менше 600 пікселів, що важливо для мобільних користувачів, які можуть переглядати сторінку помилки.

Щодо JavaScript, логіка обробки подій може бути вбудована безпосередньо в .cshtml. Наприклад, у Home/Index.cshtml JavaScript ініціалізує карусель Slick Carousel, як показано вище, або обробляє кліки на кнопках. Альтернативно,

скрипти можна розмістити в окремому файлі (до прикладу, wwwroot/js/site.js) і підключити через <script src="~/js/site.js"></script> у _Layout.cshtml.

4.2.5 **Тестування клієнтської частини**

Тестування клієнтської частини проводилося для забезпечення коректної роботи всіх елементів інтерфейсу та їхньої адаптивності. Процес включав наступні кроки:

1. Запуск проєкту**:** Проєкт запускався за допомогою команди dotnet run у терміналі, що відкриває веб-сервер на адресі http://localhost:5000. Це дозволило перевірити відображення всіх сторінок у браузері.

2. Перевірка каруселі**:** Функціональність Slick Carousel тестувалася вручну, переконуючись, що слайдер фотографій команди на сторінці Team.cshtml автоматично прокручується, а на мобільних пристроях перемикається на один слайд із коректним відступом.

3. Адаптивність**:** Перевірялася коректність відображення сторінок на різних розмірах екранів (десктоп, планшет, смартфон) через інструменти розробника браузера (до прикладу, DevTools у Chrome). Наприклад, переконувалися, що меню розгортається як кнопка-гамбургер на екранах менше 768 пікселів, а текст і зображення масштабовані належним чином.

4. Функціональність форм**:** На сторінках, таких як Index.cshtml чи Directions.cshtml, тестувалися форми подання заявок, перевіряючи, чи коректно відображаються поля введення та чи активуються валідаційні повідомлення при неправильному введенні даних.

5. Обробка помилок**:** Сторінка Error.cshtml тестувалася шляхом штучного створення помилок (до прикладу, звернення до неіснуючого маршруту), щоб переконатися, що повідомлення про помилку відображаються чітко і з відповідним ідентифікатором запиту.

Тестування проводилося ітеративно, із внесенням змін у стилі чи JavaScript на основі виявлених проблем. В ситуації, якщо карусель не завантажувалася коректно, перевірялися шляхи до зображень у моделі або коректність ініціалізації скриптів.

На рисунку 4.5, 4.6 зображено структуру проєкту в Solution Explorer.

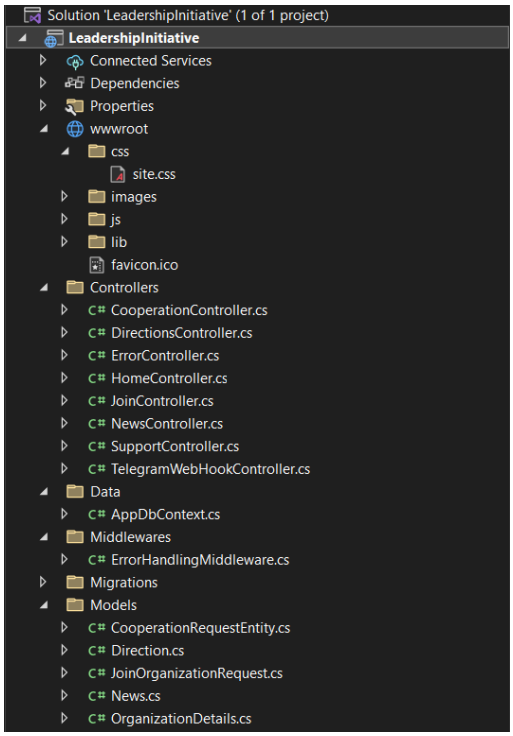


Рис. 4.5 Структура проєкту в Solution Explorer

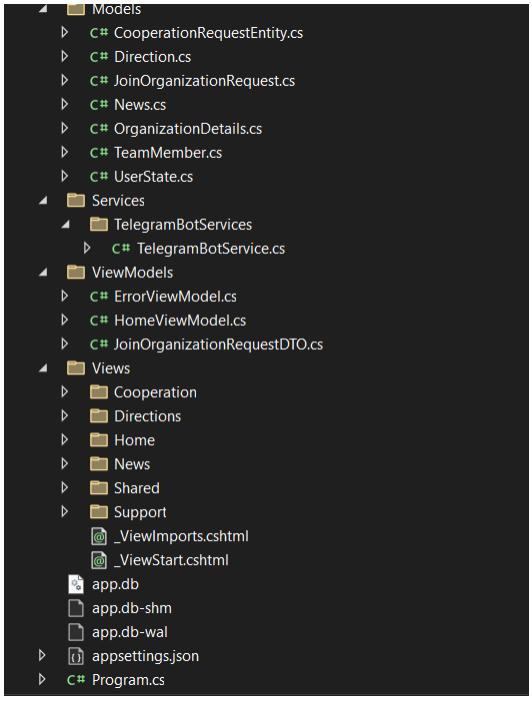


Рис. 4.6 Структура проєкту в Solution Explorer

4.3 **Реалізація серверної частини**

Серверна частина веб-додатку "Лідерська Ініціатива" розроблена на основі фреймворку ASP.NET Core MVC, який забезпечує модульну архітектуру, високу продуктивність і зручність інтеграції з різними технологіями, такими як Entity Framework Core (EF Core) для роботи з базою даних і Telegram Bot API для обробки вебхуків. ASP.NET Core MVC було обрано через його підтримку патерну Model-View-Controller, що дозволяє чітко розділити логіку обробки даних, бізнес-логіку та відображення даних, а також через кросплатформність і вбудовані інструменти для створення безпечних і масштабованих веб-додатків. Реалізація серверної частини включає кілька ключових етапів, кожен із яких детально описаний нижче.

4.3.1 **Ініціалізація проєкту**

Розробка розпочалася зі створення нового проєкту в середовищі Visual Studio 2022, використовуючи шаблон ASP.NET Core Web Application із архітектурою MVC. Цей шаблон автоматично генерує базову структуру проєкту, включаючи необхідні файли та налаштування для запуску веб-додатку. Проєкт також можна створити через командний рядок за допомогою команди dotnet new mvc -o LeadershipInitiative.

Ця команда створює новий проєкт у папці LeadershipInitiative, ініціалізуючи базові каталоги, такі як /Controllers, /Views, /Models, /wwwroot, а також основні файли конфігурації, такі як Program.cs і appsettings.json.

Файл Program.cs: У цьому файлі налаштовується основний пайплайн обробки запитів (request pipeline) і додаються сервіси, необхідні для роботи додатку. Наприклад, підключення до бази даних SQLite, ін’єкція залежностей для контексту бази даних, а також інтеграція з Telegram Bot API.

Файл appsettings.json: Цей файл містить конфігураційні дані, такі як рядок підключення до бази даних і токен для Telegram-бота:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=app.db"
  },
```

```
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
 "TelegramBotToken":
"7830690633:AAGu3NU4TJp6zp4M0P40HohaeVPi9sjV4u0",
  "AppUrl": "https://6176-46-231-230-174.ngrok-free.app"
 }
```

Рядок підключення (Data Source=LeaderInitiative.db) вказує на файл бази даних SQLite, який зберігається в корені проєкту. Логування налаштоване для виведення інформаційних повідомлень, що допомагає відстежувати поведінку додатку під час розробки.

На Рис. 4.7 зображено приклад створеної бази даних app.db, відкритої в інструменті SQLite Browser.

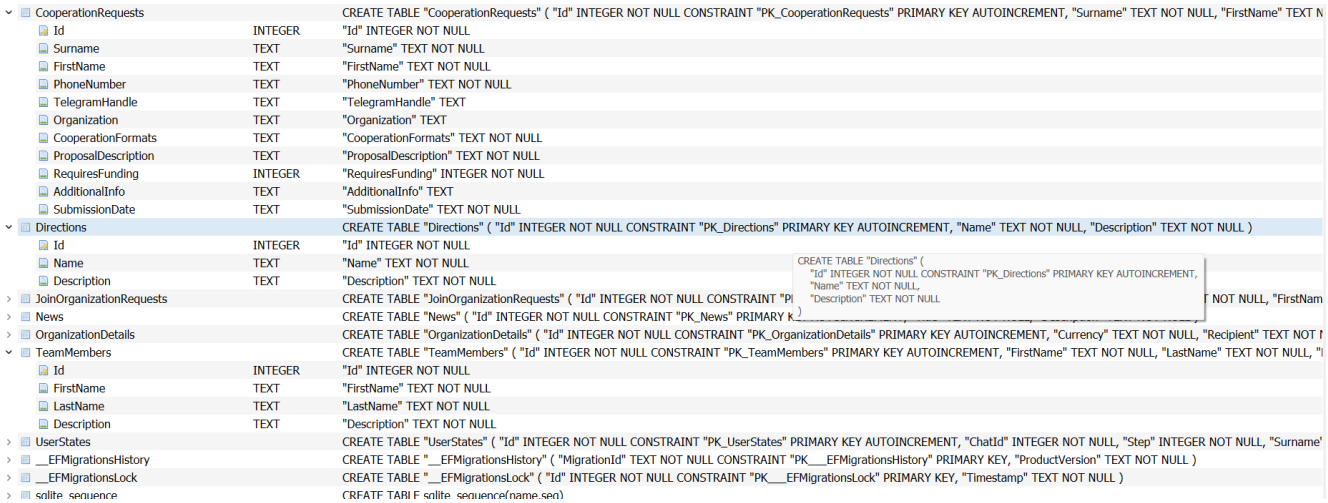


Рис. 4.7 База даних

4.3.2 **Створення моделей і контексту**

Моделі (такі як, Direction, TeamMember) визначено в папці Models. Ці моделі

автоматично мапуються на відповідні таблиці в базі даних завдяки Entity Framework Core.

Клас AppDbContext, розташований у папці /Data, є контекстом бази даних, який налаштовує доступ до SQLite через EF Core.

Метод OnModelCreating дозволяє ініціалізувати базу даних із початковими даними (seeding), що корисно для тестування. Наприклад, додано три напрями діяльності та одного члена команди. EF Core автоматично створює відповідні таблиці при першому запуску додатку або після застосування міграцій через команди:

dotnet ef migrations add InitialCreate

dotnet ef database update

4.3.3 **Реалізація контролерів**

Контролери в проєкті відповідають за обробку HTTP-запитів і повернення відповідних представлень або даних. Вони розташовані в папці /Controllers.

На рис 4.8 зображений контроллер Home, який:

- ініціалізує _contex за допомогою інєкції залежностей;
- створює модель, звертаючись до бази даних;
- передає модель в відповідне представлення home/index.cshtml.

