МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ ОСНОВ ПРОГРАМУВАННЯ»**

Виконав: студент 4 курсу, групи 6.1220
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Б. Рахауі
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук, Циммерман Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я А

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Білалові Рахауі
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка вебзастосунку для самостійного вивчення основ програмування

керівник роботи Циммерман Геннадій Анатолійович
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 05.06.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розділи Вступ, Аналіз потреб клієнта, Управління та планування проєкту, Вибір методології, Аналіз та специфікація вимог, Проєктування структури БД, Реалізація. та тестування проєкту

4. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.01.2024	
2.	Збір вихідних даних.	01.02.2024	
3.	Обробка методичних та теоретичних	01.03.2024	
	джерел.		
4.	Розробка розділів 1-6.	01.04.2024	
5.	Реалізація системи. Розробка розділів 7-8	01.05.2024	
6.	Оформлення та нормоконтроль	15.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	23.06.2024	

Студент _____ Б. Рахауі
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Циммерман
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебзастосунку для самостійного вивчення основ програмування»: 109 с., 113 рис., 5 табл., 10 джерел.

ВЕБЗАСТОСУНОК, ДИНАМІЧНИЙ КОНТЕНТ, КУРСИ ПРОГРАМУВАННЯ, ОСНОВИ ПРОГРАМУВАННЯ, РОЗРОБКА БЕКЕНДУ, РОЗРОБКА ФРОНТЕНДУ, САМОСТІЙНЕ ВИВЧЕННЯ.

Об’єкт дослідження – розробка вебзастосунку для навчання основ програмування, що забезпечує інтерактивний та гнучкий процес самостійного навчання.

Мета роботи: створення вебзастосунку, який відповідає сучасним стандартам освіти та забезпечує ефективне засвоєння знань з основ програмування.

Методи дослідження – системний аналіз, проектування, програмування та тестування.

У роботі описано процес розробки вебзастосунку CodeCraftingLab, який заснований на технологіях Next.js, TailwindCSS, MongoDB, Nodemailer. Особливу увагу приділено реалізації інтерактивних курсів з відеоматеріалами та квізами для оцінки знань користувачів, а також розробці функціоналу для відстеження прогресу навчання.

Застосунок розроблено з використанням хмарних сервісів Amazon Web Services, що забезпечує його надійність та масштабованість.

Результатом роботи є готовий до використання вебзастосунок CodeCraftingLab, який пропонує зручний і ефективний інструмент для самостійного вивчення основ програмування.

SUMMARY

Bachelor’s Qualifying Theses "Development of a Web Application for Self-Learning Programming Fundamentals": 109 pages, 113 figures, 5 tables, 10 references.

WEB APPLICATION, DYNAMIC CONTENT, PROGRAMMING COURSES, PROGRAMMING BASICS, BACKEND DEVELOPMENT, FRONTEND DEVELOPMENT, SELF STUDY.

Object of the study: Development of a web application for learning programming fundamentals that facilitates an interactive and flexible self-learning experience.

Aim of the study: To create a web application that meets contemporary educational standards and ensures effective knowledge acquisition of programming basics.

Method of research: Systematic analysis, design, programming, and testing.

The thesis describes the development process of the CodeCraftingLab web application, built on Next.js, Tailwind CSS, MongoDB, and Nodemailer technologies. Emphasis is placed on implementing interactive courses with video materials and quizzes for evaluating user knowledge, as well as developing functionalities for tracking learning progress.

The application utilizes Amazon Web Services for cloud-based hosting, ensuring reliability and scalability.

The project resulted in a fully functional CodeCraftingLab web application, offering a user-friendly and effective tool for self-learning the fundamentals of programming.

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій навички програмування стали ключовим фактором вдосконалення бізнесу. У зв’язку з цим вивчення основ програмування шляхом самостійного вивчення стало важливою стратегією для людей, які прагнуть досягти успіху в сфері інформаційних технологій.

Цей проект присвячений розробці та впровадженню вебзастосунків, спрямованих на самостійне вивчення основ програмування. Загальна мета — створити платформу, яка забезпечує гнучке та оперативне навчальне середовище, ефективно адаптоване до конкретних потреб користувачів та технічних характеристик.

Використання передових технологій є найбільш важливим для забезпечення якості та зручності використання програми. Зокрема, Next.js використовується для створення швидкого та ефективного вебінтерфейсу, MongoDB служить основою для надійного зберігання даних, Amazon Web Services (AWS) забезпечує гнучкість і масштабованість, а NodeMailer дозволяє безперешкодно відправляти повідомлення.

Процес розробки зосереджений на створенні інтуїтивно зрозумілого та зручного для користувача інтерфейсу, оптимізованого для процесу навчання. Крім того, розроблені функціональні можливості, що полегшують навчання за допомогою текстових курсів, доповнених відео та вікторинами для комплексної оцінки знань.

У роботі проведено дослідження механізму взаємодії з базою даних MongoDB для забезпечення надійності та ефективності зберігання даних. Результат перелічених зусиль – створення вебзастосунку CodeCraftingLab-універсальної платформи, призначеної для самостійного вивчення основ програмування за допомогою текстових курсів, доповнених відеороликами та інтерактивними завданнями.

Цей додаток є ідеальним інструментом для тих, хто має намір набути навичок програмування, які є основою для подальшого успіху в динамічній IT-індустрії. Незалежно від того, чи є вони початківцями розробниками або досвідченими професіоналами, які прагнуть розширити свій набір навичок, CodeCraftingLab надасть надійну навчальну платформу для розвитку їх кар’єри.

1 ВСТУП ДО ПРОГРАМУВАННЯ ТА ЙОГО ВАЖЛИВІСТЬ

У сучасному світі інформаційних технологій, навчання програмуванню стає ключовим елементом кар’єрного зростання і успішного виконання різних ІТ-завдань. Програмування – це не тільки інструмент для створення програм і розробки технічних рішень, але і визначає нашу взаємодію з сучасним світом: від мобільних додатків до великих веб-систем, від штучного інтелекту до блокчейн-технологій.

1.1 Значення програмування в сучасному світі

Програмування – це не тільки засіб створення програм і розробки технічних рішень. Це живильне джерело для розвитку цифрового світу та впровадження новітніх технологій. Інженери програмного забезпечення створюють програми, які відображають наші потреби, спрощують наше життя та сприяють зростанню економіки.

Вміння програмувати відкриває двері для інновацій та створення нових можливостей для покращення якості життя.

1.2 Важливість самостійного вивчення програмування

Самостійне вивчення програмування необхідно для розвитку компетенцій і навичок, які є ключовими в сучасній індустрії. Крім того, це важливий інструмент для саморозвитку.

Здатність самостійно вчитися та здобувати нові знання має вирішальне значення для успіху в будь-якій галузі. Завдяки самостійному навчанню людина розкриває свій власний потенціал, розширює кругозір і розвиває

творче і критичне мислення.

1.3 Роль веб-додатка для самостійного вивчення програмування

Завдання розробки веб-додатку для самонавчання програмуванню є важливим кроком у забезпеченні доступу до якісної освіти для всіх. Використання сучасних технологій та інтерактивних методів навчання дозволяє створити ефективну платформу для розвитку навичок програмування, що відповідає вимогам сучасного ринку праці і дозволяє будь-якому бажаючому навчитися програмувати.

2 САМОСТІЙНЕ ВИВЧЕННЯ ПРОГРАМУВАННЯ: ВИКЛИКИ ТА ПЕРЕВАГИ

Самостійне вивчення програмування є важливим етапом у розвитку компетентностей та навичок, необхідних у сучасній індустрії. Цей процес дозволяє розвивати креативні навички та критичне мислення, а також адаптуватися до змін у технологічному середовищі.

2.1 Виклики самостійного вивчення програмування

Самостійне вивчення програмування може бути складним і ставити перед тими хто навчається ряд викликів.

2.1.1 Обсяг інформації

Поле програмування постійно розвивається, і для того, щоб стати компетентним в цій галузі, необхідно засвоїти велику кількість інформації. Починаючи з основ програмування і закінчуючи складними алгоритмами та технологіями, студентам доведеться вивчати широкий спектр матеріалів.

2.1.2 Самостійний пошук ресурсів

Одним із викликів самостійного вивчення програмування є необхідність пошуку відповідних та надійних ресурсів для навчання. Інтернет пропонує безліч матеріалів, але вибір правильних джерел та оцінка їхньої якості може бути складною задачею.

2.1.3 Розв’язання проблем

У процесі навчання програмування студентам часто доводиться стикатися з різноманітними проблемами та помилками. Вирішення цих проблем вимагає від них аналітичного мислення та навичок пошуку рішень.

2.2 Переваги самостійного вивчення програмування

Хоча самостійне вивчення програмування може бути викликом, воно також має численні переваги.

2.2.1 Гнучкість у темпі навчання

Студенти можуть розвивати власний темп навчання, працюючи з матеріалами у зручний для них час і темп.

2.2.2 Вибір матеріалів

Самостійне вивчення дозволяє студентам вибирати та використовувати матеріали, які найбільше відповідають їхнім потребам та стилю навчання.

2.2.3 Розвиток самостійних навичок

У процесі самостійного вивчення студенти отримують навички самостійної роботи та вирішення проблем, що є важливими у будь-якій професійній діяльності, зокрема у програмуванні.

3 АНАЛІЗ ПОТРЕБ КЛІЄНТА ТА НЕТЕХНІЧНІ СПЕЦИФІКАЦІЇ

3.1 Вираження потреб клієнта

3.1.1 Початкове вираження потреб

Клієнт потребує розробки вебзастосунку для самостійного вивчення основ програмування.

3.1.2 Складність початкового вираження потреб

Оскільки ускладнюється можливість уточнення потреб через пряме спілкування з клієнтом, тому важливо інтерпретувати ці потреби якомога більш точно та функціонально. Для цього, крім своєї ролі розробника, ми будемо виконувати роль клієнта. Вираження потреб клієнта має ключове значення у проекті.

3.1.3 Інтерпретація вираження потреб

На основі вищезазначеного, сформулюємо таку версію потреб клієнта:

– користувацький інтерфейс: клієнт шукає користувацький інтерфейс, який спрощує процес навчання. Візуальна привабливість, разом з логічністю та інтуїтивною організацією, є ключовими для забезпечення того, що користувачі можуть легко навчатися через різні розділи освітнього контенту. Крім того, зручний інтерфейс сприяє приємнішому та доступнішому процесу навчання;
– гнучкий та доступний інтерфейс: простота та доступність є пріоритетом для клієнта. Застосунок повинен бути легкодоступним з різних пристроїв, щоб користувачі могли вибирати час та місце

для самостійного навчання. Ця гнучкість підвищує залученість користувачів до навчального процесу, сприяючи більш персоналізованому та зручному досвіду;

– комплексний контент: клієнт шукає застосунок, який пропонує комплексне покриття навчального напрямку основи програмування. Це вимога виходить за межі простого представлення фундаментальних концепцій, включаючи також специфіку синтаксису різних мов програмування. Різноманітність освітнього контенту допомагає користувачам отримати міцну та універсальну основу;
– міцний зв’язок з різними підходами до самонавчання: клієнт шукає застосунок, який надає можливість навчатися за різними підходами. Оскільки всі користувачі різні, вони мають різні потреби щодо підходу до навчання. Деякі можуть віддавати перевагу навчанню на основі текстового контенту, тоді як інші можуть віддавати перевагу відео контенту;
– регулярні оновлення та додаткові ресурси: клієнти наголошують на необхідності регулярних оновлень та додаткових ресурсів. Вони очікують динамічної платформи, яка залишається в актуальності з трендами галузі, надаючи додаткові ресурси, такі як інформативні статті, розширені уроки та посилання на відповідний зовнішній контент;
– сумісність та інтеграція: сумісність з популярними браузерами та безперешкодна інтеграція з іншими інструментами є ключовими вимогами. Клієнти хочуть безперешкодного користувацького досвіду, щоб платформа гармонійно взаємодіяла з їх цифровим робочим середовищем, усуваючи можливі перешкоди для використання.

У підсумку, глибоке врахування зазначених потреб клієнта створює міцний фундамент для розробки CodeCraftingLab. Відповідно до цих

очікувань, цей вебзастосунок має потенціал стати корисним ресурсом для програмістів, пропонуючи збагачений, персоналізований та галузево орієнтований досвід самонавчання.

3.2 Нетехнічні специфікації

На основі вираження потреб клієнта, ми пропонуємо нетехнічну специфікацію, яка відповідає кожній з висловлених клієнтом потреб.

3.2.1 Специфікації для панелі адміністрування

Аутентифікація та управління доступом: адміністратор отримує доступ до панелі адміністрування за допомогою авторизованих адрес електронної пошти.

Управління категоріями: адміністратори можуть створювати, редагувати та видаляти категорії згідно з потребами клієнта.

Управління курсами: адміністратори мають повний контроль над створенням, редагуванням та видаленням курсів, забезпечуючи постійне оновлення освітнього контенту.

Управління квізами: можливість створення квізів з програмування доступна адміністраторам, з функцією редагування для налаштування питань та відповідей. Непотрібні квізи також можуть бути видалені.

Резервне копіювання та відновлення: адміністратори можуть регулярно робити резервні копії бази даних, забезпечуючи безпеку даних. Крім того, відновлення з попередніх резервних копій можливе у разі потреби.

Додаткові зауваження.

Безпека: Усі взаємодії в панелі адміністрування є безпечними, з відповідними дозволами.

Аутентифікація: В панелі адміністрування реалізована безпечна система аутентифікації для адміністраторів.

Адаптивний дизайн: Забезпечується адаптивний дизайн для оптимального користувацького досвіду на різних пристроях.

Підтримувані мови: Обидві частини проекту підтримують принаймні одну основну мову (англійську).

3.2.2 Специфікації для основного застосунку

Інтуїтивна навігація: Клієнти можуть легко отримати доступ до категорій, курсів та квізів для плавної навігації.

Функціональність квізів: клієнт має можливість вибирати рівень складності та кількість питань перед початком квізу. Доступні різноманітні питання з програмування, з результатами, що відображаються негайно після завершення квізу.

Підтримка користувача: спеціальна форма дозволяє клієнту звертатися за підтримкою, він надає ім’я, електронну адресу та повідомлення для отримання необхідної допомоги.

4 УПРАВЛІННЯ ТА ПЛАНУВАННЯ ПРОЄКТУ

Управління проєктом відіграє важливу роль у реалізації програмних проєктів з кількох причин. По-перше, воно забезпечує системний підхід до планування, організації та реалізації проєктів, забезпечуючи ефективний розподіл ресурсів та ефективне керування проєктом для виконання строків. По-друге, це допомагає визначити внутрішню об’єктивність, зрозумілість та виконуваність, зменшує ризик затримок і забезпечує умову, що проєкт розвивається за планом, покращує комунікацію та сприяє співпраці між учасниками, зацікавленими сторонами та клієнтами, спілкування стає легшим, надаючи спільне розуміння цілей, прогресу та потенційних проблем. Це також забезпечує те, що всі залучені в процес розуміють один одного і можуть швидко вирішувати будь-які питання чи вносити зміни. По-третє, це дозволяє менеджерам проектів встановлювати стратегії ідентифікації та зменшення ризиків, тим самим зменшуючи вплив потенційних проблем на терміни та бюджет проєкту. Крім того, це дозволяє оптимізувати планування ресурсів та бюджету, ефективно використовуючи ресурси та забезпечуючи, що проєкт залишається в межах бюджетних обмежень. Нарешті, ефективне управління проектом забезпечує видимість прогресу проєкту, зміни, якщо вони відбуваються з часом за потреби, що в кінцевому підсумку оптимізує проєкт відповідно до очікувань зацікавлених сторін.

4.1 Гнучка методологія: SCRUM

У нашому проекті використано методологію SCRUM разом з JIRA як ефективний інструмент, який дозволяє вирішувати різноманітні завдання, що стосуються нашої розробки, включаючи організацію роботи та успішний розвиток нашого вебзастосунку для самостійного вивчення основ програмування. За допомогою SCRUM ми забезпечуємо систематичний

підхід до планування, організації та реалізації проєктів, розподіл ресурсів та ефективне керування проєктом для виконання строків. Використовуючи SCRUM разом з JIRA, ми можемо відстежувати завдання та проблеми, що стосуються проекту, такі як впровадження нових функцій, виправлення помилок або оновлення працездатності, пріоритизувати їх та відстежувати прогрес до їх завершення. Ми використовуємо можливості гнучкого управління проектом в SCRUM та JIRA, створюючи дошки для візуалізації завдань, визначаючи спринти та розбиваючи наш процес розробки на ітерації, які легко обробляти. Крім того, ми налаштовуємо робочі процеси та типи завдань JIRA, унікальні для вимог CodeCraftingLab, відрізняючи розробку нового модуля, рефакторинг змісту курсу або вирішення технічних проблем. Ми інтегруємо JIRA з інструментами розробки, такими як Bitbucket та GitHub, щоб відстежувати зміни коду відносно вказаних завдань або дефектів. Крім того, ми збираємо відгуки та пропозиції через JIRA, щоб записувати запити на функції або пропозиції щодо поліпшень, запропонованих користувачами, і призначаємо пріоритети для їх виконання. JIRA також виступає як центральний репозиторій для документів проекту, таких як керівництва користувача, замітки про реліз та технічна документація, забезпечуючи зручний доступ до них. JIRA та SCRUM – це інструменти, які дозволяють CodeCraftingLab дотримуватися структурованої рамки відстеження проблем, управління проектом та організації, що в свою чергу збільшує ефективність вебзастосунку, який вводить студентів в основи програмування.

4.2 Написання всіх завдань проекту

Перший спринт **CCL Sprint 1: Створення фронтенду для панелі адміністратора**

- сторінка входу;
- макет адмін-панелі;

– макет сторінки категорій;
– макет сторінки курсів;
– макет форми створення нового курсу;
– макет сторінки квізів;
– макет форми створення нового квізу;
– макет сторінки налаштувань.

Другий спринт **CCL Sprint 2: Створення бекенду для панелі адміністратора**

– сторінка входу;
– адмін-панель;
– сторінка категорій;
– сторінка курсів;
– форма створення нового курсу;
– сторінка квізів;
– форма створення нового квізу;
– сторінка налаштувань.

Третій спринт **CCL Sprint 3: Створення фронтенду для основного сайту**

– головна сторінка;
– сторінка категорій;
– сторінка курсів;
– сторінка квізів;
– сторінка контактів.

Четвертий спринт **CCL Sprint 4: Підключення бекенду панелі адмін до основного сайту**

– підключення до бази даних;
– відображення категорій та курсів;
– відображення квізів;
– інтеграція контактної форми;
– тестування та виправлення помилок.

4.3 Визначення балів історії

Було використано послідовність Фібоначчі для визначення балів історії, враховуючи рівень складності завдань і їх важливість у проекті. Цей підхід допомагає нам не лише оцінити час та зусилля, потрібні для виконання кожного завдання, але й чітко визначити їх пріоритетність для успішного завершення проекту. Наприклад, завдання з високою важливістю та великою складністю можуть мати більше балів історії, щоб відображати їхню вагомість у проекті та необхідність приступити до їх вирішення найшвидше. Завдання з меншою важливістю або складності можуть мати меншу кількість балів, але все одно вони враховані у нашому плануванні, щоб забезпечити повноту роботи та прогрес у всіх аспектах проекту. Такий підхід допомагає зберегти баланс між важливими та менш важливими завданнями, забезпечуючи ефективне керування проектом та досягнення поставлених цілей. Таблиця 4.1 показує розподіл балів історії за послідовністю Фібоначчі для різних завдань у проєкті.

Таблиця 4.1 – Оцінка балів історії за послідовністю Фібоначчі

Важливість завдання	Рівень складності	Бали історії
Низька	Легкий	1
Низька	Середній	2
Низька	Важкий	3
Середня	Легкий	2
Середня	Середній	3
Середня	Важкий	5
Висока	Легкий	3
Висока	Середній	5
Висока	Важкий	8

Як показано в таблицях 4.2, 4.3, 4.4, 4.5 і рисунках 4.1, 4.2, 4.3, 4.4, нами було встановлено бали історії наступним чином:

CCL Sprint 1: Створення фронтенду для панелі адміністратора

Таблиця 4.2 – Таблиця оцінених балів історії для завдань першого спринту

Завдання	Оцінені бали історії
Сторінка входу	1 бал
Макет адмін-панелі	2 бал
Макет сторінки категорій	3 бал
Макет сторінки курсів	3 бал
Макет форми створення нового курсу	3 бал
Макет сторінки квізів	3 бал
Макет форми створення нового квізу	3 бал
Макет сторінки налаштувань	1 бал

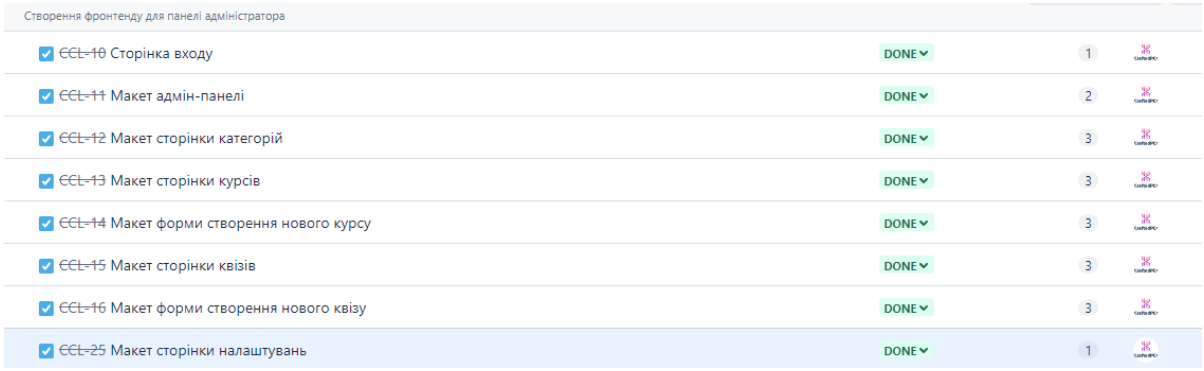

Рисунок 4.1 – Завдання першого спринту з JIRA

CCL Sprint 2: Створення бекенду для панелі адміністратора

Таблиця 4.3 – Таблиця оцінених балів історії для завдань другого спринту.

Завдання	Оцінені бали історії
Сторінка входу	2 бал
Адмін-панел	1 бал
Сторінка категорій	3 бал
Сторінка курсів	3 бал
Форма створення нового курсу	8 бал
Сторінка квізів	3 бал
Форма створення нового квізу	5 бал
Сторінка налаштувань	3 бал

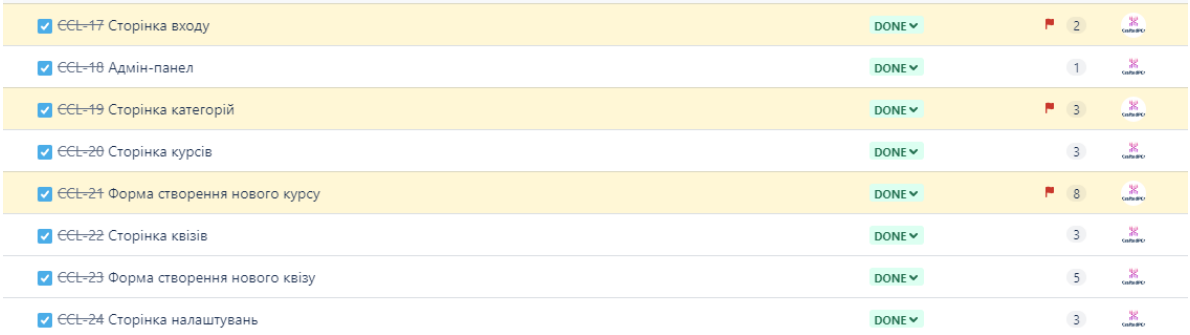

Рисунок 4.2 – Завдання другого спринту з JIRA

CCL Sprint 3: Створення фронтенду для основного сайту

Таблиця 4.4 – Таблиця оцінених балів історії для завдань третього спринту

Завдання	Оцінені бали історії
Головна сторінка	5 бал
Сторінка категорій	3 бал
Сторінка курсів	3 бал
Сторінка квізів	3 бал
Сторінка контактів	1 бал

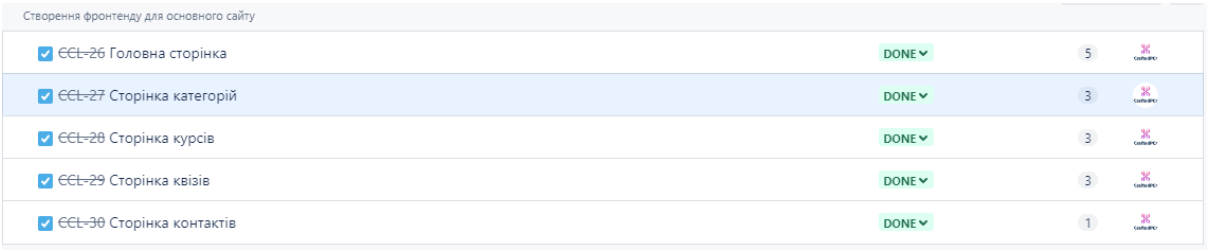

Рисунок 4.3 – Завдання третього спринту з JIRA

CCL Sprint 4: Підключення бекенду панелі адмін до основного сайту

Таблиця 4.5 – Таблиця оцінених балів історії для завдань четвертого спринту

Завдання	Оцінені бали історії
Підключення до бази даних MongoDB	2 бал
Відображення категорій та курсів	5 бал
Відображення квізів	5 бал
Інтеграція контактної форми	3 бал
Тестування та виправлення помилок	5 бал

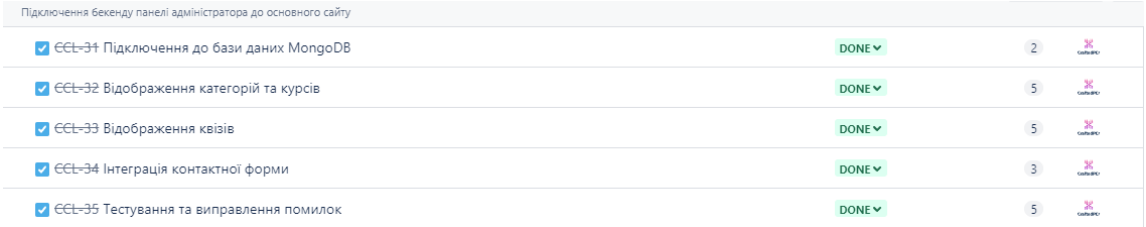

Рисунок 4.4 – Завдання четвертого спринту з JIRA

4.4 Тривалість спринтів

Встановлення тривалості спринтів відрізняється від одного проекту до іншого, від однієї команди до іншої, і не існує загального правила або головної формули для розрахунку тривалості, оскільки існує ряд факторів, що можуть впливати на тривалість, таких як розмір команди (кількість

працівників у проекті), швидкість (кількість балів історій, які ви можете завершити за один день), кількість робочих годин на день і багато інших факторів. Тому це питання зазвичай вирішується командами під час щоденних стендап-зустрічей.

