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

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка серверної частини сервісу для продажів навчальних подій»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Нечипорук Андрій Михайлович

Керівник: ***Клебан Ю.В., старший викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса* ***Загоруйко Любов Василівна***

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

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
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента
Нечипорука Андрія Михайловича

1. Тема роботи: ***Проєктування та розробка серверної частини сервісу для продажів навчальних подій***.

Керівник роботи: Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: Microsoft Visual Studio, Postman, GitHub, SQL Server Management Studio, Google Documents.

4. Перелік завдань, які належить виконати: дослідити сучасні рішення архітектури проєктів на основі технології ASP.NET WEB API; проаналізувати підходи до побудов реляційної бази даних; спроектувати базу даних, яка буде покривати вимоги проекту; підібрати актуальні бібліотеки для застосування під час розробки; здійснити дослідження на наявність конкурентів; дослідити предметну область; виконати безпосередню розробку проєкту та оптимізувати його; підготувати інтерфейс для взаємодії з клієнтом.

5. Перелік графічного матеріалу: відображення досліджень та результатів, отриманих під час написання кваліфікаційної роботи у вигляді рисунків.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В.	01.12.2023	01.12.2023
2	Клебан Ю. В.	01.12.2023	01.12.2023
3	Клебан Ю. В.	01.12.2023	01.12.2023

7. Дата видачі завдання:

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи	до 31.10.2023 р.	
2	Створення технічного завдання	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 10.12.2023 р.	
4	Написання розділу 1	до 01.02.2024 р.	
5	Написання розділу 2	до 01.03.2024 р.	
6	Написання розділу 3	до 01.04.2024 р.	
7	Тестування серверної частини	до 20.04.2024 р.	
8	Виправлення помилок	до 01.05.2024 р.	
9	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи/проєкту	до 31.05.2024 р.	
10	Здача кваліфікаційної роботи на кафедру	31.05.2024 р.	

Студент: _____ Андрій НЕЧИПОРУК

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Проєктування та розробка серверної частини сервісу для продажів навчальних подій*

Автор: *Нечипорук Андрій Михайлович*

Науковий керівник: *Клебан Ю.В., старший викладач кафедри ЕММІТ*

Захищена «……»…… 2024 року.

Пояснювальна записка до кваліфікаційної роботи: *75 с., 25 рис., 5 додатків, 27 джерел.*

Ключові слова: *навчання, платформа, веб-застосунок, серверна частина, база даних, події, ASP.NET*

Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту було проєктування та розробка серверної частини сервісу для продажів навчальних подій, яку в подальшому можна буде використовувати для інтеграції з клієнтською частиною. Цей проєкт містить в собі функціонал для перегляду та фільтрації подій, керування всіма можливими процесами, пов'язаними з ними, а також реєстрації на саму подію. Функціонал серверної частини веб-додатку передбачає два типи користувачів: інструктори, які можуть продавати навчальні події, а також учні, які можуть реєструватися на ці події та отримувати навчальні матеріали.

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of the client part of the educational platform*

Author: *Andrii Nechyporuk*

Scientific supervisor: *Kleban Y., Senior Lecturer at DEMMIT*

Defensed «……»…… of 2024.

Explanatory note to the qualification work: *75 p., 25 pic., 5 attachments, 27 sources.*

Keywords: *training, platform, web application, backend, database, events, ASP.NET*

Summary of the paper:

The purpose of the qualification work/project was to design and develop the server side of the service for selling educational events, which can be used to integrate with the client side of the application in the future. This project includes functionality for viewing and filtering events, managing all possible processes related to them, and registering for the event itself. The functionality of the server side of the web application has two types of users: instructors who can sell training events, as well as students who can register for these events and receive training materials.

ЗМІСТ

ВСТУП

У сучасному світі присутній стрімкий розвиток інформаційних технологій, а їх споживачі дедалі більше вибирають цифрове середовище для отримання різноманітних послуг, освіта не залишається осторонь цього трансформаційного процесу. Сучасна освітня система стикається з викликами глобалізації, розширенням доступу до знань та потребою у наданні якісних інструментів для навчання в онлайн-форматі.

Особливо вразливою стала сфера навчання та тренінгів, де інноваційні підходи можуть змінити традиційні моделі. Велика кількість освітніх ініціатив та подій потребують ефективної та зручної платформи для організації та продажу. З цієї точки зору, роль серверної частини сервісу для продажів навчальних подій стає надзвичайно актуальною.

Навчальні платформи та онлайн-сервіси стають справжнім мостом між освітніми ініціативами та їх аудиторією, дозволяючи забезпечити гнучкість у навчанні, збільшити доступність освіти та відкрити нові можливості для учнів усіх вікових груп. Висока конкуренція у цьому сегменті створює необхідність для розробки інноваційних рішень, що дозволяють виділитися серед інших та задовольняти високі вимоги користувачів.

Більшість рішень, які присутні на ринку, орієнтовані на продаж та поширення власних продуктів, або пропонують комплексні навчальні курси, не беручи до уваги сегмент курсів, які пропонують відносно невеликий об’єм інформації. Таким чином це ускладнює можливість поділитися досвідом людям, котрі мають значні здобутки знань у певній сфері, та ще не здобули популярності безпосередньо як ті, що пропонують послуги у сфері освіти.

Актуальність цієї теми полягає в необхідності проєктування та розробки бекенду додатку для продажу навчальних подій, яка буде підкріплена попередніми дослідженнями та врахує достатньо аспектів для задоволення потреб майбутнього користувача.

Метою даного дослідження є створення серверної частини для продажу навчальних подій. Він має не лише врахувати сучасні технологічні можливості, а й надасть майбутнім користувачам необхідний функціонал для можливості купівлі доступу до подій та курсів у сфері освіти та навчання.

Об’єктом дослідження є сервіс для продажу навчальних подій, а предметом дослідження є інформаційні технології з проєктування та розробки серверної частини сервісу для продажу навчальних подій.

Для досягнення поставленої мети було встановлено наступні завдання, які слугуватимуть планом виконання цієї роботи:

1. Виконати огляд наявних конкурентів на ринку та здійснити аналіз їхніх продуктів.
2. Побудова UML діаграми, яка відобразить модель функціональності серверної частини.
3. Проєктування бази даних, яка забезпечить надійне та зручне зберігання внутрішньої інформації веб-додатку.
4. Знаходження архітектурного рішення проєкту на основі кращих практик проєктування на фреймворку ASP.NET WEB API.
5. Обрати технічний стек для написання бекенду. Знайти сучасні бібліотеки та використати їх з метою розширення функціоналу.
6. Здійснення розробки функціоналу для серверної частини веб-додатку

Для виконання досліджень було використано такі методи: порівняльний аналіз, моделювання діаграм варіантів використання, аналіз літератури та джерел, проєктування бази даних за допомогою реляційного підходу, а також ручне тестування для перевірки коректності роботи компонентів системи.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ПРОЄКТУВАННЯ ТА РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ ДЛЯ ПРОДАЖІВ НАВЧАЛЬНИХ ПОДІЙ

1.1. Опис предметного середовища

Jet Study Project - це веб-додаток, спрямований на організацію та зручне проведення навчальних подій різного формату, таких як курси, вебінари, майстер класи та інші. Платформа дозволяє студентам зареєструватися на подію та отримати доступ до необхідних матеріалів у зручному та інтуітивно зрозумілому інтерфейсі.

Функціональною моделлю проєкту JetStudy є:

1. Реєстрація та авторизація користувачів:
    - студенти та інструктори можуть створити облікові записи на платформі;
    - вони можуть увійти за допомогою email та пароля або через сторонні сервіси, такі як Google та інші;
2. Підвищення користувачів до рівня інструктора:
    - користувач сервісу, який бажає стати інструктором, може подати свою заяву, яку розгляне адміністратор;
    - в свою чергу, адміністратор може прийняти або відхилити наявну заяву;
    - після схвалення заявки, відповідному клієнту надаються права, з якими він може здійснювати дії, доступні для інструктора;
3. Створення та редагування подій:
    - інструктори можуть створювати нові навчальні події на платформі, вказуючи необхідну інформацію, таку як назва, дата, час, тривалість, програма, вартість тощо;
    - вони можуть також редагувати або видаляти існуючі події;
4. Модерація подій адміністратором

- після створення чернетки події, її потрібно надіслати на модерацію;
- адміністрація додатку може опублікувати подію, або відправити назад на доопрацювання у разі потреби;

5. Пошук та фільтрація подій:
    - користувачі можуть шукати події за різними критеріями, такими як назва, напрямок, формат, мова тощо;
    - вони можуть також фільтрувати події за статусом, наприклад, майбутні, активні або завершені;
6. Реєстрація на подію та управління учасниками:
    - студенти можуть зареєструватися на обрану подію, вибравши необхідну кількість місць;
    - інструктори можуть переглядати та модерувати список учасників, особливо у випадку, якщо кількість місць обмежена;

Процесом діяльності проєкту є:

1. Створення події:
    - інструктор створює нову навчальну подію, заповнюючи необхідну інформацію про неї;
    - він також додає матеріали та ресурси, необхідні для проведення події;
2. Реєстрація на подію:
    - студент реєструється на обрану подію, обираючи кількість місць та здійснюючи оплату, якщо це потрібно;
3. Проведення події:
    - на день проведення події студенти отримують доступ до необхідних матеріалів та посилань для участі в ній;
    - інструктори проводять подію, надаючи учасникам необхідну інформацію та навчальний матеріал.

1.2. Огляд наявних аналогів

Огляд наявних на ринку платформ необхідно здійснювати з багатьох важливих причин, а саме: для визначення слабких та сильних сторін конкурентів, розуміння стандартів якості, знаходженню інноваційних рішень, розуміння трендів, формування списку потреб користувачів. Для знаходження основних переваг та недоліків, тобто здійснення огляду, було обрано 3 освітніх платформи: Udemy, Coursera, Skillshare.

1. Udemy

Udemy – це одна з найбільших і найбільш популярних онлайн-платформ для навчання та продажу освітніх курсів. На Udemy можна знайти тисячі курсів з різних предметів, від програмування та маркетингу до музики та мистецтва. Платформа пропонує можливість як вивчення власним темпом, так і участі в інтерактивних відеоуроках. Інструктори можуть виставляти свої курси на продаж та заробляти на цьому, а користувачі мають доступ до широкого спектру освітніх ресурсів. Переглянути типовий дизайн сторінки з курсами можна на рисунку 1.1.

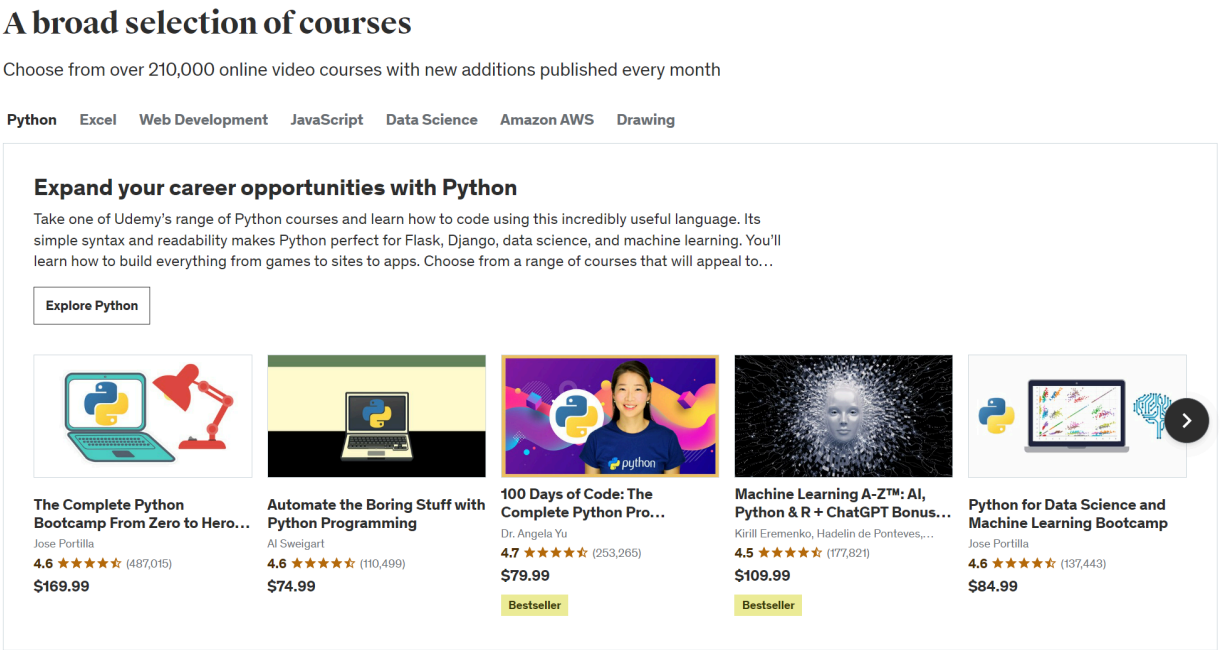

Рис. 1.1. Інтерфейс курсів на сайті Udemy [1]

Переваги:

Udemy має широкий вибір курсів з різних областей, що дозволяє користувачам знаходити навчання в практично будь-якій сфері. Багато курсів на Udemy доступні за помірними цінами, а деякі навіть можуть бути безкоштовними, що робить освіту доступною для широкого кола користувачів. На платформі викладають експерти з різних галузей, що гарантує якість навчання.

Недоліки:

Якість курсів може значно варіюватися, оскільки вони створюються різними інструкторами.

2. Coursera

Coursera – це платформа, яка співпрацює з університетами та організаціями з усього світу для надання онлайн-курсів. На відміну від деяких інших платформ, Coursera пропонує віртуальні уроки, розроблені фахівцями у галузі та провідними університетами. Курси на Coursera можуть бути безкоштовними або платними, а також можуть призводити до отримання сертифікатів або навіть онлайн-ступенів. Переглянути дизайн карток для продажу курсів на цьому сайті можна на рисунку 1.2.

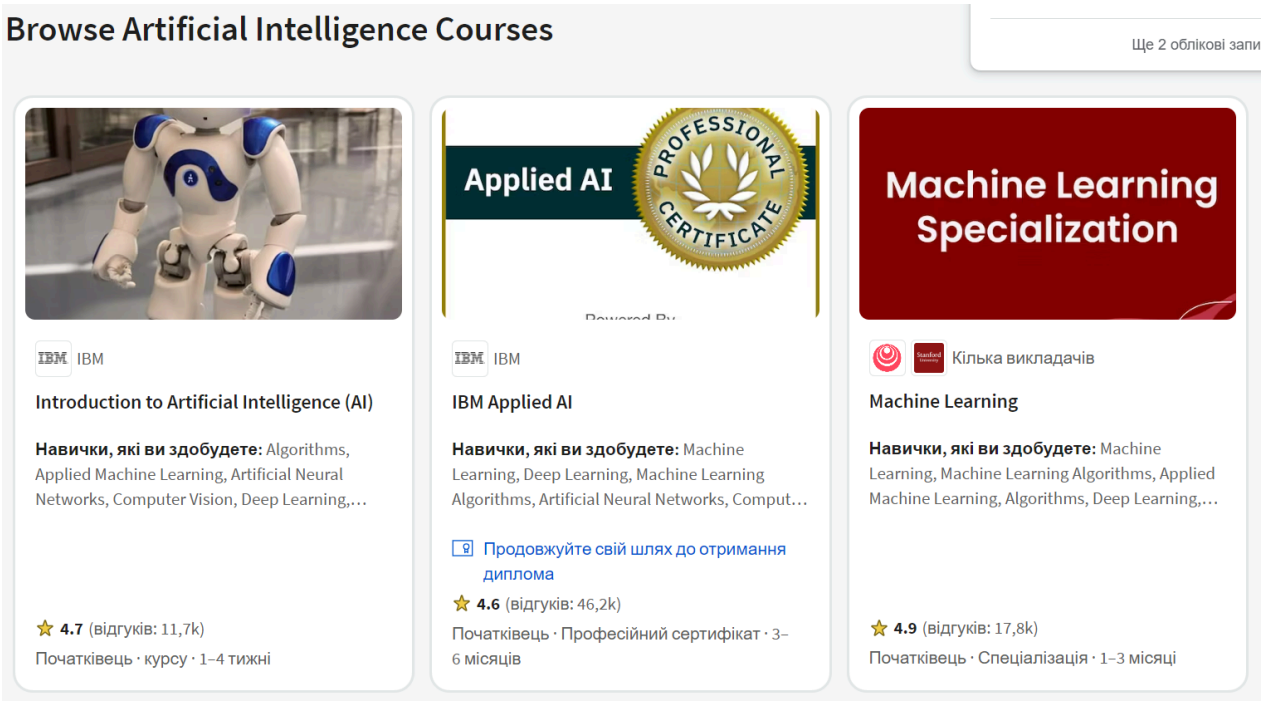

Рис. 1.2. Інтерфейс курсів на сайті Coursera [2]

Переваги:

Coursera співпрацює з університетами та надає можливість отримати офіційні сертифікати та навіть онлайн-ступені від світових університетів. Платформа пропонує курси з різних областей, включаючи технічні та гуманітарні науки. Відеоконтент на Coursera в переважній більшості виглядає професійно і структуровано.

