

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:

«Система автоматизації онлайн-навчання»

Виконав студент групи ІПЗ-20-1
КУЧЕР Віктор Олександрович

Керівник роботи:
ГОРДІЄНКО Ірина Валеріївна

Консультант:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2024

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

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КУЧЕР Віктор Олександрович**

Керівник роботи: **ГОРДІЄНКО Ірина Валеріївна**

Тема роботи: **«Система автоматизації онлайн-навчання»**,

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

Вихідні дані для роботи: **процеси автоматизації навчання в онлайн-середовищі.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Гордієнко І.В., Чижмотря О.Г.	27.02.2024р.	27.02.2024р.
2	Гордієнко І.В., Чижмотря О.Г.	27.02.2024р.	27.02.2024р.
3	Гордієнко І.В., Чижмотря О.Г.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Пояснювальна записка до випускної роботи містить текстову частину, викладену на 60 сторінках друкованого тексту, має 26 сторінок додатків. Список використаних джерел містить 30 найменувань і займає 3 сторінки. В роботі наведено 44 рисунків та 15 таблиць. Загальний обсяг роботи - 86 сторінок.

Метою роботи є створення системи автоматизації онлайн навчання з використанням сучасних технологій для поліпшення освітнього процесу, забезпечення доступу до навчальних матеріалів та інтерактивної взаємодії між студентами та викладачами.

Архітектура системи онлайн навчання побудована на основі веб-технологій, що забезпечує зручний доступ та управління навчальними ресурсами. Важливим компонентом системи є інтеграція інструментів для створення та проведення тестів, а також системи управління курсами, що дозволяє автоматизувати процеси реєстрації на курси, відстеження успішності студентів та управління навчальними матеріалами.

Система має зручний інтерфейс, який дозволяє користувачам легко організовувати та проводити онлайн заняття, встановлювати навчальні цілі, та забезпечувати зворотний зв'язок. Інтерактивні можливості системи сприяють активному залученню студентів до навчального процесу, а також поліпшенню якості навчання через використання сучасних методів та технологій.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, АВТОМАТИЗАЦІЯ, ОНЛАЙН НАВЧАННЯ, СИСТЕМА УПРАВЛІННЯ КУРСАМИ, ІНТЕРАКТИВНІСТЬ, ОСВІТНІ ТЕХНОЛОГІЇ

					ІПЗ.КР.Б-121-24-ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Кучер В.О.			Система автоматизації онлайн-навчання			Літ.	Арк.	Аркушів	
Керівник		Гордієнко І.В.								3	86
Рецензент		Савіцький Р.С.			Житомирська політехніка, Група ІПЗк-22-1						
Зав. каф.		Вакалюк Т.А.									

ABSTRACT

The explanatory note to the final work contains a text part set out on 60 pages of printed text, has 26 pages of appendices. The list of used sources contains 30 titles and occupies 3 pages. The paper contains 44 figures and 15 tables. The total amount of work is 86 pages.

The goal of this work is to create an online learning automation system using modern technologies to improve the educational process, provide access to learning materials, and facilitate interactive interaction between students and teachers.

The architecture of the online learning system is based on web technologies, ensuring convenient access and management of educational resources. An essential component of the system is the integration of tools for creating and conducting tests, as well as a course management system that allows automating processes such as course registration, tracking student progress, and managing learning materials.

The system has a user-friendly interface that enables users to easily organize and conduct online classes, set educational goals, and provide feedback. The interactive capabilities of the system promote active student engagement in the learning process and improve the quality of education through the use of modern methods and technologies.

KEYWORDS: WEB APPLICATION, AUTOMATION, ONLINE LEARNING, COURSE MANAGEMENT SYSTEM, INTERACTIVITY, EDUCATIONAL TECHNOLOGIES

					ИПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН-НАВЧАННЯ.....	9
1.1 Постановки задачі	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновки до першого розділу.	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН- НАВЧАННЯ.....	24
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	24
2.2 Розробка бази даних системи	31
2.3 Проектування та реалізація алгоритмів роботи системи	33
2.4 Реалізація функціоналу системи автоматизації онлайн-навчання	35
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН-НАВЧАННЯ.....	41
3.1 Порядок встановлення та налаштування параметрів платформи	41
3.2 Структура інтерфейсу системи автоматизації онлайн-навчання.	42
3.3 Тестування системи	53
Висновки до третього розділу	53
ВИСНОВКИ.....	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	57
ДОДАТКИ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (Application Programming Interface)