```
public class HomeController : Controller
{
    private readonly AppDbContext _context;

    0 references
    public HomeController(AppDbContext context)
    {
        _context = context;
    }

    [Route("home")]
    0 references
    public async Task<IActionResult> Index()
    {
        var viewModel = new HomeViewModel
        {
            Title = "Лідерська Ініціатива",
            TeamMembers = await _context.TeamMembers.ToListAsync()
        };

        ViewBag.Directions = await _context.Directions.ToListAsync();

        return View(viewModel);
    }

    0 references
    public async Task<IActionResult> Error()
    {
        ViewBag.Directions = await _context.Directions.ToListAsync();
        return View();
    }
}
```

Рис. 4.8 Контроллер Home

4.3.4 **Middleware**

Для обробки помилок на сервері створено кастомний middleware ErrorHandlingMiddleware, розташований у папці /Middlewares. Цей middleware перехоплює виключення, логує їх і перенаправляє користувача на сторінку помилки.

Middleware реєструється в Program.cs:

Логування помилок у консоль допомагає розробникам швидко виявляти проблеми, наприклад, якщо запит до бази даних завершився невдачею через неправильний рядок підключення

4.3.5 **Інтеграція вебхука**

Для інтеграції з Telegram Bot API у проєкті додано сервіс TelegramBotService, який обробляє вхідні вебхуки від Telegram. Реєстрація вебхука виконана в Program.cs. Клас TelegramBotService відповідає за обробку оновлень від Telegram.

4.3.6 **Запуск**

Проєкт запускається за допомогою команди dotnet run у терміналі Visual Studio або командному рядку. Після запуску сервер доступний за адресою http://localhost:5070. У режимі розробки web-застосунок автоматично перезапускається при зміні коду завдяки інструменту dotnet watch. Це значно прискорює розробку, дозволяючи швидко тестувати зміни, до прикладу, у контролерах чи представленнях. При запуску сервер ініціалізує базу даних через EF Core (якщо міграції ще не застосовані), реєструє вебхук для Telegram і починає обробляти запити.

На Рис. 4.9 зображено виглд консолі після запуску проєкту через dotnet run, де видно, що сервер працює на http://localhost:5070, а також ініціалізацію бази даних і вебхука.

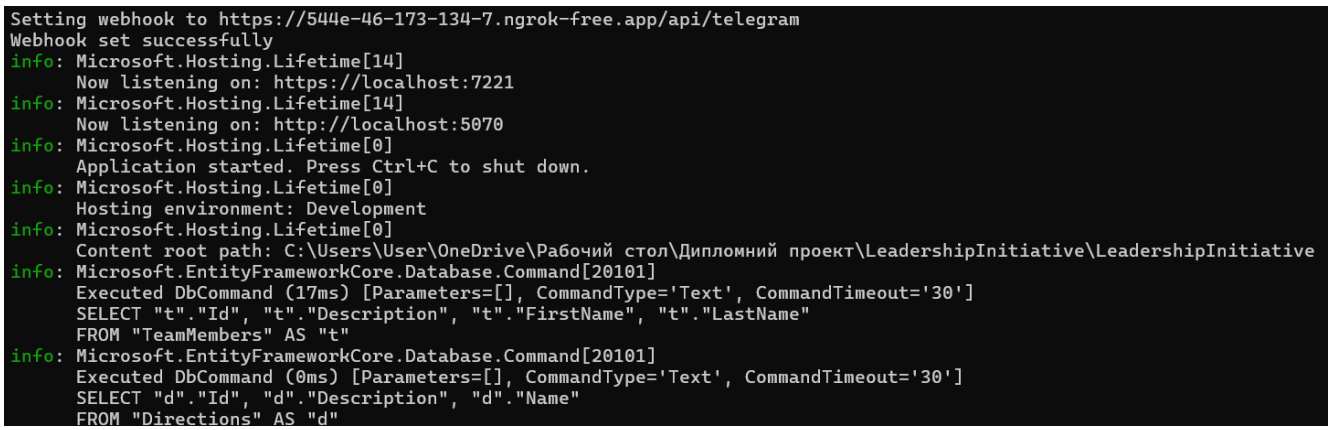


Рис. 4.9 Консоль після запуску проєкту

4.4 **Розгортання проєкту**

Проєкт тестувався локально за допомогою Visual Studio та ngrok для інтеграції з Telegram. Розгортання на віддалений сервер не виконувалося, але проєкт готовий до розгортання на платформах, сумісних із ASP.NET Core (наприклад, Azure).

4.5 **Тестування застосунку**

Тестування веб-додатку "Лідерська Ініціатива" проводилося вручну з метою забезпечення коректної роботи всіх його компонентів, включаючи клієнтську частину, серверну логіку, інтеграцію з базою даних і обробку вебхуків. Ручне тестування було обрано як основний метод через невеликий масштаб проєкту, що дозволяє швидко виявляти помилки та адаптувати процес до змін у функціональності без значних витрат на автоматизацію. Процес тестування включав кілька етапів, кожен із яких спрямований на перевірку специфічних аспектів додатку, а також оцінку його поведінки в реальних умовах використання молодіжною аудиторією.

Перевірка відображення сторінок:

Першим етапом тестування було забезпечення правильного відображення всіх ключових сторінок веб-додатку. Зокрема, перевірялися представлення, такі як Index.cshtml, Error.cshtml, Directions.cshtml, Team.cshtml, News.cshtml і Requisites.cshtml, на предмет коректності відображення даних, отриманих із бази

даних через Entity Framework Core. Наприклад:

- на сторінці index.cshtml тестувалася коректність відображення каруселі членів команди, тексту про організацію та посилань на напрями діяльності. перевірялося, чи зображення (до прикладу, /images/слайд 2.jpeg) завантажуються без помилок, а текст (опис місії) відображається без обрізання;

- на сторінці error.cshtml симулювалися помилки (до прикладу, звернення до неіснуючого маршруту /nonexistentpage), щоб переконатися, що контролер errorcontroller повертає сторінку з правильним повідомленням (а саме, "сторінку не знайдено" для коду 404) і моделлю errorviewmodel. перевірялася також адаптивність макета до різних розмірів екранів, включаючи смартфони та планшети.

Цей етап проводився шляхом запуску додатку через команду dotnet run із доступом до http://localhost:5000 у браузерах Chrome, Firefox і Edge, а також із використанням інструментів розробника (DevTools) для імітації різних роздільних здатностей.

Тестування каруселі Slick Carousel

Другий етап зосередився на перевірці функціональності каруселі Slick Carousel, яка використовується на сторінці Team.cshtml для відображення членів команди. Тестування проводилося на різних розмірах екранів, щоб переконатися в адаптивності та коректній роботі слайдера:

- на десктопних пристроях (роздільна здатність 1920x1080) перевірялася відображення трьох слайдів одночасно з активними стрілками й точками навігації, а також автоматичне прокручування з інтервалом 3000 мс, як задано в JavaScript;

- на планшетах (емульована роздільна здатність 768x1024) тестувалася зміна налаштувань на два слайди, а на смартфонах (емульована роздільна здатність 480x800) – на один слайд, як зазначено в responsive-параметрах:

responsive: [{ breakpoint: 768, settings: { slidesToShow: 2 } }, { breakpoint: 480, settings: { slidesToShow: 1 } }]

Тестування включало штучне створення помилок, таких як видалення зображення з папки /wwwroot/images/ або порушення шляху в моделі, щоб

переконатися, що карусель видає коректне повідомлення про помилку (до прикладу, через alt-текст) і не ламає всю сторінку.

Перевірка обробки вебхука через Ngrok:

Третій етап тестування був присвячений перевірці інтеграції з Telegram Bot API через вебхук, реалізований у TelegramBotService. Для цього використовувався інструмент Ngrok, який створює тимчасовий публічний URL для локального сервера, запущеного на http://localhost:5070.

Процес включав наступні кроки:

- запуск ngrok командою ngrok http 5070, що генерувало url, https://abc123.ngrok.io;
- налаштування вебхука в telegram bot api через виклик setwebhookasync із url /api/telegram, як визначено в program.cs;
- симуляція запитів від telegram шляхом надсилання повідомлень боту (наприклад, команда /start) через клієнт telegram і перевірка відповіді (а саме, "ласкаво просимо до лідерської ініціативи!") на сервері;
- аналіз логів у консолі visual studio для підтвердження обробки оновлень (update) і відсутності помилок.

Тестування включало також перевірку граничних сценаріїв, таких як надсилання некоректних JSON-даних або переривання з’єднання, щоб переконатися, що сервер коректно обробляє ці випадки через middleware ErrorHandlingMiddleware.

На Рис. 4.10 зображено екран тестування вебхука через Ngrok, де видно згенерований URL, вхідні запити від Telegram і відповідь сервера, що підтверджує успішну обробку вебхука.

Рис. 4.10 Форма тестування веб-хука через Ngrok

Переваги ручного тестування

Ручне тестування, застосоване в цьому проєкті, має ряд переваг, які роблять його доречним для "Лідерської Ініціативи" на поточному етапі:

- гнучкість, а саме, розробники можуть швидко адаптувати тести до нових функцій або змін у дизайні, наприклад, додавання нового напряму в карусель або оновлення тексту на сторінці index.cshtml. це дозволяє реагувати на відгуки користувачів чи вимоги адміністраторів у реальному часі;

- швидкість, тобто, для невеликого проєкту з обмеженою кількістю сторінок і функціоналом (перегляд даних, обробка вебхуків) ручне тестування є швидшим і дешевшим, ніж налаштування автоматизованих тестів із фреймворками, такими як xunit чи selenium;

- інтуїтивне розуміння, що дає можливість розробниками безпосередньо оцінити користувацький досвід, до прикладу, чи зручно перемикати слайди в каруселі або чи зрозуміле повідомлення про помилку на error.cshtml;

- можливість виявлення візуальних дефектів, а саме, за допомогою ручного тестування, яке дозволяє легко помітити проблеми з адаптивністю (обрізання тексту на мобільних пристроях) або неправильне відображення зображень, що складно автоматизувати.

Незважаючи на переваги, ручне тестування має певні недоліки, які можуть стати актуальними при масштабуванні проєкту:

- відсутність повторюваності, тобто, кожен тест залежить від дій

тестувальника, що може призводити до пропуску помилок при повторних перевірках;

- обмежена покриття, що говорить про те, що не всі можливі комбінації даних (до прикладу, великі списки членів команди чи помилки в базі даних) можуть бути протестовані вручну;

- залежність від людського фактора, а саме, помилки тестувальника можуть вплинути на якість результатів, наприклад, пропуск перевірки вебхука при слабкому з’єднанні.

Додаткові аспекти тестування:

Окрім основних етапів, проводилися додаткові перевірки:

-тестування форм, а саме, на сторінках, таких як cooperation/index.cshtml, перевірялася коректність обробки форм подання заявок, включаючи валідацію полів (таких як, обов’язковість імені чи email) і збереження даних у базі через контролер;

