Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

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

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення веборієнтованої системи для клініки "Здоров'я родини"»

Виконав: студент 2 курсу групи ІСТС-21
спеціальності
126 "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)
Сидон Тарас Петрович
(прізвище та ініціали)
Керівник Самотій Т.С.
(прізвище та ініціали)
Керівник Сторожук О.Л.
(прізвище та ініціали)
Рецензент Крошний І. М.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
_____Сторожук О.Л.
"7" 02 2024 року

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

Сидону Тарасу Петровичу
(прізвище, ім'я, по батькові)

1.Тема роботи: Розроблення веборієнтованої системи для клініки "Здоров'я родини"

керівник роботи асистент Самотій Тетяна Сергіївна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "6" лютого 2024 року, №С-87

2.Термін подання студентом роботи 10 червня 2024р

3. Вихідні дані до роботи Постановка задачі та її формалізація. Вихідні дані та прототипи схожих вебсистем. Огляд алгоритмів та програмних засобів для розроблення програмного застосунку. Література за тематикою роботи.

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

Вступ
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки
Список використаних джерел
Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка презентаційного матеріалу для доповіді (слайди для доповіді загальним обсягом 10-12 слайдів).

6. Дата видачі завдання 7 лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел згідно поставленого завдання та формування функціональних вимог.	07.02.-21.02.	Виконано
2	Огляд стану проблемної області.	21.02.-04.03.	Виконано
3	Інформаційне забезпечення та математичне забезпечення.	04.03.-24.03.	Виконано
4	Програмне забезпечення	24.03.-28.04.	Виконано
5	Технічне забезпечення	28.04.-19.05.	Виконано
6	Висновки	19.05.-24.05.	Виконано
7	Оформлення пояснювальної записки	24.05.- 09.06.	Виконано
8	Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту	10.06.2024	Виконано

Студент _____ (підпис) **Сидон Т.П.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Самотій Т.С.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Сторожук О.Л.** (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 76 сторінок пояснювальної записки, 37 рисунків, 2 додатки, 15 джерел.

Дипломна робота присвячена розробці інформаційної веборієнтованої системи для клініки "Здоров'я родини". Для написання роботи використовувалась мова програмування C#, а також стек технологій: ASP.NET Core, HTML5, JavaScript, SASS, ORM Entity Framework, технологія Identity. Розроблена система являє собою вебсайт, який дозволяє переглядати інформацію про послуги клініки, лікарів та їхні спеціалізації, залишати заявку на укладання декларації, а також записуватися на прийом онлайн.

Система передбачає реєстрацію користувачів у ролі пацієнтів для здійснення онлайн-запису на прийом до лікаря. У режимі адміністратора у системі реалізовані такі можливості, як створення та редагування сторінок сайту, додавання, редагування та видалення інформації про лікарів та їхні спеціалізації, перегляд списку всіх записів на прийом, а також можливість оновлення інформації про послуги.

Розроблена система забезпечує зручний інтерфейс для користувачів, швидкий доступ до необхідної інформації, а також безпечний механізм зберігання особистих даних пацієнтів завдяки використанню сучасних технологій аутентифікації та авторизації.

Ключові слова: інформаційний вебсайт, ASP.NET Core, HTML5, SASS, C#, JavaScript, Entity framework.

ABSTRACT

The thesis contains 76 pages of explanatory note, 37 figures, 2 appendices, 15 sources.

The thesis is devoted to the development of an information web-oriented system for the "Family Health" clinic. The C# programming language was used to write the work, as well as a stack of technologies: ASP.NET Core, HTML5, JavaScript, SASS, ORM Entity Framework, Identity technology. The developed system is a website that allows you to view information about the clinic's services, doctors and their specializations, leave an application for a declaration, and make an appointment online.

The system provides for the registration of users as patients to make an online appointment with a doctor. In the administrator mode, the system implements such possibilities as creating and editing site pages, adding, editing and deleting information about doctors and their specializations, viewing a list of all appointments, as well as the ability to update information about services.

The developed system provides a convenient interface for users, quick access to the necessary information, as well as a secure mechanism for storing personal data of patients thanks to the use of modern authentication and authorization technologies.

Keywords: information website, ASP.NET Core, HTML5, SASS, C#, JavaScript, Entity framework.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити веборієнтовану інформаційну систему для клініки "Здоров'я родини", що дозволяє збирати та представити інформацію про послуги клініки, лікарів та їх спеціалізації, з можливістю здійснення онлайн-запису на прийом.

Технології:

- Мови програмування: C#, JavaScript
- Для розробки бази даних: MSSQL Server
- Для взаємодії з базою даних: Entity Framework
- Мова структурованої розмітки: HTML5

Вимоги до системи:

- *Загальні вимоги:*

  - Реалізація простого, зручного та зрозумілого інтерфейсу.
  - Адаптація вебсистеми під різні розміри клієнтського екрану.

- *Клієнтська частина*:

  - Можливість перегляду інформації про послуги клініки.
  - Можливість перегляду інформації про лікарів та їх спеціалізації з фотографіями.
  - Реалізація перевірки аутентифікації клієнтів.
  - Можливість здійснення запису на прийом тільки для зареєстрованих та залогінених користувачів.

- *Адміністративна частина*:

  - Реалізація логінації для входу в адмін-панель.
  - У режимі адміністратора передбачити можливість для таких операцій:

- Додавання, видалення та редагування сторінок сайту та фотографій.
- Додавання, видалення та редагування інформації про лікарів та їх спеціалізації.
- Додавання, видалення та редагування інформації про послуги клініки.
- Можливість перегляду записів клієнтів на прийом.

- *Додаткові вимоги:*

- Забезпечення безпеки особистих даних пацієнтів завдяки використанню сучасних технологій аутентифікації та авторизації.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

MVC (Model-View-Controller) - шаблон проектування: Патерн проектування, що розділяє додаток на три основні компоненти: модель, представлення та контролер. Це допомагає організувати код та полегшує його підтримку.

EF (Entity Framework) - технологія: ORM-технологія від Microsoft для роботи з базами даних. Вона дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід.

LINQ (Language Integrated Query) - мова інтегрованих запитів: Технологія в .NET, яка дозволяє використовувати запити до колекцій даних у стилі SQL безпосередньо в коді C#.

API (Application Programming Interface) - програмний інтерфейс застосунку: Набір засобів та протоколів, які дозволяють різним програмам взаємодіяти між собою.

ORM (Object-Relational Mapping) - об'єктно-реляційне відображення: Технологія, яка дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід, замість традиційних SQL-запитів.

ПК (персональний комп’ютер): Комп'ютер, призначений для індивідуального використання.

RDBMS (Relational Database Management System) - система керування базами даних (СКБД): Програмне забезпечення для створення, управління та обслуговування реляційних баз даних, де дані зберігаються у вигляді таблиць.

БД - бази даних.

ВСТУП

За останні кілька років світ значно змінився. Зміни та корективи торкнулись багатьох сфер життя. Не оминуло це і медичну сферу.

У сучасному суспільстві охорона здоров'я є однією з ключових галузей, що не тільки забезпечує фізичне та психічне благополуччя людей, але й впливає на загальний розвиток суспільства. Приватні клініки сімейного здоров'я займають особливе місце у системі медичного обслуговування, надаючи пацієнтам індивідуалізовану, високоякісну та доступну медичну допомогу. У цій бакалаврській роботі здійснено огляд ролі та важливості вебсистем приватних клінік сімейного здоров'я в контексті сучасної медичної практики.

У цьому дослідженні розглянуто основні аспекти функціонування таких систем для приватних клінік, їхні можливості та обмеження, вплив на задоволення потреб пацієнтів, а також досліджено їхній внесок у медичну систему в порівнянні з іншими видами медичних установ. Аналізуючи ці аспекти, ми зрозуміли, як веборієнтовані інформаційні системи приватних клінік сімейного здоров'я сприяють покращенню якості медичного обслуговування та розвитку сучасного здоров'я людей.

Такий підхід дозволив глибше освітити сутність та важливість вебсистем для приватних клінік сімейного здоров'я в сучасному медичному ландшафті, а також виявити потенційні напрямки їх подальшого розвитку та удосконалення.

Актуальність роботи. У сучасному світі, де швидкість і зручність стали необхідністю, інтернет і цифрові технології радикально змінили підхід до організації медичних послуг. Багато людей, шукаючи якісну та персоналізовану медичну допомогу, використовують інтернет для пошуку інформації про лікарів та клініки. Вони хочуть мати можливість обирати конкретного спеціаліста, який відповідає їхнім потребам і має відповідний досвід у лікуванні певних проблем.

Онлайн-платформи і вебсайти клінік стали ідеальними інструментами для цього. Вони не лише надають інформацію про кваліфікацію та спеціалізацію лікарів, але й дозволяють пацієнтам зручно обирати зручний для них час і дату візиту. Це важливо не лише для ефективного планування дня, а й для мінімізації часу очікування на прийом у лікаря. Пацієнти можуть зареєструватися на прийом онлайн, обираючи

зі списку доступних часів той, який їм підходить, що значно спрощує процес отримання медичної допомоги.

Такий підхід сприяє покращенню взаємодії між лікарями і пацієнтами, забезпечує більш високу якість обслуговування і робить медичну систему більш доступною та зручною для всіх учасників. Крім того, він сприяє збереженню часу як для пацієнтів, так і для медичних працівників, що дозволяє більш ефективно використовувати ресурси медичного закладу.

Мета роботи. Розробка сайту для приватної клініки сімейного здоров'я має на меті створення віртуальної платформи, що забезпечує зручний доступ пацієнтів до медичних послуг. Сайт надає можливість ознайомлення з послугами клініки, профілями лікарів, графіками прийому та онлайн записом на прийом. Головною метою є забезпечення пацієнтам можливості зручного та ефективного планування свого візиту, що сприяє покращенню якості обслуговування та задоволенню потреб клієнтів.

Об'єктом дослідження є вебсистема приватної клініка сімейного здоров'я, яка надає індивідуалізовану медичну допомогу пацієнтам, зокрема шляхом доступу до різноманітних медичних послуг, консультацій та діагностики.

Предметом дослідження є способи застосування функціональних можливостей технологій у розробленій вебсистемі приватної клініки сімейного здоров'я на базі фреймворку ASP.NET Core, зокрема технологій Entity Framework та Identity.

Практичне значення розробки сайту для клініки сімейного здоров’я полягає в наступних аспектах: покращення комунікації з пацієнтами; впровадження системи онлайн-запису на прийом; наявність структурованої інформації про лікарів, послуги, спеціалізації; розміщення на сайті статей, порад, рекомендацій та іншої освітньої інформації з питань здоров'я; можливість пацієнтам ділитися своїм досвідом через залишення відгуку про роботу клініки на сайті; сучасний, функціональний та зручний у користуванні сайт може сприяти формуванню позитивного іміджу клініки; оптимізація бізнес-процесів. А подальша інтеграція з внутрішніми системами клініки, такими як електронна медична картка пацієнта, фінансовий облік та інші,

дозволить автоматизувати ряд процесів та підвищити ефективність управління клінікою.

Впровадження таких рішень сприятиме зменшенню адміністративного навантаження на персонал та покращенню якості обслуговування пацієнтів. Крім того, це забезпечить більш точний контроль за медичними даними та сприятиме більшій прозорості у взаємодії з пацієнтами.

РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ

Огляд проблемної області для розробки сайту клініки сімейного здоров'я охоплює кілька ключових аспектів, які включають потреби пацієнтів, виклики, з якими стикаються клініки, та сучасні технологічні рішення для подолання цих викликів.

1. **Потреби пацієнтів**

Пацієнти сучасних медичних закладів мають низку очікувань від цифрових сервісів:

- ✓ **Доступність інформації**: Пацієнти хочуть мати легкий доступ до інформації про послуги, лікарів, розклад прийому та контактні дані.
- ✓ **Зручність запису на прийом**: Можливість записатися на прийом онлайн, без необхідності телефонних дзвінків.
- ✓ **Швидкий доступ до медичних даних**: Пацієнти очікують можливість перегляду своєї медичної історії, результатів аналізів та інших важливих даних.
- ✓ **Зворотний зв'язок**: Можливість залишити відгук про роботу клініки та отримати відповіді на питання.

2. **Виклики для клінік**

- ✓ **Ефективне управління пацієнтами**: Потреба в системах, які допомагають оптимізувати роботу персоналу, зменшити кількість пропущених прийомів та покращити комунікацію з пацієнтами.
- ✓ **Конкуренція**: Багато клінік борються за увагу пацієнтів, і сучасний веб-сайт може бути конкурентною перевагою.
- ✓ **Безпека даних**: Захист конфіденційної медичної інформації пацієнтів є критичним аспектом.
- ✓ **Інтеграція з внутрішніми системами**: Необхідність інтеграції вебсайту з електронними медичними записами, фінансовими системами та іншими внутрішніми процесами.

3. **Технологічні рішення**

- ✓ **Використання сучасних вебтехнологій**: ASP.NET Core, HTML5, JavaScript, SASS для створення швидких, надійних та зручних веб-додатків.
- ✓ **Використання системи управління контентом (CMS) в рамках ASP.NET Core:** Для легкого оновлення інформації на сайті можна інтегрувати рішення, що підтримуються фреймворком або створювати власні CMS-модулі.
- ✓ **Розробка рішень для онлайн-запису на базі ASP.NET Core**: Інтеграція з системами онлайн-бронювання або розробка власних рішень для зручного планування прийому.
- ✓ **Захист даних у вебдодатках на основі ASP.NET Core:** Впровадження сучасних стандартів безпеки, таких як SSL-шифрування, використання технології Identity для автентифікації та авторизації, захист від атак типу SQL Injection та інші методи захисту даних.

4. **Порівняння з існуючими рішеннями**

Огляд існуючих вебсайтів клінік показує різні рівні реалізації цих вимог:

- ✓ **Передові рішення**: Клініки, які мають повноцінні онлайн-сервіси, що включають електронні медичні картки, онлайн-консультації та інтеграцію з мобільними додатками.
- ✓ **Базові рішення**: Сайти, що надають основну інформацію про клініку, але не мають інтерактивних функцій або систем онлайн-запису.

У переважній більшості приватні клініки використовують базові рішення, оскільки розроблення повноцінної функціональної системи потребує вкладання в цей процес значної частини капіталу. Дозволити собі таку розробку можуть тільки великі корпорації. У нашому регіоні приватні клініки перебувають швидше на стадії становлення, що обумовлює використанння «легких» систем, вартість розробки яких в рази менша, але і функціонал теж значно обмежений.

Доступність інформації є ключовим аспектом для будь-якого вебсайту клініки сімейного здоров'я. Це включає не лише забезпечення легкого доступу до потрібної інформації, але й те, щоб вона була зрозумілою та корисною для різних категорій

користувачів. Ось кілька основних пунктів, які слід враховувати для забезпечення доступності інформації на вебсайті медичного центру:

✓ **Структурованість і навігація.** Важливо мати чітку структуру сайту з логічною ієрархією сторінок та розділів. Навігаційні меню повинні бути зрозумілими та легкими для користувача.

✓ **Інформативність**: Інформація про послуги клініки, медичні процедури, лікарів та їхні спеціалізації повинна бути детальною і зрозумілою. Використання простої мови без спеціалізованих термінів допоможе зрозуміти інформацію широкому колу користувачів.

✓ **Візуальна привабливість**: Вебсистема повинен мати привабливий та професійний дизайн, що сприяє залученню і утриманню уваги користувачів. Використання фотографій, відео та інших медіа може поліпшити сприйняття інформації.

✓ **Адаптивний дизайн**: Вебсайт повинен бути адаптивним до різних типів пристроїв (комп'ютери, планшети, мобільні телефони), щоб користувачі з різних пристроїв могли легко отримувати доступ до інформації.

✓ **Пошукова оптимізація (SEO)**: Важливо оптимізувати сайт для пошукових систем, щоб потенційні пацієнти могли знаходити клініку через пошукові запити про медичні послуги та лікарів.

✓ **Мовні можливості**: Забезпечення можливості перегляду вебсайту різними мовами (якщо це потрібно для місцевого населення або туристів) може покращити доступність інформації.

✓ **Спілкування з користувачами**: Наявність контактних форм та можливості залишати відгуки може покращити взаємодію з пацієнтами та забезпечити їм необхідну підтримку.

Ці пункти допоможуть зробити інформацію на вебсайті клініки доступною, зрозумілою та корисною для різних категорій користувачів.

Розглянемо типові рішення для веборієнтованих систем, представлені у нашому регіоні перебування.

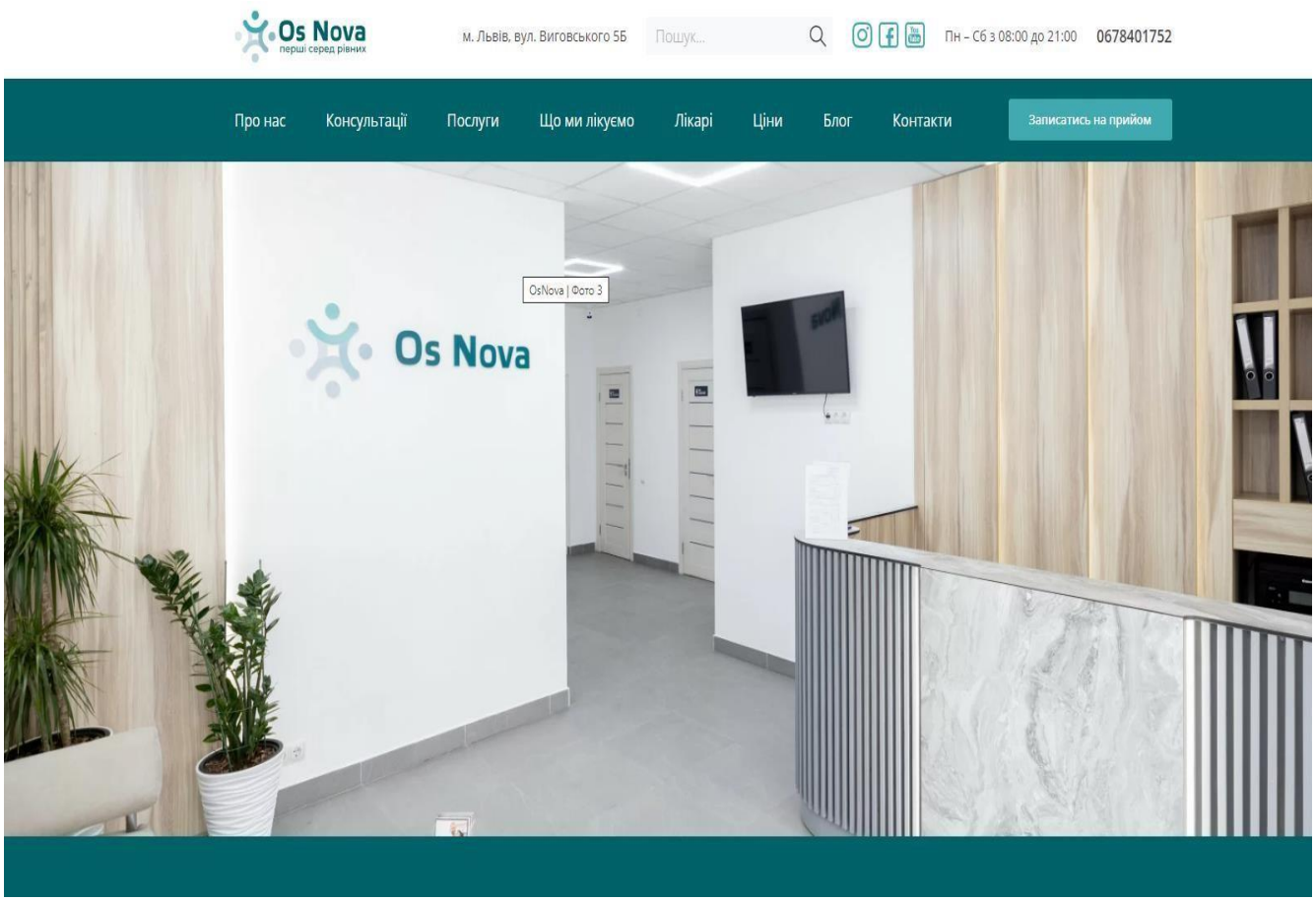


Рисунок 1.1 – Головна сторінка сайту клініки «Osnova»

Сайт клініки «Osnova» (Osnova-clinic.com.ua) спеціалізується на наданні комплексної інформації про медичні послуги клініки, доступ до спеціалістів та зручні інструменти для пацієнтів. Головна сторінка сайту відразу створює враження професійного та інноваційного медичного закладу, що використовує сучасні технології для покращення обслуговування пацієнтів.

На сайті представлено детальні описи медичних послуг, що надаються клінікою, включаючи терапевтичні, діагностичні та консультативні послуги. Інформація про лікарів містить їхні професійні профілі, кваліфікації та досвід, що допомагає пацієнтам зробити усвідомлений вибір.

Система онлайн-запису дозволяє пацієнтам не дозволяє легко та швидко записатися на прийом до лікаря, що суттєво збільшує час очікування та робить незручності у користування послугами клініки (рис.1.2). Для запису необхідно кожен раз заповнювати форму та надсилати її й очікувати зворотнього дзвінка. Що може призводити до нервування потенційних клієнтів. Адже вони неможуть розпланувати

свій час тут і зараз, а змушені чекати дій від сторонніх осіб. Відсутність підтвердження запису або зворотного зв'язку може викликати невпевненість у пацієнтів щодо успішності запису. Це вносить незручності у повсякденне життя.

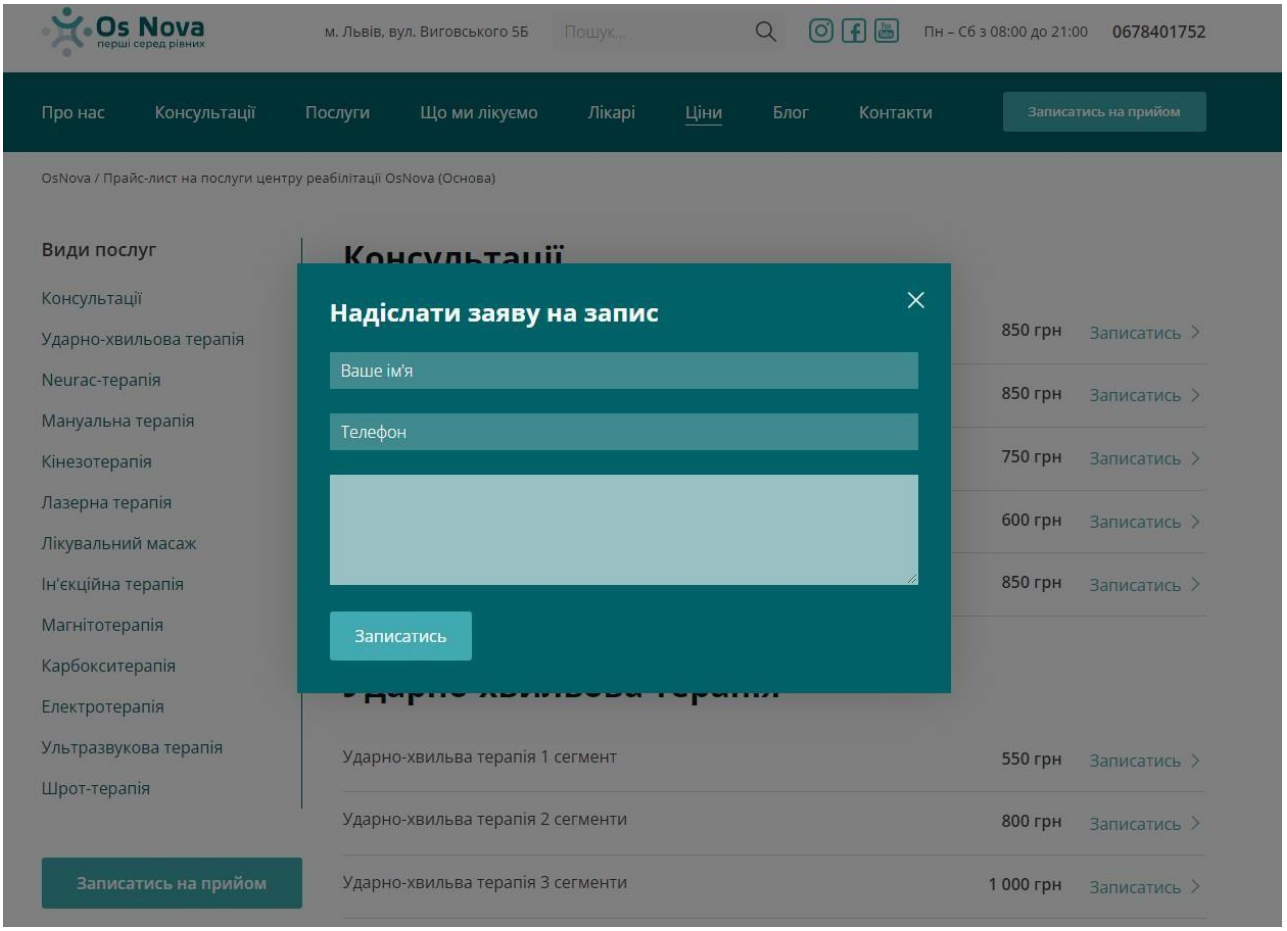


Рисунок 1.2 – Можливості запису на прийом: клініка «Osnova»

З позитивних аспекиів системи можна відзначити актуальні новини, статті та поради щодо здоров’я, що допомагає користувачам залишатися в курсі останніх медичних новин та тенденцій. Додатково, сайт має розділи з часто задаваними питаннями, контактну інформацію, мапу розташування клініки та годинами роботи. Завдяки сучасному дизайну та функціональності, сайт сприяє підвищенню рівня обслуговування пацієнтів.

При аналізі сайту "Osnova-clinic" можна звернути увагу ще на кілька потенційних недоліків, які можуть вплинути на загальне враження та користувацький досвід:

- ✓ **Швидкість завантаження**. Сайт може мати повільну швидкість

завантаження, що негативно впливає на користувацький досвід, особливо для користувачів з повільним інтернет-з'єднанням.

✓ **Мобільна версія.** Недостатньо оптимізована мобільна версія сайту може призвести до незручностей при користуванні з мобільних пристроїв, що є важливим фактором, враховуючи зростаючу кількість мобільних користувачів.

Сайт "МЕДІКАВЕР" створений для надання повної інформації про послуги клініки, спеціалістів (рис. 1.3). Перші враження від відвідування сайту – це сучасний, зручний і зрозумілий інтерфейс з інтуїтивно зрозумілою навігацією. Головна сторінка містить короткий огляд основних послуг клініки та останні новини, що допомагає користувачам швидко орієнтуватися.