CI/CD – безперервна інтеграція та безперервна доставка (Continuous Integration/Continuous Delivery)

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

IDE – інтегроване середовище розробки (Integrated Development Environment)

JSON – JavaScript об'єктний синтаксис (JavaScript Object Notation)

MVC – схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

QA – забезпечення якості (Quality Assurance)

REST – передача представницького стану (Representational State Transfer)

SDK – набір для розробки програмного забезпечення (Software Development Kit)

SQL – мова структурованих запитів (Structured Query Language)

UI – інтерфейс користувача (User Interface)

UX – досвід користувача (User Experience)

БД – база даних (Database)

ПЗ – програмне забезпечення (Software)

СУБД – система управління базами даних (Database Management System)

ВСТУП

Актуальність теми. Сучасні інформаційні технології суттєво змінюють освітній процес, відкриваючи нові можливості для автоматизації навчання. Актуальність обраної теми обумовлена необхідністю покращення доступу до якісної освіти, особливо в умовах глобальної пандемії COVID-19, що спричинила масовий перехід до дистанційного навчання. Системи автоматизації онлайн-навчання дозволяють не тільки зберегти якість освіти на високому рівні, але й підвищити ефективність навчального процесу завдяки інтерактивним елементам, персоналізації навчання та постійному моніторингу результатів студентів. Сучасні напрямки застосування інформаційних технологій у сфері освіти охоплюють використання адаптивних навчальних систем, аналіз великих даних для оптимізації освітнього процесу, а також впровадження штучного інтелекту для підтримки навчання.

Метою даної роботи є розробка та впровадження системи автоматизації онлайн-навчання з використанням сучасних веб-технологій. Основний результат, якого має бути досягнуто, – це створення платформи, що забезпечує зручний доступ до навчальних матеріалів та автоматизацію управління навчальними процесами.

Для здійснення даної дипломної роботи були поставлені такі задачі:

1. Провести аналіз існуючих систем автоматизації онлайн навчання та їх функціональних можливостей.
2. Визначити вимоги до системи автоматизації онлайн навчання вищого навчального закладу.
3. Розробити архітектуру системи і її компонентів.
4. Розгортання та тестування розробленої системи на навчальному закладі.
5. Оцінити ефективність впровадженої системи та надати рекомендації для подальшого розвитку.

Об'єктом дослідження є процеси автоматизації навчання в онлайн-середовищі. Предмет дослідження міститься в межах об'єкта і включає розробку, впровадження та оцінку ефективності системи автоматизації онлайн-навчання.

Предметом дослідження є розробка та впровадження системи автоматизації онлайн-навчання, яка включає в себе створення функціональних модулів для управління курсами, організації онлайн-занять, моніторингу успішності студентів, а також інтеграцію сучасних технологій для забезпечення інтерактивності та персоналізації навчального процесу. Особлива увага приділяється аналізу та застосуванню веб-технологій, методів управління навчальними матеріалами та комунікаційними засобами, що сприяють ефективному та зручному використанню системи студентами та викладачами.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН-НАВЧАННЯ

1.1 Постановки задачі

Метою кваліфікаційної роботи бакалавра є створення системи автоматизації онлайн-навчання, що використовує сучасні технології для забезпечення зручності користування та можливості масштабування в майбутньому. У сучасному світі онлайн-навчання стає все більш популярним і затребуваним, оскільки дозволяє здобувати освіту незалежно від географічного розташування. Однак існуючі системи онлайн-навчання часто стикаються з проблемами ефективності управління, інтерактивності та персоналізації навчального процесу.

Завданням даної роботи є розробка системи, яка забезпечить високий рівень автоматизації процесів навчання, зокрема створення, адміністрування курсів, моніторинг прогресу учнів, взаємодію між учасниками навчального процесу та забезпечення безпеки даних.

Система автоматизації онлайн навчання складається з таких основних компонентів:

1. Модуля управління навчальними матеріалами: завантаження, зберігання та організація навчальних ресурсів, таких як лекції, презентації, відеоматеріали та інші документи.

2. Модуль інтерактивної взаємодії: дозволяє студентам і вчителям спілкуватися через веб-конференції та чати.

3. Модуль контролю успішності: надає статистику успішності та автоматично перевіряє розуміння учнями матеріалу за допомогою тестів і завдань.

4. Адміністративний модуль включає керування системою, управління користувачами, перегляд статистики та додавання нового функціоналу.

Основні етапи розвитку:

1. Розробка дизайну веб-сайту:

1.1. Розробка концепції дизайну системи.