-продуктивність, тобто, оцінювалася швидкість завантаження сторінок (до прикладу, час відгуку index.cshtml не перевищував 2 секунди) на різних пристроях.

Безпека, до прикладу, перевірялася відсутність витоку даних , щоб перевірити, чи не відображаються деталі помилок користувачам) і коректна обробка неавторизованих запитів.

Роль тестування у проєкті:

Тестування забезпечило високу якість веб-додатку "Лідерська Ініціатива", гарантуючи, що користувачі бачать коректні дані, карусель працює плавно, а вебхук із Telegram інтегрується без збоїв. Ручний підхід дозволив швидко адаптувати продукт до потреб молодіжної аудиторії, наприклад, удосконаливши адаптивність каруселі чи текст повідомлень про помилки.

Таким чином, тестування веб-додатку "Лідерська Ініціатива" шляхом ручного підходу виявилося ефективним для поточних цілей проєкту, хоча в майбутньому, при розширенні функціоналу, може бути доцільним уведення автоматизованих тестів.

ВИСНОВКИ

У процесі виконання дипломної роботи було досягнуто мети – забезпечення ефективної взаємодії волонтерів з організацією "Лідерська Ініціатива" шляхом створення веб-додатку, який оптимізує процеси подання заявок на приєднання, співпрацю та інформування. Об’єктом дослідження став процес управління інформацією та взаємодії в рамках лідерських ініціатив, а предметом – програмне забезпечення для управління даними громадської організації, що дозволило зосередитися на розробці технологічного рішення, яке відповідає сучасним потребам молодіжного середовища.

Для реалізації поставлених завдань було проведено комплексний аналіз існуючих веб-сайтів, які підтримують молодіжні ініціативи, що дозволив визначити сильні та слабкі сторони подібних платформ і сформувати основу для порівняння. Огляд засобів розробки програмного забезпечення клієнт-серверного застосунку виявив оптимальні технології, такі як ASP.NET Core MVC, SQLite, Entity Framework Core, JavaScript і Razor, які забезпечують високу продуктивність, гнучкість і зручність інтеграції. На основі цього проведено формування вимог до веб-додатку, що включало забезпечення інтуїтивно зрозумілого інтерфейсу, адаптивності до різних пристроїв, підтримки асинхронної обробки запитів і інтеграції з зовнішніми сервісами, такими як Telegram.

Спроектування архітектури клієнт-серверного застосунку базувалося на моделі MVC, де серверна частина, реалізована на ASP.NET Core, обробляє запити через контролери, моделі забезпечують структуру даних у SQLite, а клієнтська частина, створена з використанням Razor і JavaScript, генерує інтерактивний інтерфейс. Розробка включала створення ключових компонентів: сторінок для подання заявок (до прикладу, Cooperation/Index.cshtml), каруселі команди (Team.cshtml), інтеграцію вебхуків для сповіщень і middleware для обробки помилок. Тестування застосунку проводилося вручну, що дозволило перевірити

коректність відображення сторінок, функціональність каруселі на різних екранах і обробку вебхуків через Ngrok, підтвердивши стабільність і зручність використання.

Результатом роботи став функціональний web-застосунок "Лідерська Ініціатива", який успішно вирішує поставлені завдання, забезпечуючи зручний доступ до інформації про організацію, спрощуючи процеси подання заявок і покращуючи комунікацію через інтеграцію з Telegram. Використання сучасних технологій і гнучкої архітектури дозволяє легко розширювати функціонал у майбутньому, наприклад, додавання авторизації чи аналітики активності волонтерів. Таким чином, розроблений web-застосунок не лише відповідає меті роботи, а й створює міцну основу для подальшого розвитку цифрових інструментів підтримки молодіжних ініціатив.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ:

1. Тарасюк М.С., Замрій І.В. “Розробка веб-системи для організації допомоги та благодійності: цифровий інструмент підтримки у часи соціальних та гуманітарних викликів”, VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 145-148.

2. Тарасюк М.С., Замрій І.В. “Захист інформації у веб-застосунку для підтримки молодіжних ініціатив ГО «Лідерська Ініціатива»: Технологічні і соціальні аспекти безпеки цифрового середовища” Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 202-203

3.Тарасюк М.С., Замрій І.В. “Захист інформації у веб-застосунку для підтримки молодіжних ініціатив ГО «Лідерська Ініціатива»: технологічні і соціальні аспекти безпеки цифрового середовища” Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 333-334

ПЕРЕЛІК ПОСИЛАНЬ

1. Freeman, A. Pro ASP.NET Core 7, Tenth Edition. Apress, 2023. 1024 p. URL: https://doi.org/10.1007/978-1-4842-9244-0

2. Galloway, J., Wilson, N. Professional ASP.NET Core MVC. Wrox, 2023. 850 p. URL: https://www.wrox.com/book/professional-aspnet-core-mvc

3. Lerman, J., Miller, B. Programming Entity Framework: Code First. O'Reilly Media, 2022. 576 p. URL: https://www.oreilly.com/library/view/programming-entity-framework/9781449312948

4. Miller, R., Johnson, S. Entity Framework Core in Action, 2nd Edition. Manning Publications, 2023. 700 p. URL: https://www.manning.com/books/entity-framework-core-in-action-second-edition

5. Owens, M. The Definitive Guide to SQLite. Apress, 2021. 450 p. URL: https://doi.org/10.1007/978-1-4842-6894-6

6. Welling, L., Thomson, L. Web Database Applications with PHP and MySQL. O'Reilly Media, 2022. 592 p. URL: https://www.oreilly.com/library/view/web-database-applications/9780596005436

7. Flanagan, D. JavaScript: The Definitive Guide. O'Reilly Media, 2023. 1096 p.URL: https://www.oreilly.com/library/view/javascript-the-definitive/9781491952023

8. Bhargava, A. Mastering Webhooks: A Comprehensive Guide. Packt Publishing, 2024. 300 p. URL: https://www.packtpub.com/product/mastering-webhooks/9781803245678

9. Troelsen, A., Japikse, P. Pro C# 10 with .NET 6. Apress, 2023. 1200 p. URL: https://doi.org/10.1007/978-1-4842-7869-7

10. Smith, S. Building Telegram Bots: Develop Bots in 12 Programming Languages. Apress, 2022. 350 p. URL: https://doi.org/10.1007/978-1-4842-8768-2

11. Haverbeke, M. Eloquent JavaScript, 4th Edition. No Starch Press, 2024. 472 p. URL: https://eloquentjavascript.net/

12. Griffiths, I. Programming C# 10. O'Reilly Media, 2023. 800 p. URL: https://www.oreilly.com/library/view/programming-c-10/9781098117804/

13. Lock, A. ASP.NET Core in Action, 3rd Edition. Manning Publications, 2023. 450 p. URL: https://www.manning.com/books/asp-net-core-in-action-third-edition

14. Gaffney, J. et al. "SQLite: Past, Present, and Future." VLDB Endowment, 2022. URL: https://www.vldb.org/pvldb/vol15/p3535-gaffney.pdf

15. Ofoegbu, J. Exploring the React Slick Carousel Library. Medium, 2023. (доступ як до оглядової літератури) URL: https://medium.com/@JulietOma/exploring-the-react-slick-carousel-library-794e3b33602b

16. Microsoft. ASP.NET Core Documentation. URL: https://docs.microsoft.com/en-us/aspnet/core (date of access: 05.05.2025)

17. Microsoft. Entity Framework Core Documentation. URL: https://docs.microsoft.com/en-us/ef/core (date of access: 05.05.2025)

18. SQLite. Official Documentation. URL: https://www.sqlite.org/docs.html (date of access: 05.05.2025)

19. Mozilla Developer Network. JavaScript Documentation. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (date of access: 05.05.2025)

20. Microsoft. C# Programming Guide. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide (date of access: 05.05.2025)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для підтримки молодіжних ініціатив громадської огранізації «Лідерська ініціатива»»

Виконав студент 4 курсу
Групи ПД-42
Тарасюк Максим Сергійович
Керівник роботи
завідувач кафедри ІПЗ, д.т.н., професор Замрій Ірина Вікторівна

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підтримка управління молодіжними ініціативами громадської огранізації.

Об’єкт дослідження – процес управління інформацією та взаємодії в рамках лідерських ініціатив.

Предмет дослідження – програмне забезпечення для управління даними громадської організації.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Здійснити огляд та аналіз існуючих веб-сайтів, що використовуються для для підтримки молодіжних ініціатив.
2. Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку для підтримки молодіжних ініціатив.
3. Сформувати вимоги до розробки клієнт-серверного застосунку для підтримки молодіжних ініціатив.
4. Спроектувати архітектуру клієнт-серверного застосунку для підтримки молодіжних ініціатив.
5. Розробити клієнт-серверний застосунок для підтримки молодіжних ініціатив.
6. Провести тестування клієнт-серверного застосунку для підтримки молодіжних ініціатив.

3

АНАЛІЗ АНАЛОГІВ

Показник	**VolunteerMatch**	**Idealist**	**Mobilize**	LeadershipInitiative
Інформація про організацію	Базові профілі	Детальні профілі	Обмежена інформація	Детальні профілі, включаючи напрями діяльності, новини, реквізити
Заявки та форми	Форма реєстрації	Вакансії, події	Події	Форми для приєднання та співпраці з валідацією даних
