

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Онлайн платформа для навчання»

Виконав студент групи ІПЗ-20-2
Палій Іван Володимирович

Керівник роботи:
Левченко Антон Юрійович

Консультант:
Єфремов Юрій Миколайович

Рецензент:
Лобанчикова Надія Миколаївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Палій Іван Володимирович**

Керівник роботи: **Левченко Антон Юрійович**

Тема роботи: **«Онлайн платформа для навчання»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **Онлайн платформа для навчання**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	27.02.2024р.	27.02.2024р.
2	Єфремов Ю.М.	27.02.2024р.	27.02.2024р.
3	Єфремов Ю.М.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з онлайн платформи для навчання та пояснювальної записки. Пояснювальна записка до випускної роботи містить 55 сторінок, 31 ілюстрацій та 9 таблиць.

Метою роботи є реалізація зручного додатку для онлайн навчання та отримання додаткової інформації в цілях розвитку з використанням актуальних технологій та інтеграцією з базою даних для зберігання даних.

Сайт побудований на монолітній архітектурі, що забезпечує простоту моніторингу, швидкого росту та налагодження проекту. Для тестування та поліпшення продуктивності ми можемо використовувати стандартні інструменти розробки.

Додаток має зручний, лаконічний інтерфейс, який дозволяє користувачам легко обирати курси, ознайомлюватись з уроками та виконувати завдання.

КЛЮЧОВІ СЛОВА: КУРСИ, МОНОЛІТ, САЙТ, ОНЛАЙН ПЛАТФОРМА ДЛЯ НАВЧАННЯ, ВЧИТЕЛЬ, УЧЕНЬ

					ІПЗ.КР.Б-121-24-ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Онлайн платформа для навчання	Літ.	Арк.	Аркушів
Розроб.	Палій І.В.						4	71
Керівник	Левченко А.Ю							
Рецензент	Лобанчикова Н.М.					Житомирська політехніка, група ІПЗ-20-2		
Зав. каф.	Вакалюк Т.А.							

ABSTRACT

The bachelor's thesis consists of an online learning platform and an explanatory note. The explanatory note to the final thesis contains 55 pages, 31 illustrations and 9 tables.

The aim of the work is to implement a convenient application for online learning and obtaining additional information for development purposes using up-to-date technologies and integration with a database for data storage.

The site is built on a monolithic architecture that provides ease of monitoring, rapid growth and project debugging. We can use standard development tools to test and improve performance.

The application has a user-friendly, concise interface that allows users to easily select courses, familiarize themselves with lessons, and complete assignments.

KEYWORDS: COURSES, MONOLITH, WEBSITE, ONLINE LEARNING PLATFORM, TEACHER, STUDENT

					ІПЗ.КР.Б-121-24-ІІЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ НАВЧАННЯ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	14
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЕКТУВАННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ НАВЧАННЯ....	20
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	20
2.2 Розробка бази даних системи	22
2.3 Проектування та реалізація алгоритмів роботи системи.....	27
2.4 Реалізація функціоналу онлайн платформи для навчання	28
Висновки до другого розділу:	38
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН ПЛАТФОРМОЮ ДЛЯ НАВЧАННЯ	39
3.1 Проектування інтерфейсу обробки даних.....	39
3.2 Структура інтерфейсу онлайн платформи для навчання	40
3.3 Тестування платформи.....	48
Висновки до третього розділу	51
ВИСНОВКИ.....	52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	53
ДОДАТКИ.....	56

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – прикладний програмний інтерфейс

БД – База даних

СУБД – система управління базами даних

IDE – Інтегроване середовище розробки

CRUD – основні операції, що використовуються для управління даними
(Create , Read , Update , Delete)

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. В сучасному світі майже неможливо уявити життя без веб-технологій. Кожна компанія, магазин, університет мають власні сайти, адже це дуже важлива складова, навіть для маленьких компаній. Сайт спрощує доступ до інформації або послуги певної компанії чи магазину.

Онлайн платформи для навчання стали ще більш актуальними під час війни в Україні. Війна створила численні труднощі для студентів, викладачів і шкіл, змушуючи їх шукати альтернативні способи продовження освітнього процесу. Онлайн платформи надають можливість студентам отримувати освіту незалежно від місця їх перебування, що є критично важливим в умовах військових дій. Вони також дозволяють викладачам продовжувати свою професійну діяльність, забезпечуючи стабільність освітнього процесу навіть в найважчі часи. Завдяки онлайн платформам, студенти можуть мати доступ до якісної освіти, що сприяє їхньому особистому і професійному розвитку, навіть під час війни.

Більше того, пандемія COVID-19 показала критичну важливість онлайн навчання, коли традиційні методи освіти стали недоступними через карантинні заходи. Онлайн платформи для навчання стали основним засобом забезпечення безперервності освітнього процесу. В умовах війни, ці платформи продовжують відігравати ключову роль, надаючи можливість навчання навіть під час евакуації або в умовах тимчасового переміщення.

Таким чином, розробка і впровадження онлайн платформи для навчання є надзвичайно актуальною і важливою темою, яка відповідає сучасним викликам і потребам суспільства, особливо в умовах кризи.

Метою випускної кваліфікаційної роботи бакалавра є розробка онлайн платформи для навчання з використанням сучасних технологій веб-розробки. Цей сайт буде включати інтерактивні функції, такі як список курсів, записуватись на курс, виконувати завдання курсів, переглядати відео курси, а також можливість взаємодії з користувачами через коментарі та залишати рейтинг курсам.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

Для розробки онлайн платформи для навчання нам необхідно:

1. **Провести детальний аналіз ринку:** дослідити функціональні можливості та недоліки існуючих сайтів для онлайн навчання. Це допоможе виявити, які функції потребують вдосконалення та які нові можливості можуть бути корисними для користувачів.
2. **Вибрати оптимальні технології:** визначити оптимальні інструменти для розробки фронтенду і бекенду та вибрати відповідну базу даних. Це забезпечить надійну, стабільну і продуктивну роботу платформи.
3. **Розробити сайт на основі зібраних даних:** створити зручний користувацький інтерфейс, впровадити ключові функції, такі як список курів, створення та редагування курсів вчителем, проходження курсів та уроків. Особлива увага буде приділена забезпеченню безпеки даних користувачів.
4. **Оптимізувати і тестувати платформу:** використати сучасні методи оптимізації для підвищення продуктивності та масштабованості сайту. Провести всебічне тестування всіх функцій для забезпечення стабільної та надійної роботи системи.

Об'єктом дослідження є процес створення онлайн платформи для навчання з нуля, використовуючи популярні веб-технології. Дослідження охоплює всі етапи розробки: від планування та проектування архітектури до написання коду та тестування. Процес включає розробку адаптивного інтерфейсу користувача, реалізацію реєстрації, запису на курси, перегляду курсів та відгуків, а також забезпечення безпеки даних та швидкодії системи.

Предметом дослідження є можливості і актуальність стеку технологій, таких як MongoDB, Prisma ORM, Fastify, React, Node.js. Досліджується складність розробки та використання як окремих функцій, так і їх сукупність в рамках описаного стеку на основі онлайн платформи для навчання.

					ІПЗ.КР.Б-121-24-ПЗ	Арк. 8
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ НАВЧАННЯ

1.1 Постановка задачі

Метою випускної кваліфікаційної роботи бакалавра є розробка онлайн платформи для навчання з використанням сучасних технологій веб-розробки.

В сучасному світі онлайн навчання стає все більш актуальним, особливо в умовах глобалізації та цифровізації. Багато людей вважають за краще отримувати знання та навички в зручному для себе темпі і місці. Онлайн платформи для навчання дозволяють не тільки отримувати нові знання, але й підвищувати кваліфікацію без відриву від основної діяльності.

На даний момент існує багато платформ для онлайн навчання, кожна з яких має свої особливості та переваги. Найефективніші платформи, такі як Udemy, Coursera та Khan Academy, пропонують широкий спектр курсів з різних дисциплін, інтерактивні завдання, відеоуроки та можливість взаємодії з викладачами і іншими студентами.

Наш сайт спеціалізується на створенні інтерактивних курсів, які включають список курсів, можливість записуватися на курс, виконувати завдання, переглядати відеоуроки, а також взаємодіяти з іншими користувачами через коментарі та залишати рейтинг курсам. Для досягнення цієї мети ми будемо використовувати сучасні технології веб-розробки.

Для розробки онлайн платформи для навчання необхідно:

- Управління курсами: створення і редагування курсів для вчителів, відображення тестів, відео, та текстових уроків, реєстрацію на курс при покупці.
- Автоматизоване ведення бази даних користувачів: включає реєстрацію нових студентів, оновлення інформації про існуючих користувачів, а також управління курсами на які вони записані. Це дозволить підтримувати актуальну інформацію про кожного користувача, його зацікавленості та взаємодії з платформою.

- Фінансовий облік: управління фінансовими транзакціями, включаючи облік платежів за курс.
- Аналітика та звітність: збір та аналіз даних про користувачів, активність на платформі, ефективність курсів та фінансові показники. Це включає відстеження кількості учасників курсів, прогресу навчання, оцінок та коментарів, а також доходів від продажу курсів. Така аналітика дозволяє вчителям та адміністраторам приймати обґрунтовані рішення щодо покращення якості курсів, підвищення залученості користувачів та оптимізації фінансових стратегій.

В результаті маємо створити сайт на якому будь-який користувач зможе створити акаунт для проходження курсів, отримати відповідну кількість уроків, курсів, і проходження інтерактивних завдань.

Розіб'ємо розробку на етапи:

1. Розробка дизайну сайту
 - Створення базового стилю для всього додатку
 - Створення різних UI компонентів
 - Персоналізований дизайн для сторінок
2. Побудова архітектури БД
3. Планування серверної частини
4. Розробка основного функціоналу
 - Авторизація користувача
 - Панель вчителя
 - Створення та редагування курсів
 - Створення та редагування інформації учнів
 - Створення та реалізація уроків курсів
5. Поєднання бекенду і фронтенду
6. Тестування додатку

В результаті дотримання плану маємо онлайн платформу для навчання, яка дозволяє користувачам записуватися на курси, переглядати відеоуроки, виконувати завдання, залишати коментарі та оцінки. Вчителі мають можливість

					ІПЗ.КР.Б-121-24-ПЗ	Арк. 10
Зм.	Арк.	№ докум.	Підпис	Дата		

створювати та редагувати курси, управляти студентами та аналізувати статистику навчання і фінансові показники.

1.2 Аналіз аналогів програмного продукту.

При розробці онлайн платформи для навчання необхідно провести детальний аналіз конкурентів. Розглянемо популярні онлайн платформи для навчання з різних країн, щоб обрати найкращі функції та визначити ті, яких немає на більшості сайтів.

Одним з таких додатків є Coursera (Рис. 1.1).

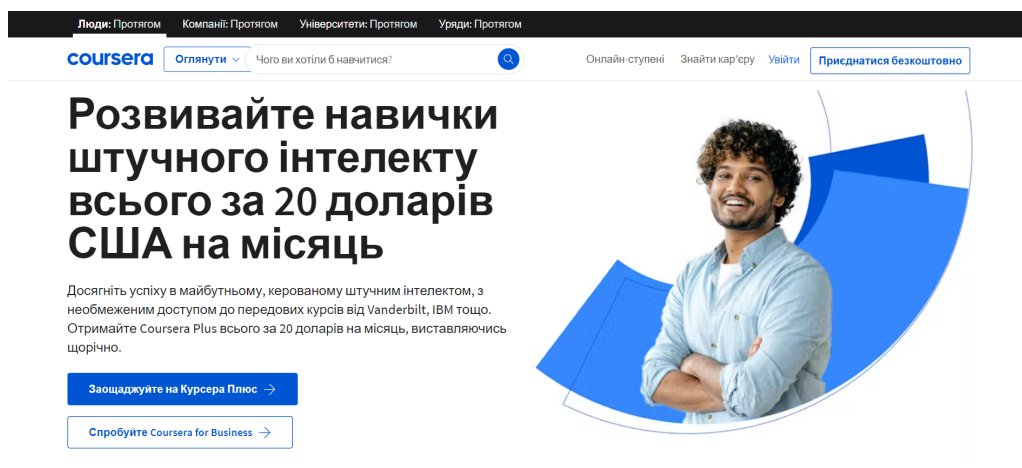


Рис. 1.1. Сайт онлайн платформи для навчання Coursera

Coursera є однією з найпопулярніших платформ для онлайн навчання. Вона надає користувачам доступ до великої кількості курсів від провідних університетів та компаній.

Основними перевагами Coursera є:

- Сучасний приємний дизайн
- Великий вибір курсів та спеціалізацій
- Детальна інформація про викладачів
- Наявність сертифікатів по закінченню курсів

З недоліків можна визначити:

- Висока вартість деяких курсів
- Обмежений безкоштовний доступ до матеріалів

Наступним популярним додатком є Udey (Рис. 1.2).

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

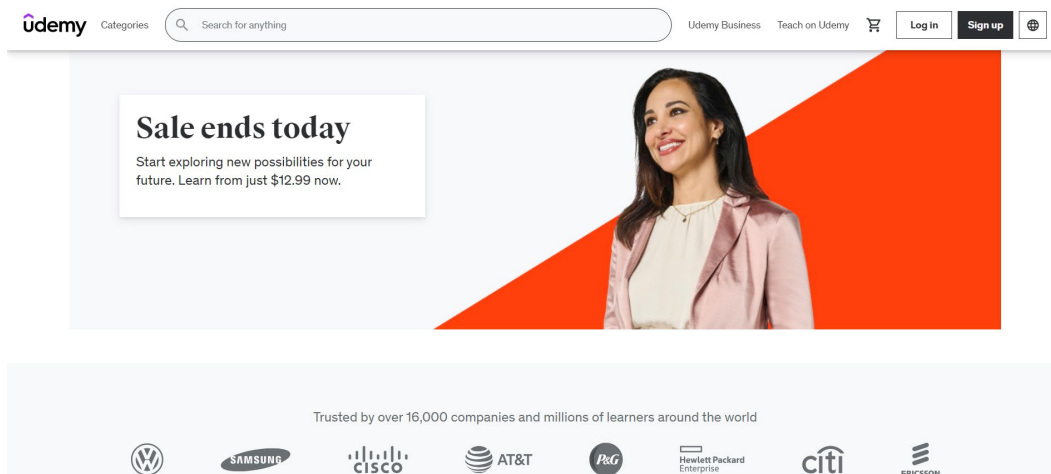


Рис. 1.2. Сайт онлайн платформи для навчання Udemy

Перевагами цього сайту є:

- Великий вибір курсів за доступними цінами
- Можливість створення власних курсів для викладачів
- Постійні акційні пропозиції

Проте сайт має низку недоліків:

- Варіативна якість курсів
- Відсутність підтримки від викладачів на деяких курсах

Duolingo – платформа для вивчення мов (Рис. 1.3).