1.2. Створення компонентів інтерфейсу користувача

- 1.3. Дизайн сторінок
2. Побудова архітектури користувача додатку:
 - 2.1. Вибір правильного фреймворку
 - 2.2. Проектування елементів інтерфейсу користувача
 - 2.3. Забезпечує взаємодію з серверною частиною.
3. Побудова архітектури серверної частини додатку:
 - 3.1. Вибір бекенд-технології
 - 3.2. Розробка API, яка дозволить взаємодіяти з клієнтською частиною
 - 3.3. Інтеграція до бази даних
4. Розробка системи авторизації та ролей користувачів:
 - 4.1. Створення інструментів для входу та реєстрації.
 - 4.2. Розігрування ролей користувачів (студенти, вчителі, адміністратори)
5. Розробка основного функціоналу включає в себе:
 - 5.1. Розробку логіки роботи модулів.
 - 5.2. Розробка бази даних для зберігання даних
 - 5.3. Інтеграція бекенду та взаємодія з клієнтами
6. Тестування додатку: перевірка роботи системи:
 - 6.1. Функціональне тестування системи.
 - 6.2. Тестування графічного інтерфейсу.

Результатом виконання цього плану є створення ефективної та зручної системи автоматизації онлайн-навчання, яка сприятиме підвищенню якості освіти та задоволенню потреб сучасного суспільства.

1.2 Аналіз аналогів програмного продукту

Щоб створити систему автоматизації онлайн-навчання, необхідно ретельно вивчити та оцінити існуючі продукти на ринку, які вже використовуються в цій галузі. Нижче наведено огляд п'яти найвідоміших і ефективних платформ онлайн-навчання.

Welcome to the Moodle community

The place to get support, ask and answer questions and contribute to the open source learning platform, Moodle LMS.

[Get involved](#) →

 47,814,568+ Courses

 162,797+ Sites

 241+ Countries

[Discover Moodle](#)
[Download Moodle LMS 4](#)
[Plugins](#)
[Looking for your Moodle site?](#)

Якщо ви продовжуєте переглядати цей веб-сайт, ви погоджуєтесь з нашою політикою:
[Site policy](#), [Privacy notice](#), [Cookies policy](#)

Продовжити

Рисунок 1.1. - Платформа Moodle

Moodle - це безкоштовна онлайн-система керування навчанням, яка дозволяє викладачам створювати власний приватний веб-сайт, наповнений динамічними курсами, які продовжують навчання в будь-який час і в будь-якому місці [1].

Moodle є однією з найпоширеніших платформ управління навчанням у всьому світі. Вона пропонує широкий спектр функцій як для вчителів, так і для студентів, включаючи управління курсами, завдання, тести та форум.

Основні можливості:

- Створення та управління курсами.
- Оцінювання та зворотний зв'язок.
- Увімкнений форум.
- Інтеграція з іншими системами через API.
- Великий вибір плагінів для розширення функціоналів.

Переваги:

- Відкритий код та можливість кастомізації.
- Велика спільнота користувачів та розробників.
- Розширені можливості завдяки плагінам.

Недоліки:

- Складність налаштування та адміністрування.
- Менше інтуїтивний інтерфейс для навчання.