Інтерактивність	Відсутня	Відсутня	Відсутня	Асинхронне оновлення даних, інтерактивні форми
Інтеграція з месенджерами	Відсутня	Відсутня	Відсутня	Інтеграція з Telegram-ботом для обробки запитів
Комунікація	Через email	Форум спільноти	Email/SMS	Через Telegram-бот для оперативного інформування
Доступність	Web, iOS, Android	Web	Web, iOS, Android	Web (з можливістю розширення)
Аналітика	Відсутня	Відсутня	Відстеження залученості	Відсутня (з можливістю додавання в майбутньому)
Цільова аудиторія	Волонтери	Волонтери, активісти	Організатори, волонтери	Волонтери, активісти, потенційні партнери
Особливості	Широкий вибір проєктів	Глибокі профілі, форум	Управління подіями, сповіщення	Інтеграція з Telegram, автоматизація заявок, динамічне інформування

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

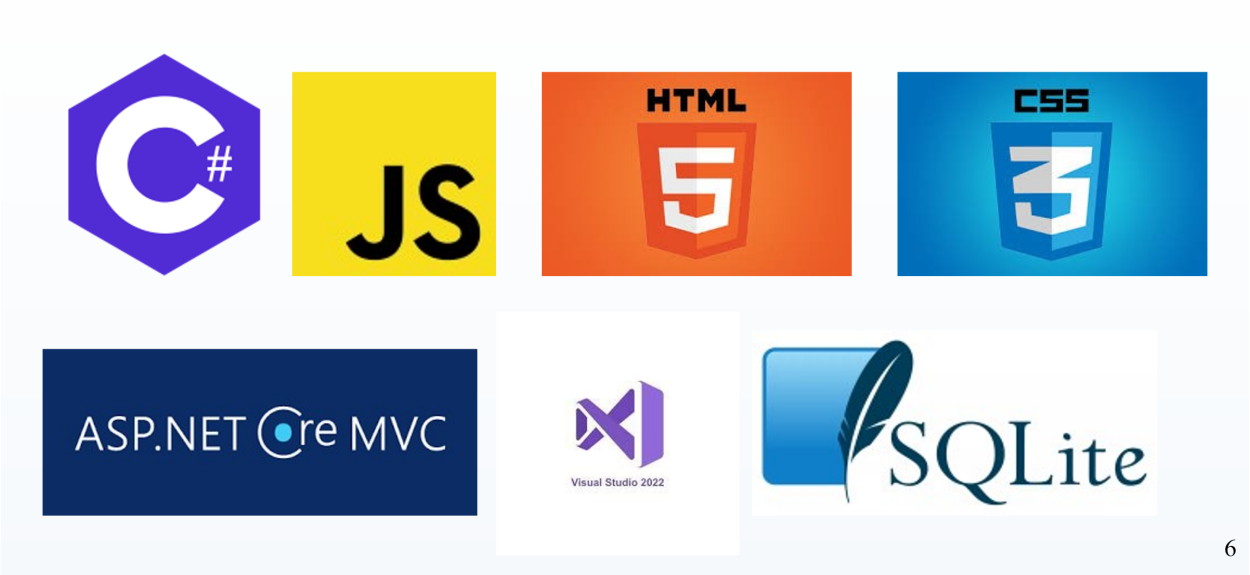
1. Відображення інформації про напрями діяльності організації
2. Відображення інформації про команду та новини
3. Відображення реквізитів та контактних даних
4. Подання заявок на вступ до організації
5. Подання заявок на співпрацю
6. Валідація даних у формах (наприклад, телефон, email)
7. Інтерактивне відображення контенту за допомогою каруселей
8. Використання іконок для візуального оформлення
9. Надсилання сповіщень через Telegram-бота
10. Обробка Telegram-запитів через вебхуки
11. Обробка помилок за допомогою Middleware
12. Ведення логування помилок
13. Відображення сторінки з повідомленням про помилку

Нефункціональні вимоги:

1. Завантаження сторінок за менше ніж 2 секунди
2. Адаптивне відображення на різних пристроях (ПК, мобільні телефони).
3. Стабільна робота під час звичайного навантаження.
4. Захист від веб-загроз (SQL-ін'єкції, XSS, CSRF)
5. Передача даних через HTTPS
6. Хешування персональних даних
7. Підтримка ручного і модульного тестування

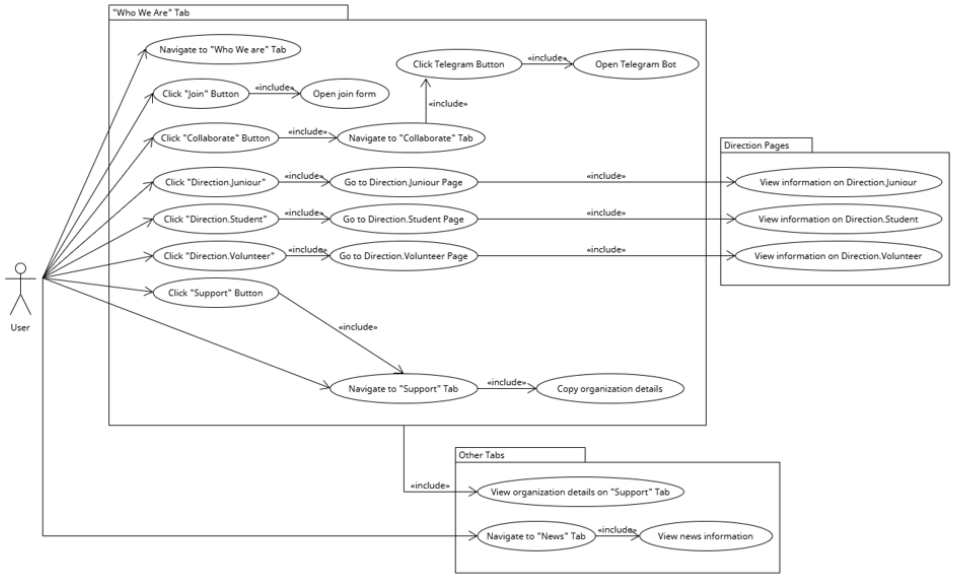
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



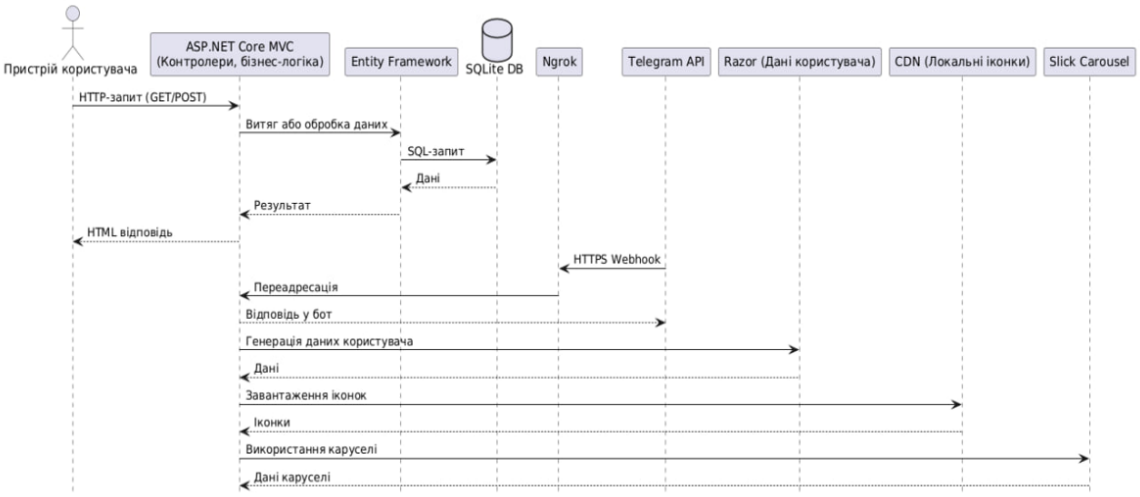
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



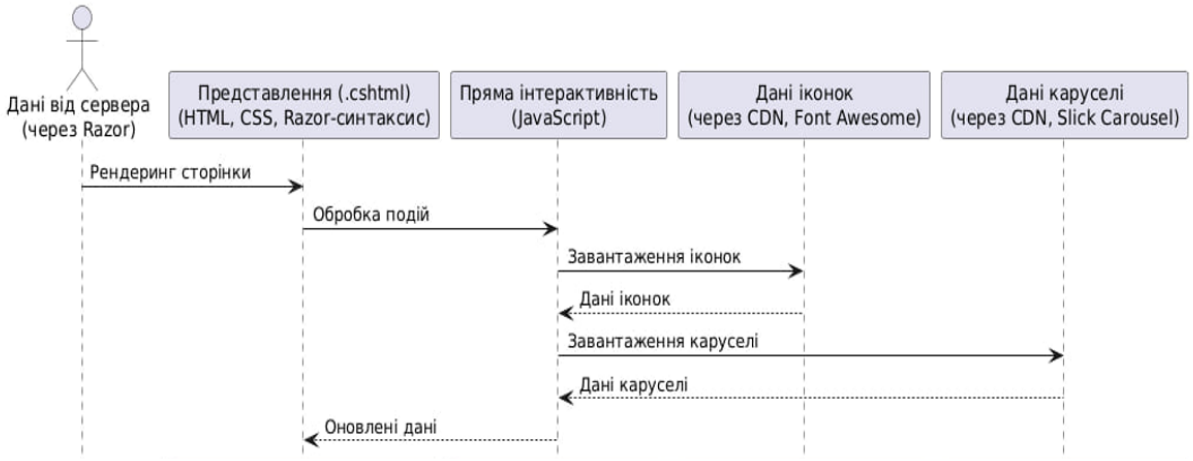
7

ДІАГРАМА РОЗГОРТАННЯ КЛІЄНТ-СЕРВЕРНОЇ МОДЕЛІ



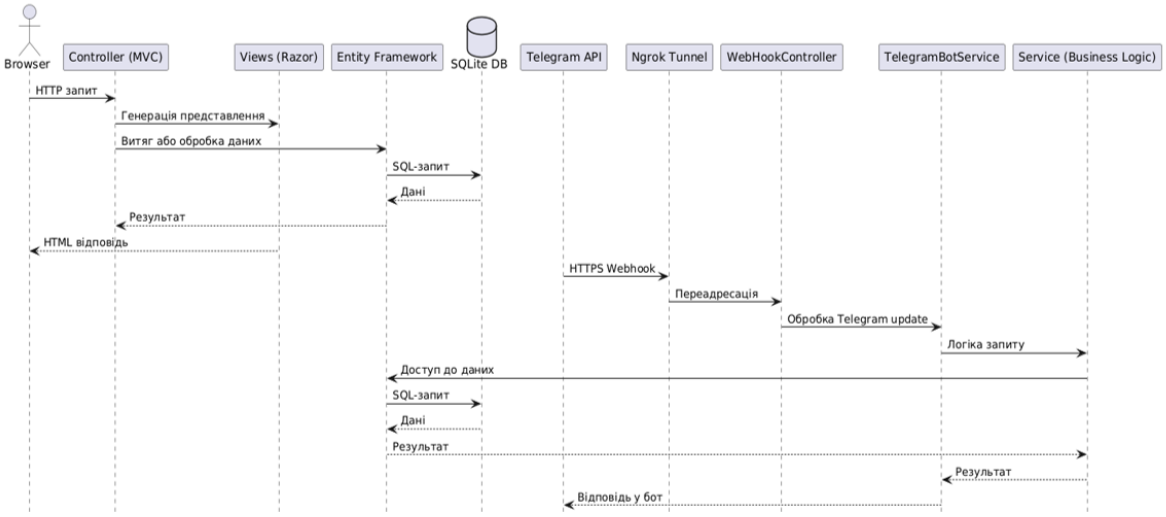
8

ДІАГРАМА СТРУКТУРИ КЛІЄНТСЬКОЇ ЧАСТИНИ



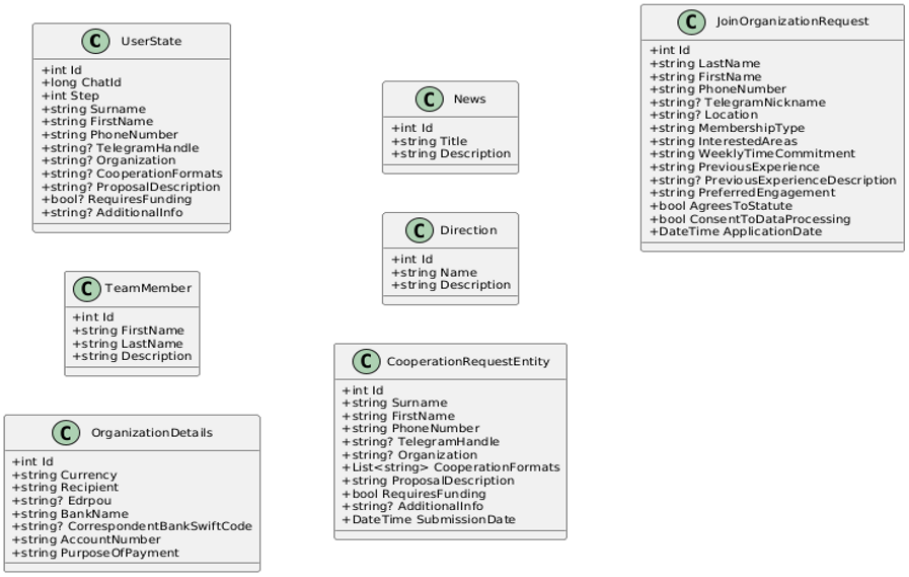
9

ДІАГРАМА ПОСЛІДОВНОСТІ



10

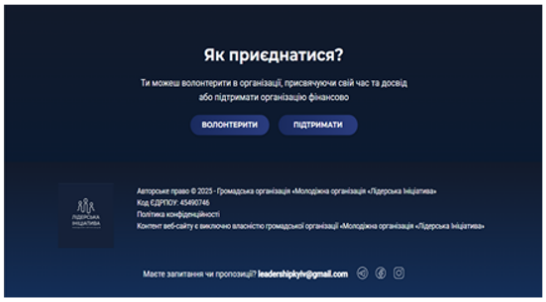
ДІАГРАМА КЛАСІВ



11

ЕКРАННІ ФОРМИ

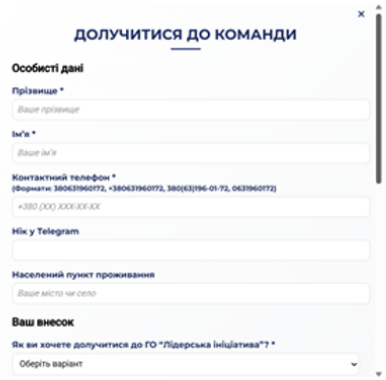
Хедер



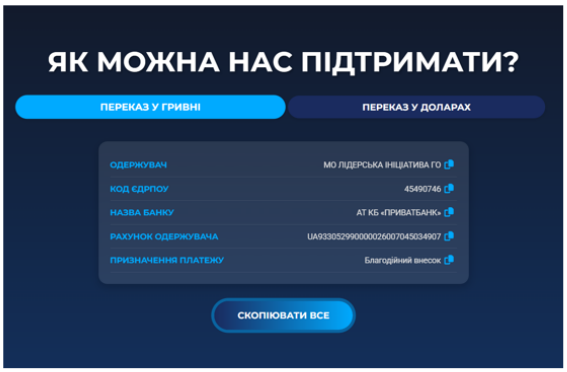
Футер

12

ЕКРАННІ ФОРМИ



Форма долучення до команди



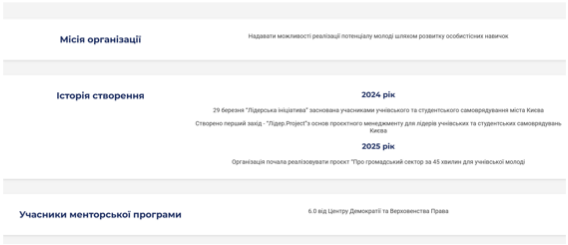
Форма копіювання реквізитів

13

ЕКРАННІ ФОРМИ

Наша команда

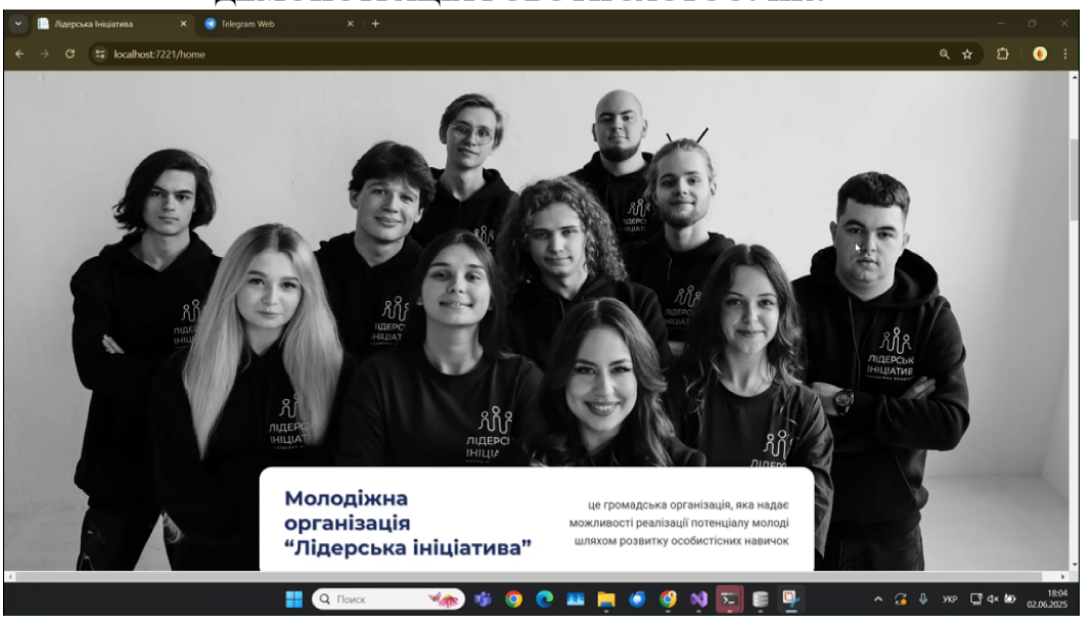
Напрями організації



Інформація про організацію

14

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ



15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Тарасюк М.С., Замрій І.В. “Розробка веб-системи для організації допомоги та благодійності: цифровий інструмент підтримки у часи соціальних та гуманітарних викликів”, VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 145-148.

2. Тарасюк М.С., Замрій І.В. “Захист інформації у веб-застосунку для підтримки молодіжних ініціатив ГО «Лідерська Ініціатива»: Технологічні і соціальні аспекти безпеки цифрового середовища” Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 202-203

3.Тарасюк М.С., Замрій І.В. “Захист інформації у веб-застосунку для підтримки молодіжних ініціатив ГО «Лідерська Ініціатива»: технологічні і соціальні аспекти безпеки цифрового середовища” Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 333-334

ВИСНОВКИ

1. Проведено огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку для підтримки молодіжних ініціатив. Для реалізації обрано технологічний стек: ASP.NET Core MVC для серверної частини, Entity Framework Core для роботи з базою даних та Telegram Bot API для інтеграції з месенджером.
2. Сформовано детальні вимоги до розробки клієнт-серверного застосунку для підтримки молодіжних ініціатив, включаючи специфікацію функціоналу (перегляд інформації, подання заявок, інформування через Telegram) та нефункціональні аспекти (швидкодія, масштабованість).
3. Спроектовано архітектуру клієнт-серверного застосунку для підтримки молодіжних ініціатив, використовуючи патерн MVC для розділення логіки представлення, даних і бізнес-логіки, а також інтеграцію з Telegram через вебхуки для обробки запитів у реальному часі.
4. Розроблено клієнт-серверний застосунок "Лідерська Ініціатива", який забезпечує перегляд інформації про організацію (напрями діяльності, команду, новини, реквізити), подання заявок на приєднання та співпрацю через веб-форми, а також інформування користувачів через Telegram-бота.
5. Проведено тестування клієнт-серверного застосунку для підтримки молодіжних ініціатив. Функціональне тестування підтвердило коректність роботи всіх модулів (перегляд даних, подання заявок, відправлення повідомлень через Telegram). Нефункціональне тестування показало, що застосунок відповідає вимогам швидкодії та адаптивності (коректне відображення на пристроях із різною роздільною здатністю екрану).

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Вихідний код контексту AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using LeadershipInitiative.Models;

namespace LeadershipInitiative.Data
{
   public class AppDbContext : DbContext
   {
      public DbSet<Direction> Directions { get; set; }
      public DbSet<TeamMember> TeamMembers { get;
set; }
      public DbSet<News> News { get; set; }
      public DbSet<OrganizationDetails>
OrganizationDetails { get; set; }
      public DbSet<CooperationRequestEntity>
CooperationRequests { get; set; }
      public DbSet<UserState> UserStates { get; set; }
      public DbSet<JoinOrganizationRequest>
JoinOrganizationRequests { get; set; }

      public
AppDbContext(DbContextOptions<AppDbContext>
options)
         : base(options)
      {
      }

      protected override void
OnModelCreating(ModelBuilder modelBuilder)
      {
         modelBuilder.Entity<Direction>().HasData(
            new Direction { Id = 1, Name =
"Лідер.Junior", Description = "Напрям для молодших
учасників, орієнтований на розвиток лідерських
навичок." },
            new Direction { Id = 2, Name =
"Лідер.Student", Description = "Напрям для студентів,
які хочуть розвивати свої лідерські якості." },
            new Direction { Id = 3, Name =
"Лідер.Volunteer", Description = "Напрям для
волонтерів, які прагнуть долучитися до громадських
ініціатив." }
         );

         modelBuilder.Entity<TeamMember>().HasData(
            new TeamMember { Id = 1, FirstName =
"Даніїл", LastName = "Буднік", Description =
"Менеджер напряму «Лідер.Volunteer»" },

            new TeamMember { Id = 2, FirstName =
"Михайло", LastName = "Ващенко", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 3, FirstName =
"Ірина", LastName = "Бондаренко", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 4, FirstName =
"Ірина", LastName = "Мирончук", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 5, FirstName =
"Катерина", LastName = "Добривечір", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 6, FirstName =
"Катерина", LastName = "Кошова", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 7, FirstName =
"Софія", LastName = "Квас", Description = "Менеджер
напряму «Лідер.Student»" },

            new TeamMember { Id = 8, FirstName =
"Олеся", LastName = "Лішенко", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 9, FirstName =
"Микола", LastName = "Васюк", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 10, FirstName =
"В'ячеслав", LastName = "Опанасенко", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 11, FirstName =
"Вікторія", LastName = "Ступак", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 12, FirstName =
"Максим", LastName = "Тарасюк", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 13, FirstName =
"Олександр", LastName = "Щербина", Description =
"Менеджер напряму «Лідер.Student»" },

            new TeamMember { Id = 14, FirstName =
"Владислав", LastName = "Рудик", Description =
"Менеджер напряму «Лідер.Student»" }
            );


modelBuilder.Entity<OrganizationDetails>().HasData(
            new OrganizationDetails
            {
               Id = 1,
               Currency = "UAH",
               Recipient = "МО ЛІДЕРСЬКА
ІНІЦІАТИВА ГО",
               Edrpou = "45490746",
               BankName = "АТ КБ «ПРИВАТБАНК»",
               AccountNumber =
"UA933052990000026007045034907",
               PurposeOfPayment = "Благодійний
внесок"
```

```
            },
            new OrganizationDetails
            {
               Id = 2,
               Currency = "USD",
               Recipient = "YOUTH ORGANIZATION
LEADERSHIP INITIATIVE",
               Edrpou = null,
               BankName = "JSC CB 'PRIVATBANK',
KYIV, UKRAINE",
               AccountNumber =
"UA093052990000026002035031733",
               PurposeOfPayment = "Charitable donation",
               CorrespondentBankSwiftCode =
"CHASUS33"
            });
      }
   }
}
```