У цьому проекті, для окремого учасника команди, ми можемо використовувати, наприклад, таку формулу для розрахунку тривалості спринтів:

$$\text{Тривалість(днів)} = \frac{\text{Балиісторії} \times \text{Середнякількістьбалівісторіїнадень}}{\text{Робочапотужність(годин)надень}}.$$

У нашому випадку, щоб бути реалістичним щодо можливості та темпу роботи, і оскільки цей проект реалізовується одноосібно, то розумний діапазон балів історії на день може бути від **1** до **3** балів. Отже, давайте розрахуємо тривалість кожного спринту:

Тривалість спринту 1:

$$\text{Тривалість(днів)} = \frac{19 \times 2}{5} = 7.6\text{день}.$$

Отже, як показано на рисунку 4.5, перший спринт може тривати 8 робочих днів

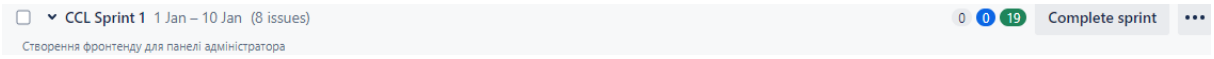

Рисунок 4.5 – Тривалість спринту 1

Тривалість спринту 2:

$$\text{Тривалість(днів)} = \frac{28 \times 2}{5} = 11.2\text{день}.$$

Отже, як показано на рисунку 4.6, другий спринт може тривати 12 робочих днів, починаючи точно після завершення першого спринту.

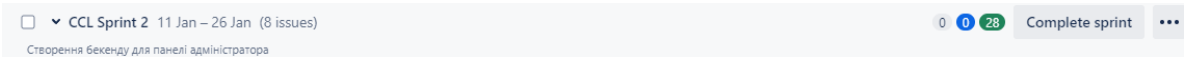

Рисунок 4.6 – Тривалість спринту 2

Тривалість спринту 3:

$$\text{Тривалість(днів)} = \frac{15 \times 2}{5} = 6\text{день.}$$

Третій спринт може тривати 6 робочих днів, як показано на рисунку 4.7, починаючи точно після завершення другого спринту.

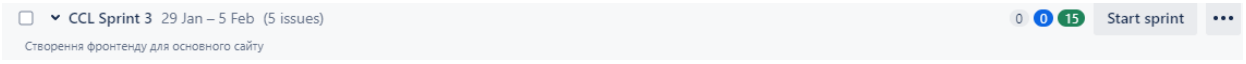

Рисунок 4.7 – Тривалість спринту 3

Тривалість спринту 4:

$$\text{Тривалість(днів)} = \frac{20 \times 2}{5} = 8\text{день.}$$

Нарешті, як показано на рисунку 4.8, четвертий спринт може тривати 8 робочих днів, починаючи точно після завершення третього спринту.

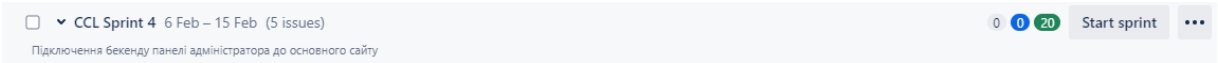

Рисунок 4.8 – Тривалість спринту 4

Загалом, реалізація проекту CodeCraftingLab повинна тривати приблизно 6 тижнів.

5 ВИБІР МЕТОДОЛОГІЇ

5.1 NextJS

При виборі методології виконання проєкту було взято до уваги низку особливостей NextJS. Розглянемо ці особливості.

Серверний рендер та генерація статичного сайту: Next.js надає виняткову гнучкість, підтримуючи як серверний рендер (SSR), так і генерацію статичного сайту (SSG). SSR дозволяє генерувати сторінки на сервері перед їхнім відправленням клієнту, що поліпшує швидкість завантаження та індексацію пошуковими системами. З іншого боку, генерація статичного сайту дозволяє попередньо відрендерити сторінки під час побудови застосунку, що гарантує постійну продуктивність та ефективне кешування.

Зручний розвиток: Next.js спрощує процес розробки завдяки різноманітним корисним функціям. Наприклад, автоматичне розбиття коду автоматично ділить код на невеликі шматки, що покращує початкові часи завантаження та навігацію в додатку. Крім того, маршрутизація на основі файлів дозволяє чітко та інтуїтивно організувати маршрути, а вбудована підтримка CSS та Sass спрощує стилізацію компонентів.

Ефективна інтеграція з React: Будучи побудованим на React, Next.js використовує функції React, такі як повторне використання компонентів, ефективне керування станом та гнучка система композиції. Тому розробники React знайдуть Next.js знайомим та зможуть скористатися своїми вже напрацьованими навичками.

Покращені продуктивність: Next.js наголошує на продуктивності, що призводить до швидших та більш реактивних додатків. Завдяки автоматичному розбиттю коду, завантажується лише потрібний код для кожної сторінки, що зменшує розмір JavaScript-пакетів та поліпшує час

завантаження. Крім того, SSR та генерація статичного сайту сприяють плавному та послідовному користувацькому досвіду, навіть за умов невигідної мережі.

Міцна екосистема: Next.js підтримується активною та динамічною спільнотою, що виражається вичерпною документацією, навчальними посібниками, прикладами коду та онлайн-підтримкою. Крім того, у Next.js є багата екосистема плагінів та бібліотек, що спрощує інтеграцію додаткових функцій, таких як аутентифікація, обробка форм та модульні тести, для задоволення конкретних потреб кожного проекту.

5.2 TailwindCSS

При виборі методології виконання проєкту було взято до уваги низку особливостей TailwindCSS. Розглянемо ці особливості.

Спочатку утилітарний підхід: Tailwind CSS використовує підхід спочатку утилітарний, надаючи велику колекцію класів-утилітарів, які безпосередньо застосовують властивості стилю. Цей підхід усуває потребу у власному CSS і дозволяє швидко створювати послідовні та реактивні дизайни, застосовуючи класи безпосередньо в HTML-розмітці.

Налаштовуваний та конфігуруємий: Хоча Tailwind CSS має повний набір попередньо визначених класів-утилітарів, він дуже налаштовуваний та конфігуруємий. Розробники можуть легко налаштовувати тему за замовчуванням, додавати нові класи-утилітари або змінювати існуючі, щоб відповідати своєму системі дизайну та вимогам свого проекту.

Швидкий розвиток: TailwindCSS прискорює процес розробки, оптимізуючи робочий процес стилізації. З попередньо визначеними класами-утилітарами, які охоплюють широкий спектр властивостей CSS, розробники можуть швидко прототипувати, ітерувати та будувати компоненти

користувацького інтерфейсу без переходу між CSS-файлами або написання повторюваних стилів.

Реактивний дизайн: TailwindCSS полегшує створення реактивних дизайнів за допомогою своїх класів-утилітарів реакції. Ці класи дозволяють розробникам застосовувати різні стилі в залежності від розмірів екрану, дозволяючи створювати макети, що ідеально адаптуються до різних пристроїв та розмірів вікон відображення.

Низька специфічність та модульний CSS: TailwindCSS сприяє модульній архітектурі CSS. Використовуючи класи-утилітари безпосередньо в HTML-розмітці, розробники можуть створювати модульні компоненти, які легше підтримувати, рефакторити та масштабувати. Крім того, низька специфічність класів-утилітарів зменшує ймовірність конфліктів CSS та робить стилізацію більш передбачуваною.

5.3 MongoDB

Використання бази даних NoSQL, такої як MongoDB, замість традиційної SQL-бази даних, має декілька переваг залежно від конкретних потреб нашого застосунку. Розглянемо ці переваги.

Гнучка схема: NoSQL-бази даних, такі як MongoDB, використовують гнучку модель схеми, що дозволяє динамічно зберігати дані у порівнянні з жорсткою схемою SQL-баз даних. Ця гнучкість особливо корисна в сценаріях, де структура даних постійно змінюється або коли маємо різноманітні типи даних.

Масштабованість: NoSQL-бази даних, як правило, призначені для горизонтального масштабування, що означає, що вони можуть обробляти великі обсяги даних та високі швидкості читання та запису, розподіляючи дані по кількох вузлах або серверах. Ця масштабованість добре підходить для

додатків, обсяги даних яких швидко зростають або мають великі навантаження трафіку.

Висока продуктивність: NoSQL-бази даних оптимізовані для конкретних випадків використання, таких як зберігання документів (MongoDB). Ця спеціалізація часто призводить до високої продуктивності для передбачуваних навантажень, особливо коли мова йде про великі обсяги даних або складні запити.

Простота використання: MongoDB є популярною NoSQL-базою даних, відомою своєю гнучкістю та простотою використання. Її модель даних, орієнтована на документи, добре підходить для динамічного характеру додатків Next.js, що дозволяє розробникам легко зберігати та маніпулювати складними даними JSON.

Плавна інтеграція: Next.js підтримує безперервну інтеграцію MongoDB за допомогою додаткових бібліотек та модулів, таких як next-connect або mongoose. Ці інструменти полегшують підключення та маніпулювання базою даних MongoDB з додатка Next.js, забезпечуючи плавний досвід розробки.

Управління API: Next.js дозволяє легко створювати маршрути API, які служать мостом між фронтендом застосунку та базою даних MongoDB. Ці API-маршрути можуть використовуватися для виконання операцій CRUD (створення, читання, оновлення, видалення) з даними, збереженими в MongoDB, надаючи чистий та безпечний інтерфейс для взаємодії з базою даних з додатка Next.js.

6 ГЛИБОКИЙ АНАЛІЗ ТА СПЕЦИФІКАЦІЯ ВИМОГ

У цьому розділі проводиться аналіз проекту:

6.1 Вибір та опис акторів

Рисунок 6.1 демонструє двох основних акторів у системі CodeCraftingLab.

6.1.1 Адміністратор

- це користувач з повним доступом до системи, який відповідає за управління контентом та налаштуваннями вебзастосунку;
- адміністратор може створювати, редагувати та видаляти курси, категорії, тестові завдання, а також керувати іншими аспектами застосунку;
- однак, через відсутність можливості відстеження активності користувачів, адміністратор не може надсилати повідомлення чи сповіщення користувачам.

6.1.2 Відвідувач

- це користувач, який відвідує веб-додаток CodeCraftingLab з метою самостійного вивчення основ програмування;
- відвідувач може переглядати доступний контент, виконувати тестові завдання та взаємодіяти з матеріалами для навчання;

– у системі відсутній процес реєстрації для відвідувачів, тому вони не мають особистих облікових записів і не можуть отримувати повідомлення від адміністратора.

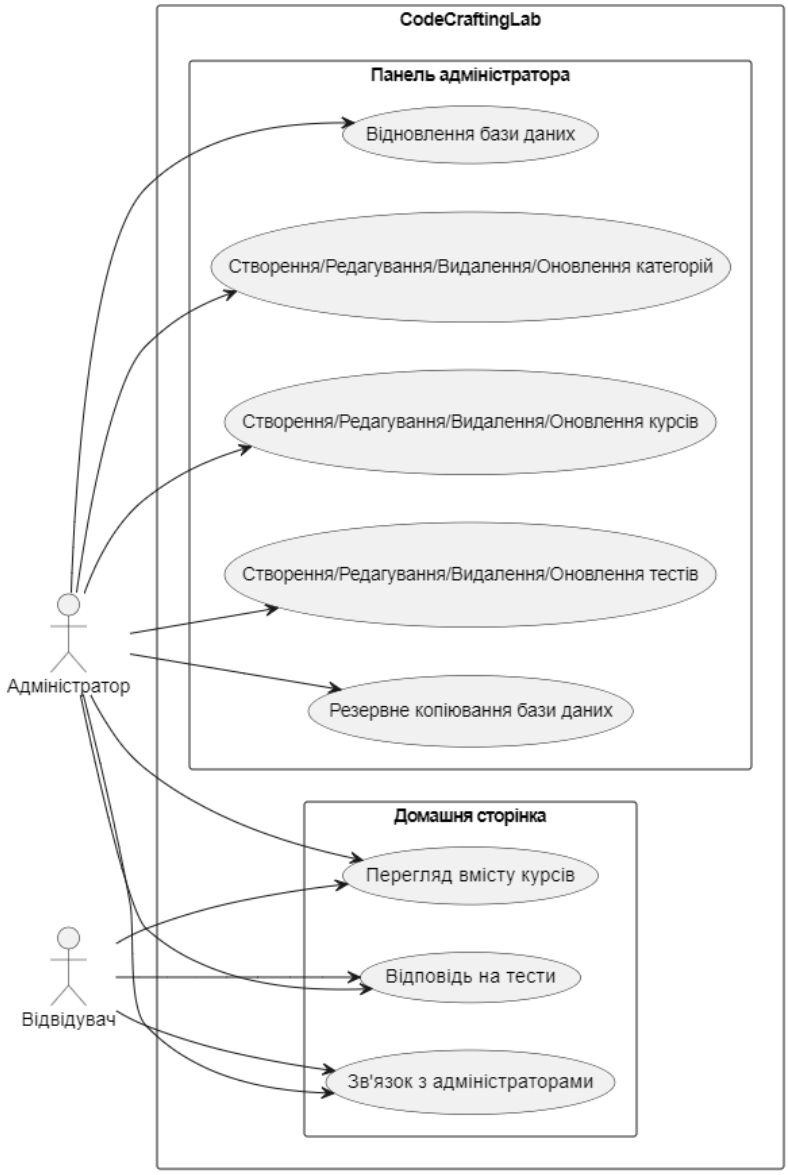

Рисунок 6.1 – Діаграма варіантів використання

6.2 Варіанти використання

Загальний опис варіантів використання системи CodeCraftingLab демонструє рисунок 6.1.

Перегляд вмісту курсів: Користувач може переглядати доступний вміст курсів на головній сторінці системи. Це дозволяє відвідувачам ознайомитися з наявними матеріалами перед їх участю.

Відповідь на тести: Відвідувачі можуть взяти участь у тестуванні, відповідаючи на питання, що стосуються вмісту курсів. Це дозволяє їм перевірити свої знання та оцінити рівень їхнього розуміння матеріалу.

Контакт з адміністраторами: Користувачі можуть звертатися до адміністраторів системи за допомогою контактної форми. Це дає їм можливість вирішувати питання, що стосуються курсів або функціонування системи.

Створення/редагування/видалення/оновлення категорій курсів: Адміністратор системи має можливість керувати категоріями курсів, додаючи, редагуючи, видаляючи або оновлюючи їх. Це дозволяє адміністратору організувати та підтримувати структуру курсів.

Створення/редагування/видалення/оновлення курсів: Адміністратор може керувати самими курсами, додаючи нові, редагуючи існуючі, видаляючи або оновлюючи інформацію про них. Це дозволяє адміністратору створювати та модернізувати курси відповідно до потреб користувачів.

Створення/редагування/видалення/оновлення тестів курсів: Адміністратор може керувати тестами курсів, додавати нові, редагувати існуючі, видаляти або оновлювати їхні питання та відповіді. Це дозволяє адміністратору забезпечити оцінку користувачів та перевірку їх розуміння матеріалу.

Резервне копіювання бази даних: Адміністратор може створювати резервні копії бази даних системи для забезпечення безпеки даних та можливості відновлення в разі потреби.

Відновлення бази даних: Адміністратор може відновлювати базу даних з резервної копії, якщо виникла потреба у відновленні даних.

6.3 Класи аналізу

Як показано на рисунках 6.2 і 6.3, розподіл класів за різними категоріями та виділенням важливих класів можна описати:

Граничний клас:

- користувач (User): представляє користувача системи, який може бути адміністратором або відвідувачем (рис 6.2);
- курс (Course): представляє навчальний курс у системі з полями, такими як назва, опис, батьківська категорія, зображення, вміст і відео (рис 6.2);
- категорія (Category): представляє категорію курсів з полями, такими як назва, батьківська категорія та зображення (рис 6.2);
- тест (Quiz): представляє тест для курсу з питаннями і відповідями (рис 6.2).

Керуючий клас:

- адміністратор (Administrator): керує адміністративними функціями системи, такими як управління категоріями курсів, курсами, тестами, резервним копіюванням та відновленням бази даних (рис. 6.3);
- курс менеджер (CourseManager): відповідає за управління курсами, включаючи створення, редагування, видалення, оновлення інформації про курс (рис 6.3);
- категорія менеджер (CategoryManager): відповідає за управління категоріями курсів, включаючи створення, редагування, видалення та оновлення інформації про категорії (рис 6.3);

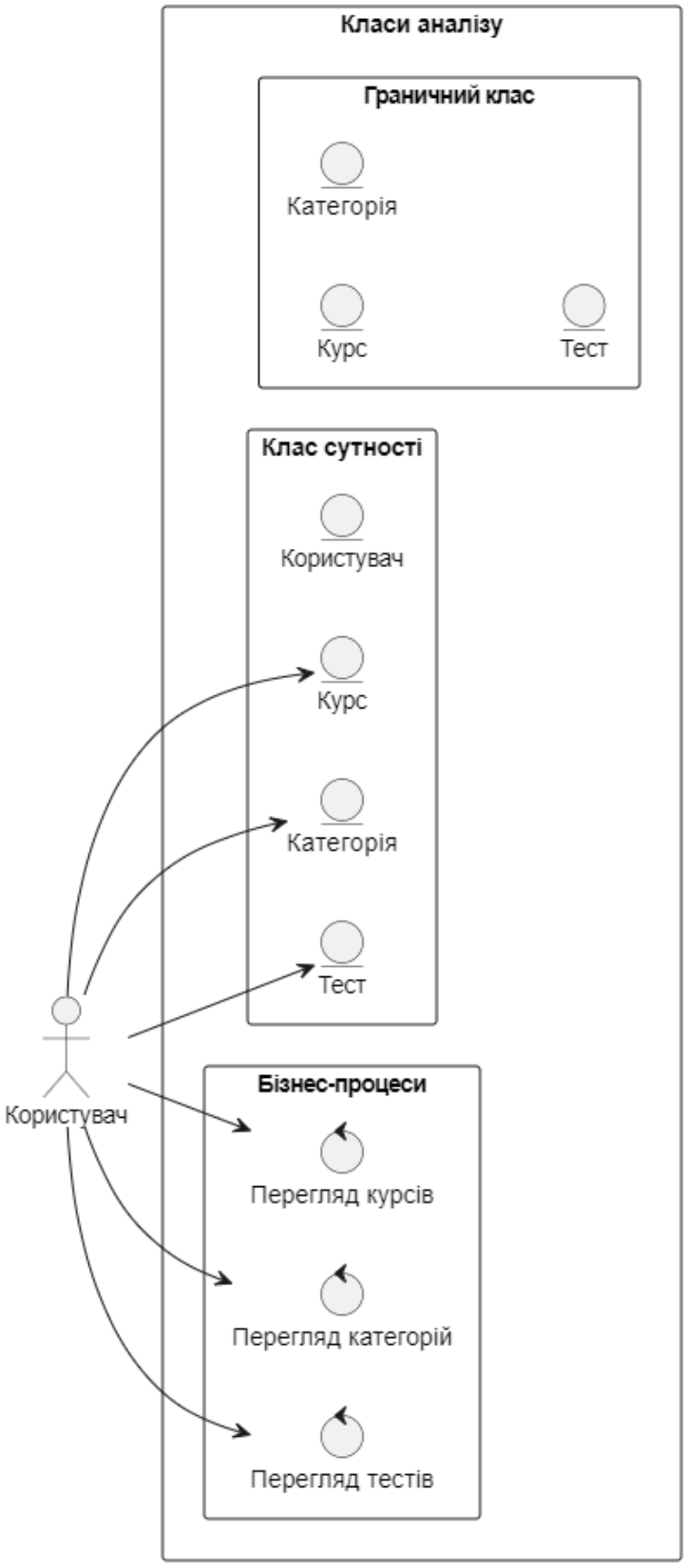

Рисунок 6.2 – Діаграма класів аналізу для варіанту використання системи актором Користувач

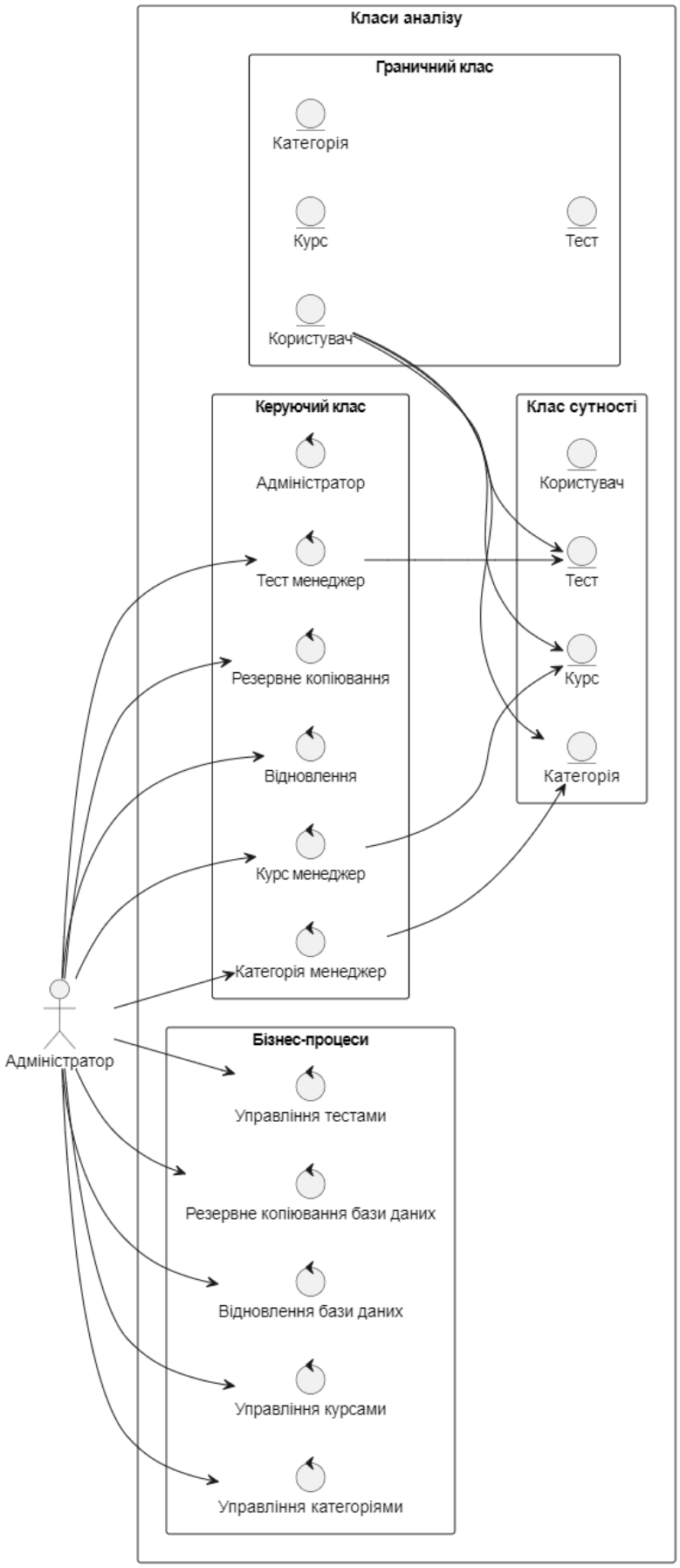

Рисунок 6.3 – Діаграма класів аналізу для варіанту використання системи актором Адміністратор

- тест менеджер (QuizManager): відповідає за управління тестами, включаючи створення, редагування, видалення, оновлення інформації про тести (рис 6.3);
- резервне копіювання (BackupManager): керує процесом резервного копіювання бази даних (рис 6.3);
- відновлення (RestoreManager): керує процесом відновлення бази даних з резервних копій (рис 6.3).

Клас сутності:

- користувач (User): поля: ідентифікатор, ім’я, електронна пошта, роль;
- курс (Course): поля: ідентифікатор, назва, опис, батьківська категорія, зображення, вміст, відео;
- категорія (Category): поля: ідентифікатор, назва, батьківська категорія, зображення;
- тест (Quiz): поля: ідентифікатор, назва, питання, відповіді.

Бізнес-процеси.

Управління курсами (Manage Courses):

- створення курсу (Create Course): адміністратор може створювати нові курси, вказуючи назву, опис, батьківську категорію, зображення, вміст і відео курсу;
- редагування курсу (Edit Course): адміністратор може змінювати існуючі дані курсу, такі як назва, опис, зображення, вміст і відео;
- видалення курсу (Delete Course): адміністратор може видаляти курси з системи.

Управління категоріями (Manage Categories):

- створення категорії (Create Category): адміністратор може створювати нові категорії курсів, вказуючи назву та батьківську категорію;
- редагування категорії (Edit Category): адміністратор може змінювати назву або батьківську категорію існуючих категорій;

– видалення категорії (Delete Category): адміністратор може видаляти категорії курсів з системи.

Управління тестами (Manage Quizzes):

– створення тесту (Create Quiz): адміністратор може створювати нові тести для курсів, вказуючи питання та варіанти відповідей;
– редагування тесту (Edit Quiz): адміністратор може змінювати питання та варіанти відповідей існуючих тестів;
– видалення тесту (Delete Quiz): адміністратор може видаляти тести з системи.

Резервне копіювання бази даних (Backup Database):

– запуск резервного копіювання (Backup): адміністратор може запустити процес резервного копіювання бази даних для збереження даних.

Відновлення бази даних (Restore Database):

– запуск відновлення (Restore): адміністратор може запустити процес відновлення бази даних з резервної копії для відновлення даних.

6.4 Взаємодія класів аналізу в об’єктно-орієнтованій парадигмі

У об’єктно-орієнтованій парадигмі взаємодія між класами аналізу відіграє ключову роль у визначенні того, як різні компоненти системи співпрацюють для досягнення різноманітних функціональностей. Ця взаємодія характеризується наступними ключовими аспектами:

Граничний клас:

– клас користувач представляє користувачів системи, які можуть бути адміністраторами або відвідувачами;
– клас курс представляє навчальні курси у системі, включаючи поля, такі як назва, опис, батьківська категорія, зображення, вміст та відео;

- клас категорія представляє категорії курсів з полями, такими як назва, батьківська категорія та зображення;
- клас тест представляє тести для курсів з питаннями та відповідями.