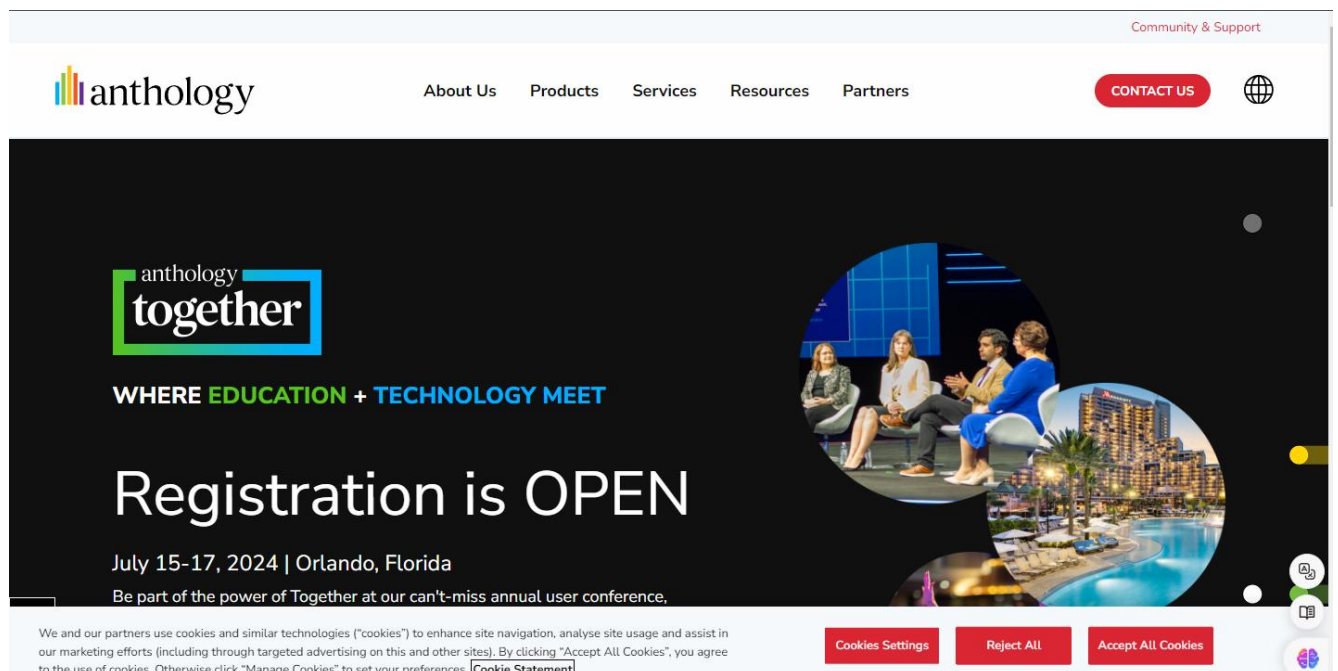


Рисунок 1.2. - Платформа Blackboard Learn

Blackboard Learn є комерційною платформою управління навчанням з великими можливостями як для онлайн, так і для змішаного навчання.

Blackboard Learn має найшвидший темп розвитку продукту на ринку та справді є LMS для сучасного викладання та навчання у вищій освіті [2].

Основні можливості:

- Створення та управління курсами.
- Інтеграція з іншими системами освіти.
- Мобільні додатки для доступу до навчальних матеріалів.
- Інструменти для аналітики та звітності.
- Відеоконференції.

Переваги:

- Інтуїтивний інтерфейс.
- Потужні аналітичні можливості.
- Професійна підтримка.

Недоліки:

- Велика вартість.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						12

- Закритий код, обмежені умови кастомізації.



Рисунок 1.3. - Платформа Canvas

Canvas LMS — це відкрита, розширювана система керування навчанням, яка підсилює приголомшливість кожного [3].

Основні можливості:

- Інтуїтивний інтерфейс для створення курсів.
- Інтеграція з іншими сервісами та додатками.
- Мобільний доступ.
- Інструменти для комунікації і співпраці.
- Потужні аналітичні можливості.

Переваги:

- Зручний та модний інтерфейс.
- Могутня система сповіщень та повідомлень.
- Відкрите API для розширення функціоналу.

Недоліки:

- Вартість ліцензії може бути великою.
- Обмежена можливість кастомізації у порівнянні з Moodle.