Вихідний код файду HomeController.cs

```
using LeadershipInitiative.Data;
using LeadershipInitiative.ViewModels;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

public class HomeController : Controller
{
   private readonly AppDbContext _context;

   public HomeController(AppDbContext context)
   {
      _context = context;
   }

   [Route("home")]
   public async Task<IActionResult> Index()
   {
      var viewModel = new HomeViewModel
      {
         Title = "Лідерська Ініціатива",
         TeamMembers = await
_context.TeamMembers.ToListAsync()
      };

      ViewBag.Directions = await
_context.Directions.ToListAsync();

      return View(viewModel);
   }

   public async Task<IActionResult> Error()
   {
      ViewBag.Directions = await
_context.Directions.ToListAsync();
      return View();
   }
}
```

Вихідний код файду JoinController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;
using LeadershipInitiative.Data;

namespace LeadershipInitiative.Controllers
{
   public class JoinController : Controller
   {
      private readonly AppDbContext _context;

      public JoinController(AppDbContext context)
      {
         _context = context;
      }

      [HttpPost]
      public async Task<IActionResult>
Submit([FromForm] JoinOrganizationRequestDTO
model)
      {
         var validationResults = new
List<ValidationResult>();
         var validationContext = new
ValidationContext(model);
         bool isValid =
Validator.TryValidateObject(model, validationContext,
validationResults, true);

         if (!isValid)
         {
            foreach (var error in validationResults)
            {
               Console.WriteLine($"Error:
{error.ErrorMessage} (Member:
{error.MemberNames.FirstOrDefault()})");
            }
            return Json(new { success = false, errors =
validationResults.Select(v => v.ErrorMessage) });
         }

         var joinRequest = new JoinOrganizationRequest
         {
            LastName = model.LastName,
            FirstName = model.FirstName,
            PhoneNumber = model.PhoneNumber,
            TelegramNickname =
model.TelegramNickname,
            Location = model.Location,
            MembershipType = model.MembershipType,
            InterestedAreas = model.InterestedAreas,
            WeeklyTimeCommitment =
model.WeeklyTimeCommitment,
            PreviousExperience =
model.PreviousExperience,
            PreviousExperienceDescription =
model.PreviousExperienceDescription,
            PreferredEngagement =
model.PreferredEngagement,
            AgreesToStatute = model.AgreesToStatute,
            ConsentToDataProcessing =
model.ConsentToDataProcessing,
            ApplicationDate = model.ApplicationDate
         };

         await
_context.JoinOrganizationRequests.AddAsync(joinRequ
est);
```

```
            await _context.SaveChangesAsync();

            return Json(new { success = true });
        }
    }
}
```

Вихідний код файду TelegramWebHookController.cs

```
using Microsoft.AspNetCore.Mvc;
using Telegram.Bot.Types;
using
LeadershipInitiative.Services.TelegramBotServices;

namespace LeadershipInitiative.Controllers
{
    [Route("api/telegram")]
    [ApiController]
    public class TelegramWebhookController :
ControllerBase
    {
        private readonly TelegramBotService _botService;

        public
TelegramWebhookController(TelegramBotService
botService)
        {
            _botService = botService;
        }

        [HttpPost]
        public async Task<IActionResult>
Post([FromBody] Update update)
        {
            try
            {
                Console.WriteLine($"Received Telegram
webhook request at {DateTime.Now}");
                if (update == null)
                {
                    Console.WriteLine("Update is null");
                    return BadRequest("Update is null");
                }
                Console.WriteLine($"Processing update:
{update.Message?.Text ?? "No message"} from chat
{update.Message?.Chat.Id ?? 0}");
                await _botService.HandleUpdate(update);
                return Ok();
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error processing
Telegram update at {DateTime.Now}:
{ex.Message}\nStackTrace: {ex.StackTrace}");
                return StatusCode(500, "Internal server error");
            }
        }
    }
}
```

Вихідний код моделі JoinOrganizationRequest.cs

```
using System.ComponentModel.DataAnnotations;

public class JoinOrganizationRequest
{
    [Key]
    public int Id { get; set; }

    [Required(ErrorMessage = "Прізвище є
обов'язковим.")]
    [StringLength(100, ErrorMessage = "Прізвище не
може перевищувати 100 символів.")]
    public string LastName { get; set; }

    [Required(ErrorMessage = "Ім'я є обов'язковим.")]
    [StringLength(100, ErrorMessage = "Ім'я не може
перевищувати 100 символів.")]
    public string FirstName { get; set; }

    [Required(ErrorMessage = "Контактний телефон є
обов'язковим.")]

