МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: **«РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ ЦЕНТРОМ КІНЕЗІТЕРАПІЇ “ОСЕРЕДОК ЗДОРОВ’Я”»**

Виконав: студент 4 курсу, групи КН-4
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп’ютерні науки
освітньо-професійної програми «Комп’ютерні науки»
Сачук Назар Ігорович

Керівник: *старший викладач кафедри ЕММІТ*
Клебан Юрій Вікторович

Рецензент: *Front-end Developer “DOODLE”, LLC*
Місай Володимир Віталійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від « 18 » травня 2023 р.

Острог, 2023

Міністерство освіти і науки України

Національний університет «Острозька академія»

Факультет: *економічний*
Кафедра: *економіко-математичного моделювання та інформаційних технологій*
Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

З А В Д А Н Н Я
на кваліфікаційну роботу/проєкт студента

Сачука Назара Ігоровича

1. Тема роботи: Розробка системи управління реабілітаційним центром кінезітерапії «Осередок Здоров'я»

керівник проєкту Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ.

Затверджено наказом ректора НаУОА від 31 жовтня 2022 року №77.

2. Термін здачі студентом закінченої роботи/проєкту: 31 травня 2023 року.

3. Вихідні дані до роботи/проєкту: дана робота полягає у розробці системи управління реабілітаційним центром кінезітерапії «Осередок Здоров'я» у розробці серверної частини якої було використано ASP.NET фреймворк, архітектурою було обрано Clean Architecture, Entity Framework Core для побудови та взаємодії з базою даних SQL, бібліотека MediatR, яка реалізує CQRS патерн, бібліотека ErrorOr була використана для обробки помилок та бібліотека Swagger для тестування HTTP запитів до серверу. Для розробки клієнтської частини було використано бібліотеку React та набір готових компонентів MUI, для взаємодії з серверною частиною було використано бібліотеку Axios.

4. Перелік завдань, які належить виконати: розробити серверну частину системи управління реабілітаційним центром кінезітерапії «Осередок Здоров'я», що включає в себе побудову архітектури за принципами Clean Architecture, спроектувати та побудувати базу даних для обробки та зберігання інформації, розробити всі необхідні ендпоїнти для забезпечення обробки та видачі інформації для клієнтської

частини, розробити клієнтську частину, завдяки якій всі види користувачів, такі як клієнт, адміністратор та тренер зможуть взаємодіяти між собою за визначеними ролями, тобто різними функціоналом доступним для кожного із них.

5. Перелік графічного матеріалу: рисунки, таблиці, лістинги.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В.	01.12.2022	01.12.2022
2	Клебан Ю. В.	01.12.2022	01.12.2022
3	Клебан Ю. В.	01.12.2022	01.12.2022

7. Дата видачі завдання: 01.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1.	Затвердження теми роботи/проєкту	до 31.10.22.	
2.	Постановка технічного завдання	до 01.12.22	
3.	Підбір та побудова архітектури проекту	до 25.02.23	
4.	Проектування та розробка бази даних	до 14.03.23	
5.	Розробка серверної частини	до 01.04.23	
6.	Тестування серверної частини	до 08.04.23	
7.	Підбір технологій для розробки клієнтської частини	до 15.04.23	
8.	Розробка клієнтської частини	до 13.05.23	
9.	Попередній захист кваліфікаційної роботи/проєкту	до 18.05.2023	
10.	Здача кваліфікаційної роботи/проєкту на кафедру	до 31.05.2023	

Студент: _____ Назар САЧУК

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *Розробка системи управління реабілітаційним центром кінезітерапії "Осередок Здоров'я"*

Автор: *Сачук Назар Ігорович*

Науковий керівник: *Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ.*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *66 с., 31 рис., 1 табл., 20 джерел..*

Ключові слова: *система управління, архітектура, серверна частина, клієнтська частина, API, база даних, сервіс, репозиторій, бібліотека*

Короткий зміст праці:

Завданням кваліфікаційної роботи/проєкту, було розробка системи управління реабілітаційним центром кінезітерапії "Осередок Здоров'я". Дана робота є описом реалізації серверної та клієнтської частини даного проекту. Серверна частина була реалізована на .NET CORE 6, де для побудови надійної архітектури було використано Clean Architecture а також CQRS патерн, SQL база даних була побудована за допомогою методу Code First, використовуючи бібліотеку Entity Framework. Клієнтська частина побудована на бібліотеці React та MUI, використовуючи мову JavaScript. Для взаємодії з серверною частиною було використано бібліотеку Axios. Результатом роботи є розробка ефективної та безпечної системи управління.

The task of the qualification work/project was the development of a management system for the rehabilitation center of kinesiotherapy. This work is a description of the implementation of the server and client parts of this project. The server-side was implemented using .NET CORE 6, where Clean Architecture and CQRS pattern were utilized to build a reliable architecture. The SQL database was constructed using the Code First approach with the help of the Entity Framework library. The client-side was built using the React library and MUI, utilizing JavaScript. The Axios library was used for interaction with the server-side. The result of the work is the development of an efficient and secure management system.

ЗМІСТ

ВСТУП

Невід’ємною частиною сучасного світу є вебсервіси, вони є результатом інноваційних рішень для надання відповідних послуг для будь-яких видів діяльності. Це означає, що підприємства мають прагнути до створення власних вебсервісів, розуміючи як ці технології можуть допомогти покращити їх діяльність, а також домогтись успіху на ринку.

Сьогодення успішно пристосувало нас до культури користування вебдодатками різноманітних сервісів, починаючи з виклику таксі, де можна виставити своє місцеперебування з точністю до метра та закінчуючи можливістю підписувати, представляти і навіть оформлювати документи в електронному форматі без жодних сумнівів у їх безпечності. Вважаю, дана тенденція є абсолютно позитивною, оскільки дарує комфорт у виконанні різноманітних, як побутових, так і ділових задач та економить найважливіше для будь-якої людини, а саме — час.

Певний період тому, внаслідок проблем зі спиною, я почав відвідувати реабілітаційний центр кінезотерапії "Осередок здоров'я". Під час моїх занять я помітив, що в управлінні центром використовується блокнот, де заноситься майже уся інформація. Це викликало моє зацікавлення, і я почав задумуватись про те, як зробити цю систему доступною через вебсервіс.

У мене з'явилась ідея створення вебсервісу для реабілітаційного центру, який би дозволяв управляти всіма процесами центру онлайн. Впевнений, що такий вебсервіс спростив би роботу персоналу центру та забезпечив зручність і комфорт клієнтам.

Для реалізації цього вебсервісу потрібно було б детально проаналізувати потреби реабілітаційного центру і розробити відповідну систему, що враховувала б усі їхні вимоги та особливості. Створення вебсервісу для реабілітаційного центру мало б великий потенціал для поліпшення його роботи. Вебсервіси стають все більш популярними в сучасному світі, і вони дійсно мають великий потенціал для поліпшення різних галузей діяльності. Важливо лише грамотно використовувати їхні

можливості та добре продумати процес їхньої реалізації, щоб отримати максимальну користь для підприємства і його клієнтів.

Отже, реабілітаційному центру кінезітерапії “Осередок здоров’я” потрібно розробити систему управління, адже без автоматизації певних процесів підприємство жертвує досить важливими складовими досвіду користувача, а саме: простотою використання — можливістю клієнта у два кліки узгодити дату та час заняття з реабілітологом/тренером або ж практичністю — адміністратору отримати кількість проведених занять конкретним тренером за певний період для нарахування заробітної плати, тощо. Сам користувач у даному програмному продукті уособлюватиме не тільки адміністратора та тренера, а і самого клієнта. Звідси слідує, що функціонал повинен бути максимально інтуїтивно зрозумілим, проте достатньо деталізованим для специфічних операцій.

Метою дослідження є розробка та прикладна реалізація системи управління, досить схожої до спортивного залу, проте зі своєю унікальною специфікою. Крім цього, будь-яке дослідження полягає у покращенні попередніх та надбанні нових навичок, тому не менш важливим є зробити свої знання більш ґрунтовними та широкими у сфері розробки вебдодатків.

Для досягнення мети дослідження потрібно виконати наступний ряд завдань:

- Виконати аналіз конкурентних програмних рішень.
- Ознайомитись з теоретичним матеріалом.
- Спроєктувати серверну частину.
  - Проєктування БД.
  - Вибір архітектури.
  - Вибір технологій.
- Розробити серверну частину
  - Побудувати архітектуру.
  - Побудувати БД.
  - Створити репозиторії доступу до БД.
  - Створити Контролери.

        - Створити авторизацію та автентифікацію.
    - Розробити клієнтську частину
        - Розробити авторизацію та автентифікацію.
        - Розробити інтерфейс для зручного використання сервісу.

Об’єктом дослідження є створення зручного інтерфейсу для взаємодії між користувачами системи управління. Предмет — стек технологій для проєктування та розробки програмного продукту.

Дана робота описуватиме процес проєктування та розробки системи управління реабілітаційним центром кінезітерапії “Осередок здоров’я”. Вибір на користь цієї теми був зроблений завдяки бажанню зробити зручнішою взаємодію між мною - клієнтом та тренером або адміністратором, та оптимізувати роботу самого підприємства.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ БІЗНЕС ПРОЦЕСІВ

1.1. Постановка проблеми, опис базових понять бізнес-процесу

Безумовно, одним із найважливіших ресурсів є час. А чи задумувались ми скільки ми його втрачаємо на прослуховування пустих гудків, без гарантії, що слухавку таки піднімуть? Навіть якщо вам пощастило та ви не очікували довго, не забувайте, що людський фактор ніколи не спить, тобто інформація котру ви передаєте або приймаєте не сприймається якою вона мала б бути спочатку. Особисто мені набагато зручніше у два “кліки” записатись на вільну годину, перелік яких буде відображено, а тренеру, відповідно, прийняти або ж відхилити пропозицію.

Розв'язанням даної проблеми я вбачаю розробку системи управління, котра матиме наступний функціонал:

- Запис клієнта на тренування.
- Перегляд розкладу тренувань для різних користувачів.
- Відстежування оплати клієнтів.
- Розрахунок заробітної плати тренерам.

Для більш чіткого розуміння було створено use-case діаграми, на котрих зображений функціонал для кожного з користувачів. До прикладу можливі дії для клієнта ми можемо побачити на (Рис. 1.1).

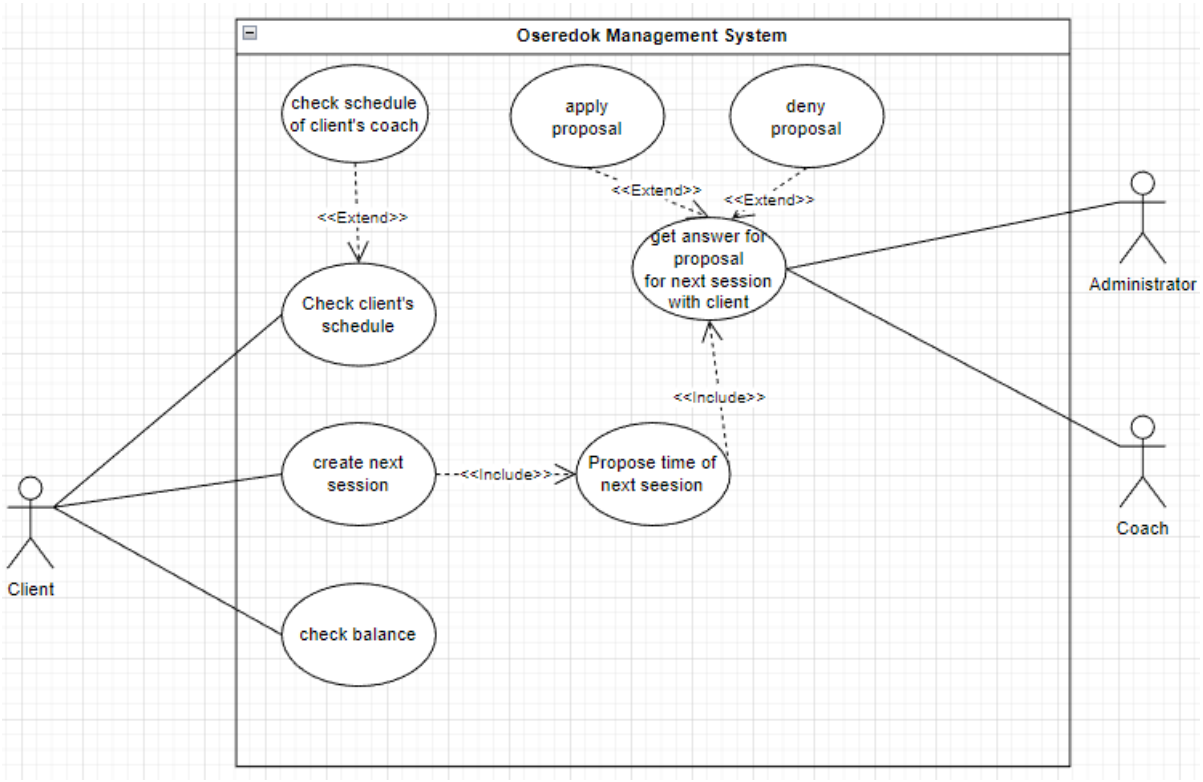

Рис. 1.1.Use Case діаграма можливих дій клієнта системи управління «Oseredok Management System».

Джерело: [Створено автором].

На даній діаграмі(див. Рис. 1.1) зображено кроки, які повинен зробити клієнт для створення пропозиції наступної сесії. Після чого адміністратор або ж тренер повинен підтвердити або ж відмовити запропонований час. Також до його можливостей належать: перегляд розкладу свого тренера для створення доцільних пропозицій наступних занять або ж перегляд свого балансу, у якому буде відображено кількість грошей на особовому рахунку та відповідно кількість можливих занять або ж боргів.

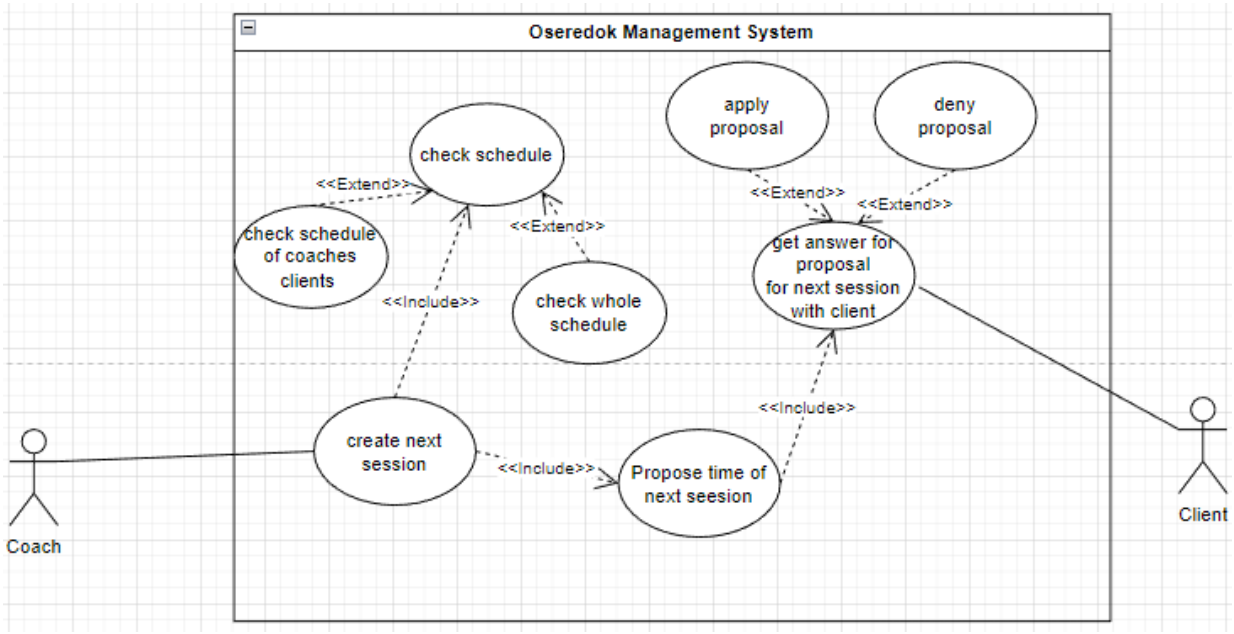

Рис. 1.2.Use Case діаграма можливих дій тренера системи управління «Oseredok Management System».