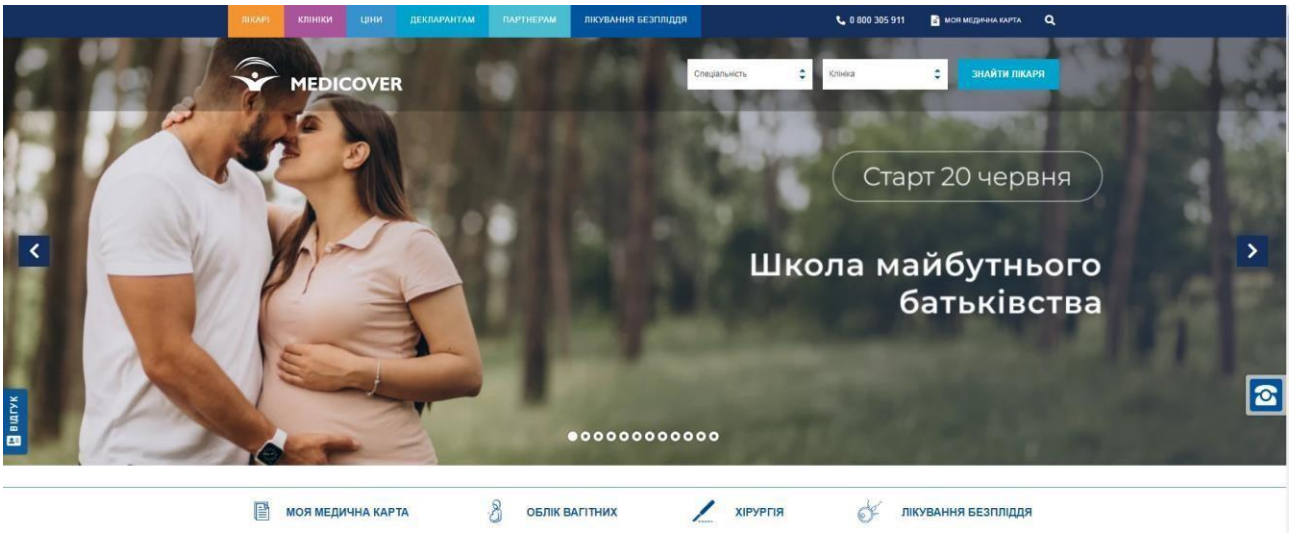


Рисунок 1.3 – Головна сторінка сайту клініки «МЕДІКАВЕР»

На сайті можна знайти детальні профілі лікарів, опис їхніх спеціалізацій та досвіду. Розділ онлайн-запису дозволяє пацієнтам бронювати прийом (рис.1.4), але теж з необхідністю очікування зворотних телефонних дзвінків. Також доступні контактні дані клініки, карта розташування. Система пошуку та фільтрів допомагає швидко знаходити потрібну інформацію. В цілому, сайт "МЕДІКАВЕР" створює враження професійного та надійного медичного закладу, що піклується про зручність та інформованість своїх пацієнтів. Але доопрацьовувати та покращувати системи у

рамках комунікації з пацієнтами теж необхідно.

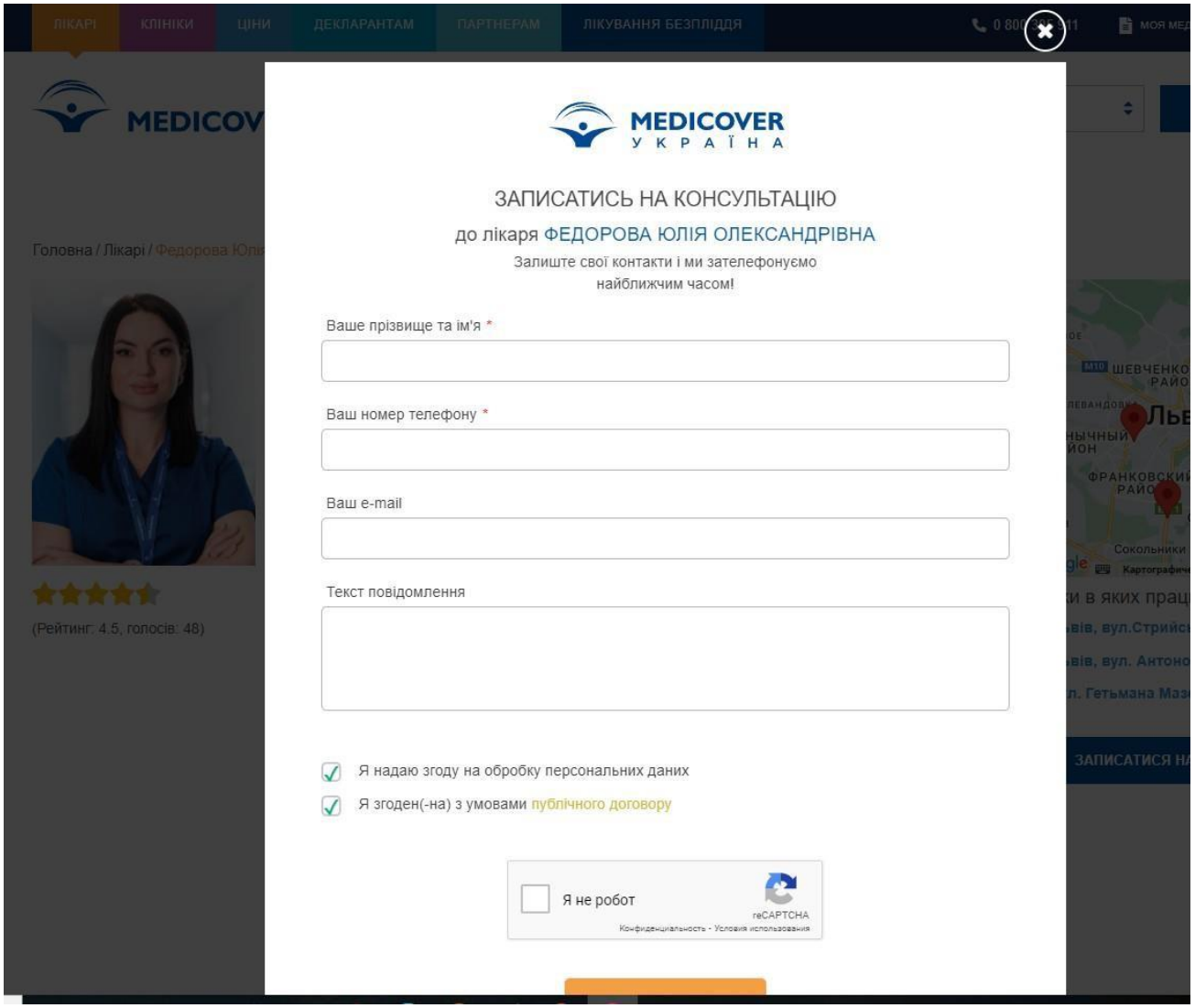


Рисунок 1.4 – Можливості запису на прийом: клініка «МЕДІКАВЕР»

Для обох сайтів можна виділити спільні проблеми, а саме: швидкість завантаження та недостатньо оптимізована мобільна версія сайту може призвести до незручностей при користуванні з мобільних пристроїв, що є важливим фактором, враховуючи зростаючу кількість мобільних користувачів. А також по суті відсутність системи онлайн- записів пацієнтів.

Тема про приватні клініки сімейного здоров'я є актуальною в сучасному світі з огляду на зростання попиту на індивідуалізовану та високоякісну медичну допомогу. Ці клініки відповідають на потреби пацієнтів у зручних умовах отримання медичної допомоги, підвищуючи доступність послуг та сприяючи покращенню загального

стану здоров'я населення. З огляду на це ми робимо спробу усунути вказані проблеми для клініки «Здоров’я родини» в рамках покращення комунікації з пацієнтами. Вебсистема може забезпечити ефективніший канал комунікації між клінікою і пацієнтами, дозволяючи оперативно отримувати інформацію про послуги, графік роботи, новини та інші важливі оновлення та впровадження системи онлайн-запису на прийом дозволяє зменшити час, витрачений на телефонні дзвінки, та спростити процес бронювання для пацієнтів. Це також допоможе оптимізувати роботу адміністративного персоналу.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Що таке шаблон проектування?

Шаблон проектування (Design Pattern) - це загальний та повторно використовуваний підхід до розв'язання типових проблем у програмуванні. Це не конкретний алгоритм або код, або скоріше загальний опис того, як вирішити певну проблему.

Шаблони проектування дозволяють розробникам використовувати вже перевірені та ефективні рішення для типових завдань в програмуванні. Вони дозволяють створювати програмне забезпечення, яке є більш структурованим, гнучким та легким для розуміння та підтримки.

Шаблони проектування можна поділити на три основні категорії:

- ✓ Структурні шаблони - це шаблони, які визначають взаємодію між об'єктами, щоб утворити більш складну структуру. Наприклад, шаблон "Decorator" дозволяє додавати нові функції до існуючих об'єктів без зміни їхньої структури.
- ✓ Породні шаблони - це шаблони, які визначають спосіб створення об'єктів. Наприклад, шаблон "Factory Method" дозволяє створювати об'єкти певного типу, не вказуючи конкретний клас цього об'єкта.
- ✓ Поведінкові шаблони - це шаблони, які визначають спосіб взаємодії об'єктів для вирішення певних задач. Наприклад, шаблон "Observer" дозволяє одному об'єкту слідкувати та реагувати на зміни в іншому об'єкті.

Застосування шаблонів проектування допомагає покращити якість програмного забезпечення, знизити його складність та вартість розробки, а також полегшити його подальше вдосконалення та розширення.

2.2 MVC-паттерн

MVC (Model-View-Controller) - це архітектурний поведінковий шаблон (рис.2.1), який дозволяє відокремити логіку програми від її представлення та сприяє покращенню організації коду. Цей підхід особливо корисний у веброзробці, де можна виділити дані, їх відображення та логіку обробки від окремих частин програми.

Модель (Model) в MVC відповідає за представлення даних та бізнес-логіку додатка. Вона не повинна містити жодного коду, який стосується відображення або способу взаємодії з користувачем. Модель може містити логіку обробки даних, валідації та збереження в базі даних.

Вид (View) - подання відповідає за представлення даних користувачеві. Перегляд даних,отриманих з моделі, щоб користувачі могли взаємодіяти з додатком. Подання неповинно містити логіку програми, а лише подання даних.

Контролер (Controller) в MVC відповідає за обробку вхідних даних від користувача та вибір відповідного виду для відображення. Контролер взаємодіє з моделлю для отримання та збереження даних, а також відповідає за відображення правильного виду для користувача.

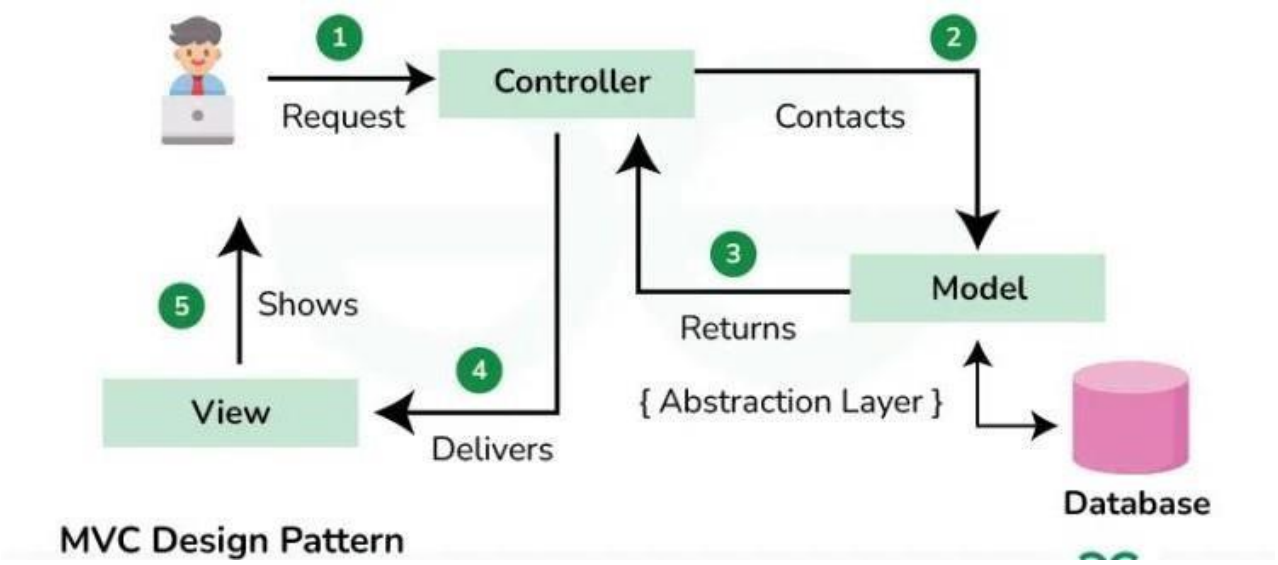


Рисунок 2.1 – Архітектурний поведінковий шаблон MVC

Шаблон MVC дозволяє створювати додатки, які є більш гнучкими та легкими для розуміння та редагування. Він сприяє розділенню логіки програми на легкокеровані компоненти, що робить код більш чистим та підтримуваним.

2.2.1 Чому народився MVC

MVC з'явився як реакція на проблеми, пов'язані з розвитком складних програмних систем, особливо у веброзробці. Ось кілька причин, чому виникла потреба в шаблоні MVC:

- ✓ **Розділення відповідальностей**: В розвитку програмних систем важливо розділити бізнес-логіку, представлення даних та логіку взаємодії з користувачем. MVC допомагає відокремити ці аспекти програми, що полегшує розробку та підтримку коду.
- ✓ **Підтримка вдосконалень та розширюваність**: Шаблон MVC забезпечує гнучкість в роботі з різними частинами програми, що дозволяє додавати нові функції та вдосконалення без зміни інших частин коду.
- ✓ **Покращення тестування**: Розділення логіки на модель, представлення та контролер дозволяє виконувати більш ефективне тестування кожного компонента окремо, що сприяє підвищенню якості програмного забезпечення.
- ✓ **Зручність управління взаємодією з користувачем**: Шаблон MVC дозволяє легко взаємодіяти з інтерфейсом користувача та керувати подіями, які виникають під час взаємодії з додатком.

Усі ці фактори призвели до того, що шаблон MVC став дуже популярним у розробці програмного забезпечення, особливо у веброзробці, де важливо мати чітко визначену структуру додатку та ефективно управляти його розвитком.

У дизайні програмного забезпечення, шаблон MVC (Model-View-Controller) виконує ключову роль у розподілі функціональності додатку на логічно відокремлені компоненти. Кожен з цих компонентів відповідає за певний аспект роботи програми:

✓ Модель (Model) - представляє бізнес-логіку та дані додатку. Не залежить від інтерфейсу користувача або способу їх відображення. Забезпечує доступ до даних та методів для їх обробки та збереження.

✓ Вид (View) - відповідає за представлення даних користувачу та інтерфейс взаємодії з користувачем. Отримує дані для відображення від моделі та відображає їх у відповідному форматі. Може містити логіку для відображення даних у відповідності до вимог дизайну.

✓ Контролер (Controller) - керує взаємодією між моделлю та виглядом. Обробляє вхідні дані від користувача та відправляє їх до моделі для подальшої обробки. Відповідає за вибір відповідного виду для відображення даних користувачу [2].

Розподіл функціональності між цими компонентами дозволяє створювати додатки, які є більш структурованими та підтримуваними. MVC дозволяє розділити логіку програми на компоненти, які можуть бути розроблені та тестовані окремо, що полегшує розробку та підтримку програмного забезпечення [1].

2.2.2 Переваги використання MVC

Використання шаблону MVC у програмуванні дозволяє чітко розділити логіку програми на три основні частини: модель (дані та бізнес-логіка), вид (представлення даних користувачу) та контролер (управління взаємодією з користувачем). Це сприяє поліпшенню якості коду, його структурованості та підтримці, оскільки кожен компонент може бути розроблений та тестований незалежно. Такий підхід робить програми більш гнучкими та легко розширюваними, оскільки дозволяє додавати нові функції без необхідності змінювати вже існуючий код[9].

2.2.3 Недоліки MVC

Хоча шаблон MVC (Model-View-Controller) має численні переваги, він також має свої недоліки:

✓ **Складність**: Реалізація шаблону MVC може бути складнішою порівняно з іншими підходами через необхідність розділення логіки на три окремі компоненти та правильну їх взаємодію.

✓ **Збільшення кількості файлів**: Використання шаблону може призвести до збільшення кількості файлів у проекті через необхідність створення окремих файлів для моделі, виду та контролера.

✓ **Потреба у додаткових зусиллях**: Реалізація MVC вимагає додаткових зусиль для правильної реалізації та підтримки кожного компонента, що може збільшити витрати часу та зусиль на розробку.

✓ **Збільшення складності відлагодження**: Розділення логіки на компоненти може ускладнити відлагодження програми, оскільки для виявлення та виправлення помилок може знадобитися велика кількість кроків та взаємодій між компонентами.

✓ **Потреба в додатковому навчанні**: Реалізація шаблону MVC може вимагати від розробників додаткового часу на ознайомлення та навчання з принципами та правилами цього шаблону.

Незважаючи на ці недоліки, використання шаблону MVC може бути дуже корисним для створення структурованих та легко підтримуваних програм, особливо в разі великих та складних проектів.

2.3. Аналіз можливостей та функцій мови програмування C#

Мова програмування C# (C-Sharp) володіє широким спектром можливостей і функцій, що роблять її популярним вибором для розробки різноманітних додатків. Ось деякі з найбільш важливих можливостей та функцій C#.

✓ **Об'єктно-орієнтоване програмування:** C# підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє створювати класи, об'єкти, спадкування та інші концепції ООП для підтримки шаблонів проектування та структурування коду.

✓ **Можливості мови LINQ**: Language Integrated Query (LINQ) - це функціональна можливість C#, яка дозволяє виконувати запити до джерел даних (наприклад, масивів, колекцій, баз даних) безпосередньо в коді.

✓ **Динамічні типи даних**: C# підтримує використання динамічних типів даних, що дозволяє змінювати типи змінних під час виконання програми.

✓ **Підтримка делегатів та подій**: C# має підтримку делегатів, які дозволяють передавати функції як параметри, і подій, які дозволяють взаємодіяти з подіями в програмі.

✓ **Асинхронне програмування**: C# надає можливості для асинхронного програмування з використанням ключових слів async та await, що дозволяє створювати ефективні та швидкодіючі додатки.

✓ **Підтримка платформи .NET**: C# є однією з основних мов програмування для платформи .NET, що дозволяє розробляти додатки для різних платформ, включаючи Windows, Linux та macOS.

✓ **Гнучкість та розширюваність**: C# дозволяє створювати різноманітні типи додатків, від маленьких консольних програм до великих вебдодатків та ігор.

Ці можливості роблять мову програмування C# потужним інструментом для розробки програмного забезпечення різного роду та складності.

2.4 ASP.NET

ASP.NET є фреймворком розробки вебдодатків, розробленим компанією Microsoft на основі мови програмування C#. Основні особливості ASP.NET включають:

- ✓ **Мова програмування C#**: ASP.NET використовує мову програмування C# для створення динамічних вебсайтів та вебдодатків.
- ✓ **Модель програмування на основі подій**: ASP.NET використовує модель програмування на основі подій, де сервер обробляє події, що виникають на вебсторінці, і генерує відповіді користувачеві.

- ✓ **ASP.NET Web Forms та ASP.NET MVC**: ASP.NET має два основні підпроекти - ASP.NET Web Forms та ASP.NET MVC (Model-View-Controller), які надають різні підходи до створення вебдодатків.
- ✓ **Висока продуктивність**: ASP.NET надає широкий набір інструментів та функціональностей, які дозволяють створювати швидкі та ефективні веб-додатки.
- ✓ **Підтримка мов програмування**: Окрім C#, ASP.NET також підтримує використання інших мов програмування, таких як Visual Basic.NET, F# та ін.
- ✓ **Широкі можливості розширення**: ASP.NET надає можливості для розширення функціональності за допомогою власних компонентів та сторонніх бібліотек.
- ✓ **Інтеграція з іншими продуктами Microsoft**: ASP.NET інтегрується з іншими продуктами Microsoft, такими як SQL Server, Azure, SharePoint, що робить його популярним вибором для розробки вебдодатків на платформі Microsoft.

ASP.NET є потужним інструментом для розробки вебдодатків, який надає широкий функціонал та можливості для створення різноманітних вебпроектів.

ASP.NET має свої переваги та недоліки, які варто враховувати при виборі цього фреймворку для розробки вебдодатків.

2.4.1 Плюси ASP.NET

Швидкість та продуктивність: ASP.NET відомий своєю високою продуктивністю і швидкодією, що дозволяє розробникам створювати швидкі веб-додатки.

Широкі можливості розширення: Фреймворк надає багато можливостей для розширення функціональності за допомогою сторонніх бібліотек та компонентів.

Інтеграція з іншими продуктами Microsoft: ASP.NET інтегрується з іншими продуктами Microsoft, такими як SQL Server, Azure, SharePoint, що робить його популярним вибором для розробки вебдодатків на платформі Microsoft.

Безпека: ASP.NET має вбудовані механізми захисту, що дозволяють захищати вебдодатки від різних видів атак.

Підтримка мов програмування: Окрім C#, ASP.NET також підтримує використання інших мов програмування, таких як Visual Basic.NET, F# та ін.

2.4.2 Мінуси ASP.NET:

Висока вартість розгортання: Використання ASP.NET може бути витратним, особливо для малих та середніх підприємств.

Специфічність для платформи Windows: ASP.NET більш специфічний для платформи Windows, що обмежує можливості розгортання на інших операційних системах.

Складність: Реалізація ASP.NET може бути складнішою порівняно з іншими підходами через необхідність розділення логіки на різні компоненти.

Залежність від Microsoft: Використання ASP.NET вимагає залежності від продуктів та технологій Microsoft, що може бути обмеженням для деяких розробників.

Важкість вивчення для початківців: Для новачків вивчення ASP.NET може бути складним, особливо якщо вони не мають досвіду роботи з платформою Microsoft.

Хоча ASP.NET має свої недоліки, він залишається популярним інструментом для розробки вебдодатків завдяки своїм широким можливостям та підтримці від Microsoft.

2.4.3 ASP.NET Core Identity

ASP.NET Core Identity - це фреймворк для управління аутентифікацією, авторизацією та управлінням користувачами в додатках ASP.NET Core. Цей фреймворк надає готові рішення для таких завдань, як реєстрація та аутентифікація користувачів, керування ролями та доступом до ресурсів, а також забезпечує можливість робити це з використанням різних постачальників даних (наприклад, Entity Framework Core).

ASP.NET Core Identity можна налаштувати на використання бази даних SQL Server для зберігання таких даних, як імена користувачів, паролі та файли конфігурації. Звичайно, також можна використовувати Mongo DB і т.д., щоб зберігати відповідні дані.

Основні можливості ASP.NET Core Identity:

Аутентифікація дозволяє користувачам реєструватися та входити в систему за допомогою різних методів аутентифікації, таких як пароль, зовнішні служби аутентифікації (Google, Facebook, Twitter тощо) або двофакторна аутентифікація.

Авторизація надає можливість керувати доступом користувачів до різних частин додатка за допомогою ролей та політик авторизації.

Керування ролями дозволяє створювати ролі користувачів та надавати їм відповідні права доступу до функціоналу додатка.

Зберігання користувачів та ролей. Інформація про користувачів, їх паролі та ролі зберігаються в базі даних за допомогою вбудованих механізмів ASP.NET Core Identity або можуть бути розширені за допомогою власних реалізацій.

Підтримка зовнішніх постачальників дозволяє інтегрувати зовнішні служби аутентифікації для спрощення процесу входу користувачів (наприклад, вхід через обліковий запис Google або Facebook).

Підтримка токенів доступу: Дозволяє генерувати та використовувати токени доступу для забезпечення безпеки взаємодії з API додатка.

ASP.NET Core Identity дозволяє розробникам швидко та ефективно впроваджувати функціонал управління користувачами та безпеки в своїх додатках ASP.NET Core.

2.5 Що таке ORM

ORM (Object-Relational Mapping) - це підхід до розробки програмного забезпечення, який дозволяє розробникам використовувати об'єктно-орієнтований код для роботи з реляційними базами даних. Основна ідея полягає в тому, щоб автоматично перетворювати дані між об'єктною моделлю програми і таблицями бази даних.

Головні переваги ORM.

✓ Уникнення прямих SQL-запитів: розробники можуть працювати з об'єктами у своїх програмах, не вдаючись до прямих SQL-запитів, що спрощує розробку та підтримку коду.

✓ Зменшення повторення коду: ORM дозволяє використовувати уніфіковані методи для взаємодії з базою даних, що зменшує кількість дублювання коду.

✓ Синхронізація структури даних: ORM допомагає зберігати синхронізованою структуру об'єктної моделі та схеми бази даних.

✓ Підтримка реляційних зв'язків: ORM дозволяє зручно використовувати реляційні зв'язки між об'єктами, такі як один до одного, один до багатьох, багато до багатьох.

✓ Підтримка різних СУБД: більшість ORM-фреймворків підтримують різні системи управління базами даних, що дозволяє переносити додатки між різними базами даних без змін у коді.

Одним із недоліків ORM є можливість втрати продуктивності в результаті автоматичного генерування неоптимальних SQL-запитів. Також, в деяких випадках виникають складнощі з відлагодженням та налаштуванням ORM-фреймворків.

2.6 Entity Framework

Entity Framework (EF) - це набір технологій для роботи з базами даних у середовищі .NET Framework. Він дозволяє розробникам працювати з даними у форматі об'єктно-орієнтованого коду, замість написання запитів мовою SQL. Основні компоненти Entity Framework включають модель даних (Entity Data Model), набір API для доступу до даних (Data Access API), а також інструменти для мапування об'єктів на дані в базі даних.

Плюси Entity Framework:

- ✓ **Зручний для використання:** EF надає високорівневий API для роботи з базами даних, що полегшує розробку та підтримку коду.
- ✓ **Модель даних за замовчуванням:** EF автоматично генерує модель даних на основі структури бази даних, що зменшує необхідність вручну визначати об'єктну модель.
- ✓ **Мапування об'єктів на дані:** EF дозволяє мапувати об'єкти доменної моделі на таблиці бази даних без необхідності написання складних запитів SQL.
- ✓ **Підтримка LINQ:** Entity Framework інтегрується з мовою запитів LINQ (Language Integrated Query), що дозволяє виконувати різноманітні операції з даними у виразній та зрозумілій формі.
- ✓ **Підтримка різних баз даних:** EF підтримує різні системи управління базами даних, такі як Microsoft SQL Server, MySQL, PostgreSQL тощо.

Мінуси Entity Framework:

- ✓ **Втрата продуктивності:** У деяких випадках EF може бути менш продуктивним у порівнянні з написанням ручних оптимізованих SQL-запитів.
- ✓ Обмежені можливості оптимізації: Неспроможність EF автоматично генерувати оптимальні SQL-запити може призвести до неефективності деяких операцій з базою даних.

