Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ НА .NET

Текстова частина до курсової роботи

Керівник курсової роботи
ст.викладач Борозенний С.О.
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2022 р.
Виконав студент БП ІПЗ-4
Мальков Є.В.
(прізвище та ініціали)
“____” _____ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

доцент, к.ф.-м.н.

_____С. С. Гороховський

(підпис)

“____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Малькову Єгору Вікторовичу факультету інформатики 3 курсу

ТЕМА Розробка веб-застосунку на .NET

Вихідні дані: Веб-застосунок для допомоги з житлом, транспортом та речами для українців, що постраждали в наслідок воєнних дій.

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Платформа .NET. Особливості, розвиток, перспективи

2 Робота з базами даних та веб-розробка за допомогою .NET

3 Розробка програмного продукту

Висновки

Список літератури

Дата видачі “___” _____ 2021 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розробка веб-застосунку на .NET

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	Листопад 2021	
2.	Огляд технічної літератури за темою роботи.	Лютий 2022	
3.	Написання першого розділу	Лютий 2022	
4.	Написання другого розділу	Лютий 2022	
5.	Написання третього розділу	Березень 2022	
6.	Початок розробки застосунку, дизайн бази даних	Березень 2022	
7.	Розробка бізнес-логіки застосунку	Квітень 2022	
8.	Розробка веб-частини, завершення роботи над застосунком	Квітень 2022	
8.	Написання висновків розділу	Квітень 2022	
10.	Корегування роботи	Травень 2022	
11.	Розробка презентації	Травень 2022	
12.	Захист курсової роботи	Травень 2022	

Студент Мальков Єгор Вікторович

Керівник Борозенний Сергій Олександрович

“_____”_____

ЗМІСТ

АНОТАЦІЯ

У роботі розглядаються основні теоретичні відомості про платформу .NET, її особливості, переваги та недоліки. Було досліджено сферу використання технології, особливості роботи з базами даних, інструменти для веб-розробки, фреймворки для побудови застосунків.

У практичній частині було розроблено веб-сайт для допомоги українцям, постраждалим в наслідок воєнних дій на території країни, використано кросплатформені фреймворки: EF Core для взаємодії з базою даних, ASP.NET Core для побудови веб-застосунку, СКБД SQL Server.

ВСТУП

Інформаційні технології займають все більш значущу роль у нашому житті, проникаючи у всі його сфери.

Веб-розробка є одним з найбільш активно зростаючих напрямків. Розвиток технологій, глобальні пандемії, цифровізація – усе це активно сприяє розвитку напрямку. Навіть маленькі організації прагнуть мати свій власний веб-сайт, адже сьогодні це є одним з основних елементів для побудови іміджу компанії.

На сьогодні існує широкий вибір інструментів та засобів веб-розробки. Одним із популярних рішень є платформа .NET, розроблена компанією Microsoft. На початку свого існування .NET Framework була розрахована на роботи саме з операційними системами сімейства Windows, проте згодом було випущено модульну платформу .NET Core, сумісну з іншими операційними системами, що дозволило відкрити багато нових напрямків використання технології, збільшило зацікавленість розробників та представників великого бізнесу.

Розробка програмного продукту для допомоги постраждалим в наслідок воєнних дій є дуже актуальною в наші дні, оскільки кілька мільйонів українців були вимушені покинути свої домівки через військові дії, деякі з них мають серйозні проблеми з фінансами. Застосунок ставить на меті допомогти усім українцям з пошуком нового житла, будь-яких речей, що можуть стати у нагоді у побуті, волонтерів, що допоможуть дістатися з однієї точки до іншої. Таким чином, програма матиме функціонал, що має допомогти постраждалим задовольнити найактуальніші потреби.

Мета курсової роботи: дослідити засоби та методи та підходи для розробки веб-застосунків за допомогою платформи .NET, використати отримані навички для розробки програмного продукту для допомоги українцям.

Робота включає в себе три розділи.

1. ПЛАТФОРМА .NET. ОСОБЛИВОСТІ, РОЗВИТОК, ПЕРСПЕКТИВИ.

1.1 Зародження та розвиток .NET

Зародження платформи .NET почалося наприкінці 90-х років минулого століття. Засновник компанії Microsoft Білл Гейтс сказав: «Ми закладемо основи для інтернету нового покоління». Основною метою розробки було прагнення охопити усі існуючі та майбутні програмні рішення компанії для надання доступу до мережі інтернет. На початку проект отримав назву «Next Generation Windows Services», проте, поставлені цілі не були досягнені і ім'я було змінено на .NET Framework. Перша версія була випущена у 2002 році, у ній було додано інструменти для веб-розробки. З кожною наступною версією додавалося багато нового різностороннього функціоналу: підтримка нових протоколів, типів та структур даних, бібліотек для розробки веб-застосунків та програм під управлінням ОС. У 2007 році компанія заявила, що вихідні коди платформи стануть доступними для усіх. Проте увесь цей час лишався один недолік – неможливість роботи на операційних системах відмінних від Windows, тому у 2014 році було анонсовано .NET Core – кросплатформну перебудовану версію .NET Framework, яка побачила світ у 2016 році. Разом з тим паралельно продовжував розвиток .NET Framework. Проте, станом на зараз, Microsoft вирішила об'єднати .NET Framework та .NET Core у новий проект, який отримав скорочену назву – просто .NET.

1.2 Особливості платформи .NET

За більше ніж 20 років розвитку платформа знайшла своє застосування у багатьох напрямах, головним та найпопулярнішим з них наразі є веб-розробка. Відповідно до інформації, наданої компанією Microsoft, .NET 6 має базові бібліотеки, програмні інтерфейси для:

1. Побудови веб-застосунків за допомогою фреймворку ASP.NET Core, Blazor
2. Cloud-Native розробки

3. Розробки desktop застосунків за допомогою фрейм WPF, WinForms
4. Розробки мобільних застосунків, використовуючи фреймворк Xamarin
5. Розробки ігор за допомогою Unity
6. IoT розробки
7. Модельного машинного навчання за допомогою бібліотеки ML.NET

Однією з особливостей платформи є пакетний менеджер NuGet, що дозволяє легко та швидко встановлювати потрібні пакети без використання будь-яких сторонніх сервісів.

Microsoft активно розвиває та підтримує інтегровану середу розробки Visual Studio, що має величезний список інструментів для побудови різних застосунків використовуючи .NET.

.NET базується на об’єктно-орієнтовному програмуванні, що дозволяє розбивати програми на менші частини, якими легше управляти.

.NET Core є модульним, що дає багато гнучкості при розробці, він може бути встановлений як частина застосунку або ж потребувати окремої інсталяції. На одній машині можуть бути встановлені різні версії, кожна з яких буде використовуватися для окремого проекту.

.NET має величезну спільноту, отже майже будь-яка проблема може бути вирішена за короткий проміжок часу. Компанія Microsoft пропонує безліч курсів, відео, документації, що повинні допомогти у вивченні основ середовища та вирішенні головних проблем.

.NET підтримує багато мов програмування, основною з яких є C#.

1.3 Мова програмування C#

Перша версія мови C# вийшла у 2002 році, за своєю структурою та дизайном мова була схожа на Java. Microsoft характеризувала її як «універсальна, проста та сучасна об’єктно-орієнтовна мова».

З плином часу та виходом нових версій, C# ставав все більш унікальним, отримуючи певні особливості:

1. Query Expressions – набори інструкцій, що дозволяють працювати з колекціями даних за схожим з SQL синтаксисом.
2. Делегати – вказівники на методи, що допомагають реалізувати пізнє зв’язування.
3. Анонімні типи дозволяють інкапсулювати набори властивостей, доступних тільки для читання, у окремі об’єкти без необхідності визначення нового типу.
4. Auto-implemented properties роблять код біль стислим та лаконічним, якщо методи доступу до властивостей не потребують додаткової логіки.
5. Extensions methods дозволяють додавати методи до існуючих типів, не створюючи при цьому похідних типів
6. Асинхронне програмування за допомогою async/await.

І це далеко не весь список особливостей мови, з кожною новою версією C# надає розробникам можливості писати все більш лаконічний та гнучкий код одночасно. Завдяки цьому, C# активно використовується у багатьох сферах від веб-розробки до розробки ігор та машинного навчання.

2. РОБОТА З БАЗАМИ ДАНИХ ТА ВЕБ-РОЗРОБКА ЗА ДОПОМОГОЮ .NET

2.1 Технологія ADO.NET

ADO.NET є технологією, що дозволяє отримати доступ управління даними, які зберігаються у базах даних або інший джерелах.

Для підключення до бази даних спочатку треба визначити строку підключення, у якій вказати назву серверу, бази даних, до якої маємо підключитися, також треба вказати відповідний логін та пароль. Рядок підключення може також містити інші параметри, які будуть налаштовувати інші параметри по типу аутентифікації, періоду встановлення підключення, шифрування.

Після визначення рядку підключення, його можна відкрити, використовуючи клас SqlConnection з простору імен Microsoft.Data.SqlClient. Слід використовувати конструкцію using, яка дозволяє автоматично закривати підключення по завершенню блока коду, який працює з базою даних, таким чином ми контролюємо процес звільнення ресурсів та коректності роботи з базою.

Для взаємодії з базою даних через команди використовується клас SqlCommand, який приймає на вхід підключення та текст команди. Результат команди можна отримати за допомогою вихідних параметрів або класу SqlDataReader, який використовується для читання даних.

Загалом, функціонал класу SqlCommand надає багато можливостей для отримання таблиць, скалярних значень, додавання та виконання збережених процедур, транзакцій.

Ще одним класом способом взаємодії з базою даних є клас DataSet з простору імен System.Data, який представляє таблиці у вигляді відповідних колонок та рядків.