[RegularExpression(@"^(?:\+?380|0)?(?:\(?\d{2}\)?[\s-
]?)?\d{3}[\s-]?\d{2}[\s-]?\d{2}$", ErrorMessage =
"Невірний формат телефону. Використовуйте один із
форматів: 380631960172, +380631960172, 380(63)196-
01-72, 0631960172.")]
    [StringLength(16, ErrorMessage = "Телефон не може
перевищувати 16 символів.")]
    public string PhoneNumber { get; set; }

    [StringLength(50, ErrorMessage = "Нік у Telegram
не може перевищувати 50 символів.")]
    public string? TelegramNickname { get; set; }

    [StringLength(100, ErrorMessage = "Назва
населеного пункту не може перевищувати 100
символів.")]
    public string? Location { get; set; }

    [Required(ErrorMessage = "Оберіть, як ви хочете
долучитися.")]
    [StringLength(20, ErrorMessage = "Недопустима
довжина значення.")]
    public string MembershipType { get; set; }

    [Required(ErrorMessage = "Оберіть хоча б один
напрям.")]
    [StringLength(500, ErrorMessage = "Список
напрямів не може перевищувати 500 символів.")]
    public string InterestedAreas { get; set; }

    [Required(ErrorMessage = "Вкажіть, скільки часу
готові приділяти.")]
    [StringLength(20, ErrorMessage = "Недопустима
довжина значення.")]
    public string WeeklyTimeCommitment { get; set; }

    [Required(ErrorMessage = "Оберіть, чи брали
участь у діяльності раніше.")]
    [StringLength(3, ErrorMessage = "Недопустима
довжина значення.")]
    public string PreviousExperience { get; set; }

    [StringLength(500, ErrorMessage = "Опис досвіду не
може перевищувати 500 символів.")]
```

```
    public string? PreviousExperienceDescription { get;
set; }

    [Required(ErrorMessage = "Оберіть зручну форму
залучення.")]
    [StringLength(20, ErrorMessage = "Недопустима
довжина значення.")]
    public string PreferredEngagement { get; set; }

    [Required(ErrorMessage = "Необхідно погодитися зі
Статутом.")]
    public bool AgreesToStatute { get; set; }

    [Required(ErrorMessage = "Необхідно надати згоду
на обробку даних.")]
    public bool ConsentToDataProcessing { get; set; }

    public DateTime ApplicationDate { get; set; } =
DateTime.Now;
}
```

Вихідний код моделі Direction.cs

```
using System.ComponentModel.DataAnnotations;

namespace LeadershipInitiative.Models
{
    public class Direction
    {
        [Key]
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        public string Name { get; set; }