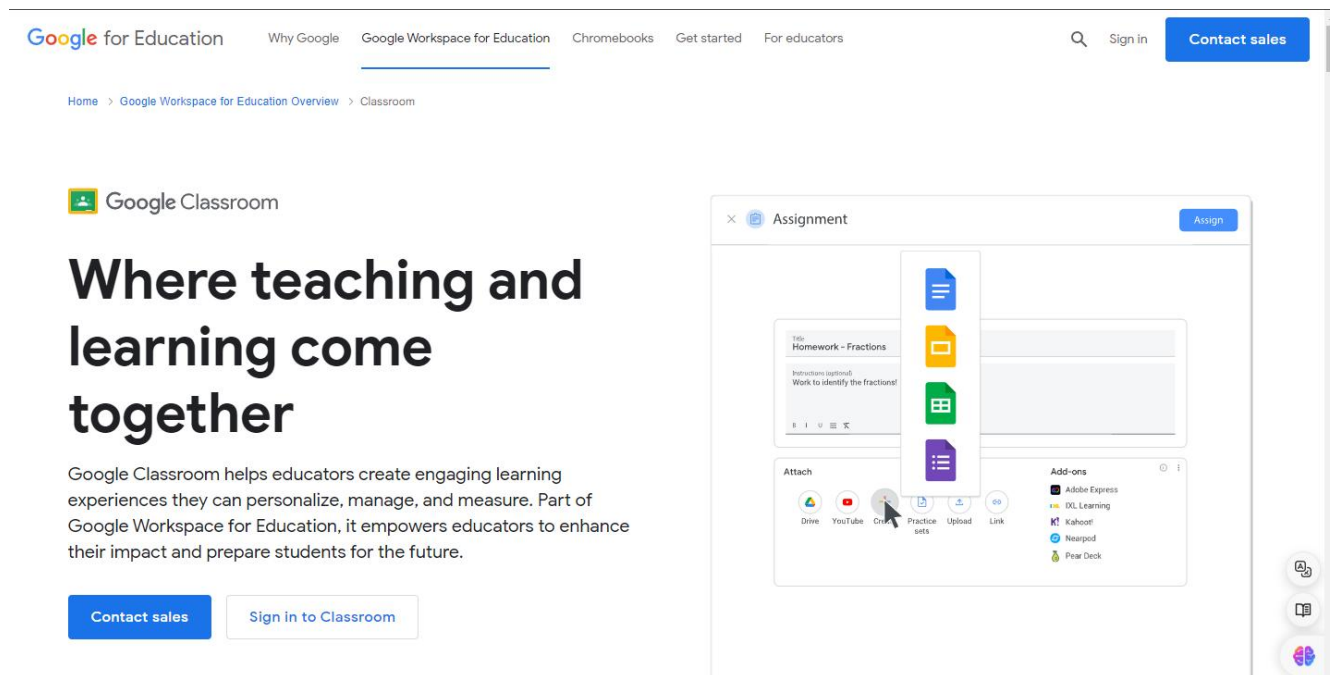


Рисунок 1.4. - Платформа Google Classroom

Google Classroom - це передова онлайн-система управління навчанням, розроблена Google з метою сприяння ефективній комунікації та співпраці між викладачами та студентами [4].

Основні можливості:

- Управління курсами та завданнями.
- Спільне використання документів та файлів.
- Інтеграція з іншими сервісами Google.
- Створення й оцінка тестів.
- Мобільний доступ.

Переваги:

- Безкоштовне використання.
- Інтеграція з Google сервісами.
- Простота використання.

Недоліки:

- Обмеження функціонала у порівнянні з професійними LMS.
- Відсутність деяких інструментів для розширеного управління навчанням.

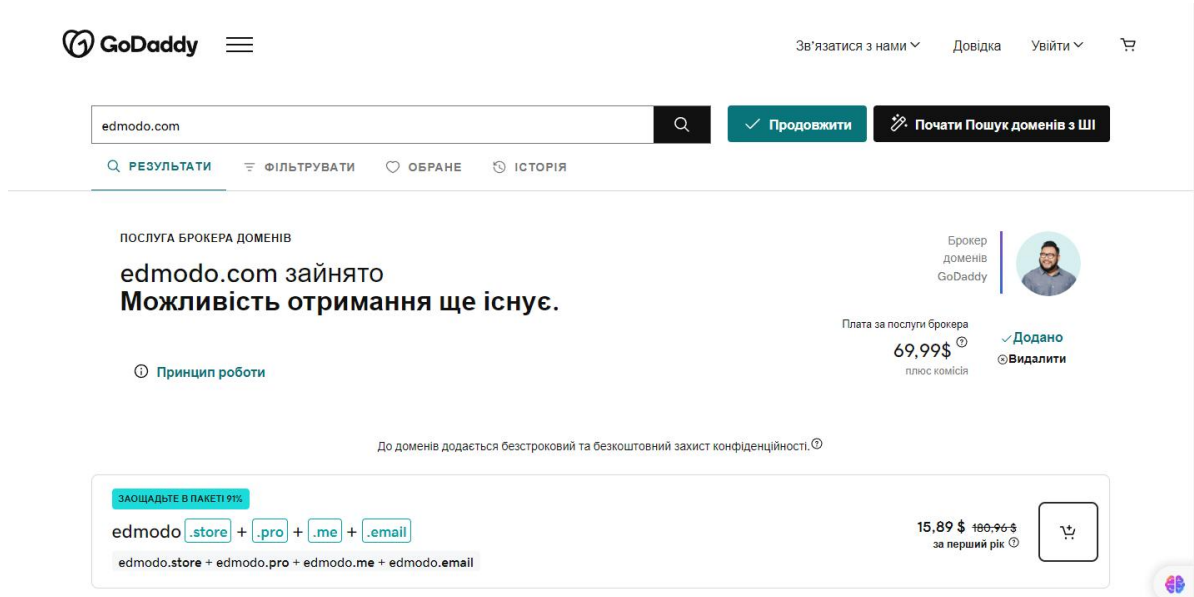


Рисунок 1.5. - Платформа Edmodo

Edmodo — освітня технологічна платформа, що пропонує комунікацію, співпрацю та можливість тренерської роботи для загальноосвітніх шкіл, коледжів та викладачів [5].