Проте, нам не завжди зручно взаємодіяти з базами даних у такий спосіб. Написання чистих SQL запитів іноді є нелегким процесом та може стати причиною багатьох помилок. Також код програм стає доволі громіздким та

заплутаним. В багатьох випадках ми можемо скористатися надбудовою над ADO.NET під назвою Entity Framework, що надає можливості для зв’язування об’єктів C# та бази даних, налаштування зав’язків, таблиць, обмежень прямо у коді програми.

2.2 Entity Framework

Entity Framework – це ORM фреймворк, використовуючи який, ми можемо абстрагуватися від прив’язки до бази даних та працювати з даними як з об’єктами відповідних класів, незважаючи на тип сховища. Entity Framework підтримує багато різних СКБД, маючи вбудовані провайдери для кожної з них.

Фреймворк надає можливості для роботи з існуючими базами даних, та створенням нових на основі відповідних класів та налаштованих у коді зв’язків, також є можливість наповнення новоствореної бази початковими даними.

Незважаючи на описаний функціонал, ми також можемо працювати з базою даних напряму, використовуючи «сирі» SQL запити.

Взаємодія з базою даних відбувається через клас контексту даних, який наслідується від класу DbContext. Клас контексту надає основні можливості для роботи з базою: налаштування конфігурації бази даних, зав’язків та обмежень, властивостей підключення, перевірку наявності бази, створення бази даних за її відсутності. За допомогою методу контексту SaveChanges() ми зберігаємо зміни у базу даних, у результаті усі зміни, що відбулися після останнього виклику будуть записані до бази даних. Використовувати клас контексту слід за допомогою конструкції using, аналогічно до SqlConnection у чистому ADO.NET.

Для кожної таблиці в базі ми маємо створити відповідний клас або кілька класів, які називають Entity. Для використання цих Entity у класі контексту треба створити відповідні поля DbSet, кожне з яких представляє набір об’єктів таблиці у базі даних.

Entity Framework підтримує 3 типи завантаження зв’язних даних:

1. Lazy loading – дані завантажуються в контекст тоді, коли ми звертаємось до навігаційних властивостей.
2. Eager loading – дані завантажуються із першим зверненням за допомогою методу Include()
3. Explicit loading – дані завантажуються за допомогою методу Load() у потрібний момент

2.3 Налаштування зав'язків та обмежень у Entity Framework

Для налаштування зав'язків та обмежень у Entity Framework використовуються механізми анотації та Fluent Api.

Для використання Fluent Api використовується перевизначений метод OnModelCreating класу контексту. За допомогою FluentApi ми можемо налаштувати зв'язки між таблицями:

1. Один-до-одного за допомогою методів HasOne(), WithOne()
2. Один-до-багатьох за допомогою методів HasOne(), WithMany()
3. Багато-до-багатьох за допомогою методів HasMany(), WithMany()

Для встановлення зовнішнього ключа використовується метод HasForeignKey().

Зв'язки між таблицями бувають трьох типів:

1. Повністю визначені, в такому випадку обидва класи містять посилання на інший, для зв'язків багато-до-багатьох мають бути присутні відповідні колекції об'єктів у кожному класі, для зв'язків один-до-багатьох з одного боку клас має містити колекцію, з іншого одиничний об'єкт та зовнішній ключ.
2. Без зовнішнього ключа. Використовується у зв'язках один-до-багатьох, відрізняється від попереднього відсутністю зовнішнього ключа.
3. Однонаправлені, в такому випадку тільки одна з сторін має посилання на іншу.

FluentApi також містить методи для визначення обмежень таблиць, за допомогою відповідних методів ми можемо вказати обов’язковість поля, назву таблиці, колонки (за замовчуванням використовується відповідна назва класу та поля), обмеження щодо довжини поля, діапазону допустимих значень, індекси.

Використання анотацій дозволяє налаштувати обмеження рядків та таблиць безпосередньо у коді відповідного класу, проте має вужчий функціонал і стосується лише об’єкта над яким була додана анотація. Наприклад, анотація [Key] та виклик Fluent Api методу HasKey() мають однаковий вплив та вказують, яке поле є ключем сутності, проте якщо сутність має складений ключ, то використати анотацію ми не можемо, в такому випадку викликається метод HasKey(), у який в якості аргументів передаються поля, які мають увійти до складу ключа.

2.4 LINQ у Entity Framework

Вибірку з бази даних та взаємодію з нею можна використовувати за допомогою технології LINQ, яка дозволяє отримувати дані простим способом, використовуючи вирази близькі за формою до SQL. Уся простота полягає у тому, що ми можемо писати будь-які запити, використовуючи лише відповідні методи колекцій.

LINQ дозволяє нам працювати з вибіркою, фільтрацією, групуванням, об’єднанням таблиць. Також реалізовано методи для представлення агрегатних функцій, різниці множин, їх перетину та об’єднання.

За замовчуванням усі запити, що повертають об’єкти класів є відслідковуваними, тобто дані розміщуються у кеші. Усі зміни з кожним об’єктом запам’ятовуються та будуть внесені до бази даних після виклику методу збереження у класі контексту. Проте, іноді нам не треба змінювати отримані дані, для збереження ресурсів та упевненості у незмінності даних можна використовувати метод AsNoTracking(), який сигналізує, що дані не треба кешувати. Будь-які зміни до таких об’єктів не матимуть жодного впливу на базу

після виклику методу збереження, оскільки дані не додавалися в контекст при завантаженні.

Якщо ж дані відслідковуються, то при виконанні нового запиту результуючі дані будуть додані до контексту тільки в тому випадку, якщо до цього не були присутні в ньому.

LINQ використовує відкладене виконання, тобто при створенні запиту він не виконується негайно. Виконання запиту відбувається тільки коли програма доходить до точки, яка потребує даних з запиту або якщо результатом запиту є скалярна величина.

При виклику запиту LINQ ми можемо отримати результат у колекціях, що представляють інтерфейси IEnumerable та IQueryable. Якщо ми позначемо результат як IEnumerable, то усі дані будуть спочатку завантажені в пам'ять, а фільтрація відбудеться уже безпосередньо там, на відміну від цього IQueryable надає віддалений доступ до бази даних, таким чином запит до бази даних формується з усіх LINQ запитів, коли ми доходимо до точки, у якій нам потрібен результат запиту, відбувається створення відповідного SQL запиту, який додатково оптимізується та повертає уже готовий результат, тобто уся фільтрація відбувається безпосередньо у базі даних. Такі запити потребують менше пам’яті та пропускної здатності мережі, проте можуть виконуватися повільніше.

2.5 Вибір методу взаємодії з базою даних

Отже, було розглянуто технологію ADO.NET та її надбудову Entity Framework. Для більшості сучасних рішень використання Entity Framework повністю задовольнить усі потреби, пришвидшить написання та читабельність коду, зробить структуру проекту більш лаконічною. Зручні механізми налаштування зв’язків та обмежень значно пришвидшать роботу у разі, якщо база даних ще не була створена, оскільки вона буде створена автоматично. Механізм LINQ дозволяє працювати з колекціями об’єктів з бази даних та майже повністю усуває потребу у написанні чистих SQL запитів.

Таким чином, Entity Framework задовольняє більшість сучасних потреб та може бути використаний у багатьох рішеннях, існують й інші фреймворки від сторонніх розробників, проте вони поступаються функціоналом, зручністю та якістю.

2.6 Фреймворк ASP.NET Core

ASP.NET Core – це фреймворк для розробки веб-застосунків на платформі .NET. Його особливостями є модульність, легкість та простота конфігурації. Оскільки фреймворк розвивається компанією Microsoft, то ми можемо використовувати увесь фунціонал платформи .NET у повному обсязі, зокрема зручну зв’язку з Entity Framework, розглянутому у попередньому розділі.

Вхідною точкою у застосунку, його головним класом є файл Program.cs. При створенні шаблонного проекту, він вже має у собі базовий функціонал для тестового запуску. Спочатку створюється об’єкт будівельник за допомогою методу WebApplication.CreateBuilder(), потім на ньому викликається метод Build(), який власне створює екземпляр класу WebApplication, останнім є виклик методу Run() створеного об’єкту для запуску застосунку. Через об’єкт WebApplicationBuilder ми можемо отримати доступ до багатьох властивостей: інформації про середовище, налаштувань хоста, конфігурації, налаштувань логування, сервісів.

В процесі роботи застосунку кожен запит проходить через конвеєр так званих middleware компонентів. В процесі проходження запиту по елементам middleware може формуватися деяка відповідь або проводитися аналіз запиту. Наприклад вбудовані middleware для підтримки сесій, аутентифікації та авторизації. В з них ми можемо отримати усю інформацію про запит та відповідь.

2.7 Dependency Injection в ASP.NET Core

Механізм Dependency Injection дозволяє зробити елементи програми, що взаємодіють слабозв'язаними. Об'єкти зазвичай взаємодіють між собою через абстракції, що надає більше гнучкості та адаптованості. Також цей механізм допомагає налаштувати життєвий цикл певних компонентів. Вбудований механізм ASP.NET Core має три типи реєстрації залежностей:

1. Transient залежності створюються під час кожного нового звернення до сервісу, тобто за один запит може певні об'єкти можуть бути створені кілька разів.
2. Scoped залежності створюються по одному екземпляру на кожний запит
3. Singleton об'єкти створюються при першому зверненні до сервісу та живуть протягом усього часу роботи програми.

Розробникам варто розуміти, який з типів використовувати для кожного сервісу, адже неправильний вибір може призвести до уповільнення роботи програми або ж певних помилок.

Отримати залежності можна багатьма способами: через конструктори класів, через вищезгадану властивість об'єкту WebApplication, через параметр методу у middleware, через властивість RequestServices контексту запиту. Механізм автоматично підставляє потрібний сервіс коли виникає потреба в його використанні.