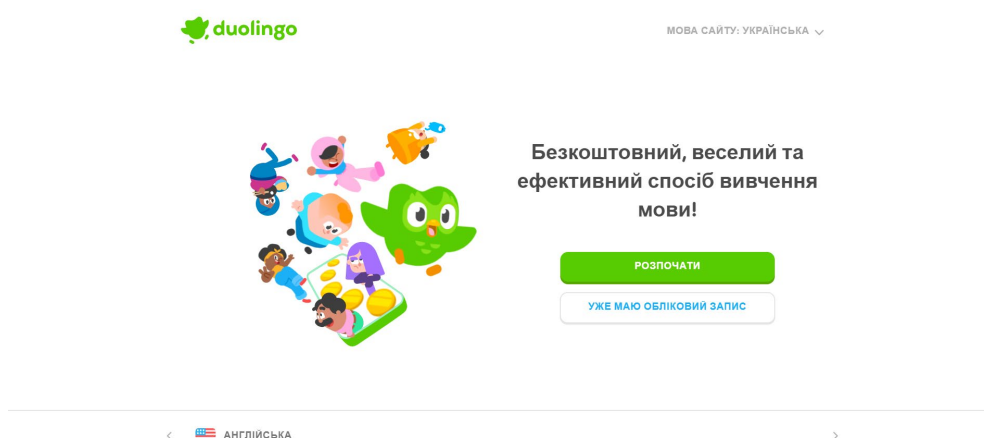


Рис. 1.3. Сайт онлайн платформи для навчання Doulingo

Сайт має декілька переваг серед конкурентів:

- Ігровий підхід до навчання

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

- Безкоштовний доступ до більшості функцій
- Можливість вивчення багатьох мов

З недоліків можна зазначити:

- Обмежена глибина знань для просунутих рівнів
- Мінімалістичний дизайн

edX – платформа для онлайн навчання від провідних університетів світу (Рис. 1.4).

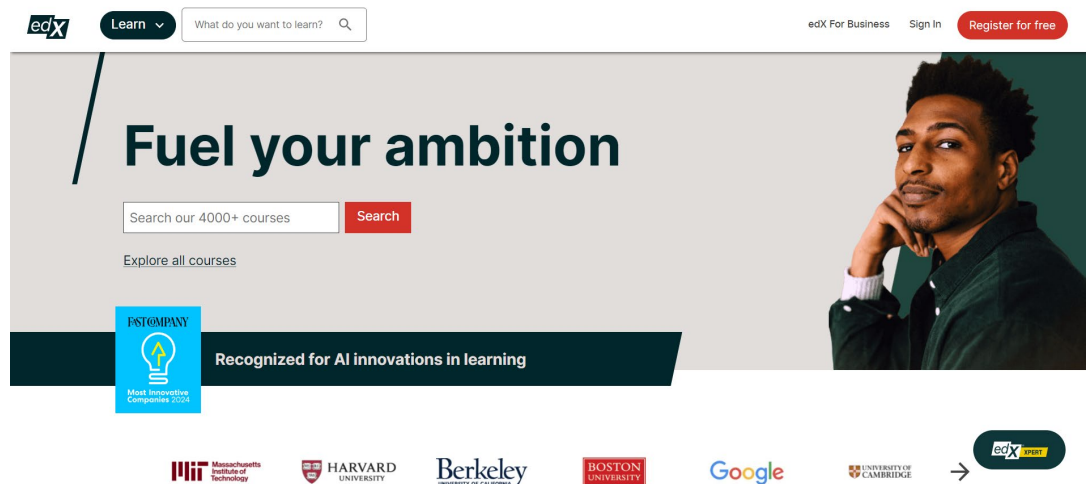


Рис. 1.4. Сайт платформи для навчання edX

Серед переваг:

- Курси від провідних університетів
- Наявність професійних сертифікатів
- Інтерактивні елементи навчання

Щодо недоліків:

- Висока вартість сертифікатів
- Деякі курси доступні лише англійською мовою

На основі проведених нами досліджень можемо скласти порівняльну таблицю платформ за деякими пунктами (табл. 1.1).

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

Таблиця порівняння сайтів тренажерних залів

Характеристика	Coursera	Udemy	Duolingo	edX
Сучасність дизайну	+	+	+-	+
Наявність інформації про викладачів	+	+-	-	+
Акції	+	+	-	+-
Зручність інтерфейсу	+	+-	+	+
Додаткова інформація для користувача	+	+-	+-	+
Додаткові особливості сайту	+	+	+	+

Переглянувши платформи конкурентів, можна зазначити основні відмінності: деякі платформи надають вичерпну інформацію про курси та викладачів, інші – роблять акцент на зручності користувача та додаткових функціях.

Враховуючи інформацію, отриману в ході дослідження, було вирішено створити онлайн платформу для навчання, яка буде відповідати сукупності функцій наведених платформ та введення власних функцій. Серед основних: авторизація, присвоєння ролей, списки курсів, завдань, викладачів та детальна інформація про них. Платформа буде сучасною, функціональною та зручною для користувачів.

1.3 Визначення варіантів використання та об'єктно-орієнтованої структури системи.

Додаток буде побудований на монолітній архітектурі. Серверна частина буде розроблена з використанням сучасного фреймворку для JavaScript – Node.js. Для фронтенду буде використано фреймворк React.js – відкриту JavaScript бібліотеку для створення інтерфейсів користувача,

яка вирішує проблеми часткового оновлення вмісту вебсторінки в односторінкових застосунках. В якості бази даних обрано MongoDB.

Node.js — це платформа для виконання JavaScript-коду на сервері, заснована на рушію V8 від Chrome. Вона стала популярною завдяки своїм унікальним можливостям та архітектурі, що дозволяють створювати масштабовані та ефективні мережеві додатки.

Переваги Node.js:

- **Висока продуктивність:** Node.js використовує асинхронну, подієво-орієнтовану архітектуру, що дозволяє обробляти багато одночасних запитів без блокування вводу-виводу. Це робить його ідеальним для створення реальних додатків, таких як чати та ігри, які потребують швидкого оновлення даних.
- **Масштабованість:** Завдяки легкій та модульній архітектурі Node.js легко масштабувати додатки як горизонтально, так і вертикально. Це дозволяє додавати більше мікросервісів у вже існуючі додатки, забезпечуючи їх гнучкість та стійкість.
- **Використання єдиної мови програмування:** Node.js дозволяє розробникам використовувати JavaScript як на стороні клієнта, так і на стороні сервера. Це спрощує процес розробки, оскільки розробники можуть використовувати одну мову для всього проекту, що також сприяє швидшому навчанню та адаптації нових членів команди.
- **Підтримка численних інструментів:** Node.js має розширену підтримку різних інструментів, таких як npm (Node Package Manager), що дозволяє легко тестувати та управляти залежностями проекту.

Node.js є потужним інструментом для розробки сучасних сайтів завдяки його високій продуктивності, масштабованості, можливості використовувати єдину мову програмування на клієнті та сервері, а також активній спільноті та підтримці численних інструментів.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

React.js, або просто React, є однією з найпопулярніших бібліотек для створення користувацьких інтерфейсів на JavaScript. Розроблена та підтримувана Facebook, вона вперше була представлена у 2013 році.

Переваги React.js:

- **Компонентний підхід:** React використовує компонентний підхід до розробки, що означає, що користувацький інтерфейс складається з невеликих, повторно використовуваних компонентів. Кожен компонент управляє своїм власним станом та логікою, що сприяє легкості управління кодом та підвищує його повторне використання. Це полегшує розробку та тестування великих додатків, роблячи їх більш організованими та зручними для підтримки.
- **Віртуальний DOM:** React використовує віртуальний DOM, який є копією реального DOM. Це дозволяє React ефективно оновлювати користувацький інтерфейс, порівнюючи зміни у віртуальному DOM з реальним DOM та мінімізуючи кількість прямих маніпуляцій з ним. Такий підхід значно покращує продуктивність додатків, особливо тих, які мають складні та динамічні інтерфейси.
- **Однонаправлений потік даних:** React підтримує однонаправлений потік даних, що спрощує розуміння та відстеження стану додатка. Це допомагає уникнути непередбачуваних змін стану та полегшує налагодження коду. Компоненти отримують дані через пропси та можуть викликати зміни стану за допомогою функцій, переданих їм від батьківських компонентів.
- **Потужна екосистема та інструменти:** React має велику екосистему бібліотек та інструментів, таких як React Router для маршрутизації, Redux для управління станом та багато інших. Крім того, React DevTools дозволяє розробникам зручно налагоджувати та аналізувати React додатки.

React.js забезпечує високу продуктивність, зручність у розробці та підтримці додатків, а також має потужну екосистему та активну

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

спільноту. Це робить його чудовим вибором для створення сучасних сайтів.

Fastify — це веб-фреймворк для Node.js, який забезпечує високу продуктивність, низький накладний витрати та гнучкість. Він розроблений для створення швидких та надійних веб-додатків та API.

Переваги Fastify:

- **Висока продуктивність:** Fastify має високу продуктивність завдяки використанню асинхронної, подієво-орієнтованої архітектури Node.js. Це дозволяє обробляти великий обсяг одночасних запитів без значних витрат на ресурси.
- **Гнучкість та розширюваність:** Fastify забезпечує гнучкість у налаштуванні та розширенні функціональності за допомогою плагінів. Це дозволяє легко додавати нові функції та інтеграції до додатка.
- **Зручність у використанні:** Fastify має простий та зрозумілий API, що полегшує розробку та підтримку додатків. Він також має підтримку для TypeScript, що забезпечує додаткову безпеку коду.

Prisma ORM — це сучасний ORM (Object-Relational Mapping) інструмент для Node.js, який забезпечує зручну роботу з базами даних. Він підтримує різні бази даних, включаючи MongoDB, і надає потужні інструменти для моделювання та роботи з даними.

Переваги Prisma ORM:

- **Простота у використанні:** Prisma забезпечує простий та зрозумілий синтаксис для роботи з базами даних. Це дозволяє розробникам швидко створювати та підтримувати моделі даних.
- **Автоматичне генерування запитів:** Prisma автоматично генерує SQL-запити на основі моделей даних, що спрощує роботу з базами даних та знижує ймовірність помилок.
- **Міграції баз даних:** Prisma надає інструменти для управління міграціями баз даних, що дозволяє легко вносити зміни до структури бази даних та підтримувати її в актуальному стані.

MongoDB — популярна база даних NoSQL, яка надає розробникам можливість зберігати та обробляти великі обсяги даних у гнучкий та ефективний спосіб.

Переваги MongoDB:

- **Гнучкість:** MongoDB використовує документо-орієнтовану модель даних, де дані зберігаються у форматі BSON (бінарна версія JSON). Це дозволяє зберігати складні, вкладені структури даних, що робить MongoDB дуже гнучкою у порівнянні з реляційними базами даних.
- **Висока продуктивність:** Завдяки своїй архітектурі MongoDB забезпечує високу швидкість запису та читання даних. Вона оптимізована для роботи з великими наборами даних і може ефективно обробляти запити завдяки індексам, які підтримують різні типи пошуку, включаючи текстовий та геопросторовий.
- **Масштабованість:** MongoDB легко масштабується як горизонтально, так і вертикально. Це дозволяє обробляти великі обсяги даних та підтримувати високий рівень продуктивності навіть при збільшенні навантаження.
- **Інструменти та підтримка:** MongoDB має велику кількість сторонніх інструментів та бібліотек, які дозволяють інтегрувати її з різними мовами програмування та фреймворками. Крім того, MongoDB Inc. надає потужну комерційну підтримку та хмарне рішення MongoDB Atlas, що спрощує керування базою даних.

MongoDB забезпечує гнучкість, продуктивність та масштабованість, що робить її чудовим вибором для сучасних сайтів та великих даних. Завдяки своїм численним перевагам вона стала однією з найпопулярніших баз даних серед розробників по всьому світу.

Висновки до першого розділу

На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки онлайн платформи для

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

навчання. Для створення фронтенду було обрано React, що забезпечує високий рівень інтерактивності та зручність для користувачів. Для бекенду було вирішено використовувати Node.js та Fastify, які дозволяють створити масштабовану та продуктивну серверну частину. MongoDB була обрана як основна база даних завдяки її гнучкості та можливості роботи з великими обсягами даних. Prisma ORM забезпечить зручне управління базою даних та моделювання даних. Цей набір технологій дозволить створити надійну, функціональну та сучасну платформу для навчання, яка буде відповідати всім встановленим вимогам та очікуванням користувачів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ НАВЧАННЯ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.

Перш ніж розпочати розробку додатку, важливо чітко визначити його майбутній функціонал, враховуючи потреби користувачів та цілі бізнесу. Для цього потрібно ретельно проробити прецеденти використання, або сценарії, що описують взаємодію користувачів з додатком, та визначити вимоги до функціоналу на основі цих сценаріїв.

Прецеденти використання допомагають ідентифікувати потреби користувачів та основні функції, які повинен виконувати додаток. Після створення прецедентів можна переходити до розробки бази даних та її налаштування, оскільки це дасть можливість краще зрозуміти, які дані необхідно зберігати для підтримки функціоналу додатку.

Важливо, щоб структура бази даних відповідала потребам функціоналу та забезпечувала ефективне зберігання та доступ до інформації. Такий підхід дозволяє зменшити ймовірність неузгодженостей між функціоналом додатку та його базою даних, що в свою чергу сприяє зручності розвитку та підтримки системи в майбутньому.

Вимоги користувачів:

Вчитель:

1. Авторизація
2. Перегляд статистики курсів
3. Перегляд статистики
4. Слідкування за користувачами
5. Створення/видалення/редагування курсів
6. Створення/видалення/редагування уроків
7. Створення/видалення/редагування завдань

Користувач:

1. Авторизація

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

2. Перегляд курсів
3. Перегляд уроків
4. Перегляд завдань
5. Перегляд/придбання курсів
6. Оцінювання курсів

Функціональні вимоги:

- Додаток має забезпечувати авторизацію зі збереженням токenu, хешування паролю
- Редагування профілю користувача.
- Створення та редагування курсів.
- Створення та редагування уроків.
- Створення та редагування завдань.
- Зарахування на курси.
- Відображення курсів.
- Додавання та редагування профілів користувачів.