Основні можливості:

- Створення та управління курсами.
- Соціальні інструменти для спілкування і співпраці.
- Інструменти для оцінювання та зворотного зв'язку.
- Мобільний доступ.
- Інтеграція з іншими сервісами та додатками.

Переваги:

- Соціальний підхід до навчання.
- Простота використання.
- Безкоштовний базовий функціонал.

Недоліки:

- Обмежений функціонал у порівнянні з повноцінними LMS.
- Обмежена кастомізація та можливість розширення.

Доступні онлайн-платформи для навчання та підвищення кваліфікації можна розрізнити за основними характеристиками. Платформа Moodle має широкий спектр функцій і можливості кастомізації, але її використання є менш

простим. Canvas і Blackboard Learn мають потужну аналітику та простий інтерфейс, але вони не мають багато функцій кастомізації та дизайну. Хоча Google Classroom є простим у використанні та інтегрується з іншими продуктами Google, він не такий функціональний, як попередні платформи. Edmodo поєднує функції соціальних мереж і платформ для навчання, але його продуктивність нижча, ніж у інших платформах.

Таблиця 1.1

Таблиця порівняння платформ онлайн-навчання

Характеристика	Moodle	Blackboard Learn	Canvas	Google Classroom	Edmodo
Створення та управління курсами	+	+	+	+	+
Оцінювання та зворотний зв'язок	+	+	+	+	+
Інструменти для аналітики та звітності	+	+	+	-	-
Можливість кастомізації	+	-	-	-	-
Простота використання	-	+	+	+	+

Було вирішено створити нову систему автоматизації навчання в Інтернеті, враховуючи описані характеристики платформ. Нова платформа повинна мати простий у використанні дизайн, інтуїтивний дизайн, можливість інтеграції з іншими системами, потужні аналітичні інструменти, високу безпеку даних і можливість кастомізації, щоб задовольнити потреби навчального закладу.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вебдодаток буде побудований на клієнт-серверній архітектурі. Серверна частина буде представлена у вигляді API та буде написана з використанням сучасного бекенд фреймворку NestJS. Для клієнтської частини буде використано фреймворк NextJS, який побудований на основі бібліотеки ReactJS. Як сховище буде використовуватись база даних PostgreSQL.

NestJS - це фреймворк для розробки серверних застосунків на Node.js з використанням TypeScript. Він базується на концепції архітектурного шаблону Model-View-Controller (MVC) та пропонує багато інших корисних ідей та інструментів для створення високоякісних, масштабованих та легко тестованих додатків [6].

MVC — це шаблон проектування, який використовується для відокремлення інтерфейсу користувача (подання), даних (модель) і логіки програми (контролер) [7].

Переваги NestJS включають:

- TypeScript: написаний на TypeScript, що дозволяє збільшити надійність та зручність розробки за рахунок строго типізованого коду.
- Модульність: підтримує модульну архітектуру, що дозволяє легко організовувати код та зменшує його складність.
- Dependency Injection: вбудований механізм Dependency Injection, що дозволяє легко впроваджувати залежності та забезпечувати високу розширюваність додатків.
- Вбудована підтримка WebSocket: дозволяє легко розробляти реально-часні додатки.
- Легка тестованість: завдяки модульній структурі та вбудованим механізмам для тестування, робить тестування додатків легким та зручним.
- Інтеграція з іншими інструментами: може працювати з багатьма іншими інструментами, такими як Swagger, GraphQL, TypeORM та інші.

NextJS - це фреймворк для розробки вебдодатків на React. Його особливість полягає в тому, що він надає можливість рендерингу сторінок на сервері (Server-Side Rendering), а також підтримує статичний генерацію (Static Site Generation) та клієнтський рендеринг (Client-Side Rendering) [8].