2.8 Конфігурація

Важливу роль у процесі розробки та запуску програми грає його налаштування: вказання параметрів логування, налаштування підключення до бази даних, параметри середовища, у якому запущена програма. У ASP.NET Core можна отримувати параметри конфігурації з:

1. Аргументів командної стрічки
2. Змінних середовища
3. Об'єктів в пам'яті

4. Файлів (JSON, XML…)
5. Хмари Azure
6. Своїх власних джерел, під які має бути створено провайдер

Зазвичай, основна конфігурація застосунку вказується у файлі appsettings.json.

2.9 Маршрутизація

Важливою частиною розробки програми є налаштування маршрутизації. Вхідні запити мають співставлятися з маршрутами для обрання елементу програми, що буде оброблювати запит. Об’єкт WebApplication містить у собі методи Map…(), які використовуються для опису кінцевих точок, що будуть оброблювати запит по певному маршруту. У параметрах методу ми можемо вказати патерн маршруту, обмеження вхідних параметрів, їх обов’язковість. При використанні контролерів більш зручним способом є маршрутизація за допомогою атрибутів.

За допомогою використання відповідних методів сімейства Map або атрибутів маршрутизації ми можемо вказати, які саме методи може оброблювати даний елемент.

В середині контролеру ми отримуємо доступ до усіх параметрів маршруту, можна явно указати, звідки саме беруться дані на вхід: з форми, з тіла запиту, зі стрічки запиту.

Результатом обробки запиту певною точкою може бути виклик певного методу або ж певні дані, для повернення яких використовується ResultsApi, який надає доступ до повернення багатьох різних видів результатів: файлів, HTML сторінок, даних у різних форматах, статусних кодів, переадресація.

Зазвичай у контролерах ми використовуємо певні сервіси, що реалізують деяку бізнес логіку, отримати доступ до них можна за допомогою механізму Dependency Injection.

2.10 MVC та Razor

Частиною фреймворку ASP.NET Core є фреймворк ASP.NET Core MVC, який використовує патерн MVC, дана концепція розділяє програму на три компоненти:

1. Моделі – описують дані, що використовуються для взаємодії з базою даних, також можуть містити логіку щодо валідації даних.
2. Представлення – відповідають за візуальну частину та користувацький інтерфейс
3. Контролери – створені для взаємодії користувача з застосунком, пов’язують моделі та представлення, містять логіку обробки запитів.

Для реалізації представлень в ASP.NET Core MVC використовуються сторінки Razor, що мають розширення .cshtml, тобто відбувається рендеринг сторінок на стороні серверу. Razor сторінки можуть містити у собі звичайний HTML та конструкції C#. Як правило, для кожного контролеру в проекті створюється підкаталог у папці Views, який називається по імені контролера. За допомогою Razor, ми можемо використовувати будь-які конструкції мови C# у коді HTML: умовні конструкції, цикли, створювати об’єкти, отримувати доступ до певних полів.

Передавати дані з контролера у представлення можна трьома способами:

1. Через модель представлення
2. Через ViewBag
3. Через ViewData

ViewBag та ViewData схожі за своєю роботою, обидва представляють словник з пар ключ-значення, проте через ViewData ми можемо передавати лише прості об’єкти по типу int, bool, string, а ViewBag може містити і складніші структури даних та будь-які інші об’єкти класів. Також ми можемо передавати дані у більш безпечний спосіб через модель представлення. Для цього в

контролері у метод View() передається параметром об’єкт, а у представленні відповідно встановлюється його тип за допомогою директиви @model.

За допомогою директиви @inject ми можемо отримати будь-які зареєстровані за допомогою Dependency Injection сервіси та використовувати їх методи у коді на сторінці.

2.11 Можливості ASP.NET Core

Загалом вбудовані функції фреймворку покривають більшість потреб для розробки сучасних веб-застосунків. Фреймворк має широкий набір функціоналу для реалізації обробки запитів, ведення логів, аутентифікації та авторизації. ASP.NET Core має зручні інтерфейси взаємодії з іншими фреймворками на кшталт Entity Framework. Розробники мають можливість писати як WEB API застосунки для взаємодії з окремим клієнтським застосунком за допомогою REST, так і використовувати механізми Razor для генерації сторінок на стороні серверу.

3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Аналіз проблеми

З початком повномасштабної війни в Україні тисячі людей втратили своє житло, мільйони стали вимушеними переселенцями. Багато українців втратили роботу та не мають грошей на задоволення банальних проблем: переміщення по країні, покупка нових речей, продуктів.

Метою розроблюваного застосунку є створення платформи для допомоги постраждалим внаслідок війни. Зареєстровані користувачі зможуть розміщувати оголошення щодо допомоги з житлом, транспортом та речами. Ті, хто потребує допомоги, зможуть переглядати оголошення та інформацію про особу, що його розмістила, матимуть доступ до контактної інформації особи, зможуть задати будь-які питання на платформі у чаті. Платформа надаватиме зручний та простий інтуїтивно зрозумілий інтерфейс, що допоможе швидко знайти потрібне. Алгоритми фільтрації та пошуку допомагатимуть пришвидшити та оптимізувати пошук.

3.2 Вибір засобів розробки

Для написання застосунку було обрано один з популярних стеків для веб-розробки. Використано останні версії фреймворків для отримання можливостей використовувати найсучасніші підходи та засоби розробки. В процесі використовувалися наступні технології та засоби:

1. Платформа .NET 6 та десята версія мови C#
2. СКБД MS SQL Server для управління базою даних
3. SQL Server Management Studio для перегляду та налагодження бази даних
4. ORM Entity Framework Core для побудови взаємодії з базою даних
5. Фреймворк ASP.NET Core MVC для створення веб-застосунку з серверним рендерингом за допомогою механізму Razor
6. IDE Rider від JetBrains для написання коду

Обраний стек технологій надає можливість швидко розробляти, запускати та тестувати застосунки, адже усі технології розробляються та підтримуються компанією Microsoft, а інтегрована середа розробки Rider створена спеціально для платформи .NET.

Проект розроблювався з використанням тришарової архітектури, такий спосіб дає змогу зробити програму більш модульною та надає можливості для розробки кожної частини паралельно. Кожен з шарів має свою певну роль:

1. Data Access Layer зберігає інформацію про моделі, які описують використовувані сутності, також тут міститься клас контексту даних, репозиторії для взаємодії з базою даних та зв’язку з наступний рівнем
2. Business Logic Layer містить у собі набір сервісів для обробки отриманих даних, обчислень, фільтрацій. Він отримує дані від представлення та передає до рівня доступу до даних.
3. Presentation Layer відповідає з взаємодію з користувачем. Тут налаштовуються параметри веб-застосунку, описуються алгоритми обробки запитів. У випадку використання MVC тут розміщуються контролери, які оброблюють конкретні запити та представлення з усіма додатковими бібліотеками та компонентами.

3.3 Створення та робота з базою даних за допомогою Entity Framework Core

Оскільки застосунок розроблювався з нуля, було прийнято рішення щодо вибору Code-First підходу до розробки бази даних, тобто спочатку пишеться код відповідних сутностей, алгоритми їх взаємодії та обмежень. Потім за допомогою механізмів Entity Framework Core при створюється база даних за схемою, яку ми написали у коді. Цей підхід дозволяє скоротити час при розробці застосунку з нуля.

Класи моделей виглядають як звичайні C# класи, кожне поле співставляється з відповідною колонкою в базі даних (Рисунок 3.1). На даному прикладі зображено клас Item, що описує сутність речі, оголошення про яку можна виставити. Як

бачимо клас не містить жодних атрибутів, навіть атрибут Key можна опустити, оскільки за замовчуванням колонка з назвою Id стає первинним ключем. Усі поля, що представляють не nullable типи є обов'язковими та помічаються NOT NULL у базі даних. Також на даному прикладі демонструється використання навігаційних властивостей. Клас речі містить посилання на колекцію зображень, що ілюструє зв'язок один-до-багатьох, також є посилання на клас користувача, але вже з іншого боку.

```
public class Item
{
    [Key]
    6 usages
    public Guid Id { get; set; }
    7 usages
    public string Name { get; set; }
    7 usages
    public string Description { get; set; }
    7 usages
    public bool New { get; set; }
    6 usages
    public string Location { get; set; }
    7 usages
    public DeliveryType DeliveryType { get; set; }
    17 usages
    public IEnumerable<ItemImage> Images { get; set; }
    18 usages
    public User Owner { get; set; }
}
```

Рисунок 3.1 – структура класу моделі

Налаштування зв'язків відбувається у перевизначеному методі OnModelCreating() (Рисунок 3.2). Як бачимо, за допомогою методів HasOne(), HasMany(), WithOne(), WithMany() встановлюються зв'язки один-до-багатьох та багато-до-багатьох, якщо це повністю визначені зв'язки, то додатково встановлюється зовнішній ключ за допомогою методу HasForeignKey().