Джерело: [Створено автором].

Ця діаграма (див. Рис.1.2) зображає можливі дії тренера. а саме: перегляд розкладу з різними фільтрами, створення пропозиції наступного заняття, котру повинен погодити або ж відмовити клієнт. Планується також надати йому можливість переглядати кількість своїх заняття зі статусом “виконано”, для кращої оцінки своєї праці.

На наступній діаграмі (див. Рис.1.3) описано можливості дій адміністратора, вони є досить схожими до можливостей тренера проте додається можливість переглядати баланс клієнта та вираховувати заробітну плату тренеру. Надалі цей перелік буде збільшуватись, адже адміністрування включає в себе досить багато процесів.

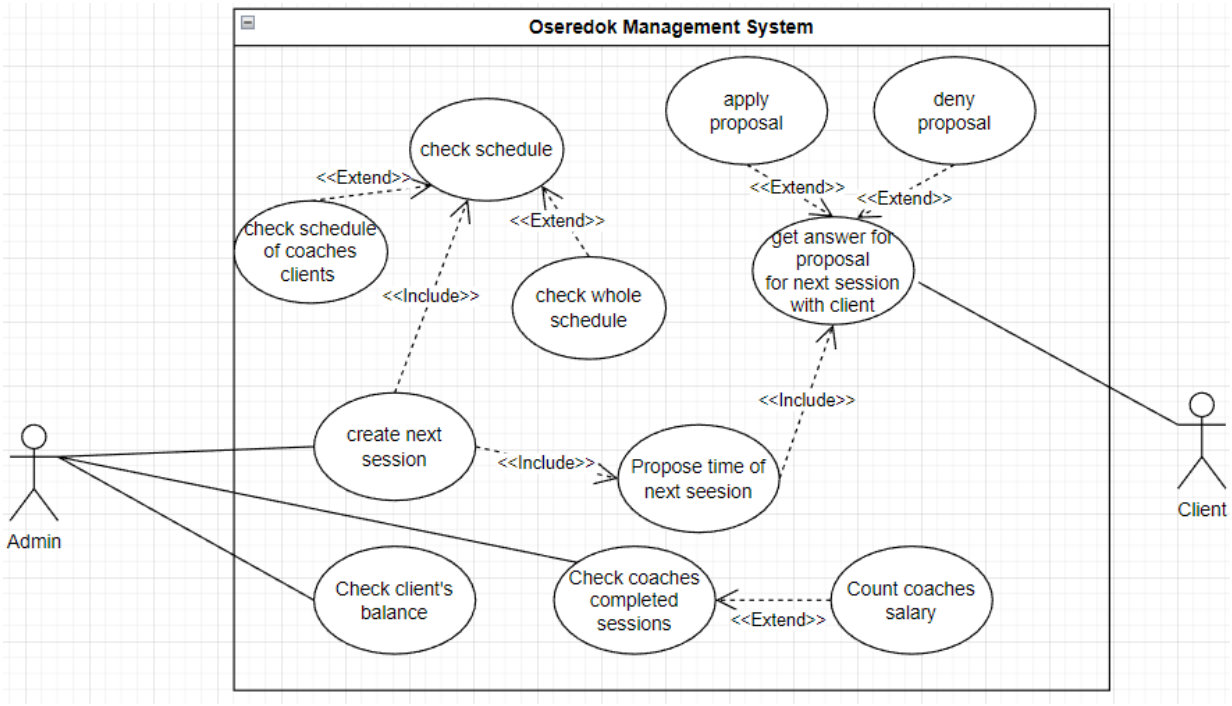

Рис. 1.2.Use Case діаграма можливих дій адміністратора системи управління «Oseredok Management System».

Джерело: [Створено автором].

Опис наведених вище процесів дозволить зробити управління більш якісним та забезпечить комфортні умови праці як для тренерів так і для адміністратора, а для самого розробника це є невід'ємною складовою формування вимог до програмного продукту.

1.2. Аналіз конкурентів на ринку ПЗ

Вважаю важливим для розуміння, що ринок є досить насиченим програмними рішеннями даної проблематики. Проте мені не вдалось знайти програмних рішень для управління невеликими спортивними залами, котрим абсолютно не вигідно мати масивну та відповідно дорогу систему управління. Сам аналіз дав мені зрозуміти, які основні вимоги до програмних продуктів даного типу і дозволив виділити функціонал, котрий було заплановано реалізувати.

1. jSolutions - це хмарна система для автоматизації управлінських та облікових завдань підприємств. jSolutions дозволяє не лише повністю

автоматизувати бізнес-процеси, а й мінімізувати витрати на користування системою. Дане рішення не є вузькоспеціалізованим на роботу саме з спортивними залами, проте загальний функціонал дозволить його пристосувати до конкретних задач.

Сайт конкурента “jSolutions”:[https://jsolutions.ua/ua/preimuschestva-oblachnoy-sistemy-jsolutions]

. 2. appointer. Дана система забезпечує управління спортивним центром без зайвих зусиль, структурований облік, детальна статистика і аналітика, ефективні ділові рішення — основні завдання, які вирішує сучасна CRM система для фітнес-клубу від компанії Appointer.

Сайт конкурента “appointer”:[https://appointer.ua/programa-dlya-fitnes-centriv/#abilities]

3. Fitness Pro - ефективний помічник для оптимізації бізнес-процесів та забезпечує широкий функціонал від взаємодії з клієнтами до управління робочими процесами співробітників. До найважливіших переваг належать зручний інтерфейс взаємодії та відповідно легке вивчення його персоналом

Сайт конкурента “Fitness Pro”:[https://u-s-c.com.ua/ai-helps/fitness-pro/]

Проаналізувавши попередніх конкурентів я зміг провести SWOT-аналіз свого програмного рішення(див. Таблиця 1.1)

Таблиця 1.1

SWOT-аналіз програмного продукту ”Oseredok Management System”

Сильні сторони	Слабкі сторони
Простий інтерфейс Достатній функціонал	Програмний продукт зроблений для конкретного підприємства та не може бути використаним для будь-якого іншого

	Конкуренти мають ширший функціонал
Можливості	Загрози
Кросплатформеність Розширення функціоналу	Поява більш зручного та функціонального конкурента

Джерело: створено автором

Поспілкувавшись з своїм тренером та адміністрацією, ми дійшли висновку, що усі перелічені вище програмні рішення мають надто широкий функціонал, котрий є зайвим для невеликого спортзалу, додатково оснащеного для занять кінезітерапією. А якщо буде з’являтись потреба у розширенні функціоналу, архітектура рішення дозволяє його розширювати.

РОЗДІЛ 2. ПРИКЛАДНА РОЗРОБКА ВЕБДОДАТКА

2.1. Опис архітектурного рішення та технологічного стека серверної частини

Вибір архітектури перед побудовою веб-додатку є надзвичайно важливим етапом, який має значний вплив на успіх проекту. Архітектура веб-додатку визначає структуру, організацію та взаємозв'язок між різними компонентами системи.

Ось декілька ключових причин, чому вибір архітектури є таким важливим аспектом:

- Масштабованість: Правильна архітектура дозволяє легко масштабувати веб-додаток під високе навантаження або збільшення функціональності в майбутньому. Гнучкість та легкість розширення є критичними аспектами для забезпечення ефективної роботи додатку під зростаючим навантаженням.
- Продуктивність: Правильно побудована архітектура дозволяє забезпечити високу продуктивність веб-додатку. Це означає швидкий відгук системи на запити користувачів, ефективне використання ресурсів сервера та оптимальну обробку даних.
- Розширюваність: Вибір правильної архітектури дозволяє легко додавати нові функціональні можливості до веб-додатку. Це дозволяє зберігати гнучкість та конкурентоспроможність продукту, а також швидко реагувати на зміни вимог користувачів та ринку.
- Підтримка: Правильно обрана архітектура забезпечує легку підтримку та супровід веб-додатку. Команда розробників повинна мати зрозуміле розподілення компонентів та логіку системи, щоб легко здійснювати виправлення помилок, вдосконалення та оптимізацію.
- Безпека: Архітектура веб-додатку впливає на безпеку системи. Правильно побудована архітектура може допомогти уникнути багатьох загроз безпеці, таких як атаки злому, втрати даних та інші види кіберзлочинності.
- Швидкість розробки: Правильна архітектура може прискорити процес розробки веб-додатку. Якщо вибрана архітектура забезпечує готові рішення

для типових задач, розробники можуть сконцентруватись на реалізації унікальної функціональності, що прискорює весь процес розробки.

Узагалі, правильно обрана архітектура допомагає забезпечити надійність, ефективність та гнучкість веб-додатку. Це робить його конкурентоспроможним на ринку та забезпечує задоволення вимог користувачів.

2.1.1. Опис архітектурного рішення серверної частини

Вибір архітектури програмного продукту є одним з ключових питань при розробці. Оскільки основними функціями, котрі вона повинна задовольняти є структурованість та розширюваність, моїм вибором стала “Чиста архітектура”[2], котру часто називають “Цибулевою” завдяки її головному принципу, а саме розділення компонентів на шари з напрямком до центру - ядра. Даний принцип називають інверсією залежностей, він забезпечує направленість усіх шарів всередину, що дозволяє легко розширювати рішення без втручань до нижчих рівнів (див Рис. 2.1).

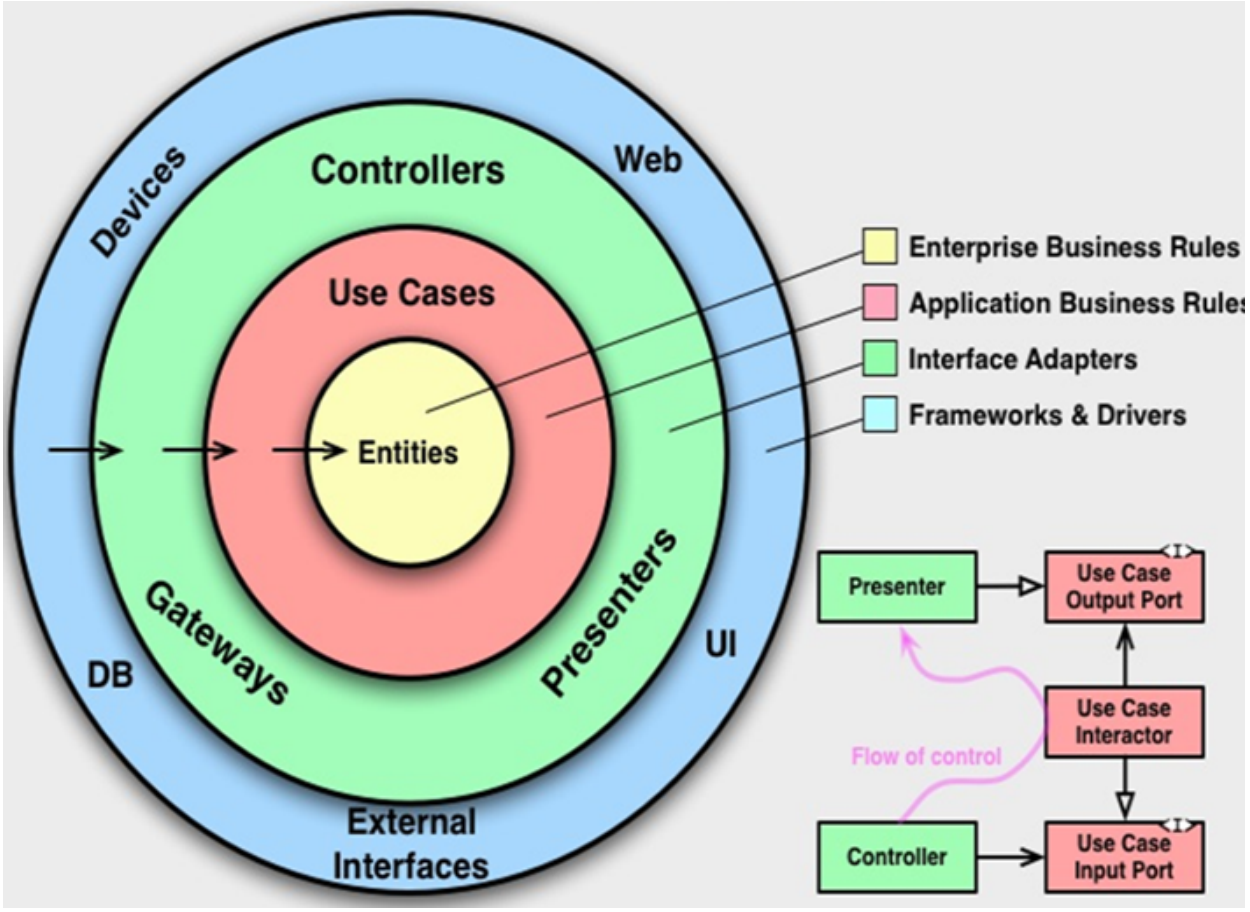

Рис. 2.1 Діаграма “Чистої архітектури”.

джерело:[2]

Дана архітектура дозволяє розробляти рішення більш модульним, це дозволяє прискорити розробку та забезпечити більш високий рівень стабільності роботи, а також допомагає уникнути небажаних помилок у проєктування

Чиста архітектура має наступний ряд характеристик:

- В центрі даного виду архітектури завжди є моделі, котрі використовуються як сутності об’єктів, котрі зберігаються в БД. За допомогою цих моделей проводиться моделювання БД та налаштовується багато аспектів для правильного зберігання даних.
- Другий рівень зазвичай використовуються інфраструктури, необхідної для функціонування доменного шару. Тобто обробляються запити до бази даних, представлені у репозиторіях, що забезпечує структурованість проєкту

- Зовнішні рівні зображають компоненти, котрі дуже часто змінюються, зазвичай це користувацький інтерфейс або допоміжна інфраструктура додатка. В рамках зовнішнього рівня охоплюються веббраузер, мобільні аплікації, телевізійні додатки та інші типи клієнтських аплікацій для доступу до функціонала.

Також було використано патерн “Репозиторій”, котрий є посередником між шаром доступу до даних та доменним.

Пропоную розглянути скелет програмного продукту “Oseredok Management System”.(див Рис. 2.2)

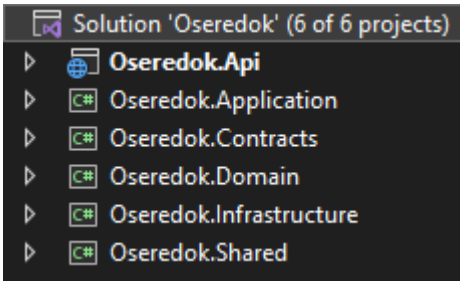

Рис. 2.2 Скелет програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Для кращого розуміння бізнес процесів пропоную розглядати архітектуру від найнижчого рівня до найвищого, тобто від Domain з сутностями до самої Api, в котрій ми можемо тестувати запити наших контролерів.(див Рис. 2.3)

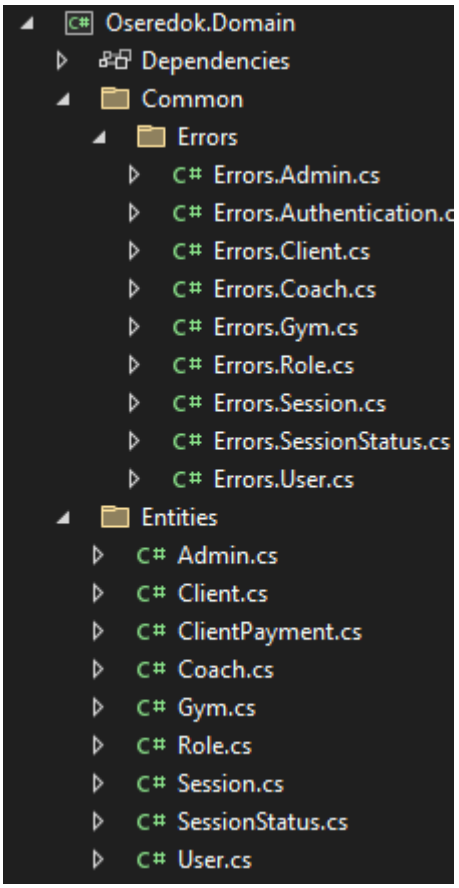

Рис. 2.3 Скелет шару Domain програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

У даному шарі знаходяться сутності - Entities, котрі завдяки Entity Framework [3] описують вимоги до проектування бази даних. Пропоную розглянути одну з них, а саме User.