- ✓ Складність відлагодження: У деяких випадках відлагодження EF може бути складнішим, особливо при роботі зі складними мапуваннями об'єктів.
- ✓ Залежність від Microsoft: Entity Framework є продуктом Microsoft і може бути менш підходящим варіантом для розробки на інших платформах або в середовищах, де бажано уникати використання пропрієтарного програмного забезпечення.
- ✓ Великий обсяг даних: При роботі з великим обсягом даних може збільшуватися час виконання операцій з базою даних через велику кількість звернень до неї з боку EF.

Хоча Entity Framework має свої недоліки, він залишається популярним інструментом для роботи з базами даних у середовищі .NET завдяки своїм зручним API та широкому спектру функцій.

2.7 HTML5

HTML5 - це остання версія мови розмітки гіпертексту, яка використовується для створення та відображення вебсторінок. Вона має кілька переваг і недоліків, які варто враховувати при її використанні.

Переваги HTML5:

✓ Підтримка мультимедіа. HTML5 дозволяє вбудовувати аудіо, відео та графіку без використання додаткових плагінів, таких як Flash.

✓ Покращена семантика. HTML5 має нові теги, які дозволяють краще визначати структуру документа (наприклад, <header>, <footer>, <nav>), що полегшує розуміння контенту для пошукових систем та асистентів.

✓ Підтримка локального сховища. HTML5 включає можливості локального сховища, яке дозволяє зберігати дані на стороні клієнта без необхідності використання cookies.

✓Підтримка вебдодатків. HTML5 має API, які дозволяють створювати веб-додатки, які можуть працювати в автономному режимі та використовувати різноманітні функції пристрою (наприклад, геолокацію, камеру).

✓ Покращена підтримка форм. HTML5 включає нові типи елементів форм, такі як <input type="date">, <input type="email">, що полегшують створення та валідацію форм.

Недоліки HTML5:

✓ Сумісність браузерів. Хоча більшість сучасних браузерів підтримують HTML5, деякі старі версії браузерів можуть не відображати сторінки правильно.

✓ Брак стандартизації. У деяких випадках різні браузери можуть інтерпретувати HTML5 по-різному, що може викликати проблеми з сумісністю.

✓ Потреба у великій кількості коду. Деякі функції HTML5 вимагають більше коду порівняно зі старішими версіями HTML.

✓ Потреба в нових знаннях та навичках. Для використання всіх можливостей HTML5 може знадобитися поглиблене розуміння технологій веброзробки.

HTML5 є потужним інструментом для створення вебсторінок та додатків, але розробники повинні враховувати його переваги та недоліки для досягнення найкращих результатів [5].

2.8 Sass

Sass (Syntactically Awesome Stylesheets) - це метамова CSS, що додає багато нових функцій та можливостей до звичайного CSS. Основні переваги Sass включають [15]:

✓ **Змінні**. Можливість використовувати змінні для зберігання значень, таких як кольори, розміри шрифтів, відступи тощо, що дозволяє легко змінювати стиль додатка.

✓ **Вкладеність**. Можливість вкладати один селектор в інший, що полегшує структуру та розуміння CSS-коду.

✓ **Міксіни**. Можливість використовувати міксіни для створення повторюваних стилів, що дозволяє уникнути дублювання коду.

✓ **Умовні оператори**. Можливість використовувати умовні оператори (наприклад, if та else), що дозволяє створювати більш динамічні стилі.

✓ **Імпорт**. Можливість імпортувати файли Sass в інші файли, що дозволяє розділити стилі на логічні блоки.

✓ **Розширення селекторів**. Можливість розширювати стилі одного селектора іншим, що дозволяє створювати більш зручний та компактний CSS-код.

Що ж до недоліків, Sass вимагає додаткового кроку компіляції у звичайний CSS перед використанням на вебсайті, що може бути необхідним додатковим кроком у процесі розробки. Також, використання деяких функцій Sass може призвести до генерації більш великого обсягу CSS, ніж потрібно, що може погіршити продуктивність вебсайту.

2.9 Проектування бази даних

Проектування бази даних (ПБД) - це процес створення структури бази даних, яка відображає потреби бізнесу та дозволяє ефективно зберігати та управляти даними. ПБД включає в себе визначення сутностей (таблиць), атрибутів (стовпців), відносин між сутностями та інших характеристик бази даних.

Основні етапи проектування бази даних включають:

- ✓ **Аналіз вимог**: Визначення потреб бізнесу та вимог до даних, які повинні бути збережені та оброблені базою даних.
- ✓ **Створення моделі даних**: Розроблення концептуальної моделі даних, яка відображає сутності та відносини між ними, а також логічної моделі даних, яка визначає структуру таблиць та зв'язки між ними.
- ✓ **Нормалізація**: Процес розбиття таблиць на менші, оптимізовані для зберігання даних, з метою уникнення аномалій та забезпечення цілісності даних.
- ✓ **Фізичне проектування**: Визначення способу збереження даних на рівні операційної системи та обрання відповідної технології бази даних (наприклад, MySQL, SQL Server, PostgreSQL тощо).

- ✓ **Реалізація та тестування**: Створення бази даних за розробленим проектом та перевірка її працездатності та відповідності вимогам.
- ✓ **Експлуатація та підтримка**: Після введення в експлуатацію база даних підтримується та модифікується відповідно до змін в бізнес-вимогах.

Проектування бази даних є критично важливим етапом в розробці будь-якого інформаційного системи, оскільки від правильності його виконання залежить ефективність та надійність роботи системи в цілому.

Entity Framework-це рішення ORM, яке може автоматично пов'язувати класи C# з таблицями в базі даних. Entity Framework Core підтримує найпопулярніші СУБД, включаючи MySQL, MS SQL Server, SQLite, та PostgreSQL. У цьому випадку ми керуємо базою даних у MS SQL Server за допомогою Entity Framework core.

Додамо класи, які представлятимуть дані, до проекту. В папці Domain у структурі проекту знаходиться перелік усіх класів-доменних моделей. Назви розроблених класів, що представляють таблиці у БД HealthClinic, подані нижче:

- ✓ Article
- ✓ Declaration
- ✓ Doctor
- ✓ Item
- ✓ MedService
- ✓ Order
- ✓ Page
- ✓ PageBase
- ✓ Patient
- ✓ Speciality

Властивості, присутні у класах, відповідатимуть окремому стовпцю у відповідній таблиці бази даних.

Для взаємодії з БД через Entity Framework Core потрібен контекст даних - клас, успадкований від Microsoft.EntityFrameworkCore.DbContext. Додамо до проекту новий клас під назвою AppDbContext. У конструкторі класу AppDbContext через параметр options передаватимуться налаштування контексту даних. Виклик

Database.EnsureCreated() у конструкторі забезпечує створення БД за визначеною моделлю, якщо вона ще не існує.

Контекст даних, через який здійснюється взаємодія з БД, передається як сервіс через механізм впровадження залежностей (Dependency Injection). Для роботи з Entity Framework Core необхідно додати клас контексту даних до колекції сервісів програми.

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(Config.ConnectionString));
```

Підключивши контекст даних, ми можемо звертатися до БД за його допомогою. Далі, застосувавши механізм міграції БД, технологія Entity Framework автоматично зв'яжеться з SQL Server і, використовуючи вказівки з класу AppDbContext, створить БД та таблиці у ній.

Таким чином, була створена БД HealthClinic, яка містить 15 таблиць.

На зображенні представлена структура бази даних HealthClinic, яка складається з 15 таблиць:

1. AspNetUsers: Таблиця зберігає інформацію про користувачів. Поля включають Id, Discriminator, FirstName, SecondName, LastName, Address, UserName, Email, NormalizedEmail, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber.
2. AspNetRoles: Таблиця зберігає інформацію про ролі користувачів. Поля включають Id, Name, NormalizedName.
3. AspNetUserRoles: Таблиця зв'язує користувачів з їхніми ролями. Поля включають UserId, RoleId.
4. AspNetUserClaims: Таблиця зберігає інформацію про заяви користувачів. Поля включають Id, UserId, ClaimType.
5. AspNetRoleClaims: Таблиця зберігає інформацію про заяви ролей. Поля включають Id, RoleId, ClaimType.
6. AspNetUserLogins: Таблиця зберігає інформацію про логіни користувачів. Поля включають UserId, LoginProvider, ProviderKey, ProviderDisplayName.
7. AspNetUserTokens: Таблиця зберігає токени користувачів. Поля включають UserId, LoginProvider, Name.

8. Items: Таблиця зберігає інформацію про елементи. Поля включають Id, MedServiceId, Quantity, Date, Time.
9. Orders: Таблиця зберігає інформацію про замовлення. Поля включають OrderId, PurchasedDate, BuyerId.
10. MedServices: Таблиця зберігає інформацію про медичні послуги. Поля включають Id, Name, DoctorId, PriceBase.
11. Doctors: Таблиця зберігає інформацію про лікарів. Поля включають Id, FirstName, SecondName, LastName, Address, SpecialityId, Qualification, Degree, Education, Internship.
12. Declarations: Таблиця зберігає інформацію про декларації. Поля включають Id, Series, Number, DoctorId.
13. Pages: Таблиця зберігає інформацію про сторінки. Поля включають Id, PageName, Title, Subtitle, TitleImagePath, MetaTitle, MetaDescription.
14. Articles: Таблиця зберігає інформацію про статті. Поля включають Id, ArticleName, Title, Text, Subtitle, TitleImagePath, MetaTitle, MetaDescription.
15. Specialitys: Таблиця зберігає інформацію про спеціальності. Поля включають Id, Name.

Ключові зв'язки між таблицями:

✓ AspNetUserRoles зв'язує таблиці AspNetUsers і AspNetRoles.

✓ AspNetUserClaims пов'язана з AspNetUsers.

✓ AspNetRoleClaims пов'язана з AspNetRoles.

✓ AspNetUserLogins і AspNetUserTokens пов'язані з AspNetUsers.

✓ Items має зовнішній ключ MedServiceId, який пов'язаний з таблицею MedServices.

✓ Orders має зовнішній ключ BuyerId, який пов'язаний з таблицею AspNetUsers.

✓ MedServices має зовнішній ключ DoctorId, який пов'язаний з таблицею Doctors.

✓ Declarations має зовнішній ключ DoctorId, який пов'язаний з таблицею Doctors.

✓ Doctors має зовнішній ключ SpecialityId, який пов'язаний з таблицею Specialitys.

Ця структура забезпечує інтегровану систему для управління користувачами, ролями, медичними послугами, замовленнями та іншими пов'язаними даними.

Можна зазначити, що кількість таблиць в базі даних перевищує кількість розроблених доменних моделей для цього проекту. Це пояснюється тим, що додаткова технологія Identity також створює таблиці для зберігання інформації, пов'язаної з користувачами. Схема бази даних представлена на рисунку 2.2.

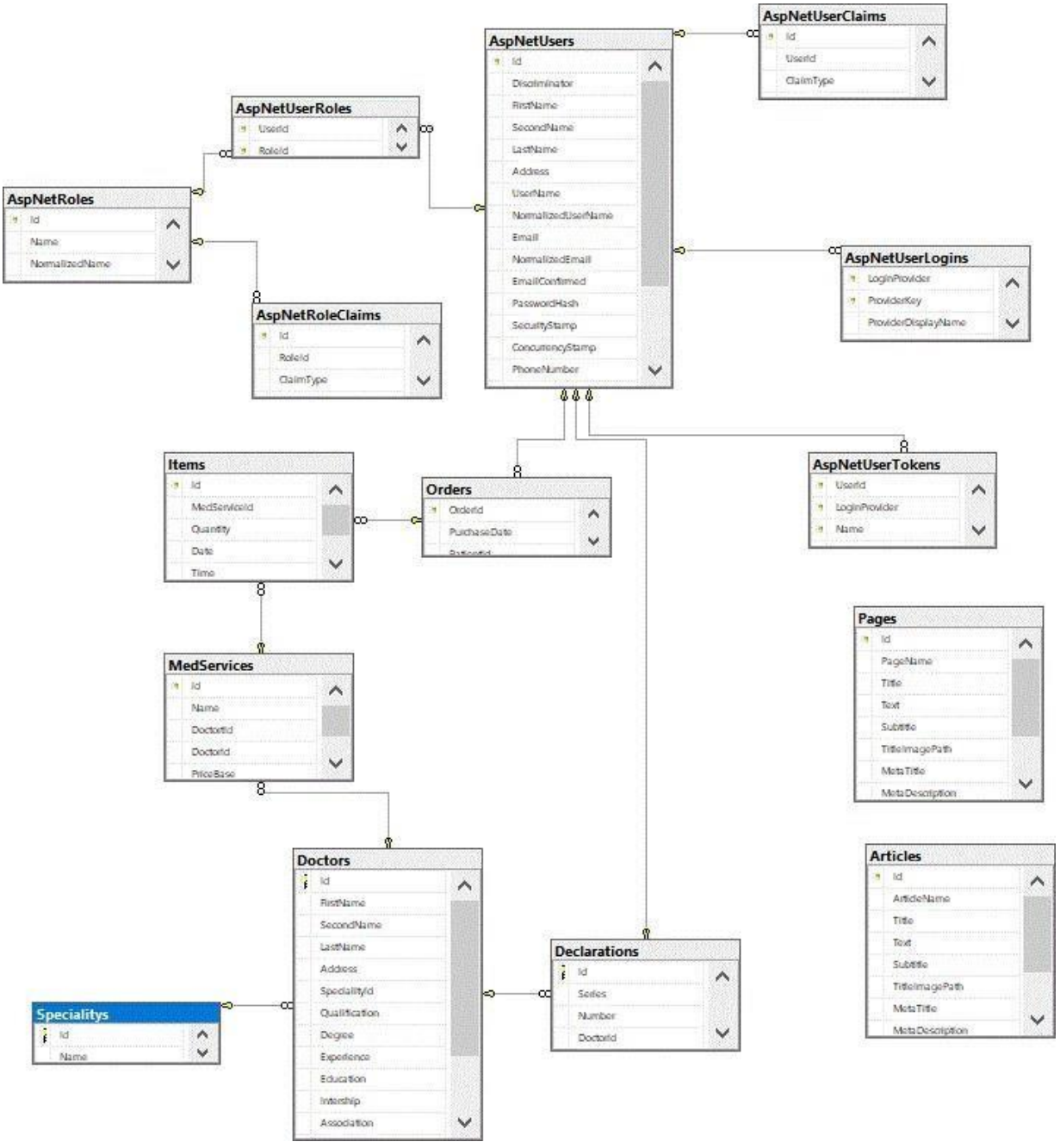


Рисунок 2.2 – Схема БД проекту

3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Мета розробки

Мета створення інформаційної вебсистеми для клініки сімейного здоров'я полягає у наданні потенційним клієнтам детальної інформації про послуги, що надаються, а також у можливості здійснення запису для отримання цих послуг. Це також допоможе спростити облік записів та процес надання медичних послуг, які пропонує клініка.

При розробці вебсистеми важливо створити інтуїтивно зрозумілий інтерфейс, який не створюватиме негативного досвіду для користувачів і не спонукатиме їх залишити сайт. Дизайн сайту повинен бути простим, з елементами інтерактиву, та привабливим.

Особливо важливо врахувати, що значна частина інтернет-трафіку припадає на мобільні пристрої з низьким розширенням екрану. Тому вебсистема повинна бути адаптивною, забезпечуючи оптимальне відображення і функціональність на будь-якому пристрої.

Практичне значення розробки сайту для клініки сімейного здоров'я включає:

Покращення комунікації з пацієнтами. Забезпечення швидкого та зручного зв'язку між клінікою та пацієнтами.

Впровадження системи онлайн-запису на прийом. Дозволяє пацієнтам записатися на прийом до лікаря в зручний для них час.

Наявність структурованої інформації про лікарів, послуги та спеціалізації: Пацієнти можуть легко знайти необхідну інформацію про медичний персонал та доступні послуги.

Розміщення статей, порад та рекомендацій з питань здоров'я. Освітній контент сприятиме підвищенню обізнаності пацієнтів.

Можливість залишати відгуки про роботу клініки. Пацієнти можуть ділитися своїм досвідом, що сприятиме покращенню якості послуг.

Формування позитивного іміджу клініки. Сучасний, функціональний та зручний сайт допоможе створити позитивний образ клініки.

Оптимізація бізнес-процесів. Автоматизація та спрощення багатьох

адміністративних завдань, що дозволить ефективніше використовувати ресурси клініки.

Ці аспекти підкреслюють важливість створення сучасної вебсистеми, яка буде корисною як для пацієнтів, так і для персоналу клініки.

3. 2 Процес розробки

Для розробки проекту на C# ми скористаємося безкоштовним і повнофункціональним середовищем розробки - Visual Studio Community 2022.

Ми почали створення нового проекту у Visual Studio. Спершу, у вікні "Create a new project" ми обрали шаблон для нового проекту. Серед доступних варіантів ми вибрали **ASP.NET Core Web App**, який дозволяє створювати вебдодатки на основі .NET Core.

Наступним кроком було налаштування нашого проекту у вікні "Configure your new project" (рис.3.1). Ми назвали свій проект "HealthClinic" і обрали місце для збереження файлів проекту на нашому комп'ютері. Назва проекту і рішення були однакові — "HealthClinic".

Після цього, у вікні "Additional information" ми вибрали додаткові параметри для нашого проекту. Ми обрали .NET 6.0 як версію фреймворку, що забезпечує довгострокову підтримку. Ми не вказували тип аутентифікації, залишивши його значення на "None", та увімкнули конфігурацію для використання HTTPS. Опцію для Docker не активували.

Після завершення налаштувань і створення проекту, ми перейшли до робочого середовища Visual Studio, де відкрилось основне вікно з нашим новим проектом "HealthClinic". У цьому вікні видно структуру проекту з файлами і папками, включаючи основні файли проекту, такі як Program.cs та appsettings.json. На панелі інструментів ми також побачили інформацію про ASP.NET Core, яка пропонує ресурси для навчання і роботи з цією платформою, зокрема, документацію, інструкції для публікації додатків в Azure і поради щодо використання IDE.

Таким чином, ми створили новий ASP.NET Core вебдодаток у Visual Studio під

назвою "HealthClinic" і перейшли до подальшої розробки.

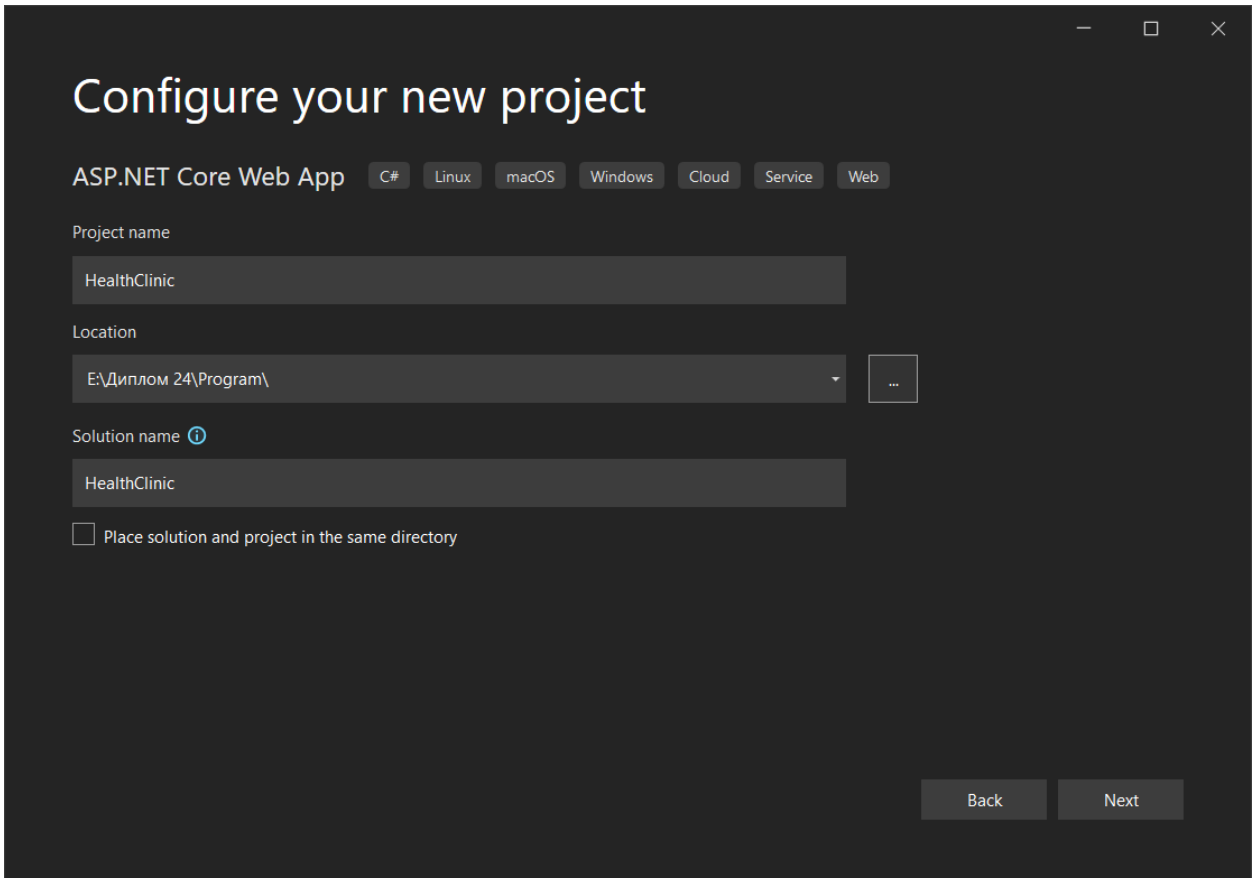


Рисунок 3.1 – Вікно конфігурації нового проекту у Visual Studio 2022

Для прискорення розробки проекту та додавання додаткових функцій, таких як оптимізація зображень, ми встановимо наступні розширення:

ImageOptimizer: Це розширення дозволяє автоматично оптимізувати зображення, зменшуючи їх розмір без втрати якості. Це допомагає пришвидшити завантаження вебсторінок та зменшити обсяг трафіку.

Bundler & Minifier: Це розширення об'єднує та мінімізує CSS і JavaScript файли, що зменшує кількість HTTP-запитів і прискорює завантаження сторінок.

WebCompiler: Це розширення компілює файли SASS, LESS, CoffeeScript та інші на льоту, що полегшує роботу з цими препроцесорами і зберігає час на компіляцію.

ZenCoding: Це розширення прискорює написання HTML та CSS за допомогою скорочень (snippets), дозволяючи швидко створювати складні структури з меншою кількістю коду.

AddNewFile: Це розширення спрощує процес створення нових файлів у проекті, дозволяючи швидко додавати нові файли будь-якого типу без необхідності проходити через кілька діалогових вікон.

FileNesting: Це розширення дозволяє групувати пов'язані файли разом у файловій системі проекту, роблячи структуру проекту більш організованою і полегшуючи навігацію.

Встановлення цих розширень значно полегшить і пришвидшить процес розробки, дозволяючи вам зосередитися на написанні коду та реалізації нових функцій.

Встановлені розширення показано на рисунку 3.2.

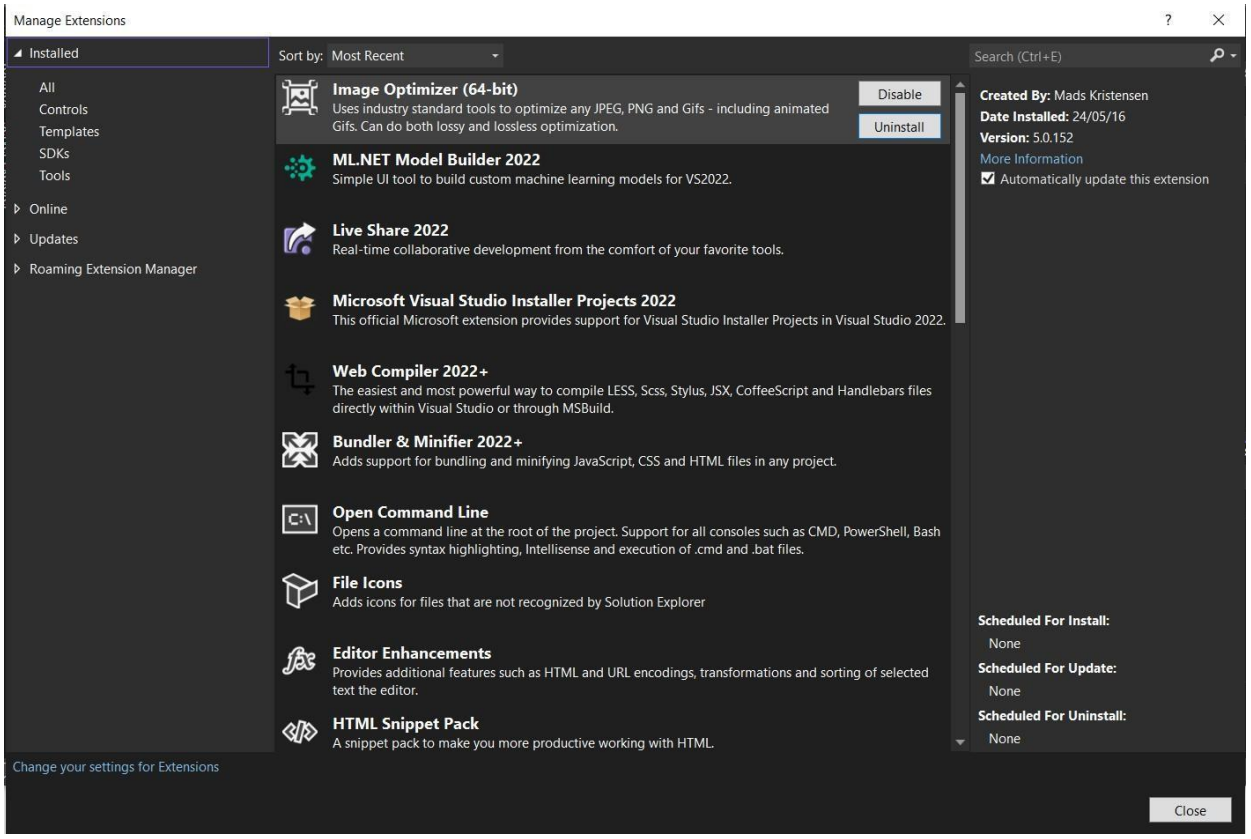


Рисунок 3.2 – Встановлені розширення

Наприклад, одним із найпоширеніших викликів у веброзробці є ефективне керування скриптами та стилями на вебсторінках. Часто сайти включають велику кількість різноманітних файлів скриптів і стилів, що може значно уповільнювати завантаження сторінки: кожен додатковий запит до сервера збільшує час загрузки, а кожен невикористаний біт інформації займає цінне мережеве простір.

Для оптимізації роботи вебдодатків застосовуються такі інструменти, як бандлінг та мініфікація. Бандлінг дозволяє об'єднувати різні файли скриптів та стилів в один, зменшуючи кількість запитів до сервера. Це покращує час завантаження сторінки і зменшує навантаження на сервер. Мініфікація, у свою чергу, зводить до мінімуму розмір файлів, видаляючи зайві пропуски, коментарі та стискаючи код. Це зменшує обсяг передаваних даних і покращує продуктивність завантаження сторінок.

У ASP.NET Core ці процеси можна автоматизувати під час розробки, що спрощує управління складними наборами скриптів та стилів і забезпечує оптимальну продуктивність вебдодатків. Такий підхід не лише поліпшує користувацький досвід, але й зменшує витрати ресурсів сервера, що є важливим фактором для сучасних веб-застосунків.

Коли ми розпочинали розробку нашого вебдодатку у середовищі ASP.NET Core, одним з перших кроків було встановлення кількох ключових пакетів (рис.3.3), які значно полегшили наші завдання.

Серед них був пакет Microsoft.AspNetCore.Identity.EntityFrameworkCore, який став основою для реалізації системи ідентифікації в нашому додатку. Цей пакет, заснований на Entity Framework Core, надав нам мощні інструменти для автентифікації та авторизації користувачів, що є критичним аспектом для багатьох вебзастосунків.

Також ми встановили Microsoft.EntityFrameworkCore, сучасний об'єктно-реляційний мапер (ORM), який дозволяє нам легко взаємодіяти з базами даних через LINQ запити, відстежування змін та міграції схем. Це значно спростило роботу з реляційними базами даних, такими як SQL Server, який ми використовуємо у нашому проекті.

Для зручності розробки та дизайну баз даних ми встановили також пакет Microsoft.EntityFrameworkCore.Design, який допоміг нам з генерацією коду та іншими аспектами дизайну, необхідними для Entity Framework Core.

Окрім цього, щоб забезпечити ефективну налагодження вебзастосунку, ми використовуємо Microsoft.Extensions.Logging.Debug, який дозволяє виводити журнальні повідомлення у вікно налагодження, спрощуючи процес відладки та

моніторингу додатку.

Не менш важливим інструментом для нас став Microsoft.VisualStudio.Web.CodeGeneration.Design, який автоматизує генерацію контролерів та подарунків для ASP.NET Core. Цей інструмент значно прискорив створення нового функціоналу та забезпечив консистентність нашого коду.

Завдяки цим пакетам ми змогли значно покращити якість нашого коду, спростити процес розробки і забезпечити швидке впровадження нових функцій у нашому вебдодатку.

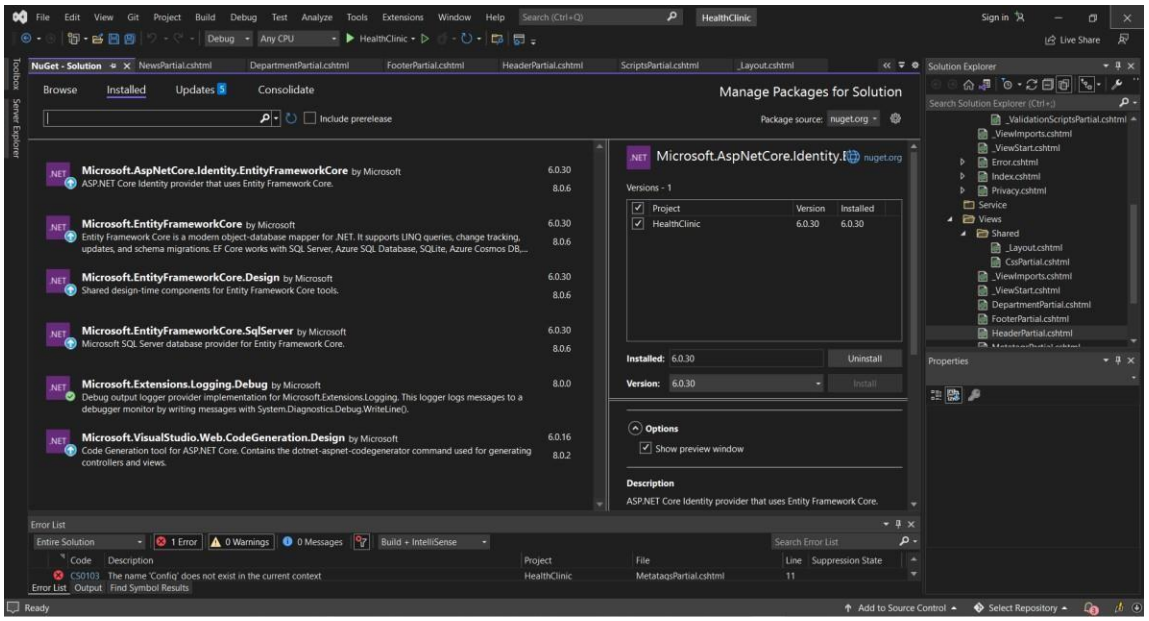


Рисунок 3.3 – Встановлені пакети

Початкова структура проекту відображена у вікні Solution Explorer, як показано на рисунку 3.4.

Базова структура документа ASP.NET Core Web Application включає наступні основні файли та папки:

- ✓ wwwroot - статичні файли (CSS, JavaScript, зображення).
- ✓ Controllers - контролери, які обробляють HTTP-запити.
- ✓ Views - представлення, що відображають дані.
- ✓ Models - моделі, що представляють дані додатка.
- ✓ Program.cs - точка входу додатка, містить метод Main.
- ✓ Startup.cs - конфігурація додатка, налаштування служб і middleware.

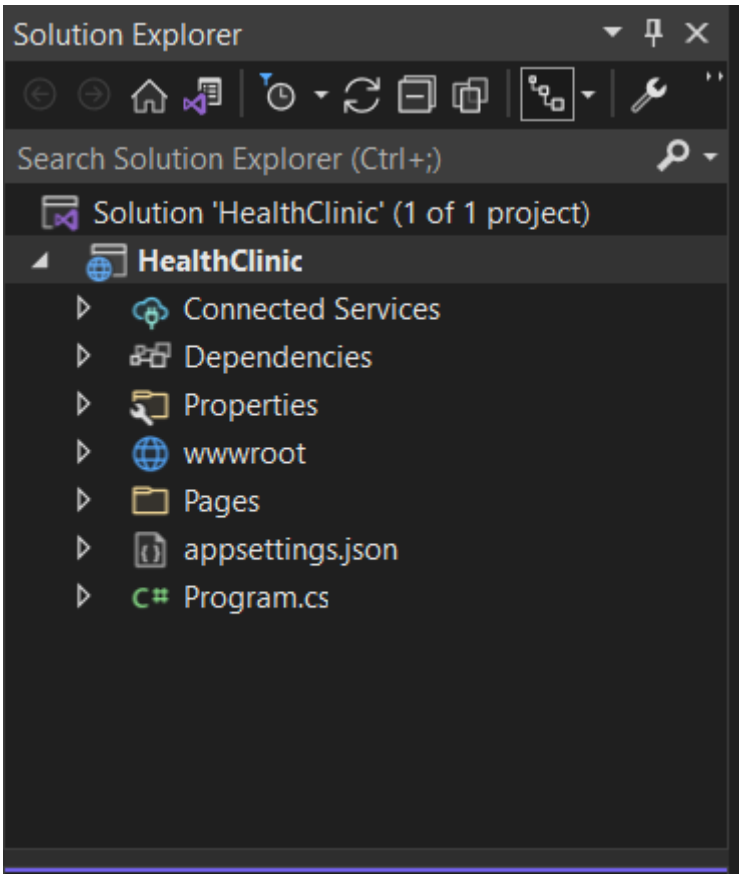


Рисунок 3.4 – Структура проекту на початку розробки

Під час розробки нашого проекту ми прийняли рішення організувати структуру каталогів для зручності управління кодом і ресурсами. Ось деякі з основних кроків, які ми здійснили:

У каталозі `wwwroot` ми створили ряд підкаталогів для зберігання статичних файлів, таких як зображення, CSS-файли, файли скриптів та інші необхідні ресурси. Це дозволяє нам логічно організувати ресурси, які використовуються на сторінках вебдодатку.

Додатково, ми вирішили використовувати області (`Areas`) у нашому проекті. Кожна область, така як `Admin` і `Order`, відповідає за конкретну функціональність нашого додатку. Наприклад, у області `Admin` ми реалізували роль адміністратора сайту, створивши всі необхідні файли для управління адміністративними функціями. У області `Order` реалізована логіка вибору медичної послуги та лікаря, який її надає- та здійснення запису на прийом.

Також ми створили каталог `Service`, де зосередили додаткові сервіси для проекту, такі як, наприклад, серіалізація та десеріалізація об'єктів у формат JSON. Це дозволяє нам легко керувати розширенням функціоналу і використовувати загальнодоступні сервіси у різних частинах додатку.

У директорії `Controllers` ми організували контролери, які відповідають за логіку стартової сторінки, інформаційних сторінок та виводу послуг для неавторизованих користувачів. Це спрощує керування кодом і робить його більш структурованим і зрозумілим для розробників.

Після встановлення всіх необхідних каталогів, файлів, сервісів і доменних моделей, а також налаштування контексту бази даних, ми внесли зміни до файлу `Startup.cs`. Особливу увагу ми приділили механізму впровадження залежностей (Dependency Injection, DI), що є основою для створення слабопов'язаних компонентів у нашому додатку. Цей підхід зробив нашу систему гнучкішою і більш легкою для розширення.

Dependency Injection (DI) – це програмний шаблон, який дозволяє створювати залежності між об'єктами таким чином, щоб ці залежності надавалися зовнішнім способом, а не створювалися всередині самого об'єкта. У контексті ASP.NET Core це означає, що класи можуть оголошувати свої залежності через конструктори, а фреймворк автоматично надасть ці залежності при створенні об'єкта.

В результаті цих кроків ми досягли більшої структурованості і ефективності у розробці нашого вебдодатку, що сприяє покращенню якості коду і зручності підтримки проекту.

На рисунку 3.5. зображена структура проекту, а на рисунку 3.6. представлений клас Startup.cs з механізмами впровадження залежностями.

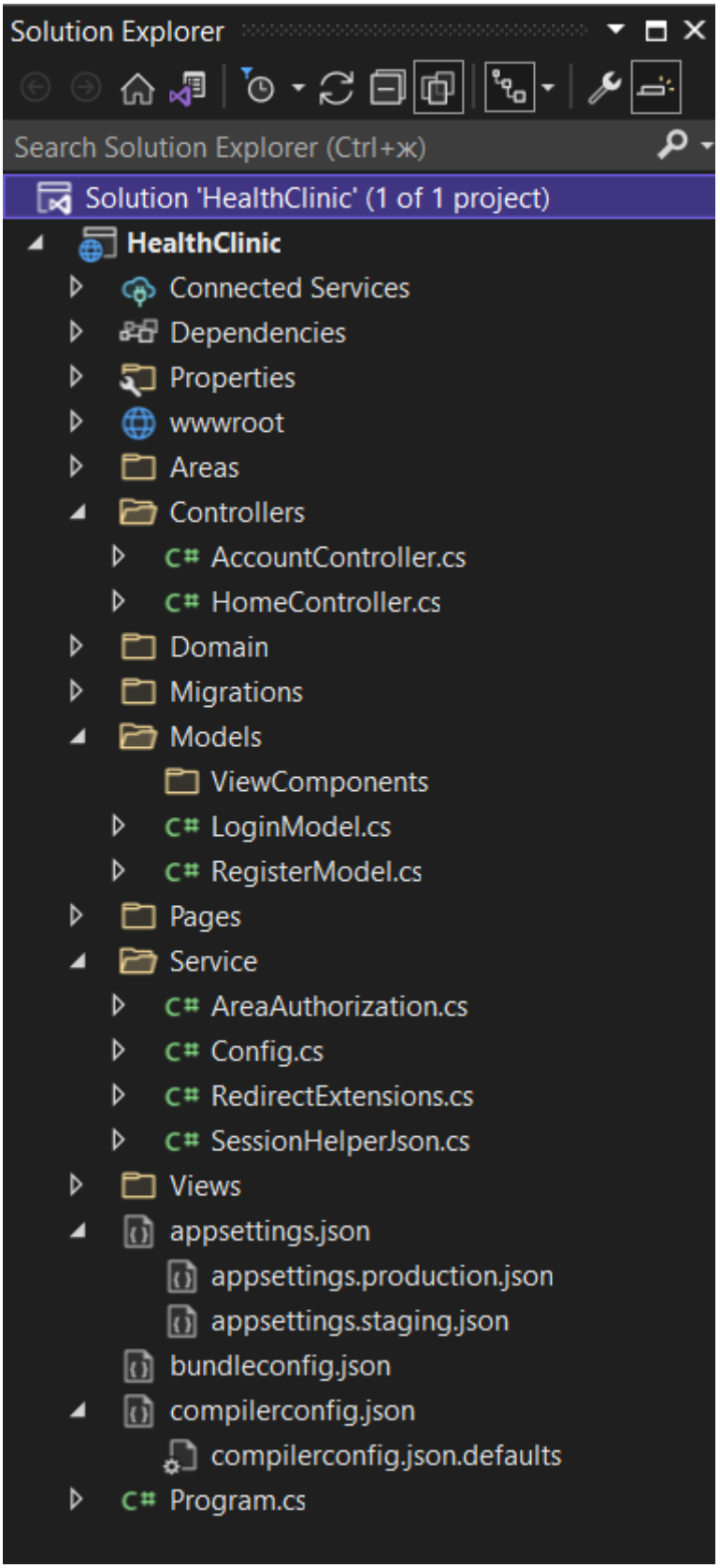
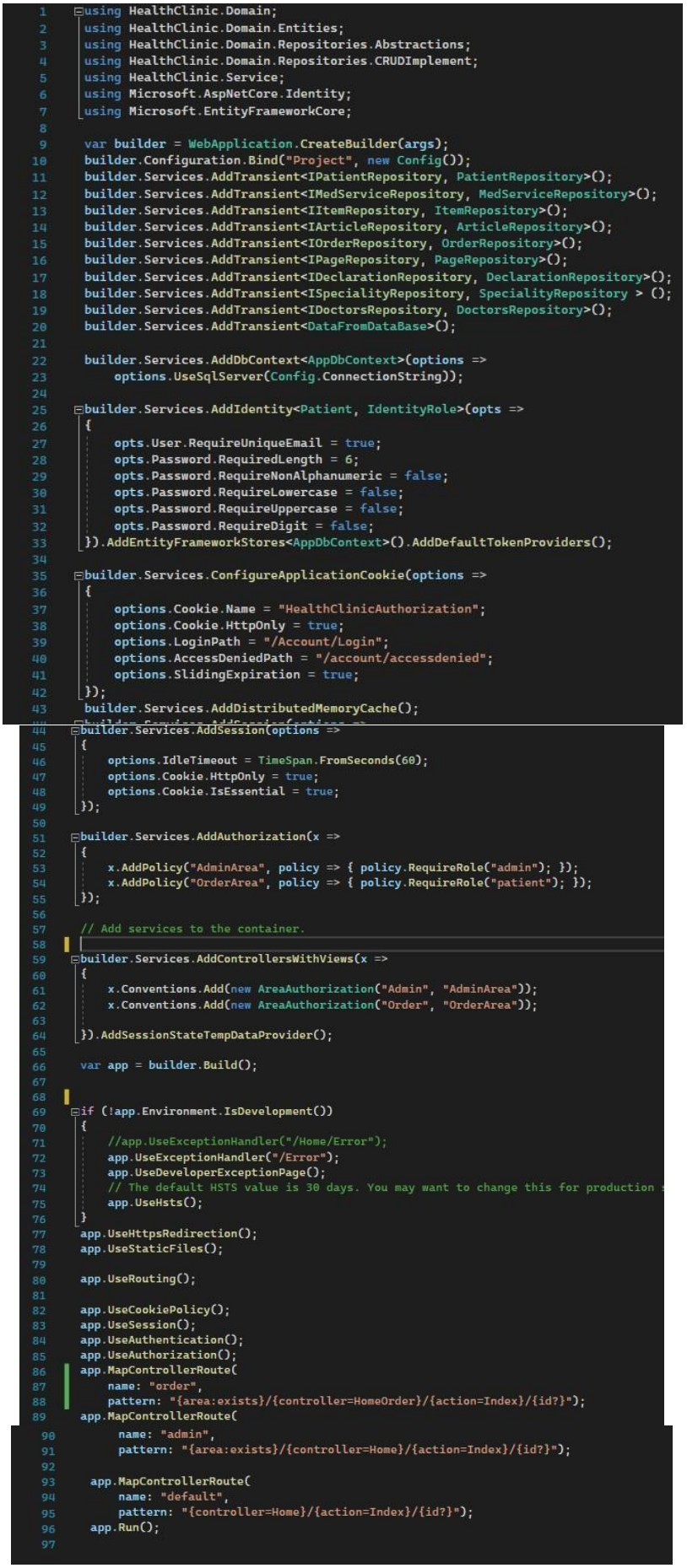


Рисунок 3.5 - Структура проекту