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    builder.Entity<Chat>().HasMany(navigationExpression: e :Chat => e.Users).WithMany(navigationExpression: e :User => e.Chats);
    builder.Entity<Message>().HasOne(navigationExpression: e :Message => e.Sender).WithMany().HasForeignKey(e :Message => e.SenderId);
    builder.Entity<Message>().HasOne(navigationExpression: e :Message => e.Chat).WithMany(navigationExpression: e :Chat => e.Messages);
    builder.Entity<Property>().HasOne(navigationExpression: e :Property => e.Owner).WithMany();
    builder.Entity<Item>().HasOne(navigationExpression: e :Item => e.Owner).WithMany();
    builder.Entity<Transport>().HasOne(navigationExpression: e :Transport => e.Owner).WithMany();
    builder.Entity<Property>().HasMany(navigationExpression: e :Property => e.Images).WithOne(navigationExpression: e :PropertyImage => e.Property).HasForeignKey(e :PropertyImage => e.PropertyId);
    builder.Entity<Item>().HasMany(navigationExpression: e :Item => e.Images).WithOne().HasForeignKey(e :ItemImage => e.ItemId);

}
```

Рисунок 3.2 – встановлення зв’язків

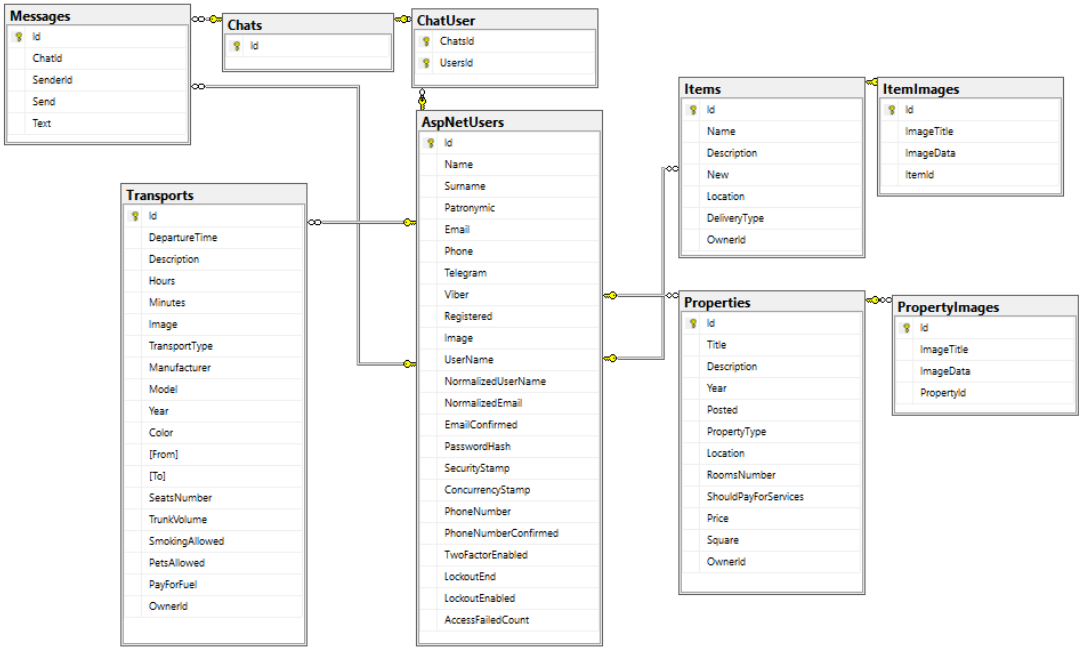

Рисунок 3.3 – схема бази даних

Оскільки в нашому випадку клас контексту наслідується на від DbContext, а від IdentityDbContext, то при генерації бази даних додатково створюються таблиці для ідентифікації користувачів та збереження їх аутентифікаційних даних. На рисунку 3.3 зображено схему бази даних, яка включає тільки основні таблиці, які використовуються у роботі застосунку, додаткові таблиці створені за допомогою Identity опущені, оскільки не використовуються на даному етапі. Як бачимо, Entity Framework успішно створив базу даних за заданою схемою, для зв’язків багато-до-багатьох було створено додаткові проміжні таблиці.

Також можна побачити, що таблиця, створена на основі класу Users, отримала назву AspNetUsers та багато додаткових полів які слугують для роботи з

аутентифікацією, авторизацією та підтвердженням акаунту користувача. Для того щоб використовувати даний функціонал, клас User був унаслідований від класу IdentityUser. Також контекст був параметризований даним класом для того, щоб зрозуміти що саме використовувати механізму Identity.

Створену базу даних можна дослідити одразу після першого запуску застосунку через розширення інтегрованої середи розробки або ж через відповідний застосунок певної СКБД, у нашому випадку це SQL Server Management Studio.

3.4 Написання бізнес-логіки

На рівні бізнес-логіки було розроблено сервіси для взаємодії з базою даних через репозиторії та допоміжні класи для трансформації інформації між інтерфейсами представлення та доступу до даних. Сервіси для взаємодії з базою даних містять набори методів для отримання відповідних даних з репозиторіїв а також методи фільтрації вибірки (рисунок 3.4). Як бачимо на прикладі, метод отримує на вхід набір параметрів, які позначаються знаком «?», тобто є nullable, оскільки у останніх версіях C# типи, що передаються за посиланням не є nullable за замовчуванням. Даний метод використовується для пошуку даних на сторінці за вказаними параметрами, якщо параметр не вказано користувачем, то він отримує значення null, метод отримує вибірку даних та використовує LINQ конструкції для фільтрації даних на повернення користувачеві.

```
public async Task<IEnumerable<Property>> GetByFilter(int? type, int? services, bool? free, string? location, string? search)
{
    var properties:IEnumerable<Property> = await _repository.GetAllAsync();
    if (type != null)
    {
        if (type == 2)
            properties = properties.Where(e:Property => e.PropertyType == PropertyType.Room);
        else if(type==3)
            properties = properties.Where(e:Property => e.PropertyType == PropertyType.Flat);
        else if(type==4)
            properties = properties.Where(e:Property => e.PropertyType == PropertyType.House);
    }

    if (services != null)
    {
        if (services == 2)
            properties = properties.Where(e:Property => e.ShouldPayForServices);
        else if (services == 3)
            properties = properties.Where(e:Property => e.ShouldPayForServices == false);
    }

    if (free is true)
        properties = properties.Where(e:Property => e.Price == 0);

    if (location != null)
        properties = properties.Where(e:Property => e.Location.ToLower().Contains(location.ToLower()));
    if (search != null)
        properties = properties.Where(e:Property => e.Title.ToLower().Contains(search.ToLower()) || e.Description.ToLower().Contains(search.ToLower()));
    return properties;
}
```

Рисунок 3.4 – метод фільтрації вибірки

Також у програмі присутні два додаткових сервіси: EnumHelper, що слугує для переведення значень enum у string для відображень у представленні, PagerUtil допомагає у пагінації, отримуючи на вхід список та розділяючи його на відповідні проміжки для кожної сторінки.

3.5 Розробка рівня презентації

Рівень презентації служить для взаємодії користувача з системою. В даному випадку було використано патерн MVC: контролери та Razor сторінки представлення.

Першим чином треба виконати налаштування програми, для цього додаємо у файл appsettings.json стрічку підключення до бази даних (рисунок 3.5)

```
{
  "ConnectionStrings": {
    "DataSource": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=H4Udb;Integrated Security=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Рисунок 3.5 – файл appsettings.json

Після цього ми зможемо використати цю стрічку для підключення до бази даних (рисунок 3.5). Отримавши стрічку за допомогою вбудованих методів, ми можемо встановити залежність контексту, передавши у неї рядок підключення, також додаємо інформацію про клас, що буде використовуватися в рамках механізму Identity.

Наступним кроком є встановлення залежностей сервісів та репозиторіїв (рисунок 3.6). Це відбувається також за допомогою вбудованого механізму. Важливим кроком є аналіз вибору життєвого циклу кожної залежності, адже якщо деякі з них пов’язані, то може виникнути ситуація, коли залежність яка потрібна іншій не буде створена на момент її потреби, в такому випадку застосунок видасть помилку.

```
var connection:string? = builder.Configuration.GetConnectionString(name:"DataSource");

services.AddDbContext<DatabaseContext>(optionsAction:options => options.UseSqlServer(connection!));
services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<DatabaseContext>();

services.AddTransient<IChatRepository,ChatRepository>();
services.AddTransient<ITransportRepository,TransportRepository>();
services.AddTransient<IItemRepository,ItemRepository>();
services.AddTransient<IPropertyRepository,PropertyRepository>();
services.AddTransient<IChatRepository,ChatRepository>();
services.AddTransient<IUserRepository,UserRepository>();

services.AddTransient<IUserService,UserService>();
services.AddTransient<IChatService,ChatService>();
services.AddTransient<IPropertyService,PropertyService>();
services.AddTransient<ITransportService,TransportService>();
services.AddTransient<IItemService,ItemService>();
services.AddTransient<IItemService,ItemService>();
```

Рисунок 3.6 – налаштування підключення до бази та Dependency Injection

Після реєстрації усіх потрібних залежностей та middleware можна перейти до розробки контролерів. Контролер є звичайний класом, який анотується спеціальним позначенням або наслідується від класу controller, через конструктор контролер може отримати залежності. Результатом повернення методу контролера може бути як HTML сторінка, так і набір певних даних, або ж нічого,

якщо його тип void. Не обов'язково кожен метод контролеру повинен відповідати певним маршрутам та приймати участь в обробці запитів, за допомогою анотацій можна позначити метод як той, що не приймає участь і є допоміжним.