Нефункціональні Вимоги:

1. Безпека:
 - Шифрування даних
 - Захист від атак
2. Надійність:
 - Стійкість до витоку пам'яті.
 - Резервне копіювання
3. Продуктивність:
 - Час завантаження сторінок
 - Пропускна спроможність

Відгук від системи до користувача

Системні вимоги:

1. Вимоги до середовища виконання:
 - Оперативна пам'ять: мінімум 2 ГБ.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

- Процесор: тактова частота не менше 1.8 ГГц.
- Підключення до мережі Інтернет.

2.2 Розробка бази даних системи

Для зберігання даних нами було вирішено обрати Prisma ORM з базою даних MongoDB. Prisma ORM – сучасний інструмент для роботи з базами даних, який надає можливість використовувати MongoDB як документо-орієнтовану систему керування базами даних. Prisma ORM забезпечує опис схем даних, підтримує зберігання документів у форматі, подібному до JSON, та має потужну мову для формування запитів. Він також дозволяє створювати індекси для різних атрибутів, що забезпечує високу ефективність обробки великих обсягів даних та спрощує взаємодію з базою даних.

База даних нашої онлайн платформи для навчання складається з кількох таблиць:

Опис кожної з таблиць бази даних наведений у таблицях 2.1 – 2.8.

Таблиця 2.1

Опис таблиці User

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
email	String	Електронна пошта
password	String	Хешований пароль
firstName	String	Ім'я користувача
lastName	String	Прізвище користувача
role	String	Роль користувача (USER)
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Таблиця 2.2

Опис таблиці Teacher

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор

email	String	Електронна пошта
password	String	Хешований пароль
firstName	String	Ім'я вчителя
lastName	Number	Прізвище користувача
role	Number	Роль користувача (TEACHER)
description	String	Опис вчителя
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Таблиця 2.3

Опис таблиці Course

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
title	String	Назва курсу
description	String	Опис курсу
category	String	Категорія курсу
price	Float	Ціна курсу
teacherId	ObjectId	Ідентифікатор вчителя
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Таблиця 2.4

Опис таблиці Enrollment

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
userId	ObjectId	Ідентифікатор користувача
courseId	ObjectId	Ідентифікатор курсу
completed	Boolean	Статус завершення курсу
createdAt	Date	Дата створення

Таблиця 2.5

Опис таблиці Rating

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
enrollmentId	ObjectId	Ідентифікатор запису на курс
courseId	ObjectId	Ідентифікатор курсу
rating	Int	Рейтинг курсу

Таблиця 2.6

Опис таблиці Comment

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
userId	ObjectId	Ідентифікатор користувача
courseId	ObjectId	Ідентифікатор курсу
content	String	Зміст коментаря
createdAt	Date	Дата створення

Таблиця 2.7

Опис таблиці Payment

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
userId	ObjectId	Ідентифікатор користувача
courseId	ObjectId	Ідентифікатор курсу
amount	Int	Сума
currency	String	Валюта
status	String	Статус оплати
createdAt	Date	Дата створення

Таблиця 2.8

Опис таблиці Lesson

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор

title	String	Назва уроку
courseId	ObjectId	Ідентифікатор курсу
contentType	String	Тип вмісту уроку (TEXT, TEST, VIDEO)
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Базова діаграма бази даних складається з:

- Users – ця таблиця містить інформацію про користувачів системи, таку як електронна адреса, пароль, ім'я, прізвище, роль користувача (USER), дату створення та оновлення запису.
- Teachers – таблиця, що містить дані про вчителів, включаючи електронну адресу, пароль, ім'я, прізвище, роль (TEACHER), опис, а також дату створення та оновлення запису.
- Courses – таблиця, що представляє курси. Вона містить інформацію про назву курсу, опис, категорію, ціну, ідентифікатор вчителя, назву та прізвище вчителя, зображення для попереднього перегляду, а також дату створення та оновлення запису.
- Lessons – ця таблиця представляє уроки, які входять до складу курсів. Вона містить дані про назву уроку, тип вмісту (TEXT, TEST, VIDEO), ідентифікатор курсу, до якого належить урок, а також дату створення та оновлення запису.
- Enrollments – таблиця, що відображає інформацію про запис на курси. Вона містить ідентифікатори користувача та курсу, статус завершення курсу, а також дату створення запису.
- Ratings – ця таблиця містить дані про оцінки, надані користувачами курсам. Вона включає ідентифікатор запису на курс, ідентифікатор курсу та значення оцінки.

- Comments – таблиця, що відповідає за коментарі, залишені користувачами до курсів. Вона містить ідентифікатори користувача та курсу, а також текст коментаря і дату створення запису.
- Payments – ця таблиця містить інформацію про платежі, включаючи ідентифікатори користувача та курсу, суму платежу, валюту, статус оплати, а також дату створення запису.

Цей набір таблиць та структура бази даних дозволяє ефективно керувати інформацією про користувачів, курси, уроки, запис на курси, рейтинги, коментарі та платежі. Використання Prisma ORM з MongoDB забезпечує гнучкість, ефективність та масштабованість нашої платформи для онлайн-навчання (рис 2.1.).

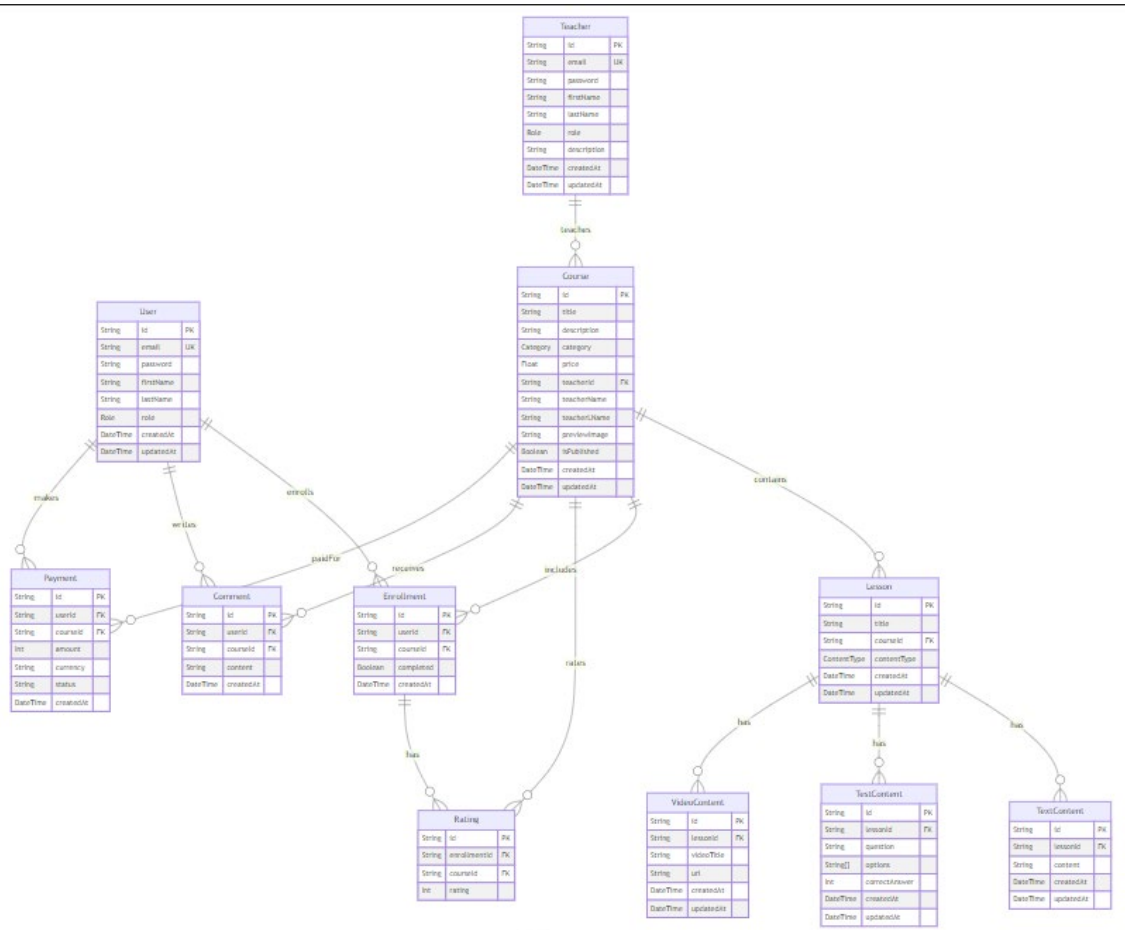


Рисунок 2.1 – Діаграма бази даних

2.3 Проектування та реалізація алгоритмів роботи системи

Сайт є простим для користувача, має весь основний функціонал сайту, що робить навігацію використання функціоналу простим та легким для юзера. Алгоритм роботи для різних ролей детально представлений нижче(рис.2.2).

Якщо авторизований вчитель:

- після авторизації вчитель має доступ до вчительської панелі;
- вчитель може створювати курси;
- вчитель має можливість керувати курсами;
- вчитель може переглядати статистику свою та курсів;
- вчитель може створювати, редагувати та видаляти курси;

Якщо авторизований користувач:

- користувач може переглядати курси;
- користувач має можливість дивитися детальну сторінку курсу;
- користувач має можливість купити курс;
- користувача може переглядати свою інформацію на сторінці профілю

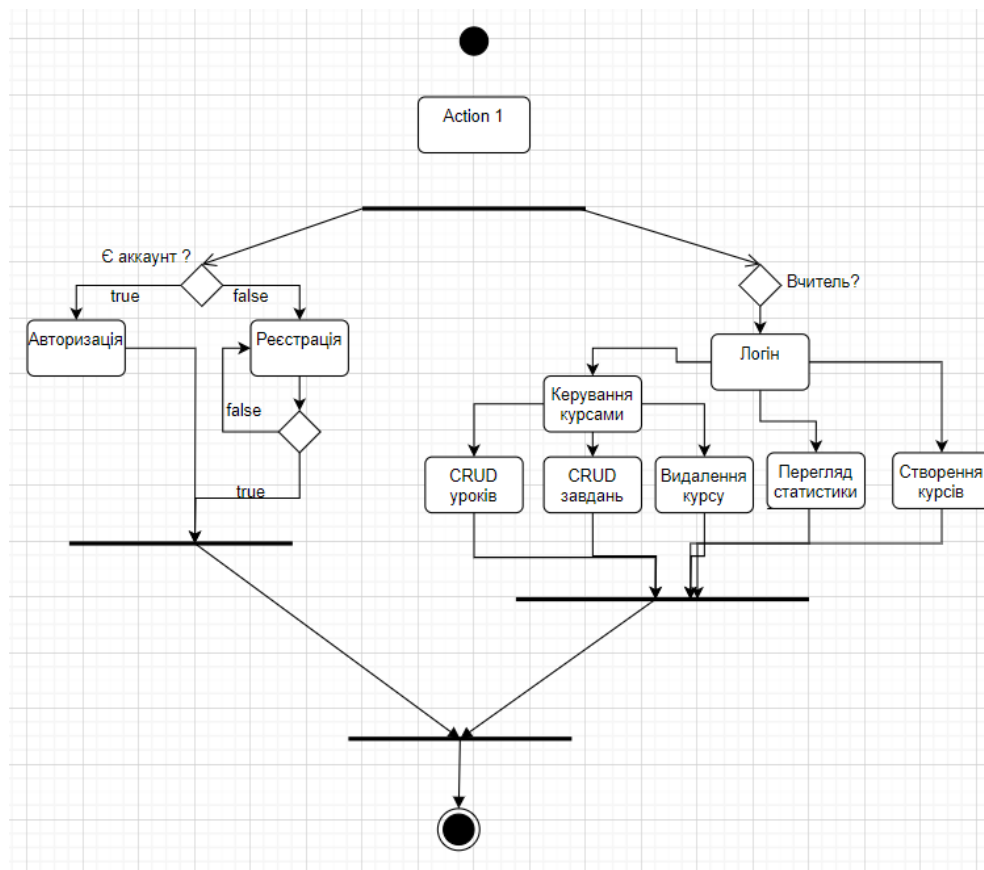


Рис.2.2. Діаграма активності

2.4 Реалізація функціоналу онлайн платформи для навчання

Розпишемо основні етапи взаємодії з сайтом. Першим етапом взаємодії є реєстрація користувача який прийшов на сайт(Рис.2.3). Процес реєстрації відбувається таким чином: під час реєстрації користувач вводить свої дані, включаючи електронну пошту, пароль, підтвердження паролю, ім'я та прізвище.