Переваги NextJS включають:

- Server-Side Rendering - це процес, який передбачає відтворення сторінок на сервері, у результаті чого створюється початковий вміст HTML, який містить початковий стан сторінки [9].
- Статична генерація: дозволяє генерувати статичний контент під час збірки додатку, що збільшує швидкість завантаження сторінок та зменшує навантаження на сервер.
- Клієнтський рендеринг: підтримує клієнтський рендеринг, що дозволяє створювати динамічний контент та інтерактивні елементи на сторінках.
- Вбудована підтримка TypeScript: має вбудовану підтримку TypeScript, що збільшує надійність та зручність розробки за рахунок строго типізованого коду.
- Легка тестованість: завдяки вбудованим механізмам для тестування, робить тестування додатків легким та зручним.
- Широкі можливості розширення: підтримує багато різних плагінів та модулів, що дозволяє розширювати його функціональність та забезпечувати більшу гнучкість розробки.

HTML (HyperText Markup Language) – це стандартна мова розмітки, яка використовується для створення та структурування веб-сторінок. HTML забезпечує основу для розміщення тексту, зображень, посилань, форм та інших елементів на веб-сторінці. Кожен елемент HTML представлений у вигляді тегів, які обрамляють контент і визначають його роль на сторінці [10].

React - це бібліотека для побудови користувацьких інтерфейсів, розроблена компанією Facebook. React дозволяє розробникам створювати великі веб-додатки,

які можуть змінювати дані без перезавантаження сторінки. Його основні переваги включають:

- Компонентний підхід: дозволяє розробникам створювати багаторазові компоненти.
- Віртуальний DOM: підвищує продуктивність, оновлюючи тільки змінені частини сторінки.
- Односторонній потік даних: забезпечує більш передбачуване управління станом додатка [11].

TypeScript - це строго типізована надмова JavaScript, яка додає статичні типи. Це дозволяє виявляти помилки на ранніх етапах розробки та покращує якість коду. Основні переваги TypeScript включають:

- Статична типізація: дозволяє виявляти помилки під час компіляції.
- Покращена інтеграція IDE: пропонує кращу підтримку в редакторах коду.
- Сумісність з JavaScript: дозволяє використовувати існуючий JavaScript-код і бібліотеки [12].

Jest - це фреймворк для тестування JavaScript, розроблений Facebook. Він дозволяє писати тести для перевірки функціональності коду та підтримує багато видів тестування, включаючи модульні, інтеграційні та функціональні тести. Переваги Jest включають:

- Легкість у використанні: простий синтаксис для написання тестів.
- Швидкість: працює швидко завдяки паралельному виконанню тестів.
- Вбудована підтримка покриття коду: автоматично генерує звіти про покриття коду [13].

Prisma - це сучасний ORM для Node.js і TypeScript. Він дозволяє розробникам працювати з базами даних за допомогою об'єктно-реляційного відображення (ORM) та автоматично генерує типи TypeScript на основі схеми бази даних.

Основні переваги Prisma включають:

- Типобезпечність: генерує типи TypeScript, забезпечуючи безпеку коду.
- Простота у використанні: забезпечує простий API для роботи з базою даних.
- Підтримка міграцій: дозволяє легко керувати змінами в структурі бази даних [14].

Node.js – це серверне середовище виконання для JavaScript, яке базується на движку V8 від Google. Воно дозволяє розробникам використовувати JavaScript для написання серверного коду, що забезпечує можливість створення масштабованих мережових додатків з високою продуктивністю. Node.js відомий своєю асинхронною, подієвою архітектурою, яка дозволяє обробляти численні запити одночасно без блокування процесу [15].

JavaScript — це мова програмування, яку розробники використовують для створення інтерактивних веб-сторінок. Від оновлення каналів соціальних мереж до відображення анімацій та інтерактивних карт, функції JavaScript можуть покращити взаємодію з веб-сайтом [16].

Swagger - це набір інструментів для створення, побудови, документування та використання RESTful веб-сервісів. Він дозволяє автоматично генерувати документацію API, що полегшує процес розробки та тестування. Основні переваги Swagger включають:

- Автоматична генерація документації: дозволяє швидко створювати та оновлювати документацію API.
- Інтерактивні API-документи: надає інструменти для тестування та взаємодії з API.
- Широка підтримка мов програмування: працює з багатьма мовами програмування та фреймворками [17].

Zustand - це невеликий і швидкий бібліотека управління станом для React. Вона забезпечує простий API для створення та управління глобальним станом додатку. Основні переваги Zustand включають:

- Простота: легка у використанні та інтеграції з React.

- Мала вага: невеликий розмір бібліотеки, що не впливає на продуктивність додатку.

- Гнучкість: дозволяє використовувати середовище для різних типів додатків [18].