```
using HealthClinic.Domain;
using HealthClinic.Domain.Entities;
using HealthClinic.Domain.Repositories.Abstractions;
using HealthClinic.Domain.Repositories.CRUDImplement;
using HealthClinic.Service;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Configuration.Bind("Project", new Config());
builder.Services.AddTransient<IPatientRepository, PatientRepository>();
builder.Services.AddTransient<IMedServiceRepository, MedServiceRepository>();
builder.Services.AddTransient<IItemRepository, ItemRepository>();
builder.Services.AddTransient<IArticleRepository, ArticleRepository>();
builder.Services.AddTransient<IOrderRepository, OrderRepository>();
builder.Services.AddTransient<IPageRepository, PageRepository>();
builder.Services.AddTransient<IDeclarationRepository, DeclarationRepository>();
builder.Services.AddTransient<ISpecialityRepository, SpecialityRepository > ();
builder.Services.AddTransient<IDoctorsRepository, DoctorsRepository>();
builder.Services.AddTransient<DataFromDataBase>();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(Config.ConnectionString));

builder.Services.AddIdentity<Patient, IdentityRole>(opts =>
{
    opts.User.RequireUniqueEmail = true;
    opts.Password.RequiredLength = 6;
    opts.Password.RequireNonAlphanumeric = false;
    opts.Password.RequireLowercase = false;
    opts.Password.RequireUppercase = false;
    opts.Password.RequireDigit = false;
}).AddEntityFrameworkStores<AppDbContext>().AddDefaultTokenProviders();

builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.Name = "HealthClinicAuthorization";
    options.Cookie.HttpOnly = true;
    options.LoginPath = "/Account/Login";
    options.AccessDeniedPath = "/account/accessdenied";
    options.SlidingExpiration = true;
});
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(60);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

builder.Services.AddAuthorization(x =>
{
    x.AddPolicy("AdminArea", policy => { policy.RequireRole("admin"); });
    x.AddPolicy("OrderArea", policy => { policy.RequireRole("patient"); });
});

// Add services to the container.

builder.Services.AddControllersWithViews(x =>
{
    x.Conventions.Add(new AreaAuthorization("Admin", "AdminArea"));
    x.Conventions.Add(new AreaAuthorization("Order", "OrderArea"));

}).AddSessionStateTempDataProvider();

var app = builder.Build();


if (!app.Environment.IsDevelopment())
{
    //app.UseExceptionHandler("/Home/Error");
    app.UseExceptionHandler("/Error");
    app.UseDeveloperExceptionPage();
    // The default HSTS value is 30 days. You may want to change this for production
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseCookiePolicy();
app.UseSession();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllerRoute(
    name: "order",
    pattern: "{area:exists}/{controller=HomeOrder}/{action=Index}/{id?}");
app.MapControllerRoute(
      name: "admin",
      pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");

 app.MapControllerRoute(
      name: "default",
      pattern: "{controller=Home}/{action=Index}/{id?}");
 app.Run();
```

Рисунок 3.6 - Вміст класу Startup.cs

Під час розробки нашого проекту ми приділили особливу увагу організації структури для зручності управління і підтримки. Ось деякі ключові кроки, які ми здійснили:

Кожна частина нашого застосунку знаходиться у відповідній папці, що сприяє зручній навігації і покращенню розподілення ресурсів. Наприклад, весь конфігураційний код, пов'язаний з базою даних (наприклад, рядок підключення), а також налаштування вебсервера (назва, URL, порт застосунку) і контактна інформація про компанію зберігаються у файлах `appsettings.json` та `launchSettings.json`.

У файлі `HealthClinic.csproj` ми також зберігаємо інформацію про конфігурацію проекту, включаючи маршрути до статичних файлів і список пакетів, які використовуються (встановлені як зовнішні ресурси чи через NuGet).

Як описано в попередньому розділі, у каталозі `Dmain` ми зосередили розроблені доменні моделі (рис.3.7). Окрім цього, у цьому каталозі ми створили дві важливі піддиректорії: `Абстракції` та `Реалізація`.

У підкаталозі `Абстракції` ми визначаємо функціонал для зв’язку з БД, створюючи абстракції, які описують методи і властивості, що використовуються в додатку. Це дозволяє нам розділити логіку взаємодії з даними від конкретної реалізації.

У підкаталозі `Реалізація` ми знаходимося класи, що реалізують ці абстракції для конкретного постачальника даних, наприклад, у папці EntityFramework ми знаходимо класи, які реалізовують взаємодію з БД через Entity Framework.

Така структура дозволяє нам легко і безболісно змінювати постачальників даних у разі необхідності. Наприклад, у випадку переїзду на інший хостинг, який підтримує інший постачальник баз даних, ми можемо змінити лише реалізаційний шар, не змінюючи абстракційний, що робить процес міграції більш безпечним і ефективним.

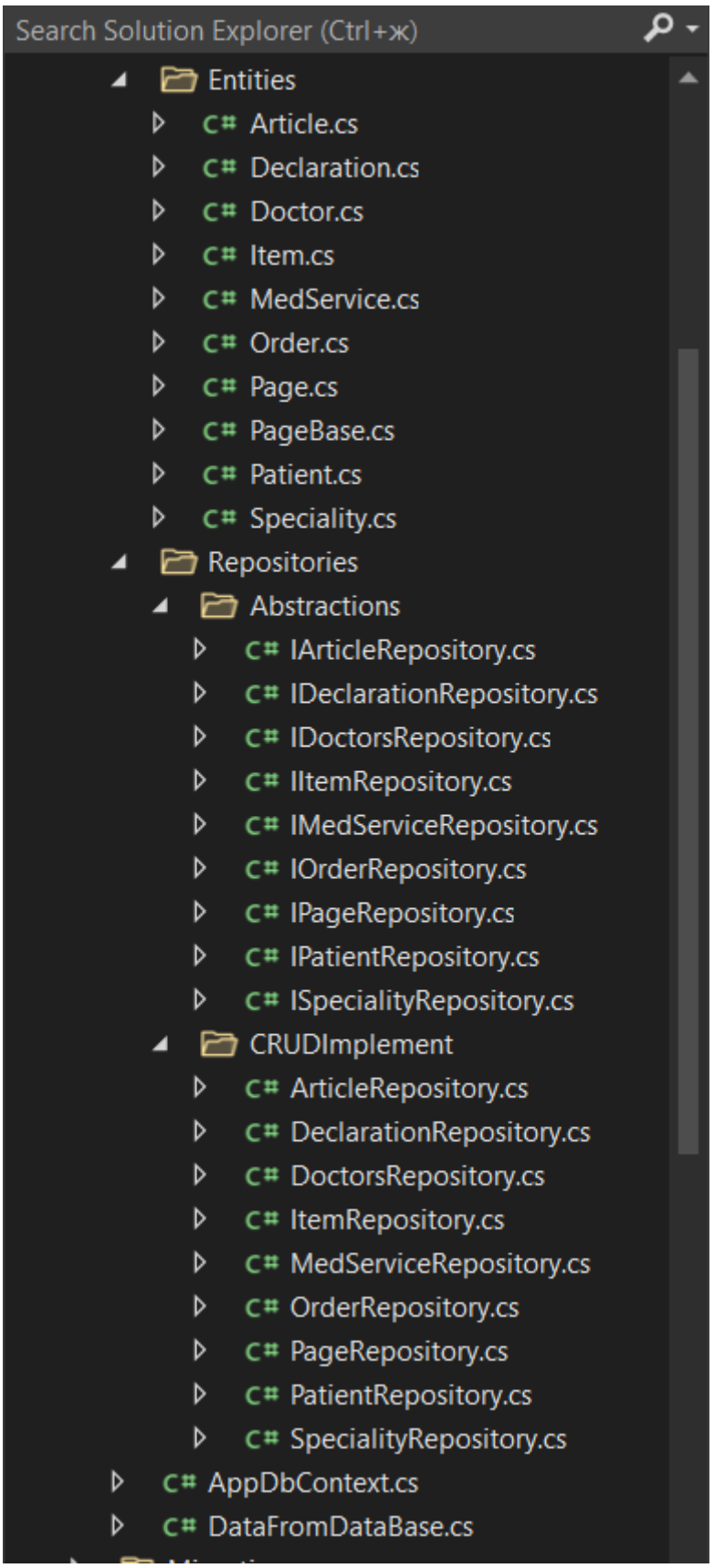


Рисунок 3.6 - Доменні моделі

Під час розробки нашого проекту ми звернули увагу на потребу взаємодії з

базою даних через Entity Framework. Для цього ми створили новий клас у папці `Domain`, який ми назвали `AppDbContext`. Цей клас успадковує від `Microsoft.EntityFrameworkCore.DbContext` і встановлює зв'язок наших сутностей з таблицями БД `HealthClinic`.

`AppDbContext` відіграє центральну роль у нашому додатку, надаючи доступ до даних і забезпечуючи CRUD-операції з доменними об'єктами. Він здійснює взаємодію з базою даних через Entity Framework, обробляючи всі операції збереження, оновлення, видалення і вибірки даних.

Окрім цього, ми використовуємо клас `DataFromDataBase`, який є сервісним компонентом нашого додатку. ` DataFromDataBase ` служить точкою входу до контексту бази даних і керує реалізаціями наших доменних моделей, які також відомі як репозиторії. Цей підхід дозволяє нам ефективно організовувати і керувати доступом до даних відповідно до сучасних практик розробки програмного забезпечення.

Таким чином, ми створили міцну основу для взаємодії з базою даних у нашому додатку, використовуючи підходи, які забезпечують чистоту коду, гнучкість і зручність у підтримці.

Під час розробки нашого проекту ми доклали зусиль для структурування обробки запитів у контролерах. Кожен запит, що надходить до нашого додатку, перш за все, обробляється відповідним контролером. У нашому випадку це HomeController та AccountController, які знаходяться у директорії Controllers проекту "HealthClinic".

Контролери у нашому додатку відповідають за обробку запитів з клієнтської сторони. Вони не зберігають жодних даних та не генерують інтерфейс користувача, але саме вони керують логікою, яка взаємодіє з базою даних та виконує різноманітні операції над даними. Контролери вибирають потрібне представлення для відображення результатів операцій, наприклад, повертаючи вигляд головної сторінки (Layout) або майстер-сторінки (рис.3.8).

Важливою частиною розробки є також налаштування стандартних маршрутів. Наприклад, HomeController обробляє запити стандартного маршруту, де явно не вказані імена контролера та методу. Це дозволяє нам структурувати і управляти

навігацією по додатку ефективно, забезпечуючи користувачам зручний та послідовний досвід використання нашого продукту.

Таким чином, використовуючи контролери у відповідних директоріях і налаштовуючи стандартні маршрути, ми забезпечуємо прозорий та ефективний процес обробки запитів у нашому вебдодатку " HealthClinic".