Система перевіряє, чи збігаються пароль та підтвердження паролю, а також чи вже існує в базі даних користувач з вказаною електронною поштою. Якщо перевірки пройдені успішно, пароль шифрується за допомогою бібліотеки bcrypt для забезпечення високого рівня безпеки. Після цього створюється новий запис користувача в базі даних з унікальним ідентифікатором. Таким чином, система гарантує унікальність облікових записів та захищеність паролів.

```
const userRegistration = async (req, reply) => {
  try {
    const { email, password, confirmPassword, firstName, lastName } = req.body;

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const existedUser = await prisma.user.findUnique({
      where: { email },
    });
    if (existedUser) {
      return reply.send("User with this email already exists");
    }
    if (password !== confirmPassword) {
      return reply
        .status(401)
        .send("Password are different! Please try again!");
    }

    const user = await prisma.user.create({
      data: { firstName, lastName, email, password: hashedPassword },
    });

    return reply.send(user);
  } catch (err) {
    return reply.send(err);
  }
};
```

Рис.2.3. Функція реєстрації користувача

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

Також є можливість реєстрації вчителя. (Рис. 2.4) Процес реєстрації вчителя відбувається таким чином: користувач вводить свої дані, включаючи електронну пошту, пароль, підтвердження паролю, ім'я, прізвище, роль та опис. Система перевіряє, чи вже існує в базі даних вказана електронна пошта серед зареєстрованих вчителів або користувачів. Якщо така електронна пошта вже існує, система видає відповідне повідомлення. Також перевіряється, чи збігаються пароль та підтвердження паролю. Якщо всі перевірки пройдені успішно, пароль шифрується за допомогою бібліотеки bcrypt для забезпечення високого рівня безпеки. Після цього створюється новий запис вчителя в базі даних з унікальним ідентифікатором. Таким чином, система гарантує унікальність облікових записів та захищеність паролів.

```
const teacherRegistration = async (req, reply) => {
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    const newTeacher = await prisma.teacher.create({
      data: {
        email,
        password: hashedPassword,
        firstName,
        lastName,
        role,
        description,
      },
    });
    return reply.send(newTeacher);
  } catch (err) {
    console.log(err);
    reply.status(401).send("Some error with registration! Try again!");
  }
};
```

Рис.2.4. Функція реєстрації вчителя

Розглянемо можливості вчителя на прикладі функцій, реалізованих на бекенді. Вчитель може створювати курси. Функція `createCourse` відповідає за створення нового курсу на платформі. Спочатку перевіряється наявність ідентифікатора вчителя (`teacherId`). Якщо ідентифікатор відсутній, функція повертає помилку зі статусом 400 і повідомленням "Teacher ID is required". Далі виконується пошук вчителя в базі даних за цим ідентифікатором. Якщо вчитель

не знайдений, функція повертає помилку зі статусом 404 і повідомленням "Teacher not found". Якщо в запиті міститься зображення прев'ю курсу, воно завантажується і зберігається на сервері. Шлях до зображення зберігається у змінній `previewImagePath`. Якщо всі перевірки пройдені успішно, створюється новий запис курсу в базі даних. Дані про курс включають назву, опис, категорію, ціну, ідентифікатор вчителя, ім'я та прізвище вчителя, а також шлях до зображення прев'ю. Якщо курс успішно створено, функція повертає відповідь зі статусом 201 та даними про створений курс. У випадку виникнення помилки, повертається помилка зі статусом 500 і повідомленням "Internal Server Error". (Рис.2.5).

```
const createCourse = async (req, reply) => {
  try {
    let previewImagePath = null;

    if (req.body.previewImage) {
      const data = await req.body.previewImage.toBuffer();
      const uploadPath = path.join(
        __dirname,
        "../uploads",
        req.body.previewImage.filename
      );
      await fs.promises.writeFile(uploadPath, data);
      previewImagePath = req.body.previewImage.filename;
    }
    const course = await prisma.course.create({
      data: {
        title: req.body.title.value,
        description: req.body.description.value,
        category: req.body.category.value,
        price: parseFloat(req.body.price.value),
        teacherId: teacher.id,
        teacherName,
        teacherLName,
        previewImage: previewImagePath,
      },
    });
    return reply.status(201).send(course);
  } catch (error) {
    console.error("Error creating course:", error);
    return reply.status(500).send({ message: "Internal Server Error" });
  }
};
```

Рис.2.5. Створення курсу

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

Після створення курсу на сторінці вчителя, курс редагувати, передивитися, змінити стан(активний\неактивний) або видалити. Найцікавіше це те що ж на сторінці курсу зі сторони вчителя. Він може додавати, видаляти, редагувати уроки, а в уроках маніпулювати завданнями, а саме створювати нові, або редагувати, або видаляти.

Розглянемо на прикладі створення уроку(Рис.2.6). Спочатку вона отримує ідентифікатор курсу з параметрів запиту та дані про урок з тіла запиту, включаючи назву уроку, тип контенту та сам контент. В залежності від типу контенту, функція виконує різні дії:

Якщо тип контенту - "TEXT", створюється новий урок з текстовим вмістом. Функція викликає метод `prisma.lesson.create`, який створює запис уроку в базі даних з відповідним текстовим вмістом.

Вона передає назву уроку, тип контенту, ідентифікатор курсу та текстовий контент у відповідні поля бази даних. Урок створюється разом з текстовим контентом, який зберігається у відповідній таблиці.

Якщо тип контенту - "TEST", створюється новий урок з тестовим вмістом. Метод `prisma.lesson.create` створює запис уроку в базі даних з відповідним тестовим вмістом. Урок створюється разом з тестовими запитаннями та варіантами відповідей, які зберігаються у відповідній таблиці.

Якщо тип контенту - "VIDEO", створюється новий урок з відео вмістом. Метод `prisma.lesson.create` створює запис уроку в базі даних з відповідним відео вмістом, включаючи назву відео та URL. Відео контент зберігається у відповідній таблиці разом з уроком.

Якщо тип контенту не відповідає жодному з визначених, функція повертає помилку зі статусом 400 та повідомленням "Invalid content type!".

Після успішного створення уроку, функція повертає відповідь зі статусом 201 та даними про створений урок. У разі виникнення помилки, функція повертає помилку зі статусом 500 та повідомленням "Internal server error".

```

const createLesson = async (req, reply) => {
  try {
    if (contentType === "TEXT") {
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          TextContent: {
            create: { content },
          },
        },
        include: {
          TextContent: true,
        },
      });
    } else if (contentType === "TEST") {
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          TestContent: {
            create: content,
          },
        },
        include: {
          TestContent: true,
        },
      });
    } else if (contentType === "VIDEO") {
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          VideoContent: {
            create: {
              videoTitle: content.videoTitle,
              url: content.url,
            },
          },
        },
        include: {
          VideoContent: true,
        },
      });
    }
    reply.status(201).send(newLesson);
  }
}

```

Рис.2.6. Створення уроку

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

Вчитель також може додавати завдання на цій сторінці. Розглянемо цей важливий процес (Рис.2.7). Функція `updateLesson` використовується для оновлення існуючого уроку в базі даних. Ось як відбувається процес оновлення завдання в уроках: Під час виконання запиту перевіряється наявність `lessonId` у параметрах запиту.

Якщо `lessonId` відсутній, повертається відповідь зі статусом 400 (Bad Request) і повідомленням про помилку.

Функція використовує `prisma.lesson.findUnique` для пошуку уроку за його ідентифікатором (`lessonId`).

Якщо урок не знайдено, повертається відповідь зі статусом 404 (Not Found) і повідомленням про відсутність уроку.

Функція готує об'єкт `updateData`, який містить нові дані для уроку, включаючи заголовок уроку (`title`), а також новий вміст текстових, тестових і відео завдань.

Перед створенням нових завдань видаляються всі існуючі завдання для вказаного уроку: **TextContent**: Видаляються всі текстові завдання, які належать до цього уроку, і створюються нові на основі переданих даних. **TestContent**: Видаляються всі тестові завдання, які належать до цього уроку, і створюються нові на основі переданих даних. **VideoContent**: Видаляються всі відео завдання, які належать до цього уроку, і створюються нові на основі переданих даних.

Функція використовує `prisma.lesson.update` для оновлення уроку з новими даними, які зберігаються в змінній `updatedLesson`.

Функція повертає оновлений урок у відповідь на запит. Якщо виникає помилка під час оновлення, функція повертає відповідь зі статусом 500 (Internal Server Error) і повідомленням про помилку.

Ця функція забезпечує повне оновлення уроку, включаючи видалення старого вмісту і додавання нового, що дозволяє легко керувати уроками в онлайн платформі для навчання.

```

const updateLesson = async (req, reply) => {
  try {
    const updateData = {
      title,
      TextContent: {
        deleteMany: { lessonId },
        create:
          textContents?.map((content) => ({ content: content.content })) ||
        [],
      },
      TestContent: {
        deleteMany: { lessonId },
        create:
          testContents?.map((content) => ({
            question: content.question,
            options: content.options,
            correctAnswer: parseInt(content.correctAnswer, 10),
          })) || [],
      },
      VideoContent: {
        deleteMany: { lessonId },
        create:
          videoContents?.map((content) => ({
            videoTitle: content.videoTitle,
            url: content.url,
          })) || [],
      },
    };
    const updatedLesson = await prisma.lesson.update({
      where: { id: lessonId },
      data: updateData,
    });

    return reply.send(updatedLesson);
  } catch (err) {
    console.error("Error updating lesson:", err);
    return reply.status(500).send(err);
  }
};

```

Рис.2.7. Створення завдання

Також, як раніше було озвучено Вчитель може активувати да деактивувати курси, що буде впливати на їх відображення користувачам (Рис.2.8 – 2.9). Функції `activateCourse` та `deactivateCourse` використовуються для зміни статусу курсу, тобто для його активації та деактивації відповідно. Функція `activateCourse` витягує ідентифікатор курсу (`id`) з параметрів запиту, використовує метод `prisma.course.update`, щоб знайти курс за його

ідентифікатором та оновити його статус, встановивши `isPublished` у `true`, повертає оновлений курс у відповідь, і якщо під час виконання виникає помилка, функція повертає відповідь зі статусом 500 (Internal Server Error) і повідомленням про помилку. Функція `deactivateCourse` витягує ідентифікатор курсу (`id`) з параметрів запиту, використовує метод `prisma.course.update`, щоб знайти курс за його ідентифікатором та оновити його статус, встановивши `isPublished` у `false`, повертає оновлений курс у відповідь, і якщо під час виконання виникає помилка, функція повертає відповідь зі статусом 500 (Internal Server Error) і повідомленням про помилку. Ці функції забезпечують зміну статусу курсу між активним та неактивним, що дозволяє контролювати доступ до курсів на платформі.

```
const activateCourse = async (req, reply) => {
  try {
    const { id } = req.params;
    const course = await prisma.course.update({
      where: { id },
      data: { isPublished: true },
    });
    return reply.send(course);
  } catch (err) {
    return reply.status(500).send(err);
  }
};
```

Рис.2.8. Активування курсу

```
const deactivateCourse = async (req, reply) => {
  try {
    const { id } = req.params;
    const course = await prisma.course.update({
      where: { id },
      data: { isPublished: false },
    });
    return reply.send(course);
  } catch (err) {
    return reply.status(500).send(err);
  }
};
```

Рис.2.9. Деактивування курсу

Також хотів би додати про купівлю курсу (Рис.2.10), перевірку чи зарахований користувач на курс (Рис.2.11) та зарахування (Рис.2.12). чи

відрахування (Рис.2.13). користувача з курсу. Функція `handlePayment` обробляє платіж за курс, створює об'єкт `paymentIntent` за допомогою `Stripe`, записує інформацію про платіж у базу даних, та додає користувача до курсу. У разі успіху функція повертає повідомлення про успішний платіж та зарахування, у разі помилки — статус 500 і повідомлення про помилку.

Функція `checkEnrollment` перевіряє, чи користувач записаний на курс, і повертає булеве значення, що вказує наявність зарахування. У разі помилки повертає статус 500 і повідомлення про помилку.

Функція `enrollInCourse` записує користувача на курс, створюючи відповідний запис у базі даних. У разі успіху повертає інформацію про зарахування, у разі помилки — статус 500 і повідомлення про помилку.

Функція `leaveCourse` дозволяє користувачу покинути курс, видаляючи відповідний запис з бази даних. У разі успіху повертає повідомлення про успішне покидання курсу, у разі помилки — статус 500 і повідомлення про помилку.

```
const handlePayment = async (req, reply) => {
  const paymentIntent = await stripe.paymentIntents.create({
    amount: amountInCents,
    currency: "usd",
  });
  await prisma.payment.create({
    data: {
      userId,
      courseId,
      amount: amountInCents,
      currency: "usd",
      status: "successful",
    },
  });
  reply.status(200).send({
    message: "Payment successful, enrollment created",
    paymentIntent,
    enrollment,
  });
} catch (error) {
  console.error("Error handling payment:", error);
  return reply.status(500).send(error);
}
```

Рис.2.10. Обробка оплати

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						36
Зм.	Арк.	№ докум.	Підпис	Дата		

```
const checkEnrollment = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const { id: userId } = req.user;

    const enrollment = await prisma.enrollment.findFirst({
      where: { courseId, userId },
    });
    reply.send({ enrolled: !!enrollment });
  } catch (error) {
    console.error("Error checking enrollment:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};
```

Рис.2.11. Перевірка чи зарахований

```
const enrollInCourse = async (req, reply) => {
  try {
    console.log(req.user)
    const { courseId } = req.params;
    const id = req.user.userId;

    const enrollment = await prisma.enrollment.create({
      data: { courseId, userId: id },
    });
    reply.send(enrollment);
  } catch (error) {
    console.error("Error enrolling in course:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};
```

Рис.2.12 Зарахувати на курс

```
const leaveCourse = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const { userId } = req.user;
    await prisma.enrollment.deleteMany({
      where: { courseId, userId },
    });
    reply.send({ message: "Course left successfully" });
  } catch (error) {
    console.error("Error leaving course:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};
```

Рис.2.13 Зарахувати на курс

В кінцевому результаті, маємо низку функцій які створюють і взаємодіють з багатьма об'єктами. Функціонал забезпечує коректну роботу сайту і простоту використання сайту користувачами.

Висновки до другого розділу:

В процесі проектування онлайн платформи для навчання було проведене детальний аналіз більшості сценаріїв використання програмного продукту відповідно до вимог програмного забезпечення. Виходячи з цієї інформації була спроектована структура сайту. Проектування включало в себе моделювання структури проекту та взаємодії між їх складовими.

Також було розроблено структуру бази даних для зберігання значень та взаємодії між собою та серверною частиною, таких як користувачі, курси, уроки, вчителі.

Був наведений опис одного з основних компонентів додатку – створення курсів, їх редагування та видалення.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН ПЛАТФОРМОЮ ДЛЯ НАВЧАННЯ

3.1 Проектування інтерфейсу обробки даних

Налаштування додатку передбачає розгортання всіх його компонентів і конфігурацію як кожного елементу окремо, так і всієї системи загалом. Для відображення архітектури платформи була створена діаграма розгортання (Рис. 3.1).