Керуючий клас:

- клас Адміністратор керує адміністративними функціями системи, такими як управління категоріями курсів, курсами, тестами, резервним копіюванням та відновленням;
- клас CourseManager відповідає за управління курсами, включаючи створення, редагування та видалення;
- клас CategoryManager відповідає за управління категоріями курсів, включаючи створення, редагування та видалення;
- клас QuizManager відповідає за управління тестами, включаючи створення, редагування та видалення;
- клас BackupManager контролює процес резервного копіювання бази даних;
- клас RestoreManager контролює процес відновлення бази даних з резервної копії.

Клас сутності:

- клас Користувач:
  - поля: ідентифікатор, ім’я, електронна пошта, роль;
- клас Курс:
  - поля: ідентифікатор, назва, опис, батьківська категорія, зображення, вміст, відео;
- клас Категорія:
  - поля: ідентифікатор, назва, батьківська категорія, зображення;
- клас Тест:
  - поля: ідентифікатор, назва, питання, відповіді.

Бізнес-процеси:

- управління курсами:

  - створення курсу: адміністратор може створювати нові курси з вказанням різних параметрів;
  - редагування курсу: адміністратор може змінювати наявні дані курсу;
  - видалення курсу: адміністратор може видаляти курси з системи.
- управління категоріями:
  - створення категорії: адміністратор може створювати нові категорії курсів;
  - редагування категорії: адміністратор може змінювати існуючі категорії;
  - видалення категорії: адміністратор може видаляти категорії з системи.
- управління тестами:
  - створення тесту: адміністратор може створювати нові тести для курсів;
  - редагування тесту: адміністратор може змінювати наявні тести;
  - видалення тесту: адміністратор може видаляти тести з системи;
- резервне копіювання бази даних:
  - запуск резервного копіювання: адміністратор може запустити процес резервного копіювання бази даних для збереження даних.
- відновлення бази даних:
  - запуск відновлення: адміністратор може запустити процес відновлення бази даних з резервної копії для відновлення даних.

6.5 Взаємодія об’єктів у системі за допомогою діаграм послідовності

Важливість діаграм послідовності в системі CodeCraftingLab полягає у тому, що вони дозволяють відобразити послідовність взаємодій між об’єктами системи та користувачами. Це допомагає краще зрозуміти, як працює система і як вона реагує на різні взаємодії користувачів. В рамках нашої системи, я планую розглянути різні послідовності, особливо у підсистемі "Панель адміністратора". Наприклад, такі послідовності, як вхід в систему "Панель адміністратора", створення нової категорії, створення нового курсу, створення нового тесту і т. д. Ці послідовності дозволять нам краще розібратися у функціоналі системи та її можливостях.

6.5.1 Діаграма послідовності для процесу входу

Як показано на рисунку 6.4, діаграма послідовності процесу входу користувача до панелі адміністратора виглядає наступним чином: Користувач відвідує сторінку входу в адмін-панель і натискає кнопку "Увійти через Google". На цьому етапі відбувається обробка аутентифікації користувача через Google API. Якщо користувач успішно авторизований як адміністратор, відбувається створення сесії, і користувач автоматично перенаправляється на панель інструментів. На панелі інструментів відображається привітання користувачу. У випадку, якщо користувач не авторизований адміністратором, він отримує відхилення входу і повідомлення про помилку.

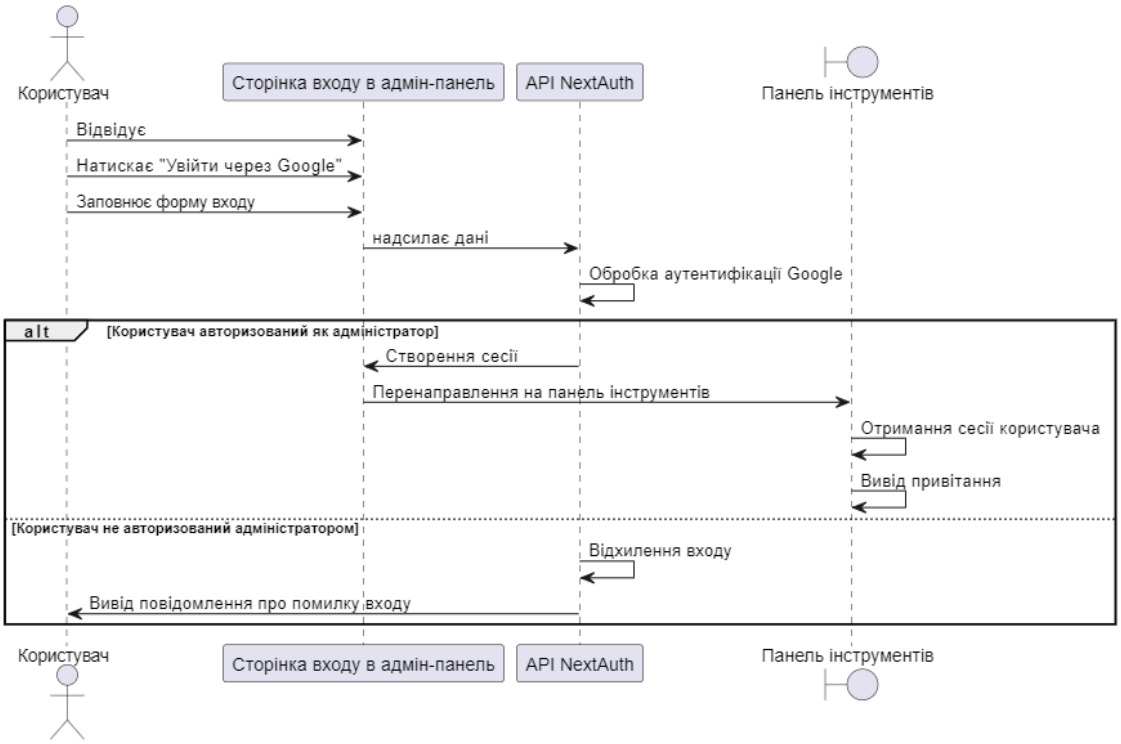

Рисунок 6.4 – Діаграма послідовності процесу входу користувача на адмін-панель

6.5.2 Діаграма послідовності створення нової категорії

Як показано на рисунку 6.5, діаграма послідовності створення нової категорії виглядає наступним чином: Процес створення нової категорії починається з переходу адміністратора до сторінки категорій через навігаційне меню. Після цього відправляється запит GET до API категорій для отримання списку наявних категорій з бази даних. Після отримання списку категорій, адміністратор заповнює форму нової категорії, вибираючи назву та завантажуючи зображення. Якщо він вибирає батьківську категорію, він також може обрати її зі списку. Після завершення заповнення форми, адміністратор натискає кнопку "Зберегти", і дані про нову категорію відправляються POST запитом до API категорій для збереження в базі даних.

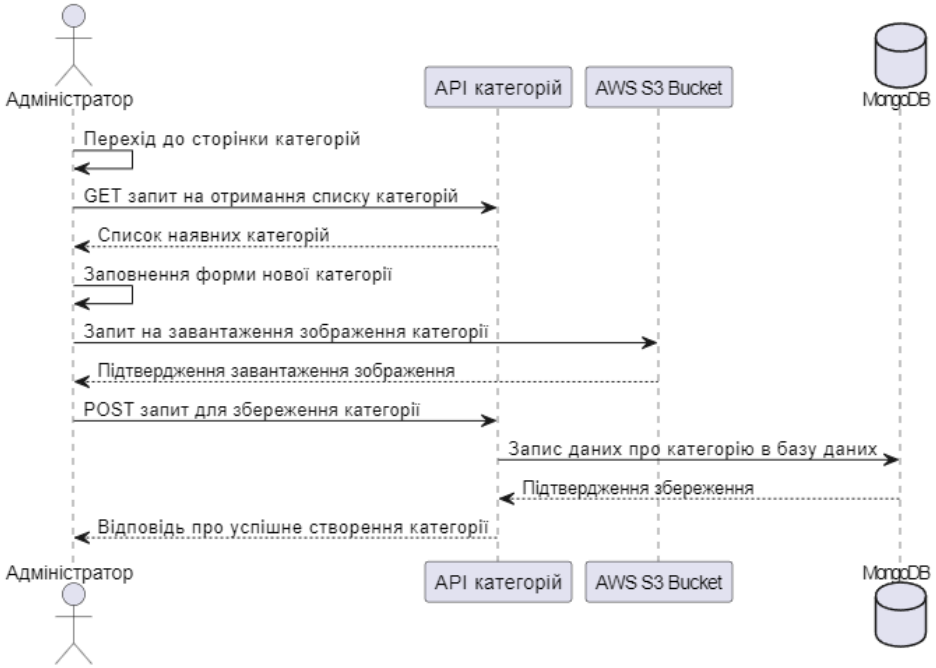

Рисунок 6.5 – Діаграма послідовності створення нової категорії

6.5.3 Діаграма послідовності редагування категорії

Рисунок 6.6 демонструє процес редагування категорії, який описано у діаграмі послідовності. Процес редагування категорії розпочинається з переходу адміністратора до сторінки категорій через навігаційне меню в адмін-панелі. Далі адміністратор обирає категорію для редагування, натискаючи кнопку "Редагувати" поряд з нею. Після цього здійснюється GET-запит до API категорій для отримання даних про обрану категорію, і форма редагування автоматично заповнюється цими даними. Адміністратор вносить необхідні зміни в поля форми, і якщо він змінює зображення категорії, відправляється POST-запит на завантаження нового зображення до AWS S3 Bucket. Після внесення всіх змін адміністратор натискає кнопку "Зберегти", після чого відправляється PUT-запит до API категорій для оновлення даних про категорію в базі даних. Після успішного оновлення адміністратор отримує підтвердження про успішне оновлення категорії.

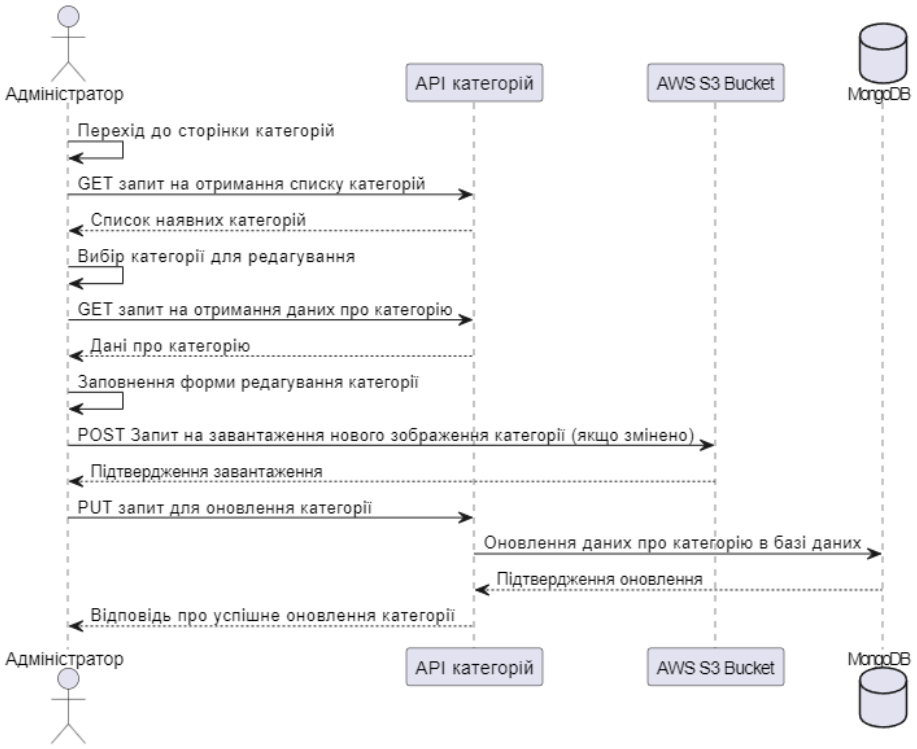

Рисунок 6.6 – Діаграма послідовності редагування категорії

6.5.4 Діаграма послідовності видалення категорії

Рисунок 6.7 показує діаграму послідовності видалення категорії, який описано у діаграмі послідовності. Процес видалення категорії починається з переходу адміністратора до сторінки категорій через навігаційне меню в адмін-панелі. Після цього адміністратор обирає категорію для видалення, натискаючи кнопку "Видалити" поряд з нею. На новій сторінці відображається підтверджувальне повідомлення, яке запитує адміністратора, чи він впевнений у своєму виборі. Якщо адміністратор підтверджує видалення, відправляється DELETE-запит до API категорій для видалення обраної категорії з бази даних. Після успішного видалення категорії з бази даних, всі пов’язані з нею курси також видаляються. Після видалення адміністратор повертається на головну сторінку категорій.

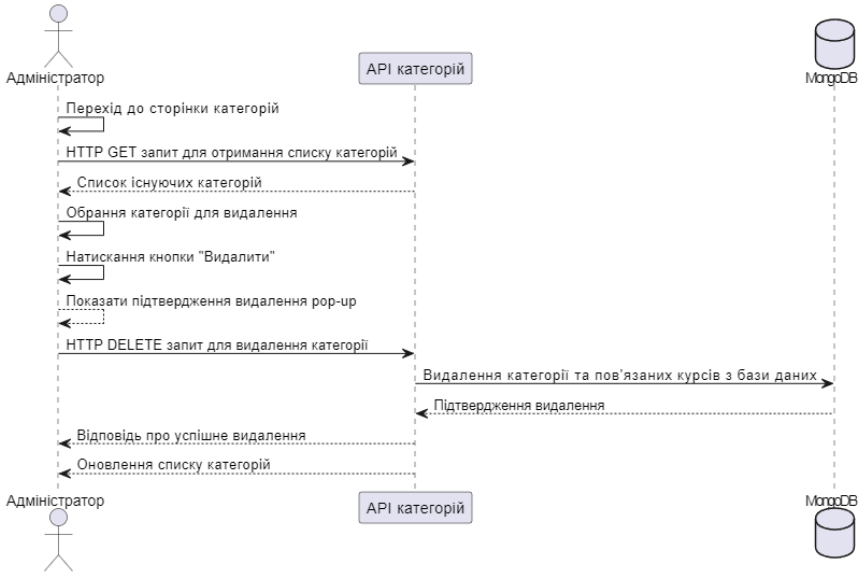

Рисунок 6.7 – Діаграма послідовності видалення категорії

6.5.5 Діаграма послідовності видалення всіх категорій

Рисунок 6.8 показує діаграму послідовності для видалення всіх категорій, який описано у діаграмі послідовності. Процес видалення всіх категорій розпочинається з переходу адміністратора до сторінки "Категорії" через навігаційне меню в адмін-панелі. Після цього адміністратор натискає кнопку "Видалити всі категорії". На екрані з’являється підтверджувальне повідомлення, що питає адміністратора, чи він впевнений у своєму виборі. Після підтвердження видалення адміністратор надсилає HTTP DELETE запит на видалення всіх категорій до відповідного API. Сервер отримує цей запит і видаляє всі категорії та пов’язані з ними курси з бази даних. Після успішного видалення адміністратор бачить, що список категорій порожній на сторінці "Категорії".

6.5.6 Діаграма послідовності пошук існуючої категорії

Рисунок 6.9 показує діаграму послідовності пошуку існуючої категорії, який описано у діаграмі послідовності. Процес пошуку категорії розпочинається з переходу адміністратора до сторінки категорій через навігаційне меню. Після цього адміністратор вводить назву категорії в поле пошуку.

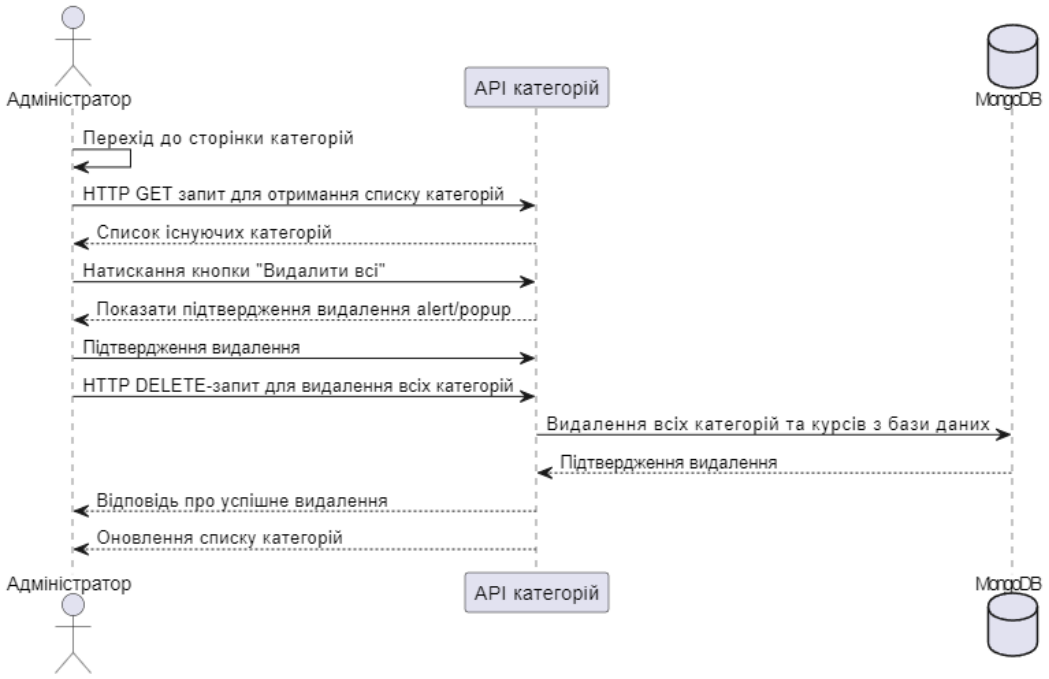

Рисунок 6.8 – Діаграма послідовності видалення всіх категорій

Після введення назви, відправляється GET-запит до сервера для пошуку категорії. Категорії API пересилає цей запит до бази даних MongoDB, де здійснюється пошук категорії за введеною назвою. Після цього результати пошуку надсилаються назад до CategoriesAPI, який повертає їх адміністратору.

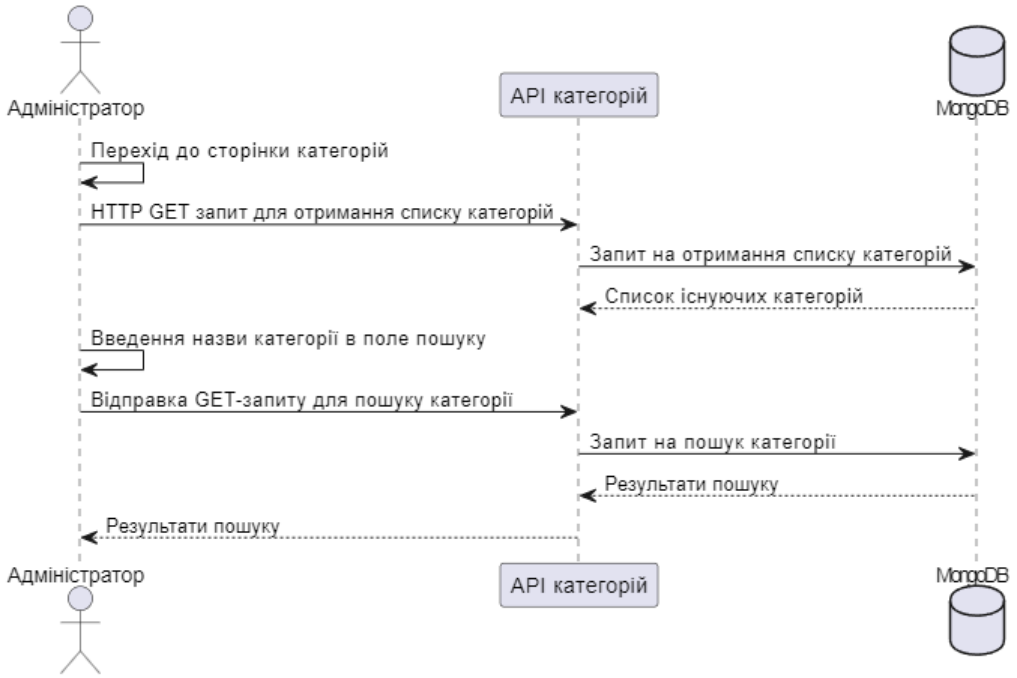

Рисунок 6.9 – Діаграма послідовності пошук існуючої категорії

6.5.7 Діаграма послідовності створення нового курсу

Процес створення нового курсу розпочинається з переходу адміністратора на сторінку курсів через навігаційне меню. Тут адміністратор отримує список існуючих курсів за допомогою HTTP GET запиту до відповідного API курсів. Після цього він натискає кнопку "Додати новий курс" і переходить на сторінку створення нового курсу. На цій сторінці адміністратор отримує список існуючих категорій за допомогою HTTP GET запиту до API категорій, щоб обрати батьківську категорію для нового курсу. Після заповнення форми курсу, включаючи назву, опис, контент та URL відео з YouTube, адміністратор завантажує зображення курсу за допомогою запиту на AWS S3 Bucket. Перед натисканням кнопки "Зберегти", проводиться перевірка, щоб переконатися, що всі обов’язкові поля заповнені, а також виконується перевірка формату URL відео з YouTube за допомогою RegExp. Після заповнення всіх необхідних полів та завантаження зображення, адміністратор натискає кнопку "Зберегти", що відправляє POST запит на API курсів для створення нового запису про курс. Після успішного створення курсу адміністратор не отримує жодного повідомлення, замість цього він автоматично перенаправляється назад на сторінку курсів. Детальна діаграма послідовності створення нового курсу показана на рисунку 6.10.

6.5.8 Діаграма послідовності редагування курсу

Процес редагування курсу розпочинається з переходу адміністратора до сторінки курсів через навігаційне меню. Після обрання курсу для редагування адміністратор надсилає HTTP GET запит до API курсів для отримання даних про обраний курс. Далі він перевіряє список існуючих категорій і, якщо це необхідно, натисканням на відповідне поле вибору батьківської категорії відправляється HTTP GET запит до API категорій.

Після цього дані про обраний курс та список категорій заповнюються на сторінці редагування курсу.

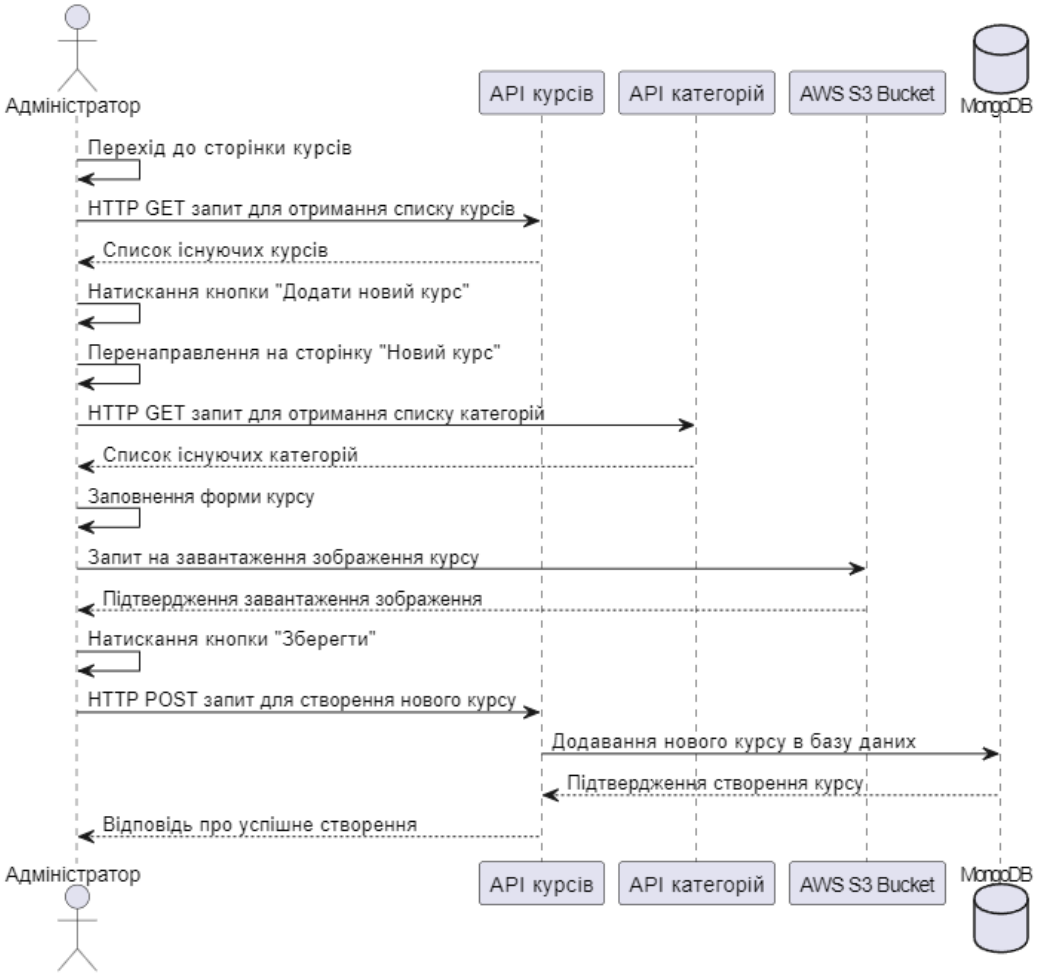

Рисунок 6.10 – Діаграма послідовності створення нового курсу

Далі адміністратор може внести необхідні зміни, у тому числі оновити зображення курсу, надіславши запит на оновлення до AWS S3 Bucket. Після цього адміністратор натискає кнопку "Зберегти", що викликає HTTP PUT запит до API курсів для оновлення даних про курс у базі даних MongoDB. Після успішного оновлення, адміністратор отримує відповідь про успішність від API курсів. Діаграма послідовності редагування курсу згадується на рисунку 6.11 нижче.

Рисунок 6.11 – Діаграма послідовності редагування курсу

6.5.9 Діаграма послідовності видалення курсу

Процес видалення курсу розпочинається з переходу адміністратора на сторінку курсів через навігаційне меню. Після досягнення цієї сторінки адміністратор бачить список доступних курсів. Коли адміністратор вирішує видалити певний курс, він натискає кнопку "видалити" поряд з цим курсом. Після натискання кнопки видалення з’являється підтвердження у вигляді спливаючого вікна (popup), яке питає адміністратора підтвердити видалення курсу. Якщо адміністратор підтверджує видалення, надсилається HTTP DELETE запит до API курсів для видалення конкретного курсу. Цей запит включає ідентифікатор курсу (course_id), який потрібно видалити. API курсів взаємодіє з базою даних MongoDB, щоб видалити запис про курс. Після успішного видалення курсу з бази даних, API курсів надсилає підтвердження

про успішне видалення. Діаграма послідовності видалення курсу показана на рисунку 6.12.