На рисунку 3.7 можна побачити багато особливостей методів контролера. По-перше завдяки анотаціям було позначено методи, на які вони мають відповідати, другий методи має анотацію authorize, тому доступ неавторизованим користувачам буде заборонено. На прикладі першого методу ми можемо побачити використання параметрів запиту стрічки які отримуються як параметри методу. Також бачимо приклади використання ViewBag для відправки різних типів даних у представлення. В кінці метод відправляє у відповідь сторінку, назва якої співпадає з назвою методу, оскільки не вказана параметром, в якості моделі в даному випадку передається список.

```
[HttpGet]
2 usages  Eg Er
public async Task<IActionResult> Index(int? pageNumber,int? type, string? free, string? from, string? to, DateTime? date)
{
    List < SelectListItem > typeList = new() {
        new SelectListItem {
            Text = "Тип транспорту", Value = "1"
        },
        new SelectListItem {
            Text = "Легковий", Value = "2"
        },
        new SelectListItem {
            Text = "Автобус", Value = "3"
        },
    };
    ViewBag.From = from;
    ViewBag.Type = typeList;
    ViewBag.Free = free is "on";
    ViewBag.To = to;
    ViewBag.Date = date!=null? date.Value.ToString(format:"yyyy-MM-dd") : "";
    int pageSize = 5;
    var list:dynamic? = await _transportService.GetByFilter(from,to,type,ViewBag.Free,date);
    return View(PagerUtil.PaginatedList<Transport>.CreateAsync(list, pageIndex:pageNumber ?? 1, pageSize));
}

[HttpGet]
[Authorize]
Eg Er *
public IActionResult Create()
{
    ViewBag.Method = 0;
    return View();
}
```

Рисунок 3.7 – Методи контролера

Тип моделі вказується явно за допомогою анотації @model, як бачимо на рисунку 3.8, на сторінці використовується одночасно HTML розмітка та вставки C# коду. Перша умовна конструкція відображає кнопку якщо користувач пройшов аутентифікацію, в даному випадку використовується аутентифікація на основі cookie, тобто при заповненні форми та успішному логіні користувачеві встановлюються аутентифікаційні cookie, які потім надсилаються на сервер при кожному запиті. Також фрагменти коду можна побачити для встановлення параметру value у двох елементах input, там використовуються дані, передані за допомогою ViewBag.

```
@using H4U.BLL.Utils
@model H4U.BLL.Utils.PagerUtil.PaginatedList<H4U.DAL.Models.Transport>


@if (Context.User.Identity.IsAuthenticated)
{
    <a class="btn btn-help" asp-controller="Transport" asp-action="Create">Допомогти!</a>
}
<div class="container container-cards">
<form class="card card-styled card-padding">
    <div class="row g-3">
        <div class="form-group col-md-4">
            <input class="form-control" name="from" placeholder="Звідки" value="@ViewBag.From" type="text"/>
        </div>
        <div class="form-group col-md-4">
           <input class="form-control" name="to" placeholder="Куди" value="@ViewBag.To" type="text"/>
        </div>
```

Рисунок 3.8 – Сторінка Razor

3.6 Основний функціонал

Основний функціонал порталу полягає у наданні допомоги постраждалим та розміщенні оголошень усіма тими, хто бажає та має можливість допомогти їм.

Незареєстрованому користувачу доступний такий функціонал:

1. Реєстрація нового акаунту
2. Перегляд головної сторінки сайту
3. Перегляд категорій оголошень: транспорт, речі, нерухомість
4. Перегляд детальної інформації про розміщене оголошення та його автора
5. Фільтрація оголошень у категорії за різними параметрами

Якщо користувач спробує перейти на сторінку, яка доступна тільки для авторизованих, то він буду перенаправлений на сторінку входу у систему.

Зареєстрований користувач додатково має можливості:

1. Увійти у систему
2. Редагувати інформацію про себе
3. Створювати\редагувати свої оголошення
4. Переглядати повний список свої оголошень
5. Переглядати список своїх чатів
6. Писати повідомлення іншим користувачам

Реєстрація користувача потребує обов'язкового заповнення полів ПІБ, електронної пошти, паролю, номеру телефону. Опціонально можна вказати наявність месенджерів Viber та Telegram.

3.7 Головна сторінка. Реєстрація та вхід

Головна сторінка зустрічає користувача привітальним повідомленням та списком усіх категорій (рисунок 3.9), коли користувач не авторизований, то верхнє меню містить лише два пункти: перший для переходу на сторінку логіну, другий для переходу на певну категорію оголошень.

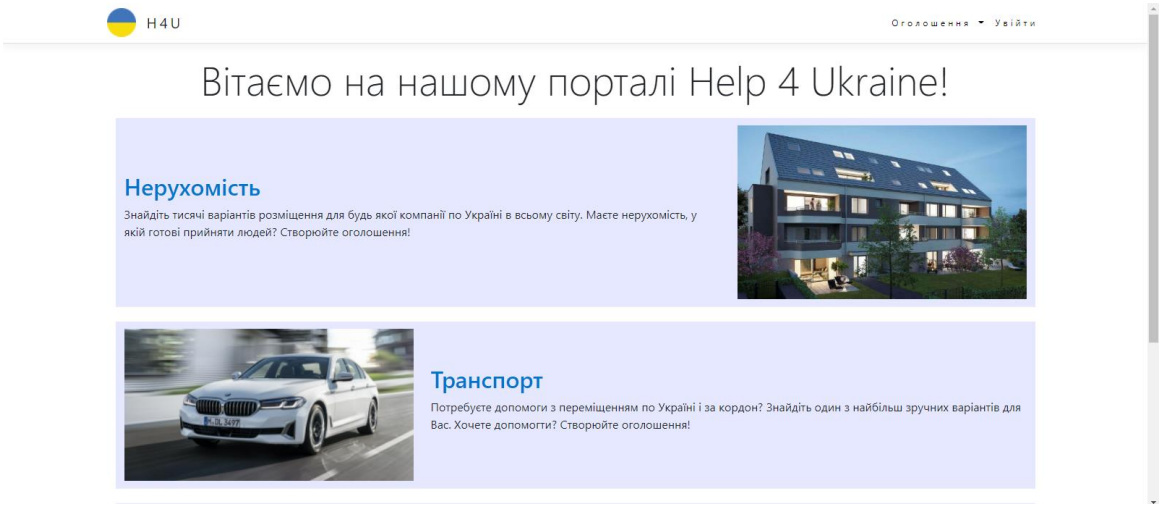

Рисунок 3.9 – Головна сторінка

Якщо користувач має акаунт, то він може увійти за допомогою логіну та паролю (рисунок 3.10), якщо не введено дані, або ж вони невірні, тоді користувач отримає відповідне повідомлення (рисунок 3.11).

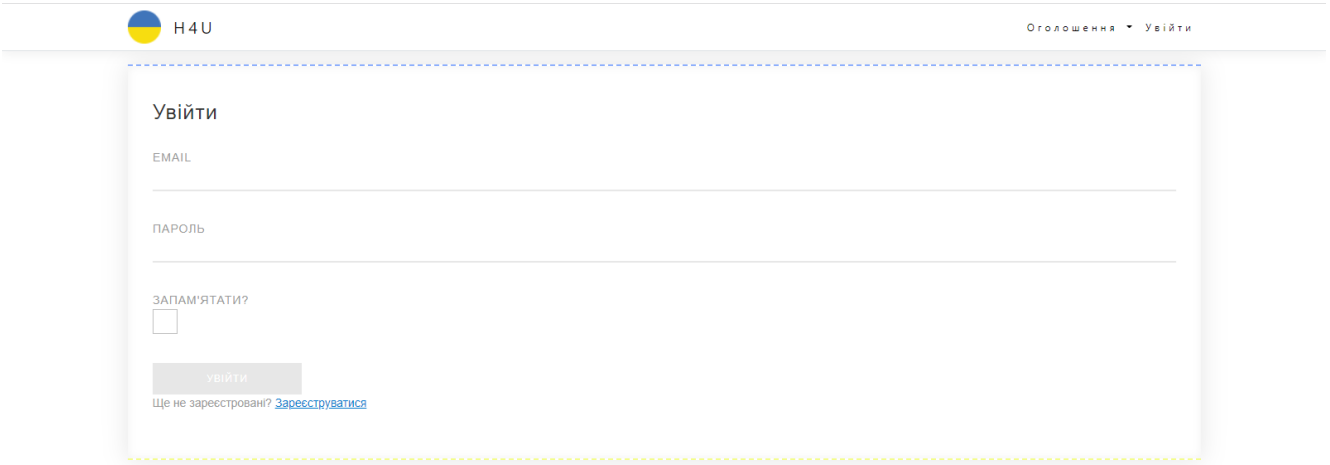

Рисунок 3.10 – Сторінка логіну

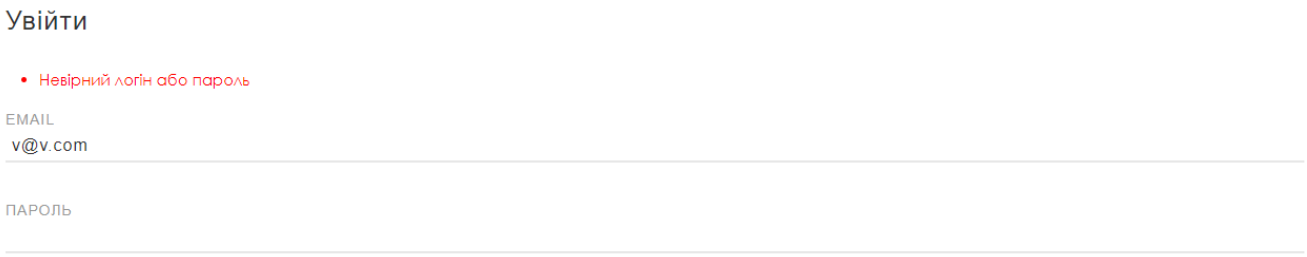

Рисунок 3.11 – Повідомлення про помилку при вході

Якщо потрібна реєстрація нового акаунту, тоді користувачеві слід натиснути відповідне посилання у низу форми.

При реєстрації (рисунок 3.12) необхідно заповнити усі поля форми, при залишенні їх пустими або введенні некоректних даних під відповідним полем з’явиться повідомлення про помилку, або повідомлення буде виведене зверху форми у випадках: акаунт з відповідною електронною поштою уже зареєстровано у системі, пароль не відповідає нормам надійності.

Якщо реєстрація пройшла успішно, то автоматично здійсниться вхід в акаунт та користувач буде перенаправлений на головну сторінку.