```
<!DOCTYPE HTML>

<html>
    <head>

        @await Html.PartialAsync("MetatagsPartial.cshtml")
        @await Html.PartialAsync("CssPartial.cshtml")

    </head>
    <body class="homepage is-preload">
        <div id="page-wrapper">

            <!-- Header -->
            @await Html.PartialAsync("HeaderPartial.cshtml")

            <!-- Main -->
                <section id="main">
                    <div class="container">
                        <div class="row">
                            <div class="col-12">

                                <!-- MainContent -->
                                    @RenderBody()

                            </div>
                            <div class="col-12">

                                <!-- Department -->
                                    @await Html.PartialAsync("DepartmentPartial.cshtml")

                            </div>
                            <div class="col-12">

                                <!-- News -->
                                    @await Html.PartialAsync("NewsPartial.cshtml")

                            </div>
                        </div>
                    </div>
                </section>

            <!-- Footer -->
            @await Html.PartialAsync("FooterPartial.cshtml")

        </div>

        <!-- Scripts -->
            @await Html.PartialAsync("ScriptsPartial.cshtml")

    </body>
</html>
```

Рисунок 3.8 - Код сторінки _Layout.cshtml

Під час розробки нашого вебдодатку ми активно використовували концепцію майстер-сторінок для організації і структурування виглядів. На стартовій сторінці, як показано на рисунку 3.8, центральна частина формується динамічно і підставляється за допомогою виклику @RenderBody().

Одним із ключових аспектів нашого підходу є використання часткових представлень, які також підключені до майстер-сторінки, як показано на рисунку 3.8. Це дозволяє нам ефективно організовувати розмітку та відокремлювати компоненти інтерфейсу, що сприяє покращенню супроводженості коду та зменшує дублювання. Кожне часткове представлення відповідає за конкретну частину UI, наприклад, меню навігації, футер або блок зі статичним контентом.

Цей підхід не лише полегшує управління великими обсягами коду, але і робить проект більш масштабованим і гнучким. Він дозволяє швидше знаходити відповідні фрагменти коду та забезпечує консистентність у вигляді всього додатку. Таким чином, ми досягли збалансованого підходу до створення і підтримки веб-інтерфейсу нашого проекту " HealthClinic ".

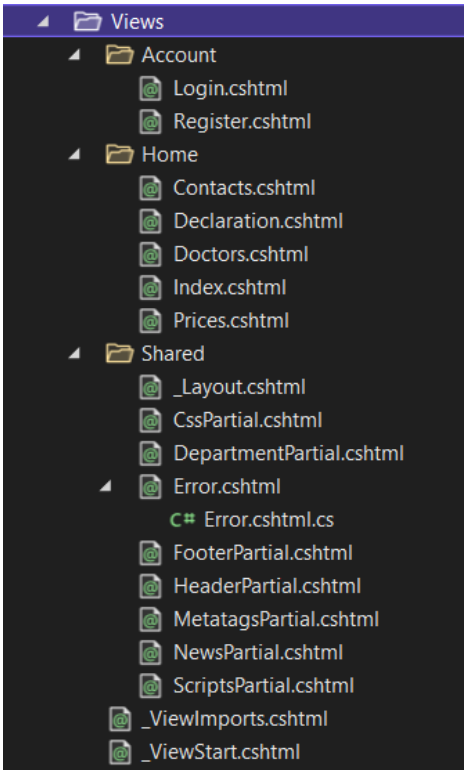


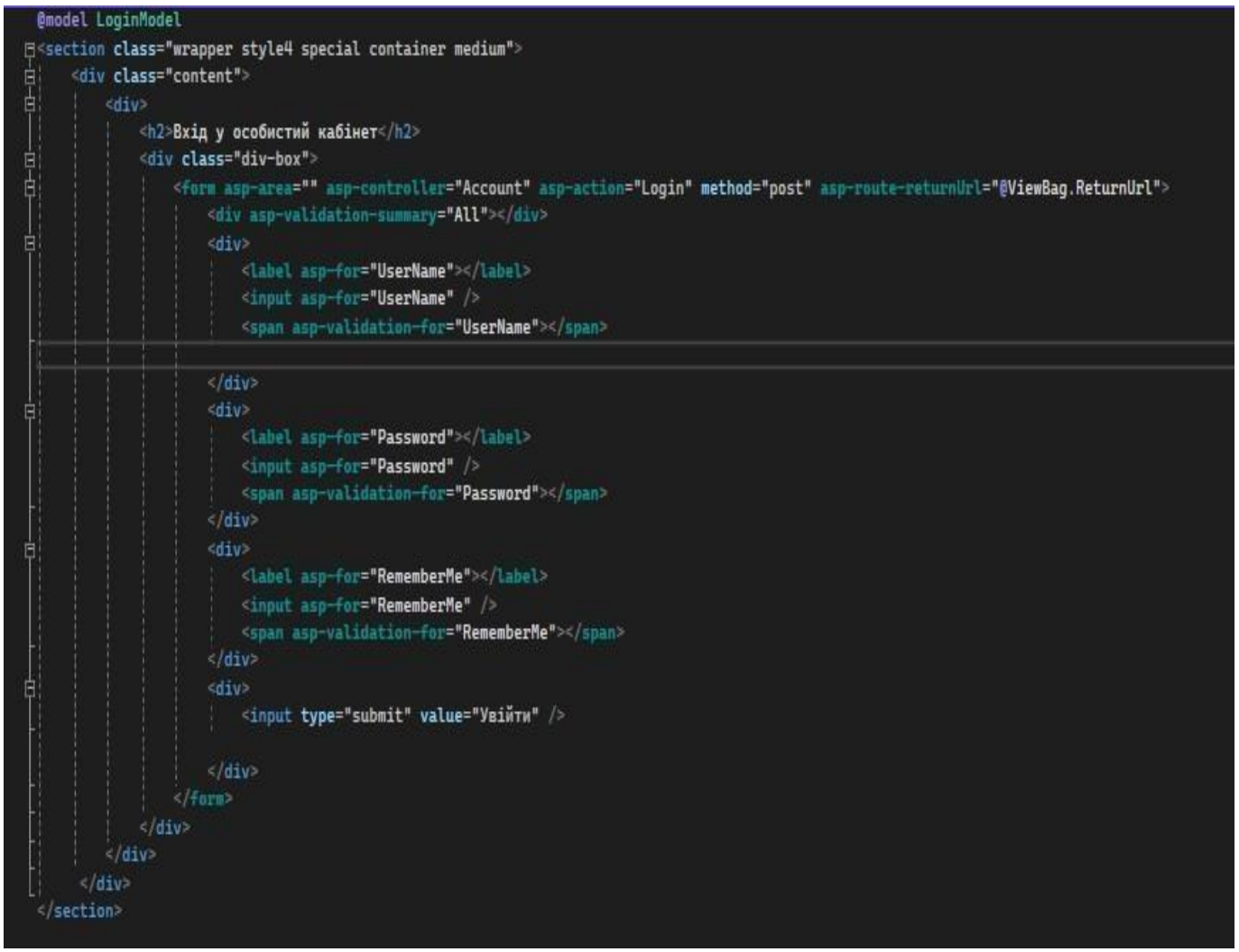
Рисунок 3.9 - Вигляд директорії Views з частковими представленнями

Під час розробки нашого вебдодатку ми активно використовували контроллери для обробки різних аспектів функціоналу. Один з ключових контролерів - AccountController, відповідає за реалізацію логіки аутентифікації та логінізації користувачів. Метод контроллера, що відповідає за процес логіну, зображено на рисунку 3.10. Для створення інтерфейсу форми логіну ми використали представлення, зображене на рисунку 3.11, яке сформоване за допомогою tag-хелперів. У контроллері реалізовано методи для реєстрації нових пацієнтів (рис.3.12).

```
82          [HttpGet]
83          [AllowAnonymous]
            0 references
84          public IActionResult Login(string returnUrl)
85          {
86              ViewBag.ReturnUrl = returnUrl;
87              return View(new LoginModel { ReturnUrl = returnUrl });
88          }
89          [HttpPost]
90          [AllowAnonymous]
            0 references
91          public async Task<IActionResult> Login(LoginModel model)
92          {
93
94              if (ModelState.IsValid)
95              {
96                  IdentityUser user = await userManager.FindByNameAsync(model.UserName);
97                  if (user != null)
98                  {
99                      await signInManager.SignOutAsync();
100                     Microsoft.AspNetCore.Identity.SignInResult result = await signInManager.PasswordSignInAsync((Patient)user, model.Password, model.RememberMe, false);
101                     if (result.Succeeded)
102                     {
103                         return Redirect(model.ReturnUrl ?? "/");
104                     }
105                 }
106                 ModelState.AddModelError(nameof(LoginModel.UserName), "Невірний логін або пароль");
107             }
108             return View(model);
109         }
110
111         [Authorize]
            0 references
112         public async Task<IActionResult> Logout()
113         {
114             await signInManager.SignOutAsync();
115             return RedirectToAction("Index", "Home");
116         }
117     }
118
119 }
```

Рисунок 3.10 - Методи контроллера, що відповідає за процес логіну

Tag-хелпери є зручним інструментом для генерації HTML-елементів у ASP.NET Core, оскільки вони надають більш чистий та зрозумілий код порівняно з традиційними HTML-хелперами. Використання таких інструментів спрощує супроводження та підтримку коду, а також забезпечує підтримку IntelliSense у середовищі розробки, наприклад, у Visual Studio.



```
@model LoginModel
<section class="wrapper style4 special container medium">
    <div class="content">
        <div>
            <h2>Вхід у особистий кабінет</h2>
            <div class="div-box">
                <form asp-area="" asp-controller="Account" asp-action="Login" method="post" asp-route-returnUrl="@ViewBag.ReturnUrl">
                    <div asp-validation-summary="All"></div>
                    <div>
                        <label asp-for="UserName"></label>
                        <input asp-for="UserName" />
                        <span asp-validation-for="UserName"></span>

                    </div>
                    <div>
                        <label asp-for="Password"></label>
                        <input asp-for="Password" />
                        <span asp-validation-for="Password"></span>
                    </div>
                    <div>
                        <label asp-for="RememberMe"></label>
                        <input asp-for="RememberMe" />
                        <span asp-validation-for="RememberMe"></span>
                    </div>
                    <div>
                        <input type="submit" value="Увійти" />

                    </div>
                </form>
            </div>
        </div>
    </div>
</section>
```

Рисунок 3.11- Представлення для логіну

HomeController, відповідає за обробку та відображення даних, пов'язаних з відкритими даними на сайті, такими сторінки напрямки, лікарі, декларація, ціни. Адміністратор може створювати та редагувати сторінки та статті у спеціальній адмін-панелі, а HomeController забезпечує їх вивід на вебсторінці, передаючи відповідні дані у потрібні представлення. Подібний функціонал представлений при виводі статей про лікарів під час вибору лікаря для запису на прийом (рис. 3.12).

Цей підхід дозволяє нам ефективно розподіляти функціонал за допомогою контроллерів, зберігаючи чітку відповідальність за різними частинами нашого додатку і забезпечуючи логічне розділення між вебінтерфейсом та бізнес-логікою.

```
@{
    string strTitle = "Оформлення замовлення";
    ViewBag.Title = strTitle;
    string str1 = "";
    string str2 = "";
    string str3 = "";
    string str4 = "";
    string str5 = "";
    int i = 0;
    IQueryable < Speciality > specialities=  ViewBag.Specialitys;

}
<section class="wrapper style4 special container medium">
    <div class="content">
        <div>
            <h2>@strTitle</h2>
            <div>
                <h3>Оберіть потрібні опції</h3>
                    <section>
                      <form class="form-inline">
                          <label>Оберіть напрямок</label>
                          <select name="speciality" class="form-control">

                                  @foreach(Speciality  spec in specialities)
                                  {
                                      <option value="@spec.Id">@spec.Name</option>
                                  }

                          </select>
                          <input type="submit" class="button primary" value="Обрати напрямок" style="min-width: 10em;line-height: 1em;" />
                      </form>
                      <br />
                    </section>
                <div class="div-box">
  @if (Model.Any())
                    {<div class="row">
                      @foreach (Doctor doctor in Model)
                      {

                          <div class="col-4 col-6-medium col-12-small">
                          <section class="box">
                              <a href="#" class="image featured"><img src="~/images/@doctor.Association" alt="" /></a>
                              <header>
                                  <h3>@doctor.LastName @doctor.FirstName @doctor.SecondName</h3>
                                  <h4> @switch (@doctor.SpecialityId)
                                  {   case 1:  str3="Педіатрія" ;break;
                                      case 2:  str3="Сімейна медицина" ;break;
                                      case 3:  str3="Гастроентерологія" ;break;
                                      case 4:  str3="Гінекологія" ;break;
                                      case 5:  str3="Дерматовенерологія" ;break;
                                      case 6:  str3="Дістологія" ;break;
                                      case 7:  str3="Ендокринологія" ;break;
                                      case 8:  str3="Кардіологія" ;break;
                                      case 9:  str3="Неврологія" ;break;
                                      case 10:  str3="Офтальмологія" ;break;
                                      case 11:  str3="Стоматологія" ;break;
                                      case 12:  str3="Хірургія" ;break;
                                      case 13:  str3="Анастезіологія" ;break;
                                      default:str3=" немає";break;
                                  } @str3</h4>
                              </header>
                              <p>Кваліфікація:@switch (@doctor.Qualification)
                                  {
                                      case 0:  str2=" немає";break;
                                      case 1:  str2=" перша кваліфікаційна категорія" ;break;
                                      case 2:  str2=" друга кваліфікаційна категорія" ;break;
                                      case 3:  str2=" третя кваліфікаційна категорія" ;break;
                                      default:str2=" немає";break;
                                  }
                                  @str2<br/>
                                  Досвід роботи:@doctor.Experience років
                                 <br/>
                                  Основні напрямки лікувальної роботи: @doctor.DirectionTreatment
                              </p>
                              <footer>
                                  <ul class="actions">
                                      <li> <form asp-area="Order" asp-controllar="Cart" asp-action="Buy" asp-route-id="@doctor.Id" method="post">
                                      <input class="button alt" type="submit" value="Записатись" style="min-width: 8em;line-height: 1em;"/>
                                  </form></li>
                                  </ul>
                              </footer>
                          </section>
                      </div>
```

Рисунок 3.12 - Представлення, що реалізує вивід статтей про лікарів

Структуру проекту з реалізованою адмін-панеллю можна побачити на рисунку 3.13.

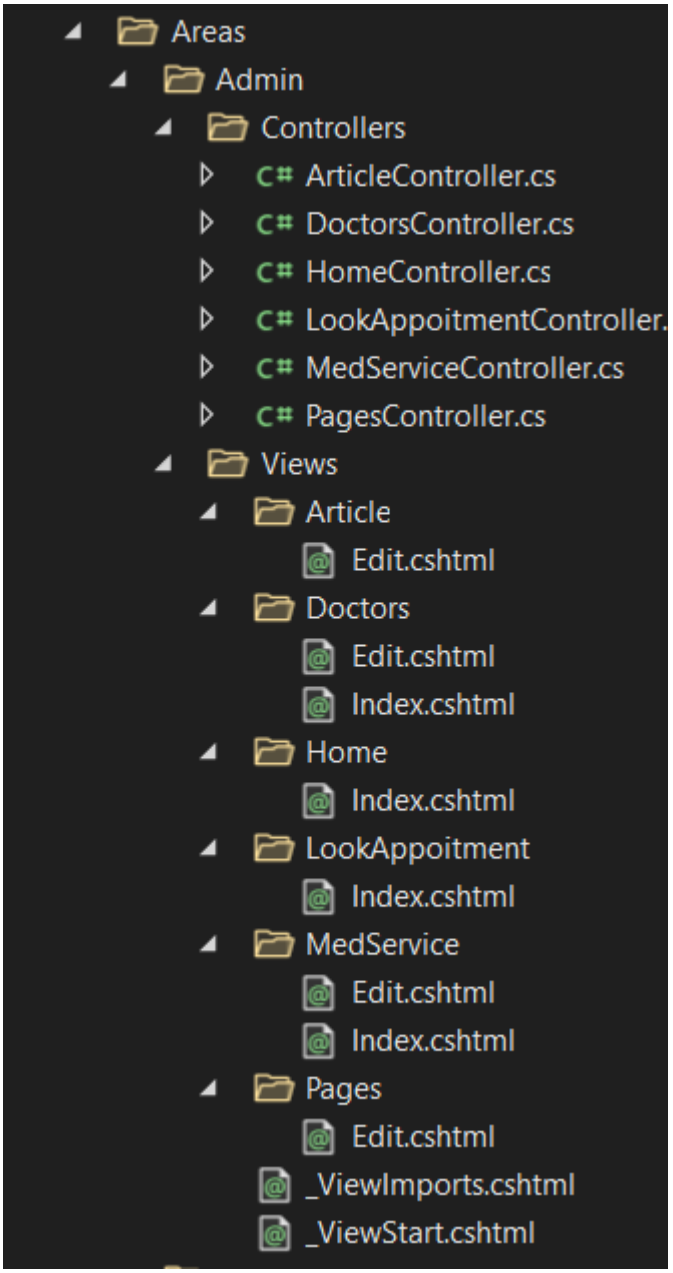


Рисунок 3.13 - Реалізація адмін-панелі

Під час розробки нашого проекту ми реалізували режим адміністратора, який забезпечує широкий функціонал. Адмін-панель включає шість контроллерів, кожен з яких відповідає за певний аспект управління сайтом.

HomeController передає модель MedService у представлення, яке візуалізує панель адміністратора і надає доступ до основного функціоналу. Код вигляду цього представлення зображено на рисунку 3.14.

ArticleController відповідає за редагування, додавання та видалення статей (рис.3.15). MedServiceController надає аналогічний функціонал медичних послуг на сайту – додавання, редагування, видалення.

PageController дозволяє видаляти та редагувати інформаційні сторінки сайту, такі як «Напрямки», «Контакти», «Лікарі» тошо. LookAppointmentController надає можливість переглядати інформацію про записи на прийом, зроблені клієнтами.

Таким чином, ми створили адмін-панель, яка забезпечує ефективне управління контентом та послугами сайту, надаючи адміністраторам необхідні інструменти для підтримки і розвитку ресурсу.

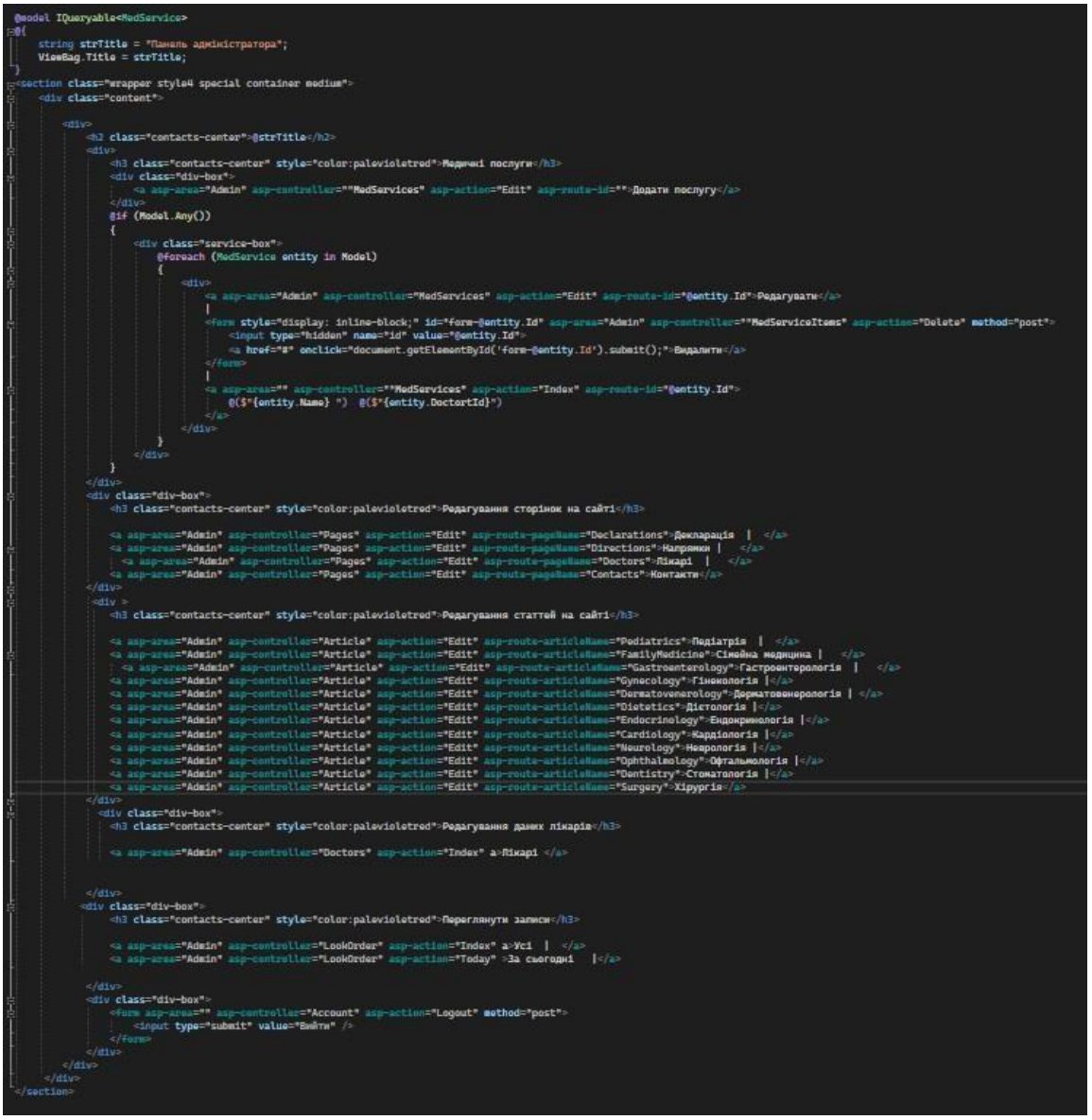


Рисунок 3.14 - Подання для адмін-панелі

```
using HealthClinic.Domain;
using HealthClinic.Domain.Entities;
using HealthClinic.Service;
using Microsoft.AspNetCore.Mvc;

namespace HealthClinic.Areas.Admin.Controllers
{
    [Area("Admin")]

    1 reference
    public class ArticleController : Controller
    {
        private readonly DataFromDataBase dataFromDataBase;
        private readonly IWebHostEnvironment hostingEnvironment;
        0 references
        public ArticleController(DataFromDataBase dataFromDataBase, IWebHostEnvironment hostingEnvironment)
        {
            this.dataFromDataBase = dataFromDataBase;
            this.hostingEnvironment = hostingEnvironment;
        }

        0 references
        public IActionResult Edit(string articleName)
        {
            var entity = dataFromDataBase.Articles.GetArticleByName(articleName);
            return View(entity);
        }