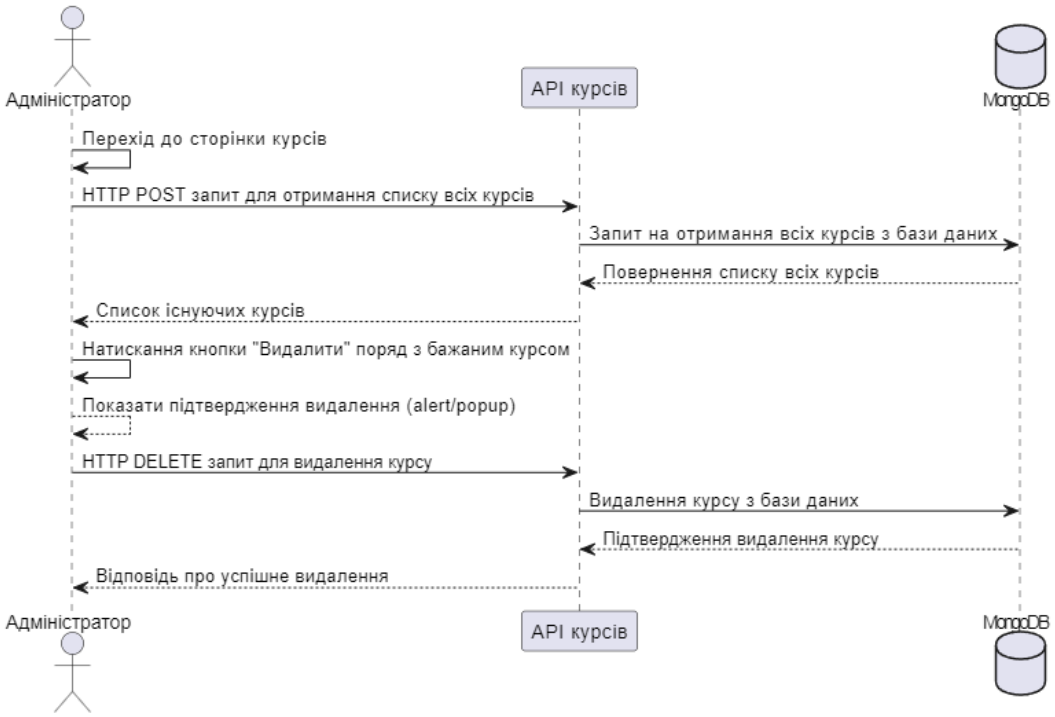

Рисунок 6.12 – Діаграма послідовності видалення курсу

6.5.10 Діаграма послідовності видалення всіх курсів

Процес видалення всіх курсів розпочинається з переходу адміністратора на сторінку курсів через навігаційне меню. Після досягнення цієї сторінки адміністратор натискає кнопку "Видалити всі курси". Після натискання кнопки, адміністратор надсилає HTTP DELETE запит до API курсів для видалення всіх курсів. API курсів взаємодіє з базою даних MongoDB і видаляє всі записи про курси. Після успішного видалення всіх курсів з бази даних, MongoDB надсилає підтвердження про успішне видалення до API курсів. Діаграма послідовності для видалення всіх курсів визначена на рисунку 6.13.

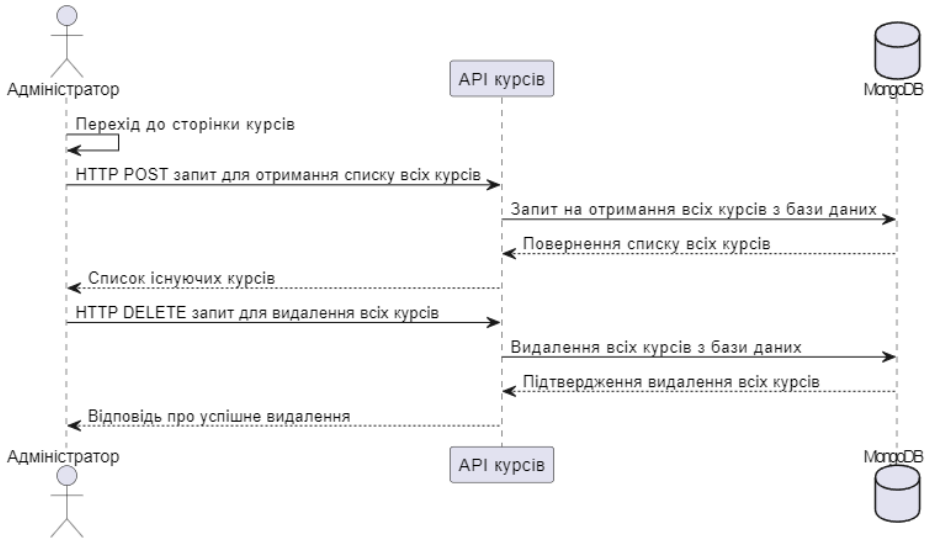

Рисунок 6.13 – Діаграма послідовності видалення всіх курсів

6.5.11 Діаграма послідовності пошук існуючої курсу

Адміністратор спочатку переходить на сторінку курсів. Після цього він вводить назву курсу в поле пошуку. Після натискання кнопки пошуку відправляється HTTP GET-запит до API курсів з назвою курсу для пошуку. API курсів звертається до бази даних MongoDB для пошуку курсу за назвою. Після успішного пошуку API повертає результати пошуку, що включають список знайдених курсів, і відображає їх адміністратору. Цей процес демонструється на рисунку 6.14.

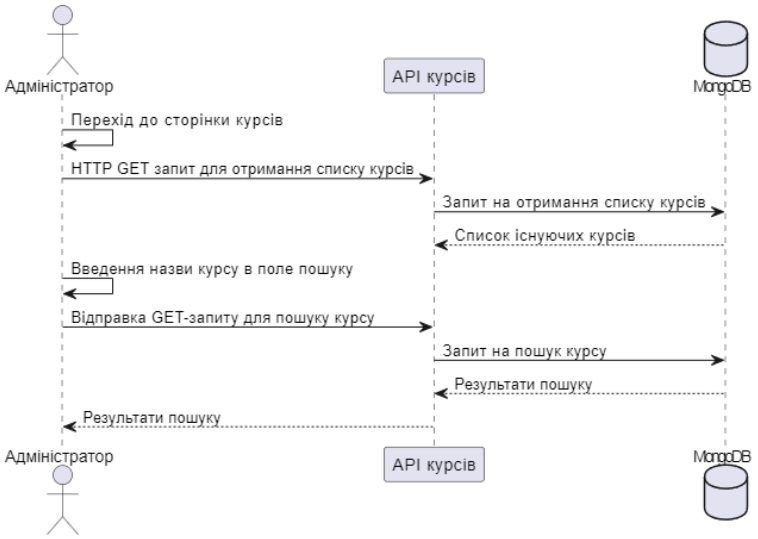

Рисунок 6.14 – Діаграма послідовності пошук існуючої курсу

6.5.12 Діаграма послідовності створення нового квізу

Процес створення нового квізу починається з переходу адміністратора до сторінки квізів через навігаційне меню. На цій сторінці адміністратор отримує список всіх існуючих квізів, який відображається у таблиці (HTTP GET запит до API квізів). Після цього адміністратор натискає кнопку "Новий квіз" і переходить на сторінку створення нового квізу. На цій сторінці адміністратор обирає рівень складності квізу (легкий/середній/важкий) та заповнює форму, вводячи питання, варіанти відповідей та вибираючи правильну відповідь. Після натискання кнопки "Зберегти" відправляється POST запит до API квізів для створення нового квізу. Дані про квіз зберігаються в базі даних MongoDB. Після успішного створення квізу адміністратор отримує підтвердження про успішне створення, після чого форма очищається, і він може створити новий квіз. Цей процес демонструється на рисунку 6.15.

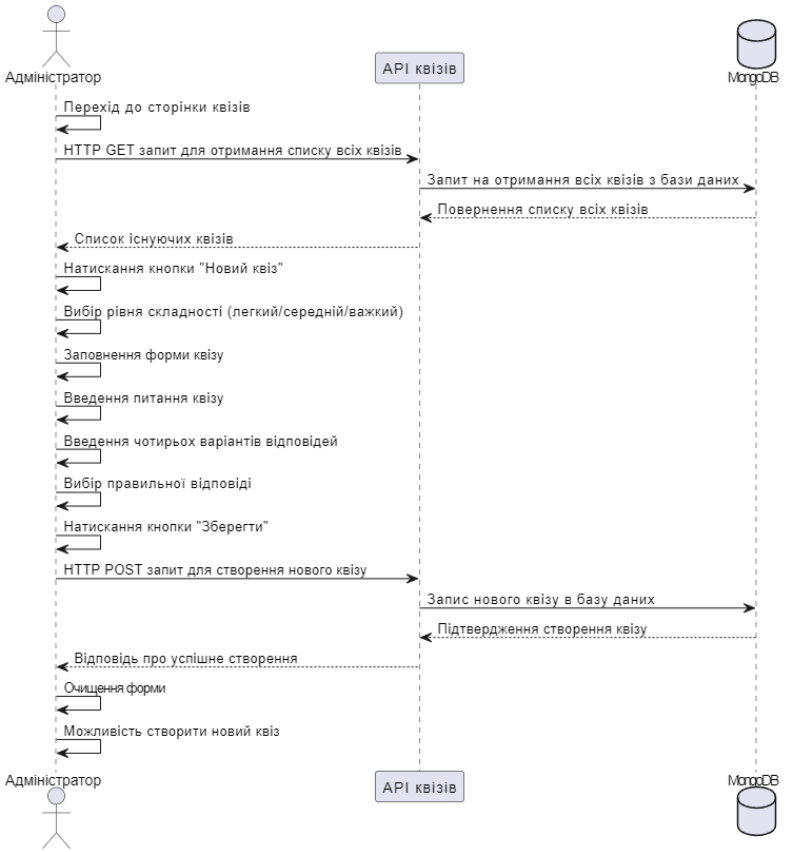

Рисунок 6.15 – Діаграма послідовності створення нового квіз

6.5.13 Діаграма послідовності редагування квізу

Після переходу на сторінку квізів, адміністратор обирає квіз для редагування, натискаючи відповідну кнопку "Редагувати". Далі він перенаправляється на сторінку редагування квізу quizzes/edit/[quiz_id], де дані квізу заповнюються автоматично за допомогою HTTP GET запиту до API квізів, який отримує дані квізу з бази даних MongoDB. Форма редагування заповнюється цими даними, і адміністратор може внести необхідні зміни, такі як зміна питання, варіантів відповідей та вибір правильної відповіді. Після внесення змін адміністратор натискає кнопку "Зберегти", що призводить до відправлення HTTP PUT запиту до API квізів для оновлення даних квізу. API квізів здійснює оновлення даних квізу в базі даних MongoDB. Після успішного оновлення адміністратора перенаправляється на сторінку квізів з оновленим списком квізів. Цей процес ілюструє діаграма послідовності, показана на рисунку 6.16.

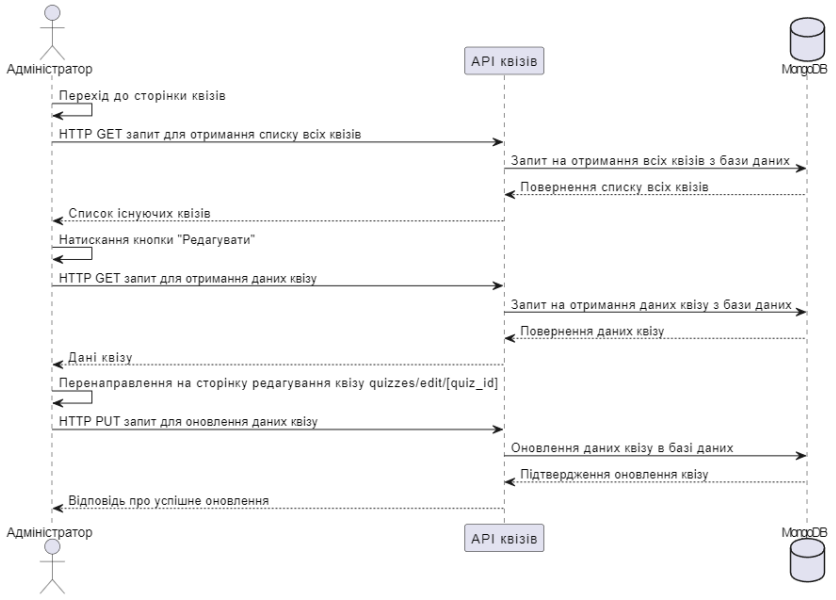

Рисунок 6.16 – Діаграма послідовності редагування квізу

6.5.14 Діаграма послідовності видалення квізу

Процес видалення квізу починається з того, що адміністратор переходить на сторінку квізів, де відображається список всіх доступних квізів, завдяки HTTP GET запиту до API квізів для отримання даних. Після цього адміністратор клікає кнопку "Видалити" поряд з бажаним квізом, після чого він перенаправляється на сторінку /quizzes/delete/[quiz_id], де з’являється підтвердження видалення. Якщо адміністратор підтверджує видалення, відправляється HTTP DELETE запит до API квізів для видалення квізу за його ідентифікатором (quiz_id). Після успішного видалення квізу з бази даних API квізів повертає відповідь про успішне видалення, яку отримує адміністратор. Цей процес ілюструє діаграма послідовності, показана на рисунку 6.17.

Рисунок 6.17 – Діаграма послідовності видалення квізу

6.5.15 Діаграма послідовності видалення всіх квізів

Процес видалення всіх квізів розпочинається з переходу адміністратора на сторінку квізів, де відображається список всіх доступних квізів, завдяки HTTP GET запиту до API квізів для отримання даних. Після цього адміністратор клікає кнопку "Видалити всі квізи", після чого відправляється HTTP DELETE запит до API квізів для видалення всіх квізів з бази даних. Після успішного видалення API квізів повертає відповідь про успішне видалення, яку отримує адміністратор. Цей процес ілюструє діаграма послідовності, показана на рисунку 6.18.

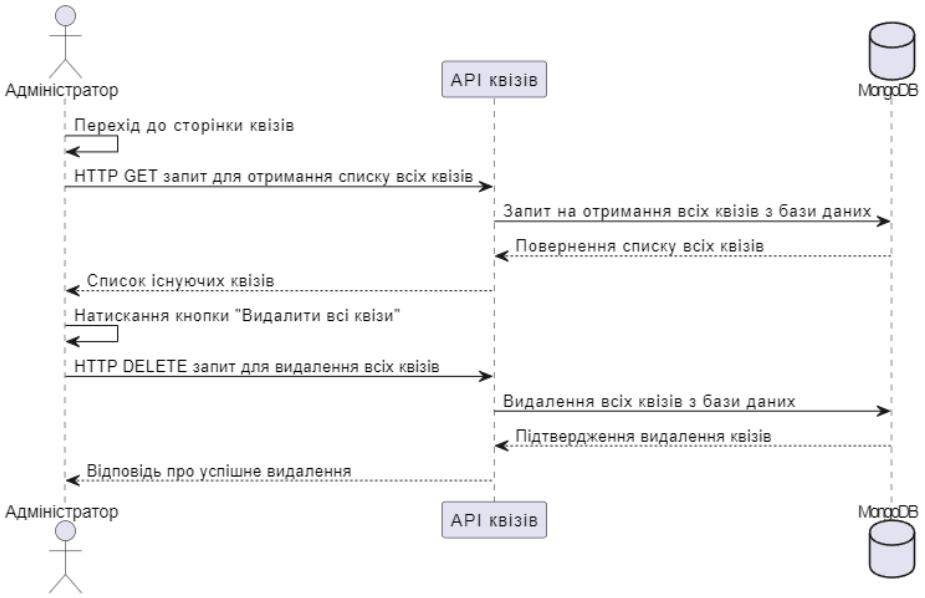

Рисунок 6.18 – Діаграма послідовності видалення всіх квізів

6.5.16 Діаграма послідовності пошуку квізу

Процес пошуку квізу розпочинається з переходу адміністратора на сторінку квізів. Після цього адміністратор вводить назву квізу у поле пошуку. Після введення назви квізу адміністратор відправляє GET-запит до API квізів для пошуку квізу за введеною назвою. API квізів звертається до бази даних MongoDB і проводить пошук квізу за введеними параметрами. Після цього результати пошуку повертаються до адміністратора через API, і

він отримує список квізів, які відповідають його пошуковому запиту. Цей процес ілюструє діаграма послідовності, показана на рисунку 6.19.

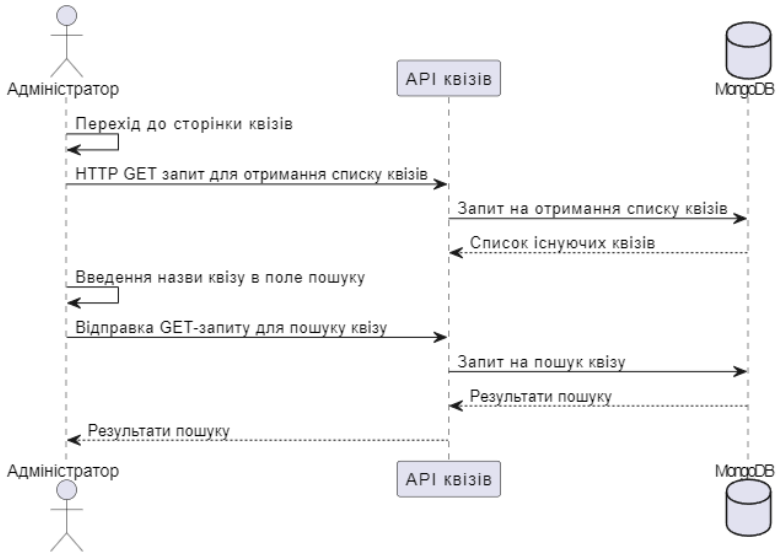

Рисунок 6.19 – Діаграма послідовності пошуку квізу

6.5.17 Діаграма послідовності резервного копіювання даних

Адміністратор відвідує сторінку налаштувань і натискає кнопку «Резервне копіювання даних». На сторінці налаштувань генерується HTTP POST-запит до API резервного копіювання з метою ініціювати процес резервного копіювання. Після цього API резервного копіювання звертається до бази даних MongoDB і виконує резервне копіювання. У випадку успішного копіювання, API надсилає відповідь з даними резервної копії на сторінку налаштувань (HTTP 200). У випадку невдачі копіювання, API також надсилає відповідь на сторінку налаштувань із статусом 500 та повідомленням про помилку. Якщо адміністратор намагається використати непідтримуваний метод (наприклад, HTTP GET), то API резервного копіювання надсилає відповідь з повідомленням про непідтримуваний метод (HTTP 405). Діаграма послідовності резервного копіювання даних наведена на рисунку 6.20.

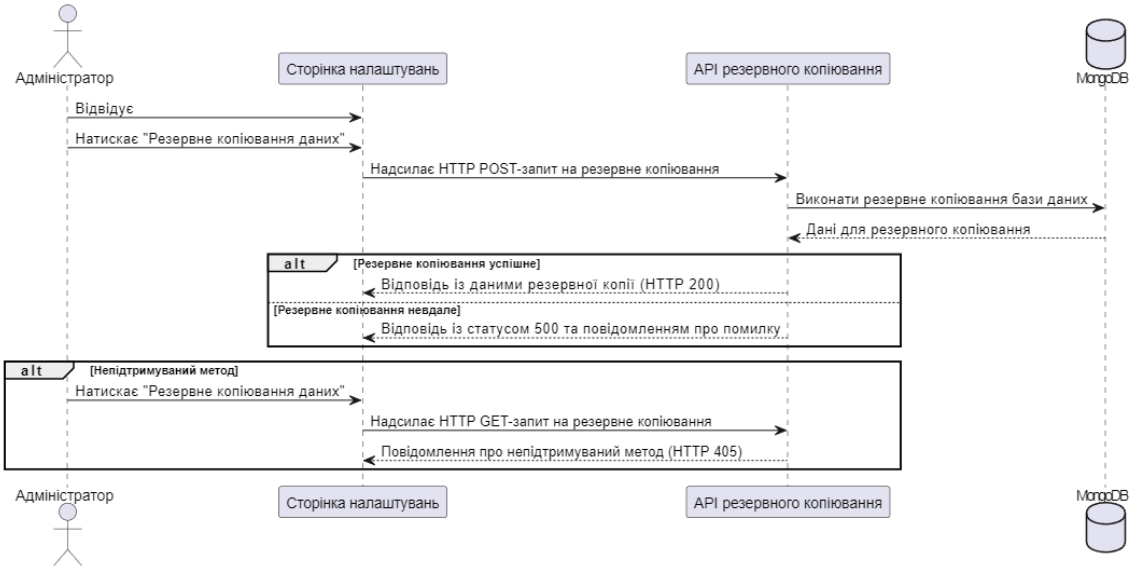

Рисунок 6.20 – Діаграма послідовності резервного копіювання даних

6.5.18 Діаграма послідовності резервного відновлення даних

Адміністратор виконує відповідний запит на сторінці налаштувань. На сторінці відновлення адміністратор вибирає файл з резервною копією даних і натискає кнопку «Відновлення даних». Після цього відбувається відправка HTTP POST-запиту до API відновлення даних з обраним файлом. API звертається до бази даних MongoDB та виконує операцію відновлення на основі отриманих даних. Якщо відновлення пройшло успішно, API надсилає відповідь з HTTP статусом 200 та повідомленням про успішне відновлення. У випадку невдачі відновлення, API надсилає відповідь з HTTP статусом 500 та повідомленням про помилку. Якщо адміністратор намагається використати непідтримуваний метод (наприклад, HTTP GET), API відновлення даних надсилає відповідь з HTTP статусом 405 та повідомленням про непідтримуваний метод. Діаграма послідовності резервного відновлення даних показана на рисунку 6.21.

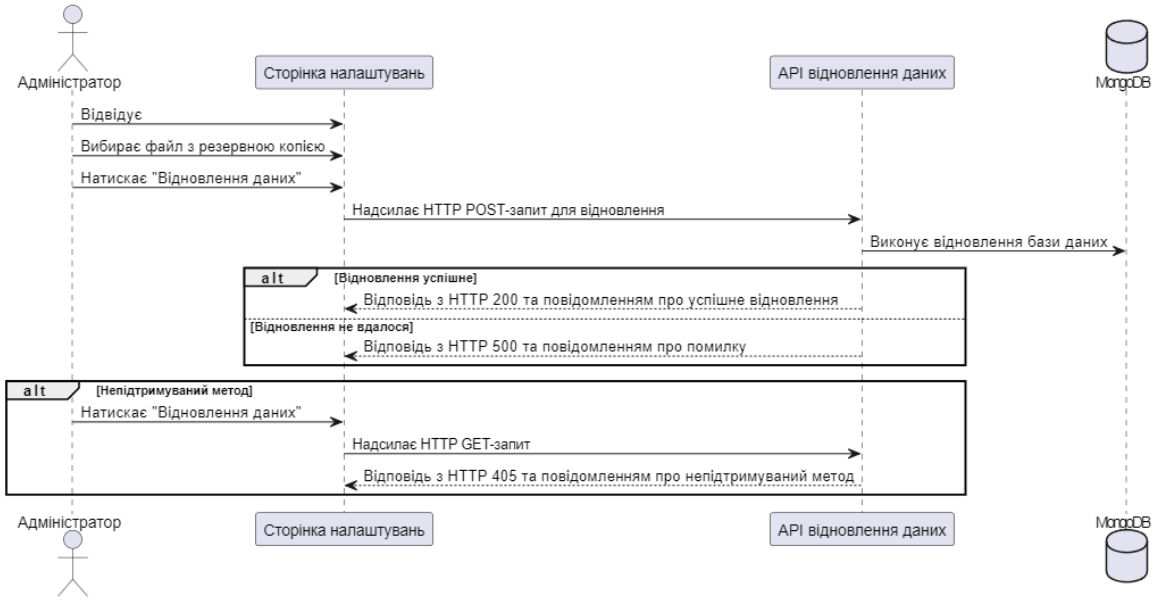

Рисунок 6.21 – Діаграма послідовності резервного відновлення даних

7 ПРОЄКТУВАННЯ СТРУКТУРИ БАЗИ ДАНИХ

Проєктування структури бази даних є ключовим аспектом нашого проекту з числа причин. Во-перше, це допомагає оптимально організувати дані для забезпечення їх ефективного зберігання та доступності, забезпечуючи консистентність та ефективність системи. Во-вторих, правильна структура даних відповідає бізнес-вимогам та потребам користувачів, забезпечуючи легкий доступ до необхідної інформації. Крім того, це дозволяє уникнути проблем з підтримкою та масштабуванням системи в майбутньому.

Наша база даних складається з 4 колекцій: accounts, categories, courses та quizzes. Щодо першої колекції categories, її можна описати наступним чином:

Кожна категорія має унікальний ідентифікатор **_id** типу objectId, **name** назву типу String, **categoryImage** зображення категорії як масив рядків Array[String], **courses** курси як масив objectId Array[objectId] який є зовнішнім ключем *FK*, і **__v** типу Int32, який представляє ключ версії. Це поле за замовчуванням, яке додається Mongoose до кожного документа в MongoDB при використанні схемних оновлень. Це поле відстежує версію документа, яка збільшується кожен раз при оновленні курса.

Кожний курс має унікальний ідентифікатор **_id** типу objectId, **title** назву типу String, **description** опис типу String, **bodytext**, що представляє вміст курсу, як тип String, **youtubeUrl** як String, **images** зображення типу Array[String], **category** категорію як зовнішній ключ FK, що представляє батьківську категорію цього курсу, як тип objectId, та **createdAt** і **updatedAt**, що показують час створення і останнього оновлення курсу, як тип Date або TIMESTAMP, і **__v** типу int, що представляє ключ версії.

Відношення між цими колекціями (**category/course**) полягає в тому, що кожна **category** категорія може мати від 0 до багатьох пов’язаних **course**

курсів, а курс має унікальну категорію, тому відношення є один-до-багатьох (**1:n).** Діаграма зв'язку сутностей між колекціями "categories: та "courses" представлена на рисунку 7.1.