Недоліки:

Деякі курси та спеціалізації можуть мати велику вартість, що робить їх менш доступними для деяких користувачів. Курси мають конкретний графік, що може бути не зручним для тих, хто шукає навчання з вільним графіком.

3. Skillshare

Skillshare – це платформа, яка спрямована на творчі та практичні навички. Вона пропонує короткі відеоуроки, які називаються "класами", з таких областей, як дизайн, фотографія, письмо, музика та багато інших. Skillshare пропонує модель підписки, яка дає користувачам необмежений доступ до всього контенту на платформі. Інструктори можуть отримувати прибуток від підписок та винагород за кожен перегляд їхніх уроків.

Ці рішення відображають різноманіття форматів та підходів до продажу освітніх послуг, враховуючи різні потреби користувачів і варіанти взаємодії з матеріалами навчання. Кожен із них має свої переваги та особливості, враховуючи як широкий спектр предметів, так і різні стилі навчання. Вигляд даної платформи можна спостерігати на рисунку 1.3.

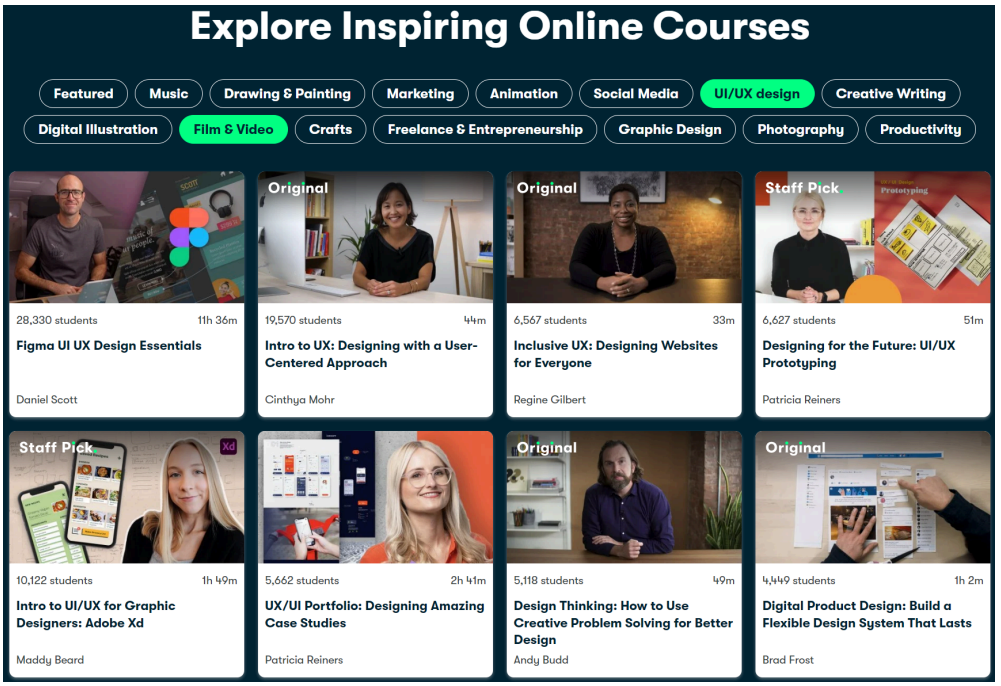

Рис. 1.3. Інтерфейс курсів на сайті Skillshare

Переваги:

Skillshare спрямований на творчі навички, і він часто пропонує практичні вправи та завдання для студентів. Система підписки дозволяє користувачам необмежено користуватися всім контентом, що вкрай вигідно для тих, хто активно вивчає кілька пов’язаних сфер одночасно. Уроки часто коротші та легше засвоюються, що підходить для тих, хто шукає конкретні навички.

Недоліки:

Уроки на Skillshare часто не призводять до офіційних сертифікатів чи ступенів. Платформа фокусується переважно на творчих галузях, що може викликати нестачу ресурсів для тих, хто шукає більш технічні або академічні курси.

Підсумок огляду

Підсумовуючи огляд платформ Udemy, Coursera і Skillshare, можна виділити декілька ключових аспектів, які важливо врахувати при розробці майбутнього додатку для продажу навчальних подій:

1. Важливо враховувати широкий вибір курсів та високу якість навчального контенту, щоб привертати різний аудиторій і забезпечити задоволення від навчання;

2. Співпраця з університетами та отримання офіційних сертифікатів відзначають додаток як високоякісний та забезпечують йому конкурентні переваги;

3. Забезпечення гнучкості графіку та доступності для користувачів допомагає привертати тих, хто шукає навчання в своєму темпі та режимі;

4. Врахування різних можливостей ціноутворення, таких як підписки, покупка окремих курсів чи сертифікатів, дозволяє привертати різні клієнтські сегменти.

5. Запрошення експертів та фахівців для викладання курсів додає авторитету та забезпечує високий рівень експертизи на платформі.

6. Додавання можливості для інтерактивних завдань та практичних вправ підвищує ефективність навчання та взаємодію з користувачами.

7. Розгляд можливостей вбудованих маркетингових інструментів та засобів комунікації для інструкторів і студентів для створення активної та залученої спільноти.

Успішний додаток має збалансувати ці аспекти, враховуючи специфіку цільової аудиторії та особливості ринку освітніх послуг. При цьому, наголошуючи на вищезазначених ключових елементах, додаток матиме шанс виграти у конкуренції та привертати користувачів, які цінують якість та інновації в онлайн-навчанні.

1.3. Створення технічного завдання

При розробці серверної частини на основі фреймворку для розробки ASP.NET Web API ми маємо на меті реалізувати наступний функціонал, який відповідає потребам користувачів і особливостям ринку освітніх послуг:

1. Реалізація можливості реєстрації нового користувача на платформі, зберігання його особистих даних та автоматичне створення облікового запису.
2. Розробка механізму аутентифікації користувачів за допомогою логіна та пароля для отримання доступу до особистого кабінету.
3. Реалізація можливості зміни особистої інформації користувача в особистому кабінеті.

4. Розробка механізму відновлення паролю для користувачів, які забули свій пароль.
5. Реалізація функціоналу для перегляду списку майбутніх подій та отримання детальної інформації про них.
6. Створення можливості перегляду детальної інформації про окремий курс, включаючи опис, викладачів, вартість та інші параметри.
7. Реалізація можливості запису на обраний курс та збереження інформації про участь користувача в ньому.
8. Розробка механізму для відправки рахунку на електронну пошту користувача та зарахування його на обраний курс після оплати.
9. Розробка API для взаємодії з іншими сервісами для проведення маркетингових акцій та комунікації з користувачами.
10. Реалізація засобів для збору та аналізу даних про активність користувачів на платформі з метою оптимізації пропозиції курсів та підвищення їх ефективності.

Висновки до розділу 1

Зважаючи на аналіз конкурентів та загальні принципи успішної платформи для продажу освітніх послуг, ми виявили кілька ключових аспектів, які необхідно врахувати при розробці нашого додатку.

По-перше, це розмаїття та якість контенту. Наша платформа повинна пропонувати широкий вибір курсів з високоякісним навчальним матеріалом. Так як публікатором подій може стати кожен, хто має цінний досвід, яким може поділитися, то проблеми у вирішенні цього питання не повинно бути.

По-друге, важлива гнучкість та зручність для користувачів, адже користувачам необхідно мати можливість навчатися власним темпом та зручним для них способом. Крім того, ми прагнемо створити інтерактивні можливості для навчання та практичні завдання, що підвищить ефективність процесу вивчення. Не менш

важливим є розробка ефективних маркетингових інструментів та засобів комунікації для привертання користувачів та підтримки активної спільноти.

На основі аналізу та постановки задач у розділі 1 встановлено, що врахування таких елементів, як різноманіття контенту, співпраця з університетами, гнучкий графік, правильна модель ціноутворення, а також інтерактивні завдання та практичні вправи є основними інструментами, які забезпечать ефективність платформи. Дослідження в цьому розділі чітко визначили напрямок розвитку проєкту та визначили основні вимоги до функціоналу. На цьому етапі ми створили фундаментальну базу для подальшої роботи над проектом, щоб забезпечити його успішну імплементацію та задоволення потреб користувачів.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ФУНКЦІОНАЛЬНОЇ МОДЕЛІ СЕРВЕРНОЇ ЧАСТИНИ

2.1. Створення UML діаграми для серверної частини

UML (Unified Modeling Language) - це загальноприйнята стандартизована мова, основною ціллю для використання якої є візуальне відображення моделі програмної системи. Завдяки їй розробник може використовувати для побудови моделі програмного продукту різноманітні графічні елементи та позначення, які відображатимуть його структуру та можливу поведінку у різних випадках використання. [4]

Цей інструмент є одним з обов’язкових інструментів розробника, оскільки він допомагає виконати наступні цілі:

1. Краще зрозуміти систему. UML під час створення системи використовує елементи візуальної складової, які полегшують розуміння архітектури та логіки роботи продукту.
2. Забезпечує комунікацію з командою. Стандартизовані набори позначень та діаграм є зрозумілими для всіх учасників команд, завдяки чому стають єдиними ідея та концепція проєкту.
3. Здійснити аналіз та спроєктувати продукт правильно. Використання цієї мови моделювання на ранніх стадіях розробки дозволяє уникнути критичних помилок та наблизити архітектуру до стану, в якому вона буде готова до використання в реальних умовах.

Існують наступні типи UML діаграм:

1. Діаграма класів(class diagram). Діаграма класів - це, мабуть, найпоширеніша діаграма UML. Вони представляють структуру системи, показуючи її класи, їх властивості, методи та взаємозв'язки між класами. Діаграми класів є статичними за своєю природою і надають схему структури коду системи. [5]

2. Діаграма прецедентів(use case diagram). Діаграма варіантів використання описує функціональність системи з точки зору користувача. Вони ілюструють різні способи взаємодії користувача з системою та різні шляхи, якими користувач може досягти певної мети. Діаграма варіантів використання складається з акторів (користувачів або зовнішніх систем) і варіантів використання (дій або послуг, що надаються системою).
3. Діаграма послідовності(sequence diagram). Діаграми послідовності представляють взаємодію між об'єктами або компонентами всередині системи в часі. Вони показують послідовність повідомлень, якими обмінюються об'єкти або компоненти у відповідь на зовнішні стимули.
4. Діаграми діяльності(activity diagram). Діаграми діяльності відображають потік управління в системі, вони представляють послідовність дій або операцій, які відбуваються в рамках процесу, а також точки прийняття рішень, паралельні дії та логіку розгалуження. Діаграми діяльності корисні для моделювання бізнес-процесів, робочих процесів програмного забезпечення та поведінки системи.
5. Діаграма станів(state machine diagram). Діаграма станів ілюструє різні випадки, в яких може перебувати об'єкт або система, і переходи між цими станами у відповідь на події. Вони особливо підходять для моделювання поведінки реактивних систем або об'єктів, які демонструють різну поведінку залежно від їхнього внутрішнього стану.
6. Компонентна діаграма(component diagram). Діаграми компонентів представляють фізичні компоненти системи та залежності між ними. Вони показують високорівневу структуру системи через багаторазові програмні компоненти, бібліотеки, модулі та виконувані файли, а також їхні зв'язки та залежності.
7. Діаграма розгортання(deployment diagram). Діаграма розгортання ілюструє фізичне розгортання програмних компонентів на апаратних вузлах або обчислювальних пристроях. Вони показують, як програмні артефакти, такі як

виконувані файли, бібліотеки та конфігураційні файли, розподіляються між різними апаратними вузлами або серверами в мережевому середовищі.

Ми вирішили використати другий тип для побудови концепції майбутньої системи, а саме діаграму прецедентів. Вона включає в себе такі 4 об’єкти: актор, варіант використання, зв’язки, межа системи. Їхній вигляд зображено на рисунку 2.1.

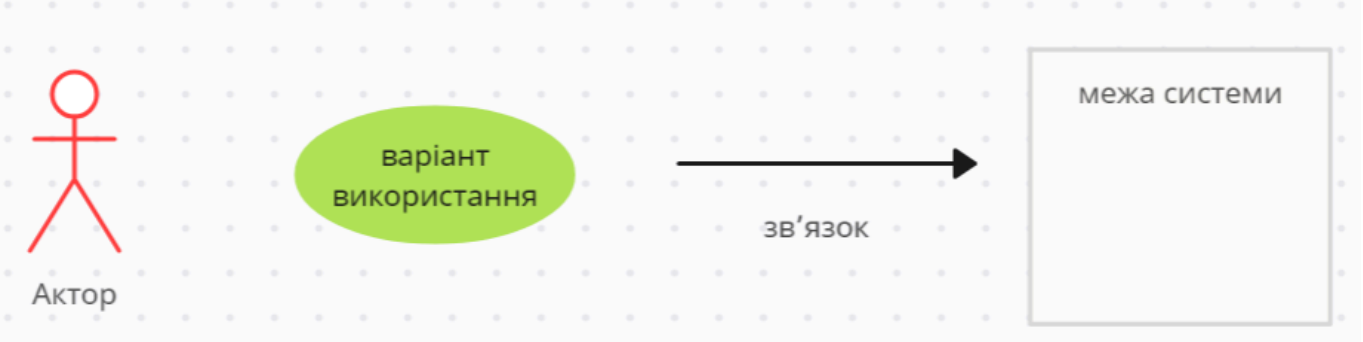

Рис. 2.1. Умовні позначення use case діаграми

Джерело: створено автором

Актор - представляє зовнішню сутність, яка взаємодіє з системою. Актором може бути користувач, інша система або будь-яка зовнішня сутність, яка взаємодіє з модельованою системою. Актори представлені у вигляді фігурок і взаємодіють з системою, беручи участь в одному або декількох варіантах використання. Актори важливі, оскільки вони допомагають визначити межі системи та ідентифікувати зацікавлені сторони, які будуть використовувати систему.

Варіант використання - з точки зору користувача, варіант використання представляє певну функціональність або поведінку системи. Він описує серію дій або кроків, які система виконує для досягнення конкретної мети користувача. Кожен варіант використання представляє одну послідовну функцію, яка має цінність для користувача. Варіанти використання представлені у вигляді овалів з мітками, які описують операції або послуги, що надаються системою.

Зв’язки - відношення включення вказує на те, що один варіант використання містить функціональність іншого варіанту використання. Він використовується для

моделювання спільної поведінки між декількома варіантами використання. Коли варіант використання містить інший варіант використання, це означає, що варіант використання, який міститься в ньому, завжди виконується як частина варіанту використання, що міститься в ньому. Відношення включення зображується у вигляді пунктирної лінії зі стрілкою, що вказує від охоплюючого варіанту використання до включеного варіанту використання.