        [HttpPost]
        0 references
        public IActionResult Edit(Article model, IFormFile titleImageFile)
        {
            if (ModelState.IsValid)
            {
                if (titleImageFile != null)
                {
                    model.TitleImagePath = titleImageFile.FileName;
                    using (var stream = new FileStream(Path.Combine(hostingEnvironment.WebRootPath, "images/", titleImageFile.FileName), FileMode.Create))
                    {
                        titleImageFile.CopyTo(stream);
                    }
                }
                dataFromDataBase.Articles.SaveArticle(model);
                return RedirectToAction(nameof(HomeController.Index), nameof(HomeController).CutController());
            }
            return View(model);
        }
    }
}
```

Рисунок 3.15 — Article-контроллер

Структура проекту, що реалізує функціонал запису до лікаря представлена на рисунку 3.16.

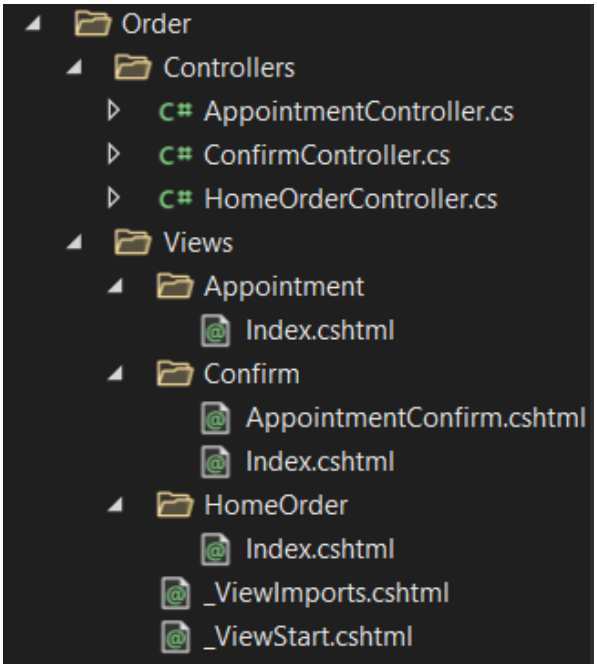


Рисунок 3.16 - Структура проекту для реалізації запису

Після того, як користувач обрав всі товари для замовлення, йому пропонується підтвердити, чи дійсно він бажає здійснити замовлення. Керування переходить до ConfirmController. Цей контролер реалізує логіку занесення до бази даних нового замовлення та перевірки користувачем його особистих даних.

Під час розробки ми створили функціональність запису до лікаря, яка зосереджена в директорії Order. Всі контроллери цієї директорії позначені атрибутом Areas, що означає, що доступ до них мають тільки авторизовані користувачі з певною роллю.

HomeOrderController відповідає за формування даних про лікарів і представлення даних при них користувачу, щоб він міг вибрати лікаря і побачити доступні для нього медичні послуги, що надає даний лікар. Також цей контролер надає можливість здійснити вибір послуги. Після цього управління переходить до контроллера (AppointmentController) (рис.3.17), який реалізує логіку схожу до логіки споживчого кошика, а саме для здійснення запису пацієнта для лікаря. Для реалізації цієї функціональності ми використали сесії.

Сесія – це серія послідовних запитів, які здійснюються в одному веббраузері протягом певного часу. Вона використовується для збереження тимчасових даних, доступних під час роботи користувача з додатком, але які не потребують постійного зберігання. Для збереження стану сесії на сервері створюється словник або хеш-таблиця, що зберігається в кеші та існує для всіх запитів, які надходять з одного веббраузера протягом певного часу. Ідентифікатор сесії зберігається в куках на клієнті і посилається на сервер з кожним запитом. Сервер використовує цей ідентифікатор для отримання необхідних даних із сесії. Куки видаляються після завершення сесії, або якщо сервер отримує куки для вже закінченої сесії, створюється нова сесія [10].

Після того, як записався до всіх лікарів, які йому були необхідні, йому пропонується підтвердити свій запис. Управління переходить до ConfirmController, який реалізує логіку занесення нового запису до бази даних та перевірку особистих даних пацієнта.

Рисунок 3.17 - AppoinmentController

3.3 Інтерфейс клієнтської частини

Під час розробки ми створили зручний інтерфейс майстер-сторінки сайту. Дизайн сайту стриманий та спокійний, а структура інтуїтивно зрозуміла, що дозволяє користувачам швидко знайти потрібну інформацію.

Навігаційне меню реалізовано у хедері(верхній чпстині) сайту, яке дозволяє перейти до більшості функціональних можливостей системи. Також у верхній частині сайту знаходяться кнопки для реєстрації та логінації клієнтів. Нижче розташована секція, що демонструє переваги клініки. У цій же секції продубльовано кнопки для реєстрації та логіну. Далі йде основна частина сайту.

В основній частині сайту розміщено інформацію про напрямки, тобто медичні спеціалізації, які представлені у клініці. Медичні спеціалізації охоплюють різні галузі медицини, де лікарі та інші медичні працівники зосереджуються на певних аспектах охорони здоров'я. Ось деякі основні медичні спеціалізації присутні на сайті: кардіологія, неврологія, гастроентерологія, дерматологія, ендокринологія, педіатрія, офтальмологія, сімейна медицина, пульмонологія, ревматологія, онкологія, психіатрія, хірургія, акушерство та гінекологія. Трохи нижче наведено інформацію про новини, тут розміщаються й статті загального характеру. Кожну новину-статтю можна переглянути окремо або почитати коментарі до неї.

У футері сайту розташовані кнопки переходу до соціальних мереж та інформація про розробника, а також контактна інформація, схема доїзду та мітки з зазначеними датами акцій, наявнії у клініці.

Також є кнопка , яка перекидає на іншу сторінку, щоб залишити відгук про діяльність клініки.

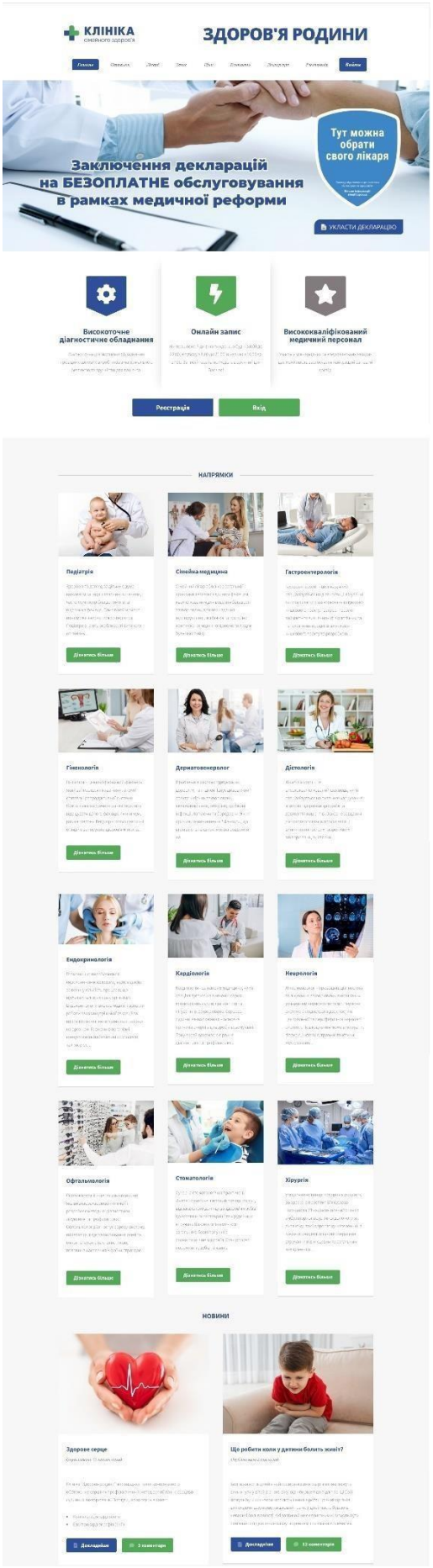


Рисунок 3.18 - Головна сторінка сайту (частина 1)

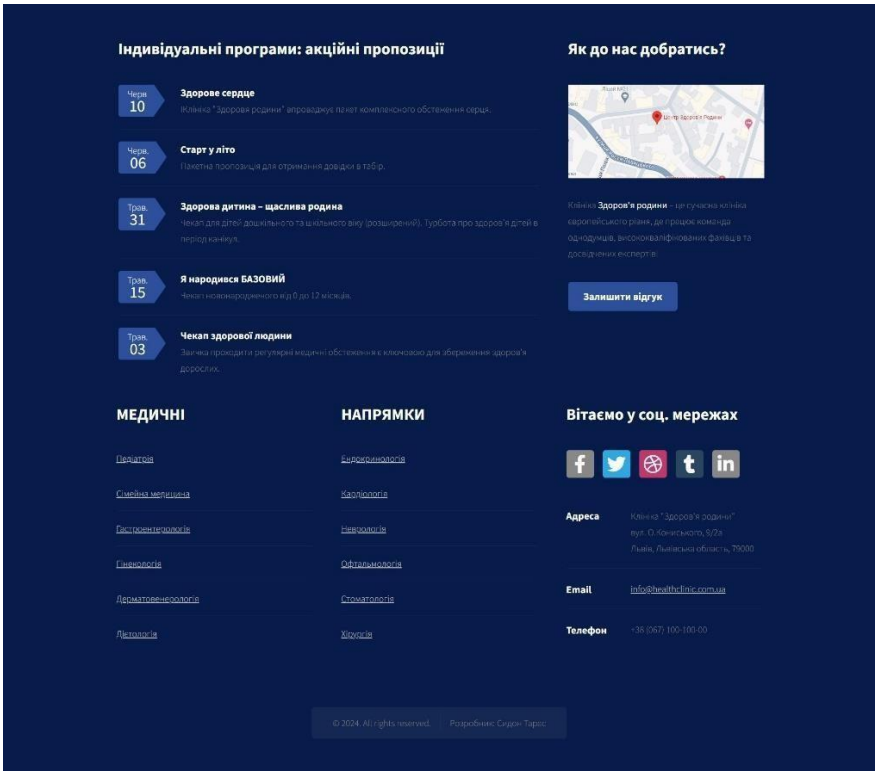


Рисунок 3.19 - Головна сторінка сайту (частина 2 - footer)

Щоб отримати доступ до адмін-панелі, потрібно ввести відповідний URL (рис.3.20) у рядку пошуку та увійти як адміністратор (рис.3.21).

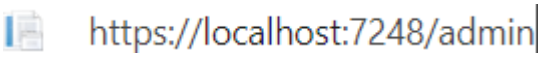


Рисунок 3.20 - Адресна стрічка

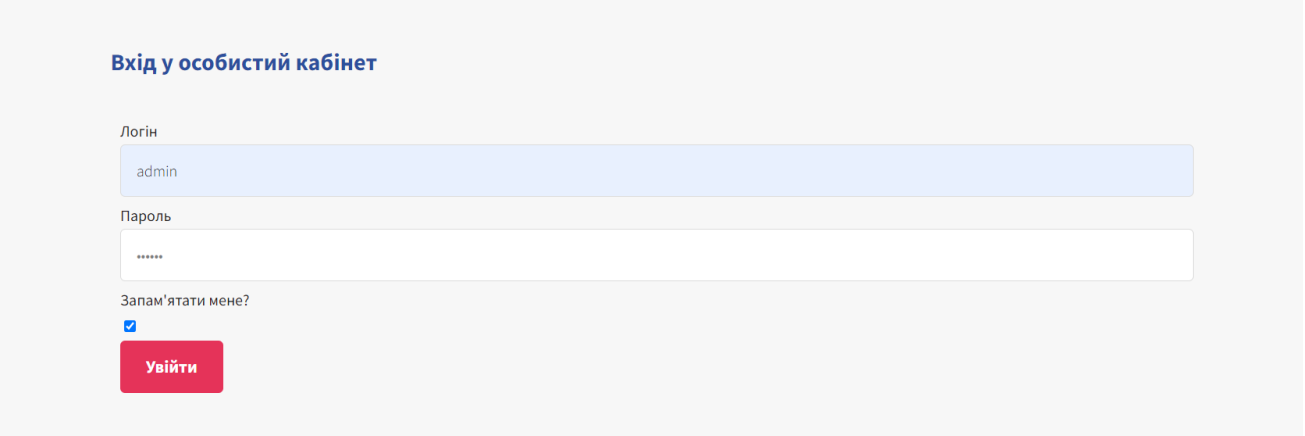


Рисунок 3.21 - Вхід у систему як адміністратор

При невірно введених даних спрацьовує захисна система та видається відповідне повідомлення (рис.3.22).

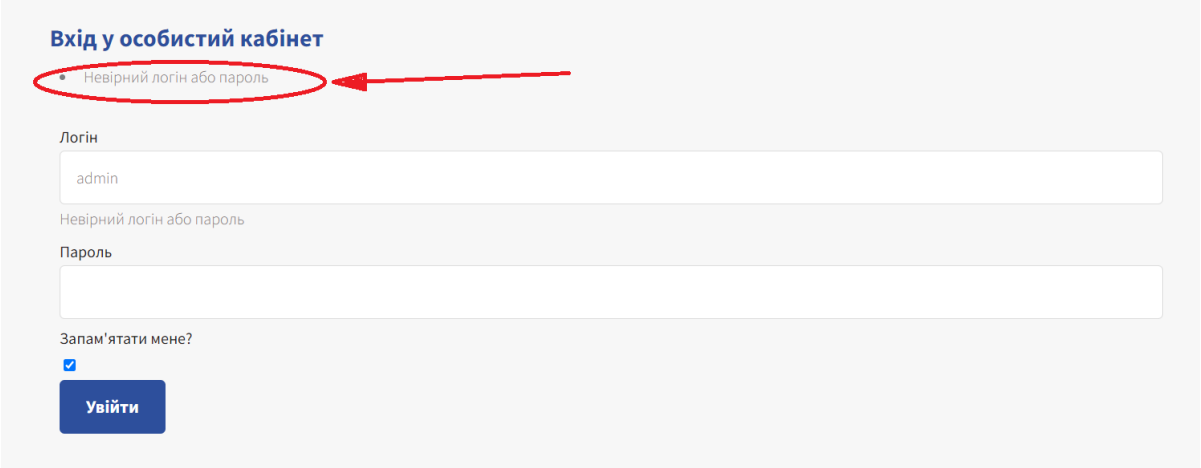


Рисунок 3.22 – Спрацювання захисної системи

Після виконання зазначених дій система перенаправить вас до панелі адміністрування, яка виглядає так, як показано на рис. 3.23.

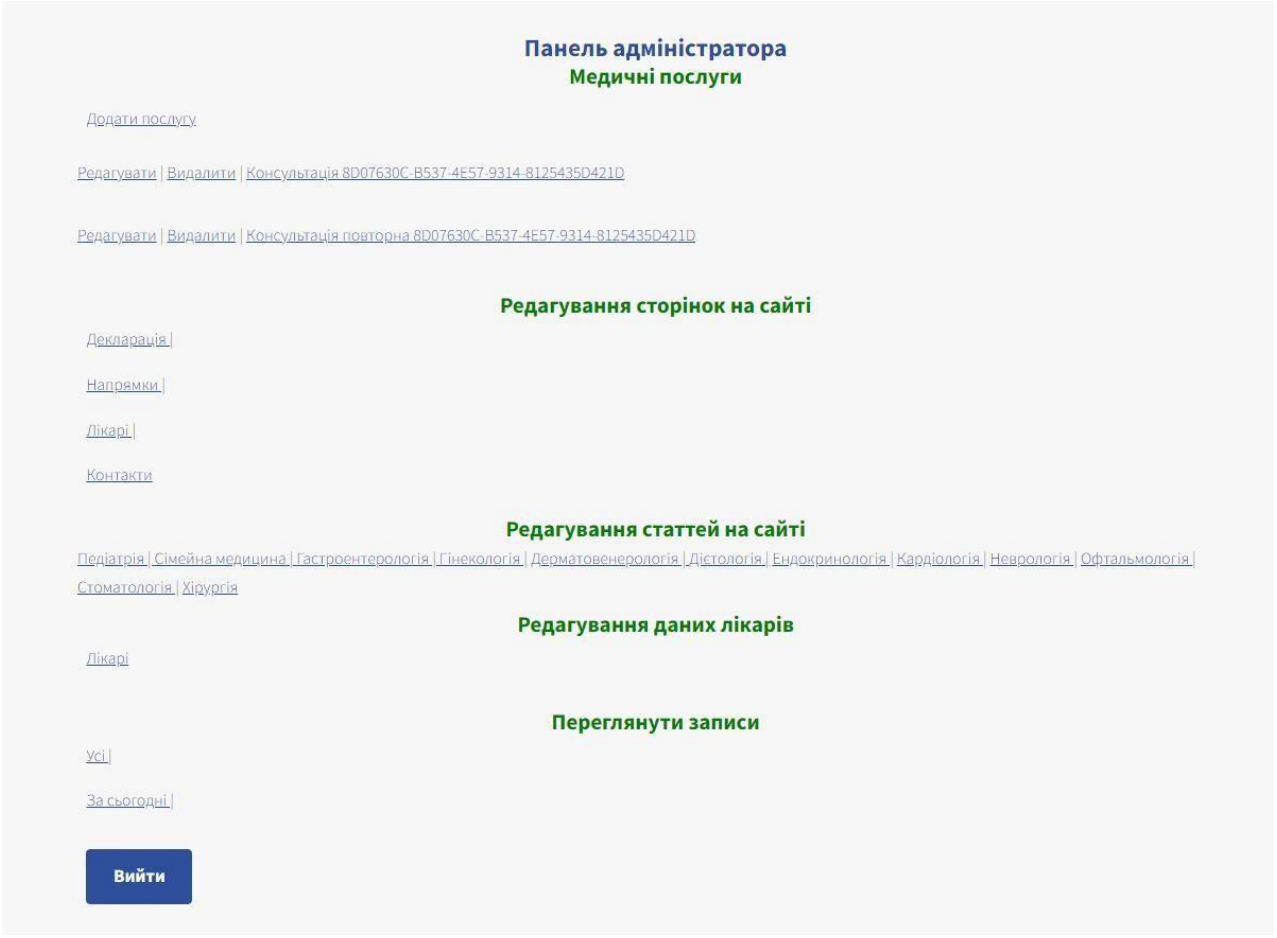


Рисунок 3.23 – Панель адміністрування

Після авторизації адміністратор сайту отримує можливість додавати, видаляти та редагувати статті на сайті, а також змінювати на сайті сторінки (рис. 3.24)

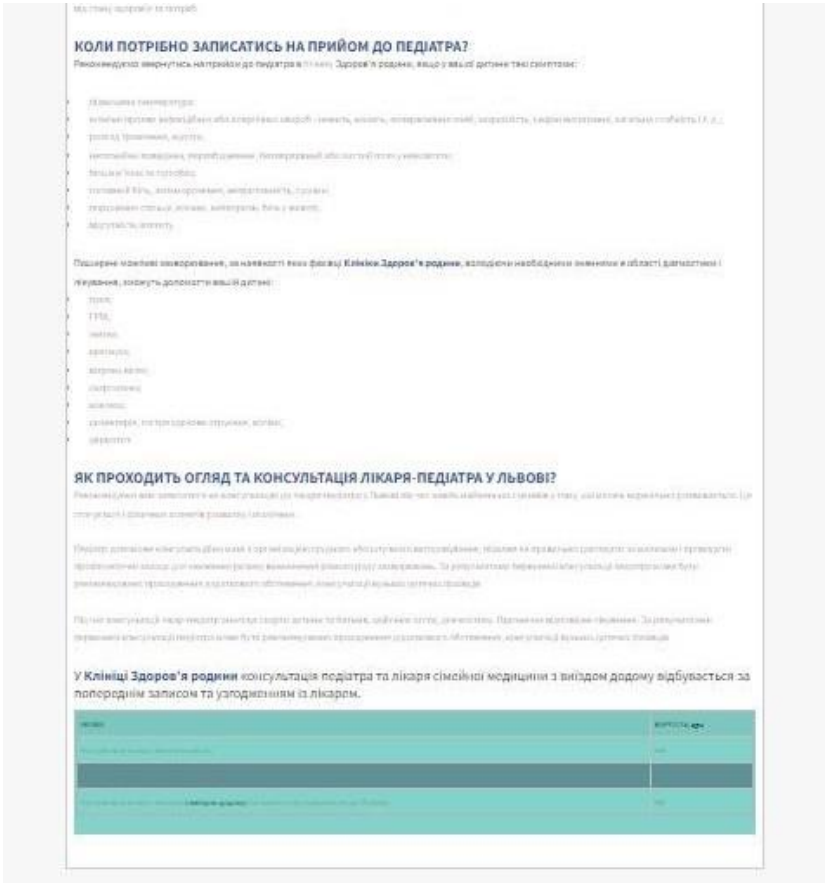


Рисунок 3.24 - Створення нової статті про «Педіатрію»