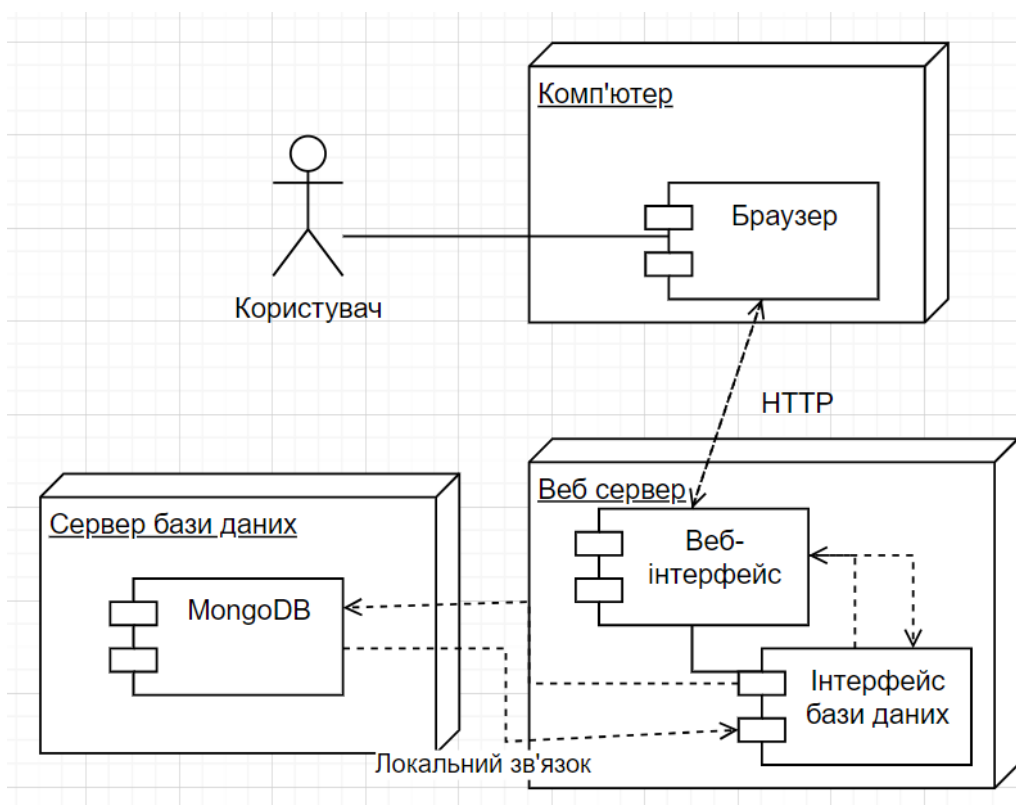


Рис. 3.1. Діаграма розгортання

Опис процесу розгортання:

- Сервер бази даних – налаштування серверної частини проекту з базою даних:
 - MongoDB – база даних, яка буде розгорнута;
- Веб сервер – налаштування клієнтської частини додатку:
 - Веб інтерфейс – інтерфейс, який бачить користувач під час використання сайту;
 - Інтерфейс бази даних – адміністративний інтерфейс для детальної інформації про стан бази даних;

- Комп'ютер – пристрій, за допомогою якого користувач отримує доступ до сайту.

3.2 Структура інтерфейсу онлайн платформи для навчання

Взаємодія користувачів відбувається через сайт. На головній сторінці користувач може побачити вітальне повідомлення, яке запрошує до платформи та розповідає чому треба вибрати саме нас. Є навігаційна панель з опціями: "Головна", "Курси", "Про Нас", "Увійти" та "Реєстрація". Також розміщено агітаційні текста для приваблення користувачів. Користувач може легко орієнтуватися по сайту завдяки чіткій структурі та зрозумілим навігаційним елементам(Рис.3.2).

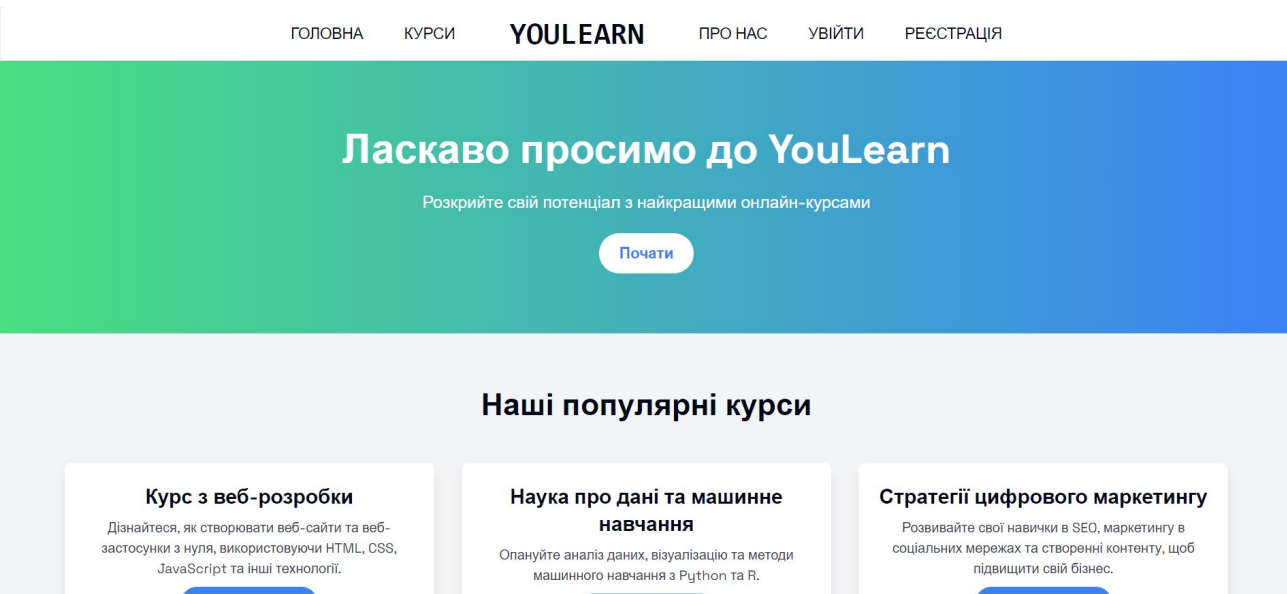


Рис. 3.2. Головна сторінка сайту

Кнопка "Голова" на навігаційній панелі сайту надає користувачам доступ до головної сторінки. Кнопка Курси перекидає на сторінку з курсами (Рис 3.3). Звідси користувач може дістатися сторінки з курсами та побачити базову інформацію про курс, назва, категорія, хто створив, ціну та дата створення.

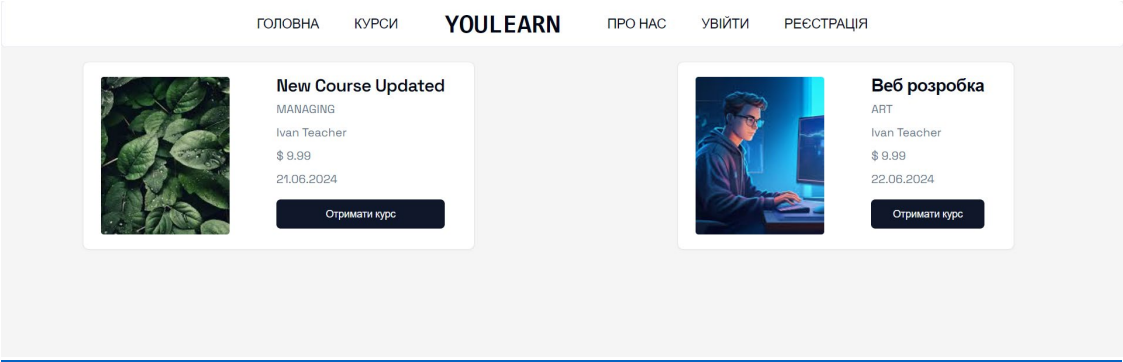


Рис. 3.3. Курси

Перейдімо до сторінки курсу (Рис 3.4.) в залежності від того якщо користувач авторизований або зарахований на курс він бачить мало кентенту прокурс. Тому давайте розглянемо реєстрацію користувача (Рис. 3.5). На цій сторінці користувач мусить ввести всю потрібну інформацію та пройти валідацію полів (Рис. 3.6).

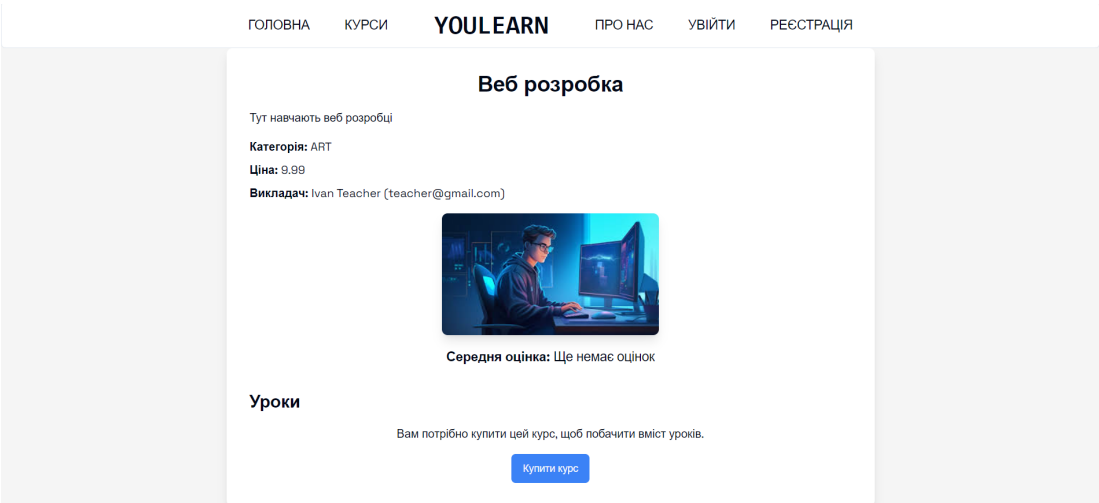


Рис.3.4. Сторінка курсу без автеризованого користувача

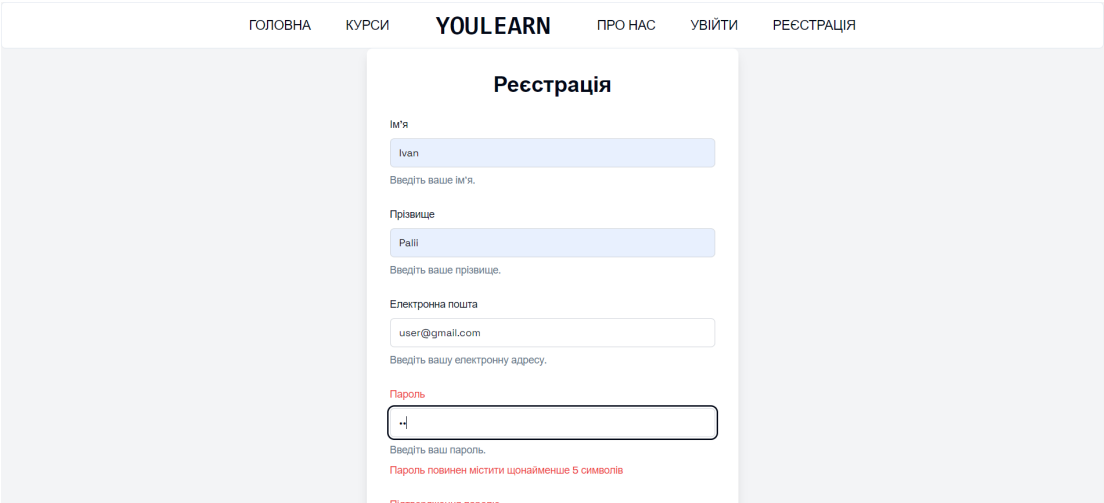


Рис.3.5. Сторінка реєстрації

Якщо користувач зареєстрований він може отримати доступ до курсу купивши його на сторінці курсу і оплати (Рис. 3.6) але перед цим він повинен увійти в аккаунт на сторінці логін (Рис. 3.7).

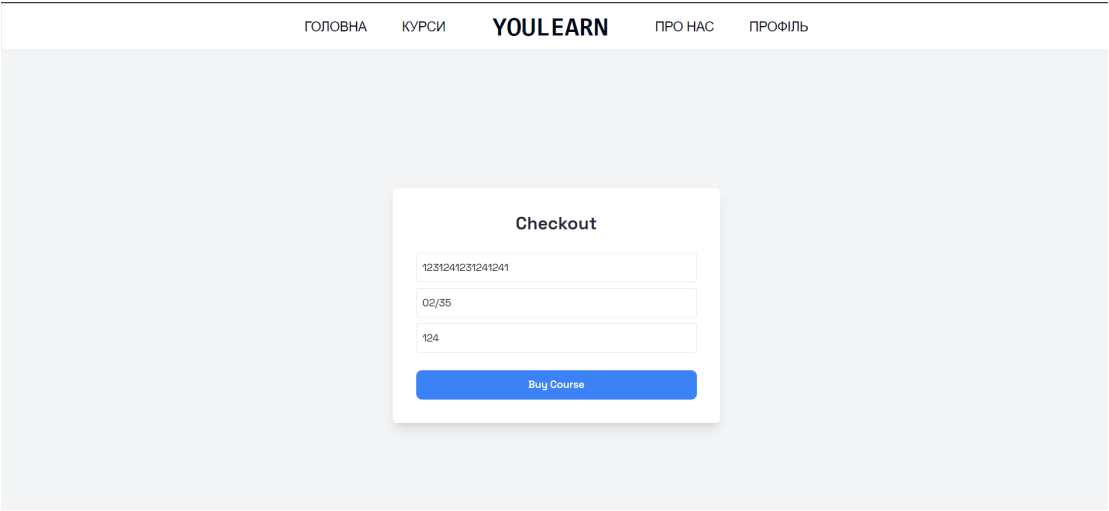


Рис.3.6. Сторінка оплати

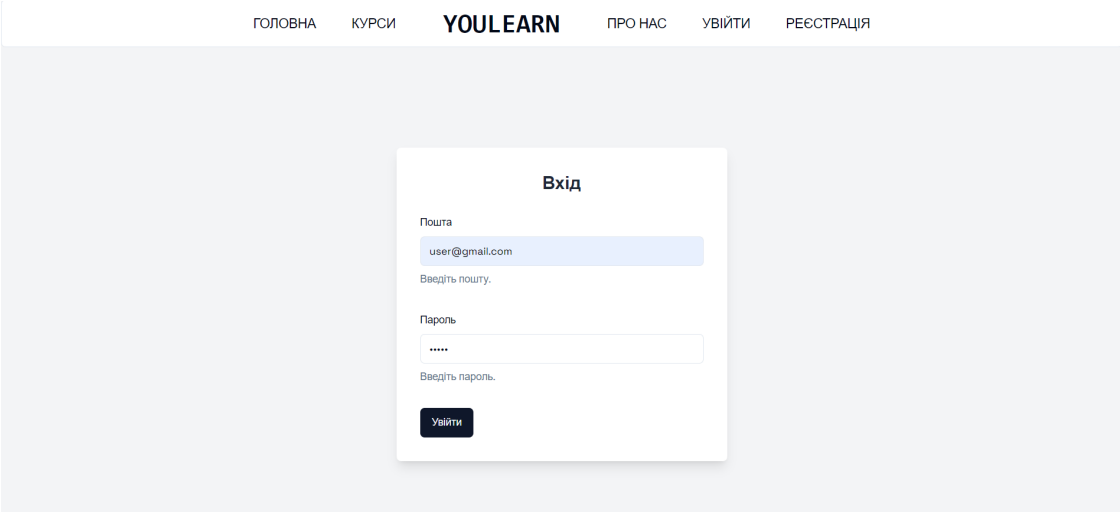


Рис.3.7. Сторінка Логін

Тепер давайте переглянемо сторінку курсу у користувача який зарахований до нього (Рис. 3.8). Тут ми бачимо всю доступну інформацію про курс включно з коментарями та рейтингом (Рис. 3.9).