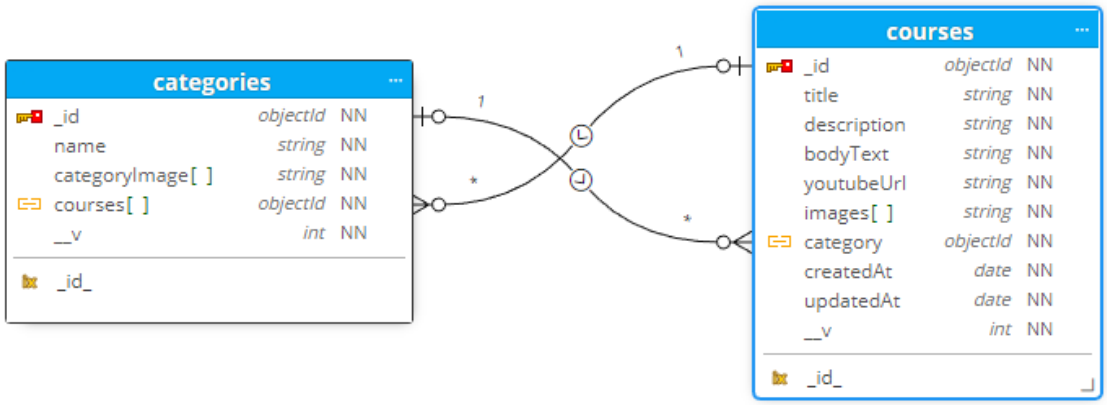

Рисунок 7.1 – Діаграма зв'язку сутностей між категоріями та курсами

Існують кілька методів і функцій, пов'язаних з категоріями. Ці функції виконують різні операції, пов'язані з категоріями, зокрема отримання, редагування, збереження, видалення та завантаження зображень. Ці методи і функції показані на рисунку 7.2.

fetchCategories():

Ця функція виконує запит до API для отримання категорій і сортує їх за алфавітом. Після отримання даних з API, вони зберігаються у стані компонента categories.

handleImageClick(image):

Ця функція викликається при кліку на ескіз зображення категорії. Вона встановлює вибране зображення для подальшого перегляду у модальному вікні.

handleCloseModal():

Ця функція закриває модальне вікно зображення категорії, встановлене за допомогою функції **handleImageClick()**.

saveCategory(ev):

Ця функція обробляє відправку форми для збереження або оновлення категорії. Вона перевіряє обов’язкові поля та надсилає дані до сервера за допомогою *POST* або *PUT* запитів, залежно від того, чи редагується існуюча категорія чи створюється нова.

editCategory(category):

Ця функція встановлює вибрану категорію для редагування. Вона заповнює поля форми даними про вибрану категорію.

deleteCategory(category):

Ця функція підтверджує видалення категорії та надсилає відповідний запит на сервер для видалення категорії. Після видалення категорії, виконується оновлення списку категорій.

uploadImage(ev):

Ця функція обробляє завантаження зображення для категорії. Вона відправляє зображення на сервер, отримує посилання на нього, та зберігає це посилання у стані **categoryImage**.

deleteAllCategories():

Ця функція підтверджує видалення всіх категорій і відправляє відповідний запит на сервер для видалення всіх категорій. Після цього вона оновлює список категорій у стані компонента.

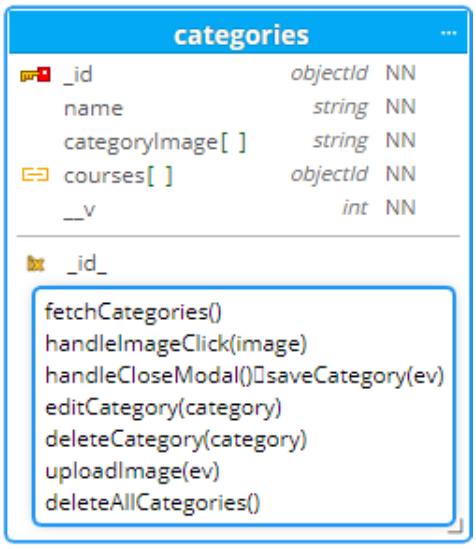

Рисунок 7.2 – Діаграма класу Категорії

Як описано на рисунку 7.4, модель курсу визначає структуру даних, що пов’язані з курсами, і операції, що виконуються з ними. Кожен курс має id (унікальний), назву **title**, опис **description**, основний зміст **bodyText**, посилання на YouTube **youtubeUrl**, зображення **images**, категорію **category**, а також мітки часу **createdAt** та оновлення **updatedAt**, які представляють час створення та останнього оновлення. Модель надає наступні методи для повного управління курсами в нашій системі: **NewCourse()** для створення, **deleteCourse()** для видалення, **EditCoursePage()** для редагування існуючих курсів. **fetchData()** отримує дані курсів з бази даних MongoDB, **handlePageChange()** реалізує пагінацію, яка відображає список курсів в організованому стилі, а **deleteAllCourses()** видаляє всі курси. Створення курсів відбувається після послідовності перевірок для забезпечення кращого досвіду користувача: **validateForm()** і **validateYoutubeUrl()**. Збереження нових та оновлених курсів відбувається за допомогою **saveCourse()**, а **uploadImage()** відповідає за завантаження зображень. Додаткові методи, такі як **handleCancel()**, **handleCloseModal()**, **handleButtonClick()** і **handleListBtnClick()**, надають зручність адміністраторам, дозволяючи скасовувати дії, закривати модальні вікна та форматувати текст в редакторі курсу.

Методи моделі Course:

NewCourse(): Цей метод ініціалізує процес створення нового курсу шляхом відображення форми для адміністратора з деталями, такими як назви, описи та інше. Після збереження даних або підтвердження їх, він обробляє процес збереження нових даних курсу.

deleteCourse(): Ця функція використовується для видалення курсу з бази даних. Адміністратор викликає її після натискання кнопки видалення, пов’язаної з конкретним курсом. Після видалення вона видаляє дані про курс з бази даних і перенаправляє адміністратора на головну сторінку курсів.

EditCoursePage(): Ця функція викликається, коли адміністратори натискають кнопку для редагування конкретного курсу. Вона перенаправляє

на форму редагування, де вони можуть змінити деталі існуючого курсу. Зазвичай це та сама функція, що і NewCourse(), але використовує інший HTTP-запит (PUT). Вона завантажує поточні дані курсу у форму редагування, дозволяючи адміністраторам вносити зміни.

fetchData(): Ця функція викликає дані, що стосуються курсів, з бази даних MongoDB. Вона може отримувати список всіх курсів разом з їх деталями, включаючи назви, описи, зображення тощо. Зазвичай ця функція викликається за допомогою HTTP-запиту GET.

handlePageChange(obj): Ця функція є допоміжною (не головною) та керує пагінацією для списку курсів. Під час розробки цієї функціональності я використовував бібліотеку React *react-paginate*.

deleteAllCourses(): Ця функція виконує асинхронний запит на видалення всіх курсів з MongoDB за допомогою *axios*, надсилаючи HTTP-запит DELETE на API курсів з параметром запиту **all**, встановленим на true. Після успішного видалення вона викликає **fetchData()**, щоб оновити дані, відображені у таблиці, після чого таблиця стає порожньою.

validateForm(): Ця функція перевіряє, що введені адміністратором дані при створенні або редагуванні курсу є коректними. Вона перевіряє, що поля назви та категорії не є порожніми, що обрані зображення, і що в полі YouTubeURL немає помилок. Якщо всі обов’язкові поля заповнені і дані мають правильний формат, функція повертає true, що вказує на те, що форма є дійсною, тому кнопка "Зберегти" активна. У протилежному випадку вона повертає false, що викликає відображення повідомлень про помилки, щоб керувати адміністраторів у виправленні їх введення.

validateYoutubeUrl(): Ця функція перевіряє, чи формат тексту, введеного у поле YouTubeUrl, відповідає формату посилання з Youtube (тому що це поле повинно дозволяти адміністраторам додавати лише посилання з YouTube). Спочатку вона перевіряє, що поле YouTubeUrl не є порожнім, а потім використовує регулярні вирази *RegExp*, щоб перевірити, чи посилання відповідає одному з прийнятних форматів. Якщо посилання схоже на один з

прийнятних форматів, вона встановлює youtubeUrlError на null, вказуючи, що URL є дійсним, в іншому випадку вона встановлює youtubeUrlError на "Неправильний URL YouTube". Ця функція допомагає переконатися, що наданий URL YouTube має правильний формат і вказує на дійсність відео YouTube, відображаючи повідомлення про помилку, якщо він не є вірним. Нижче, на рисунку 7.3, представлені прийнятні формати:

```
"https://www.youtube.com/",
"http://www.youtube.com/",
"https://youtube.com/",
"http://youtube.com/",
"youtube.com/",
"www.youtube.com/",
"m.youtube.com/",
"youtu.be/",
```

Рисунок 7.3 – Приклад прийнятного формату посилань

saveCourse(ev): Ця функція відповідає за процес збереження змін, внесених до курсу, чи це створення нового курсу, чи оновлення існуючого. Вона викликається після натискання кнопки "Зберегти курс". Під час збереження курсу, цю функцію оновлює дані курсу в базі даних MongoDB. Спочатку вона блокує типову поведінку відправки форми **ev.preventDefault()** і встановлює **showErrors** на **true**, щоб забезпечити відображення помилок валідації (якщо є які-небудь помилки). Потім вона перевіряє наявність помилок валідації, пов’язаних з **назвою**, **категорією**, **зображеннями** та **YouTubeURL**, і встановлює їх у об’єкті **errors**, якщо вони виникають. Якщо є які-небудь помилки, вона оновлює стан з помилками валідації та повертається, щоб не дозволити адміністратору зберегти курс. Якщо помилок валідації немає, вона створює об’єкт data з деталями курсу, включаючи назву, опис, YouTube URL, зображення, категорію та текст. Якщо у курсу вже є *_id*,

вона відправляє HTTP PUT-запит до **API курсів** для оновлення деталей курсу, а у випадку створення нового курсу вона відправляє HTTP POST-запит. Після успішного збереження вона встановлює goToCourses на true, що перенаправляє адміністратора на головну сторінку курсів. Якщо під час процесу збереження виникає помилка, вона реєструє цю помилку у консолі.

uploadImage(ev): Ця функція використовується для завантаження зображень для курсу. Вона використовується в компоненті форми курсу і використовує компонент завантаження зображень, що дозволяє адміністраторам вибирати зображення зі свого пристрою. Після натискання кнопки "Завантажити зображення" вона визначає поле вибору файлу, яке дозволяє вибирати файли, перевіряє, чи вибрано зображення, і якщо це файл зображення, починає процес завантаження. Усередині функції вона створює об’єкт FormData з вибраним файлом і відправляє HTTP POST-запит на Upload API для завантаження зображення на сервер **AWS**. У **Upload API**, серверний код обробляє процес завантаження файлу. Спочатку він підключається до **MongoDB** та перевіряє, чи запит від адміністратора. Потім він створює екземпляр форми multiparty для розбору даних форми, що містять завантажений файл. Файли потім завантажуються до AWS S3 Bucket за допомогою SDK для JavaScript *@aws-sdk/client-s3*. Завантажений файл має унікальне ім’я файлу і зберігається у AWS S3 Bucket з ACL, встановленим на **public-read**. Після успішного завантаження функція повертає масив згенерованих посилань S3 для завантажених зображень. Ці посилання потім використовуються для асоціації зображень з відповідним курсом.

handleCancel(): Ця функція відповідає за скасування дій, пов’язаних з створенням або редагуванням курсу. Вона викликається для закриття поточної форми або модального вікна без збереження будь-яких змін, внесених адміністратором, при натисканні на кнопку "Скасувати". Вона перенаправляє адміністратора на головну сторінку курсів ‘/courses’.

handleCloseModal(): Ця функція відповідає за закриття модальних діалогових вікон, використовуваних у процесі управління курсами. Вона

спрацьовує при натисканні адміністратором на кнопку закриття вікна перегляду зображення. У цій реалізації вона скидає змінну стану **selectedImage** на **null**, що ефективно закриває модальний діалог. Ця функція гарантує, що будь-яке обране зображення в модальному вікні деактивується при його закритті.

handleButtonClick(String, String): Ця функція приймає два параметри: **startTag** і **endTag**, які представляють відкриваючий і закриваючий теги, які мають бути вставлені навколо вибраного тексту. Усередині функції вона отримує елемент текстової області "**bodyText**" і отримує поточний текст з нього. Потім вона визначає початковий і кінцевий індекси вибраного тексту у межах текстової області. Функція зберігає вибраний текст і отримує текст перед і після вибраного діапазону. Потім вона конструює новий рядок тексту, вставляючи надані **startTag** і **endTag** навколо вибраного тексту, не переносячи на новий рядок. Після модифікації тексту вона встановлює оновлений текст як нове значення **bodyText**, а потім відновлює вибір, встановлюючи діапазон вибору таким чином, щоб включати вставлені теги, забезпечуючи, що курсор залишається в тому ж самому положенні відносно вибраного тексту.

handleListBtnClick(): Ця функція керує кліками на кнопки списків, пов’язаних з курсами. Вона форматує вибраний текст і перетворює його в список у текстовому полі контенту курсу. Усередині функції вона отримує текстову область "**bodyText**" та отримує з неї вибраний текст. Потім вона визначає його початок і кінець, зберігає вибраний текст і отримує текст перед і після вибраного діапазону. Потім перевіряє, чи містить вибраний текст розриви рядків **\n**. Якщо вибрано кілька рядків, вона створює список, розбиваючи вибраний текст за розривами рядків, фільтруючи порожні рядки, і відображаючи кожен рядок як пункт списку **<li>**. Якщо вибрано лише один рядок, вона створює один пункт списку. Після форматування пунктів списку вона вставляє їх у текст, обгортаючи їх тегами **<ul>**. Вона встановлює модифікований текст як нове значення **bodyText**. Нарешті, вона відновлює

вибір, встановлюючи діапазон вибору таким чином, щоб включити вставлений тег **<ul>**, забезпечуючи, що курсор залишається на тому ж самому місці відносно вибраного тексту.

Властивості:

- **id:** унікальний ідентифікатор для кожного курсу;
- **title**: Представляє назву курсу;
- **description**: Опис курсу;
- **bodyText**: Містить основний текстовий зміст курсу;
- **youtubeUrl**: Зберігає URL YouTube, пов’язаний з курсом;
- **images**: Масив, що містить зображення, пов’язані з курсом;
- **category**: ID категорії, до якої належить курс;
- **createdAt**: представляє мітку часу *TIMESTAMP* для відображення дати та часу створення курсу;
- **updatedAt**: редставляє мітку часу *TIMESTAMP* для відображення дати та часу останнього оновлення курсу.

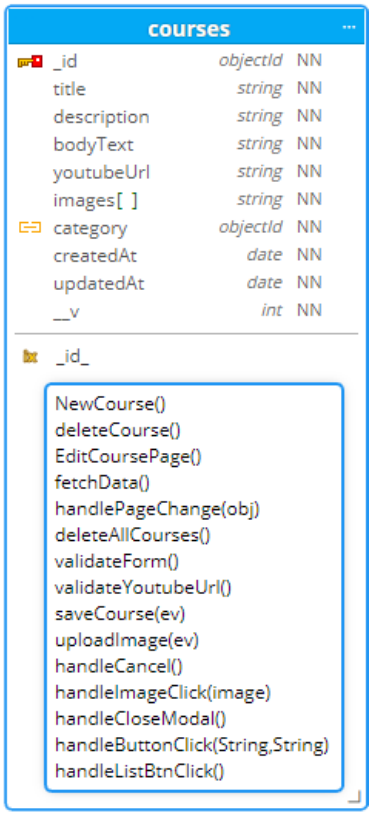

Рисунок 7.4 – Діаграма класу Курси

Далі, як описано на рисунку 7.5, ми детально обговоримо модаль **Quiz**, у веб-застосунку CodeCraftingLab адміністратори можуть керувати квізами за допомогою модального вікна. Після переходу на сторінку квізів процес розпочинається з витягування останніх даних квізів за допомогою функції **fetchQuizzes**. Перш ніж зберігати будь-які зміни, функція **validateForm** перевіряє, чи всі обов'язкові поля заповнені правильно. Для внесення змін адміністратори можуть легко змінювати деталі квізу за допомогою функцій, таких як **handleInputChange**, **handleQuestionChange**, **handleChoiceChange** та **handleCorrectChoiceChange**. Як тільки всі зміни внесено, їх можна зберегти за допомогою функції **submitForm**, яка оновлює або додає квізи за необхідності. Якщо адміністратори хочуть почати спочатку, функція clearForm скидає форму для створення нових квізів. Для зручності функції, такі як **goBack** та **deleteQuiz**, дозволяють легко переміщатися та видаляти квізи. **EditQuizPage** забезпечує функціональність для редагування існуючих квізів, а **NewQuiz** дозволяє створювати нові квізи. З цими методами і функціями, інтегрованими в систему, адміністратори можуть легко керувати квізами в межах CodeCraftingLab, забезпечуючи плавний досвід навчання для наших користувачів. Функціональності моделі квізу та пов'язані з ними методи для керування квізами в системі такі:

fetchQuizzes(): Цей метод асинхронно отримує квізи з сервера, виконуючи HTTP **GET**-запит на **quizzes API**. Він оновлює стан з отриманими квізами за допомогою setQuizzes, щоб забезпечити, що стан квізів завжди має останні дані з сервера.

validateForm(): Ця функція перевіряє, чи дані форми є валідними перед їхнім надсиланням. Вона перевіряє, чи вибрано рівень, і чи у кожного питання є текст та принаймні один варіант відповіді. Вона встановлює помилки валідації форми для кожного поля та повертає булеве значення **boolean**, що показує, чи форма є валідною.

handleInputChange(event): Цей метод відповідає за зміни в полях вводу, оновлюючи стан форми на основі введення користувача. Він

відслідковує подію **onChange** полів вводу та оновлює відповідне значення поля в стані форми.

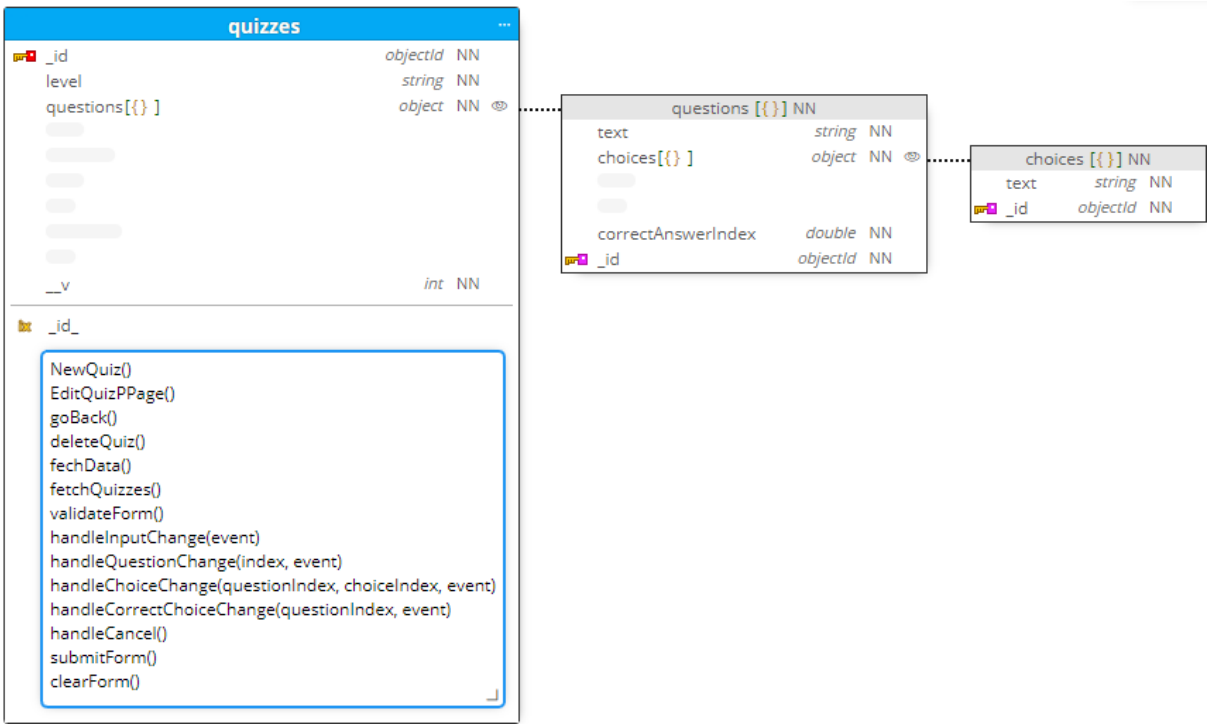

Рисунок 7.5 – Діаграма класу Квізи

handleQuestionChange(index, event): Відповідає за оновлення тексту питання в формі. Він відслідковує зміни в текстовому введенні для кожного питання та оновлює стан відповідно.

handleChoiceChange(questionIndex, choiceIndex, event): Ця функція оновлює тексти варіантів відповідей для кожного питання в формі. Вона відслідковує зміни в текстових полях вибору варіантів відповідей та оновлює стан, відображаючи ці зміни.

handleCorrectChoiceChange(questionIndex, event): Цей метод відповідає за зміни у виборі правильної відповіді в формі. Він оновлює стан з індексом правильної відповіді для кожного питання.

handleCancel(): Викликається, коли адміністратор скасовує надсилання форми або редагування. Він повертається на сторінку квізів за допомогою маршрутизатора Next.js **router.push()**, ефективно скасовуючи поточну операцію.

submitForm(event): Відповідає за надсилання форми, спочатку він перевіряє дані форми за допомогою validateForm. Якщо форма валідна, він надсилає HTTP **POST**-запит для нових квізів або HTTP **PUT**-запит для редагування на сервер. Після успішної відправки він отримує оновлені дані квізів, щоб забезпечити їхню консистентність.

clearForm(): Ця функція очищає форму, скидаючи її до початкового стану. Вона встановлює значення за замовчуванням для рівня, питань та варіантів відповідей, ефективно очищаючи будь-який ввід користувача.

goBack(): Використовується для повернення на сторінку квізів, коли користувач скасовує процес видалення. Вона використовує маршрутизатор Next.js **router.push()** для перенаправлення на сторінку квізів.

deleteQuiz(): Для видалення квізу ця функція надсилає HTTP **DELETE**-запит на **Quizzes API** для видалення вказаного квізу. Після успішного видалення вона повертається на сторінку квізів.

EditQuizPage(): Ця компонента функція відображає сторінку для редагування квізу. Вона отримує дані квізу з сервера на основі параметра **id**, а потім передає дані до компонента **QuizForm** для редагування.

NewQuiz(): Відповідає за відображення сторінки для створення нового квізу та включає компонент **QuizForm** для введення деталей квізу. Вона надає чистий інтерфейс для користувачів для створення нових квізів.

У системі CodeCraftingLab доступ до адмін-панелі через **Google API** здійснюється миттєво. При відвідуванні веб-сайту, якщо немає сеансу, користувач перенаправляється на сторінку входу в систему, де у нього є можливість авторизуватися за допомогою Google. Натискання кнопки "Увійти за допомогою Google"запускає функцію входу ("Google"), а Google API запускає процес автентифікації.

У роботі використано **NextAuth** для автентифікації та, в основному, **GoogleProvider** для *Google OAuth*. У API **NextAuth** ми визначаємо наші механізми автентифікації, включаючи ідентифікатор клієнта Google та секретний ключ, який отримали від Google Cloud Console. Властивість

зворотних викликів використовується для перевірки того, чи включена адреса електронної пошти користувача до списку електронної пошти адміністратора. Якщо це так, сеанс користувача може бути продовжений; інакше він відмовляється переходити на нього.

Щоб перевірити, чи доступні певні методи чи дії лише користувачам з правами адміністратора, У мене є функція **isAdminRequest**, яка перевіряє, чи надходить запит від користувача з правами адміністратора. Ця функція використовує функцію **getServerSession**, надану **NextAuth**, для отримання інформації про сеанс користувача з сервера. Якщо адреса електронної пошти користувача відсутня в списку електронної пошти адміністратора, завдання видає статус несанкціонованого доступу 401 і видає помилку.

У файлі **.env** ми зберігаємо важливу інформацію, таку як **Google client ID** та **secret**, а також **NextAuth secret**. Ці облікові дані використовуються в процесі автентифікації та для налаштування підключень до зовнішніх служб. Весь процес показано нижче на рисунку 7.6.

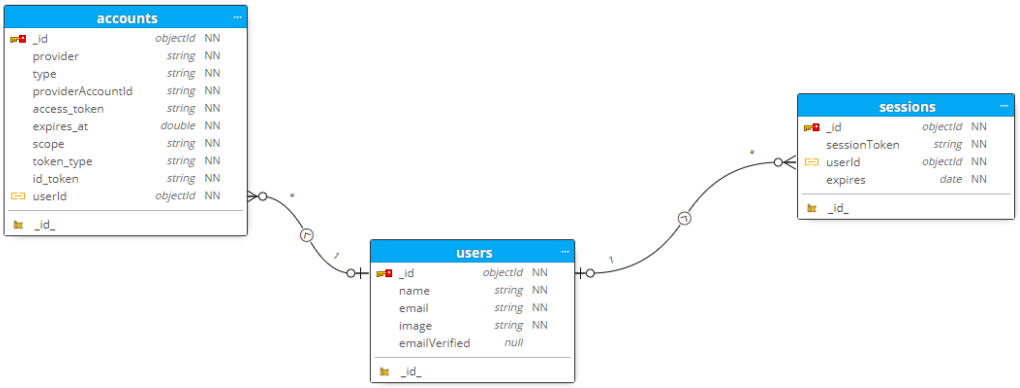

Рисунок 7.6 – Діаграма класу Користувачі / Рахунок / сеанси і зв’язок між ними

На цьому етапі у нас є все необхідне для початку реалізації проекту.

8 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОЄКТУ

Реалізація нашого вебзастосунку розділена на два основні етапи, перший крок – це впровадження панелі адміністратора, другий крок-це впровадження головної сторінки та підключення двох компонентів до моделей mongodb. Перш за все, ми створюємо проект Nextjs. у VSCode, ми створюємо новий проект **codecraftinglab-admin,** потім ми використовуємо команду *yarn create next-app codecraftinglab-admin*, ця команда створює новий Next.JS додаток з назвою "codecraftinglab-admin". Налаштування проекту показані на рисунках 8.1, 8.3 – 8.6.