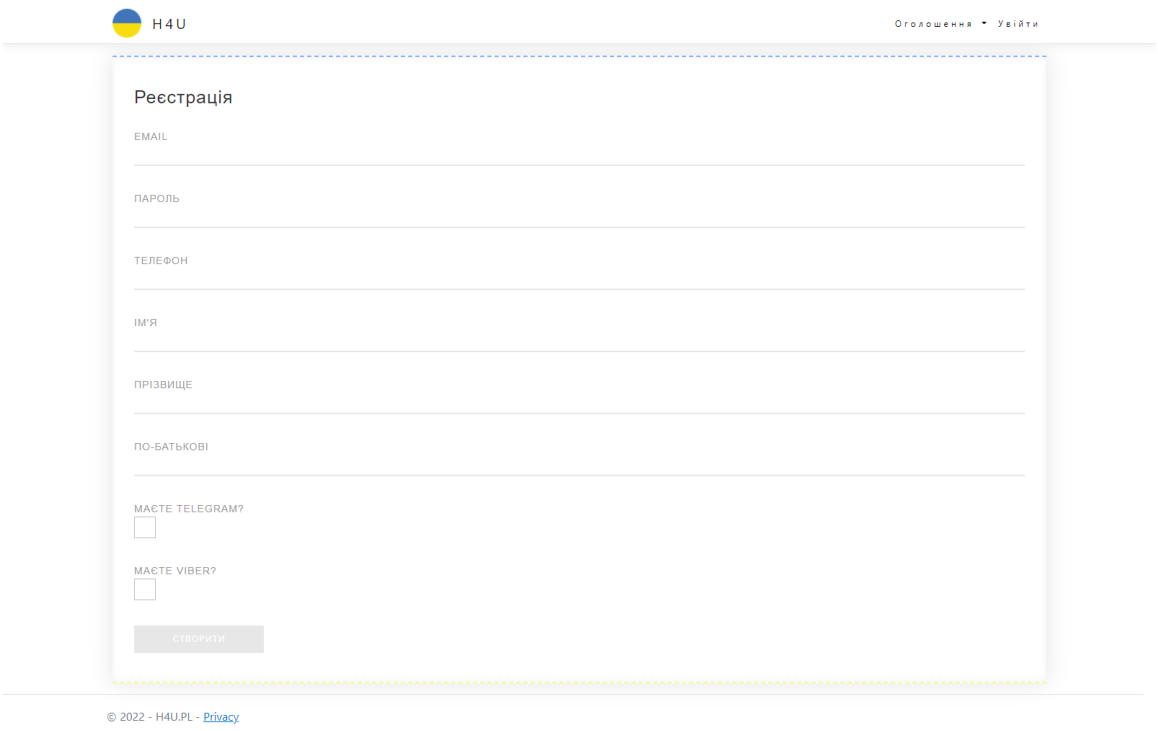

Рисунок 3.12 – Сторінка реєстрації

Верхнє меню авторизованого користувача відрізняється від звичайного, там наявний новий пункт «Мій акаунт» (рисунок 3.13), у ньому можна переглянути інформацію про свої чати та оголошення, змінити інформацію про себе.

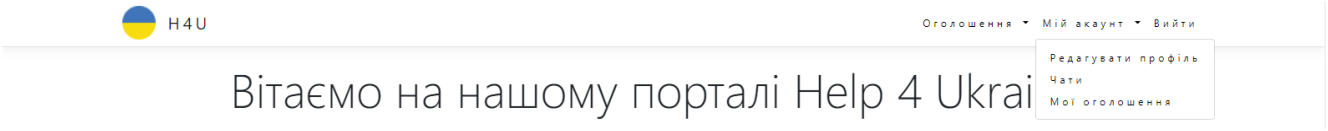

Рисунок 3.13 – Меню авторизованого користувача

3.8 Створення, редагування, перегляд оголошень, чат

Авторизовані та неавторизовані користувачі можуть переглядати список оголошень, проте авторизовані додатково мають кнопку «Допомогти», яка перенаправляє на створення оголошення у відповідній категорії. Сторінки з усіма оголошеннями виглядають доволі схожим чином, відрізняється лише набір полів, що відображається на картці та способи фільтрації (рисунок 3.14).

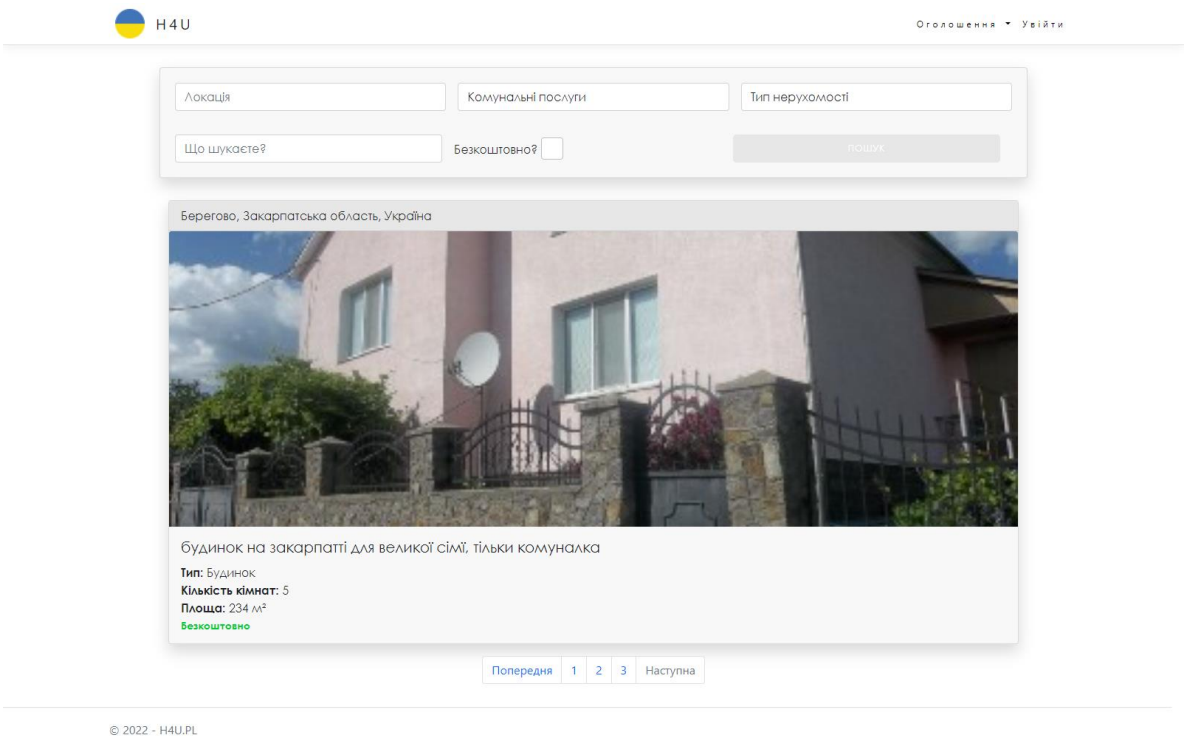

Рисунок 3.14 – Сторінка категорії нерухомості

Перегляд оголошень у категорії є зручним, оскільки завдяки механізму пагінації максимальна кількість на одній сторінці обмежена до п’яти. Також присутні механізми фільтрації даних за найголовнішими параметрами та пошук по назві та опису (рисунок 3.15).

Фільтрувати дані на сторінці нерухомості можна за:

1. Місцем розташування
2. Типом нерухомості
3. Відміткою про безкоштовність житла
4. Обов’язковістю сплати за комунальні послуги
5. Ключовими словами чи фразами в описі або назві

Фільтрувати оголошення про транспорт можна за:

1. Типом транспорту
2. Точкою відправлення
3. Точкою прибуття

4. Безкоштовністю пального
5. Датою виїзду

Оголошення про передачу речей мають такі фільтри:

1. Адреса знаходження
2. Тип доставки
3. Стан речі
4. Ключові слова чи фрази в описі або назві