        [StringLength(500)]
        public string Description { get; set; }
    }
}
```

Вихідний код файлу-представлення _Layout.cshtml

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="utf-8" />
    <meta name="viewport"
content="width=device-width, initial-scale=1.0"
/>
    <title>Лідерська Ініціатива</title>
    <link rel="stylesheet" href="~/css/site.css"
asp-append-version="true" />
    <script src="~/js/site.js" asp-append-
version="true"></script>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/slic
k-carousel/1.8.1/slick.min.css" />
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/slic
k-carousel/1.8.1/slick-theme.min.css" />
    <link
href="https://fonts.googleapis.com/css2?family
=Montserrat:wght@700&family=Roboto:wght
@400;500&display=swap" rel="stylesheet">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font
-awesome/6.4.0/css/all.min.css" />
    <style>
        html {
            scroll-behavior: smooth;
        }
        .form-group select[multiple] {
            width: 100%;
            height: 150px;
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 8px;
            font-family: 'Roboto', sans-serif;
            font-size: 0.9em;
            color: #333;
            background-color: #fff;
            transition: border-color 0.3s ease, box-
shadow 0.3s ease;
        }
            .form-group select[multiple]:focus {
                outline: none;
                border-color: #1a2b5f;
                box-shadow: 0 0 8px rgba(26, 43, 95,
0.2);
                background-color: #f8f9fa;
            }
            .form-group select[multiple] option,
.form-group select {
                padding: 5px;
            }
        .form-group input[type="text"], .form-
group input[type="tel"], .form-group textarea,
.form-group select {
            width: 100%;
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 8px;
            font-family: 'Roboto', sans-serif;
            font-size: 0.9em;
        }
            .form-group input:focus, .form-group
textarea:focus, .form-group select:focus {
                outline: none;
                border-color: #1a2b5f;
                box-shadow: 0 0 8px rgba(26, 43, 95,
0.2);
            }
        .error-message {
            color: #ff6f61;
            font-size: 0.8em;
            display: block;
        }
        .form-hint {
            font-size: 0.8em;
            color: #666;
            margin-top: 4px;
        }
    </style>
</head>
<body>
    <header>
        <nav class="header-nav sticky-nav">
```

```
        <div class="container nav-container">
            <div class="burger-menu"
onclick="toggleMenu()">
                <i class="fas fa-bars"></i>
            </div>
            <div class="nav-links">
                <a asp-controller="Home" asp-
action="Index" asp-fragment="who-we-are"
class="nav-link
@(ViewContext.RouteData.Values["Controller"
]?.ToString() == "Home" ? "active" : "")">ХТО
МИ</a>
                <div class="dropdown">
                    <a class="nav-link
@(ViewContext.RouteData.Values["Controller"
]?.ToString() == "Directions" ? "active" :
"")">НАПРЯМИ</a>
                    <div class="dropdown-content">
                        @if (ViewBag.Directions !=
null)
                        { @foreach (var direction in
ViewBag.Directions)
                            {
                                <a asp-
controller="Directions" asp-action="Index" asp-
route-
id="@direction.Id">@direction.Name</a>
                            }        }
                    </div>
                </div>
                <a asp-controller="Cooperation"
asp-action="Index" asp-fragment="cooperation-
s" class="nav-link
@(ViewContext.RouteData.Values["Controller"
]?.ToString() == "Cooperation" ? "active" :
"")">СПІВПРАЦЯ</a>
                <a asp-controller="News" asp-
action="Index" asp-fragment="news-s"
class="nav-link
@(ViewContext.RouteData.Values["Controller"
]?.ToString() == "News" ? "active" :
"")">НОВИНИ</a>
                <a asp-controller="Support" asp-
action="Index" asp-fragment="support-s"
class="nav-link
@(ViewContext.RouteData.Values["Controller"
]?.ToString() == "Support" ? "active" :
"")">ПІДТРИМАТИ</a>
            </div>
        </div>
    </nav>
    <div class="container header-container">
        <div class="logo">
            <a asp-controller="Home" asp-
action="Index">
                <img src="~/images/білий на
синьому (1).jpg" alt="Лідерська Ініціатива
Логотип" />
            </a>
        </div>
        <div class="header-box">
            <h1>Громадська організація
“Лідерська ініціатива”</h1>
            <p>ВІЗЬМИ СВОЮ ІНІЦІАТИВУ
В НАШІЙ КОМАНДІ!</p>
            <button class="btn btn-blue pulse"
onclick="openJoinForm()">ДОЄДНАТИСЬ</
button>
        </div>
    </div>
</header>

<div class="modal" id="joinModal">
    <div class="modal-content">
        <span class="close-btn"
onclick="closeJoinForm()">×</span>
        <h2>ДОЛУЧИТИСЯ ДО
КОМАНДИ</h2>
        <form id="joinForm"
onsubmit="submitJoinForm(event)">
            <div class="form-section">
                <h3>Особисті дані</h3>
                <div class="form-group">
                    <label
for="lastName">Прізвище *</label>
                    <input type="text"
id="lastName" name="LastName"
placeholder="Ваше прізвище"
oninput="validateField('lastName', this.value)">
                    <span class="error-message"
id="lastName-error"></span>
                </div>
                <div class="form-group">
                    <label for="firstName">Ім’я
*</label>
                    <input type="text"
id="firstName" name="FirstName"
placeholder="Ваше ім’я"
oninput="validateField('firstName',
this.value)">
                    <span class="error-message"
id="firstName-error"></span>
                </div>
                <div class="form-group">
                    <label
for="phoneNumber">Контактний телефон
*<br><small>(Формати: 380631960172,
+380631960172, 380(63)196-01-72,
0631960172)</small></label>
                    <input type="tel"
id="phoneNumber" name="PhoneNumber"
placeholder="+380 (XX) XXX-XX-XX"
oninput="validateField('phoneNumber',
this.value)">
                    <span class="error-message"
id="phoneNumber-error"></span>
                </div>
                <div class="form-group">
                    <label
for="telegramNickname">Нік у
Telegram</label>
                    <input type="text"
id="telegramNickname"
name="TelegramNickname"
oninput="validateField('telegramNickname',
this.value, false)">
```

```
                <span class="error-message"
id="telegramNickname-error"></span>
            </div>
            <div class="form-group">
              <label
for="location">Населений пункт
проживання</label>
              <input type="text" id="location"
name="Location" placeholder="Ваше місто чи
село" oninput="validateField('location',
this.value, false)">
              <span class="error-message"
id="location-error"></span>
            </div>
          </div>

          <div class="form-section">
            <h3>Ваш внесок</h3>
            <div class="form-group">
              <label
for="membershipType">Як ви хочете
долучитися до ГО “Лідерська ініціатива”?
*</label>
              <select id="membershipType"
name="MembershipType"
onchange="validateField('membershipType',
this.value)">
                <option value="" disabled
selected>Оберіть варіант</option>
                <option
value="Volunteer">Як волонтер/ка</option>
                <option value="Member">Як
член/киня організації</option>
                <option
value="Both">Обидва варіанти</option>
              </select>
              <span class="error-message"
id="membershipType-error"></span>
            </div>
            <div class="form-group">
              <label
for="interestedAreas">Які напрями вас
цікавлять найбільше?
*<br><small>(Утримуйте Ctrl/Cmd, щоб
обрати кілька варіантів)</small></label>
              <select id="interestedAreas"
name="InterestedAreas" multiple
onchange="validateField('interestedAreas',
Array.from(this.selectedOptions).map(option
=> option.value).join(','))">
                <option
value="LeaderJunior">Лідер.Junior (14–17
років)</option>
                <option
value="LeaderStudent">Лідер.Student
(студенти)</option>
                <option
value="LeaderVolunteer">Лідер.Volunteer
(волонтерство)</option>
                <option
value="Administrative">Адміністративна/орг
анізаційна допомога</option>
                <option
value="SMMDesign">SMM, дизайн або
фото/відео</option>
                <option
value="Other">Інше</option>
              </select>
              <input type="text"
id="interestedAreasOther"
name="InterestedAreasOther"
placeholder="Опишіть коротко свій досвід
або мотивацію (до 500 символів)"
style="display: none;"
oninput="validateField('interestedAreasOther',
this.value)">
              <span class="error-message"
id="interestedAreas-error"></span>
            </div>
            <div class="form-group">
              <label
for="weeklyTimeCommitment">Скільки часу
ви готові приділяти діяльності в організації
щотижня? *</label>
              <select
id="weeklyTimeCommitment"
name="WeeklyTimeCommitment"
onchange="validateField('weeklyTimeCommit
ment', this.value)">
                <option value="" disabled
selected>Оберіть варіант</option>
                <option
value="UpTo2Hours">до 2 годин</option>
                <option
value="3To5Hours">3–5 годин</option>
                <option
value="6PlusHours">6+ годин</option>
                <option
value="Flexible">залежить від періоду /
гнучко</option>
              </select>
              <span class="error-message"
id="weeklyTimeCommitment-error"></span>
            </div>
            <div class="form-group">
              <label
for="previousExperience">Чи брали участь у
громадській або волонтерській діяльності
раніше? *</label>
              <select id="previousExperience"
name="PreviousExperience"
onchange="validateField('previousExperience',
this.value)">
                <option value="" disabled
selected>Оберіть варіант</option>
                <option
value="Yes">Так</option>
                <option
value="No">Ні</option>
              </select>
              <textarea
id="previousExperienceDescription"
name="PreviousExperienceDescription"
placeholder="Коротко опишіть свій досвід"
style="display: none;"
```

```
oninput="validateField('previousExperienceDescription', this.value)"></textarea>
                <span class="error-message" id="previousExperience-error"></span>
                <span class="error-message" id="previousExperienceDescription-error"></span>
            </div>
            <div class="form-group">
                <label for="preferredEngagement">Яка форма залучення вам зручніша? *</label>
                <select id="preferredEngagement" name="PreferredEngagement" onchange="validateField('preferredEngagement', this.value)">
                    <option value="" disabled selected>Оберіть варіант</option>
                    <option value="Online">Онлайн</option>
                    <option value="Offline">Офлайн (у Києві)</option>
                    <option value="Both">Обидва варіанти</option>
                </select>
                <span class="error-message" id="preferredEngagement-error"></span>
            </div>
        </div>

        <div class="form-section">
            <h3>Згоди</h3>
            <div class="form-group">
                <label>
                    <input type="checkbox" id="agreesToStatute" name="AgreesToStatute" value="true" onchange="validateField('agreesToStatute', this.checked)">
                    <input type="hidden" name="AgreesToStatute" value="false">
                    Зі Статутом ГО "Лідерська ініціатива" ознайомлений/на та зобов’язуюсь його дотримуватися *
                    <a href="https://drive.google.com/file/d/1XXlFu3S813aPwtxVkmdSD9TPI79kRHSl/view?usp=sharing" target="_blank">Переглянути Статут</a>
                </label>
                <span class="error-message" id="agreesToStatute-error"></span>
            </div>
            <div class="form-group">
                <label>
                    <input type="checkbox" id="consentToDataProcessing" name="ConsentToDataProcessing" value="true" checked onchange="validateField('consentToDataProcessing', this.checked)">
                    <input type="hidden" name="ConsentToDataProcessing" value="false">
                    Відповідно до Закону України "Про захист персональних даних" надаю згоду на подальшу обробку моїх персональних даних *
                </label>
                <span class="error-message" id="consentToDataProcessing-error"></span>
            </div>
        </div>

        <button type="submit" class="btn btn-blue">Надіслати заявку</button>
    </form>
  </div>
 </div>
 <div id="resultMessage" class="result-message"></div>
 <main>
  @RenderBody()
 </main>
 <footer>
  <div class="container">
   <div class="footer-top">
    <div class="footer-logo">
     <img src="/images/білий на синьому (1).jpg" alt="Лідерська Ініціатива" />
    </div>
    <div class="footer-info">
     <p>Авторське право © 2025 - Громадська організація «Молодіжна організація «Лідерська Ініціатива»</p>
     <p>Код ЄДРПОУ: 45490746</p>
     <p>Політика конфіденційності</p>
     <p>Контент веб-сайту є виключно власністю громадської організації «Молодіжна організація «Лідерська Ініціатива»</p>
    </div>
   </div>
   <div class="footer-bottom">
    <p>Маєте запитання чи пропозиції? <a href="mailto:leadershipkyiv@gmail.com">leadershipkyiv@gmail.com</a></p>
    <div class="social-icons">
     <a target="_blank" href="https://t.me/leaderinkyiv"><img src="/images/icons8-телеграм-50.png" alt="Telegram" /></a>
     <a target="_blank" href="https://www.facebook.com/leader.in.kyiv"><img src="/images/icons8-facebook-новый-50.png" alt="Facebook" /></a>
     <a target="_blank" href="https://www.instagram.com/leader.in.kyiv/"><img src="/images/icons8-instagram-50.png" alt="Instagram" /></a>
    </div>
```

```
        </div>
      </div>
    </footer>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.8.1/slick.min.js"></script>
    <script>
      function openJoinForm() {
        const modal = document.getElementById("joinModal");
        modal.style.display = "flex";
        setTimeout(() => modal.classList.add("show"), 10);
        document.querySelectorAll('.error-message').forEach(span => span.textContent = '');
      }
      function closeJoinForm() {
        const modal = document.getElementById("joinModal");
        modal.classList.remove("show");
        setTimeout(() => {
          modal.style.display = "none";

          document.getElementById("joinForm").reset();

          document.getElementById("interestedAreasOther").style.display = "none";

          document.getElementById("previousExperienceDescription").style.display = "none";
          document.querySelectorAll('.error-message').forEach(span => span.textContent = '');
        }, 300);
      }
      function showResultMessage(message, success) {
        const resultMessage = document.getElementById("resultMessage");
        resultMessage.textContent = message;
        resultMessage.style.color = success ? "#1a2b5f" : "#ff6f61";
        resultMessage.classList.add("show");
        setTimeout(() => resultMessage.classList.remove("show"), 3000);
      }

      function validateField(fieldId, value, isRequired = true) {
        const errorSpan = document.getElementById(`${fieldId}-error`);
        let isValid = true;

        switch (fieldId) {
          case 'lastName':
          case 'firstName':
            if (isRequired && !value.trim()) {
              errorSpan.textContent = `${fieldId === 'lastName' ? 'Прізвище' : 'Ім\'я'} є обов\'язковим.`;
              isValid = false;
            }
            break;
          case 'phoneNumber':
            const phoneRegex = /^(?:\+?380|0)?(?:\(?\d{2}\)?[\s-]?)?\d{3}[\s-]?\d{2}[\s-]?\d{2}$/;
            if (isRequired && !value.trim()) {
              errorSpan.textContent = 'Контактний телефон є обов\'язковим.';
              isValid = false;
            } else if (value.trim() && !phoneRegex.test(value)) {
              errorSpan.textContent = 'Невірний формат телефону. Використовуйте один із форматів: 380631960172, +380631960172, 380(63)196-01-72, 0631960172.';
              isValid = false;
            }
            break;
          case 'telegramNickname':
          case 'location':
            if (!isRequired) {
              errorSpan.textContent = '';
              isValid = true;
            }
            break;
          case 'membershipType':
          case 'weeklyTimeCommitment':
          case 'previousExperience':
          case 'preferredEngagement':
            if (isRequired && !value) {
              errorSpan.textContent = `Оберіть ${fieldId === 'membershipType' ? 'як ви хочете долучитися' :
                fieldId === 'weeklyTimeCommitment' ? 'скільки часу готові приділяти' :
                fieldId === 'previousExperience' ? 'чи брали участь у діяльності раніше' :
                'зручну форму залучення'}.`;
              isValid = false;
            }
            break;
          case 'interestedAreas':
            const interestedAreasSelect = document.getElementById('interestedAreas');
            const selectedAreas = value.split(',').filter(v => v);
            if (isRequired && selectedAreas.length === 0) {
              errorSpan.textContent = 'Оберіть хоча б один напрям.';
              isValid = false;
            } else if (selectedAreas.includes('Other') &&
```

```
!document.getElementById('interestedAreasOth
er').value.trim()) {
                    errorSpan.textContent = 'Якщо
обрано \'Інше\', опишіть свій досвід або
мотивацію.';
                    isValid = false;
                }
                break;
            case 'interestedAreasOther':
                if
(Array.from(document.getElementById('interest
edAreas').selectedOptions).some(opt =>
opt.value === 'Other') && !value.trim()) {
                    errorSpan.textContent = 'Якщо
обрано \'Інше\', опишіть свій досвід або
мотивацію.';
                    isValid = false;
                }
                break;
            case 'previousExperienceDescription':
                if
(document.getElementById('previousExperienc
e').value === 'Yes' && !value.trim()) {
                    errorSpan.textContent = 'Якщо
\'Так\', опишіть свій досвід.';
                    isValid = false;
                }
                break;
            case 'agreesToStatute':
            case 'consentToDataProcessing':
                if (isRequired && !value) {
                    errorSpan.textContent =
`Необхідно погодитися ${fieldId ===
'agreesToStatute' ? 'зі Статутом' : 'на обробку
даних'}.`;
                    isValid = false;
                }
                break;
        }

        if (isValid) {
            errorSpan.textContent = '';
        }
    }

    async function submitJoinForm(event) {
        event.preventDefault();
        if (!validateForm()) return;

        const form =
document.getElementById("joinForm");
        const formData = new FormData(form);

        const interestedAreas =
Array.from(document.getElementById("interest
edAreas").selectedOptions)
            .map(option => option.value)
            .filter(value => value !== "Other");
        if
(document.getElementById("interestedAreas").s
electedOptions.length > 0) {
            const otherValue =
document.getElementById("interestedAreasOth
er").value.trim();
            if (otherValue)
interestedAreas.push(otherValue);
        }
        formData.set("InterestedAreas",
interestedAreas.join(","));

        try {
            const response = await
fetch('/Join/Submit', { method: 'POST', body:
formData });
            const data = await response.json();

            if (data.success) {
                closeJoinForm();
                setTimeout(() =>
showResultMessage("Ваша заявка успішно
надіслана!", true), 300);
            } else {
                closeJoinForm();
                setTimeout(() =>
showResultMessage(data.errors ?
data.errors.join(", ") : "Виникла помилка при
обробці заявки. Спробуйте ще раз.", false),
300);
            }
        } catch (error) {
            console.error('Error:', error);
            closeJoinForm();
            setTimeout(() =>
showResultMessage("Виникла помилка.
Спробуйте ще раз.", false), 300);
        }
    }

    function validateForm() {
        let isValid = true;
        const errors = {};

        validateField('lastName',
document.getElementById('lastName').value);
        validateField('firstName',
document.getElementById('firstName').value);
        validateField('phoneNumber',
document.getElementById('phoneNumber').val
ue);
        validateField('telegramNickname',
document.getElementById('telegramNickname')
.value, false);
        validateField('location',
document.getElementById('location').value,
false);
        validateField('membershipType',
document.getElementById('membershipType').
value);
        validateField('interestedAreas',
Array.from(document.getElementById('interest
edAreas').selectedOptions).map(option =>
option.value).join(','));

validateField('weeklyTimeCommitment',
```

```
document.getElementById('weeklyTimeCommitment').value);
        validateField('previousExperience', document.getElementById('previousExperience').value);

validateField('previousExperienceDescription', document.getElementById('previousExperienceDescription').value);
        validateField('preferredEngagement', document.getElementById('preferredEngagement').value);
        validateField('agreesToStatute', document.getElementById('agreesToStatute').checked);

validateField('consentToDataProcessing', document.getElementById('consentToDataProcessing').checked);

        document.querySelectorAll('.error-message').forEach(span => {
            if (span.textContent) {
                isValid = false;
            }
        });

        return isValid;
    }

    window.onclick = function (event) {
        const modal = document.getElementById("joinModal");
        if (event.target === modal) closeJoinForm();
    }

    function toggleMenu() {
        const navLinks = document.querySelector('.nav-links');
        navLinks.classList.toggle('active');
        const dropdown = document.querySelector('.dropdown');
        dropdown.classList.remove('active');
    }


document.querySelectorAll('.dropdown').forEach(dropdown => {
        const dropdownLink = dropdown.querySelector('.nav-link');
        dropdownLink.addEventListener('click', (e) => {
            if (window.innerWidth <= 768) {
                e.preventDefault();
                dropdown.classList.toggle('active');
            }
        });


dropdown.addEventListener('mouseenter', () => {
            if (window.innerWidth > 768) {
                dropdown.classList.add('active');
            }
        });


dropdown.addEventListener('mouseleave', () => {
            if (window.innerWidth > 768) {

dropdown.classList.remove('active');
            }
        });
    });


document.getElementById("interestedAreas").addEventListener('change', function () {
        const otherField = document.getElementById("interestedAreasOther");
        const selectedOptions = Array.from(this.selectedOptions).map(option => option.value);
        otherField.style.display = selectedOptions.includes("Other") ? "block" : "none";
        validateField('interestedAreas', selectedOptions.join(','));
    });


document.getElementById("previousExperience").addEventListener('change', function () {
        const descField = document.getElementById("previousExperienceDescription");
        descField.style.display = this.value === "Yes" ? "block" : "none";
        validateField('previousExperience', this.value);
    });
  </script>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

Вихідний код сервісу TelegramBotService.cs