Межа системи - це великий прямокутник або поле, яке включає всі варіанти використання та акторів на діаграмі. Вона візуально представляє обсяг або межі програмної системи, що моделюється. Все, що знаходиться в межах границі, представляє функціональність, яку система надає своїм користувачам або зовнішнім суб'єктам.

Безпосередня побудова діаграми прецедентів

Для того, щоб у всіх розробників проєкту було спільне бачення для успішного виконання поставлених задач, було вирішено побудувати UML діаграму за допомогою платформи, що знаходиться на сайті app.creately.com.

Як можна бачити на діаграмі (рис. 2.2), майбутній веб-застосунок має два основних типів користувачів, а саме учень та інструктор.

Також було створено наступні випадки використання:

1. Реєстрація. Описує процес реєстрації нового користувача в системі.
2. Вхід. Описує процес входу зареєстрованого користувача в систему.
3. Перегляд подій. Дозволяє користувачеві переглядати список доступних подій або курсів.
4. Реєстрація на подію. Запису користувача на обрану навчальну подію або курс.
5. Перегляд деталей події. Перегляду детальної інформації про обрану навчальну подію або курс.
6. Пошук подій за категоріями. Дозволяє шукати навчальні події за певними категоріями або темами.

7. Створення події. Створення нової навчальної події або курсу.
8. Редагування події. Цей випадок використання описує процес редагування існуючої навчальної події або курсу.

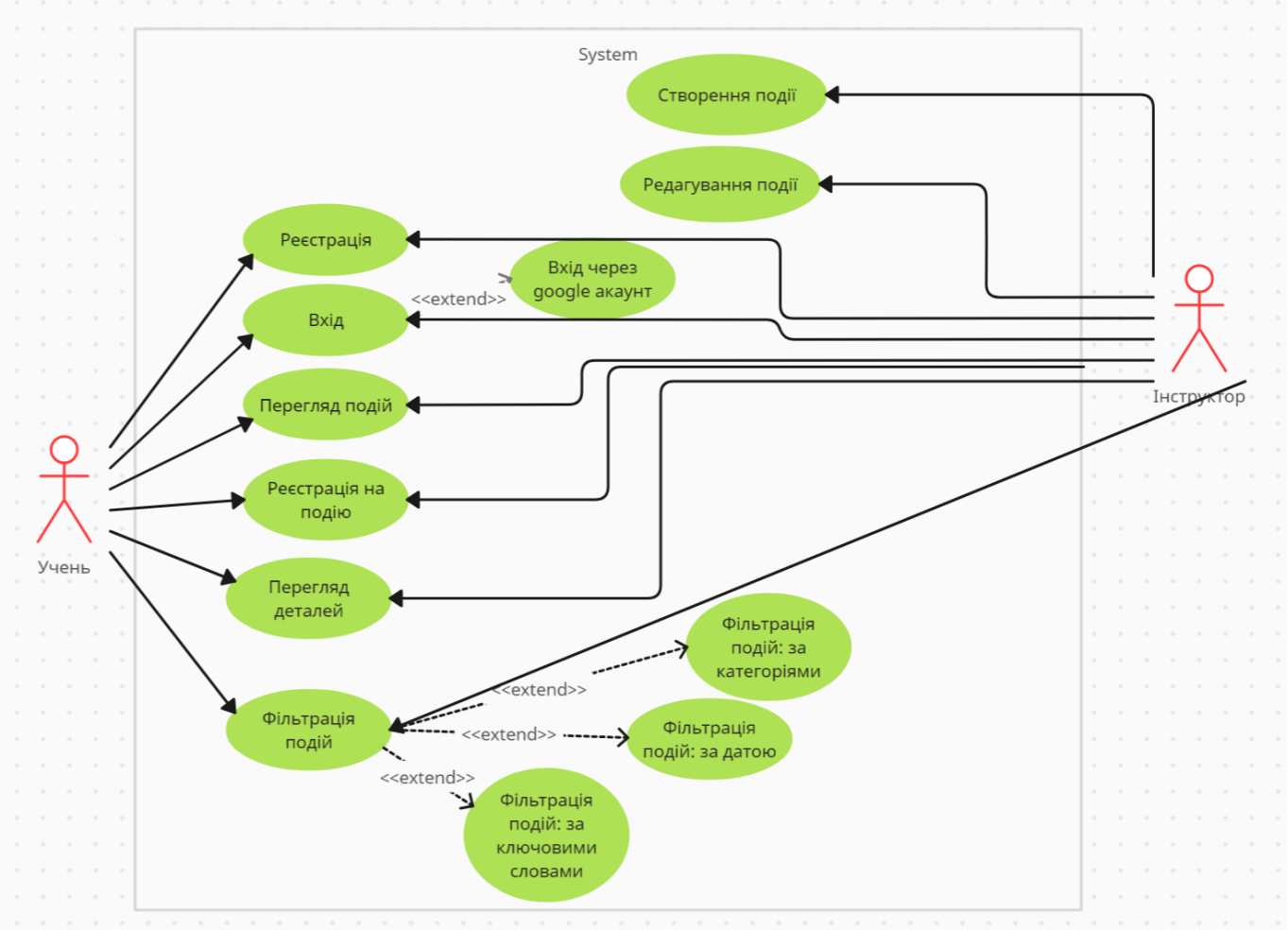

Рис. 2.2. Use case діаграма проєкту Jet Study

Джерело: створено автором

2.2. Проєктування бази даних

Бази даних - це важливий інструмент для зберігання та організації великих обсягів даних, який використовується в абсолютній більшості сучасних веб-додатків. [6]

Таке широке використання обумовлене наступними причинами:

- бази даних дозволяють зберігати великі обсяги даних у структурованому форматі, що дозволяє ефективно керувати інформацією;
- надають швидкий доступ до потрібної інформації завдяки можливості індексації та оптимізації запитів;
- дозволяють забезпечити консистентність даних шляхом використання правил цілісності та контролю доступу;
- масштабуються від невеликих локальних баз даних до великих розподілених систем;

Проєкт не може існувати без бази даних через потребу ефективного зберігання, організації та доступу до даних про користувачів, події, курси та інші важливі аспекти системи.

Існують різні види баз даних, такі як ієрархічні, мережеві, об'єктні, об'єктно-реляційні тощо. Всі ці типи БД мають широкий спектр можливостей використання та корисні в своїх індивідуальних випадках. Деякі з них швидші, що зумовлено їхніми особливістями організації даних. Інші ж навпаки повільніші, проте дозволяють набагато гнучкіше налаштовувати необхідні властивості. Однак, одним з найпоширеніших та використовуваних типів є реляційна база даних (РБД).

У реляційній базі даних, а саме її було обрано для використання у системі, інформація представлена у вигляді таблиць, де кожна таблиця складається з рядків і стовпців. Вона базується на математичній теорії реляційної алгебри та забезпечує зв'язок між різними таблицями за допомогою ключів. Реляційні бази даних дозволяють ефективно організовувати дані, забезпечуючи їх консистентність та легкість управління. [7]

Прикладами такого типу баз даних є MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

Ми виділили наступні етапи проектування БД:

1. Підготовка вимог до БД
2. Створення сутностей/таблиць
3. Створення полів, призначення для них відповідних типів даних
4. Встановлення зв’язків між таблицями
5. Заповнення даними
6. Здійснення адміністрації та підтримки бази даних

Почали ми з того, що розробили концептуальну модель бази даних, попередньо провівши дослідження щодо її майбутнього вмісту. Таким чином, ми додали всі необхідні сутності, поля, ключі та зв’язки між таблицями та позначили їх схематично.

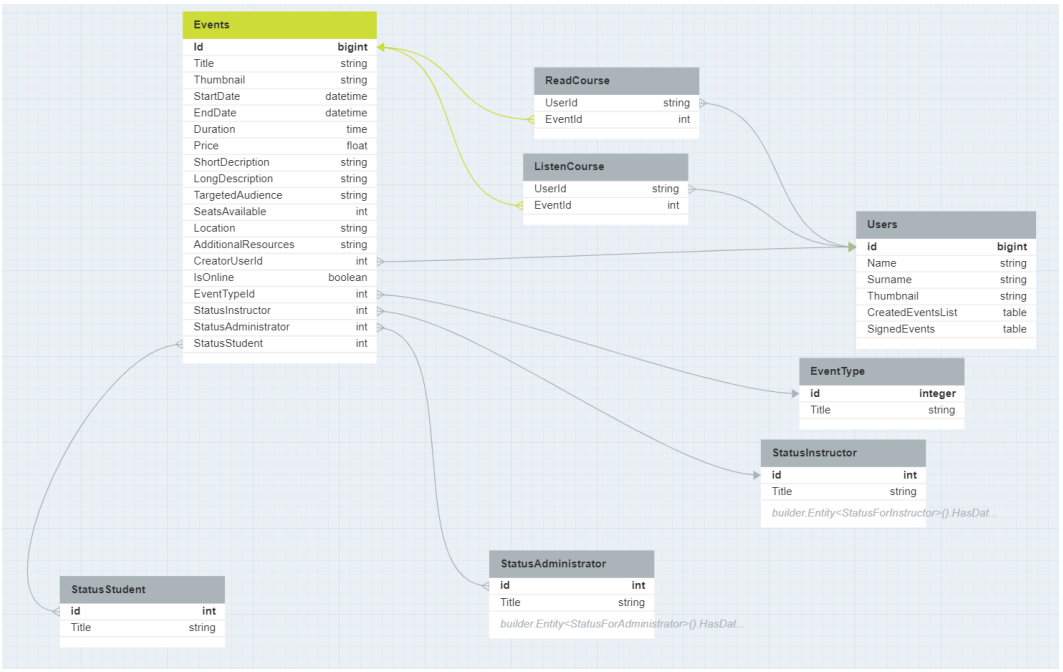

Рис. 2.3. Процес створення концептуальної моделі бази даних

Джерело: створено автором

Entity Framework (EF) - це фреймворк для роботи з базами даних у середовищі .NET. Його основна мета - спростити роботу розробника з базами даних, забезпечуючи можливість взаємодії з базою даних через об'єкти програми, а не SQL-запитами. [8]

Основні поняття Entity Framework:

1. Модель Даних:

EF дозволяє визначати модель даних за допомогою класів, які представляють об'єкти вашого додатка. Ці класи називаються сутностями (entities), і вони відображаються на таблиці в базі даних.

2. DbContext:

DbContext - це клас, який представляє контекст бази даних. Він дозволяє взаємодіяти з базою даних через LINQ-запити та методи для вставки, оновлення та видалення даних.

3. Code First та Database First:

Entity Framework підтримує два підходи до роботи з базами даних: Code First та Database First. Code First дозволяє визначати структуру бази даних за допомогою коду C# (класів), тоді як Database First використовує існуючу базу даних для автоматичної генерації класів.

4. Міграції:

EF надає механізм міграцій, який дозволяє автоматично оновлювати структуру бази даних при зміні моделі даних. Це допомагає уникнути проблем під час розвитку додатка. Entity Framework робить роботу з базами даних в .NET зручною та абстрагованою від деталей SQL, дозволяючи розробникам працювати з даними в термінах об'єктів та мови програмування C#. [9]

Існує 3 основних підходи до створення баз даних: model first, database first, code first.

Model first - в цьому випадку спершу створюється модель бази даних за допомогою Entity Relationship Diagram. Таким чином можна вручну налаштувати все максимально індивідуально, так як потрібно у конкретному випадку. Це допомагає позбутися надмірності даних та заплутаності зв’язків. Останній етап включає в себе перетворення отриманої моделі даних на код.

Database first - в цьому підході база даних спочатку створюється відповідно до вимог проекту. Після створення бази даних вона імпортується в розроблювану програму, де автоматично генерується модель даних або класи, які відображають структуру бази даних.

Code first approach - ідея даного підходу полягає в тому, щоб спершу за допомогою мови, що використовується в проєкті, описати класи і зв’язки між ними, а потім спеціальними інструментами перетворити його на набір таблиць з атрибутами та зв'язками. [10] Опишемо цей варіант детальніше, так як було вирішено, що він найкраще підходить для реалізації створення бази даних в нашому сервісі, а також відповідає нашим можливостям. Вимогами для успішного використання та відтворення всього потенціалу є:

- створення класів: за допомогою мови програмування, в цьому випадку мови C#, створити класи, які у базі даних відповідатимуть таблицям/сутностям;
- встановлення зв’язків у класах. Наприклад, для отримання зв’язку один-до-багатьох, необхідно в одному класі, вказати як поле інший, а також ідентифікатор потрібного класу. Таким чином, у базі даних також буде створений потрібний зв’язок;
- визначення контексту даних. Реалізовується за допомогою наслідування від базового класу в ASP.NET AbContext;
- налаштування правил створення бази даних та розміри полів, їх типи та властивості;
- встановлення правил створення бази даних
- виклик функції перетворення коду на базу даних за допомогою надсилання міграцій.

З урахуванням можливостей і переваг описаних підходів, було вирішено, що наша база даних буде створена за допомогою принципу code first.

Entity Framework Core дозволяє нам це зробити за зрозумілим з назви принципом - спершу за допомогою коду описуються сутності, а потім він вже виконує свою роботу та на основі цього створює базу даних. [11]

Отже, було описано 10 моделей, необхідних для роботи додатку, а саме: ApplicationToEvent, Category, Event, EventTypes, ListenCourse, ReadCourse, StatusForAdministrator, StatusForInstructor, StatusForStudent, User, Basket, BasketItem. Код головної сутності Event можна переглянути у **додатку А.**

Для коректної роботи даних було налаштовано деякі правила створення зв'язків між сутностями. Таким чином, ми налаштували поведінку об'єктів при їх видаленні, а саме заборонили видалення тих записів з бази даних, які використовуються в іншому місті.

Такі налаштування необхідно застосовувати, коли встановлення зв'язків за допомогою code convention способу неможливе, або не може на повну зобразити необхідний вигляд.

Code convention підхід являє собою варіант написання класів та їх полів таким чином, щоб наша ORM, тобто Entity Framework сам міг зрозуміти типи даних та встановити необхідні зв'язки між класами/таблицями. [13] Наведу короткий приклад використання:

- уявимо, що в нас є таблиці Student та Department. Для того, щоб зобразити зв'язок один-до-багатьох, необхідно зробити деякі маніпуляції. Так як у класі Department є поле Id з типом long int, що являє собою ідентифікатор даної таблиці, необхідно у клас Student додати поле DepartmentId з таким же типом.

```
public void Configure(EntityTypeBuilder<Event> builder)
{
    builder.HasMany(x => x.Lecturers).WithOne(x => x.Event).HasForeignKey(x
=> x.EventId).OnDelete(DeleteBehavior.Restrict);
    builder.HasMany(x => x.Students).WithOne(x => x.Event).HasForeignKey(x =>
x.EventId).OnDelete(DeleteBehavior.Restrict);
```

```
        builder.HasMany(x => x.ApplicationToEvents).WithOne(x =>
x.Event).HasForeignKey(x => x.EventId).OnDelete(DeleteBehavior.Restrict);
    }
```

Лістинг 2.1. Правила створення зв’язків у БД

Приклад зверху є варіантом застосування функцій Fluent API, що призначений для строгого встановлення правил, які мають пріоритет над code convention підходом та будуть застосовані поверх нього. Нижче наведено ключові методи Fluent API та їх призначення:

1. HasKey(), HasForeignKey(), WithMany() - для установки зв’язків між таблицями.
2. Property() - для встановлення типів даних та обмеження в пам’яті.
3. WillCascadeOnDelete() - встановлення правил видалення з бази даних.

Отже, Fluent API призначений для конфігурації зв’язків між таблицями, створення обмежень кількості символів, визначення типів даних, встановлення правил створення та видалення елементів БД, налаштування первинних та зовнішніх ключів.