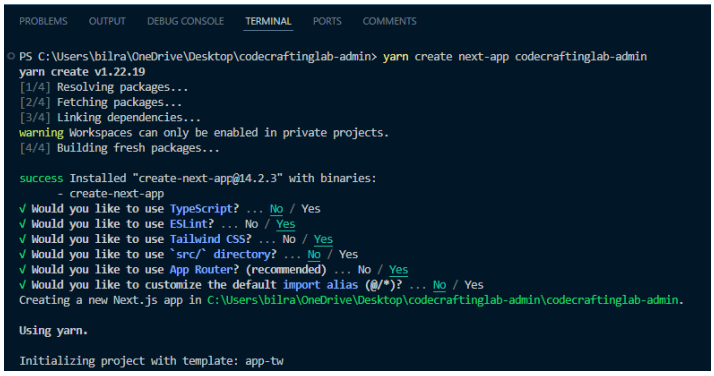

Рисунок 8.1 – Створення нового проекту Nextjs за допомогою менеджеру yarn

Початковий проект успішно створено і має структуру, яка показана на рисунку 8.2:

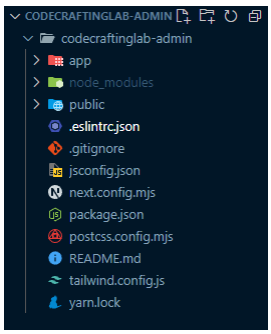

Рисунок 8.2 – Структура програми

Потім ми встановлюємо Tailwind css за допомогою команди *yarn add tailwindcss postcss autoprefixer*

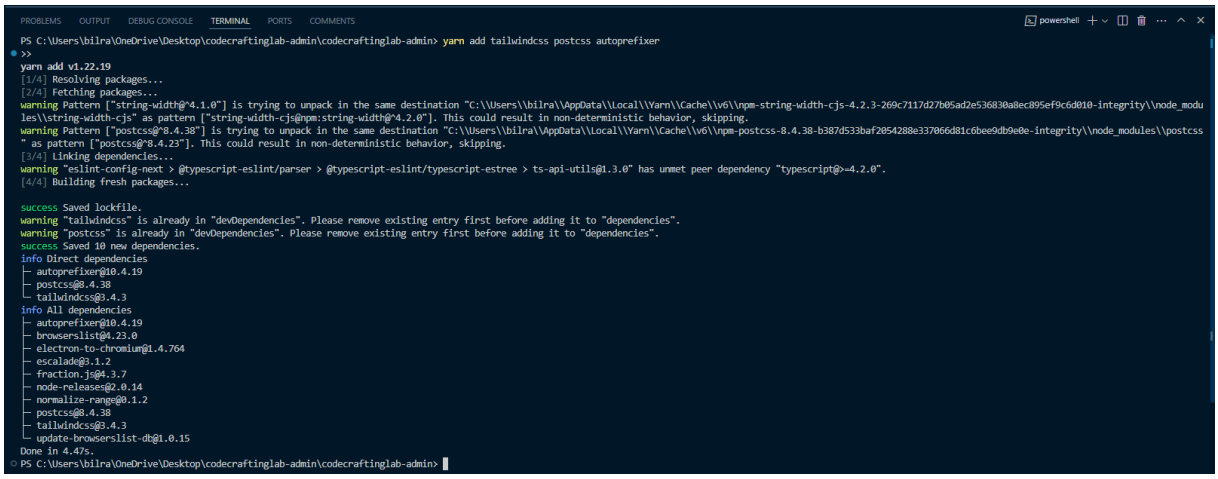

Рисунок 8.3 – Налаштування проекту для використання TailwindCSS

Потім запускаємо наступну команду *npx tailwindcss init –p*

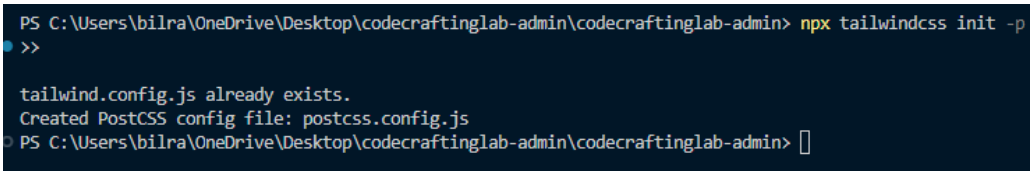

Рисунок 8.4 – Ініціалізація Tailwind CSS та генерація повного конфігураційного файлу

Потім додаємо наступну конфігурацію до *tailwind.config.js*, Вказавши ці файли в параметрі content, TailwindCSS дізнається, де шукати класи для створення службових стилів

```
content: [
  "./app/**/*.{js,ts,jsx,tsx,mdx}",
  "./pages/**/*.{js,ts,jsx,tsx,mdx}",
  "./components/**/*.{js,ts,jsx,tsx,mdx}",
],
```

Рисунок 8.5 – Конфігурація файлу tailwind.config

Рисунок 8.6 – Ініціалізація Tailwind CSS та генерація повного конфігураційного файлу

Ми додаємо рядки, зображені на рисунку 8.7, до файлу *styles\globals.css*

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Рисунок 8.7 – Імпорт базових стилів Tailwind CSS

Тепер можемо розпочати реалізацію сторінки входу , створюємо файл Layout.JS всередині папки components *components/Layout.js* , цей файл міститиме весь макет сторінки входу, а також бічне меню навігації. Кінцевий результат сторінки входу відображений на рисунку 8.8.

Після входу адміністратора відобразиться макет панелі адміністратора, результат показаний на рисунку 8.9.

Сторінка *pages\index.js* відповідає за відображення імені користувача та вітального повідомлення, рисунок 8.10 показує, як буде виглядати привітальне повідомлення.

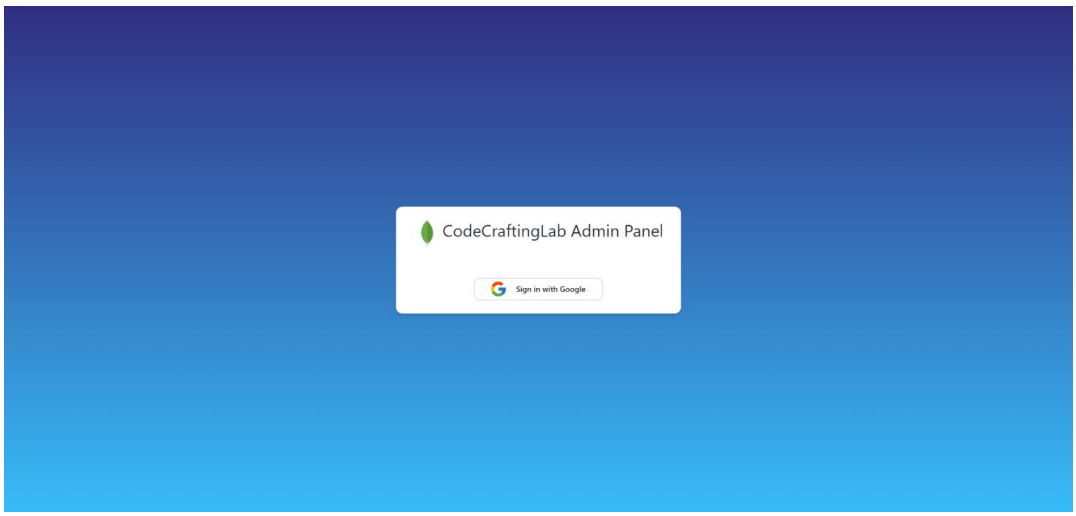

Рисунок 8.8 – Сторінка входу в адмін-панель

Рисунок 8.9 -Адмін-панель

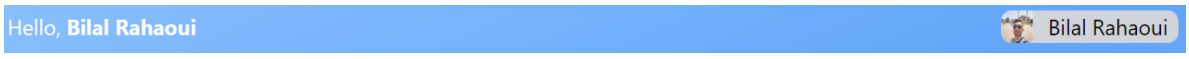

Рисунок 8.10 – Початок сеансу та вітальне повідомлення

Потім, щоб створити навігаційне меню, створюється новий файл *components/nav.js.* Рисунок 8.11 показує навігаційне меню у кінцевому продукті.

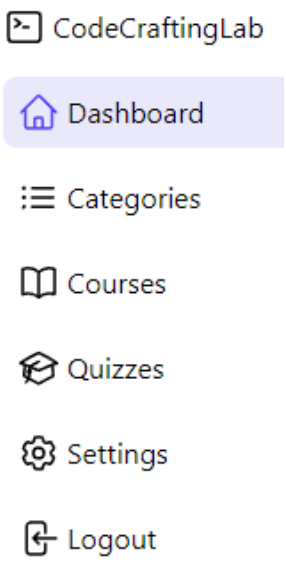

Рисунок 8.11 – Навігаційне меню

Після цього, ми створюємо сторінку **PAGES/COURSES.JS**. Результат цього кроку показаний на рисунку 8.12.

Рисунок 8.12 – Сторінка курсів

Оскільки необхідно, щоб форма відповідальна за заповнення даних курсу, була відокремлена від головної сторінки, створюємо форму курсу в окремому файлі components/CourseForm.js. Результат цього кроку показаний на рисунку 8.13.

Потім, як показано на рисунку 8.14, реалізуємо сторінку категорій. Ця сторінка відповідає за створення, редагування або видалення категорії з бази даних, і оскільки ми хочемо, щоб процес управління категоріями здійснювався на тій самій сторінці, створюємо файл *pages/categories.js*.

Рисунок 8.13 – Форма курсів

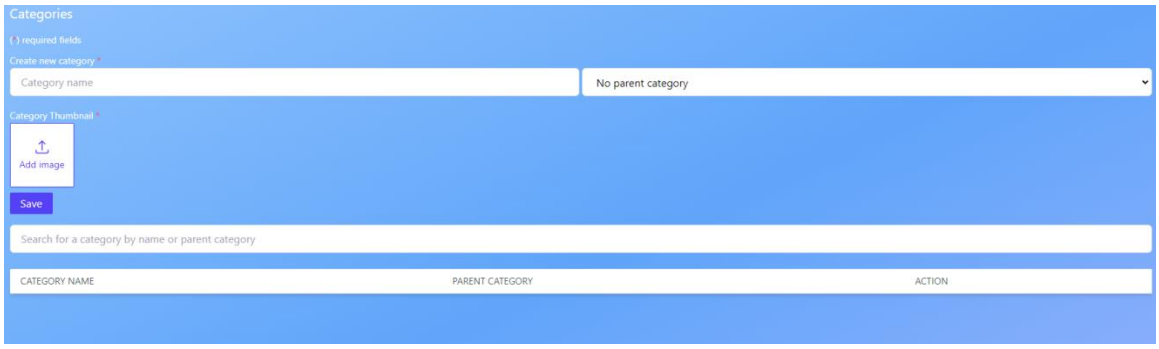

Рисунок 8.14 – Форма категорій

Після цього, як показано на рисунку 8.15, реалізуємо сторінку квізів, в цьому модулі, оскільки ми хочемо керувати квізами (створювати/редагувати) в окремій формі, ми створюємо дві сторінки *pages/quizzes.js*, відповідальний за відображення макета сторінки квізу, списку квізів, та *components/QuizForm.js*, відповідальний за управління квізами. як показано на рисунку 8.16.

Рисунок 8.15 – Сторінка квізів

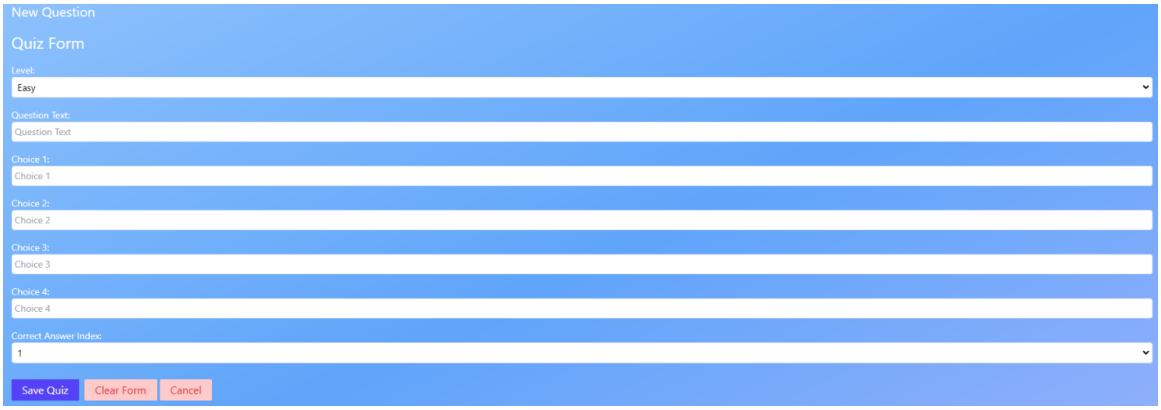

Рисунок 8.16 – Форма квізів

Потім, як показано на рисунку 8.17, реалізуємо сторінку, відповідальну за резервне копіювання / відновлення даних у/з базу даних *pages/settings.js*.

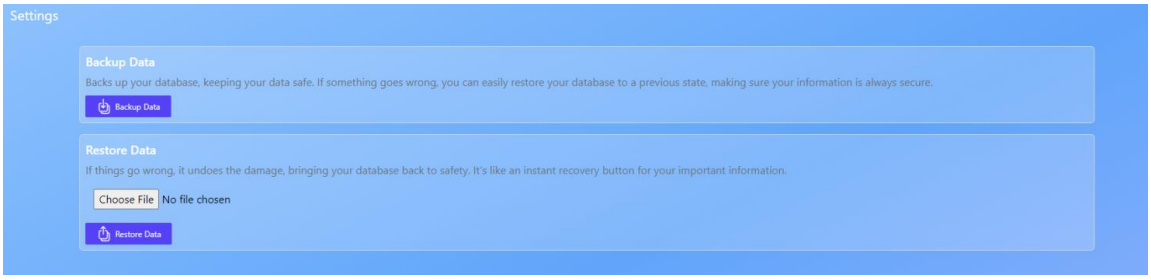

Рисунок 8.17 – Сторінка налаштувань.

Потім, приступаємо до впровадження другої частини продукту *codecraftinglab-front*, головної сторінки, де студенти можуть взаємодіяти з курсами, квізами. Подібно до адміністративної панелі, ми створюємо окремий проект *front* в окрему папку за допомогою команди *yarn create next-app front*. Налаштування проекту демонструється на рисунках 8.18 – 8.22.

Потім ми встановлюємо Tailwind css за допомогою команди *yarn add tailwindcss postcss autoprefixer*.

Потім ми запускаємо наступну команду *npx tailwindcss init -p*.

Потім ми додаємо наступну конфігурацію до *tailwind.config.js*, Вказавши ці файли в параметрі content, TailwindCSS дізнається, де шукати класи для створення службових стилів. Конфігурація доступна на рисунку 8.21.

Рисунок 8.18 – Створення нового проекту Nextjs за допомогою менеджер пакетів yarn

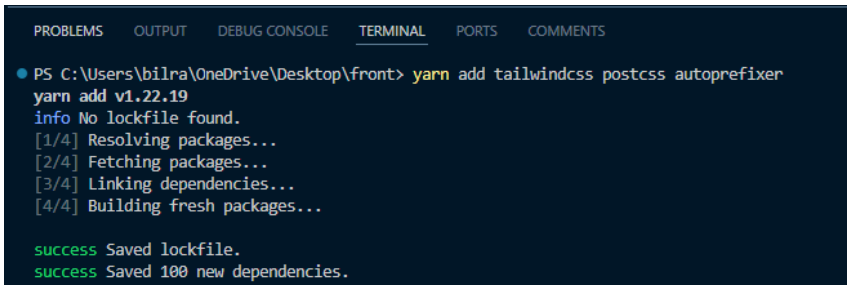

Рисунок 8.19 – Налаштування проекту для використання TailwindCSS

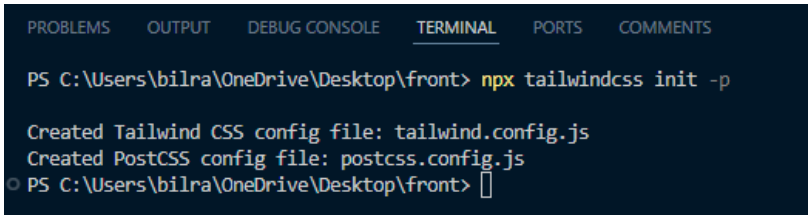

Рисунок 8.20 – Ініціалізація Tailwind CSS та генерація повного конфігураційного файлу

```
content: [
  "./app/**/*.{js,ts,jsx,tsx,mdx}",
  "./pages/**/*.{js,ts,jsx,tsx,mdx}",
  "./components/**/*.{js,ts,jsx,tsx,mdx}",
],
```

Рисунок 8.21 – Конфігурація файлу tailwind.config.

Рисунок 8.22 – Ініціалізація Tailwind CSS та генерація повного конфігураційного файлу

Щоб підключити базу даних до основної програми, копіюємо папку *models/* з проекту *codecraftinlab-admin.* Рисунок 8.23 показує структуру проекту.

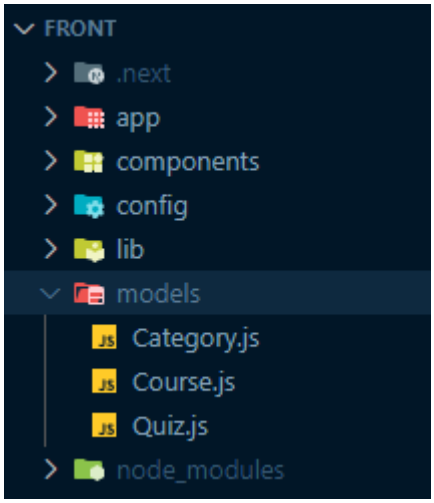

Рисунок 8.23 – Структура проекту

Тепер створюємо домашню сторінку, ця сторінка містить повзунок, який показує швидку навігацію, різні розділи сайту та іншу корисну інформацію про сайт. Ми створюємо іншу частину нашої системи, як-от верхній колонтитул header, нижній колонтитул footer, останні курси LatestCourses і т.д.. Рисунок 8.24 показує структуру папок компонентів.

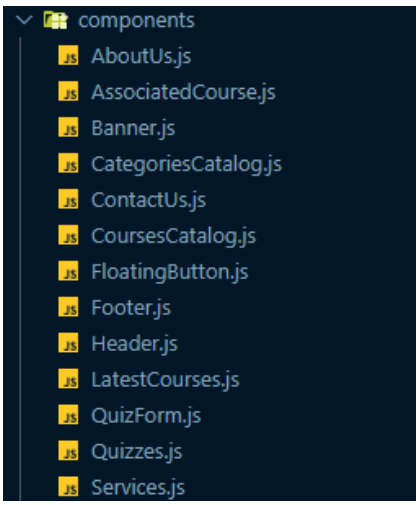

Рисунок 8.24 – Структура компонентів

Щоб використовувати ці компоненти на сайті, ми викликаємо потрібний компонент у *pages/index.js*, наприклад, цей код імпортує верхній колонтитул, банер-повзунок, картки послуг, розділ про нас, картки останніх курсів, форму зв'язку з нами і т.д.

Наприклад, код на рисунку 8.25 імпортує заголовок, банер-повзунок, картки послуг, розділ про нас, картки останніх курсів, форму зв'язку з нами.

```
export default function HomePage({ newCourses }) {
 return (
  <>
   <Header />
   <Banner />
   <Services />
   <AboutUs />
   <LatestCourses courses={newCourses} />
   <ContactUs />
   <FloatingButton />
   <Footer />
  </>
 );
}
```

Рисунок 8.25 – Приклад компонентів використаних в index.js

Таким чином, після додавання курсу до бази даних, він автоматично відображається на головній сторінці в розділі Останні курси, як показано на рисунку 8.26.

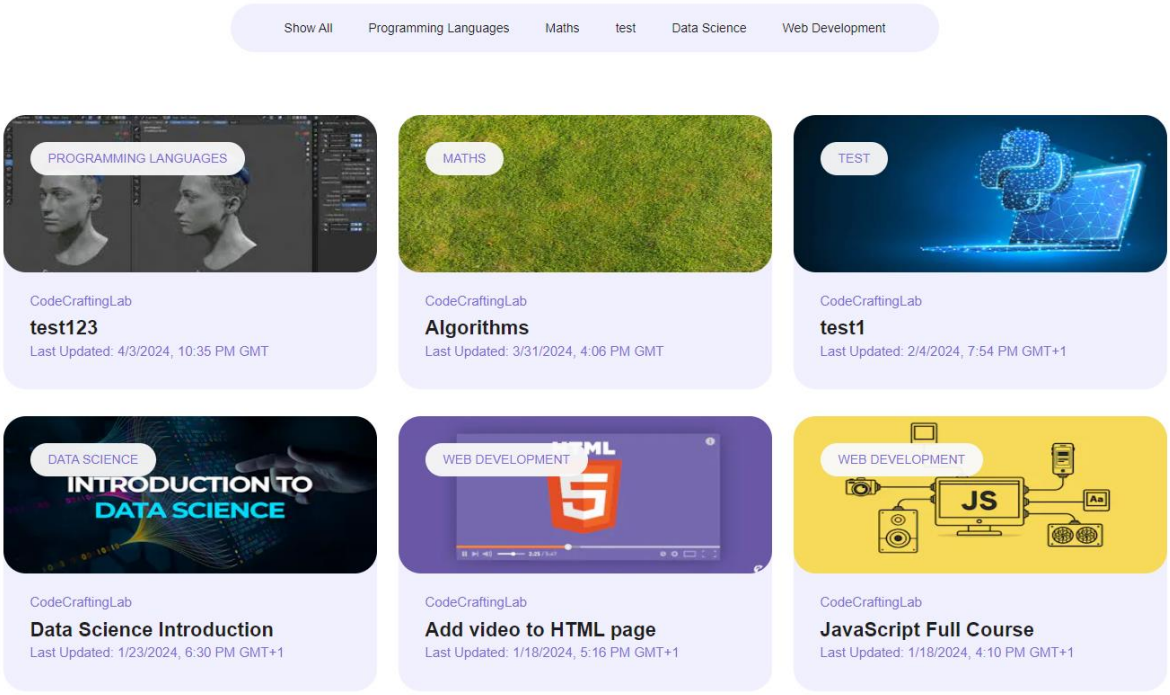

Рисунок 8.26 – Приклад розділу останні курси

Результат створення контактної форми показано на рисунку 8.27.

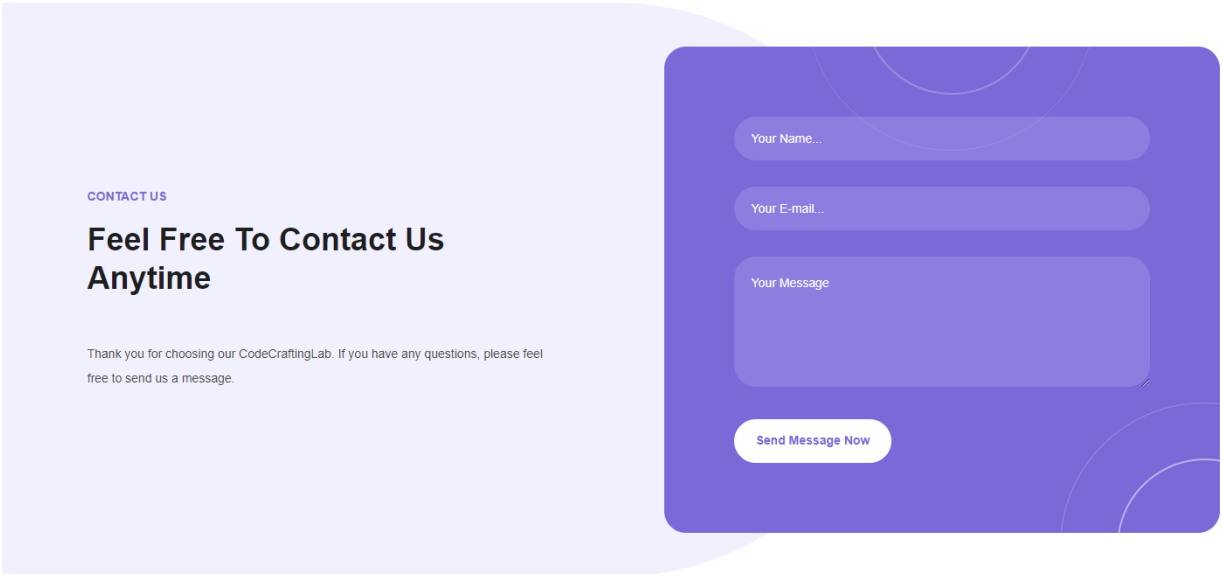

Рисунок 8.27 – Контактна форма

Після переходу на сторінку категорій студент знаходить усі перелічені категорії в базі даних, як показує рисунок 8.28.

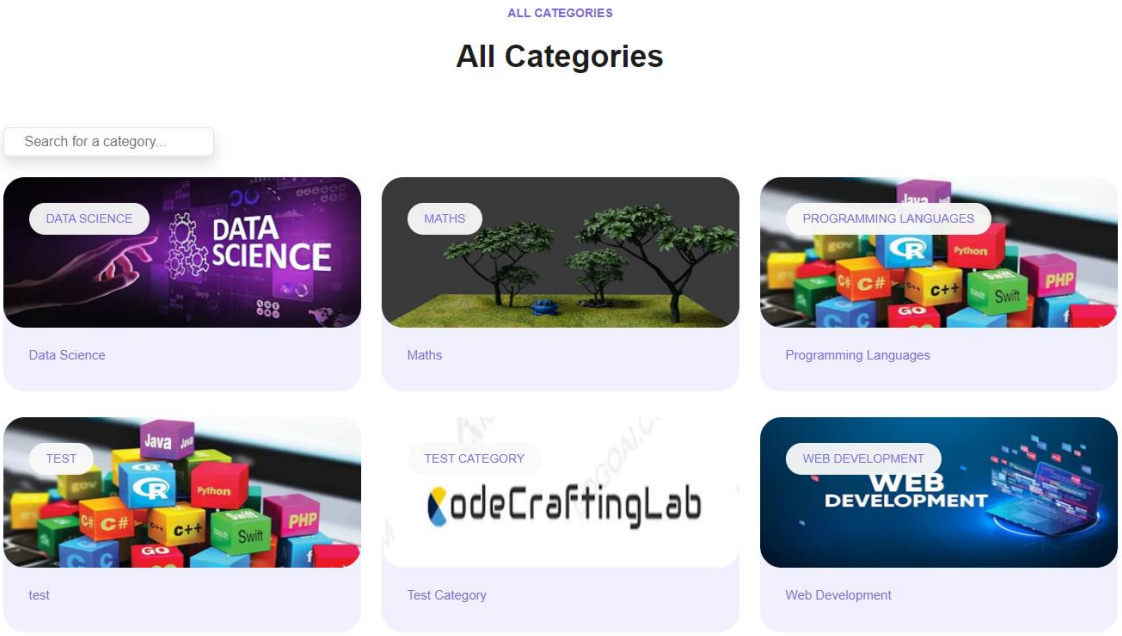

Рисунок 8.28 – Сторінка категорій

Після переходу на сторінку курсів студент знаходить усі перелічені курси в базі даних, як показує рисунок 8.29.