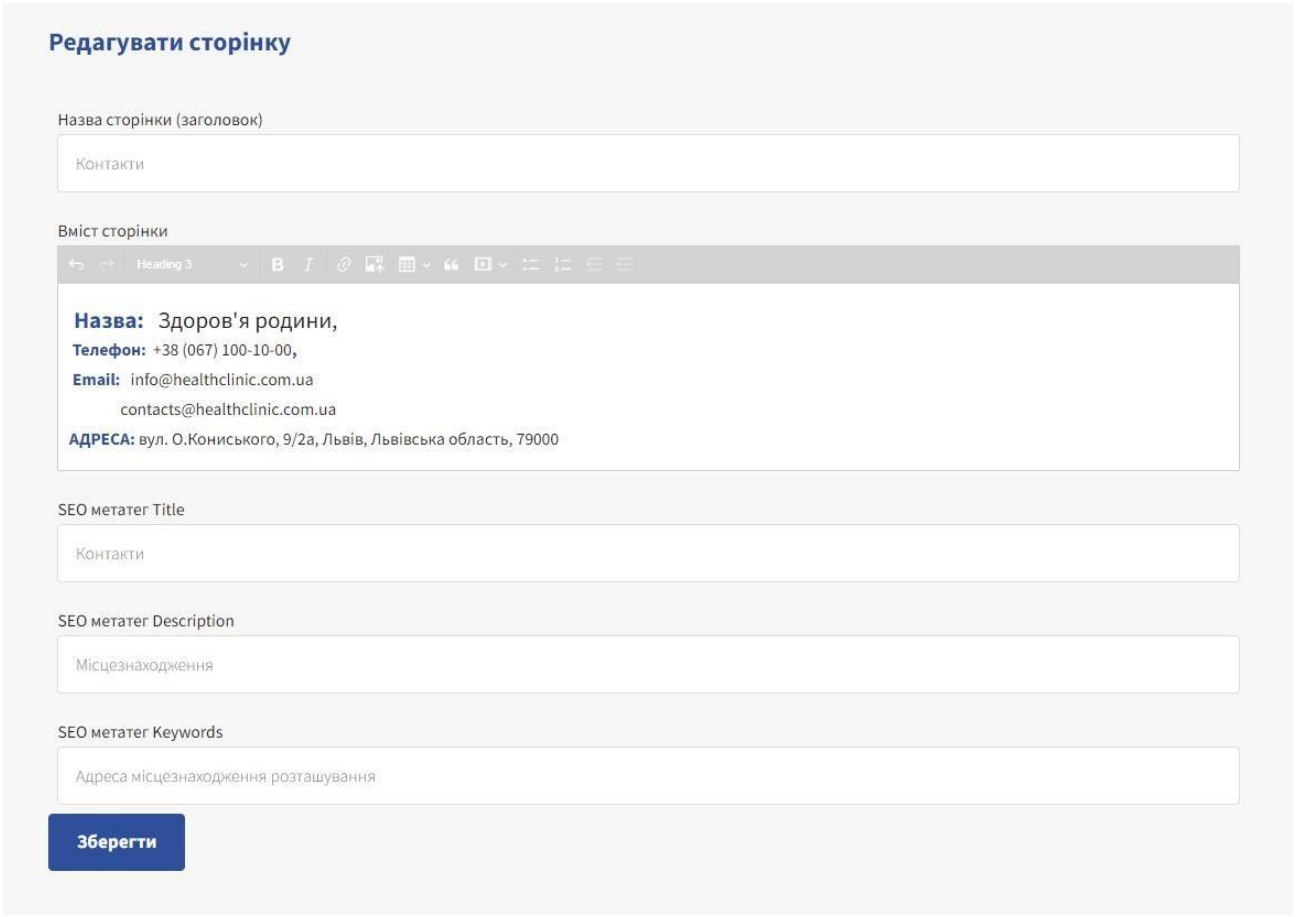


Рисунок 3.25 - Редагування сторінки «Контакти»

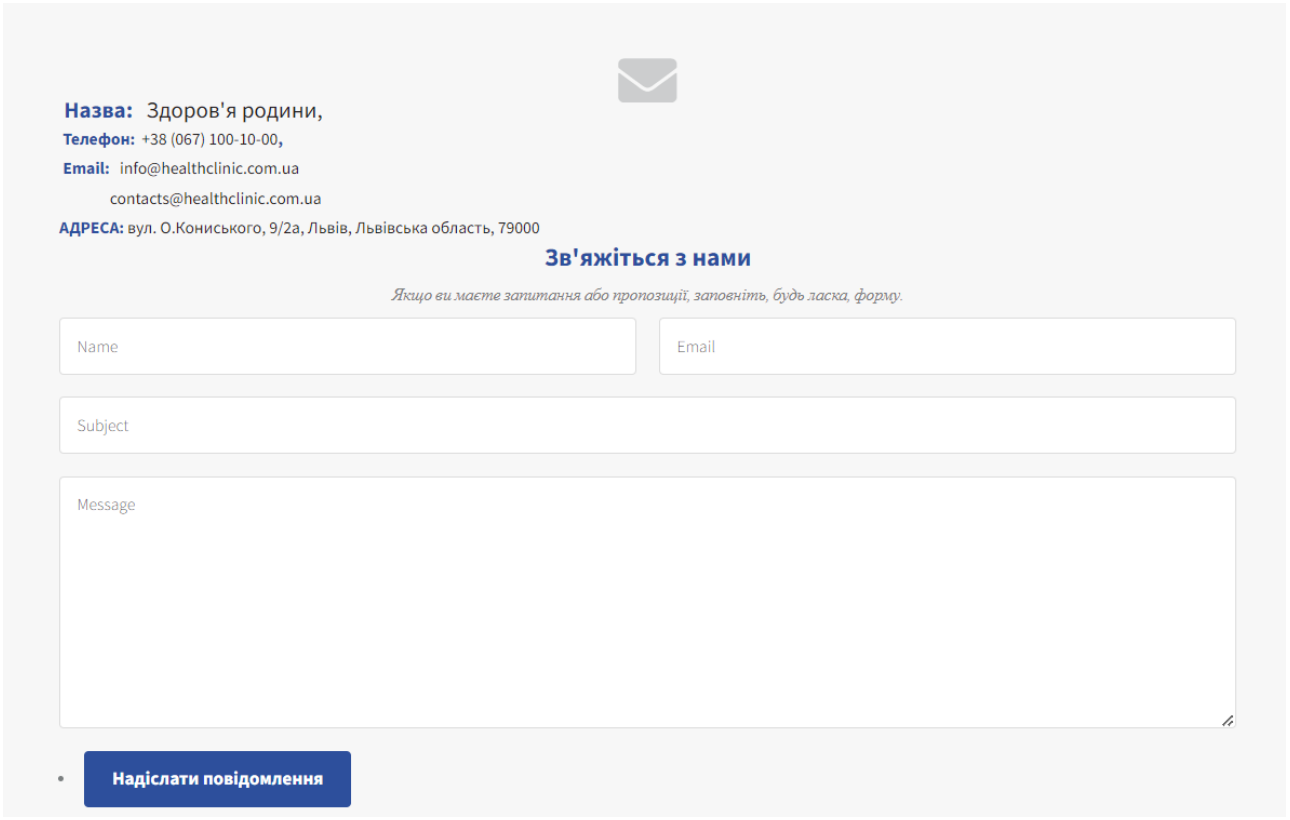


Рисунок 3.26 – Результат редагування сторінки «Контакти»

Окремий розділ призначений для роботи з сутностями «Лікарі». При виборі «Лікарі» відкривається меню для додавання або редагування інформації про лікарів. Вигляд цієї сторінки представлено на рисунку 3.27.

.

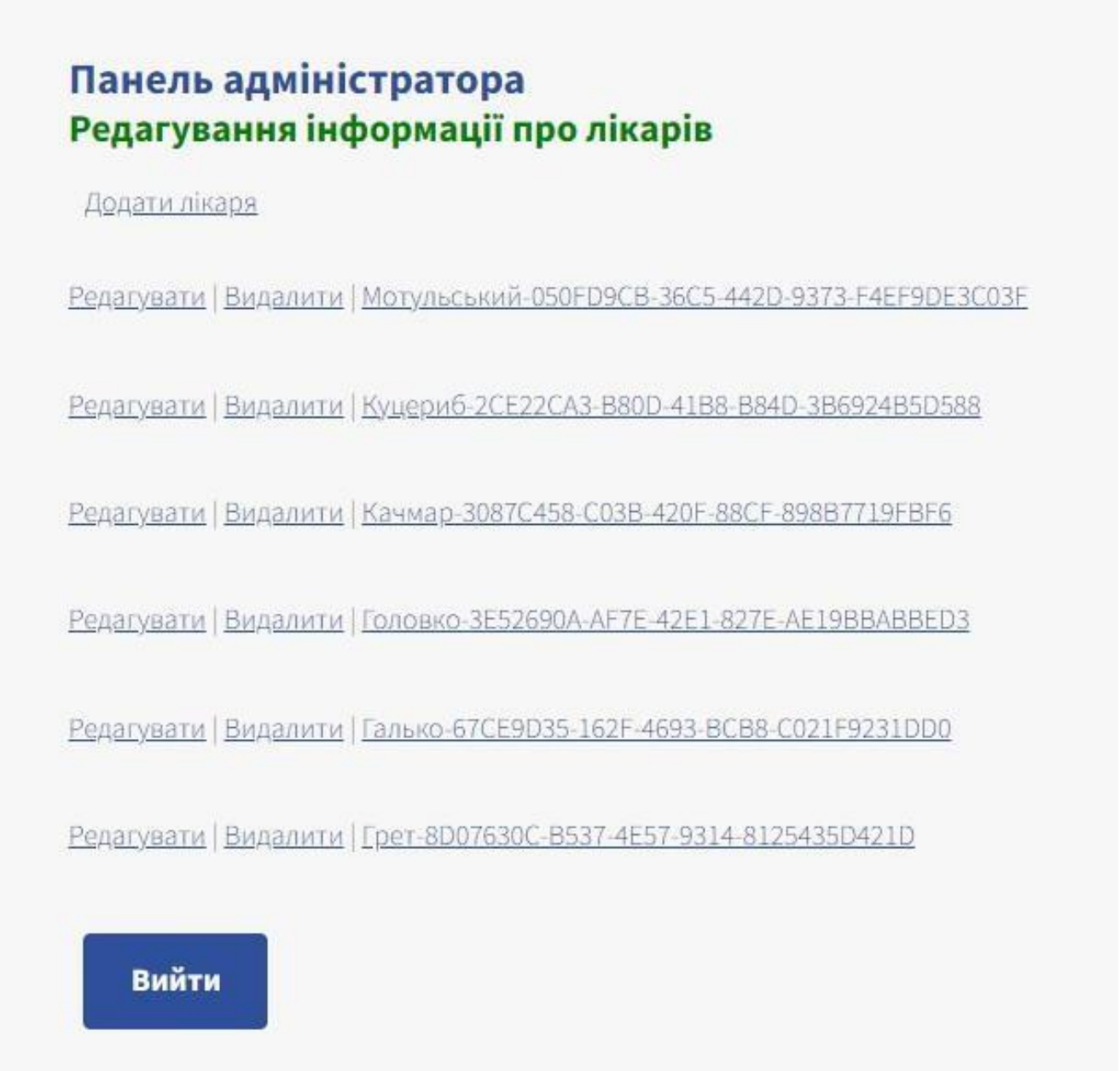


Рисунок 3.27 - Вигляд сторінки вибору даних лікарів

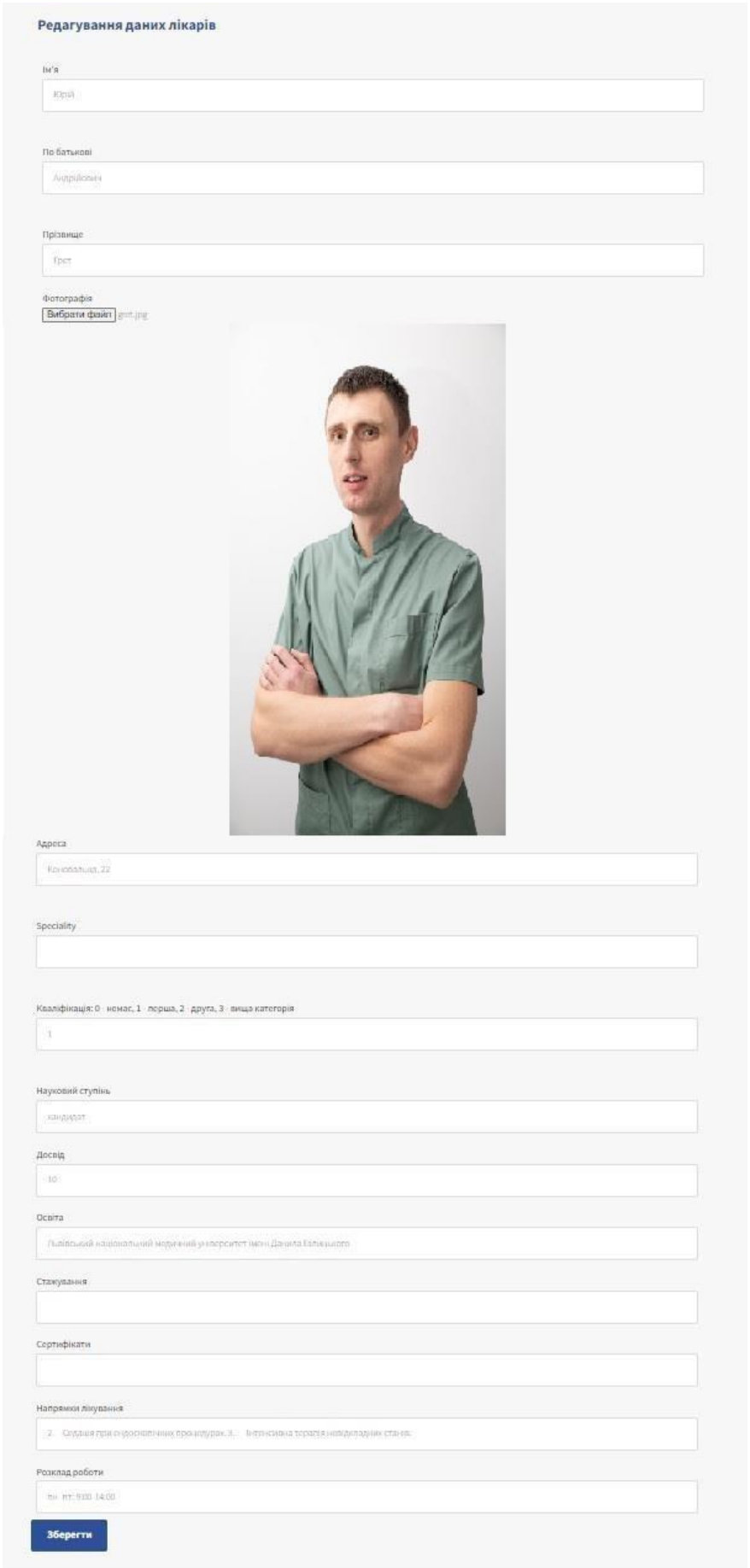


Рисунок 3.28 – Редагування інформації про лікаря

Адміністратор може створити звіт про всі записи, оформлені пацієнтами або лише про записи за поточний день. Також планується додати додатковий функціонал для розширення можливостей фільтрації даних.

Щоб клієнт(пацієнт) міг оформити замовлення, йому потрібно зареєструватися. Форма реєстрації для пацієнта показана на рисунку 3.29.

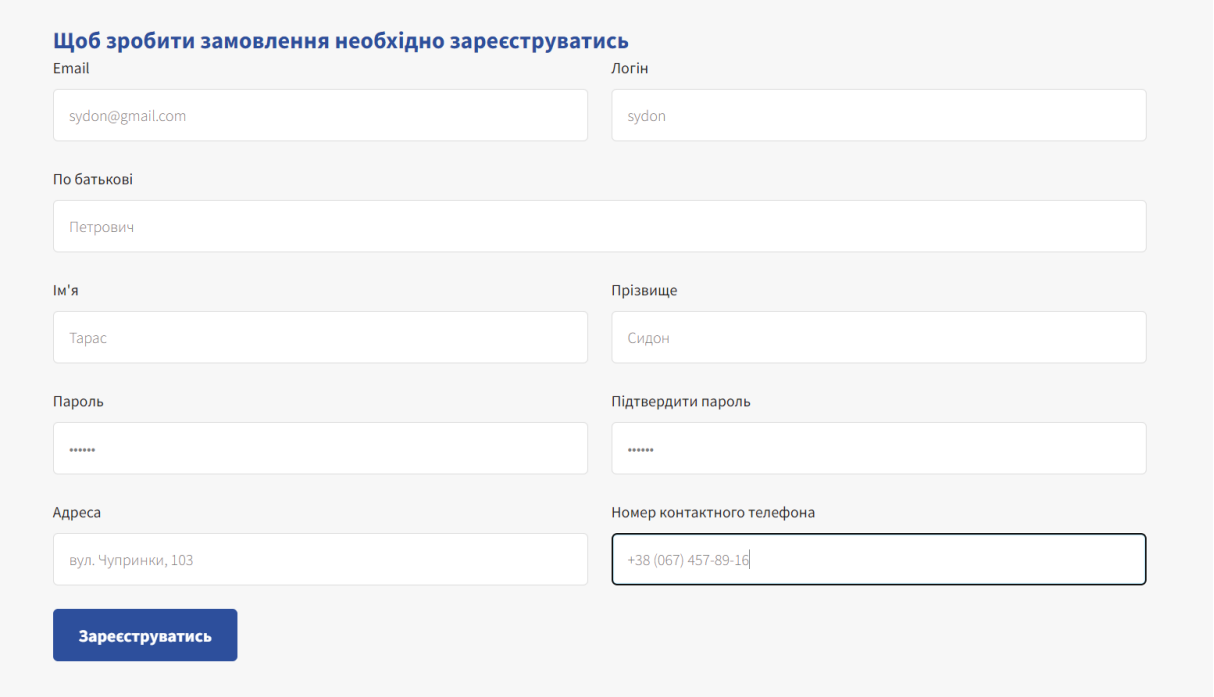


Рисунок 3.29 – Реєстрація пацієнта

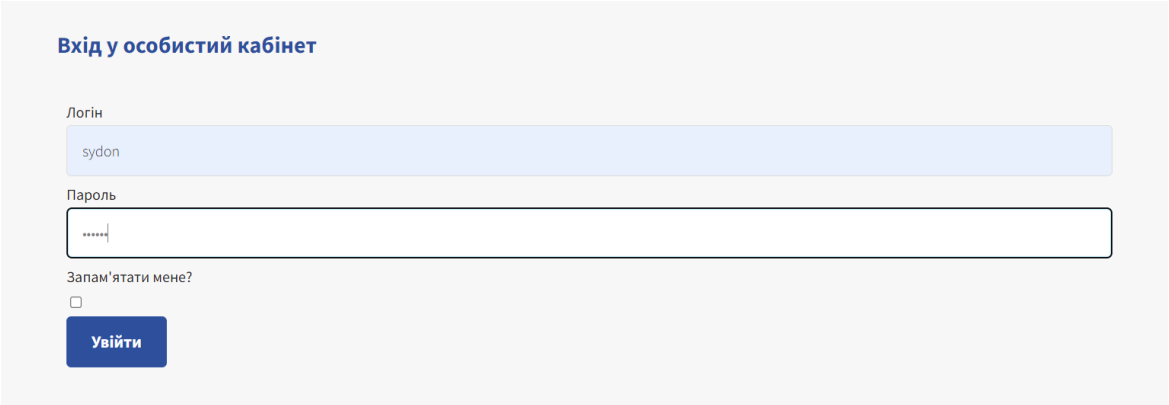


Рисунок 3.30 – Авторизація щойно зареєстрованого пацієнта

Після авторизації пацієнт здійснює запис до лікаря. Для здійснення даної дії необхідно обрати лікаря на сторінці запис. Процес обрання необхідного лікаря зображено на рисунку 3.31.

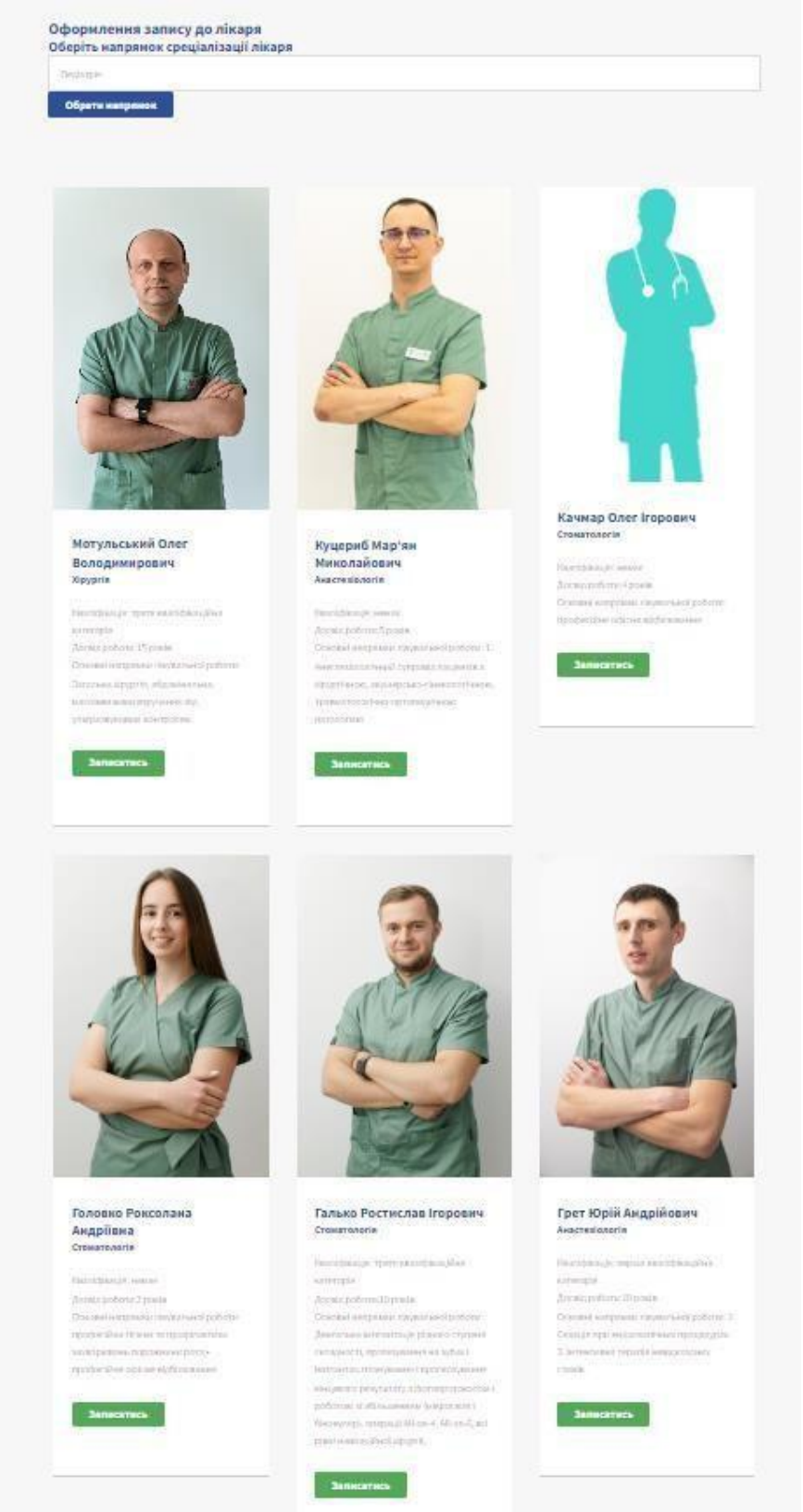


Рисунок 3.31 – Оформлення запису (частина 1)

Для зручності пошуку реалізовано фільтр за спеціальністю лікаря. Можна обрати лікаря тільки одного напряму (рис.3.32). Надалі опції для фільтру планується розширити.

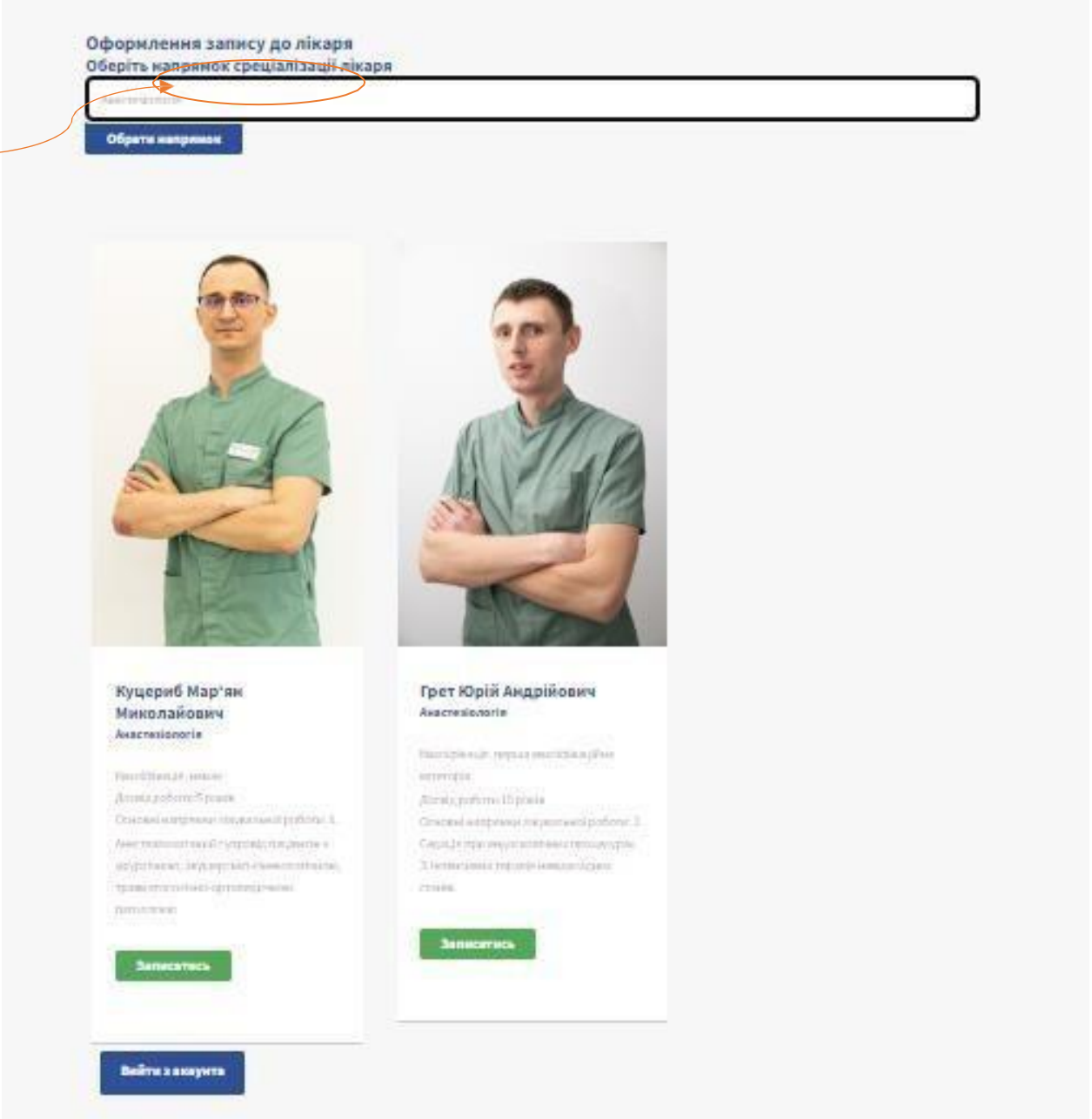


Рисунок 3.32 - Оформлення запису (частина 2)

Щоб продовжити запис необхідно натиснути на кнопку записатись і обрати послугу, час та дату запису (рис.3.33).

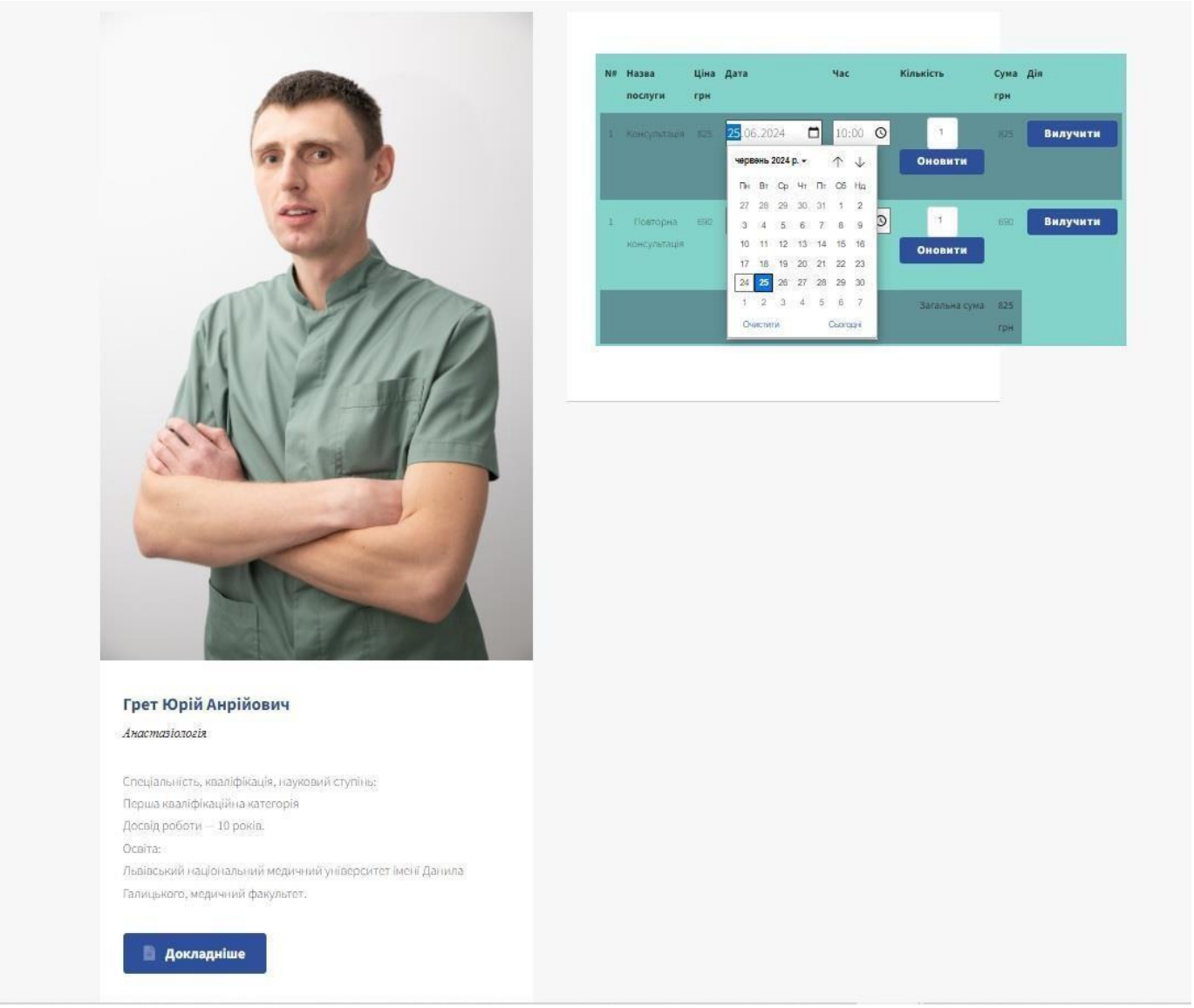


Рисунок 3.33 - Оформлення запису (частина 3)

Після вибору дати і часу консультації, а також ознайомлення з ціною даної послуги пацієнта переспрямовує на нову сторінку з повідомленням про підтвердження запису.

Наш вебсайт адаптовано для мобільних пристроїв та планшетів. Це означає, що він автоматично підлаштовується під різні розміри екранів, забезпечуючи зручний та інтуїтивний інтерфейс для користувачів незалежно від того, який пристрій вони використовують. Адаптивний дизайн сайту дозволяє легко здійснювати навігацію, переглядати контент та оформлювати замовлення як на планшетах, так і на смартфонах (рис.3.34 та рис.3.35).

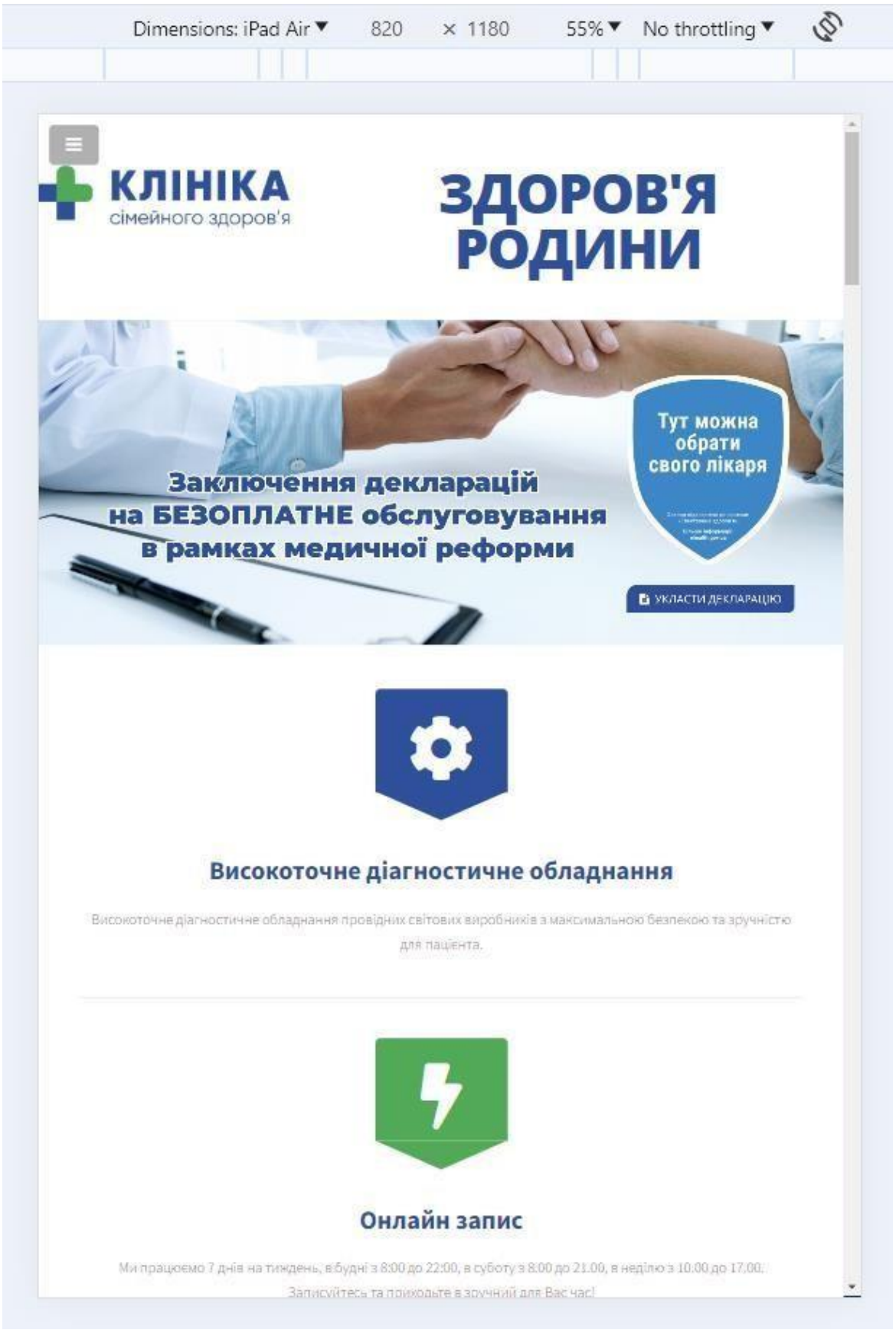


Рисунок 3.34 - Адаптація вебсистеми під планшети

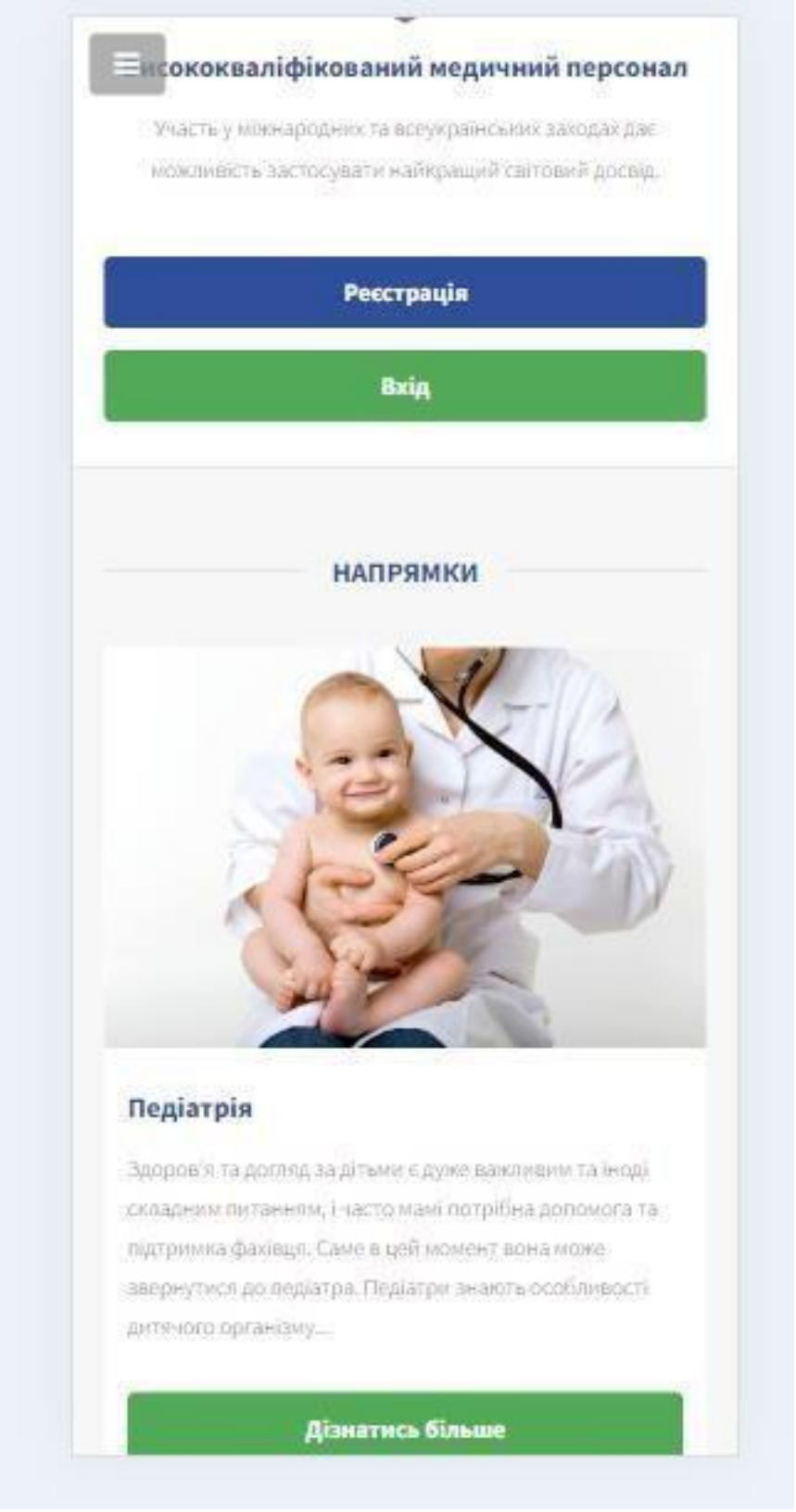


Рисунок 3.35 - Адаптація вебсистеми під смартфони

ВИСНОВКИ

У результаті виконання дипломної бакалаврської роботи була розроблена веборієнтована інформаційна система, що дозволяє пацієнтам записуватися до лікаря онлайн.

Розроблена система має на меті полегшити доступ до медичних послуг та залучити нових пацієнтів. Дана система дозволяє отримувати інформацію про лікарів та їх спеціалізації, а також дізнатися ціни на послуги та відгуки інших пацієнтів.

Завдяки розробленій системі реєстрації, користувачі мають можливість вибрати лікаря, переглянути інформацію про нього, його розклад, освіту, стаж роботи та забронювати зручний час для візиту. Особисті дані користувачів надійно захищені, оскільки для реєстрації використовувалась система Identity.

Платформа має розроблену панель для адміністрування.. Адміністратором системи може бути людина, яка не має технічної підготовки.

Зовнішній вигляд сайту автоматично адаптується під різні розміри екрану, забезпечуючи зручний інтерфейс як для користувачів настільних комп'ютерів, так і для мобільних пристроїв.

Крім того, на сайті розміщено статті, поради, рекомендації та іншу освітню інформацію з питань здоров'я. Пацієнти мають можливість ділитися своїм досвідом через залишення відгуків про роботу клініки на сайті.

Сучасний, функціональний та зручний у користуванні сайт сприяє формуванню позитивного іміджу клініки та оптимізації бізнес-процесів.

Загалом всі поставлені завдання виконано.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ian Griffiths, "Programming C# 9.0", O'Reilly Media, 2021, 800 с.
2. Adam Freeman, "Pro ASP.NET Core MVC 2", Apress, 2017, 1017 с.
3. Jon P Smith, "Entity Framework Core in Action",Manning Publications, 2018, 416 с.
4. Andrew Lock, "ASP.NET Core in Action", Manning Publications, 2018, 520 с.
5. David Flanagan, "JavaScript: The Definitive Guide", O'Reilly Media, 2020, 706 с.
6. Jon Skeet, "C# in Depth", Manning Publications, 2019, 528 с.
7. James Chambers, David Paquette, Simon Timms, "ASP.NET Core Application Development: Building an application in four sprints",Packt Publishing, 2020, 500 с.
8. Fanie Reynders, Kevin Dockx, "Modern API Design with ASP.NET Core 3: Building Cross-Platform Back-End Systems", Packt Publishing, 2020, 390 с.
9. Steven F. Lott, "Hands-On Object-Oriented Programming with C#: Build maintainable software with reusable code using C# 8 and .NET Core 3", Packt Publishing, 2020, 486 с.
10. Chris Sainty, "Blazor in Action", Manning Publications, 2020, 400 с.
11. Ricardo Peres, "Modern Web Development with ASP.NET Core 5: A deep dive into the latest .NET Core release with new chapters on Blazor, gRPC, and Identity", Packt Publishing, 2021, 438 с.
12. Steve Fenton, "Pro TypeScript: Application-Scale JavaScript Development", Apress, 2020 (3rd edition), 300 с.
13. Brian M. Korzynski, "Practical Entity Framework", Apress, 2021, 300 с.
14. Gaurav Kumar Aroraa, Lalit Kale, "Building Microservices with .NET Core: Develop skills in Reactive Microservices, database scaling, Azure Microservices, and more", Packt Publishing, 2021, 350 с.
15. Keith J. Grant, "CSS in Depth", Manning Publications, 2020, 350 с.

ДОДАТКИ

Додаток А. Лістинги коду для формування інтерфейсу клієнтської частини.

Майстер-сторінка _Layout.cshtml