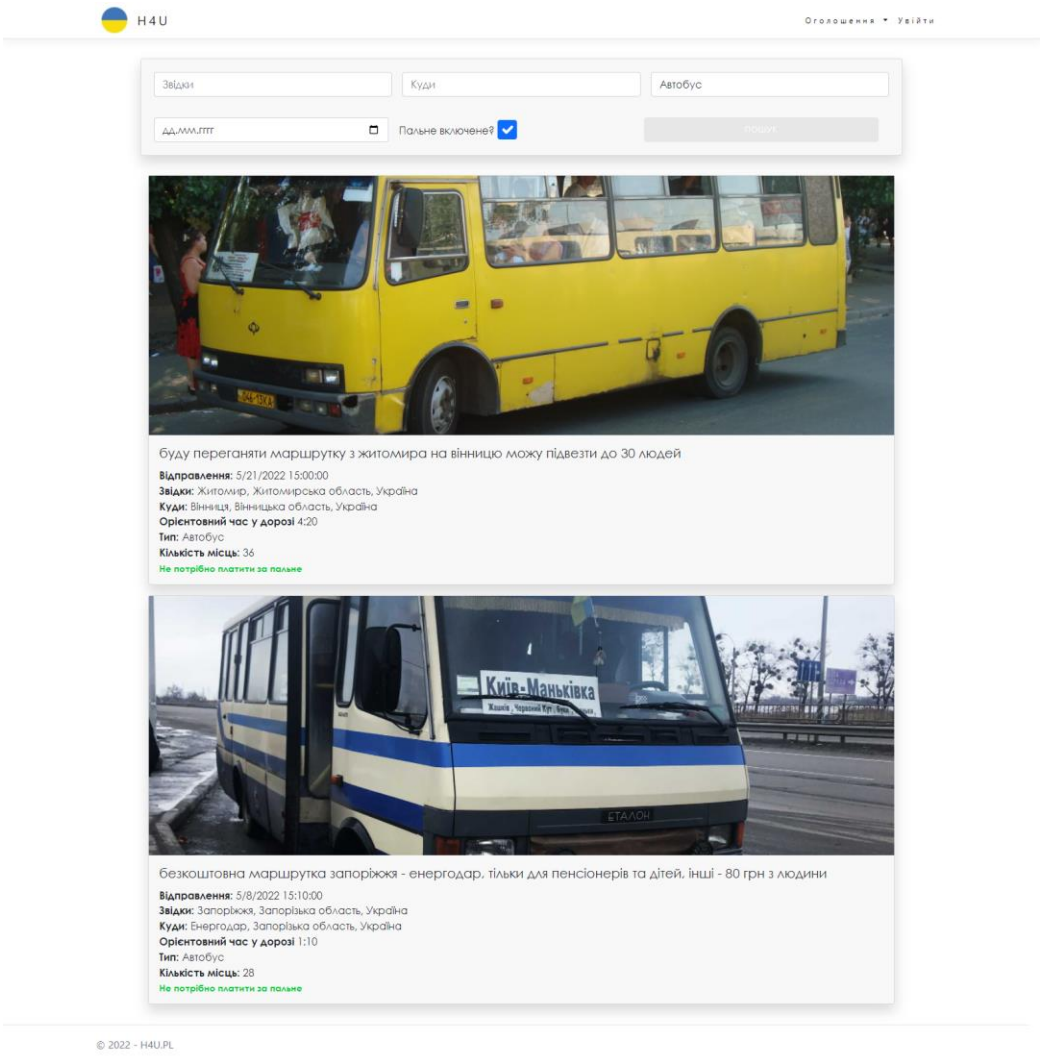

Рисунок 3.15 – Результат фільтрації

Процес створення оголошень схожий на процес реєстрації, потрібно просто правильно заповнити усі поля, у разі некоректних даних користувачу буде повідомлено про помилку. Оголошення, що потребують локацій або адрес використовують Google Maps API (рисунок 3.16) для того, щоб дані зберігалися в уніфікованому вигляді. Користувач починає вводити назву і обирає потрібний населений пункт з випадаючого списку.

Після створення оголошення воно миттєво з’явиться на сторінці з усіма оголошеннями та об’явами автора, внесення змін проходить за таким же алгоритмом як і створення та накладає такі ж самі обмеження на усі поля форми.

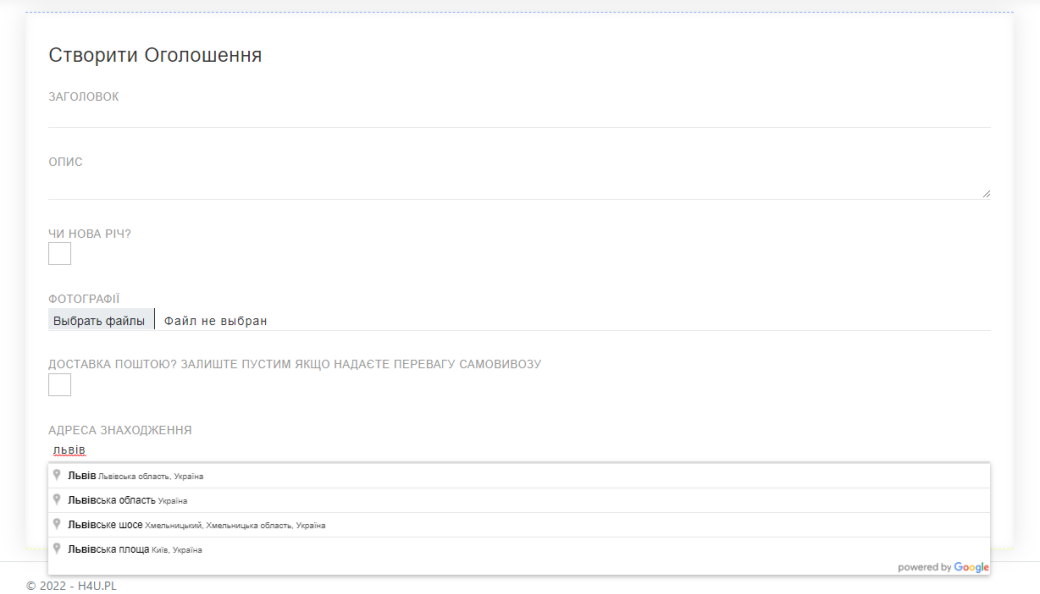

Рисунок 3.16 – Створення оголошення

При натисненні на оголошення відкривається сторінка з детальною інформацією (рисунок 3.17) на якій присутня інформація про всі поля, що були вказані при реєстрації оголошення, також знизу наявна контактна інформація про автора, з ним можна зв’язатися прямо на сайті, натиснувши кнопку «Написати».

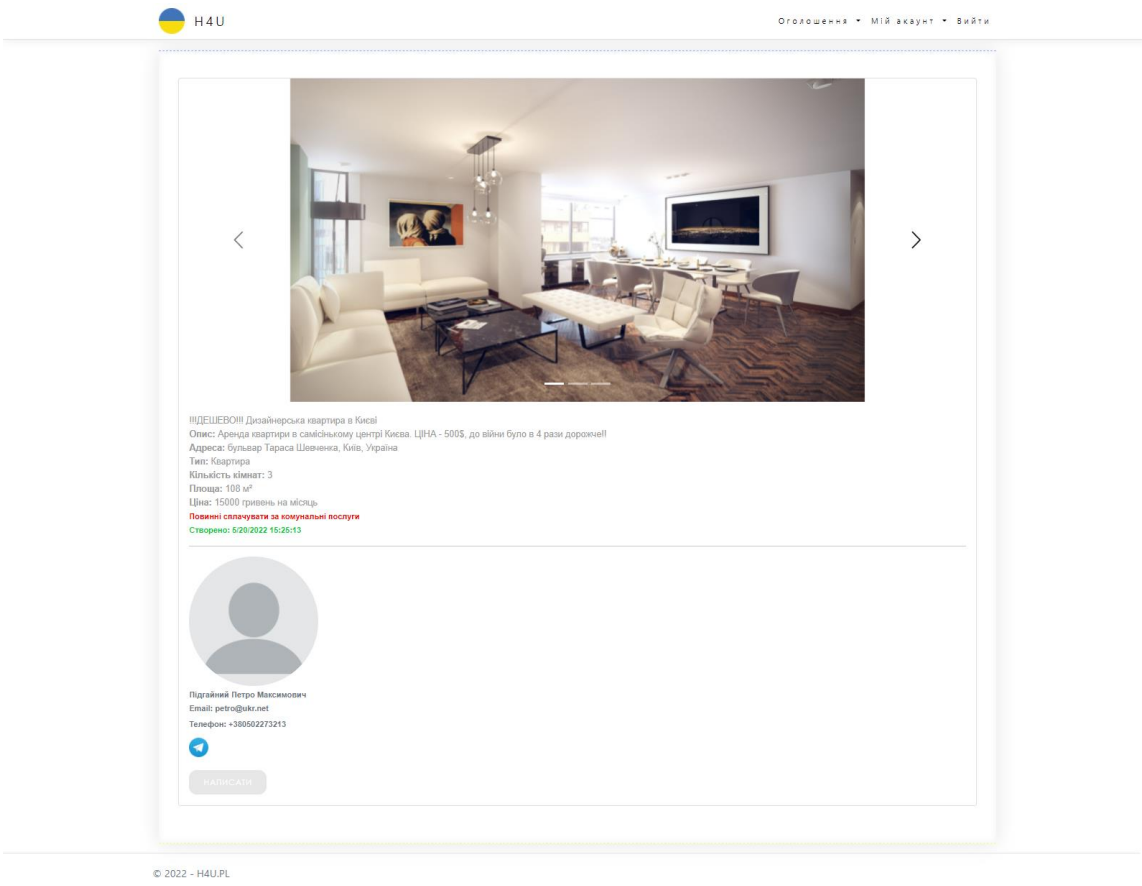

Рисунок 3.17 – Детальна інформація про оголошення

При натисненні кнопки «Написати» можливі два сценарії:

1. Якщо у вас уже був чат з цією особою, то відривається існуючий з історією повідомлень
2. Якщо не чату не було, то він створюється та відкривається

Усі повідомлення в чаті відсортовані за часом та мають мітку про дату відправки, ваші повідомлення розташовуються справа і виділені темно-синім кольором, повідомлення співрозмовника розташовані зліва та є сірими (рисунок 3.18).

Повідомлення підвантажуються без необхідності оновлення сторінки за допомогою AJAX.

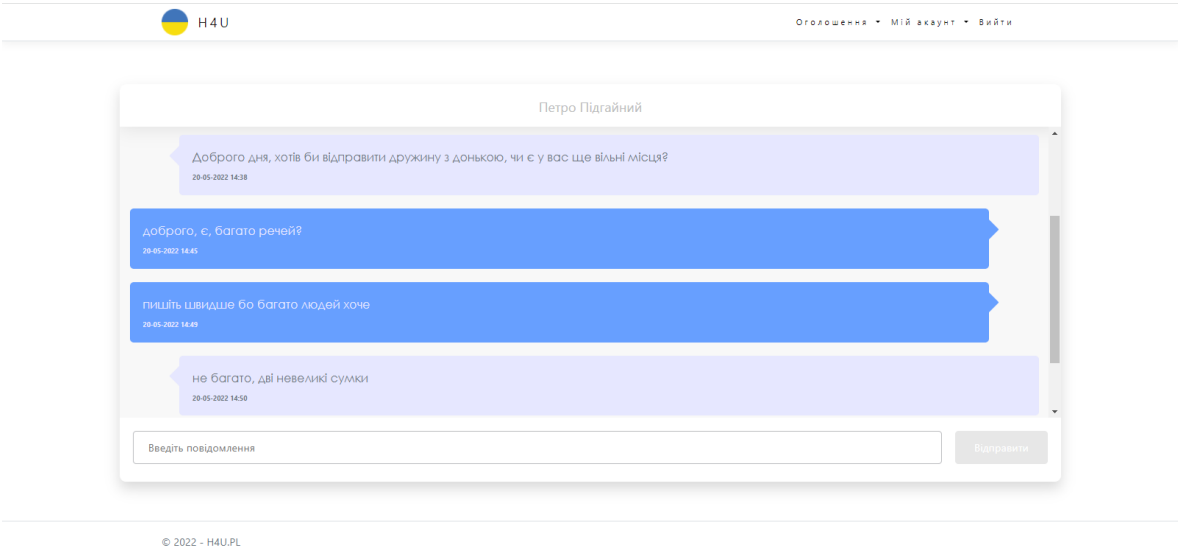

Рисунок 3.18 – Чат

Усі свої активні переписки (рисунок 3.19) можна подивитись зайшовши в пункт меню «Мій акаунт» і обравши пункт «Чати». При натисненні на один з них відкриється сторінка переписки.