Лістинг 2.1. Приклад коду однієї з сутностей, а саме сутності юзера.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Oseredok.Domain.Entities
{
    public class User
    {
        public Guid Id { get; set; }
```

```
        [Required]
        [MaxLength(50)]
        public string FirstName { get; set; }

        [Required]
        [MaxLength(50)]
        public string LastName { get; set; }

        [Required]
        [MaxLength(50)]
        public string MiddleName { get; set; }

        [Required]
        public DateTime RegDate { get; set; }

        [Required]
        [ForeignKey(nameof(Role))]
        public int RoleId { get; set; }

        [Required]
        [MaxLength(25)]
        public string PhoneNumber { get; set; }

        [Required]
        [MaxLength(50)]
        public string Email { get; set; }

        [Required]
        [MaxLength(50)]
        public string Password { get; set; }

        public Client Client { get; set; }

        public Role Role { get; set; }

        public Coach Coach { get; set; }
    }
}
```

Тут завдяки DataAnnotations ми маємо можливість описати максимальну довжину певних полів, зазначити чи обов'язкові вони для заповнення, а також, вказуючи максимальну довжину полів, ми економимо ресурси системи, оскільки

виставляємо вручну об’єм пам’яті виділений на дане поле значно менший ніж він є за замовчуванням [9]. Анотація ForeignKey є відображенням зв’язку між таблицями, тобто вказує, що юзеру належить певна роль.

Наступною текою у домені є Errors, котра забезпечує незалежність від інфраструктури шляхом розміщення помилок у доменному шарі, що дозволяє зберігати цей шар незалежним від будь-яких зовнішніх фреймворків, бібліотек або інфраструктурних залежностей. Це полегшує зміни та оновлення інфраструктури, оскільки бізнес-логіка залишається стабільною.

Наступним шаром є Application, він координує роботу доменного та інфраструктурного шару, а саме містить інтерфейси їх взаємодії.(див Рис. 2.4)

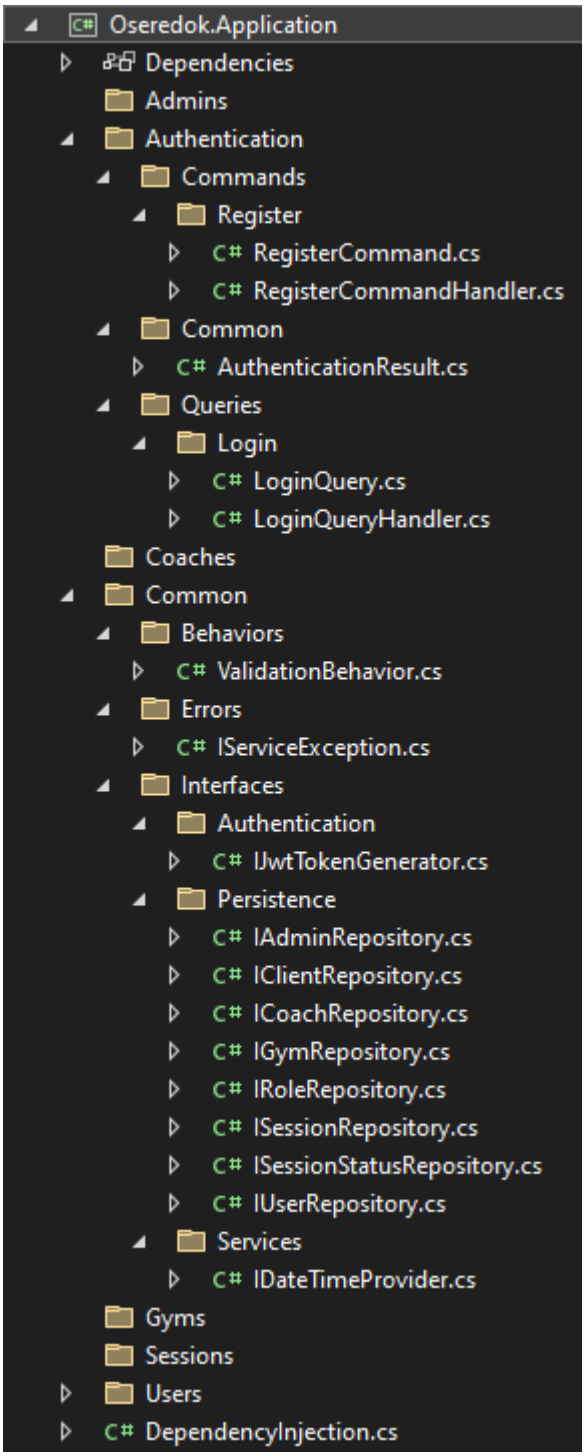

Рис. 2.4 Скелет шару Application програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Важливим є виділити використання патерну CQRS (Command Query Responsibility Segregation) , який розділяє операції читання (запити) від операцій запису (команди). Замість традиційного підходу, де об'єкт моделі представляє

одночасно стан та поведінку, CQRS використовує окремі моделі для запитів та команд.

Основні концепції CQRS:

1. Команда (Command): Представляє дію або запит на зміну стану системи. Команди є імутабельними та не мають повернутої значення. Вони виконуються для зміни даних або виклику певних дій у системі.
2. Запит (Query): Представляє запит на отримання даних з системи. Запити не змінюють стану системи і повертають значення, які відображають поточний стан системи.
3. Модель команди (Command Model): Це модель, яка обробляє команди та виконує необхідні зміни стану системи. Вона має валідувати команди, виконувати бізнес-логіку та змінювати стан системи відповідно до команди.
4. Модель запиту (Query Model): Це модель, яка підтримує запити та повертає необхідні дані для запитувача. Вона повинна бути оптимізована для ефективного та швидкого отримання даних.
5. Інфраструктура команд та запитів: Це компоненти, які дозволяють маршрутизувати команди до відповідних моделей команди та запитувати дані з моделей запиту.

MediatR є бібліотекою для реалізації патерну Mediator (Посередник) в додатках на платформі .NET. Вона надає простий спосіб реалізації Mediator pattern без прив'язки до конкретної реалізації.

Основні концепції MediatR:

1. Посередник (Mediator): Це центральний компонент, який виступає посередником між запитами та їх обробниками. Він приймає запити і маршрутизує їх до відповідних обробників. MediatR спрощує взаємодію між компонентами, зменшує залежності та дозволяє легко додавати нові запити та обробники без зміни існуючого коду.
2. Запити (Requests): Запити в MediatR представляють запити на виконання певної дії або отримання даних. Вони можуть містити необхідні параметри для виконання запиту.

3. Обробники (Handlers): Обробники в MediatR відповідають за обробку запитів. Кожен запит може мати свій відповідний обробник, який містить бізнес-логіку для обробки запиту та повернення результату.
4. Пайплайн (Pipeline): MediatR надає можливість додавати пайплайни, які дозволяють модифікувати поведінку перед виконанням запиту або після його виконання. Це дає можливість реалізувати додаткову логіку, таку як логування, авторизацію, кешування і т. д.
5. Медіатори (Mediators): MediatR також підтримує концепцію медіаторів, які можуть виступати як посередники між різними компонентами системи. Це дозволяє спростити взаємодію між компонентами та розподілити відповідальність між ними.

Використання MediatR разом з CQRS паттерном дозволяє розділити відповідальності, забезпечити більш гнучку та масштабовану архітектуру. Він спрощує розробку, тестування та розширення системи шляхом забезпечення єдиної точки взаємодії та незалежності між компонентами.

У теці Common знаходяться інтерфейси для репозиторіїв, про які пропоную розглянути детальніше. Паттерн репозиторію є частиною шаблону проектування "Repository" і використовується для роботи з постійним сховищем даних. У контексті обробників команд та запитів, репозиторій виконує завдання доступу до даних і забезпечує роботу з певними моделями або агрегатами даних.

Основні концепції паттерна репозиторію:

1. Інтерфейс репозиторію: Це контракт, який описує операції, доступні для роботи з даними. Інтерфейс визначає методи, такі як додавання, видалення, оновлення та отримання даних.
2. Реалізація репозиторію: Це конкретна реалізація інтерфейсу репозиторію. Вона виконує операції доступу до певного сховища даних, такого як база даних або файлова система. Реалізація репозиторію може включати методи для отримання, збереження, видалення та оновлення даних.

3. Об'єкт даних: Це модель або агрегат даних, з яким працює репозиторій. Об'єкт даних може мати різну структуру в залежності від вимог додатка і може включати різні властивості та методи для роботи з даними.
4. Взаємодія з репозиторієм в обробниках команд та запитів: В обробниках команд та запитів, репозиторій використовується для отримання, збереження, видалення або оновлення даних, пов'язаних з конкретною операцією. Обробники команд та запитів використовують інтерфейс репозиторію для взаємодії з даними.

Переваги використання паттерна репозиторію в обробниках команд та запитів:

1. Розділення відповідальностей: Репозиторій в архітектурі допомагає розділити логіку доступу до даних від бізнес-логіки обробки команд та запитів. Це дозволяє зберігати обробники команд та запитів незалежними від конкретного джерела даних та спрощує тестування цих компонентів.
2. Перевикористання коду: Репозиторій надає єдину точку доступу до даних для всіх обробників команд та запитів. Це дозволяє перевикористовувати код доступу до даних і спільну логіку між різними обробниками.
3. Забезпечення контролю доступу до даних: Репозиторій може включати логіку контролю доступу до даних, таку як авторизація та перевірка дозволів. Це дозволяє захистити дані від несанкціонованого доступу.
4. Замінюваність джерела даних: Завдяки використанню репозиторію, можна легко замінити конкретне джерело даних, наприклад, змінити базу даних або замінити його на зовнішній веб-сервіс. Це дозволяє зручно масштабувати або змінювати сховища даних без впливу на обробники команд та запитів.

Загалом, паттерн репозиторію допомагає створити окремий шар доступу до даних у вашій архітектурі, що забезпечує чистоту та розсіяність в коді, спрощує тестування та дозволяє гнучко працювати з даними у контексті обробників команд та запитів MediatR.

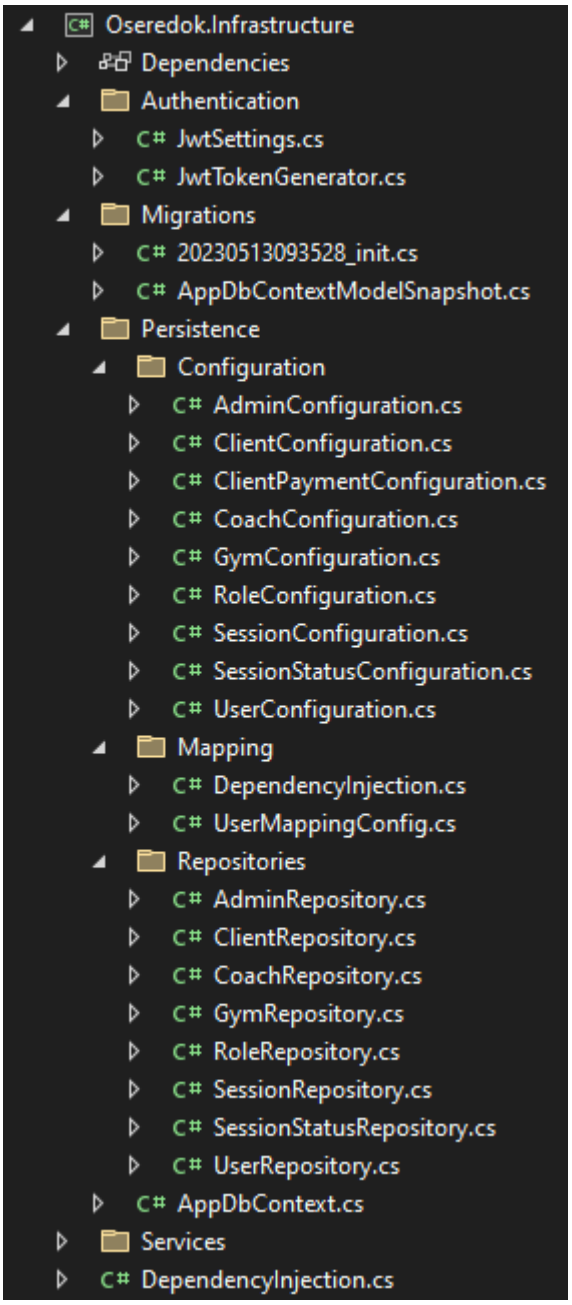

Рис. 2.5 Скелет шару Infrastructure програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Наступним шаром є Infrastructure (див Рис. 2.5) у чистій архітектурі він відповідає за всі зовнішні технічні деталі та інфраструктуру системи. Цей шар забезпечує взаємодію з зовнішніми ресурсами, такими як бази даних, зовнішні сервіси, файлові системи і т. д. Він включає в себе різноманітні технологічні компоненти, необхідні для забезпечення доступу до цих ресурсів та виконання різних операцій. В цілому він імплементує інтерфейси шару Application, тобто

містить їх реалізацію. У даному випадку у цьому шарі знаходяться міграції БД, конфігурація БД, у якій знаходяться початкові дані та реалізація різноманітних сервісів, а точніше імплементуються інтерфейси репозиторіїв та сервісів створення JWT токенів.

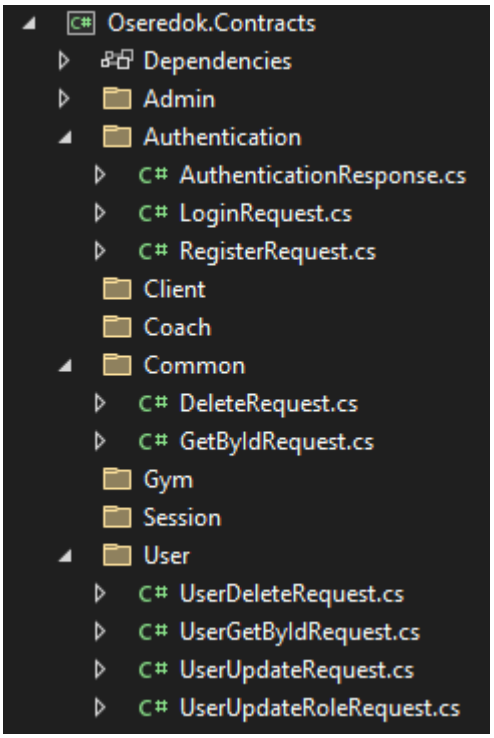

Рис. 2.6 Скелет шару Contracts програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Не таким громіздким, але не менш важливим є шар Contracts (див Рис. 2.6), у ньому зберігаються моделі запитів та відповідей, котрі використовуються у Api шарі, який буде описано пізніше. Важливо підкреслити, що цей шар забезпечує моделі передачі даних між шарами Application та Api, а для визначення моделей передачі даних між Infrastructure та Application ми використовуємо шар Shared (див Рис. 2.7)