```
using LeadershipInitiative.Data;
using LeadershipInitiative.Models;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.ReplyMarkups;
using Microsoft.EntityFrameworkCore;
using System.Text.RegularExpressions;
using System.Collections.Concurrent;

namespace LeadershipInitiative.Services.TelegramBotServices
{
  public class TelegramBotService
  {
    private readonly TelegramBotClient _botClient;
```

```
    private readonly
IDbContextFactory<AppDbContext>
_contextFactory;
    private readonly string _botToken;
    private readonly
ConcurrentDictionary<long, UserState>
_tempUserStates;

    public TelegramBotService(IConfiguration
configuration,
IDbContextFactory<AppDbContext>
contextFactory)
    {
        _botToken =
configuration["TelegramBotToken"];
        _botClient = new
TelegramBotClient(_botToken);
        _contextFactory = contextFactory;
        _tempUserStates = new
ConcurrentDictionary<long, UserState>();
    }

    public async Task
SetWebhookAsync(string webhookUrl)
    {
        try
        {
            Console.WriteLine($"Setting
webhook to {webhookUrl}");
            await
_botClient.SetWebhook(webhookUrl);
            Console.WriteLine("Webhook set
successfully");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error setting
webhook: {ex.Message}");
            throw;
        }
    }

    public async Task HandleUpdate(Update
update)
    {
        Console.WriteLine($"Received update:
{update?.Message?.Text ?? "No message"}
from chat {update?.Message?.Chat.Id ?? 0}");
        if (update.Message is { } message)
        {
            var chatId = message.Chat.Id;
            var userMessage = message.Text;

            if (userMessage == "/start")
            {
                await
SendWelcomeMessage(chatId);
                return;
            }
            else if (userMessage ==
"/cooperation")
            {
                await SendForm(chatId);
                return;
            }

            await ProcessFormInput(chatId,
userMessage);
        }
    }

    private async Task
SendWelcomeMessage(long chatId)
    {
        var welcomeMessage = "Привіт! Я бот
ГО «Лідерська ініціатива».\n" +
                             "Доступні команди:\n" +
                             "/cooperation - Заповнити
форму для співпраці";
        await _botClient.SendMessage(chatId,
welcomeMessage, replyMarkup: new
ReplyKeyboardRemove());
        Console.WriteLine($"Welcome message
sent to chat {chatId}");
    }

    private async Task SendForm(long chatId)
    {
        using var context =
_contextFactory.CreateDbContext();
        try
        {
            var existingState = await
context.UserStates.FirstOrDefaultAsync(u =>
u.ChatId == chatId);
            if (existingState != null)
            {

context.UserStates.Remove(existingState);
                await
context.SaveChangesAsync();
                Console.WriteLine($"Cleared
previous state for chat {chatId} from
database");
            }

            _tempUserStates.TryRemove(chatId,
out _);
            Console.WriteLine($"Cleared
temporary state for chat {chatId}");

            var formMessage = "Заповніть
форму для співпраці з ГО «Лідерська
ініціатива»:\n" +
                              "Введіть ваше прізвище";
            await
_botClient.SendMessage(chatId, formMessage,
replyMarkup: new ReplyKeyboardRemove());
            Console.WriteLine($"Form sent to
chat {chatId}");

            var tempState = new UserState {
ChatId = chatId, Step = 1 };
            _tempUserStates[chatId] = tempState;
            Console.WriteLine($"Initialized
temporary user state for chat {chatId}, step 1");
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error in
SendForm for chat {chatId}: {ex.Message}");
            throw;
        }
    }

    private async Task ProcessFormInput(long
chatId, string userMessage)
    {
        using var context =
_contextFactory.CreateDbContext();
        try
        {
            if
(!_tempUserStates.TryGetValue(chatId, out var
tempState))
            {
                var userState = await
context.UserStates.FirstOrDefaultAsync(u =>
u.ChatId == chatId);
                if (userState == null)
                {
                    await
_botClient.SendMessage(chatId, "Будь ласка,
почніть з команди /start або /cooperation.");
                    return;
                }

                await ProcessExistingState(chatId,
userMessage, userState, context);
                return;
            }

            Console.WriteLine($"Processing
temporary step {tempState.Step} for chat
{chatId}, input: {userMessage}");

            switch (tempState.Step)
            {
                case 1:
                    if
(!ValidateName(userMessage))
                    {
                        await
_botClient.SendMessage(chatId, "Прізвище
має містити лише літери, пробіли або дефіс.
Спробуйте ще раз:");
                        return;
                    }
                    tempState.Surname =
userMessage;
                    await
_botClient.SendMessage(chatId, "Введіть ваше
ім’я:");
                    tempState.Step++;
                    _tempUserStates[chatId] =
tempState;
                    break;

                case 2:
                    if
(!ValidateName(userMessage))
                    {
                        await
_botClient.SendMessage(chatId, "Ім’я має
містити лише літери, пробіли або дефіс.
Спробуйте ще раз:");
                        return;
                    }
                    tempState.FirstName =
userMessage;
                    await
_botClient.SendMessage(chatId, "Введіть ваш
контактний телефон (наприклад,
0631960172, 380631960172,
+380631960172):");
                    tempState.Step++;
                    _tempUserStates[chatId] =
tempState;
                    break;

                case 3:
                    if
(!ValidatePhoneNumber(userMessage))
                    {
                        await
_botClient.SendMessage(chatId, "Невірний
формат телефону. Використовуйте один із
форматів: 0631960172, 380631960172,
+380631960172. Спробуйте ще раз:");
                        return;
                    }
                    tempState.PhoneNumber =
userMessage;

                    var userState = new UserState
                    {
                        ChatId = chatId,
                        Step = 4,
                        Surname =
tempState.Surname,
                        FirstName =
tempState.FirstName,
                        PhoneNumber =
tempState.PhoneNumber
                    };

context.UserStates.Add(userState);
                    await
context.SaveChangesAsync();
                    Console.WriteLine($"Saved user
state to database for chat {chatId}, step 4");


_tempUserStates.TryRemove(chatId, out _);

                    await
_botClient.SendMessage(chatId, "Введіть ваш
нік у Telegram (або введіть 0):");
                    break;
            }
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Error in
ProcessFormInput for chat {chatId}:
{ex.Message}\nStackTrace: {ex.StackTrace}");
            await
_botClient.SendMessage(chatId, "Виникла
помилка. Спробуйте ще раз або зверніться
до адміністратора.");
            throw;
        }
    }

    private async Task
ProcessExistingState(long chatId, string
userMessage, UserState userState,
AppDbContext context)
    {
        Console.WriteLine($"Processing step
{userState.Step} for chat {chatId}, input:
{userMessage}");

        switch (userState.Step)
        {
            case 4:
                userState.TelegramHandle =
userMessage == "0" ? null : userMessage;
                await
_botClient.SendMessage(chatId, "Введіть
назву вашої організації (або введіть 0):");
                break;

            case 5:
                userState.Organization =
userMessage == "0" ? null : userMessage;
                var keyboard = new
ReplyKeyboardMarkup(new[]
                {
                new[] { new
KeyboardButton("Проведення спільного
заходу"), new KeyboardButton("Запрошення
ГО для тренінгу") },
                new[] { new
KeyboardButton("Створення волонтерської
ініціативи"), new KeyboardButton("Спільна
заявка на грант") },
                new[] { new
KeyboardButton("Допомога ГО") }
            });
                await
_botClient.SendMessage(chatId, "Виберіть
формат співпраці (можете обрати кілька
варіантів, натиснувши кілька кнопок):",
replyMarkup: keyboard);
                break;

            case 6:
                var formats =
userMessage.Split(new[] { ',' },
StringSplitOptions.RemoveEmptyEntries)
                    .Select(f => f.Trim())
                    .Where(f => new[] {
"Проведення спільного заходу",
"Запрошення ГО для тренінгу", "Створення
волонтерської ініціативи", "Спільна заявка
на грант", "Допомога ГО" }.Contains(f))
                    .Distinct()
                    .ToList();
                if (formats.Count == 0)
                {
                    await
_botClient.SendMessage(chatId, "Будь ласка,
виберіть хоча б один формат із клавіатури:");
                    return;
                }
                userState.CooperationFormats =
string.Join(", ", formats);
                await
_botClient.SendMessage(chatId, "Опишіть
пропозицію (до 500 символів):",
replyMarkup: new ReplyKeyboardRemove());
                break;

            case 7:
                if (userMessage.Length > 500)
                {
                    await
_botClient.SendMessage(chatId, "Опис
перевищує 500 символів. Спробуйте ще
раз:");
                    return;
                }
                userState.ProposalDescription =
userMessage;
                var yesNoKeyboard = new
ReplyKeyboardMarkup(new[] { new[] { new
KeyboardButton("Так"), new
KeyboardButton("Ні") } });
                await
_botClient.SendMessage(chatId, "Чи є у вас
потреба у фінансуванні (Так/Ні):",
replyMarkup: yesNoKeyboard);
                break;

            case 8:
                if (userMessage.ToLower() !=
"так" && userMessage.ToLower() != "ні")
                {
                    await
_botClient.SendMessage(chatId, "Будь ласка,
введіть 'Так' або 'Ні':");
                    return;
                }
                userState.RequiresFunding =
userMessage.ToLower() == "так";
                await
_botClient.SendMessage(chatId, "Введіть
додаткову інформацію (або введіть 0):",
replyMarkup: new ReplyKeyboardRemove());
                break;

            case 9:
                userState.AdditionalInfo =
userMessage == "0" ? null : userMessage;
                await SaveRequest(chatId,
userState);
```

```
                await
_botClient.SendMessage(chatId, "Дякуємо!
Ваша заявка обробляється. Ми зв’яжемося з
вами.", replyMarkup: new
ReplyKeyboardRemove());

context.UserStates.Remove(userState);
                break;
            }

            userState.Step++;
            context.UserStates.Update(userState);
            await context.SaveChangesAsync();
            Console.WriteLine($"Updated user state
for chat {chatId}, step {userState.Step}");
        }

        private bool ValidateName(string name)
        {
            return
!string.IsNullOrWhiteSpace(name) &&
Regex.IsMatch(name, @"^[a-zA-Za-яA-
ЯїєірЇЄІҐ\s-]+$");
        }

        private bool ValidatePhoneNumber(string
phone)
        {
            string cleanPhone =
Regex.Replace(phone, @"[^\d]", "");
            if (cleanPhone.Length < 9 ||
cleanPhone.Length > 12) return false;

            return Regex.IsMatch(phone,
@"^(\+?380)?0?6[0-9]{8}$") ||
                Regex.IsMatch(phone, @"^380[0-
9]{9}$") ||
                Regex.IsMatch(phone, @"^\+380[0-
9]{9}$");
        }

        private async Task SaveRequest(long
chatId, UserState userState)
        {
            using var context =
_contextFactory.CreateDbContext();
            await using var transaction = await
context.Database.BeginTransactionAsync();
            try
            {
                var request = new
CooperationRequestEntity
                {
                    Surname = userState.Surname,
                    FirstName = userState.FirstName,
                    PhoneNumber =
userState.PhoneNumber,
                    TelegramHandle =
userState.TelegramHandle,
                    Organization =
userState.Organization,
                    CooperationFormats =
userState.CooperationFormats != null ?
userState.CooperationFormats.Split(new[] { ", "
},
StringSplitOptions.RemoveEmptyEntries).ToLi
st() : new List<string>(),
                    ProposalDescription =
userState.ProposalDescription ?? string.Empty,
                    RequiresFunding =
userState.RequiresFunding ?? false,
                    AdditionalInfo =
userState.AdditionalInfo,
                    SubmissionDate =
DateTime.UtcNow
                };


context.CooperationRequests.Add(request);
                await context.SaveChangesAsync();
                await transaction.CommitAsync();
                Console.WriteLine($"Request saved
for chat {chatId} with ID
{request.FirstName}");
            }
            catch (Exception ex)
            {
                await transaction.RollbackAsync();
                Console.WriteLine($"Error saving
request for chat {chatId}: {ex.Message}");
                throw;
            }
        }
    }
}
```