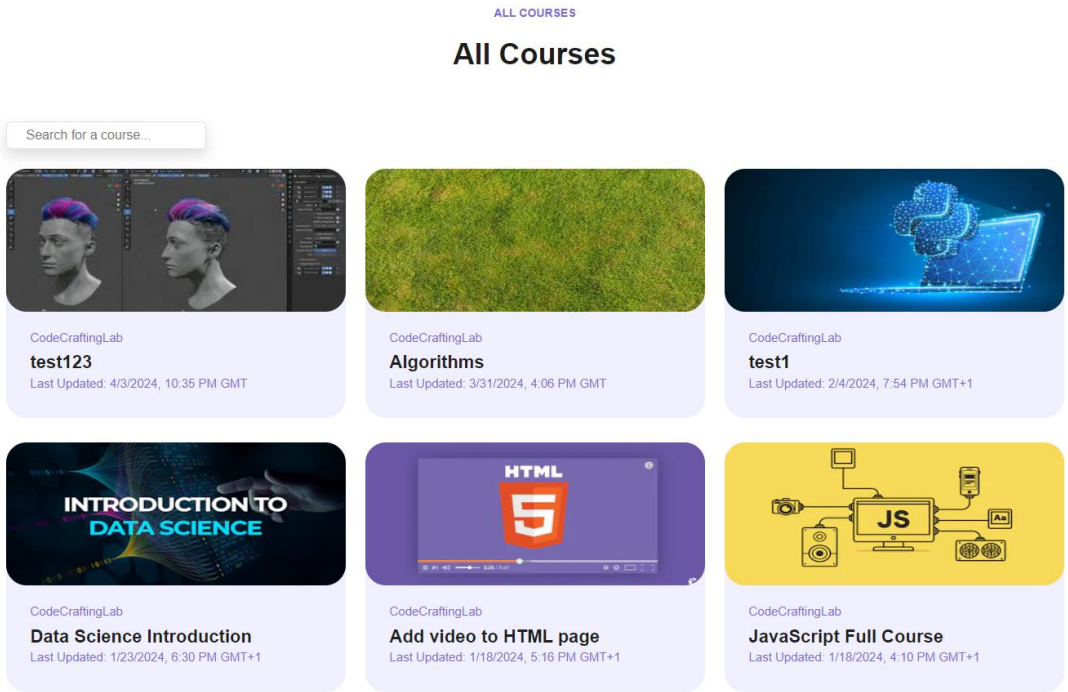

Рисунок 8.29 – Сторінка курсів

Тому, коли студент обирає курс, він перенаправляється на нову сторінку, де він може переглянути вміст курсу, результат цього процесу показано на рисунках 8.30 і 8.31.

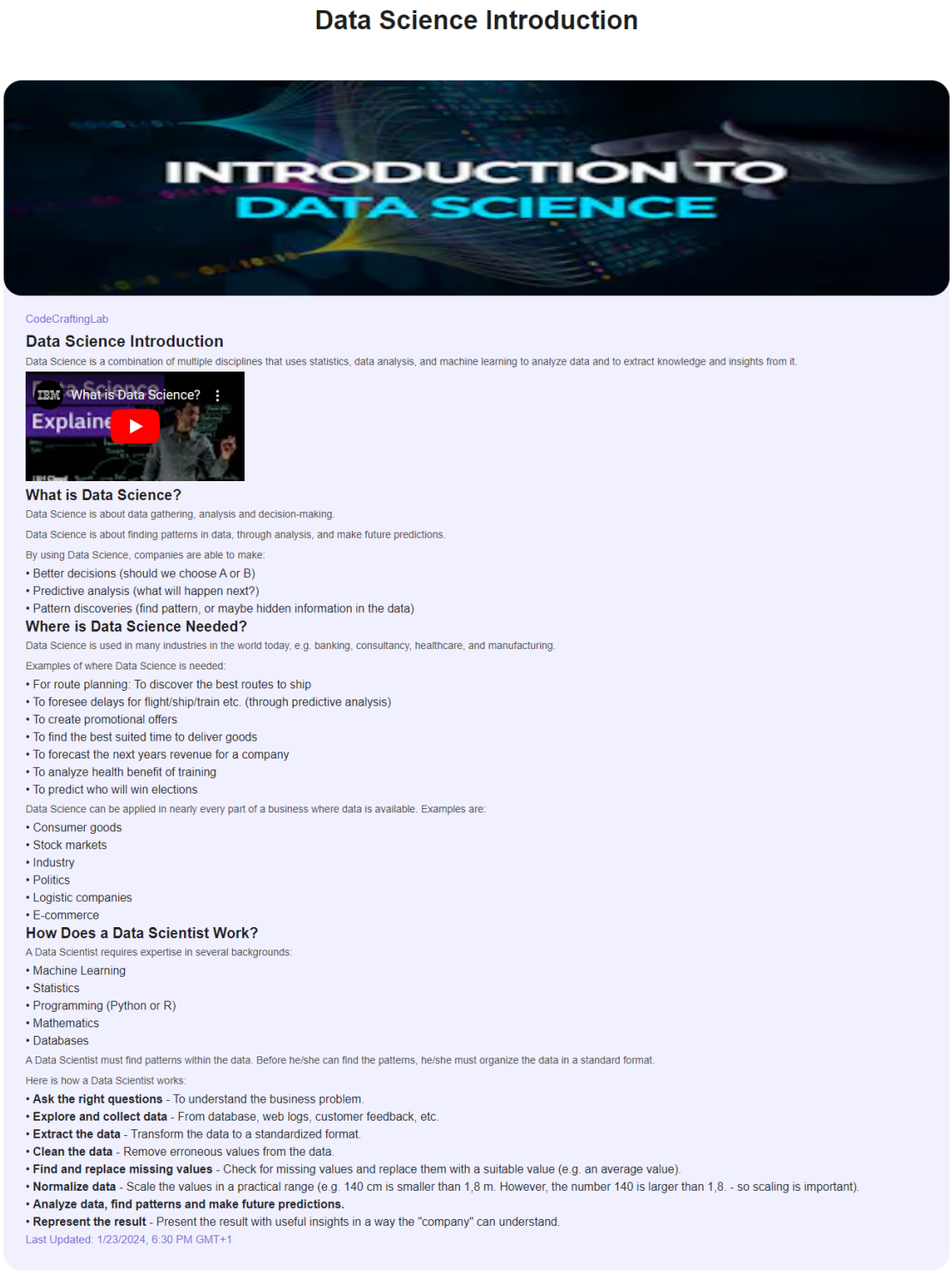

Рисунок 8.30 – Приклад вигляду курсу 1

Рисунок 8.31 – Приклад вигляду курсу 2

Нарешті, як показує рисунок 8.32, студенти можуть проходити квізи для перевірки своїх знань, перш за все, їм потрібно вибрати рівень складності, потім кількість питань (мінімум 5 і максимум 25).

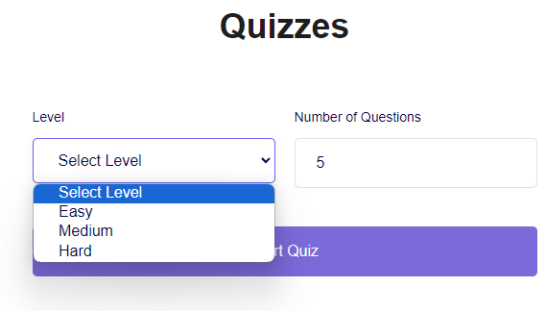

Рисунок 8.32 – Сторінка квізу

Коли студент починає квіз, згідно з його попередніми виборами, квіз починається, показуючи питання з 4 варіантами відповідей, як показує рисунок 8.33, студент повинен вибрати правильний варіант. У випадку правильної відповіді вона підсвічується зеленим кольором. У випадку неправильної відповіді обрана відповідь, а також всі інші неправильні варіанти, підсвічуються червоним кольором, а правильна відповідь – зеленим.

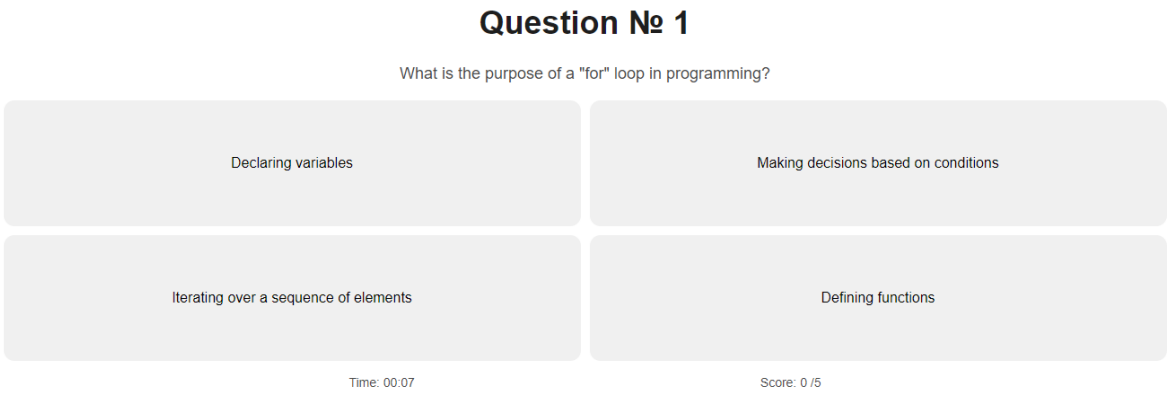

Рисунок 8.33 – Приклад запитання

Перший випадок, коли студент відповідає правильно, приклад показаний на рисунку 8.34.

Тепер у випадку неправильної відповіді, приклад показаний на рисунку 8.35.

Коли студент закінчить, він може побачити свій бал і час, витрачений на відповіді на всі запитання, як показує рисунок 8.36.

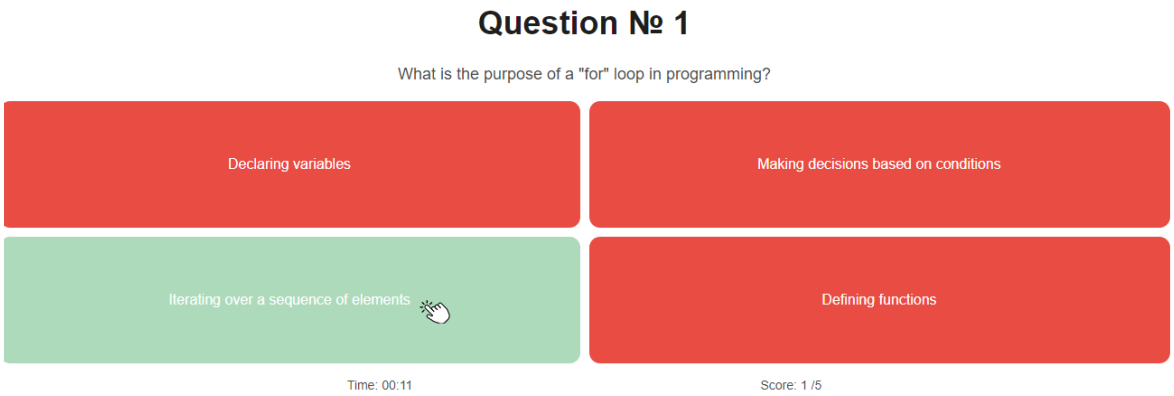

Рисунок 8.34 – Приклад правильної відповіді

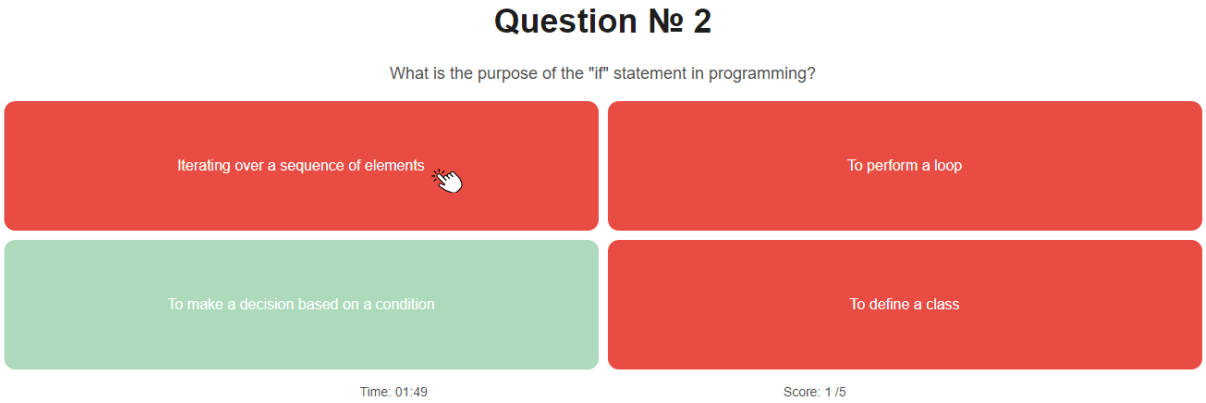

Рисунок 8.35 – Приклад неправильної відповіді

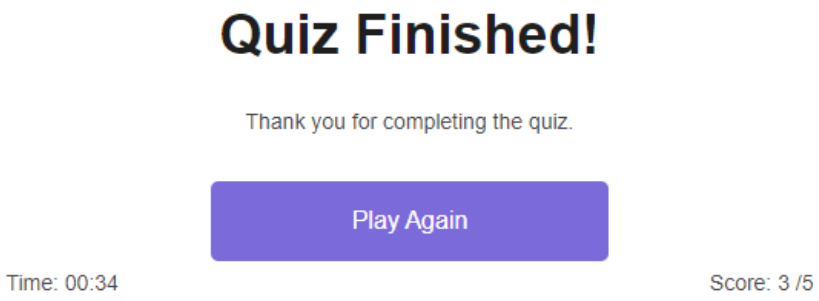

Рисунок 8.36 – Екран завершення квізу

У разі будь-якої технічної проблеми чи питань, користувачі можуть відправити електронний лист через контактну форму. Для цього ми використали NodeMailer API, створюється api *api/contact*, потім створюється шаблон HTML *templates/email.html*. На рисунку 8.37 показана відповідна форма для відправлення повідомлень службі підтримки.

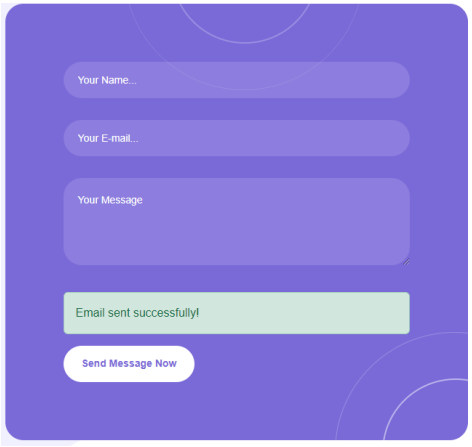

Рисунок 8.37 – Надсилання листа через контактну форму

Тестування кінцевого продукту. Етап тестування важливий, як і всі попередні етапи, добре протестований продукт зменшує кількість помилок, що призводить до кращої взаємодії з користувачем. На цьому етапі можна виконувати багато тестів, ручних або автоматизованих тестів, як тести продуктивності, тести api, але ми будемо проводити лише ручні тести.

Ручне тестування. Ручне тестування – це процес тестування, в якому ручні тести виконуються без використання будь-яких засобів автоматизації. Основна мета – виявлення програмних помилок, проблем і дефектів. Процес ручного тестування включає кілька етапів, таких як розуміння вимог і специфікацій, підготовка тестового середовища, створення тестових випадків і тестових даних, а потім обробка тестових випадків. Ручне тестування включає 3 основні типи: тестування білого ящика, тестування чорного ящика та тестування сірого ящика.

Перед початком, ми повинні налаштувати тестове середовище, таблиця 8.1 є місцем, де ми визначаємо всі необхідні налаштування.

Таблиця 8.1 – Тестове середовище

Браузер	Mozilla Firefox 126.0 (64-bit)
Операційна система	Windows 11 pro (64-bit)
Адмін email	testcclznu@gmail.com
Пароль	test123456@

Опис тестових випадків подано в таблицях 8.2 – 8.9.

Таблиця 8.2 – Успішний вхід через Google

Тестовий випадок 1	успішний вхід через Google
ID тестовий приклад	TC001
Назва	Верифікація успішний вхід в систему за допомогою облікового запису Google
Передумови	користувач повинен мати дійсний авторизований обліковий запис Google
Кроки перевірки	-Перейдіть на сторінку входу https://admin-codecraftinglab.vercel.app/ -Натисніть на кнопку "Увійти за допомогою Google". -Введіть дійсні облікові дані облікового запису Google у спливаючому вікні входу в систему google. - Надайте необхідні дозволи.
Очікуваний результат	користувач успішно авторизується і перенаправляється в панель адміністратора.
Фактичний результат	Користувач успішно увійшов в систему
Статус тесту	Успішний

На рисунку 8.38 показаний перший крок першого тестового випадку.

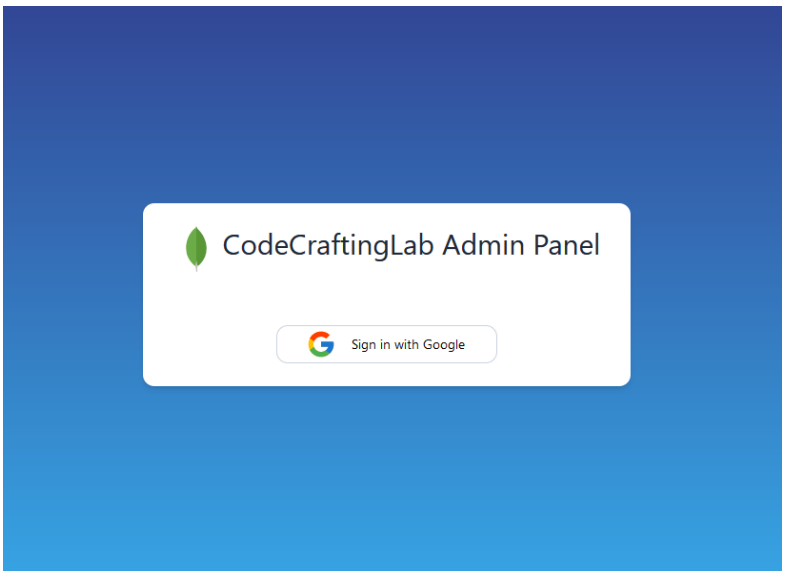

Рисунок 8.38 – Крок 1 у тестовому прикладі 1

Другий крок показаний на рисунку 8.39.

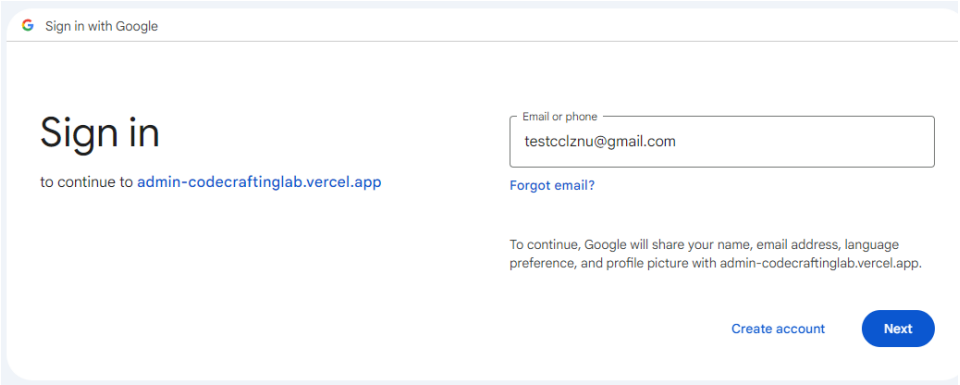

Рисунок 8.39 – Крок 2 у тестовому прикладі 1

Третій крок першого тестового випадку показаний на рисунку 8.40.

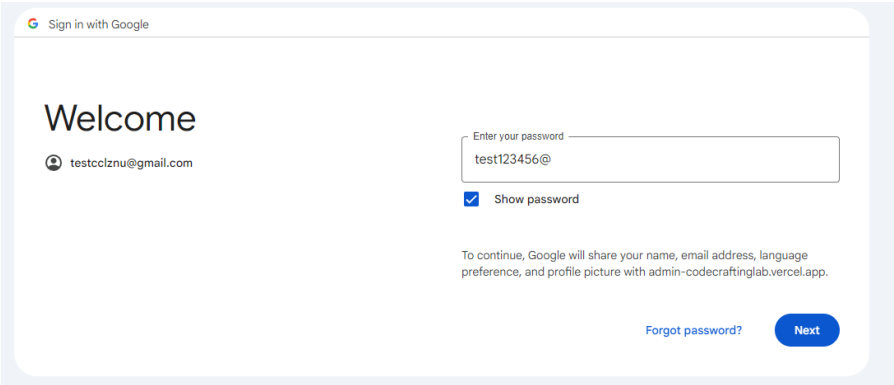

Рисунок 8.40 – Крок 3 у тестовому прикладі 1

Четвертий крок першого тестового випадку показаний на рисунку 8.41.

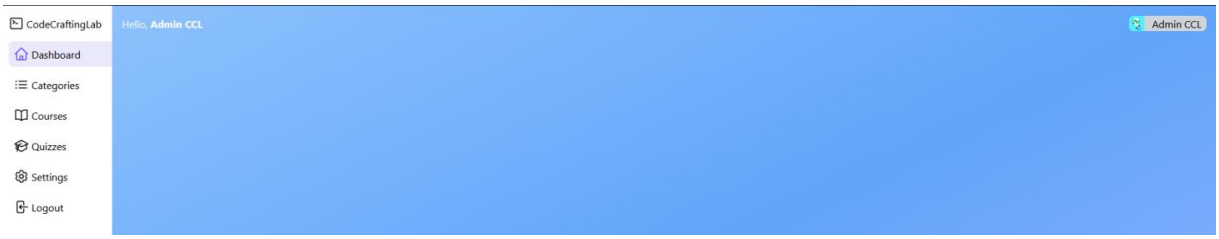

Рисунок 8.41 – Крок 4 у тестовому прикладі 1 Користувач увійшов в систему

Таблиця 8.3 – Верифікація невдалого входу через обліковий запис Google

Тестовий випадок 2	невдалий вхід через Google
ID тестовий приклад	TC002
Назва	Верифікація невдалого входу за допомогою облікового запису Google
Передумови	користувач повинен мати обліковий запис Google
Кроки перевірки	-Перейдіть на сторінку входу https://admin-codecraftinglab.vercel.app/ -Натисніть на кнопку "Увійти за допомогою Google". -Введіть дійсні облікові дані облікового запису Google у спливаючому вікні входу в систему google. - Надайте необхідні дозволи.
Очікуваний результат	відображається повідомлення про помилку, і користувач не може увійти в панель адміністратора.
Фактичний результат	Користувач не може увійти, і відображається повідомлення про помилку
Статус тесту	Успішний

Перший крок другого тестового випадку показаний на рисунку 8.42, він схожий на перший крок першого тестового випадку.

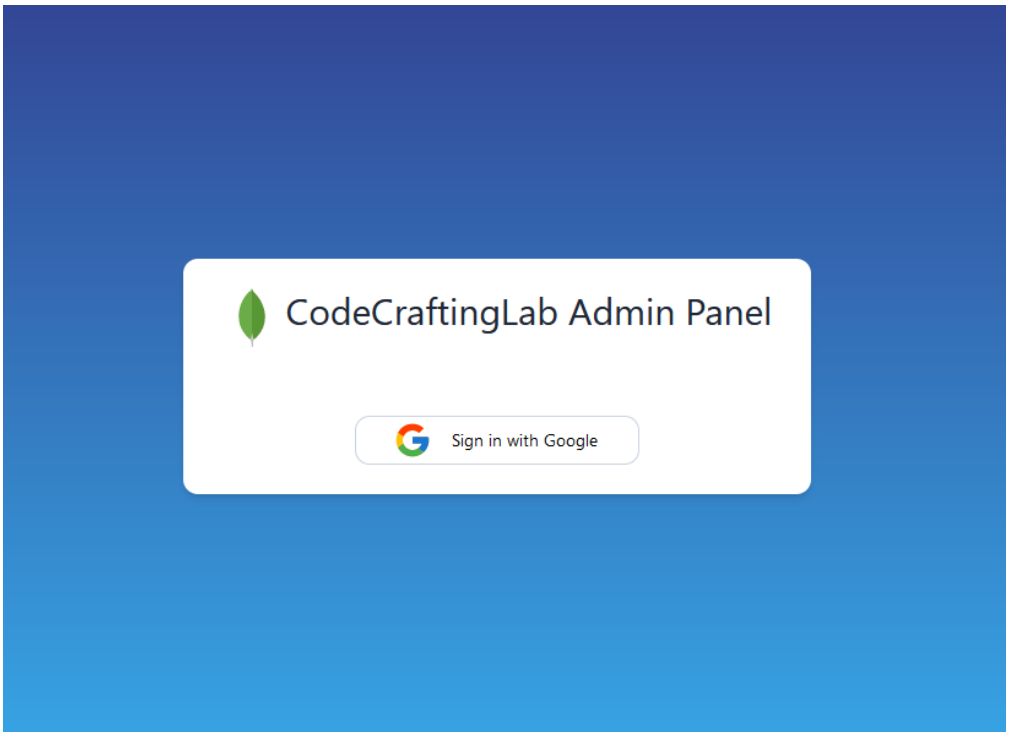

Рисунок 8.42 – Крок 1 у тестовому прикладі 2

Другий крок другого тестового випадку показаний на рисунку 8.43.

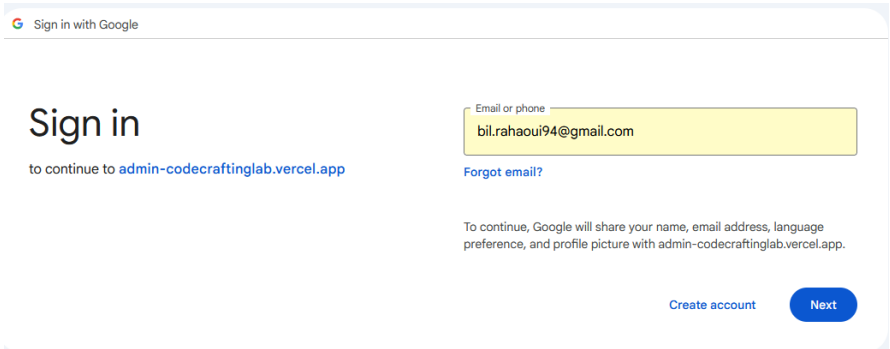

Рисунок 8.43 – Крок 2 у тестовому прикладі 2

Третій крок другого тестового випадку показаний на рисунку 8.44.

Четвертий крок другого тестового випадку показаний на рисунку 8.45.