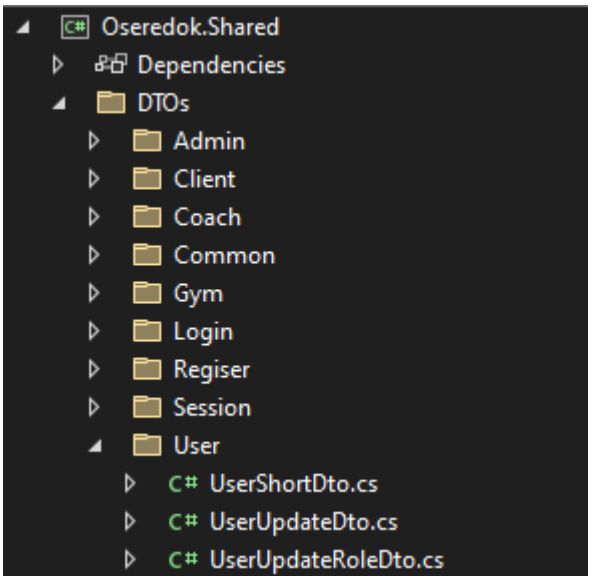

Рис. 2.7 Скелет шару Shared програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Моделі даного шару визначають, яку інформацію репозиторій з бази даних передасть до медіатора у потрібному нам вигляді, тобто включаючи в себе нові та видаляючи непотрібні поля. Використання шарів Contracts і Shared у чистій архітектурі допомагає досягти кращого розуміння в коді та забезпечити перевикористання функціональності між різними компонентами системи. Обидва шари мають свої особливості та завдання, які було описано вище.

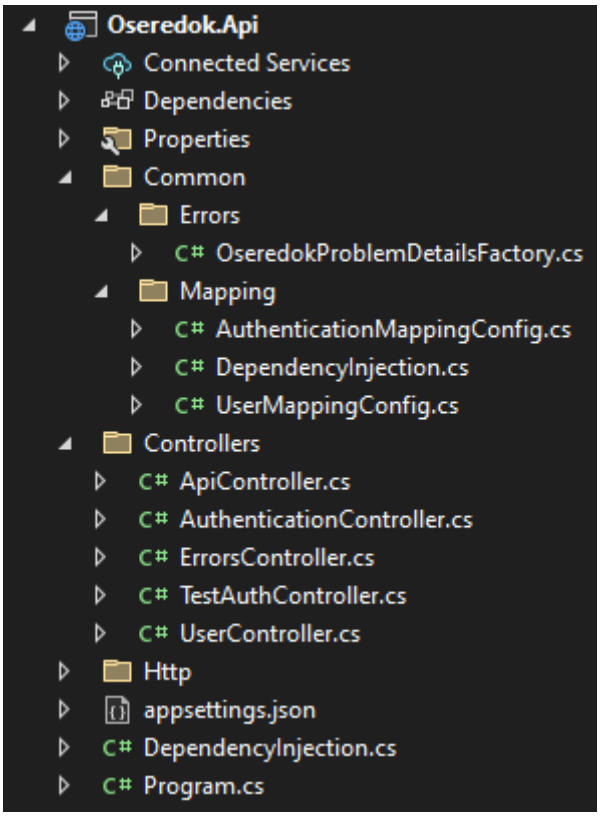

Рис. 2.8 Скелет шару Api програмного продукту “Oseredok Management System”.

Джерело:[створено автором]

Останнім шаром є Api(див Рис. 2.8). Він відповідає за надання зовнішнього інтерфейсу для взаємодії з системою. Він використовується для комунікації з клієнтськими додатками, іншими системами або сервісами. Шар Api забезпечує доступ до функціональності системи та обробку запитів і команд від клієнтів. Також включає механізми аутентифікації та авторизації для контролю доступу до функціональності системи. У моєму випадку це реалізовано за допомогою JWT токенів, про які буде розказано пізніше. Також в даному шарі знаходиться файл Program.cs, котра є точною входу в програму, у ній знаходяться конфігурації програмного рішення.

2.1.2. Опис стека технологій серверної частини

Вибір стеку технологій для розробки серверної частини веб-додатку є критичним кроком, який впливає на продуктивність, швидкість розробки, масштабованість та інші аспекти проекту. Ось кілька причин, чому вибір стеку технологій є важливим:

- Функціональність та потужність: Вибір стеку технологій повинен враховувати потреби проекту. Кожен стек має свої особливості та можливості.
- Спільнота та екосистема: Вибір популярного та активно розвиваючогося стеку технологій може забезпечити доступ до великої спільноти розробників, різноманітних інструментів та бібліотек. Це допоможе вам швидко розв'язувати проблеми, отримувати підтримку та використовувати передові розробки у своєму проекті.
- Продуктивність та швидкість розробки: Вибір стеку технологій, який має зручний інструментарій та добре знайомий розробнику, може значно покращити продуктивність та швидкість розробки. Якщо у вас вже є досвід роботи з певними технологіями, то використання їх у вашому проекті дозволить вам швидше впроваджувати нові функції та вирішувати завдання.
- Масштабованість та продуктивність: Деякі стеки технологій мають кращу підтримку масштабованості та здатність працювати з високим навантаженням. Наприклад, стеки, що використовують асинхронне програмування або використовують спеціалізовані сервери, можуть забезпечити кращу продуктивність та масштабованість.
- Безпека: Вибір стеку технологій може мати вплив на безпеку вашого веб-додатку. Деякі технології мають вбудовані механізми захисту та легше справляються з вразливостями безпеки. Важливо обрати стек, який має активну спільноту та регулярні оновлення для забезпечення безпеки вашого додатку.

Вибір стеку технологій повинен бути зроблений на основі ваших потреб, ресурсів та досвіду використання, а вивчення альтернатив, проведення досліджень

та оцінка переваг і недоліків допоможуть зробити оптимальний вибір для вашого проекту.

Для розробки програмного продукту, котрий передбачає використання різноманітних бібліотек та фреймворків було використано наступний стек технологій:

- C# - сучасна об’єктно орієнтована та строго типізована мова програмування. Дозволяє розробникам Дозволяє розробникам розробляти різноманітні види захищених та надійних програмних рішень, котрі виконуються у .NET. C# бере початок з сімейства мов C, тому розробники таких мов як: C, C++, Java, JavaScript будуть вважати її досить знайомою.[1]
- ASP.NET Core - кросплатформовий, високопродуктивний фреймворк для розробки вебдодатків. Даний фреймворк об’єднує ASP.NET MVC та ASP.NET Web API у єдину кросплатформову версію.[4]
- Entity Framework Core - проста кросплатформова та розширена версія Entity Framework, котра використовується для роботи з БД за допомогою об’єктів .NET, та спрощує написання коду для доступу до даних.[3]
- Code First - це підхід до розробки бази даних, в якому модель бази даних визначається спочатку у вигляді класів або об'єктів у програмному коді, і потім база даних створюється автоматично на основі цих класів. Замість написання SQL-скриптів для створення бази даних та таблиць, ви створюєте класи з атрибутами та властивостями, які відповідають сутностям та таблицям бази даних.
- SQL Server - це система управління базами даних (СУБД) розроблена компанією Microsoft. Вона забезпечує надійне та ефективне зберігання, обробку та управління структурованими даними.
- Mapster - це легка використання бібліотека для мапінгу об'єктів (object mapping) в .NET. Вона надає зручний та ефективний спосіб копіювання даних з одного об'єкта в інший, використовуючи різні стратегії мапінгу[11].
- Swagger - це набір інструментів для створення, документування та спілкування з веб-сервісами API. Він надає зручний спосіб описувати структуру та

функціональні можливості API за допомогою OpenAPI Specification (раніше відомого як Swagger Specification). Бібліотека Swagger надає можливості автоматичного генерування документації, інтерактивного UI (Swagger UI) для тестування API, генерації клієнтських бібліотек та багато іншого [5].

- JSON Web Token (JWT) - це стандарт (RFC 7519), котрий є стислим та самодостатнім способом захищеної передачі інформації між сторонами JSON об’єкту. Дана інформація може бути перевіреною та їй можна довіряти, бо вона має цифровий підпис. JWT можна підписати за допомогою секрету (з HMAC алгоритмом).[9]
- DTO (Data Transfer Object) - це об'єкт, який використовується для передачі даних між різними компонентами системи, такими як клієнтська частина та серверна частина, або між сервісами в мікросервісній архітектурі. Головна мета DTO - передача даних без стану та без логіки, просто як контейнер для даних.
- Dependency Injection (DI) в ASP.NET є підходом до керування залежностями, який дозволяє легко впроваджувати залежності між компонентами додатка. Це важлива концепція в розробці програмного забезпечення, що сприяє зменшенню залежностей, полегшує тестування і забезпечує більшу гнучкість та розширюваність коду.
- MediatR - це бібліотека для реалізації паттерна Mediator в .NET. Вона надає зручний спосіб розбити логіку додатка на окремі запити (Queries) і команди (Commands) і дозволяє ці об'єкти виконувати з допомогою посередника (Mediator).
- ErrorOr - це бібліотека, яка надає механізм для обробки помилок та повернення результату або помилки у функціональному стилі. Її основна ідея полягає в тому, що функції повертають об'єкти, які можуть містити результат або помилку, що дозволяє зручно та точно обробляти винятки та валідацію.

2.2. Опис реалізації серверної частини вебсервісу

Перш за все я проаналізував бізнес-процеси та виділив основні сутності для побудови БД, після перебору декількох варіантів структури обираю найбільш зручний для реалізації. Саме моделювання бази даних я проводив за допомогою методу Code First [8]. Даний метод завдяки Entity Framework Core дозволяє побудувати БД за допомогою описаних класів сутностей та контексту. Далі за допомогою можливостей фреймворку генерую міграцію та застосовую її до БД, тобто створюю її, у наступному вигляді(див. Рис. 2.9).

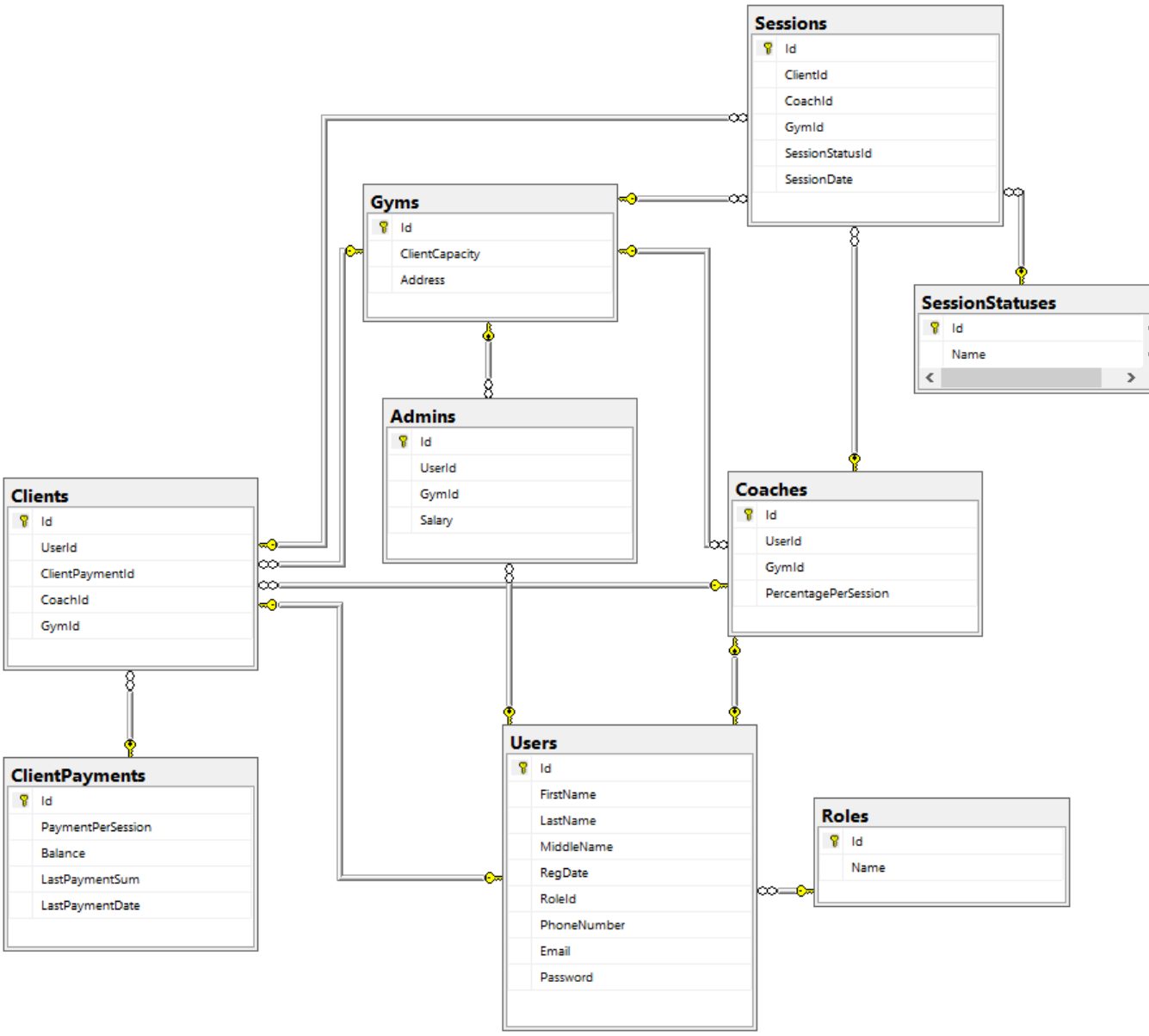

Рис 2.9. Діаграма БД програмного продукту “Oseredok Management System”.

Джерело: [створено автором].

Після створення БД, перейшов до формування архітектури проекту, як описано вище, було обрано “Clean Architecture” та CQRS патерн[10] з використанням Repository[13]. Даний вибір забезпечив набагато більшу гнучкість серверної частини, проте значно збільшив час на написання коду, адже Mediator патерн створює ще один шар між репозиторієм та контролером.

Вже маючи робочу БД та сформовану архітектуру, почав роботу над репозиторієм доступу до БД. Для кожної з сутностей створив інтерфейс, де описав базові запити та імплементував його у відповідний репозиторій, створив необхідні DTO та налаштував їх мапінг . Після цього було зроблено відповідні обробники запитів медіатора та створено DTO для них у шарі Contracts.

Далі я створив відповідні контроллери, котрі забезпечують обмін даними між сервером та клієнтом. Для тестування створених запитів використовую Swagger, котрий додає зручності у виконанні поставленої задачі.

Перш за все була розпочата робота над авторизацією та автентифікацією за допомогою JWT токену. (Рис. 2.10)

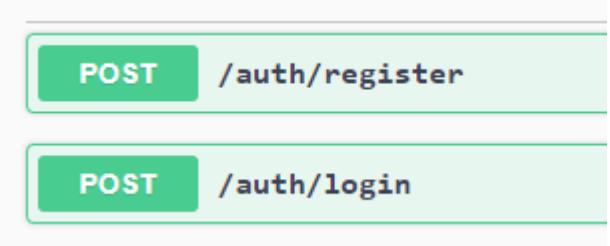

Рис. 2.10 Контролери реєстрації та автентифікації

Джерело:[створено автором]

Процес реєстрації юзера є наступним:

Перш за все за допомогою контролера з шару Api ми передаємо необхідні для реєстрації користувача дані, вони зображені у запиті RegisterRequest(див. Лістинг 2.2) з шару Contracts, котрий мапиться до RegisterCommand з шару Application(див. Лістинг 2.3) у самому ж контролері, а далі буде передана до обробника команди(див. Лістинг 2.4).

Лістинг 2.2. Код RegisterRequest.