New Course Updated

Updated Description

Категорія: MANAGING

Ціна: 9.99

Викладач: Ivan Teacher (teacher@gmail.com)



Середня оцінка: Ще немає оцінок

Уроки

Lesson

Lesson Test

Lesson Video

Рис.3.8. Сторінка курсу

Коментарі

COOL

Від: Ivan Palli

Додати коментар

Надіслати коментар

Оцініть цей курс

0

Надіслати оцінку

Всі оцінки

Оцінка: 4

Від: Ivan Palli

Рис.3.9. Рейтинг та коментарі

Тепер можна розглянути уроки та завдання (Рис. 3.10), а також сторінку текстового завдання (Рис. 3.11) та відео (Рис. 3.12).



Середня оцінка: 4.0

Уроки

Lesson

Lesson Test

Тип: TEST

Тестовий вміст

Заяпитання: what is 2 + 2?

☐ 3
☐ 4
☐ 5

Надіслати відповідь

Lesson Video

Повинути курс

Рис.3.10. Уроки

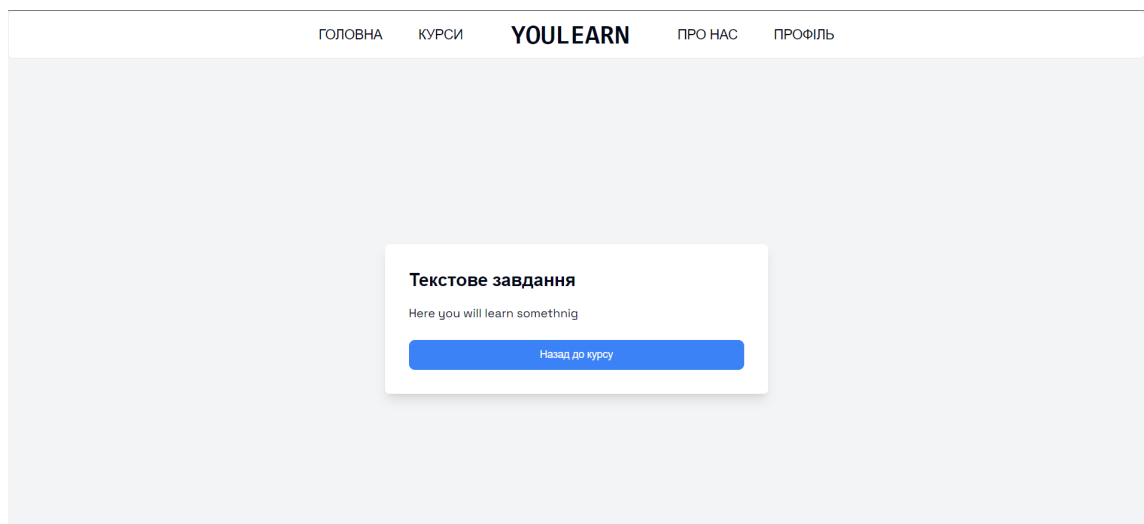


Рис.3.11. Сторінка текстового завдання

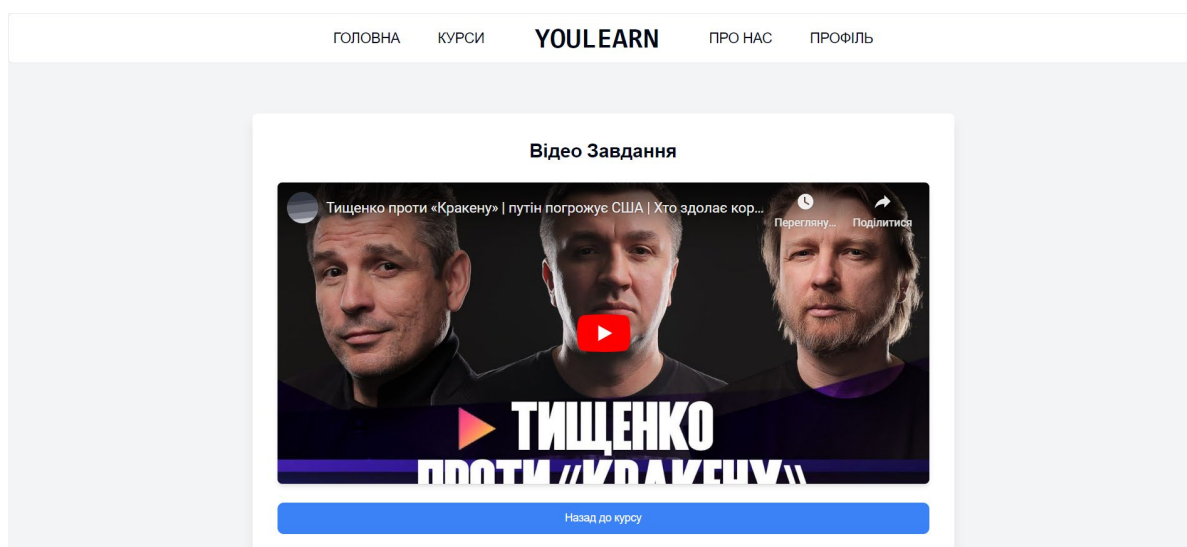


Рис.3.12. Сторінка відео завдання

Також користувач може зайти на сторінку профілю (Рис. 3.13) і побачити всю наявну інформацію про себе і відкрити налаштування (Рис. 3.14), де він може редагувати інформацію про себе

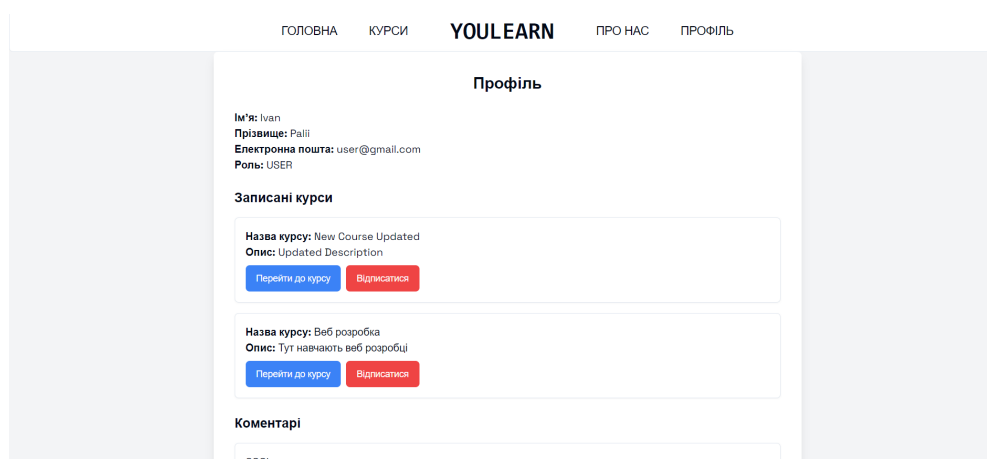


Рис.3.13. Сторінка Профілю

Рис.3.14. Сторінка налаштувань

Тепер розглянемо вчителя. Сторінка вчителя показує його основні функції, створювати курси, керувати ними та переглядати статистику (Рис. 3.15).

Рис.3.15. Сторінка вчителя

Переглянемо сторінку створення курсу де ми побачимо поля які потрібно ввести, щоб створити курс (Рис. 3.16).

Рис.3.16. Сторінка створення курсу

Після створення курсу вчителя перекидає до сторінки управління курсами де було реалізовано активаці і деактиввація курсів, перегляд їх більш розлого, та видалення курсу (Рис. 3.17)

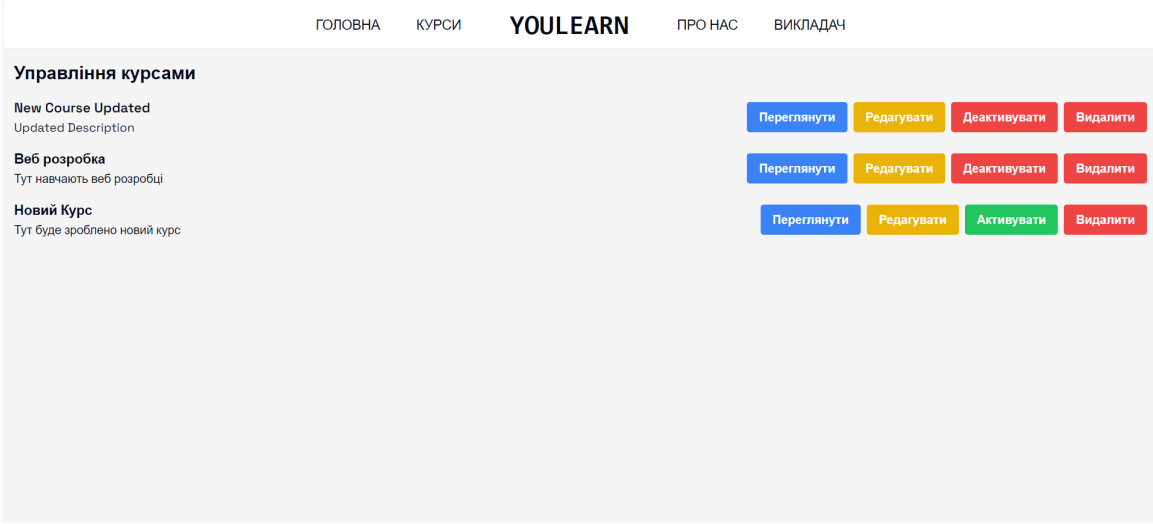


Рис.3.17. Сторінка управління курсами

Сторінка з деталями курсу виглядає так (Рис. 3.18 – 3.19). Сторінка редагування курсу виглядає так (Рис. 3.20).

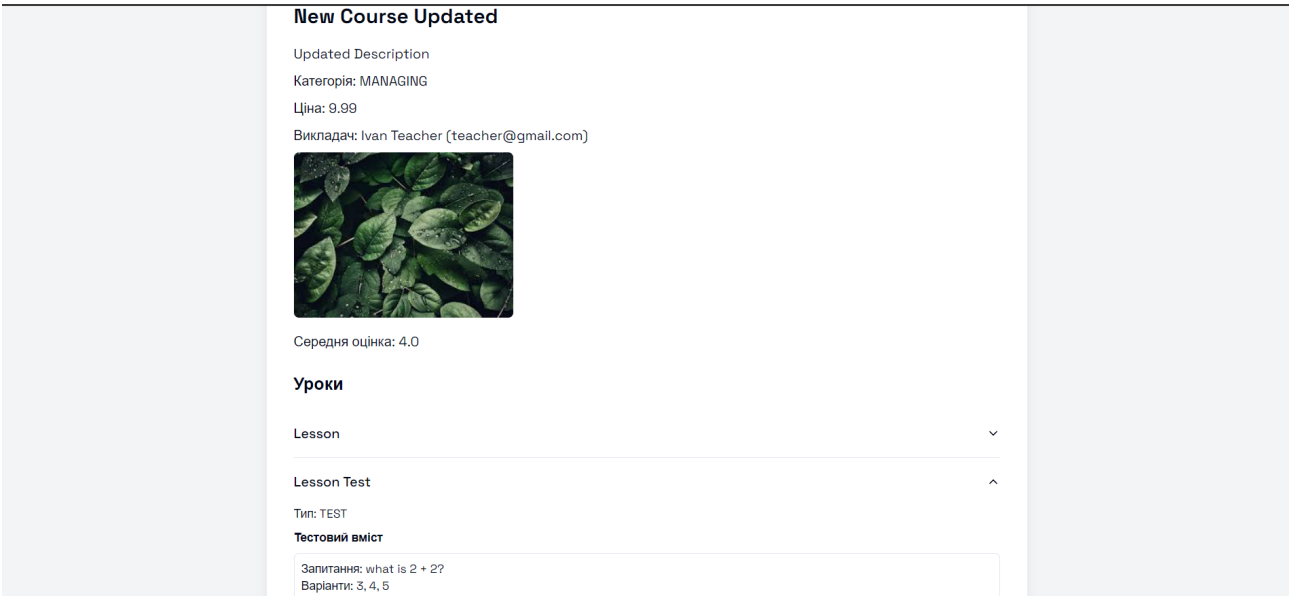


Рис.3.18. Сторінка з деталями курсу

Заявка: what is 2 + 2?
Варианты: 3, 4, 5
Правильный ответ: 1

Удалить тест

Редактировать урок

Удалить урок

Lesson Video

Добавить новый урок

Название урока

Текст

Текстовый контент

Добавить урок

Комментарии

COOL
От: Ivan Pali (user@gmail.com)
Дата: 21.06.2024, 17:37:38

Оценки

Рис.3.19. Страница с деталями курса

Редактировать курс

Название

New Course Updated

Описание

Updated Description

Категория

Управление

Цена

9.99

Предварительное изображение



Рис.3.20. Страница редактирования курса

Остались только страницы статистики (Рис. 3.21) и страница о нас (Рис. 3.22)

Панель преподавателя

Статистика

Общая количество курсов

3

Общая количество уроков

3

Общая количество записей

2

Общая количество комментариев

1

Общая количество оценок

1

Средняя оценка курса

1.3

Общий доход

\$49.95

Платежи

Курс: New Course Updated
Сумма: \$9.99
Дата: 21.06.2024
Статус: successful

Рис.3.21. Страница статистики

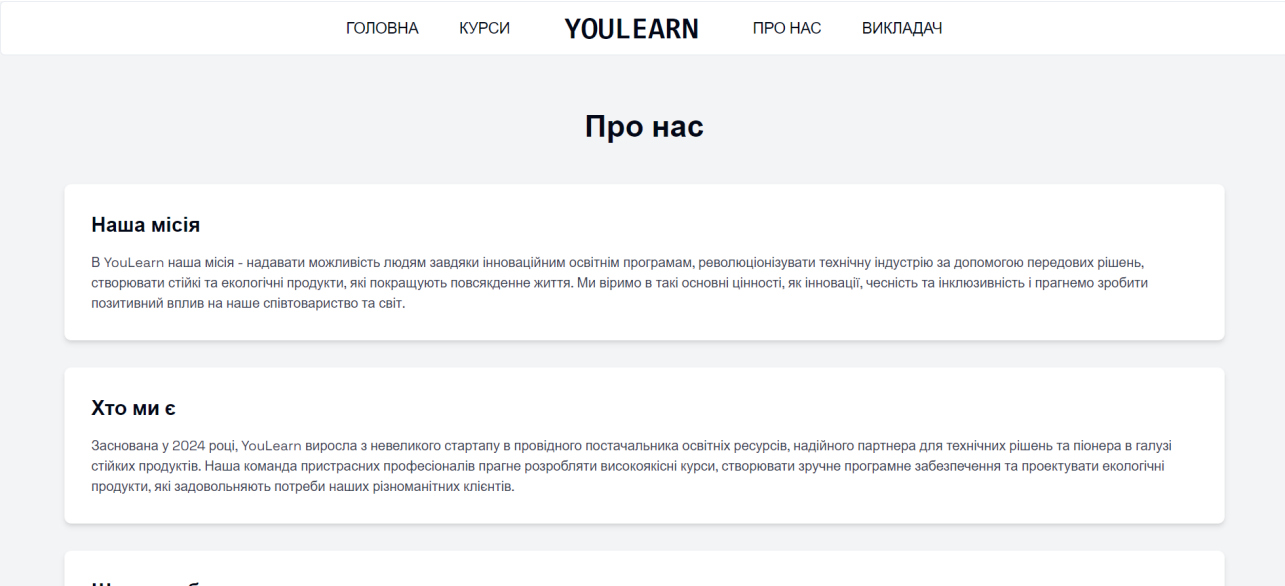


Рис. 3.22. Сторінка про нас

3.3 Тестування платформи

Тестування може проводитися як вручну, так і автоматично. Ручне тестування включає безпосереднє виконання тестових сценаріїв та аналіз результатів користувачем, що дозволяє виявляти проблеми в реальному часі та оцінювати зручність використання інтерфейсу. Цей метод ефективний для виявлення помилок, які можуть бути пропущені автоматизованими тестами, оскільки людина може звертати увагу на несподівані нюанси взаємодії з системою. Автоматизоване тестування, своєю чергою, дає змогу швидко перевіряти велику кількість сценаріїв, знижуючи ризик людської помилки та підвищуючи ефективність повторюваних тестів. Однак у цьому випадку буде використовуватися ручний аналіз для забезпечення максимальної точності та адаптивності до різних ситуацій.