```
<!DOCTYflE HTML>

<html>
	<head>

		@await Html.flartialAsync("Metatagsflartial.cshtml")
		@await Html.flartialAsync("Cssflartial.cshtml")

	</head>
	<body class="homepage is-preload">
		<div id="page-wrapper">

			<!-- Header -->
			@await Html.flartialAsync("Headerflartial.cshtml")

			<!-- Main -->
				<section id="main">
					<div class="container">
						<div class="row">
							<div class="col-12">

								<!-- MainContent -->
									@RenderBody()

							</div>
							<div class="col-12">

								<!-- Department -->
									@await
Html.flartialAsync("Departmentflartial.cshtml")

							</div>
							<div class="col-12">

								<!-- News -->
									@await
Html.flartialAsync("Newsflartial.cshtml")

							</div>
						</div>
					</div>
				</section>

			<!-- Footer -->
			@await Html.flartialAsync("Footerflartial.cshtml")

		</div>

		<!-- Scripts -->
			@await Html.flartialAsync("Scriptsflartial.cshtml")
	</body>
</html>
```

HeaderPartial.cshtml

```
@using Microsoft.AspNetCore.Identity
<section id="header">

                    <!-- Logo -->
                    <ul class="titleLogo">
                        <li><a href="#"><img src="~/images/Logo.png"
alt="Logo"></a></li>
                        <li><h1><a asp-area="" asp-controller="Home" asp-
action="Index">ЗДОРОВ"Я РОДИНИ </a></h1></li>
                    </ul>


                    <!-- Nav -->
                        <nav id="nav">
                            <ul>
                                <li class="current"><a asp-area=""
asp-controller="Home" asp-action="Index">Головна</a></li>
                                <li>
                                    <a href="#">Напрямки</a>
                                    <ul>
                                        <li><a
href="#">Педіатрія</a></li>
                                        <li><a href="#">Сімейна
медицина</a></li>
                                        <li><a
href="#">Гастроентерологія</a></li>
                                        <li>
                                            <a
href="#">Гінекологія</a>

                                        <li><a
href="#">Дерматовенерологія</a></li>
                                        <li><a
href="#">Дієтологія</a></li>
                                        <li><a
href="#">Ендокринологія</a></li>
                                        <li><a
href="#">Кардіологія</a></li>
                                        <li><a
href="#">Неврологія</a></li>
                                        <li><a
href="#">Офтальмологія</a></li>
                                        <li><a
href="#">Стоматологія</a></li>
                                        <li><a
href="#">Хірургія</a></li>

                                    </ul>
                                </li>
                                <li><a asp-area="" asp-
controller="Home" asp-action="Doctors">Лікарі</a></li>
                                <li><a asp-area="Order" asp-
controller="HomeOrder" asp-action="Index">Запис</a></li>
                                <li><a asp-area="" asp-
controller="Home" asp-action="flrices">Ціни</a></li>
                                <li><a asp-area="" asp-
controller="Home" asp-action="Contacts"> Контакти</a></li>
                                <li><a asp-area="" asp-
controller="Home" asp-action="Declaration">Декларація</a></li>
```

```
                                        <li><a asp-area="" asp-
controller="Account" asp-action="Register">Реєстрація</a></li>
                                         @if (User.Identity.IsAuthenticated)
                                        {
                                            <li><a class="button primary"
 asp-area="" asp-controller="Account" asp-action="Logout" ><span>Вийти</span></a></li>
                                        }
                                        else
                                        {
                                            <li><a class="button primary"
 asp-area="" asp-controller="Account" asp-action="Login" ><span>Вхід</span></a></li>
                                        }
                                    </ul>
                                </nav>

                            <!-- Banner -->
                                <section id="banner">
                                    <header>
                                        <h2>Howdy. This is Dopetrope.</h2>
                                        <p>A responsive template by HTML5
Ufl</p>
                                    </header>
                                </section>

                            <!-- Intro -->
                                <section id="intro" class="container">
                                    <div class="row">
                                        <div class="col-4 col-12-medium">
                                            <section class="first">
                                                <i class="icon solid
featured fa-cog"></i>
                                                <header>
                                                    <h2>Високоточне
діагностичне обладнання</h2>
                                                </header>
                                                <p>Високоточне
діагностичне обладнання провідних світових виробників з максимальною безпекою та
зручністю для пацієнта.</p>
                                            </section>
                                        </div>
                                        <div class="col-4 col-12-medium">
                                            <section class="middle">
                                                <i class="icon solid
featured alt fa-bolt"></i>
                                                <header>
                                                    <h2>Онлайн
запис</h2>
                                                </header>
                                                <p>Ми працюємо 7 днів на
тиждень, в будні з 8:00 до 22:00, в суботу з 8.00 до 21.00, в неділю з 10.00 до 17.00.
Запишуйтесь та приходьте в зручний для Вас час!</p>
                                            </section>
                                        </div>
                                        <div class="col-4 col-12-medium">
                                            <section class="last">
                                                <i class="icon solid
featured alt2 fa-star"></i>
                                                <header>

                                    <h2>Висококваліфікований медичний персонал</h2>
                                                </header>
```

```
                                                    <p>Участь у міжнародних
та всеукраїнських заходах дає можливість застосувати найкращий світовий досвід.</p>
                                                </section>
                                            </div>
                                        </div>
                                        <footer>
                                            <ul class="actions">
                                                <li><a href="#" class="button
large">Реєстрація</a></li>
                                                <li><a href="#" class="button
alt large">Вхід</a></li>
                                            </ul>
                                        </footer>
                                    </section>

                                </section>
```

FooterPartial.cshtml

```
<section id="footer">@using HealthClinic.Service
                        <div class="container">
                            <div class="row">
                                <div  class="col-8  col-12-medium">
                                    <section>
                                        <header>
                                            <h2>Індивідуальні
програми: акційні пропозиції </h2>
                                        </header>
                                        <ul class="dates">
                                            <li>
                                                <span
class="date">Черв<strong>10</strong></span>
                                                <h3><a
href="#">Здорове сердце</a></h3>
                                                <p>ІКлініка "Здоровя
родини" впроваджує пакет комплексного обстеження серця.</p>
                                            </li>
                                            <li>
                                                <span
class="date">Черв. <strong>06</strong></span>
                                                <h3><a
href="#">Старт у літо</a></h3>
                                                <p>Пакетна
пропозиція для отримання довідки в табір.</p>
                                            </li>
                                            <li>
                                                <span
class="date">Трав. <strong>31</strong></span>
                                                <h3><a
href="#">Здорова дитина - щаслива родина</a></h3>
                                                <p>Чекап для дітей
дошкільного та шкільного віку (розширений). Турбота про здоров’я дітей в період
канікул.</p>
                                            </li>
                                            <li>
                                                <span
class="date">Трав. <strong>15</strong></span>
                                                <h3><a href="#">Я
народився БАЗОВИЙ</a></h3>
                                                <p>Чекап
новонародженого від 0 до 12 місяців.</p>
                                            </li>

                                            <li>
```

```
                                                    <span
class="date">Трав.<strong>03</strong></span>
                                                    <h3><a
href="#">Чекап здорової людини</a></h3>
                                                    <p>Звичка проходити
регулярні медичні обстеження є ключовою для збереження здоров'я дорослих.</p>
                                                </li>
                                            </ul>
                                        </section>
                                    </div>
                                    <div  class="col-4  col-12-medium">
                                        <section>
                                            <header>
                                                <h2>Як до нас
добратись?</h2>
                                            </header>
                                            <a href="#" class="image
featured"><img src="~/images/pic16.jpg" alt="" /></a>
                                            <p>
                                                Клініка <strong>Здоров'я
 родини</strong> - це сучасна клініка європейського рівня,
                                                де працює команда
однодумців, висококваліфікованих фахівців та досвідчених експертів!
                                            </p>
                                            <footer>
                                                <ul class="actions">
                                                    <li><a  href="#"
class="button">Залишити  відгук</a></li>
                                                </ul>
                                            </footer>
                                        </section>
                                    </div>
                                    <div class="col-4 col-6-medium col-12-small">
                                        <section>
                                            <header>
                                                <h2>МЕДИЧНІ</h2>
                                            </header>
                                            <ul class="divided">
                                                <li><a
href="#">Педіатрія</a></li>
                                                <li><a  href="#">Сімейна
медицина</a></li>
                                                <li><a
href="#">Гастроентерологія</a></li>
                                                <li><a
href="#">Гінекологія</a></li>
                                                <li><a
href="#">Дерматовенерологія</a></li>
                                                <li><a
href="#">Дієтологія</a></li>
                                            </ul>
                                        </section>
                                    </div>
                                    <div class="col-4 col-6-medium col-12-small">
                                        <section>
                                            <header>
                                                <h2>НАПРЯМКИ</h2>
                                            </header>
                                            <ul class="divided">
                                                <li><a
href="#">Ендокринологія</a></li>
                                                <li><a
href="#">Кардіологія</a></li>
```

```
                                                    <li><a
href="#">Неврологія</a></li>
                                                    <li><a
href="#">Офтальмологія</a></li>
                                                    <li><a
href="#">Стоматологія</a></li>
                                                    <li><a
href="#">Хірургія</a></li>
                                                </ul>
                                            </section>
                                        </div>
                                        <div class="col-4 col-12-medium">
                                            <section>
                                                <header>
                                                    <h2>Вітаємо у соц.
мережах</h2>
                                                </header>
                                                <ul class="social">
                                                    <li><a class="icon brands
fa-facebook-f" href="#"><span class="label">Facebook</span></a></li>
                                                    <li><a class="icon brands
fa-twitter" href="#"><span class="label">Twitter</span></a></li>
                                                    <li><a class="icon brands
fa-dribbble" href="#"><span class="label">Dribbble</span></a></li>
                                                    <li><a class="icon brands
fa-tumblr" href="#"><span class="label">Tumblr</span></a></li>
                                                    <li><a class="icon brands
fa-linkedin-in" href="#"><span class="label">LinkedIn</span></a></li>
                                                </ul>
                                                <ul class="contact">
                                                    <li>
                                                        <h3>Адреса</h3>
                                                        <p>
                                                            Клініка
"Здоров'я родини"<br />
                                                            вул.
О.Кониського, 9/2а<br />
                                                            двів,
Львівська область, 79000
                                                        </p>
                                                    </li>
                                                    <li>
                                                        <h3>Email</h3>
                                                        <p><a
href="#">@Config.CompanyEmail</a></p>
                                                    </li>
                                                    <li>
                                                        <h3>Телефон</h3>

                <p>@Config.Companyflhone</p>
                                                    </li>
                                                </ul>
                                            </section>
                                        </div>
                                        <div class="col-12">

                                            <!-- Copyright -->
                                                <div id="copyright">
                                                    <ul class="links">
                                                        <li>ficopy; 2024. All
rights reserved.</li><li>Розробник: Сидон Тарас</li>
                                                    </ul>
                                                </div>
```

Додаток Б. Лістинги коду розроблених контролерів

AccountController.cs

```
using HealthClinic.Domain;
using HealthClinic.Domain.Entities;
using HealthClinic.Models;
using Microsoft.AspNetCore.Authorization;
using  Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace HealthClinic.Controllers
{

    [Authorize]
    public class AccountController : Controller
    {
        private readonly UserManager<flatient> userManager;
        private readonly SignInManager<flatient> signInManager;
        private readonly DataFromDataBase dataFromDataBase;
        private readonly RoleManager<IdentityRole> roleManager;
        public AccountController(UserManager<flatient> userMgr, SignInManager<flatient>
signinMgr, DataFromDataBase dataFromDataBase, RoleManager<IdentityRole> roleManager)
        {
            userManager = userMgr;
            signInManager = signinMgr;
            this.dataFromDataBase = dataFromDataBase;
            this.roleManager = roleManager;
        }

        [HttpGet]
        [AllowAnonymous]
        public IActionResult Register()
        {
            return View();
        }
        [Httpflost]
        [AllowAnonymous]
        public async Task<IActionResult> Register(RegisterModel model)
        {

            if (ModelState.IsValid)
            {
                flatient flatientAdd = new flatient
                {
                    //Id = new System.Guid().ToString(),
                    Email = model.Email,
                    UserName = model.UserName,
                    flhoneNumber = model.flhoneNumber,
                    Address = model.Address,
                    FirstName = model.FirstName,
                    LastName = model.LastName,
                    SecondName = model.SecondName
                };


                // додаємо користувача
                var result = await userManager.CreateAsync(flatientAdd, model.flassword);
                if (result.Succeeded)
                {
                    var resultAddRole = userManager.AddToRoleAsync(flatientAdd, "flatient");
                    if (resultAddRole.Result.Succeeded)
```

```
                    { // встановлюємо кукі
                        await signInManager.SignInAsync(flatientAdd, false);
                        //companyAdd.UserId = user.Id;
                        dataFromDataBase.flatients.Saveflatients(flatientAdd);
                        return RedirectToAction("Index", "Home");
                    }
                    else
                    {
                        ModelState.AddModelError(string.Empty, "НЕ встановлено роль для користувача");
                    }


                }
                else
                {
                    foreach (var error in result.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                }
            }
            return View(model);
        }

        [HttpGet]
        [AllowAnonymous]
        public IActionResult Login(string returnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View(new LoginModel { ReturnUrl = returnUrl });
        }
        [Httpflost]
        [AllowAnonymous]
        public async Task<IActionResult> Login(LoginModel model)
        {

            if (ModelState.IsValid)
            {
                IdentityUser user = await userManager.FindByNameAsync(model.UserName);
                if (user != null)
                {
                    await signInManager.SignOutAsync();
                    Microsoft.AspNetCore.Identity.SignInResult result = await signInManager.flasswordSignInAsync((flatient)user, model.flassword, model.RememberMe, false);
                    if (result.Succeeded)
                    {
                        return Redirect(model.ReturnUrl оо "/");
                    }
                }
                ModelState.AddModelError(nameof(LoginModel.UserName), "Невірний логін або пароль");
            }
            return View(model);
        }

        [Authorize]
        public async Task<IActionResult> Logout()
        {
            await signInManager.SignOutAsync();
            return RedirectToAction("Index", "Home");
        }
    }
```

```
}
```

HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using HealthClinic.Domain;

namespace HealthClinic.Controllers
{
    public class HomeController : Controller
    {
        private readonly DataFromDataBase dataFromDataBase;

        public HomeController(DataFromDataBase dataFromDataBase)
        {
            this.dataFromDataBase = dataFromDataBase;
        }
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult Declaration()
        {
            return View(dataFromDataBase.flages.GetflageByflageName("Declaration"));
        }
        public IActionResult Contacts()
        {
            return View(dataFromDataBase.flages.GetflageByflageName("Contacts"));
        }
        public IActionResult flrices()
        {
            return View(dataFromDataBase.flages.GetflageByflageName("flrices"));
        }
        public IActionResult Doctors()
        {
            return View(dataFromDataBase.flages.GetflageByflageName("Doctors"));
        }
    }
}
```

ServicesController.cs

```
using Microsoft.AspNetCore.Mvc;
using SkiVacation.Domain;

namespace SkiVacation.Controllers
{
    public class HomeController : Controller
    {
        private readonly DataManager dataManager;

        public HomeController(DataManager dataManager)
        {
            this.dataManager = dataManager;
        }

        public IActionResult Index()
        {
            return View(dataManager.TextFields.GetTextFieldByCodeWord("flageIndex"));
        }
        //public IActionResult flrivacy()
        //{
        //    return View();
        //}
        public IActionResult Contacts()
```

```
        {
            return View(dataManager.TextFields.GetTextFieldByCodeWord("flageContacts"));
        }
    }
}
```

HomeOrderController.cs

```
using HealthClinic.Domain;
using Microsoft.AspNetCore.Mvc;

namespace HealthClinic.Areas.Order.Controllers
{
    [Area("Order")]
    public class HomeOrderController : Controller
    {
        private readonly DataFromDataBase dataFromDataBase;


        public HomeOrderController(DataFromDataBase dataFromDataBase)
        {
            this.dataFromDataBase = dataFromDataBase;

        }

        public IActionResult Index(into speciality)
        {
            ViewBag.Specialitys= dataFromDataBase.Specialitys.GetSpecialitys();
            if (speciality == null)
            {
                return View(dataFromDataBase.Doctors.GetDoctors());
            }
            else
            {
                var doctors = dataFromDataBase.Doctors.GetDoctors().Where(p =>
p.SpecialityId == speciality);
                return View(doctors);
            }
        }
    }
}
```

AppointmentController.cs

```
using HealthClinic.Domain;
using HealthClinic.Domain.Entities;
using HealthClinic.Service;
using Microsoft.AspNetCore.Mvc;

namespace HealthClinic.Areas.Order.Controllers
{
    [Area("Order")]
    public class AppointmentController : Controller
    {
        private readonly DataFromDataBase dataFromDataBase;

        public AppointmentController(DataFromDataBase dataFromDataBase)
        {
            this.dataFromDataBase = dataFromDataBase;

        }
        public IActionResult Index()
        {
            var appoitment =
SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
            ViewBag.appoitment = appoitment;
             double v = (double)appoitment.Sum(i => i.MedService.flriceEnd * i.Quantity);
```

```
            //double v = 500;
            ViewBag.total = v;

            return View();
        }
        public IActionResult Buy(string id)
        {
            //var ServiceItem = new ServiceItem();
            //Guid id = new Guid();
            var appoitment =
SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
            if (appoitment == null)
            {
                appoitment = new List<Item>();
                appoitment.Add(new Item
                {
                    MedService = dataFromDataBase.MedServices.GetMedServiceByDoctorId(id),
                    Quantity = 1
                });
                SessionHelperJson.SetObjectAsJson(HttpContext.Session,      "appoitment",
appoitment);
            }
            else
            {
                appoitment =
SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
                string index = Exists(id);
                if (index == (-1).ToString())
                {
                    appoitment.Add(new Item
                    {
                        MedService =
 dataFromDataBase.MedServices.GetMedServiceById(int.flarse(id)),
                        Quantity = 1
                    });
                }
                else
                {
                    appoitment[int.flarse(index)].Quantity++;
                }
                SessionHelperJson.SetObjectAsJson(HttpContext.Session,      "appoitment",
appoitment);
            }
            return RedirectToAction("Index");
        }

        public IActionResult Remove(int id)
        {
            var                                appoitment                                =
SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
            string index = Exists(id.ToString());
            appoitment.RemoveAt(int.flarse(index));
            SessionHelperJson.SetObjectAsJson(HttpContext.Session, "appoitment",
appoitment);
            return RedirectToAction("Index");
        }
        private string Exists(string id)
        {
            var                                appoitment                                =
SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
            for (var i = 0; i < appoitment.Count; i++)
            {
                if (appoitment[i].MedService.Id.ToString() == id)
                {
```

```
                    return i.ToString();
                }
            }
            return (-1).ToString();
        }

        public IActionResult Update(string id, int[] quantities)
        {
            var appoitment = SessionHelperJson.GetObjectFromJson<List<Item>>(HttpContext.Session, "appoitment");
            string index = Exists(id.ToString());
            appoitment[int.flarse(index)].Quantity = quantities[0];
            //for (var i = 0; i < appoitment.Count; i++)
            //{
            //    appoitment[i].Quantity = quantities[i];
            //}
            SessionHelperJson.SetObjectAsJson(HttpContext.Session, "appoitment", appoitment);
            return RedirectToAction("Index"); ;
        }

    }
}
```