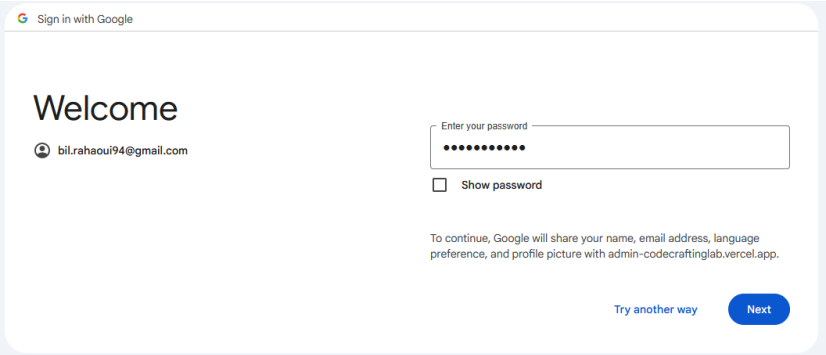

Рисунок 8.44 – Крок 3 у тестовому прикладі 2

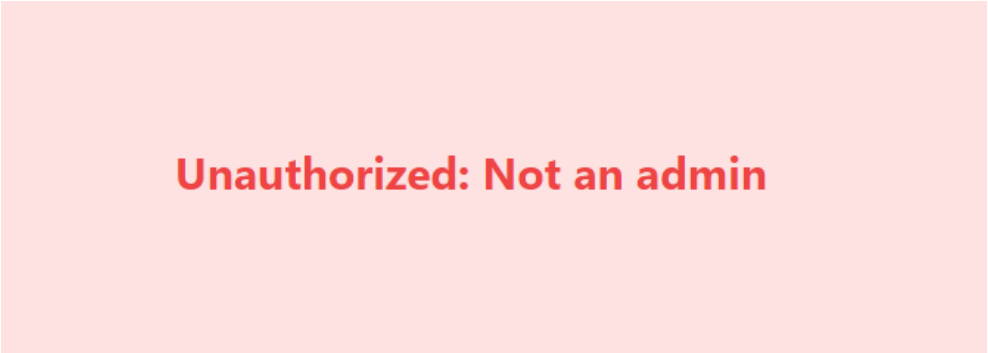

Рисунок 8.45. Крок 4 у тестовому прикладі 2. повідомлення про помилку

Таблиця 8.4 – Адміністратор успішно створює категорію

Тестовий випадок 3	Успішне створення категорії
ID тестовий приклад	TC003
Назва	Адміністратор успішно створює категорію
Передумови	Немає
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/categories -Додайте назву категорії. -Виберіть батьківську категорію "не обов’язково" - Завантажте мініатюру категорії -Натисніть кнопку Зберегти
Очікуваний результат	Нову категорію успішно створено та додано до таблиці нижче
Фактичний результат	Нову категорію успішно створено та додано до таблиці нижчепомилку
Статус тесту	Успішний

Перший крок третього тестового випадку показаний на рисунку 8.46.

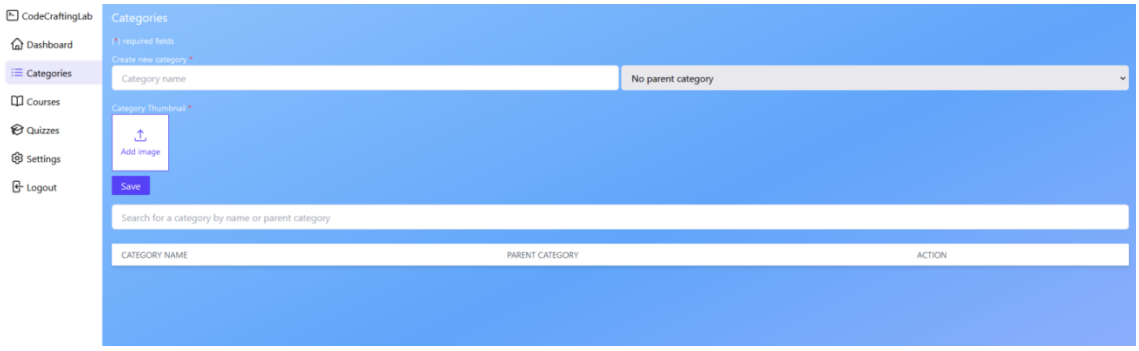

Рисунок 8.46 – Крок 1 у тестовому прикладі 3

На другому кроці третього тестового випадку адміністратор вводить назву категорії, як показано на рисунку 8.47.

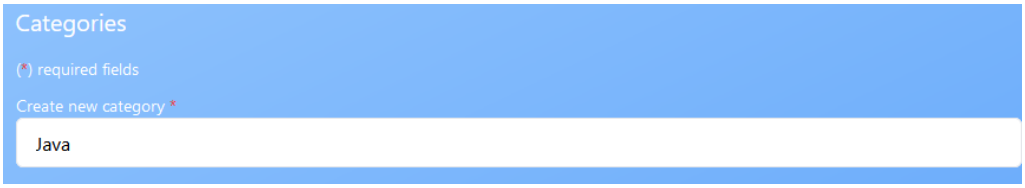

Рисунок 8.47 – Крок 2 у тестовому прикладі 3

Потім він завантажує мініатюру категорії, як показано на рисунку 8.48.

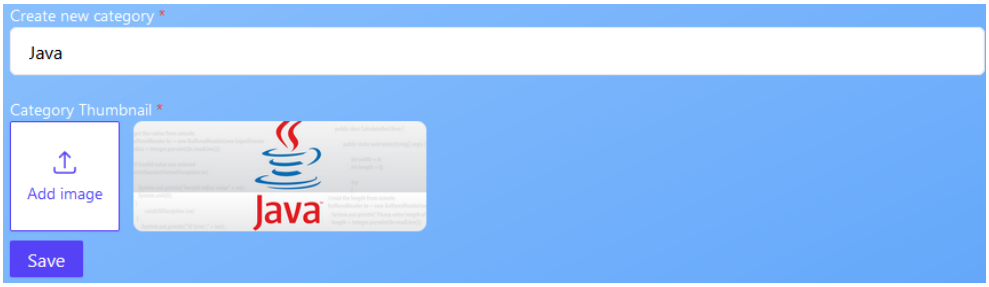

Рисунок 8.48 – Крок 3 у тестовому прикладі 3

Нарешті, як показано на рисунку 8.49, він натискає кнопку "Зберегти", щоб створити нову категорію.

Як показано на рисунку 8.50, нову категорію успішно додано до бази даних системи.

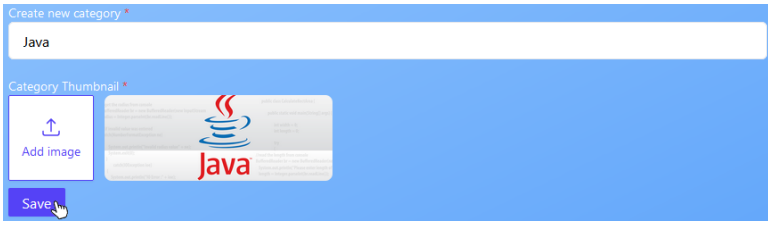

Рисунок 8.49 – Крок 4 у тестовому прикладі 3

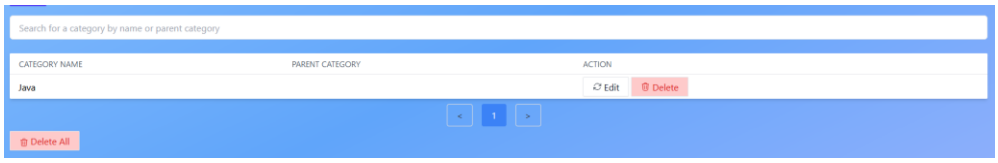

Рисунок 8.50 – Крок 5 у тестовому прикладі 3

Таблиця 8.5 – Адміністратор не заповнює назву категорії

Тестовий випадок 4	Створення категорії без назви
ID тестовий приклад	TC004
Назва	Адміністратор не заповнює назву категорії
Передумови	Немає
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/categories -Виберіть батьківську категорію "не обов’язково" - Завантажте мініатюру категорії -Натисніть кнопку Зберегти
Очікуваний результат	Відображається попередження, а категорія не створюється
Фактичний результат	Відображається попередження “Назва категорії не може бути пустою”.
Статус тесту	Успішний

Як показано на рисунку 8.51, адміністратор залишає поле назви категорії порожнім.

Рисунок 8.51 – Крок 1 у тестовому прикладі 4

На цьому кроці, як показано на рисунку 8.52, адміністратор завантажує мініатюру категорії.

Рисунок 8.52 – Крок 2 у тестовому прикладі 4

Як показано на рисунку 8.53, адміністратор натискає кнопку "Зберегти".

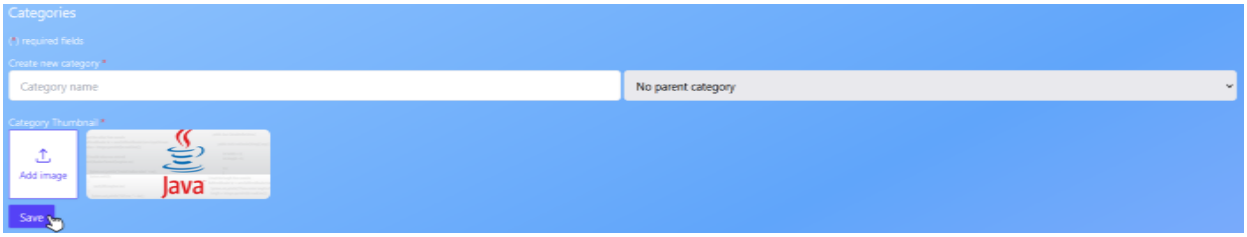

Рисунок 8.53 – Крок 3 у тестовому прикладі 4

Попереджувальне повідомлення відображається, як показано на рисунку 8.54.

Рисунок 8.54 – Крок 4 у тестовому прикладі 4

Таблиця 8.6 – Адміністратор не додає мініатюру категорії

Тестовий випадок 5	Створення категорії без зображення
ID тестовий приклад	TC005
Назва	Адміністратор не додає мініатюру категорії
Передумови	Немає
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/categories -Додайте назву категорії. -Виберіть батьківську категорію "не обов'язково" -Натисніть кнопку Зберегти
Очікуваний результат	Відображається попередження, а категорія не створюється
Фактичний результат	Відображається попередження «Потрібне зображення». а категорія не створена
Статус тесту	Успішний

Як показано на рисунку 8.55, адміністратор вводить назву категорії.

Рисунок 8.55 – Крок 1 у тестовому прикладі 5

На другому кроці, як показано на рисунку 8.56, адміністратор натискає кнопку "Зберегти" без завантаження мініатюри категорії.

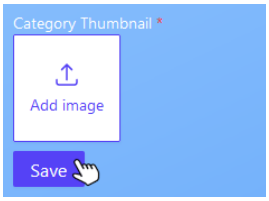

Рисунок 8.56 – Крок 2 у тестовому прикладі 5

Як показано на рисунку 8.57, відображається попереджувальне повідомлення, яке заважає адміністратору створити категорію без мініатюри.

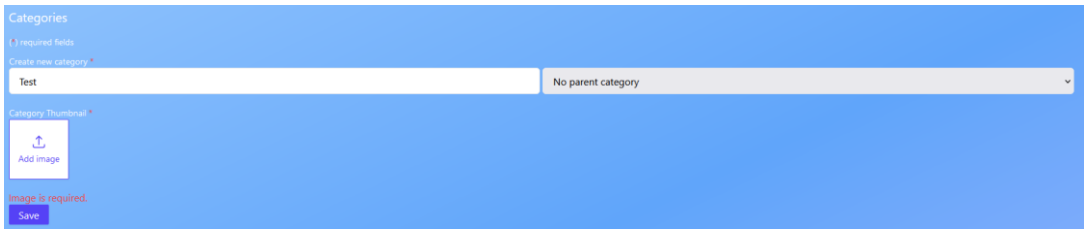

Рисунок 8.57 – Крок 3 у тестовому прикладі 5

Таблиця 8.7 – Адміністратор успішно редагує категорію

Тестовий випадок 6	Успішне редагування категорії
ID тестовий приклад	TC006
Назва	Адміністратор успішно редагує категорію
Передумови	хоча б одна категорія повинна існувати в базі даних
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/categories -Натисніть кнопку Редагувати біля потрібної категорії. -Редагувати назву -Редагувати мініатюру -Натисніть кнопку Зберегти
Очікуваний результат	Вибрану категорію успішно оновлено
Фактичний результат	Вибрану категорію успішно оновлено
Статус тесту	Успішний

Як показано на рисунку 8.58, перший крок редагування існуючої категорії – натиснути кнопку "Редагувати" поруч з потрібною категорією.

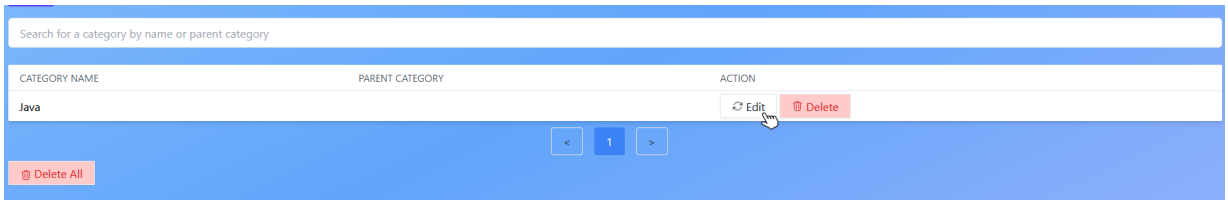

Рисунок 8.58 – Крок 1 у тестовому прикладі 6

Потім адміністратор, наприклад, як показано на рисунку 8.59, якщо він хоче змінити мініатюру категорії, має натиснути кнопку "Додати зображення".

Рисунок 8.59 – Крок 2 у тестовому прикладі 6

Потім з’являється поле вибору, адміністратор вибирає нову мініатюру, стара мініатюра замінюється новою, як показано на рисунку 8.60.

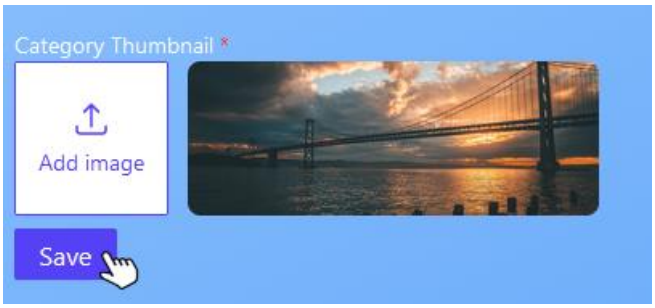

Рисунок 8.60 – Крок 3 у тестовому прикладі 6

Нарешті, після збереження змін адміністратор перенаправляється на головну сторінку категорій, де йому відображаються внесені зміни, як показано на рисунку 8.61.

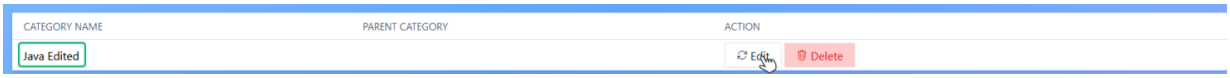

Рисунок 8.61 – Крок 4 у тестовому прикладі 6

Таблиця 8.8 – Редагування категорії з порожньою назвою категорії

Тестовий випадок 7	користувач намагається зберегти категорію з порожньою назвою категорії під час редагування
ID тестовий приклад	TC007
Назва	Редагування категорії з порожньою назвою категорії
Передумови	хоча б одна категорія повинна існувати в базі даних
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/categories -Натисніть кнопку Редагувати біля потрібної категорії. -Очистити назву категорії -Редагувати мініатюру -Натисніть кнопку Зберегти
Очікуваний результат	Відобразиться попереджувальне повідомлення
Фактичний результат	Відображається попередження "Назва категорії не може бути порожньою."
Статус тесту	Успішний

Рисунок 8.62 показує перший крок сьомого тестового випадку.

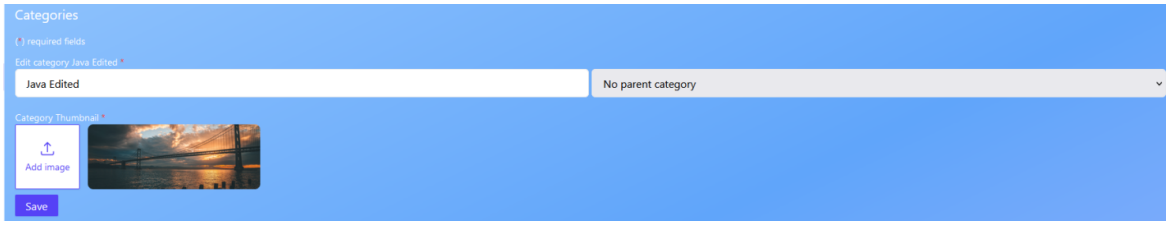

Рисунок 8.62 – Крок 1 у тестовому прикладі 7

На рисунку 8.63 адміністратор очищує назву категорії.

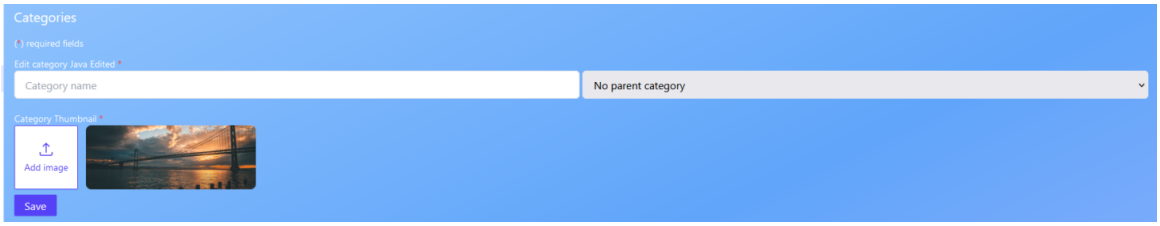

Рисунок 8.63 – Крок 2 у тестовому прикладі 7

На наступному кроці, як показано на рисунку 6.64, адміністратор натискає кнопку "Зберегти".

Рисунок 8.64 – Крок 3 у тестовому прикладі 7

На четвертому кроці сьомого тестового випадку, як показано на рисунку 8.65, відображається попереджувальне повідомлення, яке запобігає збереженню категорії з порожньою назвою, оскільки це обов’язкове поле.

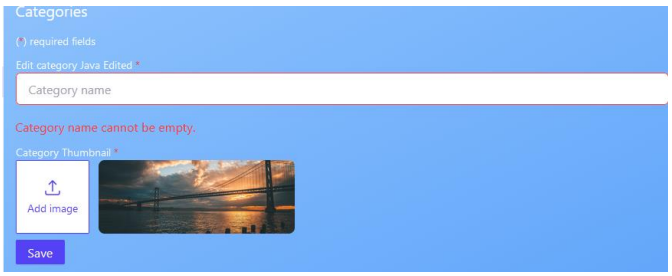

Рисунок 8.65 – Крок 4 у тестовому прикладі 7

Таблиця 8.9 – Адмін успішно створив новий курс

Тестовий випадок 8	Успішне створення нового курсу
ID тестовий приклад	TC008
Назва	Адмін успішно створив новий курс
Передумови	хоча б одна категорія повинна існувати в базі даних
Кроки перевірки	-Перейдіть на сторінку категорій https://admin-codecraftinglab.vercel.app/courses -Натисніть кнопку Додати новий курс. -Введіть назву курсу -Виберіть категорію -Завантажте мініатюру курсу -Додайте опис курсу (не обов’язково) -Додайте YoutubeUrl, який відповідає прийнятним форматам -Натисніть кнопку Зберегти
Очікуваний результат	Створюється новий курс, і користувач перенаправляється на головну сторінку курсів
Фактичний результат	Відображається попередження "Назва категорії не може бути порожньою."
Статус тесту	Успішний

На рисунку 8.66 показана кнопка для створення нового курсу, розташована за адресою /courses.

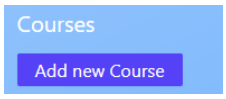

Рисунок 8.66 – Крок 1 у тестовому прикладі 8

Наступні кроки передбачають відображення форми для адміністратора, де він може ввести інформацію про курс. На рисунку 8.67 показане поле для введення назви курсу.

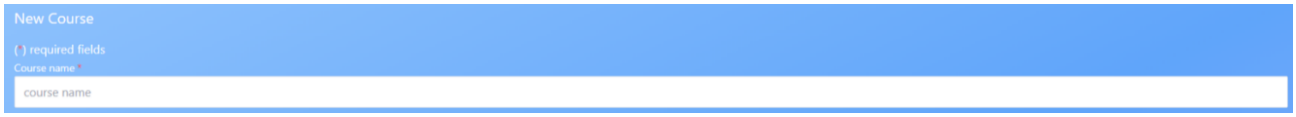

Рисунок 8.67 – Крок 2 у тестовому прикладі 8

Адміністратор заповнює поле "Назва курсу", як показано на рисунку 8.68.

Рисунок 8.68 – Крок 3 у тестовому прикладі 8

Наступне поле є випадаючим списком, де адміністратор повинен вибрати батьківську категорію для призначення новому курсу, як показано на рисунку 8.69.

Рисунок 8.69 – Крок 4 у тестовому прикладі 8

Як показано на рисунку 8.70, перелічені всі доступні категорії в базі даних.

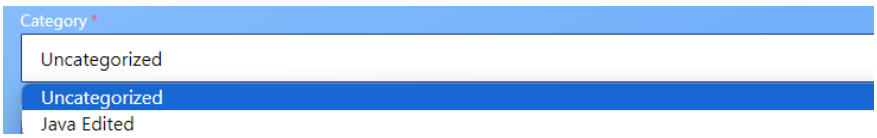

Рисунок 8.70 – Крок 5 у тестовому прикладі 8

На рисунку 8.71 показано, що батьківська категорія успішно вибрана.

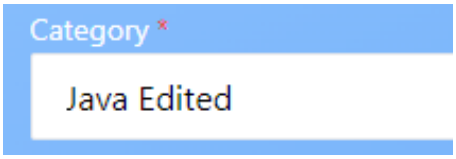

Рисунок 8.71 – Крок 6 у тестовому прикладі 8

Потім адміністратор має можливість завантажити зображення для курсу через поле вибору файлу, як показано на рисунку 8.72.

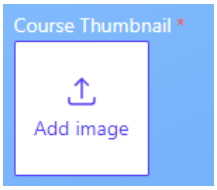

Рисунок 8.72 – Крок 7 у тестовому прикладі 8

Як показано на рисунку 8.73, зображення для курсу успішно завантажено.

Рисунок 8.73 – Крок 8 у тестовому прикладі 8

На рисунку 8.74 показано текстове поле, де адміністратор може додати опис курсу. Це поле не є обов’язковим, але воно допомагає користувачам зрозуміти, про що курс.

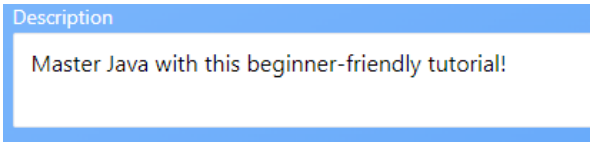

Рисунок 8.74 – Крок 10 у тестовому прикладі 8

Як показано на рисунку 8.75, вміст курсу можна додати з цього текстового поля.

Рисунок 8.75 – Крок 11 у тестовому прикладі 8

Як показано на рисунку 8.76, адміністратор може додати посилання на YouTube з цього текстового поля.

Рисунок 8.76 – Крок 12 у тестовому прикладі 8

Нарешті, для збереження змін адміністратор повинен натиснути кнопку "Зберегти", як показано на рисунку 8.77.

Рисунок 8.77 – Крок 13 у тестовому прикладі 8

Як показано на рисунку 8.78, новий курс успішно створено і додано до бази даних.

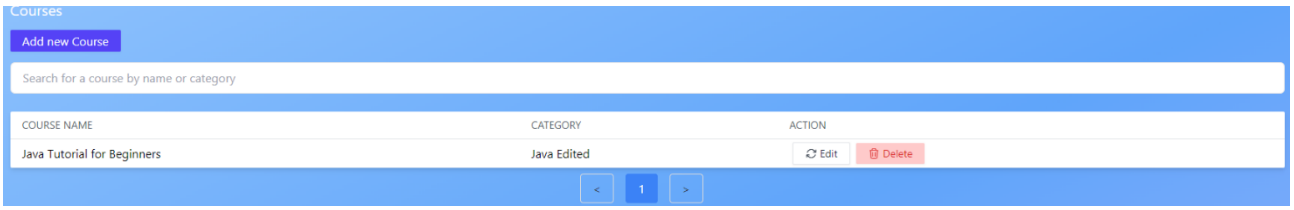

Рисунок 8.78 – Крок 14 у тестовому прикладі 8

У той же час, як показано на рисунку 8.79, новий курс "Java tutorial For beginners" також додано до основного додатку.

Рисунок 8.79 – Крок 15 у тестовому прикладі 8

Для перевірки системи, можна виконати багато тестів і сценаріїв, але ми виконали лише попередні тести через брак часу та велику кількість тестів.

ВИСНОВКИ

Метою даної кваліфікаційної роботи було проєктування та розробка інформаційної системи у форматі вебзастосунку для самостійного вивчення основ програмування, за описом вимог клієнта.

У роботі описано процес створення вебзастосунку CodeCraftingLab. У роботі реалізовано технології Next.js, TailwindCSS і MongoDB. Окрему увагу було приділено використанню бази даних MongoDB.

Процес розробки включав: створення інтерфейсу користувача, реалізацію функціоналу для навчання через текстові курси з відео, квізи для перевірки навичок користувачів, взаємодію з базою даних.

У результаті роботи було створено та протестовано на готовність до використання вебзастосунок CodeCraftingLab. Він надає можливість користувачам (студентам) самостійного вивчення основ програмування за допомогою текстових курсів з відео та квізів для перевірки навичок користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Riccardo, D. Next.js 14: All the Innovations Making Deployment Simple and Efficient. Codemotion. URL : https://www.codemotion.com/magazine/frontend/next-js-14-all-the-innovations/ (дата звернення : 04.04.2024).

2. Next.js Docs. URL : https://nextjs.org/docs (дата звернення : 04.04.2024).

3. Paul, K. Nick, B. Alexander, S.G. Amazon Web Services (AWS). Techtarget. URL: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services (дата звернення : 04.04.2024).

4. Diana, L. Sending Emails with Nodemailer Explained. Mailtrap. URL : https://mailtrap.io/blog/sending-emails-with-nodemailer/ (дата звернення : 11.05.2024).

5. Why Use MongoDB: What It Is and What Are the Benefits. Simplilearn. URL : https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb (дата звернення : 17.02.2023).

6. Sarah, L. What is Agile methodology? (A beginner’s guide). URL : https://asana.com/resources/agile-methodology (дата звернення : 02.02.2024).

7. What is Scrum? URL : https://www.scrum.org/resources/what-scrum-module (дата звернення : 05.02.2024)

8. What Is JIRA? URL : https://www.productplan.com/glossary/jira/ (дата звернення : 05.02.2024)

9. Kosta, M. Story Point to Hours: Which Estimation Approach to Choose? URL : https://intellisoft.io/story-point-to-hours-which-estimation-approach-to-choose/#:~:text=You%20can%20do%20this%20by,roughly%20equivalent%20to%204%20hours. (дата звернення : 03.02.2024).

10. Mit, T. Deep Learn About Different Types of Manual Testing. URL : https://www.kiwiqa.com/different-types-of-manual-testing/ (дата звернення : 07.02.2024).