```
namespace Oseredok.Contracts.Authentication
{
    public record RegisterRequest(

     string FirstName,

     string LastName,

     string MiddleName,

     string PhoneNumber,

     string Email,

     string Password
    );
}
```

Лістинг 2.3. Код RegisterCommand.

```
using ErrorOr;
using MediatR;
using Oseredok.Application.Authentication.Common;

namespace Oseredok.Application.Authentication.Commands.Register
{
    public record RegisterCommand(
     string FirstName,

     string LastName,

     string MiddleName,

     string PhoneNumber,

     string Email,

     string Password,

     DateTime DateOfBirth) : IRequest<ErrorOr<AuthenticationResult>>;
}
```

Лістинг 2.4. Код RegisterCommandHandler.

```
using ErrorOr;
using MapsterMapper;
using MediatR;
using Oseredok.Application.Authentication.Common;
using Oseredok.Application.Common.Interfaces.Authentication;
using Oseredok.Application.Common.Interfaces.Persistence;
using Oseredok.Domain.Common.Errors;
using Oseredok.Domain.Entities;

namespace Oseredok.Application.Authentication.Commands.Register
{
    public class RegisterCommandHandler : IRequestHandler<RegisterCommand,
ErrorOr<AuthenticationResult>>
    {
        private readonly IJwtTokenGenerator _jwtTokenGenerator;
        private readonly IUserRepository _userRepository;
        private readonly IMapper _mapper;

        public RegisterCommandHandler(
            IJwtTokenGenerator jwtTokenGenerator,
            IUserRepository userRepository,
            IMapper mapper)
        {
            _jwtTokenGenerator = jwtTokenGenerator;
            _userRepository = userRepository;
            _mapper = mapper;
        }

        public async Task<ErrorOr<AuthenticationResult>>
Handle(RegisterCommand command, CancellationToken cancellationToken)
        {
            await Task.CompletedTask;
            var UserWithSameEmail = await
_userRepository.GetUserByEmail(command.Email);
            if (UserWithSameEmail?.Email == command.Email)
            {
                return Errors.User.DuplicateEmail;
            }
            var UserWithSamePhoneNumber = await
_userRepository.GetUserByPhoneNumber(command.PhoneNumber);
```

```
            if (UserWithSamePhoneNumber?.PhoneNumber ==
command.PhoneNumber)
            {
                return Errors.User.DuplicatePhoneNumber;
            }

            var newUser = _mapper.Map<User>(command);

            await _userRepository.Add(newUser);

            var user = await _userRepository.GetById(newUser.Id);

            var token = _jwtTokenGenerator.GenerateToken(user.Value);

            return new AuthenticationResult(
                user.Value.Id,
                user.Value.Role.Name,
                user.Value.FirstName,
                user.Value.LastName,
                user.Value.Email,
                user.Value.Password,
                token);
        }
    }
}
```

На цьому етапі вхідні дані проходять невелику валідацію, а саме проводиться перевірка чи немає у БД юзерів з однаковою електронною адресою або ж номером телефону, створюється на їх основі юзер за допомогою відповідного методу у UserRepository у шарі Infrastructure(див. Лістинг 2.5) та його дані передаються до сервісу генерації токену JwtTokenGenerator(див. Лістинг 2.6) через його інтерфейс у шар Infrastructure.

Лістинг 2.5. Метод Add у UserRepository.

```
public async Task Add(User user)
        {
            user.Id = Guid.NewGuid();
            user.RoleId = 4;
            user.RegDate = DateTime.Now;
```

```
            await _ctx.AddAsync(user);
            await _ctx.SaveChangesAsync();
        }
```

Логіка програмного рішення побудована так, що будь-який користувач після реєстрації отримує роль “noRole”, і лише адміністратор має доступ до методу котрий може змінити роль користувача на потрібну. Таке рішення було визначене як оптимальне для невеликого спортивного залу, коли майбутній клієнт може зареєструватись з власного смартфону або ж пк, а вже на місці адміністратор видасть йому відповідну роль.

Лістинг 2.6. Код JwtTokenGenerator.

```
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using Oseredok.Application.Common.Interfaces.Authentication;
using Oseredok.Application.Common.Interfaces.Services;
using Oseredok.Domain.Entities;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace Oseredok.Infrastructure.Authentication
{
    public class JwtTokenGenerator : IJwtTokenGenerator
    {
        private readonly IDateTimeProvider _dateTimeProvider;
        private readonly JwtSettings _jwtSettings;

        public JwtTokenGenerator(IDateTimeProvider dateTimeProvider,
IOptions<JwtSettings> jwtOptions)
        {
            _dateTimeProvider = dateTimeProvider;
            _jwtSettings = jwtOptions.Value;
        }

        public string GenerateToken(User user)
        {
            var signingCredentials = new SigningCredentials(
              new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(_jwtSettings.Secret)),
              SecurityAlgorithms.HmacSha256);
```

```
            var claims = new[]{
                new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
                new Claim(JwtRegisteredClaimNames.GivenName,
user.FirstName),
                new Claim(JwtRegisteredClaimNames.FamilyName,
user.LastName),
                new Claim(ClaimTypes.Role, user.Role.Name),
                new Claim(JwtRegisteredClaimNames.Jti,
Guid.NewGuid().ToString()),
        };

            var securityToken = new JwtSecurityToken(
              issuer: _jwtSettings.Issuer,
              audience: _jwtSettings.Audience,
              claims: claims,
              expires:
_dateTimeProvider.UtcNow.AddMinutes(_jwtSettings.ExpiryMinutes),
              signingCredentials: signingCredentials);
            return new JwtSecurityTokenHandler().WriteToken(securityToken);
        }
    }
}
```

Після чого токен повертається до команди та разом з вхідними даними мапиться до відповіді - AuthenticationResult (див. Лістинг 2.7)

Лістинг 2.7. Код AuthenticationResult

```
namespace Oseredok.Application.Authentication.Common
{
    public record AuthenticationResult(

    Guid Id,

    string FirstName,

    string LastName,

    string Email,

    string Password,

    string Token
```

```
    );
}
```

Після цього ця відповідь повертається до контролера, де перевіряється чи були викликані обробники помилок, якщо ні, то контролер(див. Лістинг 2.8) повертає відповідь, якщо так, то саму проблему.

Лістинг 2.8. Код AuthenticationController

```
[HttpPost("register")]
        public async Task<IActionResult> Register(RegisterRequest request)
        {
            var command = _mapper.Map<RegisterCommand>(request);
            ErrorOr<AuthenticationResult> authResult = await
_mediator.Send(command);

            return authResult.Match(
                authResult =>
Ok(_mapper.Map<AuthenticationResponse>(authResult)),
                errors => Problem(errors));
        }
```

Після огляду роботи контролера реєстрації пропоную протестувати його, заповнивши наступними даними його тіло запиту RegisterRequest (див. Лістинг 2.2) даними з вже існуючим номером телефону(Див рис. 2.11)

Рис. 2.11 Тіло контролера реєстрації з вже використаним телефонним номером

Джерело:[створено автором]

Після введення даних отримуємо результат обробленої помилки, у якій зібрана коротка інформація про неї[12] (Див рис. 2.12)

.

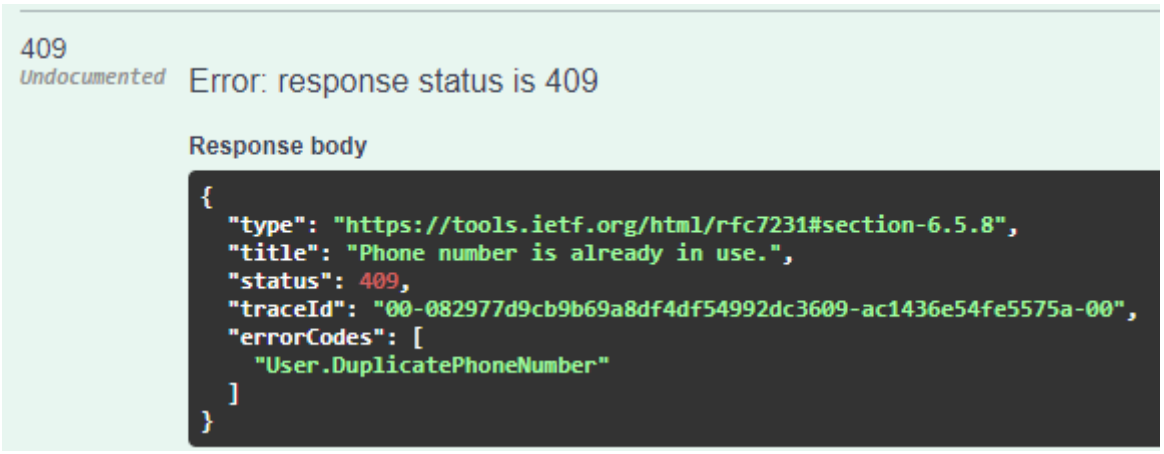

Рис. 2.12 Відповідь на запит реєстрації з вже використаним номером телефону

Джерело:[створено автором]

Повторимо Операцію, проте тепер виправимо номер телефону на новий, але введемо вже використану пошту(Див рис. 2.13).

Рис. 2.13 Тіло контролера реєстрації з вже використаною електронною поштою

Джерело:[створено автором]

Після введення даних отримуємо результат обробленої помилки, у якій зібрана коротка інформація про неї (Див рис. 2.14)

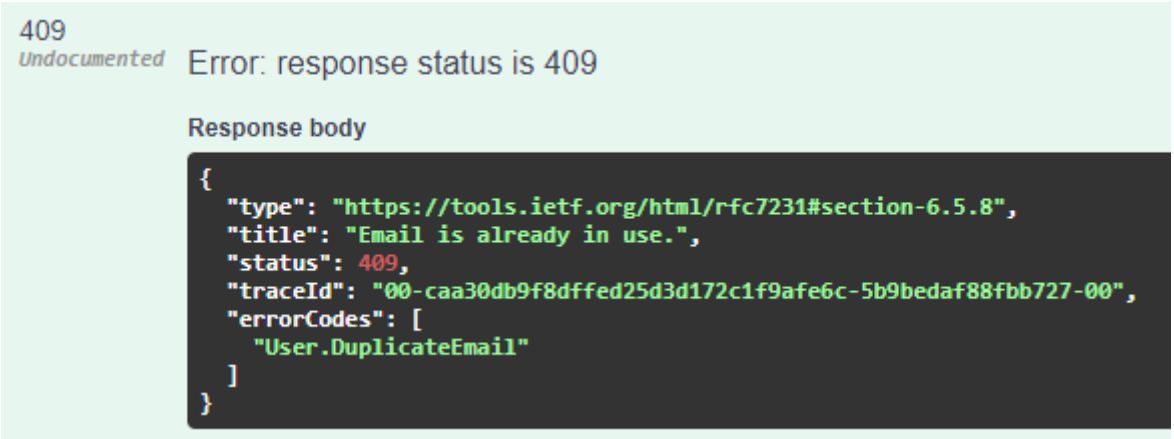

Рис. 2.14 Відповідь на запит реєстрації з вже використаним номером телефону

Джерело:[створено автором]

Тепер розглянемо випадок, коли всі умови були дотримані(Див рис. 2.15)

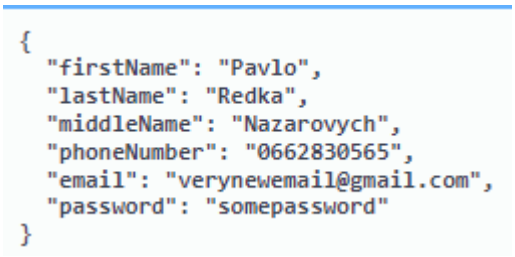

Рис. 2.15 Тіло контролера реєстрації

Джерело:[створено автором]

Після введення даних отримуємо результат AuthenticationResult(див. Лістинг 2.7), де передаються дані(Див рис. 2.16)

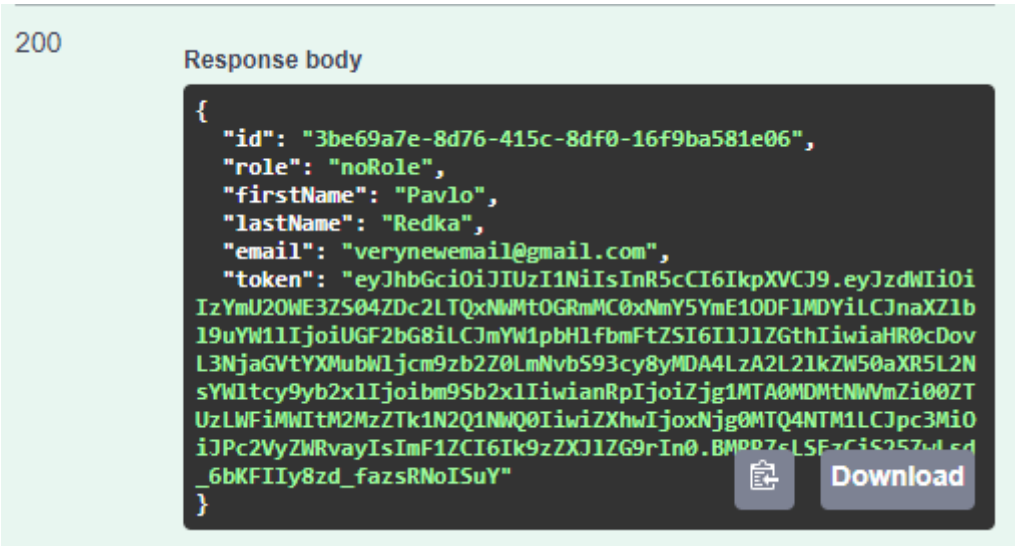

Рис. 2.16 Відповідь на запит реєстрації

Джерело:[створено автором]

На даному прикладі ми розглянули принцип роботи контролера у даному веб сервісі. Тепер можемо використати отриманий токен для авторизації користувача, що надасть йому доступ до використання дозволених ендпоїнтів.

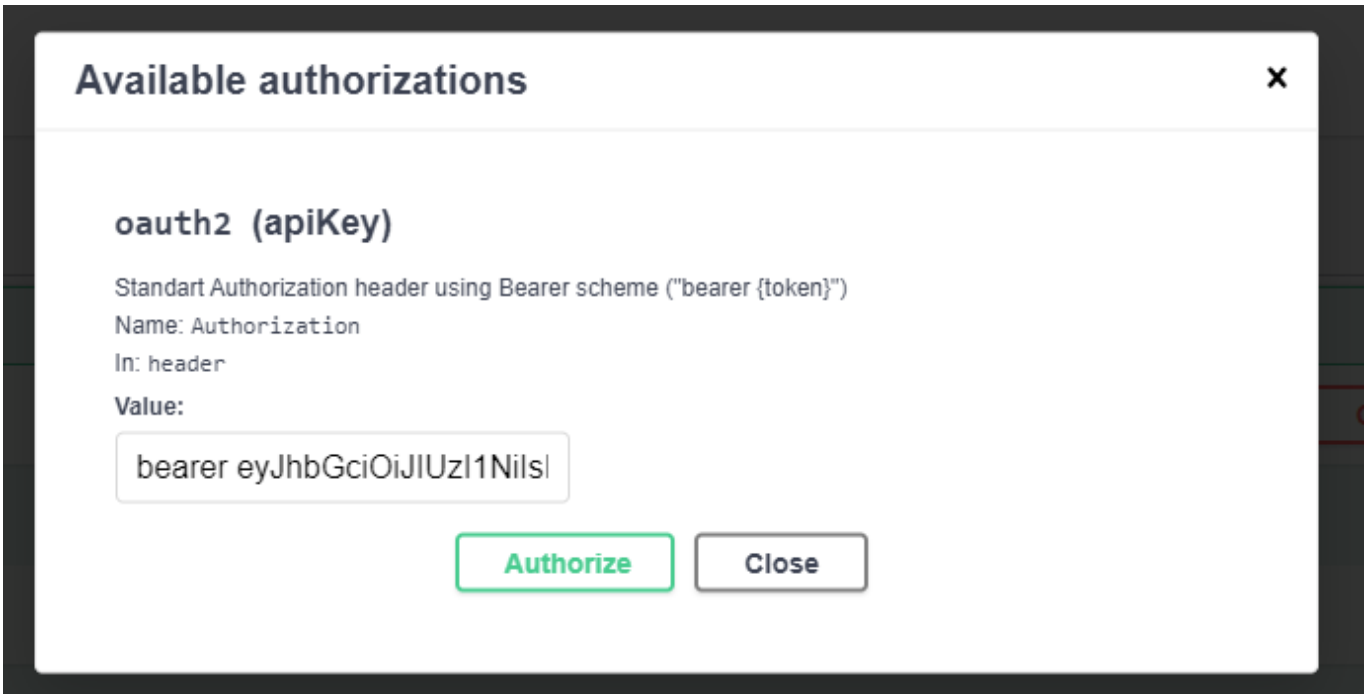

Рис. 2.16 Тестування авторизації у Swagger

Джерело:[створено автором]

Для перевірки успішності авторизації у мене є тестовий контролер TestAuth,

Лістинг 2.9. Код контролера TestAuth

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace Oseredok.Api.Controllers
{
    [Route("[controller]")]
    public class TestAuthController : ApiController
    {
        [Authorize(Roles = "admin")]
        [HttpGet]
        public IActionResult Test()
        {
            return Ok((Array.Empty<string>()));
        }
    }
}
```