Важливо пам’ятати ключову перевагу цього методу над code convention - всі ці налаштування можна вказувати не безпосередньо в класі сутності, а зберігати в будь-якому іншому місці, що збільшує зрозумілість коду, зменшує його нагромадженість та сприяє зручному розширенню функціоналу.

Під час реалізації функціоналу веб-додатку багато таблиць та зв’язків неодноразово змінювалися. Це пов’язано з тим, що на початкових стадіях розробки не завжди одразу видно всі тонкощі, які необхідно врахувати для збереження даних. Таким чином, ми дійшли до остаточної схеми бази даних, що зображена на *рисунку 2.1.4*:

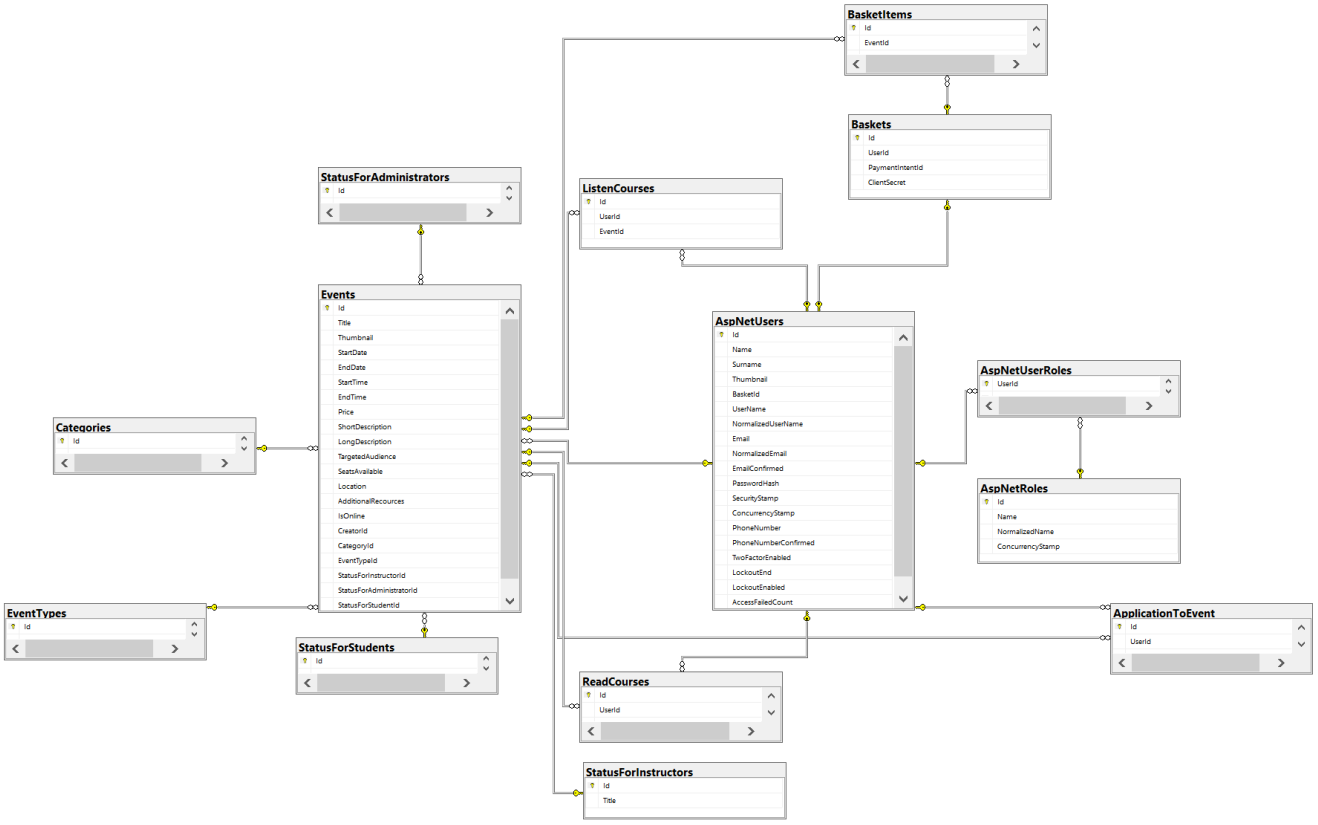

Рис. 2.4. Схема бази даних в середовищі MS SQL SERVER

Джерело: створено автором

2.3. Знаходження архітектурного рішення

2.2.1 Чиста архітектура

Clean Architecture - це концепція архітектури програмного забезпечення, яку розробив Роберт Мартін (Robert C. Martin). [15] Ця концепція базується на принципах SOLID та інших добре відомих підходів до проектування, і її ціль - створення програмного забезпечення, яке є гнучким, зрозумілим та доступним для тестування.

Clean Architecture розвивалася як відповідь на велику кількість великих та складних проектів, де традиційні архітектурні підходи часто призводили до проблем з розширюваністю та підтримкою. Вона стала результатом досліджень Роберта Мартіна та його роботи над забезпеченням, яке було б незалежним від технічних деталей та могло б легко адаптуватися до змін.

Clean Architecture включає в себе чотири основні рівні, які визначають структуру програмного забезпечення та розділяють його на логічні секції. Кожен рівень представляє собою визначений шар в системі, з власними відповідальностями. Загальну структуру такого підходу до рішення архітектури відображено на рисунку 2.5.

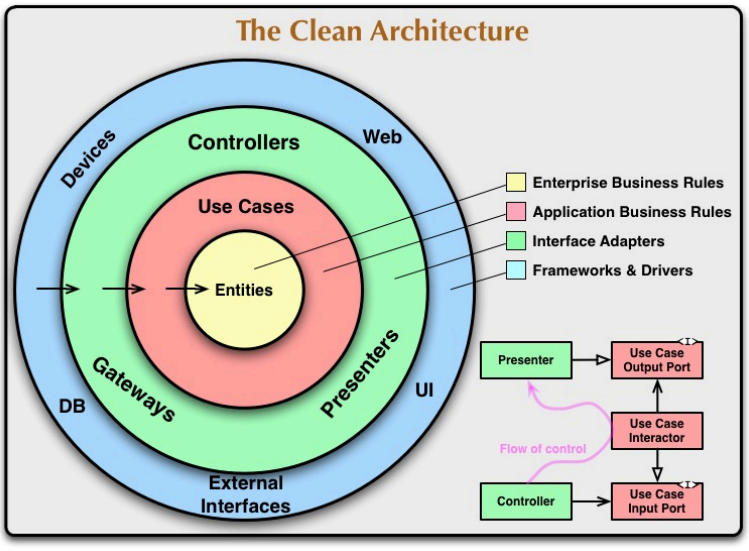

Рис. 2.5. Організація шарів у Clean Architecture [14]

Рівень Домену (Domain Layer)

Це центральний та найважливіший рівень системи. Включає в себе бізнес-логіку, сутності та правила домену. Наприклад, він включає в себе сутності та їх характеристики. Ці об'єкти можуть бути класами моделі додатку чи моделей бази даних. Здебільшого, ці сутності також використовуються для створення таблиць у базі даних.

Характеристики:

- Незалежний від будь-яких інших рівнів.
- Містить ключові бізнес-концепції.

Рівень Прикладного Програмного Забезпечення (Application Layer)

Відповідає за використання бізнес-правил та доменних об'єктів для обробки конкретних запитів та сценаріїв використання.

Характеристики:

- Керує потоком виконання програми.
- Використовує доменні об'єкти для вирішення конкретних завдань.

Рівень Інфраструктури (Infrastructure Layer)

Містить всі залежності системи від зовнішніх агентів, таких як бази даних, фреймворки, зовнішні служби тощо.

Характеристики:

- Забезпечує зовнішню взаємодію системи.
- Містить реалізації інтерфейсів та репозиторіїв.

Рівень Представлення (Presentation Layer)

Містить всі елементи, які стосуються відображення і взаємодії з користувачем. Тут за допомогою HTTP-requests можна отримати необхідні дані для завантаження сторінок на фронтенді у вигляді JSON-об’єктів.

Його основні характеристики:

- Взаємодіє з рівнем додатків та інфраструктурою.
- Перетворює дані для відображення користувачеві.

Ця архітектура розділяє відповідальності та забезпечує відокремлення бізнес-логіки від деталей інфраструктури та зовнішніх залежностей. Кожен рівень має чітко визначені відповідальності, що полегшує тестування, розширення та підтримку коду.

Використання чистої архітектури має кілька ключових переваг, серед яких можна виділити:

1. Легкість розширення.

Чиста архітектура сприяє створенню систем, які легко розширюються. Зміни в бізнес-логіці не впливають на інші частини системи, що полегшує внесення змін та додавання нового функціоналу.

2. Легкий доступ до тестування.

Кожен рівень в чистій архітектурі може бути легко та незалежно протестованим. Це дозволяє розробникам швидко виявляти та виправляти помилки, підтримуючи високий рівень якості коду.

3. Зменшення залежностей.

Архітектура дозволяє уникати прямих залежностей між компонентами системи. Це робить код більш гнучким і менше залежним від конкретних технологій чи інфраструктури.

4. Модульність та розподіл обов’язків.

Так як чиста архітектура допомагає розділити обов’язки кожної частини коду, це допомагає краще зрозуміти призначення того чи іншого модуля та змінювати його без впливу на всю систему. Це забезпечує можливість використовувати існуючий код в різних контекстах.

5. Збереження бізнес-логіки незалежною.

Бізнес-логіка розташована в центрі архітектури, і вона залишається незалежною від інших компонентів системи. Це дозволяє зберігати чистоту та ясність коду, спрощуючи його обслуговування.

6. Масштабованість.

Чиста архітектура робить систему більш масштабованою. Таким чином, у проекті можна підтримувати велику кількість компонентів та з’єднань між ними, без значного навантаження на додаток.

Загалом, чиста архітектура надає концептуальну ясність, робить систему більш стійкою до змін та сприяє розробці високоякісного програмного забезпечення.

2.2.2 Архітектура рішення проєкту

Для того, щоб організувати архітектурне рішення проєкту за принципом чистої архітектури, було вирішено створити 3 проекти у рішенні, тобто 3 шари, у кожного з яких є своя відповідальність. Для цього рішення було достатньо об’єднати ядро системи та рівень бізнес-логіки в один проєкт, тобто Core. Infrastructure в свою чергу відповідає за рівень доступу до даних, а ASP.NET Core Web API за рівень представлення даних, призначений для спілкування з клієнтом. Далі детальніше про кожен з рівнів.

Рівень Infrastructure.

Реалізований за допомогою бібліотеки класів. У ньому міститься вся бізнес-логіка проекту, за допомогою якої наш додаток має свої особливості та може виконувати корисну роботу та надавати необхідний функціонал. Зокрема в ньому розташовані DTO - об’єкти для транспортування даних на клієнтську частину, виключення, специфікації, призначені для написання запитів до бази даних, а також сервіси, у яких безпосередньо прописаний функціонал додатку.

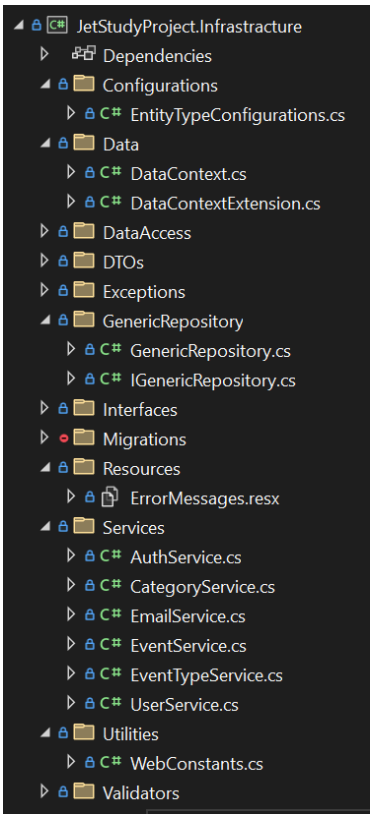

Рис. 2.6. Структура рівня Infrastructure

Джерело: створено автором

Рівень Core.

Реалізований за допомогою бібліотеки класів. На цьому рівні проєкт містить все необхідне для доступу та роботи з базою даних. Він знає про рівень Core, та йому нічого не відомо про рівень представлення, тобто API проєкт. У ньому міститься DataContext, Seeder для заповнення бази даних початковими даними, також там реалізовані паттерни репозиторій та Unit of Work.

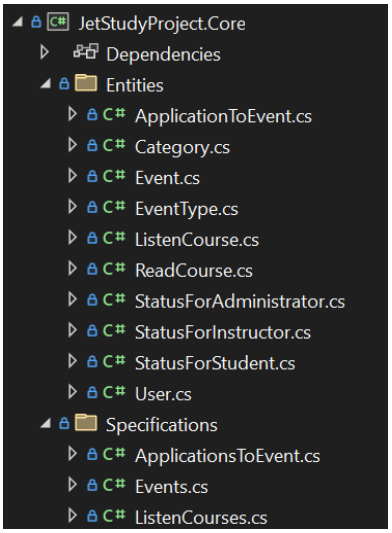

Рис. 2.7. Структура рівня Core

Джерело: створено автором

Рівень API:

Завдяки цьому рівню клієнт може спілкуватися з серверною частиною додатку, тобто передавати дані на сервер, або отримувати їх. Цей рівень є проєктом запуску, конфігурації якого реалізовані у класі Program. Там же ж розташовані контролери, у яких містяться необхідні ендпоінти. Також в user secrets лежать підключення до бази даних, а також до пошти, з якої йде розсилка для користувачів додатку.

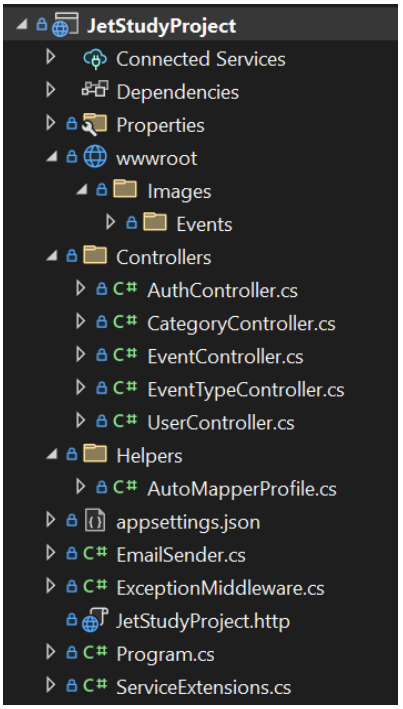

Рис. 2.8. Структура рівня API

Джерело: створено автором

В ньому знаходиться 5 контролерів, а саме AuthController, CategoryController, EventController, EventTypeController, UserController. У кожного з них я своє функціональне призначення, тому інформацію про них варто подати в детальному вигляді.

AuthController - призначений для реалізації функцій авторизації та аутентифікації користувачів.

CategoryController та *EventController* - призначені для маніпуляцій читання, редагування, видалення та додавання інформації, пов’язаної з таблицями Category та Event.

EventController - містить в собі основний функціонал веб-додатку, необхідний для роботи з подіями. Також реалізовані різні рівні доступу до можливості виконання запитів від користувачів. У інструктора, як і планувалося в завданні, доступ розширений, адже для нього критично важливо мати можливість гнучко налаштовувати свої опубліковані події.

UserController - відповідає за отримання інформації про користувача та визначає методи редагування профілю, тобто особистих даних користувача, таких як: логін, пароль, нікнейм і т.д.

Висновки до розділу 2

Під час написання цього розділу було виділено багато часу та ресурсів для досліджень можливостей проектування системи, а також втілення найкращих сучасних підходів для побудови структури проекту, бази даних, а також моделі програмної системи.