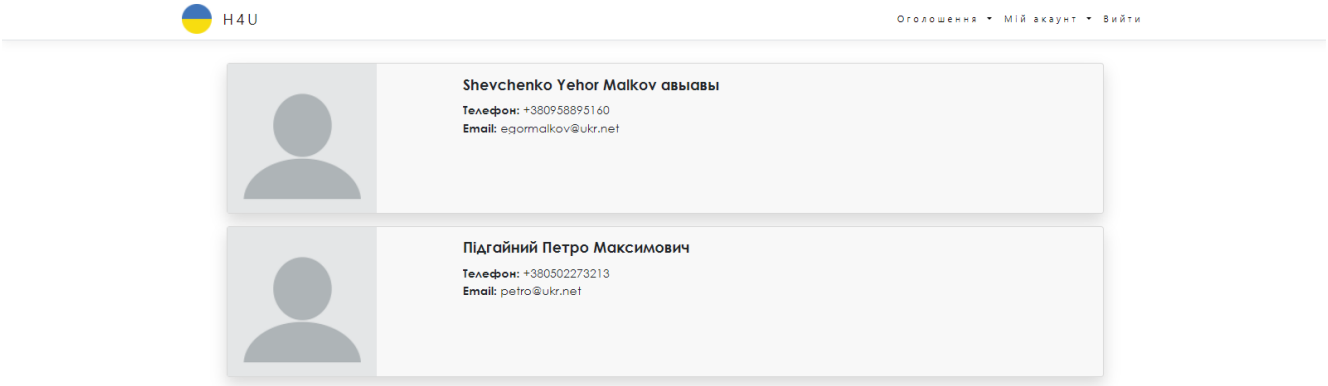

Рисунок 3.19 – Список усіх чатів

У цьому ж меню можна знайти пункт «Мої оголошення», який відкриє сторінку з усіма вашими оголошеннями (рисунок 3.20), розділеними по категоріях. При кліку на кожне з них відкривається сторінка відповідного оголошення.

На цьому основний функціонал системи завершується.

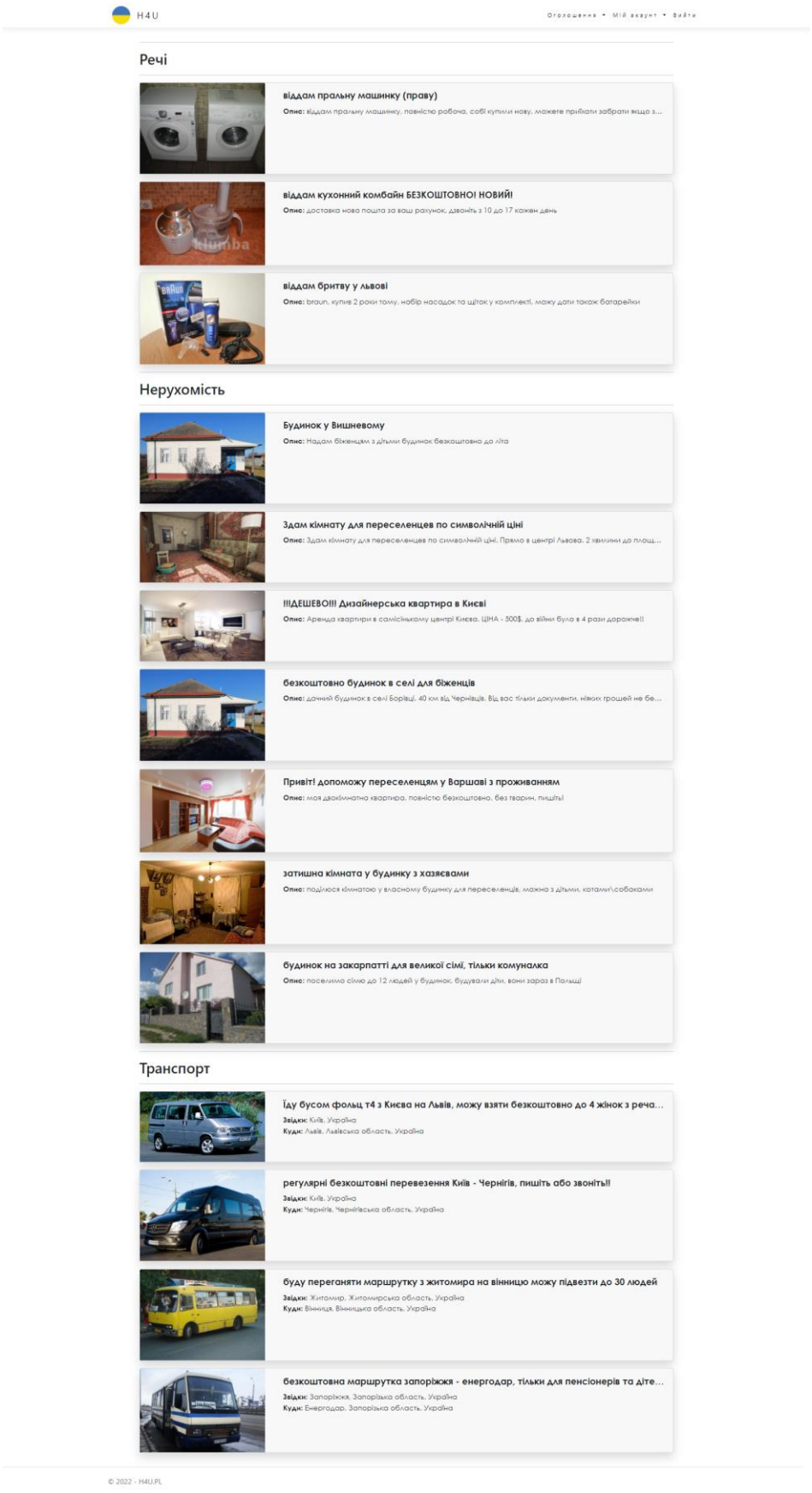

Рисунок 3.20 – Сторінка «Мої оголошення»

ВИСНОВКИ

У процесі роботи було розглянуто основні теоретичні відомості про платформу .NET та засоби побудови веб-застосунків на її основі, було розглянуто мову C#, її особливості.

У перших двох розділах розглядаються теоретичні відомості про платформу: її історія, етапи розвитку, перспективи, короткий огляд основної мови C#. Також було досліджено ключові фреймворки для розробки сучасних веб-застосунків:

1. Entity Framework – фреймворк для зручної роботи з базами даних, що надає можливості реалізації Code First підходу, налаштування обмежень та зв’язків прямо у коді, побудови запитів без використанні чистого SQL за допомогою LINQ.
2. ASP.NET Core – фреймворк для побудови веб-застосунків, що надає широкі можливості для створення сучасних програмних рішень та має широкий вбудований фунціонал: механізм Dependency Injection, логування, налаштування аутентифікації та авторизації, сесій, побудови WEB API та MVC застосунків.

Був зроблений висновок, що використання зв’язки фреймворків від Microsoft пришвидшує та полегшує розробку, оскільки вони мають зручні інтерфейси для взаємодії між собою.

Практична частина роботи була присвячена розробці веб-застосунку для допомоги українцям, постраждалим внаслідок воєнних дій. Основний функціонал застосунку полягає у можливості створення та перегляду різних оголошень, які мають мету допомогти людям, які потребують житла, речей та техніки, транспорту. Завдяки механізмам фільтрації, користувачі можуть легко відсіяти непотрібні оголошення. Платформа має функціонал для зв’язку користувачів прямо на сайті, використовуючи чат, а також надає усю контактну інформацію для спілкування через інші джерела.

Список використаної літератури

1. CLR via C# FOURTH EDITION /J. Richter . – Redmond, Washington, 2012. – 896 с. – (Microsoft Press).
2. .NET Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/
3. Руководство по ASP.NET Core 6 [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet6/
4. Introduction to Razor Pages in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-6.0&tabs=visual-studio
5. ADO.NET Overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-gb/dotnet/framework/data/adonet/ado-net-overview
6. NET Revolution : An Overview of the .Net Framework Versions [Електронний ресурс] – Режим доступу до ресурсу: https://www.clariontech.com/blog/the-.net-revolution-an-overview-of-the-.net-framework-versions
7. The Good and the Bad of .NET Framework Programming [Електронний ресурс] – Режим доступу до ресурсу: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/
8. Tutorial: Add sorting, filtering, and paging - ASP.NET MVC with EF Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-6.0
9. Dependency Injection in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0
10. Upload files in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-6.0

11. How To Implement Authentication Using Identity Model In ASP.NET Core [Електронний ресурс] / Debasis Saha. – 2019. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/how-to-implement-authentication-using-identity-model-in-asp-net-core/
12. Authentication with ASP.NET Core Identity [Електронний ресурс] Marinko Spasojevic. – 2022. – Режим доступу до ресурсу: https://code-maze.com/authentication-aspnet-core-identity/