Даний контролер вимагає роль адміністратора, а отже наш новостворений юзер не має доступу до цього методу, оскільки роллю за замовчуванням є “noRole”. Тому результатом нашого запиту є помилка(Див рис. 2.17)

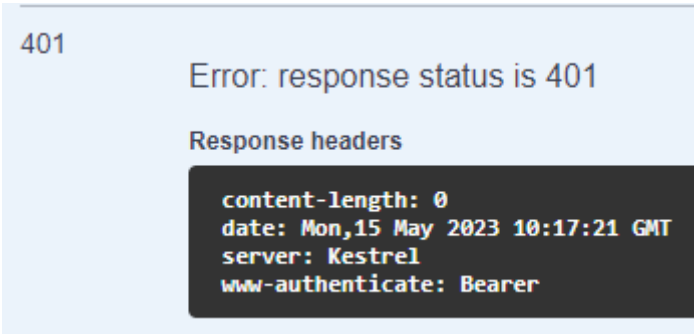

Рис. 2.17 Відповідь на запит з невідповідною роллю

Джерело:[створено автором]

Тепер пропоную авторизуватись на обліковий запис користувача з роллю адміністратора та повторити операцію. (Див рис. 2.18)

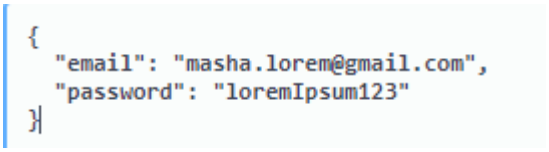

Рис. 2.18 Тіло запиту login

Джерело:[створено автором]

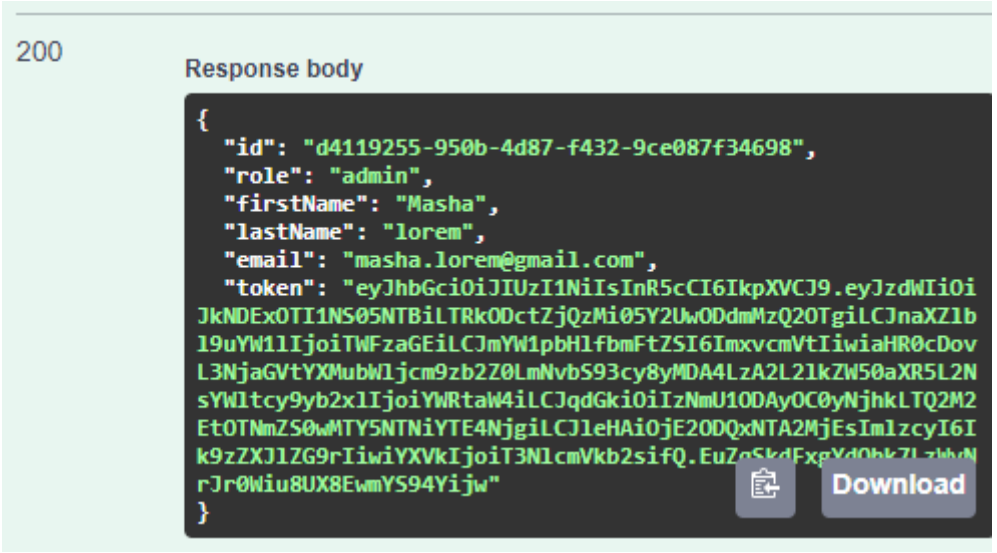

Рис. 2.19 Відповідь запиту login

Джерело:[створено автором]

Переконуюсь, що у обраного юзера роль адміністратора та отриманий з відповіді токен використовуємо для авторизації та повторюємо тестування захищеного запиту.

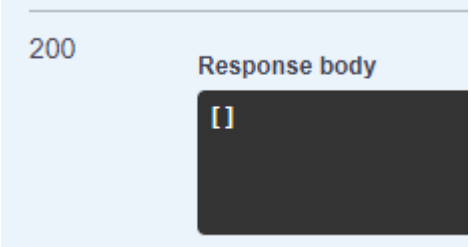

Рис. 2.19 Відповідь на запит з відповідною роллю

Джерело:[створено автором]

Статус код показує нам 200, тому робимо висновок, що все працює належним чином.

При розробці серверної частини на ASP.Net фреймворку було використано сучасні технології та інструменти, що сприяли створенню ефективної та масштабованої системи. Описаний процес розробки включав кілька ключових етапів.

Починаючи з побудови бази даних, було використано Entity Framework з методом побудови Code First. Цей підхід дозволяє описувати моделі даних у вигляді класів та автоматично створювати відповідні таблиці в базі даних. Використання Entity Framework спрощує роботу з базою даних і забезпечує зручну взаємодію з даними.Для забезпечення безпеки та аутентифікації була реалізована генерація JWT (JSON Web Token) [7]. Цей механізм дозволяє створювати токени, які містять інформацію про користувача та його права, забезпечуючи контроль доступу до ресурсів. Це спрощує процес автентифікації та забезпечує безпеку системи.Для реалізації патерну CQRS (Command-Query Responsibility Segregation) була використана бібліотека MediatR. Вона дозволяє розділити операції запису (команди) та операції читання (запити), спрощуючи структуру системи та полегшуючи розширення. Використання MediatR сприяє покращенню продуктивності системи та підтримці чистого коду.Для обробки помилок була використана бібліотека ErrorOr. Вона надає зручний механізм для повернення помилок та результатів операцій,

спрощуючи управління помилками та роботу з виключеннями.Для мапінгу даних була використана бібліотека Mapster, яка є більш ефективним аналогом бібліотеки AutoMapper. Вона дозволяє зручно та ефективно виконувати мапінг об'єктів, спрощуючи процес перетворення даних.Для тестування контролерів та взаємодії з серверною частиною було використано Swagger. Swagger надає зручний інтерфейс для тестування API, що спрощує перевірку та взаємодію з ресурсами. Це сприяє покращенню якості та стабільності системи.

Загалом, розробка серверної частини на ASP.Net фреймворку з використанням Entity Framework, генерації JWT токенів, патерну CQRS у бібліотеці MediatR, ErrorOr для обробки помилок, Mapster для мапінгу та Swagger для тестування контролерів дозволила створити потужну та надійну систему з ефективною архітектурою. Ці технології та підходи сприяють зручній розробці, підтримці та розширенню системи.

2.3. Опис архітектурного рішення та технологічного стека клієнтської частини

Вибір правильної архітектури клієнтської частини вебсервісу є критично важливим для ефективної взаємодії з користувачами. Від цього залежать продуктивність, швидкість та надійність системи. Правильна архітектура сприяє швидкій роботі, масштабованості та надійності, дозволяючи легко впроваджувати зміни та розширювати функціонал. Зручність використання для користувачів є також важливим фактором, що вимагає зрозумілого та інтуїтивного інтерфейсу. Важливо враховувати вимоги проекту, потреби користувачів, екосистему та інтеграцію з іншими компонентами системи. Неправильний вибір архітектури може призвести до проблем з масштабованістю, продуктивністю та складністю розробки.

2.3.1. Опис архітектурного рішення клієнтської частини

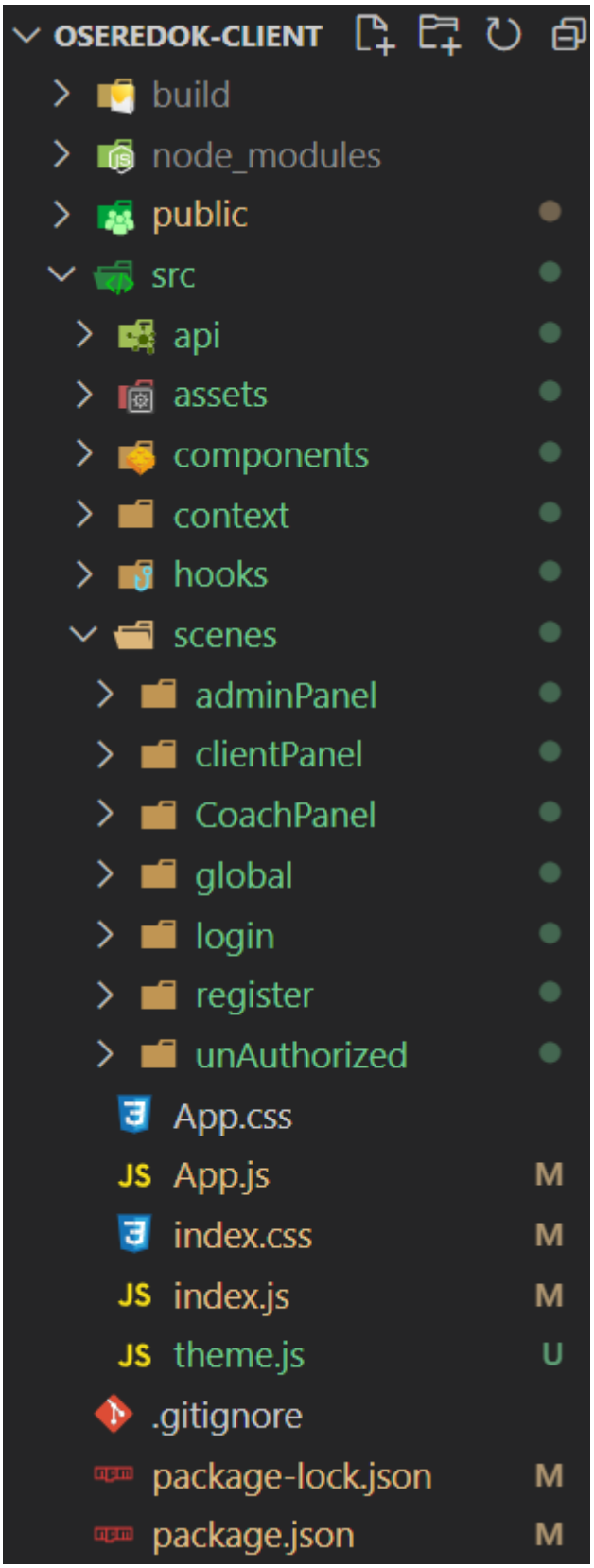

Рис. 2.19 Архітектура клієнтської частини вебдодатку

Джерело:[створено автором]

Наведена вище архітектура є доволі звичною для react застосунків, усі призначення створених тек є інтуїтивно зрозумілим і не глибоко захованими, проте хочу звернути увагу на теку “scenes”, яку найчастіше можна було б бачити як “pages”. Традиційно, у React додатки організовані з використанням тек "pages". Кожна сторінка відповідає окремому компоненту, який містить весь необхідний код для цієї сторінки. Цей підхід зручний, коли додаток складається з окремих сторінок з мінімальним переходом між ними. Наприклад, веб-сайт зі статичними сторінками може бути добрим варіантом для використання тек "pages".

Однак, з ростом складності додатків та введенням концепції односторінкового додатку (SPA), можуть виникати потреби в більш гнучкій організації коду. У таких випадках використання тек "scenes" може бути корисним.

Теки "scenes" використовуються для групування компонентів, які відповідають за конкретні розділи або сцени додатку. Замість розділення коду на окремі сторінки, компоненти, пов'язані з певною сценою, розміщуються відповідною текою "scenes". Це дозволяє краще організувати код, зменшує залежності між компонентами та сприяє повторному використанню коду.

Варто також зазначити, що ці підходи не є взаємовиключними і можуть бути поєднані. Можна використовувати теки "scenes" для групування компонентів на рівні сцен або модулів, а потім використовувати теки "pages" для організації окремих сторінок в межах кожної сцени.

2.3.2. Опис стека технологій клієнтської частини

Для розробки програмного продукту, котрий передбачає використання різноманітних бібліотек було використано наступний стек технологій:

- HTML (Hypertext Markup Language) є стандартною мовою розмітки, що використовується для створення веб-сторінок. Вона дозволяє структурувати та визначати зміст сторінки за допомогою спеціальних тегів.
- CSS (Cascading Style Sheets) є мовою стилів, що використовується для оформлення та опису зовнішнього вигляду веб-сторінок.

- JS (JavaScript) є високорівневою, інтерпретованою мовою програмування, що використовується для реалізації динамічної функціональності на веб-сторінках. Вона дозволяє взаємодіяти з користувачем, обробляти події, маніпулювати елементами сторінки, змінювати та оновлювати зміст без перезавантаження сторінки.
- React є відкритою бібліотекою JavaScript, яка використовується для розробки користувацьких інтерфейсів (UI) веб-додатків. Вона дозволяє розбити веб-інтерфейс на незалежні компоненти, які можна повторно використовувати та легко управляти. React використовує віртуальний DOM (Document Object Model), що забезпечує ефективне оновлення тільки змінених елементів на сторінці. Це дає велику швидкість та продуктивність веб-додаткам. Також підтримує розширення за допомогою додаткових бібліотек та інструментів, що сприяє розробці більш складних додатків [14].
- React Router DOM - це бібліотека JavaScript, яка дозволяє вирішувати проблему маршрутизації в React-додатках. Вона надає зручні та потужні інструменти для організації навігації між різними сторінками або компонентами в додатку. React Router DOM використовує компонентний підхід до маршрутизації, де різні URL-шляхи мають відповідні компоненти, котрі мають бути відображені [17].
- React Pro Sidebar - це бібліотека React компонентів, яка дозволяє швидко та зручно створювати бічне (sidebar) меню в React-додатках. Вона надає готові компоненти, які дозволяють відображати, приховувати та управляти бічним меню з різними функціональностями. React Pro Sidebar підтримує розкривання пунктів меню, підменю, іконки, активні пункти, зміну розміру та стилів, а також реакцію на події користувача. Ця бібліотека дозволяє швидко налаштовувати та використовувати бічне меню в додатках, забезпечуючи зручний та гнучкий інтерфейс для навігації та управління.
- MUI (Material-UI) - це популярна бібліотека React компонентів, яка надає готові інтерфейсні елементи згідно з принципами дизайну матеріального дизайну Google. MUI дозволяє розробникам швидко і легко створювати

красиві та функціональні веб-інтерфейси, використовуючи готові компоненти, такі як кнопки, поля вводу, таблиці, картки та багато інших. Бібліотека також надає потужні можливості налаштування та стилізації компонентів за допомогою тем, стилів та CSS-класів, що дозволяє відповідати вимогам проекту та забезпечувати єдиний дизайн на всій сторінці [15].