Зокрема було проаналізовано існуючі типи UML діаграм, їх властивості, переваги та недоліки і обрано use case діаграму для створення та модифікацій, які відображали б сутність майбутнього проєкту. Це дало змогу краще зрозуміти мету проєкту за допомогою візуальної складової, уніфікувати умовні позначення, завдяки яким команда була в тонусі та бачила, в якому напрямку ми рухаємося.

За допомогою code first approach було спершу створено сутності в ASP.NET проєкті, які потім разом із налаштуваннями Fluent API, code convention, та міграцій наша обрана ORM Entity Framework перетворила на базу даних з усіма необхідними зв’язками, таблицями, полями та їх властивостями.

Також було вирішено використати чисту архітектуру як базу для побудови проєкту, на основі попередніх досліджень. Основними її перевагами визначено: легкість розширення, хороша тестованість продукту, зменшення залежностей,

модульність, масштабованість, а також загальноприйняті правила, які допоможуть в отриманні консультацій від спеціалістів.

Таким чином, clean architecture є саме тією основою, яка максимально спрощує процес розробки, адже є дуже потужним підходом до проєктування систем. Вона надає змогу надавати технічну підтримку проєкту ще довго опісля його випуску, що робить її також вигідною в плані вкладу фінансових ресурсів. Таку систему не потрібно буде переробляти на нову через кілька років.

РОЗДІЛ 3
ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА НАПИСАННЯ ФУНКЦІОНАЛУ СЕРВЕРНОЇ ЧАСТИНИ

3.1 Опис необхідних засобів для розробки

3.1.1 Технічний стек використаних технологій

Основні технології:

- ASP.NET Core - фреймворк для створення швидких та надійних веб-застосунків
- C# - найпопулярніша мова з родини C. Широко використовується на платформі .NET
- Entity Framework - це ORM(object-relational mapping), яка дозволяє на основі об'єктно орієнтованого code first підходу створювати базу даних

Основні бібліотеки:

- Ardalis.Specifications - це бібліотека, яка дозволяє писати запити до бази даних, які можна використовувати багаторазово
- Automapper - дана бібліотека допомагає перетворювати громіздкі об’єкти з бази даних у ті, які клієнту буде зручно отримувати та використовувати
- FluentValidation - допомагає валідувати дані ще на сервері за допомогою Fluent API
- Google.Apis.Auth - бібліотека, призначена для авторизації користувачів у додатку через Google API
- MailKit - забезпечує взаємодію з поштовими серверами
- Swashbuckle - надає функціонал створення документації для сервісів API
- Microsoft.AspNetCore.Identity.EntityFrameworkCore - бібліотека для роботи з користувачами, ролями, доступами, ключами і т.д.

3.2 Вибір програмного забезпечення

Для розробки серверної частини веб-додатку було використано потужні програми, які дозволяють вести ефективну розробку для того, хто з ними працює, а особливо корисними вони стають, коли потрібно працювати в команді і отримувати оновлення вчасно.

Microsoft Visual Studio - це багатофункціональне середовище розробки, яке підтримує не лише родину мов програмування C, але Python та інші. У ньому присутня інтеграція з фреймворком ASP.NET, який включає в себе багато інших, тому MS Visual Studio став кращим вибором для розгортання проєкту саме в ньому.

MS SQL SERVER - це сервер для роботи з базою даних на основі SQL, в якому є не лише рядковий, але і графічний інтерфейс. Таким чином, у ньому можна з легкістю задавати типи полів, зв’язки між таблицями, а також отримувати діаграми з усією структурою бази даних, що є хорошим наочним відображенням для чудового розуміння поточного стану БД.

GitHub надає зручний інтерфейс для контролю версій коду, що є важливим для відстеження та збереження змін у проекті. Також він сприяє спільній роботі команди, дозволяючи кожному розробнику завжди мати актуальний стан проекту на руках, внести свій внесок та обговорити зміни. [16]

GitHub забезпечує високий рівень безпеки, а також дозволяє легко працювати з кодом, вносити зміни, створювати гілки та об'єднувати їх, що полегшує управління проектом.

Postman є наріз одним із найпопулярніших інструментів для тестування API. З його допомогою я успішно протестував усі API запити, написані на серверній частині.

3.3 Здійснення розробки функціоналу для серверної частини веб-додатку

3.3.1. Написання AuthService

Було прийнято рішення написати сервіс для авторизації, який ми матимемо змогу доповнювати в разі потреби. В його основі лежить використання JWT.

JWT (JSON Web Token) Authentication - це метод аутентифікації в мережі, що використовується для забезпечення безпеки між веб-сервером та клієнтом. Основна ідея полягає в тому, що сервер генерує підписаний JWT, який містить інформацію про користувача, і відправляє його клієнту. Після цього клієнт включає цей токен у кожен запит до сервера. Сервер перевіряє цей токен, розшифровує його та перевіряє його цілісність, щоб переконатися, що запит дійсно відправив автентифікований користувач. [17]

Отже, основними компонентами JWT є:

Header (Заголовок): Це JSON-об'єкт, який містить метадані про токен, такі як тип токена та алгоритм шифрування.

Payload (Навантаження): Це JSON-об'єкт, який містить інформацію про користувача або будь-яку додаткову інформацію. Payload може містити будь-яку кількість власних полів.

Signature (Підпис): Це підпис, який генерується на основі заголовка, навантаження та секретного ключа. Цей підпис використовується для перевірки цілісності токена.

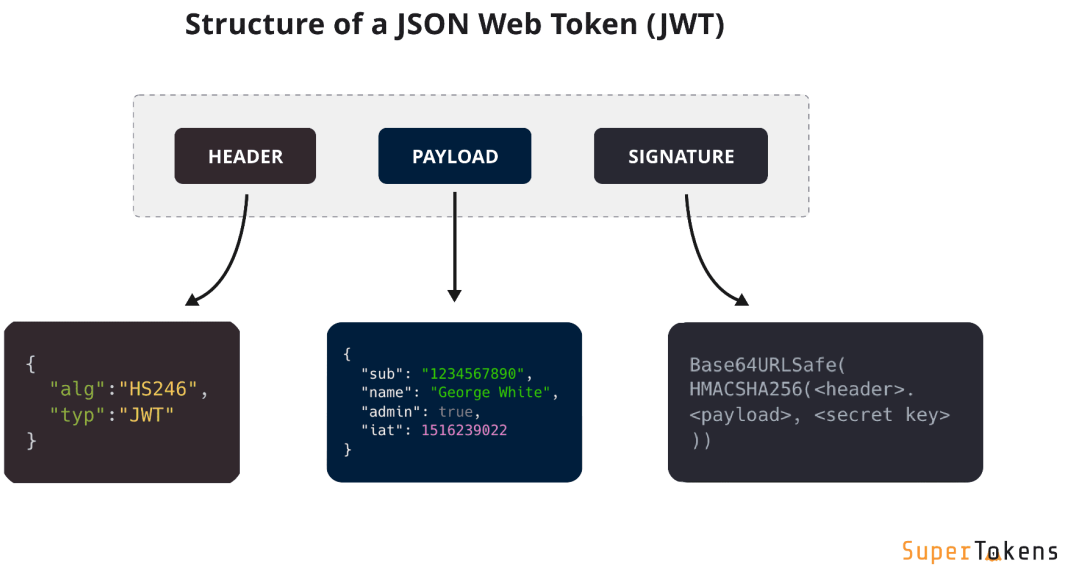

Рис. 3.1. Структура JSON Web Token

Джерело: створено автором

Використання JWT має декілька переваг:

Легкість реалізації і використання. JWT використовує простий у використанні формат JSON, що робить його легким для реалізації та використання у веб-додатках.

Незалежність від стану. Він не потребує зберігання стану сервером. Це дозволяє горизонтально масштабувати додаток без необхідності розподіленого сеансу або централізованої бази даних сеансів.

Безпека. JWT може бути підписаний та зашифрований, що дозволяє перевірити цілісність даних та їх конфіденційність. Крім того, оскільки вони передаються у вигляді токенів, ризик витоку інформації зменшується порівняно з іншими методами аутентифікації.

Універсальність. JWT можуть бути використані в будь-якій мові програмування та середовищі, оскільки вони побудовані на стандартах відкритих даних, таких як JSON та Base64.

Розширюваність. JWT мають гнучкість у включенні додаткової інформації у вигляді полів у самому токені. Це дозволяє розширювати їх функціональність для вирішення різних потреб аутентифікації та авторизації.

У цілому, використання JWT забезпечує простий та безпечний спосіб автентифікації користувачів у веб-додатках.

Таким чином, використовуючи дану технологію, було написано наступний код для створення токена:

```
private async Task<string> GenerateTokenAsync(User user)
{
    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration[
"Jwt:Key"]));
    var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);
    var roles = await _userManager.GetRolesAsync(user);
    string role;
    if (roles.Count > 0)
    {
        role = roles[0];
    }
    else
    {
        role = "null";
    }
    var claims = new[]
    {
```

```
        new Claim(ClaimTypes.NameIdentifier, user.Id),
        new Claim(ClaimTypes.Role, role),
        new Claim("Name", user.UserName),
        new Claim("Email", user.Email)
    };

    var token = new JwtSecurityToken(
        _configuration["Jwt:Issuer"],
        _configuration["Jwt:Audience"],
        claims,
        expires:
DateTime.Now.AddMinutes(Int32.Parse(_configuration.GetSecti
on("Jwt:Expires").Value)),
        signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Лістинг 3.1. Генерація токену

У проекті вже було попередньо налаштовано EmailSender, який відповідає за надсилання важливих листів на пошту клієнтів під час роботи з сервісом. Однак, так як ми перестали використовувати Identity API Endpoints, які влаштовані в .NET 8, ми також були змушені створювати нову логіку реєстрації на сервісі, яка включає в себе отримання користувачем листа для підтвердження валідності пошти. [18]

```
var confirmationCode = await
_userManager.GenerateEmailConfirmationTokenAsync(userInDb);
var callbackUrl = _helper.Action(
                        "ConfirmEmail",
                        "Auth",
                        new { userId = user.Id, code =
confirmationCode },
                        protocol:
_helper.ActionContext.HttpContext.Request.Scheme);
            EmailService emailService = new
EmailService(_configuration);
            await
emailService.SendEmailAsync(userRegisterDto.Email, "Confirm
```

```
your account",
                $"Перейдіть за посиланням, щоб підтвердити
аккаунт: <a href='{callbackUrl}'>link</a>");
```

Лістинг 3.2. Код для генерації та відправки листа підтвердження реєстрації на пошту

Наступним кроком було вирішено додати зручний спосіб входу на платформу, без постійного вводу паролю та логіну, які легко забути та важко віднайти. Хорошим варіантом було погоджено використання Google OAuth.

Google OAuth - це механізм аутентифікації, який дозволяє користувачам увійти в сторонні веб-сайти або додатки, використовуючи їх облікові записи Google. В основі цього процесу лежить концепція довіри (trust). Коли користувач використовує Google OAuth для увійдіть на інший сайт або додаток, він надає цьому сайту або додатку доступ до певної частини свого облікового запису Google без передачі пароля. Такий підхід забезпечує безпеку облікового запису Google, оскільки користувач ніколи не розголошує свій пароль сторонній службі.

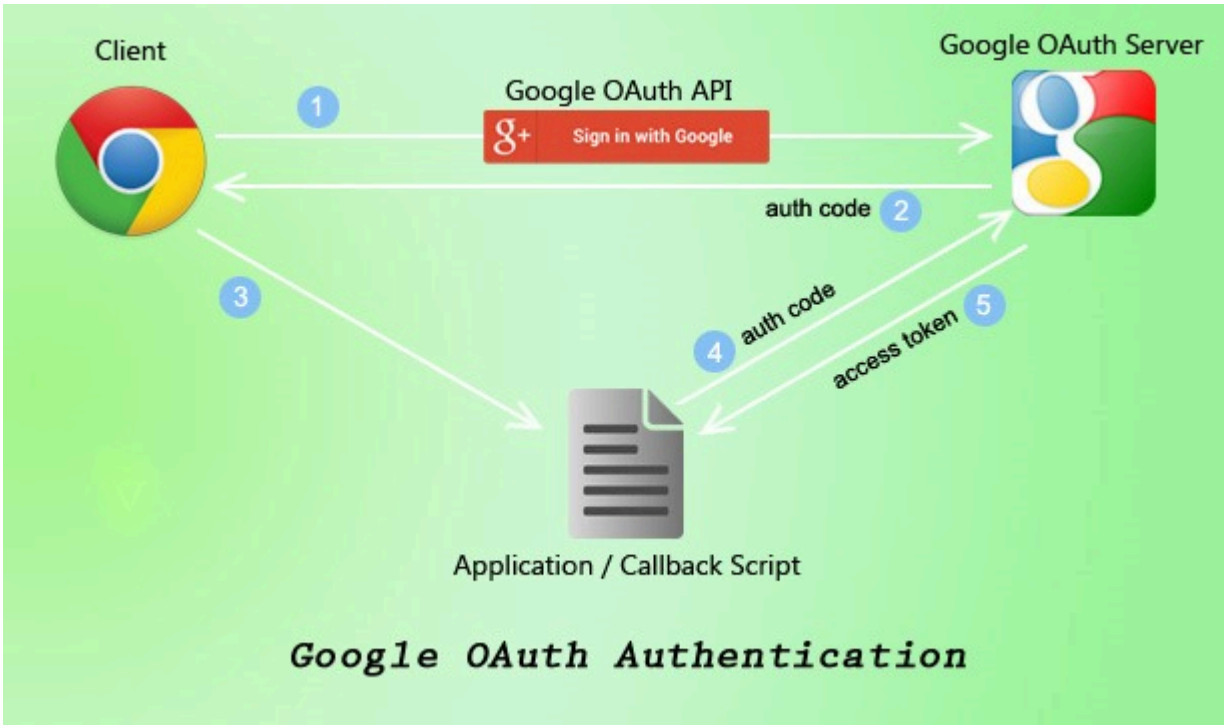

Рис. 3.2. Схематична робота Google OAuth Authentication

Джерело: створено автором

Відомо, що Google OAuth працює наступним чином:

Користувач переходить на веб-сайт або в додаток, який пропонує йому увійти за допомогою облікового запису Google.

Веб-сайт або додаток перенаправляє користувача на сторінку Google для входу. Користувач вводить свої облікові дані Google (електронна адреса та пароль) або, якщо він вже увійшов в Google на даному пристрої, підтверджує свою особу.

Після успішної аутентифікації [19] Google генерує для користувача спеціальний токен доступу (access token) та надає його веб-додатку. Веб-сайт або додаток використовує цей токен доступу для отримання доступу до певної інформації з облікового запису Google користувача, яку він погодився надати. [20]

Отже, Google OAuth дозволяє іншим веб-сайтам та додаткам отримувати обмежений доступ до облікового запису Google користувача без необхідності розголошувати його пароль. З метою розшифрування credentials, які будуть надходити з клієнту, було використано бібліотеку Google.Apis.Auth. Таким чином, було сформовано наступний код, після виконання якого на клієнтську частину веб-додатку буде надіслано JWT:

```
public async Task<LoginAnswerDto> LoginWithGoogle(string
credential)
        {
            var settings = new
GoogleJsonWebSignature.ValidationSettings()
            {
                Audience = new List<string> {
_configuration["Google:GoogleClientId"] }
            };
            var payload = await
GoogleJsonWebSignature.ValidateAsync(credential, settings);
            var user = await
_userManager.FindByEmailAsync(payload.Email);
            if (user != null)
            {
                if (!await
_userManager.IsEmailConfirmedAsync(user))
                {
                    throw new HttpException("Ви не
підтвердили свою електронну адресу",
HttpStatusCode.BadRequest);
                }
                return new LoginAnswerDto { Key = await
GenerateTokenAsync(user) };
            }
            else
            {
                throw new HttpException("Користувача з
такою електронною адресою не існує",
HttpStatusCode.BadRequest);
```

Лістинг 3.3. Код для розшифрування google token та генерації JWT
Повний код AuthService можна переглянути в **додатку Б**.

Після написання коду. який описано в попередніх абзацах, було отримано наступні ендпоінти для аутентифікації і керування профілем користувача.

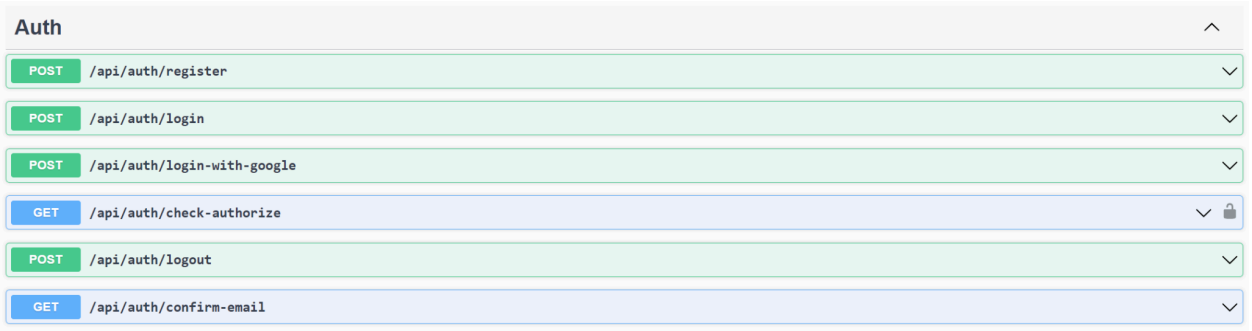

Рис. 3.3. Ендпоінти для авторизації

Джерело: створено автором

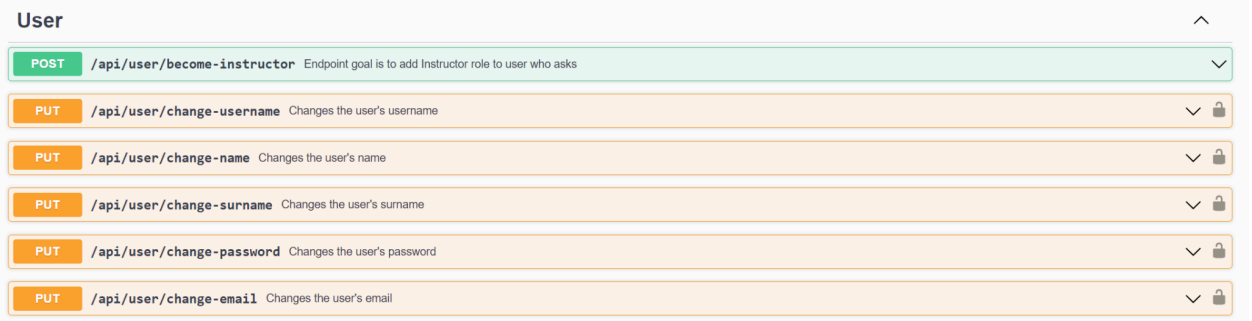

Рис. 3.4. Ендпоінти для керування профілем користувачем

Джерело: створено автором

3.3.2. Надсилання подій на клієнт

З наявних досліджень було сформовано завдання для отримання подій у вигляді трьох різних запитів по таким параметрам, як:

- наступні 7 днів
- поточний місяць
- наступний місяць

Наступний код задовольнив цю задачу:

```
public List<EventPreviewDto> GetThisWeekEventPreviews()
{
    var events =
_unitOfWork.EventRepository.GetListBySpec(new
Events.WithUserAndEventAndLectorers());

    events = events.Where(p => p.StartDate <=
```

```
DateTime.Now.AddDays(7) && p.StartDate >=
DateTime.Now.Date);

    return _mapper.Map<List<EventPreviewDto>>(events);
}

public List<EventPreviewDto> GetThisMonthEventPreviews()
{
    var events =
_unitOfWork.EventRepository.GetListBySpec(new
Events.WithUserAndEventAndLectorers());

    var monthOfTheYear = DateTime.Now.Month;
    var year = DateTime.Now.Year;

    events = events.Where(p => p.StartDate >=
DateTime.Now.AddDays(7) &&
    p.StartDate <=
DateTime.Now.AddDays(DateTime.DaysInMonth(year,
monthOfTheYear) - DateTime.Now.Day));

    return _mapper.Map<List<EventPreviewDto>>(events);
}

public List<EventPreviewDto>
GetEventPreviewsAfterThisMonths()
{
    var events =
_unitOfWork.EventRepository.GetListBySpec(new
Events.WithUserAndEventAndLectorers());

    var monthOfTheYear = DateTime.Now.Month;
    var year = DateTime.Now.Year;
    events = events.Where(p => p.StartDate >=
DateTime.Now.AddDays(DateTime.DaysInMonth(year,
monthOfTheYear) - DateTime.Now.Day));
```

```
    return _mapper.Map<List<EventPreviewDto>>(events);
}
```

Лістинг 3.4. Отримання подій за проміжком дат

Було отримано наступні ендпоінти для отримання подій:

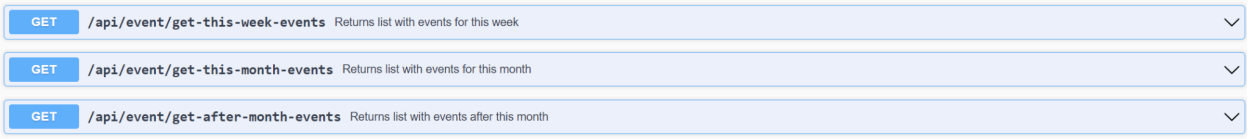

Рис. 3.5. Ендпоінти для подій

Джерело: створено автором

3.3.3. Керування подіями

Як вже раніше згадувалося, на даному етапі було вирішено зосередитися на функціоналі для лекторів, які бажають поділитися своїм досвідом і отримати від цього прибуток. Тому, було продовжено дописання EventService, який був створений для керування подіями.

В першу чергу було додано запит, який дозволяє будь-якому охочому користувачу стати лектором і публікувати події.

Наступним кроком було продумано логіку для шляху події від моменту створення до її публікації на сайті. Таким чином, було розроблено наступні функції з їх специфічними алгоритмами роботи:

1. Створення події

З клієнта ми отримуємо дані, які лектор хоче відобразити в події, а також дістаємо із сесії ідентифікатор поточного користувача. Перевіряю, чи такий користувач справді існує в системі. Якщо не отримую ствердної відповіді, завершую виконання функції із повідомленням про помилку. Використовую функцію для збереження файлу з зображенням події на сервері.

```
public async Task<string> SaveImage(IFormFile imageFile)
{
    string imageName = new
string(Path.GetFileNameWithoutExtension(imageFile.FileName)
.Take(10).ToArray()).Replace(' ', '-');
    imageName = imageName +
DateTime.Now.ToString("yymmssfff") +
Path.GetExtension(imageFile.FileName);
    var postFolder =
Path.Combine(_webHostEnvironment.WebRootPath,
WebConstants.eventsImagesPath);
    var imagePath = Path.Combine(postFolder, imageName);
    bool isExists = Directory.Exists(postFolder);

    if (!isExists)
    {
        Directory.CreateDirectory(postFolder);
    }
```

```
    using (var fileStream = new FileStream(imagePath,
FileMode.Create))
    {
        await imageFile.CopyToAsync(fileStream);
    }

    return imageName;
}
```

Лістинг 3.5. Функція збереження файлу зображення

Далі, за допомогою AutoMapper я формую новий об’єкт події і після цього зберігаю його в БД.

2. Редагування події

Особливістю цієї функції є те, що необхідно здійснити перевірки і провалідувати, чи певний користувач має право редагувати подію. Спершу ми впевнюємось, чи дана подія існує, далі звіряємо ідентифікатор засновника події з тим, хто хоче його змінити. Якщо вони не співпадають - надсилаємо повідомлення про помилку. Важливим моментом також було зробити ігнорування полів, в які не було здійснено зміни, тобто вони прийшли пустими. Для цього було здійснено наступні налаштування в профілі.

```
AutoMapper:
            CreateMap<EventEditDto, Event>()
            .ForAllMembers(opts =>
            {
                opts.AllowNull();
                opts.Condition((src, dest, srcMember) =>
srcMember != null);
            });
```

Лістинг 3.6. Налаштування AutoMapper

Після цього маппером ми оновлюємо подію, яку отримали з бази даних новими даними і зберігаємо зміни.

3. Видалення події

Перевіряємо права доступу на видалення, тобто здійснити цю функцію може лише власник, або адміністратор.

4. Надсилання події на модерацію

Після того, як лектор внесе всі необхідні правки, він може надіслати подію на модерування. Цей пункт є необхідним, адже таким чином можна запобігти розміщенню фішингових подій, які шахраї можуть використати для отримання незаконної вигоди.

```
 public async Task SendEventToModerate(int eventId, string
userId)
        {
            if (!IsExist(eventId))
            {
                throw new
HttpException(ErrorMessages.EventDoesNotExist,
HttpStatusCode.BadRequest);
            }

            var user = await
_userManager.FindByIdAsync(userId);

            if (user == null)
            {
                throw new
HttpException(ErrorMessages.InvalidUserId,
HttpStatusCode.BadRequest);
            }

            var eventToEdit =
_unitOfWork.EventRepository.GetById(eventId);

            if (eventToEdit.CreatorId != user.Id)
            {
                throw new HttpException("Ви не можете
змінювати подію, яка вам не належить",
HttpStatusCode.BadRequest);
            }

            if (!(eventToEdit.StatusForInstructorId == 1 ||
eventToEdit.StatusForInstructorId == 4))
            {
```

```
                throw new HttpException("Подія вже на
модерації, або опублікована", HttpStatusCode.BadRequest);
            }

            eventToEdit.StatusForInstructorId = 2;

_unitOfWork.EventRepository.Update(eventToEdit);
            await _unitOfWork.SaveAsync();
        }
```

Лістинг 3.7. Відправка події на модерацію

5. Публікація події

Після перевірки події на правдивість, а також на заповнення всіх необхідних полів інформацією, модератор може опублікувати подію і зробити її видимою для всіх користувачів. Після написання коду і підключення його до контроллеру, було отримано наступні ендпоінти у Swagger:

Рис. 3.6. Ендпоінти для подій

Джерело: створено автором

3.3.3. Маппінг Entities в DTO

Варто згадати також, що для маппингу було використано спеціальну бібліотеку AutoMapper [21], в якій необхідно було налаштувати профайл.

Під час виконання даної частини завдання я зіткнувся з проблемою, а саме формування шляху до зображення на сервері, щоб клієнт міг мати правильно налаштований доступ до нього. Для цього в класі Program я заінжектив IServer у профайл авто маппера.

```
builder.Services.AddSingleton(provider => new MapperConfiguration(cfg =>
{
   cfg.AddProfile(new
AutoMapperProfile(provider.CreateScope().ServiceProvider.GetService<IServer>()));
}).CreateMapper());
```

Лістинг 3.8. Інжектинг параметра IServer до AutoMapperProfile

Повний код AutoMapperProfile можна знайти у **додатку В.**

3.3.4. Репозиторій для налаштування доступу до БД

Так як проєкт має досить велику кількість моделей в базі даних, то створення для кожної окремого репозиторія було би дуже ресурсозатратним. Саме тому було прийнято рішення використати паттерн розробки Generic Repository. [22]

Іншими перевагами цього патерну є проміжний рівень абстракції між бізнес логікою програми та рівнем доступу до даних, джерело даних можна змінити за допомогою кількох стрічок коду, полегшене модульне тестування. Повний код класу репозиторію розміщений у **додатку Г.**

Поруч з цим паттерном було використано два інших паттерни - Unit of work та Specifications

Паттерн Unit of Work (Одиниця Роботи) — це паттерн проектування, який використовується для ефективного взаємодії з базою даних та керування транзакціями в програмах. Його призначення полягає в тому, щоб згрупувати ряд

операцій бази даних в одну транзакцію та визначати контекст виконання цих операцій.

Паттерн "Specifications" використовується в об'єктно-орієнтованому програмуванні для опису логічних умов, які мають виконуватися об'єктами. Головна мета цього паттерну - надати засіб для конструювання складних запитів або умов для визначення, чи задовольняє об'єкт певним критеріям.

Основні призначення патерну "Specifications":

1. Конструювання складних запитів.

Specifications дозволяють компонувати прості умови в складніші та структуровані запити. Це особливо корисно в сферах, де потрібно створювати запити з декількох умов або враховувати різні критерії.

2. Покращення читабельності коду.

Використання Specifications може покращити читабельність коду, оскільки логічні умови виражаються як окремі об'єкти, а не вкладені умовні оператори. Це робить код більш зрозумілим та легше підтримуваним.

3. Відокремлення логіки пошуку:

Паттерн Specifications дозволяє відокремити логіку пошуку від логіки бізнес-логіки. Це забезпечує вищий рівень абстракції та розділення відповідальностей між компонентами системи.

4. Підтримка повторного використання коду:

За допомогою Specifications можна створювати набори логічних умов, які можна повторно використовувати у різних частинах програми. Це сприяє підтримці і уникненню дублювання коду.

5. Розширюваність:

Паттерн "Specifications" забезпечує розширюваність системи, оскільки нові умови можуть бути додані без значних змін у вже існуючому коді.

Загалом, використання паттерну Specifications дозволяє робити код більш гнучким, підтримуваним та зручним для розширення, особливо в областях, де важлива динамічність та складність умов пошуку. Код використаного патерну специфікацій можна переглянути у **додатку Д.**

3.3.5. Функціонал корзини

Для того, щоб користувач міг зручно зберігати події, які він хоче придбати, було написано функціонал для корзини, де вони і будуть міститися після додавання. Важливим моментом було запобігти створенню кількох корзин для користувача. Тому що у разі дублювання даних відбуватиметься зайва витрата ресурсів, а у разі втрати прогресу, який здійснив користувач для додавання тих чи інших подій в корзину, можна отримати негативні відгуки та емоції від клієнтів.

Код, призначенням якого є додавання елементу до корзини, в якому також передбачено захист від створення кількох корзин для одного користувача, вказано в лістингу 3.9

```
        public async Task<ActionResult<BasketDto>> AddBasketItem(int eventId, string userId)
{
    var basket = await RetrieveBasket(userId);
    if (basket == null) basket = await CreateBasket(userId);

    var eventToAdd = _unitOfWork.EventRepository.GetById(eventId);
    var basketItemCheck = basket.BasketItems.FirstOrDefault(x => x.EventId == eventId);

    if (basketItemCheck != null)
        throw new HttpException("Подія вже у вашому кошику", HttpStatusCode.BadRequest);

    if (eventToAdd == null)
        throw new HttpException("Invalid event id", HttpStatusCode.BadRequest);

    await _unitOfWork.BasketItemRepository.Insert(new BasketItem
    {
    BasketId = basket.Id,
    EventId = eventId
    });

    await _unitOfWork.SaveAsync();
    var basketFromDb = await RetrieveBasket(userId);
    var basketToSend = _mapper.Map<BasketDto>(basketFromDb);

    return basketToSend;
}
```

Лістинг 3.9. Додавання події в корзину

3.3.6. Налаштування пагінації

Критично важливим під час розробки бекенду нашого веб-додатку є забезпечення його швидкого реагування на запити. Тому було вирішено запобігти його перевантаженості шляхом обмеження кількості подій, які можна витягнути з бази даних одним запитом. Для цього до запитів типу GET було додано функціонал, який розбиває масив всіх подій в базі даних на шматки та повертає лише ту частину даних, яку вимагає клієнт. Цей процес прийнято називати терміном пагінація. [25]

```
public List<EventPreviewDto> GetEventsPreviews(int page, int pageSize)
        {
            var events = _unitOfWork.EventRepository.GetListBySpec(new
Events.WithUserAndEventAndLectorers(page, pageSize));
            events = events
                .Skip(((page - 1) * pageSize))
                .Take(pageSize);
            return _mapper.Map<List<EventPreviewDto>>(events);
        }
```

Лістинг 3.10. Приклад використання пагінації

Також для того, щоб клієнт міг заранння отримати обчислену кількість сторінок з подіями, в залежності від кількості подій на одній сторінці, було також додано функцію GetPagesQuantity.