Спочатку перевіримо сторінку реєстрації. (Рис.3.23) Важливо переконатися, що всі необхідні поля (ім'я користувача, електронна пошта, пароль) присутні та правильно позначені. Тестування включає введення коректних даних для успішної реєстрації та перевірку обробки помилок при введенні некоректних даних, таких як неправильний формат електронної пошти (Рис.3.24) або занадто короткий пароль. Також слід перевірити функціональність кнопки "Зареєструватися" та переконатися, що після успішної

реєстрації користувач перенаправляється на правильну сторінку, а в разі помилок отримує відповідні повідомлення.

Ім'я

Введіть ваше ім'я

Введіть ваше ім'я.

Ім'я повинно містити щонайменше 2 символи

Прізвище

Введіть ваше прізвище

Введіть ваше прізвище.

Прізвище повинно містити щонайменше 2 символи

Електронна пошта

1@f.csd

Введіть вашу електронну адресу.

Пароль

.

Введіть ваш пароль.

Пароль повинен містити щонайменше 5 символів

Підтвердження паролю

Підтвердіть ваш пароль

Підтвердіть ваш пароль.

Рис. 3.23. Тестування форми для реєстрації

Вхід

Пошта

Введіть пошту

Введіть пошту.

Не дійсний імейл

Пароль

Введіть пароль

Введіть пароль.

Пароль має бути не менше 5 символів

Увійти

Рис. 3.24. Тестування правильності запитів

Для успішного проходження валідації сторінки створення курсів ми повинні заповнити всі поля і виконати умови, як от для ціни, не має бути пустим та не має бути від’ємним (Рис. 3.25).

ГОЛОВНА

КУРСИ

YOULEARN

ПРО НАС

ВИКЛАДАЧ

Створити курс

Назва

1

Опис

4

Категорія Управління

Ціна

-99

Ціна повинна бути позитивним числом

Попереднє зображення

Вибрати файл

Файл не вибрано

Створити курс

Рис. 3.25. Тестування правильності створення курсу

Також та сама валідація працює і на сторінці оновлення профілю, але я покажу інше поле «Назва». Воно немає бути пустим (Рис. 3.26). Давайте одразу перевіримо створення завдань

Редагувати урок

Назва

Перший урок

Контент

Контент є обов'язковим

Додати текстовий контент

Додати тестовий контент

Додати відео контент

Оновити урок

Рис. 3.26 Перевірка валідації уроку

Висновки до третього розділу

У даному розділі було наведено детальну схему розгортання платформи для онлайн навчання. Кожен структурний елемент системи був описаний, а також продемонстровано його взаємодію з іншими частинами платформи. Це дозволило зрозуміти архітектуру системи та взаємозв'язки між її компонентами. Далі було надано опис інтерфейсу розробленого програмного продукту. Були розглянуті всі функціональні можливості додатку, залежно від ролі користувача. Кожен тип користувача отримав свої унікальні функції та можливості в системі. Це допомагає кожному користувачу зорієнтуватись у функціоналі додатку та використовувати його відповідно до своїх потреб.

Окрім того, було продемонстровано, як працює статистика для вчителя та детальний профіль користувача. Вчителі мають повний контроль над курсами, вони можуть відилати та оновлювати всі поля. Користувачі ж отримують цікаві курси за які вони платять сумму встановлену вчителем.

Також було детально описано, як користувачі можуть реєструватися на курс, проходити завдання та переглядати і маніпулювати своєю інформацією.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						51
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було спроектовано та реалізовано платформу для онлайн навчання. Проведено аналіз предметної області та порівняння різних платформ, що надають послуги онлайн навчання. На основі цього аналізу визначено основні вимоги до платформи. В ході дослідження розроблено архітектуру, підходи та ключові технології для реалізації сайту.

Проектування платформи включало аналіз варіантів використання програмного продукту та моделювання структури взаємодії між компонентами. Також було спроектовано базу даних для зберігання інформації про користувачів, вчителів, курси, уроки, рейтинги, коментарі, різні види контенту та оплати. Особливу увагу приділено інтеграції з системами створення курсів.

Описано алгоритм роботи з додатком та послідовність варіантів його використання. Розглянуто основний функціонал, включаючи реєстрацію користувачів, створення та редагування профілів, запис на курс та надання фідбеку про курси. Також описано функції для вчителів, включаючи створення курсів та маніпуляції з ними.

Надано детальну схему розгортання платформи для онлайн навчання, описано взаємодію структурних елементів системи та наведено опис інтерфейсу для різних типів користувачів. Показано роботу статистики для вчителів, що надає інформацію про курси та діяльність вчителя.

Архітектура платформи побудована на базі сучасних вебтехнологій, що гарантує надійність та масштабованість системи. Додаток має інтуїтивний інтерфейс, який дозволяє користувачам легко реєструватися, записуватись на курси та виконувати завдання. Інтерфейс платформи є зручним та інтуїтивно зрозумілим, що сприяє легкому використанню всіх функцій сайту та задовольняє потреби різних категорій користувачів.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						52
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Coursera [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.coursera.org/>
2. Udemy [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.udemy.com/>
3. Duolingo [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.duolingo.com/>
4. Edx [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.edx.org/>
5. Zustand Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.pmnd.rs/zustand/getting-started/introduction>
6. Fastify Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://fastify.dev/docs/latest/>
7. Shadcn Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://ui.shadcn.com/docs>
8. TailWind Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://tailwindcss.com/docs/installation>
9. Next.JS Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/getting-started/installation>
10. Fastify Ecosystem [Електронний ресурс] – Режим доступу до ресурсу:
<https://fastify.dev/ecosystem/>
11. Prisma Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.prisma.io/docs>
12. Sillito, J. JavaScript Testing Recipes: Using Mocha, Chai, and Sinon. Boston: Addison-Wesley Professional, 2013. 262 p.
13. Kosmaczewski, P. JavaScript Testing with Jasmine: JavaScript Behavior-Driven Development. Birmingham: Packt Publishing Ltd, 2016. 182 p.
14. O'Brien, T. Building React and Node.js Applications with Database Integration. Birmingham: Packt Publishing Ltd, 2021. 372 p.

15. Robinson, A. Mastering JavaScript Functional Programming. Birmingham: Packt Publishing Ltd, 2019. 324 p.
16. Myers, G.J. The Art of Software Testing. 3rd ed. Hoboken: John Wiley & Sons, 2011. 320 p.
17. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.mongodb.com>
18. React Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://reactjs.org/docs/getting-started.html>
19. Node.js Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en/docs/>
20. TypeScript Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
21. Mongoose Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/guide.html>
22. React Hook Form Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hook-form.com/get-started>
23. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/docs/installation>
24. Elliott, E. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. Sebastopol: O'Reilly Media, 2014. 254 p.
25. Searls, J. The BDD Books: Cucumber. Leanpub, 2017. 120 p.
26. Liu, A. Express in Action: Writing, Building, and Testing Node.js Applications. Greenwich: Manning Publications, 2016. 375 p.
27. Wieruch, R. The Road to React: Your Journey to Master Plain Yet Pragmatic React. Independently Published, 2019. 254 p.
28. Eden, D. Learning React: Functional Web Development with React and Redux. Sebastopol: O'Reilly Media, 2017. 358 p.
29. Sams, J. Practical Node.js: Building Real-World Scalable Web Apps. 2nd ed. New York: Apress, 2016. 340 p.

- 30.Herron, A. React Design Patterns and Best Practices. Birmingham: Packt Publishing Ltd, 2018. 330 p.
- 31.Casciaro, A. Node.js Design Patterns. 2nd ed. Birmingham: Packt Publishing Ltd, 2016. 526 p.
- 32.Bhargava, V. Pro Express.js: Mastering the Node.js Framework. New York: Apress, 2014. 273 p.
- 33.Laverde, F. Hands-On JavaScript High Performance. Birmingham: Packt Publishing Ltd, 2019. 498 p.

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						55
Зм.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А

Course-controller.js

```
const path = require("path");
const prisma = require("../utils/prisma");
const { jwtVerify, jwtDecode } = require("../utils/jwt");
const fs = require("fs");
const util = require("util");
const { pipeline } = require("stream");
const pump = util.promisify(pipeline);

const createCourse = async (req, reply) => {
  try {
    const { teacherId, teacherName, teacherLName } = req.user;

    if (!teacherId) {
      return reply.status(400).send({ message: "Teacher ID is required" });
    }

    const teacher = await prisma.teacher.findUnique({
      where: { id: teacherId },
    });

    if (!teacher) {
      return reply.status(404).send({ message: "Teacher not found" });
    }

    let previewImagePath = null;

    if (req.body.previewImage) {
      const data = await req.body.previewImage.toBuffer();
      const uploadPath = path.join(
        __dirname,
        "../uploads",
        req.body.previewImage.filename
      );
      await fs.promises.writeFile(uploadPath, data);
      previewImagePath = req.body.previewImage.filename;
    }

    const course = await prisma.course.create({
      data: {
        title: req.body.title.value,
        description: req.body.description.value,
        category: req.body.category.value,
        price: parseFloat(req.body.price.value),
        teacherId: teacher.id,
      },
    });
  } catch (error) {
    console.error(error);
    return reply.status(500).send({ message: "Internal server error" });
  }
};
```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        teacherName,
        teacherLName,
        previewImage: previewImagePath,
    },
});

return reply.status(201).send(course);
} catch (error) {
    console.error("Error creating course:", error);
    return reply.status(500).send({ message: "Internal Server Error" });
}
};

const getCourse = async (req, reply) => {
    try {
        const { courseId } = req.params;

        const course = await prisma.course.findUnique({
            where: { id: courseId },
            include: {
                Lesson: {
                    include: {
                        TextContent: true,
                        TestContent: true,
                        VideoContent: true,
                    },
                },
                Comment: {
                    include: {
                        user: true,
                    },
                },
                Rating: true,
                Enrollment: true,
                Payment: true,
                teacher: true,
            },
        });

        if (!course) {
            return reply.status(404).send("Course not found");
        }

        const averageRating =
            course.Rating.reduce((sum, rating) => sum + rating.rating, 0) /
            course.Rating.length;

        return reply.send({ ...course, averageRating });
    } catch (error) {
        console.error("Error fetching course:", error);
        return reply.status(500).send("Internal server error");
    }
};

```

```

    }
  };

const getCourses = async (req, reply) => {
  try {
    const courses = await prisma.course.findMany({
      where: { isPublished: true },
    });
    reply.send(courses);
  } catch (error) {
    reply.status(500).send(error);
  }
};

const deleteCourse = async (req, reply) => {
  try {
    const courseId = req.params.id;

    if (!courseId) {
      return reply.status(400).send({ error: "Course ID is required!" });
    }

    // Find lessons associated with the course
    const lessons = await prisma.lesson.findMany({
      where: { courseId },
    });

    // Delete each lesson
    for (const lesson of lessons) {
      await prisma.lesson.delete({
        where: { id: lesson.id },
      });
    }

    // Delete the course
    await prisma.course.delete({
      where: { id: courseId },
    });

    return reply.send({ message: `Course with ID ${courseId} was removed!` });
  } catch (err) {
    console.error(`Failed to delete course with id ${courseId}:`, err);
    return reply
      .status(500)
      .send({ error: "An error occurred while deleting the course." });
  }
};

const updateCourse = async (req, reply) => {
  try {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						58
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const { id } = req.params;
const { title, description, category, price } = req.body;
const { teacherId } = req.user;
console.log(req.body);

const course = await prisma.course.findUnique({
  where: { id },
});

if (!course) {
  return reply.status(404).send({ message: "Course not found" });
}
console.log(price);
if (course.teacherId !== teacherId) {
  return reply
    .status(403)
    .send({ message: "You are not authorized to update this course" });
}

// Update the preview image if provided
let previewImagePath = course.previewImage;

if (req.body.previewImage) {
  const data = await req.body.previewImage.toBuffer();
  const uploadPath = path.join(
    __dirname,
    "../uploads",
    req.body.previewImage.filename
  );
  await fs.promises.writeFile(uploadPath, data);
  previewImagePath = req.body.previewImage.filename;
}

const updatedCourse = await prisma.course.update({
  where: { id },
  data: {
    title: title.value,
    description: description.value,
    category: category.value,
    price: parseFloat(price.value),
    previewImage: previewImagePath,
  },
});

return reply.status(200).send(updatedCourse);
} catch (error) {
  console.error("Error updating course:", error);
  return reply.status(500).send({ message: "Internal Server Error" });
}
};

```