- Axios - це бібліотека JavaScript, яка дозволяє здійснювати HTTP-запити з браузера. Вона надає простий та зрозумілий інтерфейс для виконання запитів до сервера і отримання відповідей. Axios підтримує всі типи запитів, такі як GET, POST, PUT, DELETE, і багато інших, і дозволяє встановлювати заголовки та передавати дані у вигляді параметрів запиту. Вона також підтримує обробку помилок та перехоплення, асинхронні запити та інтерцептори для обробки та модифікації запитів та відповідей. Axios є популярним вибором для роботи з API в JavaScript-додатках завдяки своїй простоті використання та широкому спектру функціональності [16].
- Formik - це бібліотека JavaScript, яка спрощує управління формами в React-додатках. Вона надає потужні інструменти для створення, валідації та обробки форм, зберігаючи стан форми та автоматично оновлюючи його при взаємодії користувача. Formik дозволяє легко виконувати операції, такі як перевірка валідності, обробка подій, відправка даних на сервер та керування поведінкою форми [19].
- Yup - це бібліотека JavaScript, яка дозволяє валідувати схему даних на клієнті та на сервері. Вона забезпечує потужну та гнучку систему валідації, яка дозволяє перевіряти різні типи даних, такі як рядки, числа, дати, об'єкти, масиви та інші. Вона надає зручний та зрозумілий інтерфейс для створення та керування схемою даних.А також підтримує інтеграцію з різними бібліотеками, такими як Formik для валідації форм.
- FullCalendar - це бібліотека JavaScript для створення інтерактивних календарів у веб-додатках. Вона надає гнучкі та потужні інструменти для відображення, навігації та управління подіями в календарі. FullCalendar дозволяє відображати різні типи подій, керувати їх розміщенням та властивостями, а також додавати

взаємодію з користувачем, таку як перетягування подій, зміна тривалості та багато іншого [18].

2.4. Опис реалізації клієнтської частини вебсервісу

Як було вказано вище, розробка велась з використанням бібліотеки React та для стилізованих компонентів використовувалась бібліотека MUI. Мене не зовсім задовільняла запропонована бібліотекою кольорова гама, тому я використав можливість самої бібліотеки створити власну тему та примінити її до усього застосунку, а також реалізувати зміну темного та світлого інтерфейсу користування. Перемикати дані режими можна у всьому додатку, оскільки клавіша їх перемикання знаходиться у глобальному компоненті хедера, що забезпечує його наявність на кожній сторінці.

Пропоную розглянути реалізацію авторизації з різними кольоровими режимами та властивостями поля паролю(Див рис. 2.20 та 2.21).

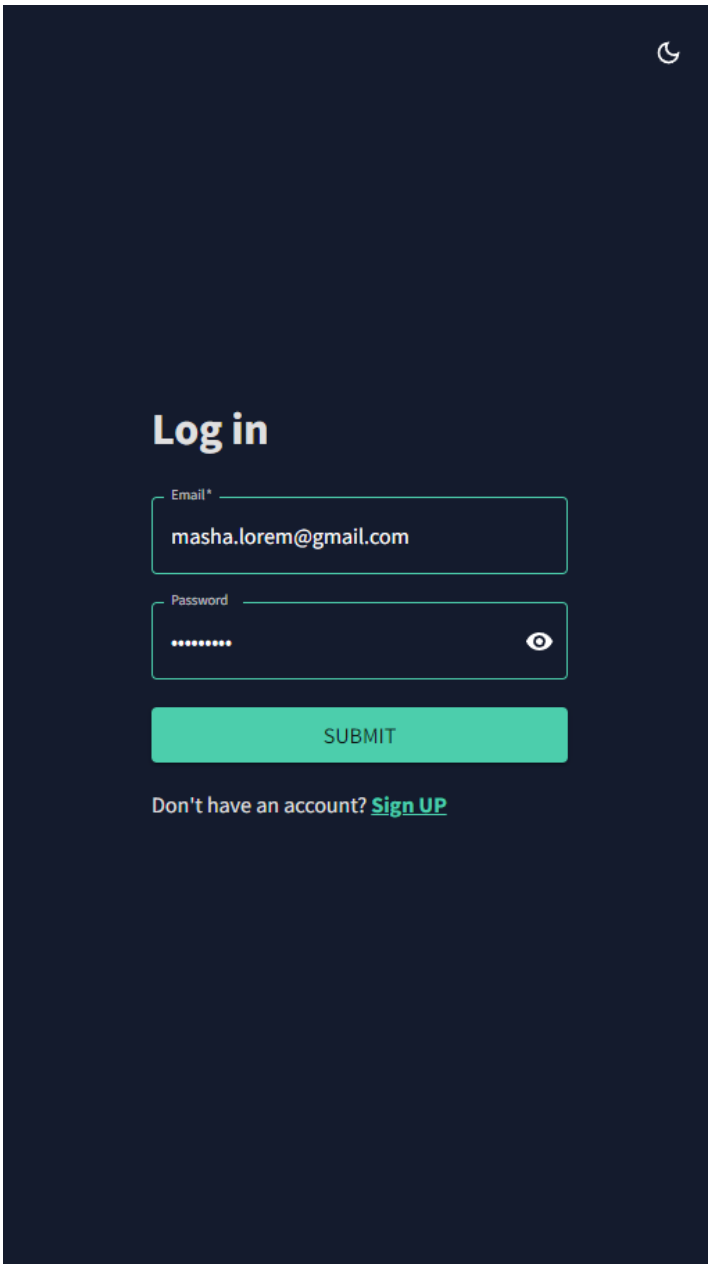

Рис. 2.20 Вигляд сторінки login з закритим паролем та темною темою

Джерело:[створено автором]

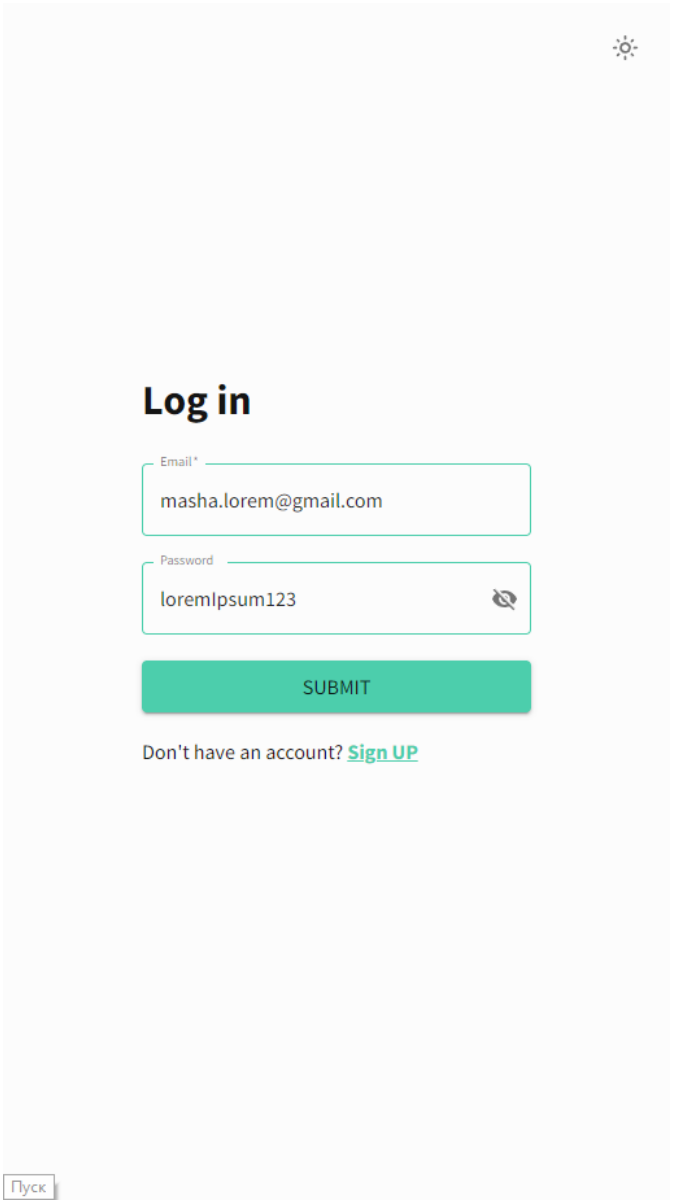

Рис. 2.21 Вигляд сторінки login з відкритим паролем та світлою темою

Джерело:[створено автором]

Для створення даної сторінки використовувались компоненти бібліотеки MUI, бібліотека axios для надсилання запиту до серверної частини, а також хуки useState та useEffect про які хочу розповісти детальніше:

- Хук useState є одним з основних хуків в React, який дозволяє створити та управляти станом компонента. Він приймає початкове значення стану та повертає масив, в якому перший елемент є поточним значенням стану, а другий елемент - функцією, яка дозволяє змінювати значення стану. За допомогою цього хука можна зберігати та оновлювати дані в компоненті, а також перерендерити компонент при зміні стану.

- Хук useEffect також є одним з основних хуків в React, який дозволяє виконувати побічні ефекти в компоненті. Він приймає функцію, яка буде виконуватися при кожному рендерінгу компонента або при зміні залежностей, які передаються як другий аргумент. Цей хук дозволяє здійснювати дії, такі як виклик API, підписка на події, маніпуляція з DOM елементами та інші операції, які потребують доступу до зовнішніх ресурсів або змінного стану.

Перший з наведених хуків використовувався для відслідковування стану полів логіну та паролю, а другий забирав текст помилки при редагуванні якогось з полів. Сам текст помилки ми отримуємо з відповіді на запит за допомогою axios(Див рис. 2.22 та 2.23).

.

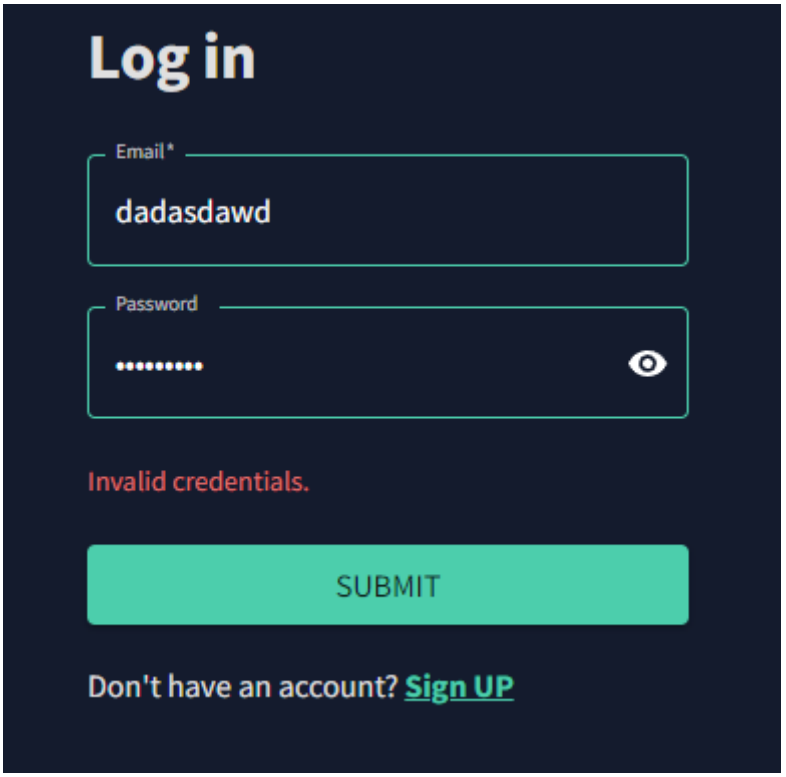

Рис. 2.22 Форма логіну після введення невірних даних

Джерело:[створено автором]

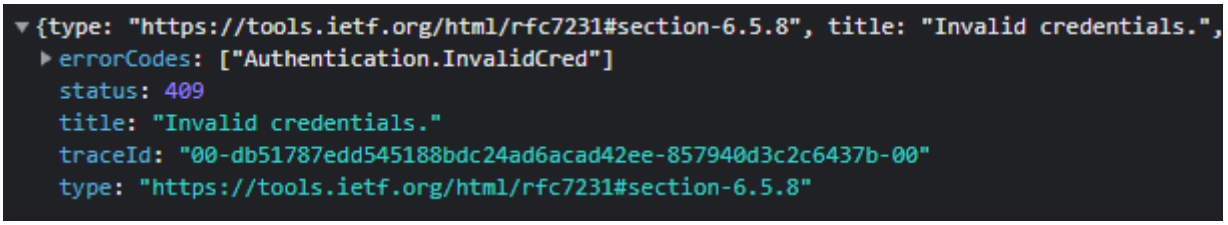

Рис. 2.23 Відповідь сервера на невірні дані

Джерело:[створено автором]

Форма реєстрації побудована подібним чином, проте ще використовує такі бібліотеки як Formik, для валідації полів, за допомогою схеми валідації(ValidationSchema), побудованій за допомогою бібліотеки yup. Помилки від серверної частини з’являються внизу форми як і в формі логіну, а помилки валідації бібліотеки Formik відображаються під кожним полем окремо, проте очевидно, що без виправлення помилок валідації, помилки від сервера не з’являться, оскільки клавіша відправки форми не буде відправляти запит до сервера. Досить зручним у валідації з використанням саме цих бібліотек, є те, що можна досить зручно вказати, що поле підтвердження паролю має таке ж значення як поле паролю всього однією строкою (Див рис. 2.24 та 2.25).

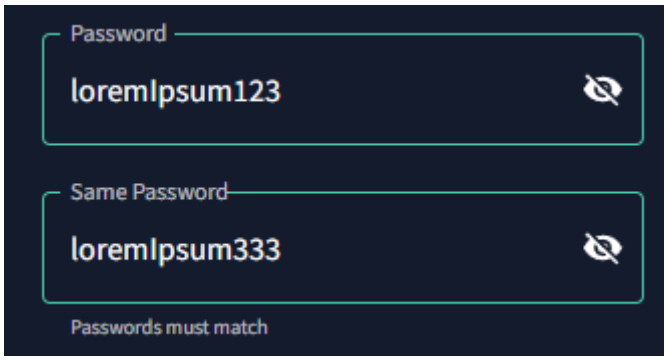

Рис. 2.24 Валідація вказує на необхідність ідентичності паролів

Джерело:[створено автором]

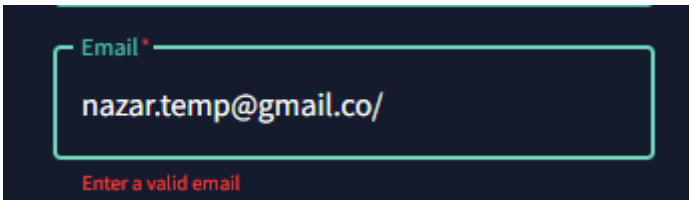

Рис. 2.24 Валідація вказує на неправильно вказану електронну пошту

Джерело:[створено автором]

Наступним до розгляду пропоную використання бокової панелі, у якій зображені основний функціонал відповідно до ролі користувача. Реалізована вона за допомогою бібліотеки React Pro Sidebar, завдяки ній вдалось набагато швидше побудувати бічне меню(Див рис. 2.26).