```
public int GetPagesQuantity(int pageSize)
        {
            var totalCount = _unitOfWork.EventRepository.GetAll().Count();
            var totalPages = (int)Math.Ceiling((decimal)totalCount /
pageSize);
            return totalPages;
        }
```

Лістинг 3.11. Обчислення кількості сторінок

3.4 Керівництво користувача для використання написаного бекенду

Для тестування API, що було реалізовані під час написання наукової роботи було обрано Swagger.

Swagger - це набір інструментів для розробки програмного забезпечення, який дозволяє створювати, документувати та спілкуватися з веб-службами на основі архітектури RESTful. Основні компоненти Swagger включають специфікації OpenAPI, інструменти для автоматизованої генерації документації та клієнтських бібліотек для веб-служб.

Нижче в тексті буде наведено конкретні приклади надісланих запитів з отриманим результатом.

Реєстрація. “api/auth/register”

Введено дані:

{
"userName": "Test",
"email": "test@gmail.com",
"password": "wrongpasswordexample",
"confirmPassword": "wrongpasswordexample"
}

Очікуваний результат: сервер поверне помилку з вказівками щодо помилок, адже пароль не містить в собі жодної цифри і великої літери.

Фактичний результат:

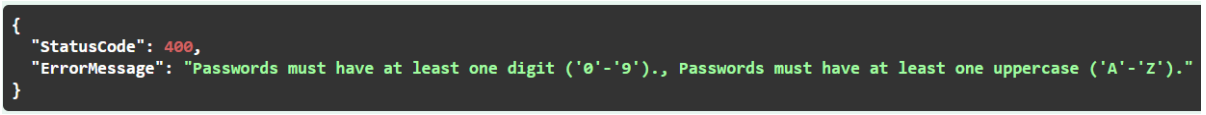

Рис. 3.7. Відповідь на запит реєстрації з некоректними даними

Джерело: створено автором

Введено дані:

{
"userName": "Test",
"email": "test@gmail.com",
"password": "CorrectPasswordexample1",
"confirmPassword": "CorrectPasswordexample1"
}

Очікуваний результат: сервер поверне код 200, тобто запит здійснено успішно.

Фактичний результат:

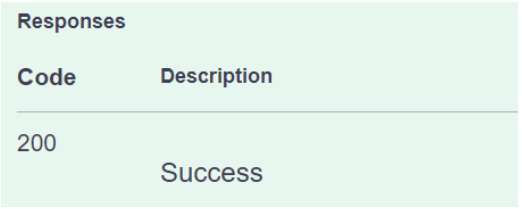

Рис. 3.8. Відповідь на запит реєстрації з коректними даними

Джерело: створено автором

Логін. “api/auth/login”

Введено дані:

```
{
  "email": "test@gmail.com",
  "password": "CorrectPasswordexample1",
  "rememberMe": true
}
```

Очікуваний результат: пошта не була підтверджена, тому сервер повинен видати помилку.

Фактичний результат:

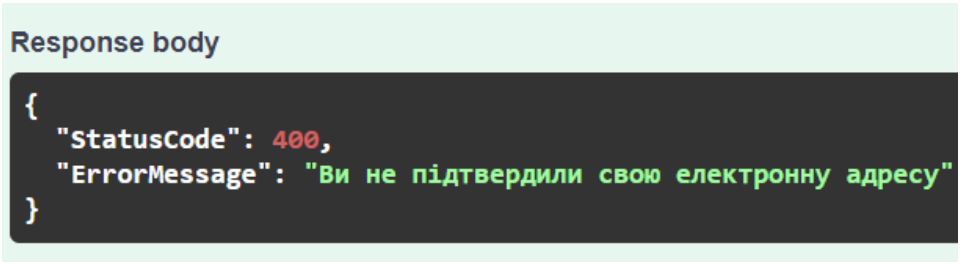

Рис. 3.9. Відповідь на запит логіну з не підтвердженою поштою

Джерело: створено автором

Логін. “api/auth/login”

Введено дані:

```
{
  "email": "admin@gmail.com",
  "password": "admin$Pass1",
  "rememberMe": true
}
```

Очікуваний результат: сервер поверне валідний токен JWT

Фактичний результат:

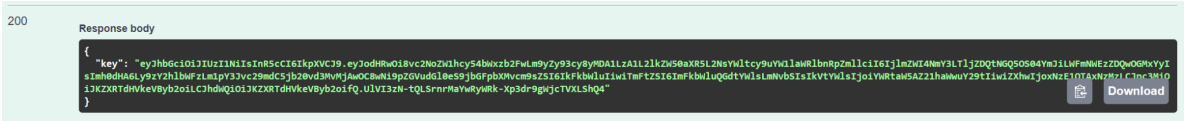

Рис. 3.10. Відповідь на запит логіну з підтвердженою поштою

Джерело: створено автором

Отримання подій з фільтрами. “api/event/parameters”

Введено дані:

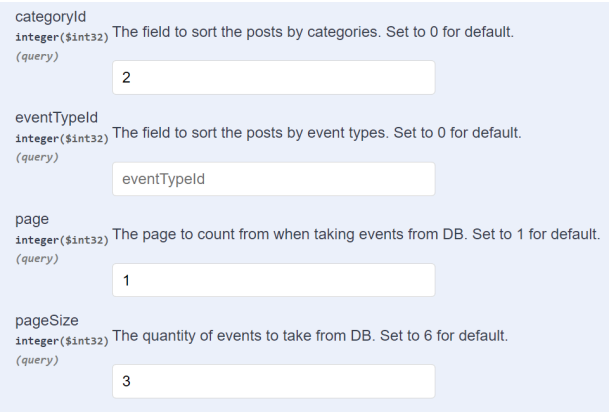

Рис. 3.11. Відповідь на запит логіну з підтвердженою поштою

Джерело: створено автором

Очікуваний результат: сервер поверне перші три події з категорії з ідентифікатором 2.

Фактичний результат:

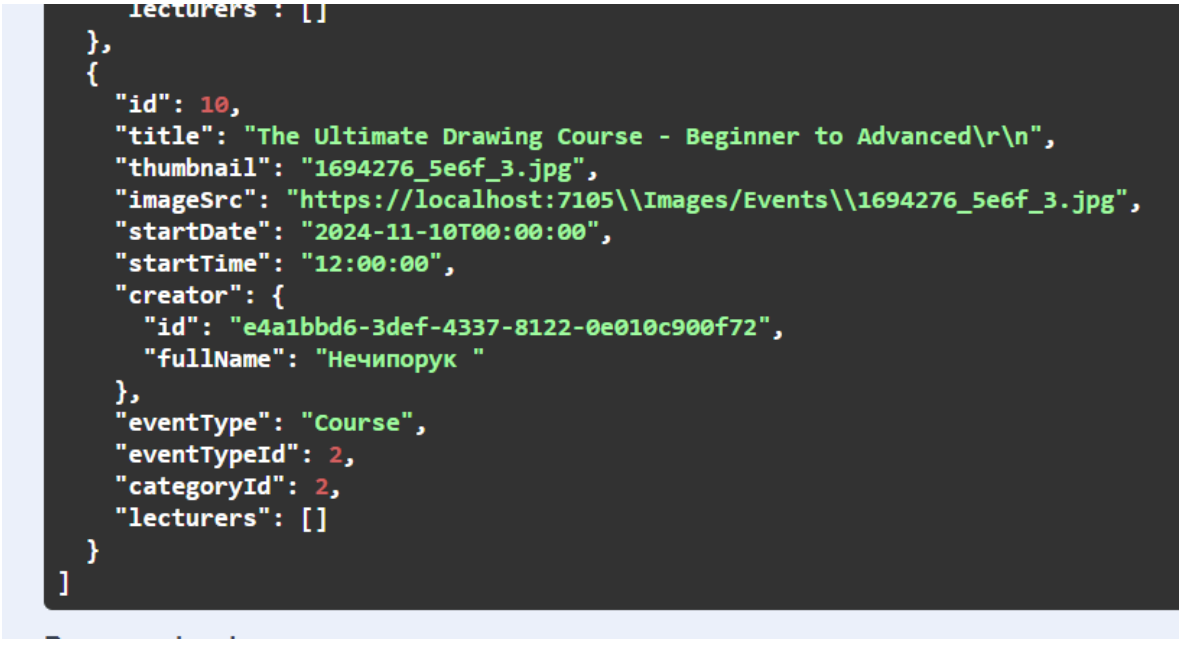

Рис. 3.12. Відповідь на запит, яка повернула 3 перші події з специфічною категорією

Джерело: створено автором

Створення події. “api/event/create”

Введено дані:

Валідний набір даних для створення події

Очікуваний результат: сервер поверне помилку 404, так як користувач не залогінений

Фактичний результат:

Рис. 3.13. Відповідь на запит без логіну

Джерело: створено автором

Логін. “api/auth/create”

Введено дані:

Валідний набір даних для створення події

Очікуваний результат: сервер поверне код 200

Фактичний результат:

Рис. 3.14. Відповідь на запит створення події з правами адміністратора

Джерело: створено автором

Таким чином, проведення ручного тестування для розробленого API довело, що всі функції працюють так, як було заплановано. Відповіді на запити успішно повертали текст оброблених помилок у випадках, в яких це потрібно, або ж повертали дані, які було запитано в серверної частини.

Висновки до розділу 3

В ході написання розділу 3 було успішно обрано стек технологій та бібліотек, завдяки яким стало можливим написати функціональну частину бекенду веб-додатку Jet Study Project. Microsoft Visual Studio слугувала середовищем для розробки. GitHub було використано для онлайн синхронізації версій додатку, як власних, так і інших розробників, що дозволило максимально швидко отримувати оновлення та валідний стан проєкту. Використання Swagger виконало всі очікування, адже з ним вдалося відловити і захендлити помилки, які виникали в процесі розробки.

Для написання проєкту було використано ASP.NET WEB API фреймворк, який є зручним середовищем для програмування на мові C#. Внаслідок цього отримали проєкт, який можна розгорнути на віддаленому сервері і отримувати доступ до API в

мережі. Крім того, через інтеграцію Entity Framework та AutoMapper, а також кількох патернів проектування, було організовано чудову роботу з базою даних. В свою чергу за інтерфейс, в якому можна взаємодіяти з даними та зберігати їх там, відповідав MS SQL SERVER, який працював безвідмовно.

Було налаштовано та написано функціонал для авторизації та автентифікації користувача. Це стало можливим завдяки дослідженню теми JWT. Використання JSON Web Token-у дозволило валідувати користувачів при вході та використанню сервісу безпечно та надійно, а також надавало в payload властивості додаткові дані, які були корисними для клієнта, наприклад зберігання ролі користувача в ньому. Було створено 3 типи користувачів: Student, Instructor, Admin, кожен з яких, в залежності від своїх прав, мав доступ до певного функціоналу.

Також було організовано корисну роботу з подіями, а саме створено функціонал для отримання подій, пагінацію, фільтрування, стоврення, редагування, видалення, модерування та інші.

В кінці написання розділу було протестовано всі ендпоінти та відображено тести деяких із них у звіті. Це свідчить про те, що дану частину завдання виконано правильно.

ВИСНОВКИ

Метою написання даної наукової роботи було проєктування та створення серверної частини сервісу для продажу навчальних подій, який стане мостом між студенти та кваліфікованими інструкторами. Після інтеграції всіх написаних запитів у клієнтську частину вони зможуть робити обмін цінними знаннями та досвідом, набутими внаслідок їхньої праці та навчання. Таким чином, обидві сторони можуть отримувати вигоду як у інтелектуальному, так і фінансовому аспекті.

Під час написання кваліфікаційної роботи було розроблено серверну частину веб-додатку Jet Study Project. Процес написання був розділений на багато етапів, а саме огляд наявних аналогів, постановка задачі, створення use case діаграми, проєктування бази даних, вибір типу архітектури проєкту, стеку технологій та програмного забезпечення, а також здійснено безпосередню розробку продукту. Зупинимось на виконанні кожного етапу окремо, так як це допоможе зрозуміти всі необхідні кроки для написання серверної частини додатку для продажу навчальних подій, або ж для написання бекенду додатку для платформ, схожих за призначенням.

Необхідно обрати та проаналізувати наявних постачальників послуг, які функціонують у сфері, яка найближча до вашого проєкту. У уьому конкретному випадку було проаналізовано такі сервіси, як Udemy, Coursera та Skillshare. Таким чином, далі було виділено наступні основні пункти, які користувачі цінують у вказаних сервісах, а також ті, які необхідно було забезпечити в проєкті: розмаїття та якість контенту, зручне та гнучке застосування сервісу, взаємодія з успішними викладачами.

Наступним кроком необхідно встановити список задач, які необхідно розв’язати. Було постановлено реалізувати наступні задачі, які будуть задовольняти цінності користувачів: реєстрація та автентифікація, редагування профілю, відновлення паролю, перегляд майбутніх подій а також їхній детальний огляд, реалізація оплати за курси.

Далі для запобігання виникнення майбутніх непорозумінь між командою, варто створити UML діаграму, яка буде висвітлювати загальне бачення на розробку проєкту кожного з його учасників. У кваліфікаційній роботі було обрано саме use case діаграму, яка за своїми властивостями найкраще підходить для теми дослідження. Таким чином, було встановлено 2 основних типи користувачі, а саме Student та Instructor. Функціонал та можливості Instructora, звісно ж, є ширшими, адже він є тим самим наповнювачем нашої бази даних, тобто продавцем своїх курсів.

Обов’язково в проєктах такого типу необхідно розробити схему бази даних перед початком безпосередньої розробки рішення. Схему БД було отримано за допомогою реляційного підходу і нормалізації даних, а створено за допомогою сучасної та потужної ORM Entity Framework, яка підтримує code first approach. Таким чином, не потрібно робити надмірно багато роботи над ручним прописуванням SQL запитів до бази даних, адже всю роботу з перетворенням класів на сутності в БД було зроблено автоматично, з урахуванням прописаних налаштувань через Fluent API.

Після успішного виконання усіх попередніх етапів здійснюється написання потрібного функціоналу, який і є серверною частиною веб-додатку. Під час написання кваліфікаційної роботи було здійснено безпосереднє написання коду, який забезпечує отримання функціоналу серверної частини. Було налаштовано автентифікацію та авторизацію на сервісі з використанням JWT токенів, які є надзвичайно корисними в проєктах даного типу. Таким чином, клієнтська частина веб-додатку мала завжди актуальні дані про стан користувача в конкретний момент часу. Було налаштовано EmailSender, який працює з поштовими сервісами за допомогою сервісу SMTP. В результаті цього користувач може отримувати розсилку листів на свою пошту, що є додатковим захистом у разі втрати паролю до свого аккаунту, або взлому. Також це є захистом від спаму зловмисників, адже під час реєстрації дані пошти потрібно підтверджувати.

Було реалізовано отримання користувачем подій з гнучким пошуком та зручною фільтрацією. Запити, які відповідають за цей функціонал також

використовують пагінацію, з якою клієнтська частина додатку може налаштувати ліниве завантаження даних, нескінченний скрол, або звичайні сторінки. Інструктор, в свою чергу, окрім попереднього функціоналу може створювати, редагувати та робити інші маніпуляції з курсами, які покращують якість надання послуг.

Критично важливо, щоб останнім кроком було тестування функціоналу серверної частини. В останньому розділі наукової роботи було виконано тестування ендпоінтів бібліотекою Swagger. Вона є життєво необхідною для перевірки роботи RESTful сервісів, а саме їх запитів. Кожен запит, функціонал якого було написано і задокументовано пройшов ретельну перевірку та відпрацьовував так, як очікувалося розробником.

Таким чином, поставлену початкову мету та допрацьований список задач, які було поставлено для створення було успішно виконано та приведено в дію. Дану кваліфікаційну роботу рекомендується використовувати як зразок побудови бекенду, що має надавати для клієнтської частини прикладний програмний інтерфейс, а також як приклад використання сучасних технологій, які було застосовано під час виконання даної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Udemy. URL: https://www.udemy.com/ (дата звернення: 14.01.2024 р.).
2. Офіційний сайт Coursera. URL: https://www.coursera.org/ (дата звернення: 14.01.2024 р.).
3. Офіційний сайт Skillshare. URL: https://www.skillshare.com/en/ (дата звернення: 14.01.2024 р.).
4. Unified Modeling Language. URL: https://shorturl.at/foLFM (дата звернення: 19.01.2024 р.).
5. Use Case Diagram. URL: https://shorturl.at/5VEf1 (дата звернення: 19.01.2024 р.).
6. Бази даних. URL: https://dou.ua/lenta/articles/types-of-databases/ (дата звернення: 02.02.2024 р.).
7. Реляційні бази даних. URL: https://foxminded.ua/reliatsiini-bazy-danykh/ (дата звернення: 02.02.2024 р.).
8. Entity Framework Code First: with Migrations. URL: https://shorturl.at/AVwqI (дата звернення: 06.02.2024 р.).
9. What are database migrations. URL: https://shorturl.at/nqu5Q (дата звернення: 08.02.2024 р.).
10. Implement Entity Framework A Code First Approach in .Net 8 API URL: https://shorturl.at/piGNM (дата звернення: 08.02.2024 р.).
11. Entity Framework Core. URL: https://learn.microsoft.com/ru-ru/ef/core/ (дата звернення: 16.02.2024 р.).
12. Visual Studio - продукт Microsoft. URL: https://visualstudio.microsoft.com/ (дата звернення: 06.02.2024 р.).
13. Common C# code conventions. URL: https://t.ly/uYuCk (дата звернення: 18.02.2024 р.).
14. Clean architecture layers. URL: https://t.ly/SwePN (дата звернення: 24.02.2024 р.).
15. The Clean Architecture - Beginner’s Guide. URL: https://t.ly/OKjLW (дата звернення: 24.02.2024 р.).

16. Connect GitHub. URL: https://docs.render.com/github (дата звернення: 25.02.2024 р.).
17. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення: 02.03.2024 р.).
18. Бібліотека .NET з відкритим кодом для IMAP, POP3 і SMTP. URL: https://t.ly/acdGA (дата звернення: 05.03.2024 р.).
19. Authentication vs Authorization – What's the Difference? URL: http://surl.li/uaxxz (дата звернення: 13.03.2024 р.).
20. OAuth 2.0. URL: https://t.ly/7p_oT (дата звернення: 15.03.2024 р.).
21. AutoMapper documentation. URL: https://docs.automapper.org/en/stable/ (дата звернення: 18.03.2024 р.).
22. Pattern Repository. URL: https://uk.wikipedia.org/wiki/Repository (дата звернення: 20.03.2024 р.).
23. Основні етапи проектування бази даних. URL: https://shorturl.at/omR9Q (дата звернення: 21.03.2024 р.).
24. APIs with ASP.NET Core. URL: https://rb.gy/mzguas (дата звернення: 17.02.2024 р.).
25. Pagination in .Net Api. URL: https://rb.gy/967s7w (дата звернення: 23.03.2024 р.).
26. Що таке Swagger? URL: https://qagroup.com.ua/publications/what-is-swagger/ (дата звернення: 25.03.2024 р.).
27. Exception Middleware in .NET Core Applications. URL: https://rb.gy/kgcgnj (дата звернення: 28.03.2024 р.).

ДОДАТОК А

Сутність Event

Лістинг програми

Аркушів 1

Острог 2024