```

const activateCourse = async (req, reply) => {
  try {
    const { id } = req.params;
    const course = await prisma.course.update({
      where: { id },
      data: { isPublished: true },
    });
    return reply.send(course);
  } catch (err) {
    return reply.status(500).send(err);
  }
};

const deactivateCourse = async (req, reply) => {
  try {
    const { id } = req.params;
    const course = await prisma.course.update({
      where: { id },
      data: { isPublished: false },
    });
    return reply.send(course);
  } catch (err) {
    return reply.status(500).send(err);
  }
};

const checkEnrollment = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const { id: userId } = req.user;

    const enrollment = await prisma.enrollment.findFirst({
      where: { courseId, userId },
    });

    reply.send({ enrolled: !!enrollment });
  } catch (error) {
    console.error("Error checking enrollment:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};

const enrollInCourse = async (req, reply) => {
  try {

    console.log(req.user)
    const { courseId } = req.params;
    const id = req.user.userId;

    const enrollment = await prisma.enrollment.create({

```

```

    data: { courseId, userId: id },
  });

  reply.send(enrollment);
} catch (error) {
  console.error("Error enrolling in course:", error);
  reply.status(500).send({ error: "Internal server error" });
}
};

const leaveCourse = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const { userId } = req.user;

    await prisma.enrollment.deleteMany({
      where: { courseId, userId },
    });

    reply.send({ message: "Course left successfully" });
  } catch (error) {
    console.error("Error leaving course:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};

const getUserEnrolledCourses = async (req, reply) => {
  try {
    const { userId } = req.user;
    const courses = await prisma.enrollment.findMany({
      where: { userId },
      include: { course: true },
    });
    reply.send(courses);
  } catch (error) {
    reply.status(500).send(error);
  }
};

module.exports = {
  createCourse,
  getCourse,
  getCourses,
  deleteCourse,
  updateCourse,
  activateCourse,
  deactivateCourse,
  enrollInCourse,
  getUserEnrolledCourses,
  leaveCourse,

```

```

    checkEnrollment,
  };

```

Lesson-controller.js

```

const prisma = require("../utils/prisma");
const { authRole } = require("../middleware/authMiddleware");

// Create a new lesson
const createLesson = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const { title, contentType, content } = req.body;

    let newLesson;
    if (contentType === "TEXT") {
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          TextContent: {
            create: { content },
          },
        },
        include: {
          TextContent: true,
        },
      });
    } else if (contentType === "TEST") {
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          TestContent: {
            create: content,
          },
        },
        include: {
          TestContent: true,
        },
      });
    } else if (contentType === "VIDEO") {
      console.log(content);
      newLesson = await prisma.lesson.create({
        data: {
          title,
          contentType,
          courseId,
          VideoContent: {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						62
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        create: {
          videoTitle: content.videoTitle,
          url: content.url,
        },
      },
    },
    include: {
      VideoContent: true,
    },
  });
} else {
  return reply.status(400).send({ error: "Invalid content type!" });
}

reply.status(201).send(newLesson);
} catch (error) {
  console.error("Error creating lesson:", error);
  reply.status(500).send("Internal server error");
}
};

// Get a lesson by ID
const getLesson = async (req, reply) => {
  try {
    const { lessonId } = req.params;

    const lesson = await prisma.lesson.findUnique({
      where: { id: lessonId },
      include: {
        TextContent: true,
        TestContent: true,
        VideoContent: true,
      },
    });

    if (!lesson) {
      return reply.status(404).send({ error: "Lesson not found" });
    }

    return reply.send(lesson);
  } catch (error) {
    console.error("Error fetching lesson:", error);
    return reply.status(500).send("Internal server error");
  }
};

// Get all lessons for a course
const getCourseLessons = async (req, reply) => {
  try {
    const { courseId } = req.params;
    const lessons = await prisma.lesson.findMany({

```

```

        where: { courseId },
    });
    reply.send(lessons);
} catch (error) {
    reply.status(500).send(error);
}
};

// Delete a lesson
const deleteLesson = async (req, reply) => {
    try {
        const { lessonId } = req.params;

        if (!lessonId) {
            return reply.status(400).send({ error: "Lesson ID is required!" });
        }

        await prisma.lesson.delete({
            where: { id: lessonId },
        });

        return reply.send({ message: `Lesson with ID ${lessonId} was removed!`
    });
    } catch (err) {
        console.log(err);
        return reply
            .status(500)
            .send({ error: "An error occurred while deleting the lesson." });
    }
};

// Update a lesson
const updateLesson = async (req, reply) => {
    try {
        const { lessonId } = req.params;
        const { title, textContents, testContents, videoContents } = req.body;

        if (!lessonId) {
            return reply.status(400).send({ error: "Lesson ID is required!" });
        }

        const lesson = await prisma.lesson.findUnique({
            where: { id: lessonId },
            include: {
                TextContent: true,
                TestContent: true,
                VideoContent: true,
            },
        });

        if (!lesson) {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						64
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    return reply.status(404).send("No lesson found!");
  }

  const updateData = {
    title,
    TextContent: {
      deleteMany: { lessonId },
      create:
        textContents?.map((content) => ({ content: content.content })) ||
    [],
  },
  TestContent: {
    deleteMany: { lessonId },
    create:
      testContents?.map((content) => ({
        question: content.question,
        options: content.options,
        correctAnswer: parseInt(content.correctAnswer, 10),
      })) || [],
  },
  VideoContent: {
    deleteMany: { lessonId },
    create:
      videoContents?.map((content) => ({
        videoTitle: content.videoTitle,
        url: content.url,
      })) || [],
  },
};

const updatedLesson = await prisma.lesson.update({
  where: { id: lessonId },
  data: updateData,
});

return reply.send(updatedLesson);
} catch (err) {
  console.error("Error updating lesson:", err);
  return reply.status(500).send(err);
}
};

// Update a task within a lesson
const updateTask = async (req, reply) => {
  try {
    const { lessonId, contentId } = req.params;
    const data = req.body;

    if (!lessonId) {
      return reply.status(400).send({ error: "Lesson ID is required!" });
    }
  }

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						65
Зм.	Арк.	№ докум.	Підпис	Дата		

```

console.log(data);
const lesson = await prisma.lesson.findUnique({
  where: { id: lessonId },
  include: {
    TextContent: true,
    TestContent: true,
    VideoContent: true,
  },
});

if (!lesson) {
  return reply.status(404).send("No lesson found!");
}

let updateData = { title: lesson.title };

if (lesson.contentType === "TEXT") {
  if (contentId) {
    // Update existing text content
    updateData.TextContent = {
      update: {
        where: { id: contentId },
        data: { content: data.content },
      },
    };
  } else {
    // Create new text content
    updateData.TextContent = {
      create: {
        content: data.content,
      },
    };
  }
} else if (lesson.contentType === "TEST") {
  if (contentId) {
    // Update existing test content
    updateData.TestContent = {
      update: {
        where: { id: contentId },
        data: {
          question: data.question,
          options: data.options,
          correctAnswer: parseInt(data.correctAnswer, 10),
        },
      },
    };
  } else {
    // Create new test content
    updateData.TestContent = {
      create: {
        question: data.question,

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						66
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        options: data.options,
        correctAnswer: parseInt(data.correctAnswer, 10),
      },
    };
  }
} else if (lesson.contentType === "VIDEO") {
  if (contentId) {
    // Update existing video content
    updateData.VideoContent = {
      update: {
        where: { id: contentId },
        data: {
          videoTitle: data.videoTitle,
          url: data.url,
        },
      },
    };
  } else {
    // Create new video content
    updateData.VideoContent = {
      create: {
        videoTitle: data.videoTitle,
        url: data.url,
      },
    };
  }
}

const updatedTask = await prisma.lesson.update({
  where: { id: lessonId },
  data: updateData,
});

return reply.send(updatedTask);
} catch (err) {
  console.error("Error updating task:", err);
  return reply.status(500).send(err);
}
};

// Delete a task within a lesson
const deleteTask = async (req, reply) => {
  try {
    const { lessonId, contentId } = req.params;

    if (!lessonId) {
      return reply.status(400).send({ error: "Lesson ID is required!" });
    }
    if (!contentId) {
      return reply.status(400).send({ error: "Content ID is required!" });
    }
  }
}

```

```

const lesson = await prisma.lesson.findUnique({
  where: { id: lessonId },
  include: {
    TextContent: true,
    TestContent: true,
    VideoContent: true,
  },
});

if (!lesson) {
  return reply.status(404).send("No lesson found!");
}

if (lesson.contentType === "TEXT") {
  await prisma.textContent.delete({
    where: { id: contentId },
  });
  return reply.send({ success: true });
} else if (lesson.contentType === "TEST") {
  await prisma.testContent.delete({
    where: { id: contentId },
  });
  return reply.send({ success: true });
} else if (lesson.contentType === "VIDEO") {
  await prisma.videoContent.delete({
    where: { id: contentId },
  });
  return reply.send({ success: true });
}

return reply.status(400).send({ error: "Invalid content type!" });
} catch (err) {
  return reply.status(500).send(err);
}
};

// Get text task details
const getTextTask = async (req, reply) => {
  try {
    const { lessonId } = req.params;
    const task = await prisma.textContent.findFirst({
      where: { lessonId },
    });
    if (!task) {
      return reply.status(404).send({ error: "Task not found" });
    }
    reply.send(task);
  } catch (error) {
    console.error("Error fetching text task:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
}

```

```

    }
  };

  // Get test task details
  const getTestTask = async (req, reply) => {
    try {
      const { lessonId } = req.params;
      const task = await prisma.testContent.findFirst({
        where: { lessonId },
      });
      if (!task) {
        return reply.status(404).send({ error: "Task not found" });
      }
      reply.send(task);
    } catch (error) {
      console.error("Error fetching test task:", error);
      reply.status(500).send({ error: "Internal server error" });
    }
  };

  // Get video task details
  const getVideoTask = async (req, reply) => {
    try {
      const { lessonId } = req.params;
      const task = await prisma.videoContent.findFirst({
        where: { lessonId },
      });
      if (!task) {
        return reply.status(404).send({ error: "Task not found" });
      }
      reply.send(task);
    } catch (error) {
      console.error("Error fetching video task:", error);
      reply.status(500).send({ error: "Internal server error" });
    }
  };

  const checkTestAnswer = async (req, reply) => {
    try {
      const { lessonId, taskId } = req.params;
      const { selectedOption } = req.body;

      const testContent = await prisma.testContent.findUnique({
        where: { id: taskId },
      });

      if (!testContent) {
        return reply.status(404).send({ error: "Test content not found" });
      }

      const isCorrect = testContent.correctAnswer === selectedOption;

```

```

    reply.send({ isCorrect });
  } catch (error) {
    console.error("Error checking test answer:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};

module.exports = {
  createLesson,
  getLesson,
  getCourseLessons,
  deleteLesson,
  updateLesson,
  updateTask,
  deleteTask,
  getTextTask,
  getTestTask,
  getVideoTask,
  checkTestAnswer,
};

```

User-controller.js

```

const bcrypt = require("bcrypt");
const prisma = require("../utils/prisma");
const { jwtSign } = require("../utils/jwt");

// Registration
const userRegistration = async (req, reply) => {
  try {
    const { email, password, confirmPassword, firstName, lastName } =
      req.body;

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const existedUser = await prisma.user.findUnique({
      where: { email },
    });

    if (existedUser) {
      return reply.send("User with this email already exists");
    }

    if (password !== confirmPassword) {
      return reply
        .status(401)
        .send("Password are different! Please try again!");
    }

    const user = await prisma.user.create({

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						70
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        data: { firstName, lastName, email, password: hashedPassword },
    });

    return reply.send(user);
} catch (err) {
    return reply.send(err);
}
};

// Log in
const userLogIn = async (req, reply) => {
    const { email, password } = req.body;

    try {
        const user = await prisma.user.findUnique({
            where: { email },
        });

        if (!user) {
            return reply.status(401).send("Invalid email or password! Try again!");
        }

        const unhashedPassword = await bcrypt.compare(password, user.password);

        if (!unhashedPassword) {
            return reply.status(401).send("Invalid email or password! Try again!");
        }

        const token = await jwtSign({ userId: user.id, role: user.role });
        reply.setCookie("token", token, { path: "/" });

        return reply.send({ token, user });
    } catch (error) {
        return reply.status(500).send(error);
    }
};

// Log out
const userLogOut = async (req, reply) => {
    reply.clearCookie("token", { path: "/" });
    return reply.send({ message: "Logout successful" });
};

const getUser = async (req, reply) => {
    try {
        const { userId } = req.user;

        const user = await prisma.user.findUnique({
            where: {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						71
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        id: userId,
      },
      include: {
        Enrollment: {
          include: {
            course: true,
          },
        },
      },
    },
  });

  if (!user) {
    return reply.send("No user found!");
  }

  return reply.send(user);
} catch (err) {
  return reply.send(err);
}
};

const getUsers = async (req, reply) => {
  const users = await prisma.user.findMany();

  if (req.cookies) {
    console.log("Cookies: " + req.cookies.access_token);
  } else if (!req.cookies) {
    console.log("NO cookies found");
  }

  return reply.send(users);
};

const updateUser = async (req, reply) => {
  try {
    const { email, firstName, lastName } = req.body;

    const { userId } = req.user;

    const user = await prisma.user.findUnique({
      where: {
        id: userId,
      },
    });

    if (!user) {
      return reply.send("No user found!");
    }

    const updatedUser = await prisma.user.update({
      where: {

```

```

        id: userId,
      },
      data: { email, firstName, lastName },
    });

    return reply.send(updatedUser);
  } catch (err) {
    return reply.send(err);
  }
};

const getUserPayments = async (req, reply) => {
  const userId = req.user.userId;

  try {
    const payments = await prisma.payment.findMany({
      where: {
        userId,
      },
    });

    reply.send(payments);
  } catch (error) {
    reply.status(500).send(error);
  }
};

const getUserComments = async (req, reply) => {
  try {
    const { userId } = req.user;

    const comments = await prisma.comment.findMany({
      where: { userId },
      include: { course: true },
    });

    reply.send(comments);
  } catch (error) {
    console.error("Error fetching user's comments:", error);
    reply.status(500).send({ error: "Internal server error" });
  }
};

// Get user's ratings
const getUserRatings = async (req, reply) => {
  try {
    const { userId } = req.user;

    const ratings = await prisma.rating.findMany({
      where: {
        enrollment: {

```

```

        userId: userId,
      },
    },
    include: {
      course: true,
      enrollment: {
        include: {
          user: true,
        },
      },
    },
  },
});

reply.send(ratings);
} catch (error) {
  console.error("Error fetching user's ratings:", error);
  reply.status(500).send({ error: "Internal server error" });
}
};

module.exports = {
  userRegistration,
  userLogIn,
  userLogOut,
  getUser,
  getUsers,
  updateUser,
  getUserPayments,
  getUserComments,
  getUserRatings,
};

```