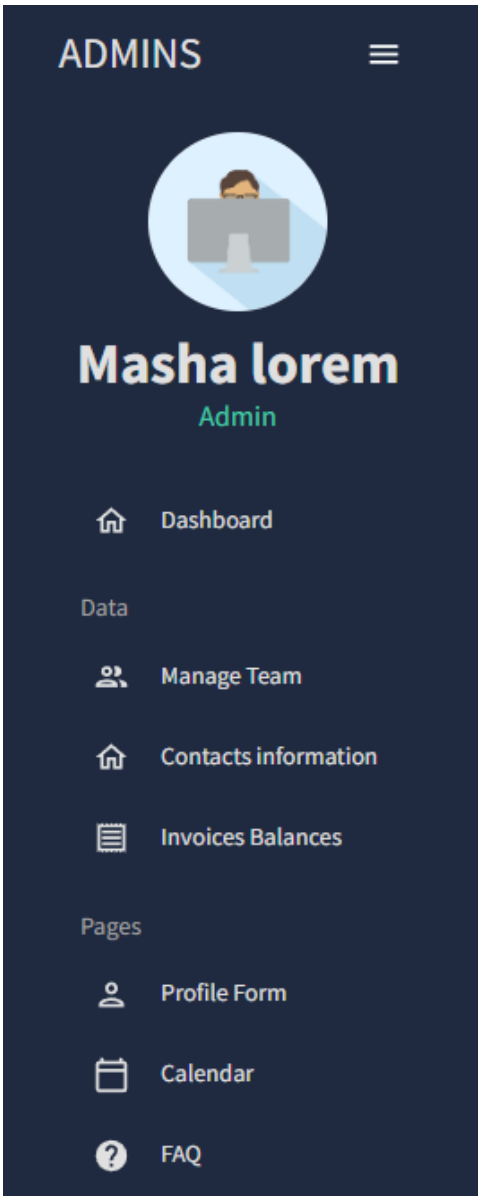

Рис. 2.26 Меню адміністратора

Джерело:[створено автором]

Однією з головних сцен є календар, на якому зображено відвідування клієнтів для кожного з тренерів, де вони можуть планувати нову сесію, або відмічати ту, яка вже відбулась. Побудована сама таблиця за допомогою бібліотеки FullCalendar в яку можна довантажувати плагіни:

- dayGridPlugin - записи у вигляді на день, тиждень, місяць (Див рис. 2.27)
- listPlugin - відображення записів у лісті(Див рис. 2.28)

Рис. 2.27 Відображення списку записів певного тренера на тиждень

Джерело:[створено автором]

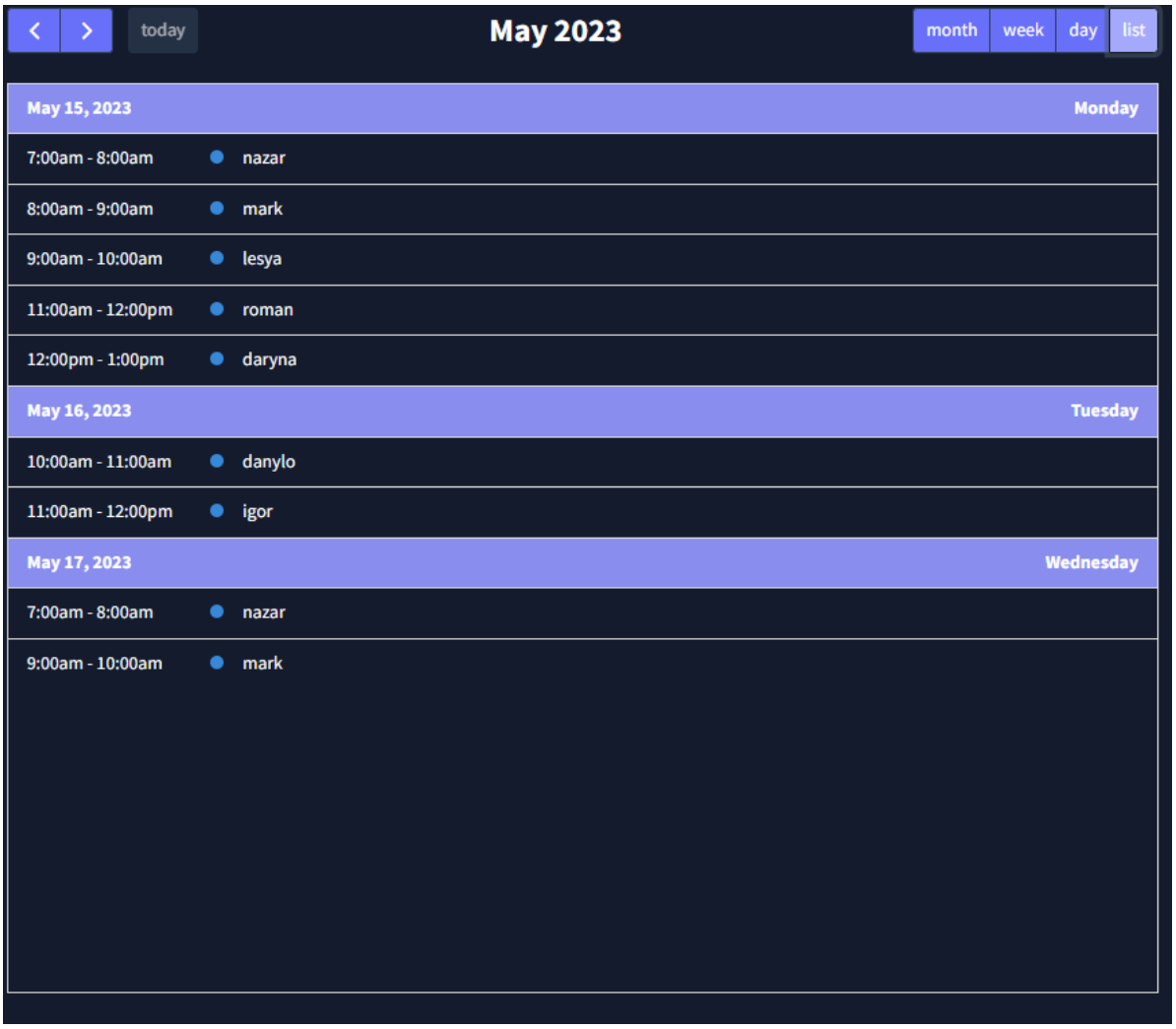

Рис. 2.28 Відображення списку записів певного тренера

Джерело:[створено автором]

На даних прикладах було розглянуто основні можливості клієнтської частини програмного продукту. При розробці клієнтської частини веб-сервісу були використані сучасні технології та бібліотеки, що сприяли створенню функціонального та зручного користувальницького інтерфейсу Для розробки клієнтської частини була використана бібліотека React, яка дозволяє ефективно будувати інтерфейси з використанням компонентного підходу. Використання React спрощує розробку, реорганізацію та підтримку коду, а також забезпечує швидку та плавну відображення змін у стані додатку.

Для стилізації та забезпечення єдиної тематики була використана бібліотека Material UI. Вона надає набір готових компонентів та стилів, що дозволяє швидко створювати красиві та сучасні інтерфейси. Використання Material UI спрощує

розробку та підтримку дизайну і забезпечує єдність вигляду елементів інтерфейсу. Для маршрутизації між сторінками використовувалася бібліотека React Router DOM. Вона дозволяє зручно керувати маршрутами та навігацією в додатку, забезпечуючи перехід між сторінками без перезавантаження всього додатку. Для валідації форм були використані бібліотеки Formik та Yup. Formik дозволяє зручно управляти станом форм та виконувати валідацію даних. Yup надає потужні та гнучкі правила валідації, що дозволяє легко перевіряти правильність введених даних. Для взаємодії з серверною частиною була використана бібліотека Axios. Вона надає зручний інтерфейс для виконання HTTP-запитів, що дозволяє отримувати та відправляти дані між клієнтом та сервером. Використання Axios спрощує роботу зі зверненнями до API та обробку отриманих даних.

Загалом, реалізація клієнтської частини веб-сервісу з використанням React, Material UI, React Router DOM, Formik, Yup та Axios дозволила створити зручний та ефективний інтерфейс. Ці технології та бібліотеки сприяють ефективній розробці і підтримці додатку, а також забезпечують його стабільність та високу якість.

ВИСНОВКИ

Впродовж виконання даної кваліфікаційної роботи було розроблено систему управління реабілітаційним центром кінезітерапії "Осередок здоров'я". Перед розробкою програмного рішення було проведено аналіз конкурентів на ринку, що дозволяє зрозуміти особливості та вимоги до функціоналу системи. На основі результатів аналізу були виділені основні вимоги, які стали фундаментом розробки.

Для забезпечення чіткого розуміння та моделювання процесів використання системи були створені use-case діаграми. Ці діаграми допомогли визначити ролі та можливості кожного користувача системи, а також виявити потенційні невизначеності та помилки у логіці бізнес-процесів. Вони послужили основою для подальшої розробки функціональності та інтерфейсу системи.

У результаті розробки було створено систему управління реабілітаційним центром кінезітерапії "Осередок здоров'я", яка відповідає основним вимогам та потребам ринку. Вона дозволяє зручно та ефективно керувати процесами реабілітації. Результатом роботи є функціональна та надійна система, яка сприяє оптимізації роботи реабілітаційного центру та покращує якість надання послуг. Використання аналізу конкурентів, use-case діаграм та відповідних технологій розробки допомогли досягти поставленої мети.

Після визначення потреб та мети розробки було виділено основні сутності та побудувно на їх базі БД, цей процес був розділений на кілька етапів, а саме: створення схеми, побудова схеми за допомогою принципу Code First, роботу котрого забезпечує Entity Framework, тестування, для впевненості, що немає проблем у роботі БД, шляхом заповнення її тестовими даними. У результаті виконання даного етапу роботи я отримав ясне бачення вимог до програмного продукту та готову базу даних, котра відповідає поставленим вимогам.

Наступним етапом розробки була побудова архітектури, адже лише після здійснення аналізу вимог до програмного продукту можна побудувати її таким чином, щоб забезпечити максимальну продуктивність, що означає мінімальну затримку при роботі та розширюваність, що дозволить при потребі продовжувати

розробку продукту у майбутньому. В даному програмному рішенні було використано один з найкращих варіантів архітектур – Clean Architecture, з імплементованим в себе CQRS патерном за допомогою бібліотеки Mediatr котрий дозволяє розділити різні шари, котрі мають свої зони відповідальності, це забезпечує дуже гнучку розширюваність.

Маючи стійку основу проекту було проведено роботу над рівнями доступу до БД, який було забезпечено за допомогою CQRS патерном та репозиторієм, ці патерни доповнюючи один одного змогли чітко розподілити та реалізувати функціонал API. Для забезпечення зв’язку між даними шарами використовувались шари Shared та Contracts. Інтерфейси методів та їх імплементація розподіляються між шарами Application та Infrastructure відповідно. Було реалізовано авторизацію та автентифікацію за допомогою JWT токену для забезпечення безпеки.

Після побудови API було проведено роботу над клієнтською частиною, у процесі розробки якої було використано низку потужних інструментів та бібліотек, що значно полегшило роботу та покращило якість кінцевого продукту.

Використання бібліотеки React у поєднанні з Material UI надало безліч переваг для створення користувацького інтерфейсу. React забезпечив організацію компонентів в зручну структуру, що сприяло модульності та повторному використанню коду. Material UI надав готові компоненти та стилі, що спростило розробку та забезпечило єдиний та привабливий зовнішній вигляд додатку.

Для забезпечення маршрутизації між різними сторінками використовувалась бібліотека React Router DOM. Вона дозволила зручно визначати шляхи та перехід між сторінками веб-сервісу, забезпечуючи зручну навігацію та взаємодію з користувачем. Для валідації форм було використано бібліотеки Formik та Yup. Це дозволило легко та зручно валідувати дані, введені користувачами у форми. Використання цих бібліотек забезпечило надійну обробку даних та запобігло можливим помилкам, забезпечуючи коректну взаємодію з користувачем. Виконання запитів до серверу реалізувала бібліотека Axios. Її простий та зручний інтерфейс дозволив легко взаємодіяти з API, здійснювати HTTP-запити, обробляти результати

та керувати станом даних веб-сервісу. Використання Axios спростило роботу з мережевими запитами та забезпечило ефективну комунікацію з сервером.

Загалом, використання React, Material UI, React Router DOM, Formik, Yup та Axios забезпечило зручність та ефективність у розробці клієнтської частини веб-сервісу. Ці бібліотеки сприяли модульності, продуктивності та надійності додатку, забезпечуючи зручний та привабливий користувацький досвід.

В майбутньому планується розширювати функціонал, більше детально тестувати та виправляти помилки, після чого розмістити проєкт на хостингу та протестувати його в реальних робочих умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A tour of the C# URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. (дата звернення: 12.02.2023 р.).
2. What Is Clean Architecture URL: https://www.c-sharpcorner.com/article/what-is-clean-architecture/. (дата звернення: 01.03.2023 р.).
3. EF Core URL: https://learn.microsoft.com/en-us/ef/core/. (дата звернення: 07.03.2023 р.). (дата звернення: 20.03.2023 р.).
4. ASP.NET Core URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0. (дата звернення: 08.03.2023 р.).
5. Get started with Swashbuckle and ASP.NET Core URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-6.0&tabs=visual-studio. (дата звернення: 2303.2023 р.).
6. Mohamad Lawand. .NET 6 - Web API Global Exceptions Handling URL:https://www.youtube.com/watch?v=fBXOgrZ3ZC4&ab_channel=MohamadLawand. (дата звернення: 12.03.2023 р.).
7. Mohamad Lawand. .NET 6 - Web API Create JSON Web Tokens (JWT) URL:https://www.youtube.com/watch?v=Y-MjCw6thao&ab_channel=MohamadLawand. (дата звернення: 13.03.2023 р.).
8. Patrick God. .NET 6 Web API & Entity Framework Core Jump Start URL:https://www.youtube.com/watch?v=K23uJdMiEpk&ab_channel=PatrickGod. (дата звернення: 17.03.2023 р.).
9. IAmTimCorey. Entity Framework Best Practices - Should EFCore Be Your Data Access of Choice

URL:https://www.youtube.com/watch?v=qkJ9keBmQWo&ab_channel=IAmTimCorey. (дата звернення: 17.03.2023 р.).

10. Amichai Mantinband. CQRS & MediatR | ASP.NET 6 REST API Following CLEAN ARCHITECTURE URL: https://www.youtube.com/watch?v=MwMVvLBSJa8. (дата звернення: 22.03.2023 р.).

11. Amichai Mantinband. Object Mapping - Mapster | ASP.NET 6 REST API Following CLEAN ARCHITECTURE URL: https://www.youtube.com/watch?v=vBs6naPD6RE. (дата звернення: 22.03.2023 р.).

12. Amichai Mantinband. Global Error Handling | ASP.NET 6 REST API Following CLEAN ARCHITECTURE URL: https://www.youtube.com/watch?v=gMwAhKddHYQ. (дата звернення: 27.03.2023 р.).

13. Repository Pattern | ASP.NET 6 REST API Following CLEAN ARCHITECTURE URL: https://www.youtube.com/watch?v=ZwQf_JQUUCQ. (дата звернення: 28.03.2023 р.).

14. React Docs URL:https://react.dev/. (дата звернення: 01.04.2023 р.).

15. Material UI URL:https://mui.com/material-ui/. (дата звернення: 04.04.2023 р.).

16. Dave Grey. React User Login and Authentication with Axios URL:https://www.youtube.com/watch?v=oUZjO00NkhY&t=783s&ab_channel=DaveGray. (дата звернення: 14.04.2023 р.).

17. Dave Grey. React Protected Routes | Role-Based Authorization | React Router v6 URL: https://www.youtube.com/watch?v=X3qyxo_UTR4&ab_channel=DaveGray. (дата звернення: 12.05.2023 р.).

18. EdRoh. Build a COMPLETE React Admin Dashboard App | React, Material UI, Data Grid, Light & Dark Mode URL:

https://www.youtube.com/watch?v=wYpCWwD1oz0&t=9747s&ab_channel=EdRoh. (дата звернення: 22.05.2023 р.).

19. Formik docs URL: https://formik.org/docs/examples/with-material-ui. (дата звернення: 11.05.2023 р.).