```
public class Event
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public string? Title { get; set; }
    public string? Thumbnail { get; set; }
    public DateTime? StartDate { get; set; }
    public DateTime? EndDate { get; set; }
    public TimeOnly StartTime { get; set; }
    public TimeOnly EndTime { get; set; }
    public double? Price { get; set; }
    public string? ShortDescription { get; set; }
    public string? LongDescription { get; set; }
    public string? TargetedAudience { get; set; }
    public int? SeatsAvailable { get; set; }
    public string? Location {  get; set; }
    public string? AdditionalRecources {  get; set; }
    public bool? IsOnline { get; set; }
    public User Creator { get; set; }
    public string CreatorId {  get; set; }
    public Category? Category { get; set; }
    public int CategoryId {  get; set; }
    public EventType? EventType { get; set; }
    public int EventTypeId { get; set; }
    public StatusForInstructor? StatusForInstructor { get; set; }
    public int StatusForInstructorId { get; set; }
    public StatusForAdministrator? StatusForAdministrator { get; set; }
    public int StatusForAdministratorId { get; set; }
    public StatusForStudent? StatusForStudent { get; set; }
    public int StatusForStudentId { get; set; }
    public virtual ICollection<ReadCourse> Lecturers { get; set; }
    public virtual ICollection<ListenCourse> Students { get; set; }
    public virtual ICollection<ApplicationToEvent> ApplicationToEvents { get; set; }
}
```

ДОДАТОК Б

AuthService

Лістинг програми

Аркушів 4

Острог 2024

```
public class AuthService : IAuthService
{
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    private readonly RoleManager<IdentityRole> _roleManager;
    private readonly IConfiguration _configuration;
    private readonly IUrlHelper _helper;
    public AuthService(UserManager<User> userManager, SignInManager<User> signInManager,
RoleManager<IdentityRole> roleManager, IConfiguration configuration, IUrlHelper helper)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _roleManager = roleManager;
        _configuration = configuration;
        _helper = helper;
    }

    public async Task Register(UserRegisterDto userRegisterDto)
    {
        var user = new User()
        {
            Email = userRegisterDto.Email,
            UserName = userRegisterDto.UserName
        };

        var result = await _userManager.CreateAsync(user, userRegisterDto.Password);

        if (!result.Succeeded)
        {
            string errors = string.Join(", ", result.Errors.Select(e => e.Description));
            throw new HttpException(errors, HttpStatusCode.BadRequest);
        }
        var userInDb = await _userManager.FindByEmailAsync(userRegisterDto.Email);

        var role = _roleManager.Roles.FirstOrDefault(x => x.NormalizedName == "STUDENT");
        if (role == null)
        {
            await _roleManager.CreateAsync(new IdentityRole("Student"));
        }
        await _userManager.AddToRoleAsync(userInDb, "Student");
        var confirmationCode = await
_userManager.GenerateEmailConfirmationTokenAsync(userInDb);
        var callbackUrl = _helper.Action(
                "ConfirmEmail",
```

```
                "Auth",
                new { userId = user.Id, code = confirmationCode },
                protocol: _helper.ActionContext.HttpContext.Request.Scheme);
        EmailService emailService = new EmailService(_configuration);
        await emailService.SendEmailAsync(userRegisterDto.Email, "Confirm your account",
            $"Перейдіть за посиланням, щоб підтвердити аккаунт: <a
href='{callbackUrl}'>link</a>");
    }

    public async Task<LoginAnswerDto> Login(UserLoginDto userLoginDto)
    {
        var user = await _userManager.FindByEmailAsync(userLoginDto.Email);

        if (user != null)
        {
            if (!await _userManager.IsEmailConfirmedAsync(user))
            {
                throw new HttpException("Ви не підтвердили свою електронну адресу",
HttpStatusCode.BadRequest);
            }
        }

        if (user == null || !await _userManager.CheckPasswordAsync(user, userLoginDto.Password))
        {
            throw new HttpException(ErrorMessages.InvalidCredentials, HttpStatusCode.BadRequest);
        }

        await _signInManager.SignInAsync(user, userLoginDto.RememberMe);

        return new LoginAnswerDto { Key = await GenerateTokenAsync(user) };
    }

    public async Task<LoginAnswerDto> LoginWithGoogle(string credential)
    {
        var settings = new GoogleJsonWebSignature.ValidationSettings()
        {
            Audience = new List<string> { _configuration["Google:GoogleClientId"] }
        };

        var payload = await GoogleJsonWebSignature.ValidateAsync(credential, settings);

        var user = await _userManager.FindByEmailAsync(payload.Email);

        if (user != null)
```

```
        {
            if (!await _userManager.IsEmailConfirmedAsync(user))
            {
                throw new HttpException("Ви не підтвердили свою електронну адресу",
HttpStatusCode.BadRequest);
            }

            return new LoginAnswerDto { Key = await GenerateTokenAsync(user) };
        }
        else
        {
            throw new HttpException("Користувача з такою електронною адресою не існує",
HttpStatusCode.BadRequest);
        }
    }

    private async Task<string> GenerateTokenAsync(User user)
    {
        var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
        var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
        var roles = await _userManager.GetRolesAsync(user);
        string role;
        if (roles.Count > 0)
        {
            role = roles[0];
        }
        else
        {
            role = "null";
        }
        var claims = new[]
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id),
            new Claim(ClaimTypes.Role, role),
            new Claim("Name", user.UserName),
            new Claim("Email", user.Email)
        };

        var token = new JwtSecurityToken(
            _configuration["Jwt:Issuer"],
            _configuration["Jwt:Audience"],
            claims,
            expires:
```

```
DateTime.Now.AddMinutes(Int32.Parse(_configuration.GetSection("Jwt:Expires").Value)),
            signingCredentials: credentials);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }

    public async Task ConfirmEmail(string userId, string code)
    {
        if (userId == null || code == null)
        {
            throw new HttpException("Bad Request", HttpStatusCode.BadRequest);
        }
        var user = await _userManager.FindByIdAsync(userId);
        if (user == null)
        {
            throw new HttpException("There is no such user", HttpStatusCode.BadRequest);
        }
        var result = await _userManager.ConfirmEmailAsync(user, code);
        if (!result.Succeeded)
            throw new HttpException("The error has occured", HttpStatusCode.BadRequest);
    }

    public async Task Logout()
    {
        await _signInManager.SignOutAsync();
    }
}
```

ДОДАТОК В

AutoMapperProfile

Лістинг програми

Аркушів 2

Острог 2024

```
public class AutoMapperProfile : Profile
{
   public AutoMapperProfile(IServer _server)
   {
      CreateMap<User, UserDto>()
         .ForMember(dest => dest.FullName, from => from.MapFrom(x => x.Name + " " +
x.Surname));
      CreateMap<ReadCourse, UserDto>()
         .ForMember(dest => dest.FullName, from => from.MapFrom(x => x.User.Name + " " +
x.User.Surname))
         .ForMember(dest => dest.Id, from => from.MapFrom(x => x.UserId));

      CreateMap<Event, EventFullDto>()
         .ForMember(dest => dest.EventType, from => from.MapFrom(x => x.EventType.Title))
         .ForMember(dest => dest.Lecturers, from => from.MapFrom(x => x.Lecturers.ToList()))
         .ForMember(dest => dest.ImageSrc, from => from.MapFrom(x =>
Path.Combine(_server.Features.Get<IServerAddressesFeature>().Addresses.FirstOrDefault(),
WebConstants.eventsImagesPath, x.Thumbnail)));

      CreateMap<EventFullDto, Event>();
      CreateMap<EventCreateDto, Event>();
      CreateMap<EventEditDto, Event>()
      .ForAllMembers(opts =>
      {
         opts.AllowNull();
         opts.Condition((src, dest, srcMember) => srcMember != null);
      });


      CreateMap<Event, EventPreviewDto>()
         .ForMember(dest => dest.EventType, from => from.MapFrom(x => x.EventType.Title))
         .ForMember(dest => dest.Lecturers, from => from.MapFrom(x => x.Lecturers.ToList()))
         .ForMember(dest => dest.ImageSrc, from => from.MapFrom(x =>
Path.Combine(_server.Features.Get<IServerAddressesFeature>().Addresses.FirstOrDefault(),
WebConstants.eventsImagesPath, x.Thumbnail)));

      CreateMap<Category, CategoryDto>();
      CreateMap<CategoryDto, Category>();

      CreateMap<EventType, EventTypeDto>();
      CreateMap<EventTypeDto, EventType>();

      CreateMap<BasketItem, BasketItemDto>()
         .ForMember(dest => dest.Title, from => from.MapFrom(x=> x.Event.Title))
```

```
            .ForMember(dest => dest.Price, from => from.MapFrom(x => x.Event.Price))
            .ForMember(dest => dest.ImageSrc, from => from.MapFrom(x =>
Path.Combine(_server.Features.Get<IServerAddressesFeature>().Addresses.FirstOrDefault(),
WebConstants.eventsImagesPath, x.Event.Thumbnail)));
        CreateMap<BasketItemDto, BasketItem>();

        CreateMap<Basket, BasketDto>()
            .ForMember(dest => dest.Items, from => from.MapFrom(x => x.BasketItems.ToList()));
    }
}
```

ДОДАТОК Г

Патерн репозиторій

Лістинг програми

Аркушів 2

Острог 2024

```
public class GenericRepository<TEntity> : IGenericRepository<TEntity> where
TEntity : class
{
    private readonly DataContext _ctx;
    private readonly DbSet<TEntity> _dbSet;

    public GenericRepository(DataContext ctx)
    {
        _ctx = ctx;
        _dbSet = _ctx.Set<TEntity>();
    }

    public void Delete(TEntity obj)
    {
        if (_ctx.Entry(obj).State == EntityState.Detached)
        {
            _dbSet.Attach(obj);
        }
        _dbSet.Remove(obj);
    }

    public void Delete(object id)
    {
        TEntity entityToDelete = _dbSet.Find(id);
        Delete(entityToDelete);
    }

    public IEnumerable<TEntity> GetAll(
       Expression<Func<TEntity, bool>> filter = null,
       Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
       params string[] includeProperties)
    {
        IQueryable<TEntity> query = _dbSet;

        if (filter != null)
        {
            query = query.Where(filter);
        }

        foreach (var includeProperty in includeProperties)
        {
            query = query.Include(includeProperty);
        }

        if (orderBy != null)
        {
            return orderBy(query).ToList();
```

```
        }
        else
        {
            return query.ToList();
        }
    }

    public TEntity GetById(object id)
    {
        return _dbSet.Find(id);
    }

    public async Task Insert(TEntity obj)
    {
        await _dbSet.AddAsync(obj);
    }

    public void Update(TEntity obj)
    {
        _dbSet.Attach(obj);
        _ctx.Entry(obj).State = EntityState.Modified;
    }

    public IEnumerable<TEntity> GetListBySpec(ISpecification<TEntity>
specification)
    {
        return ApplySpecification(specification).ToList();
    }

    public TEntity? GetFirstBySpec(ISpecification<TEntity> specification)
    {
        return ApplySpecification(specification).FirstOrDefault();
    }

    private IQueryable<TEntity> ApplySpecification(ISpecification<TEntity>
specification)
    {
        var evaluator = new SpecificationEvaluator();
        return evaluator.GetQuery(_dbSet, specification);
    }
}
```

ДОДАТОК Д

Патерн Specifications

Лістинг програми

Аркушів 1

Острог 2024

```
public static class Events
{
    public class ByEventIdWithUserAndEventAndLectorers : Specification<Event>
    {
        public ByEventIdWithUserAndEventAndLectorers(int eventId)
        {
            Query
                .Where(x => x.Id == eventId)
                .Include(x => x.Creator)
                .Include(x => x.Lecturers)
                .ThenInclude(x => x.User)
                .Include(x => x.ApplicationToEvents)
                .Include(x => x.EventType);
        }
    }
    public class WithUserAndEventAndLectorers : Specification<Event>
    {
        public WithUserAndEventAndLectorers(int page, int pageSize)
        {
            Query
                .Include(x => x.Creator)
                .Include(x => x.Lecturers)
                .ThenInclude(x => x.User)
                .Include(x => x.ApplicationToEvents)
                .Include(x => x.EventType);
        }
    }
}
```