Chakra UI - це модульна бібліотека компонентів для React, що дозволяє швидко створювати привабливі та доступні інтерфейси користувача. Основні переваги Chakra UI включають:

- Модульність: надає набір багаторазових компонентів.
- Доступність: забезпечує високу доступність для користувачів з обмеженими можливостями.

- Темізація: дозволяє легко змінювати зовнішній вигляд компонентів за допомогою тем [19].

Formik - це бібліотека для управління формами в React. Вона дозволяє легко створювати форми, валідувати дані та обробляти події. Основні переваги Formik включають:

- Легкість у використанні: простий API для створення форм.
- Валідація: підтримка валідації даних за допомогою бібліотек, таких як Yup.

- Управління станом форм: забезпечує автоматичне управління станом форм та обробку подій [20].

PostgreSQL, часто називається просто Postgres, - це вільна та відкрита реляційна система управління базами даних (СУБД), що дозволяє зберігати та керувати великими обсягами даних. PostgreSQL підтримує стандарт SQL та включає в себе багато розширень, що забезпечують розширені функціональні можливості, такі як географічні запити, повнотекстовий пошук, JSON-операції та багато інших [21].

Основні риси PostgreSQL включають:

- Розширюваність: підтримує велику кількість розширень та модулів, що дозволяють розробникам додавати нові функції та можливості.

- Безпека: має широкий набір функцій безпеки, таких як шифрування даних, захист від вторгнень та контроль доступу.

- Надійність: високий рівень стійкості до помилок та збоїв, що дозволяє забезпечити надійну роботу бази даних.

ORM — це програмне забезпечення, призначене для допомоги розробникам ООП у взаємодії з реляційними базами даних [22].

Express - це мінімалістичний та гнучкий веб-фреймворк для додатків Node.js, що надає широкий набір функцій для мобільних та веб-додатків [23].

Visual Studio Code — це легкий, але потужний редактор вихідного коду, який працює на робочому столі та доступний для Windows, macOS і Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов і середовищ виконання [24].

Docker - це платформа, призначена для розробки, обміну та запуску контейнерних додатків [25].

React Testing Library дуже легке рішення для тестування компонентів React. Він надає легкі службові функції на додаток до react-dom та react-dom/test-utils таким чином, щоб заохочувати кращі методи тестування [26].

React Query — одна з найкращих бібліотек для керування станом сервера. Він чудово працює одразу, без конфігурації та може бути налаштований відповідно до ваших уподобань у міру розвитку програми [27].

Axios - це клієнт HTTP на основі Promise для node.js та браузера. Він ізоморфний (= він може працювати у браузері та nodejs з тією ж базою кодів). На стороні сервера він використовує рідний http-модуль node.js, тоді як на клієнті (браузер) він використовує XMLHttpRequests [28].

Multer — це проміжне програмне забезпечення node.js для обробки multipart/form-data, яке в основному використовується для завантаження файлів [29].

Висновки до першого розділу.

Проведено аналіз предметної області, за допомогою якого були визначені основні вимоги до системи автоматизації онлайн-навчання, основною задачею якої є забезпечення ефективного управління освітнім процесом, підвищення інтерактивності та доступності навчальних матеріалів, а також полегшення комунікації між студентами та викладачами.

В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані та порівняні різні платформи для онлайн-навчання. Було визначено їхній стандартний функціонал, переваги та недоліки. На основі цього аналізу були сформовані основні вимоги до власної системи автоматизації онлайн-навчання.

На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки вебдодатку. Було обрано такі інструменти, як NextJS для клієнтської частини, NestJS для серверної частини, PostgreSQL для зберігання даних, а також інші технології. Ці технології забезпечують високу продуктивність, масштабованість, надійність та зручність у розробці та підтримці системи автоматизації онлайн-навчання.

Таким чином, проведений аналіз та визначення інструментальних засобів створили міцну основу для подальшої розробки системи, що відповідатиме сучасним вимогам та очікуванням користувачів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН-НАВЧАННЯ

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

Система автоматизації онлайн-навчання надає веб-інтерфейс, через який користувачі можуть отримувати доступ до навчальних матеріалів, проходити курси, взаємодіяти з викладачами та іншими студентами. Функціональність системи залежить від активності користувачів. Користувачі можуть бути розділені на дві основні групи: студенти, які проходять навчальні курси, та викладачі, які створюють та проводять ці курси. Курси містять деталі про навчальні матеріали, завдання, тести та інші важливі відомості, які можуть зацікавити студентів.

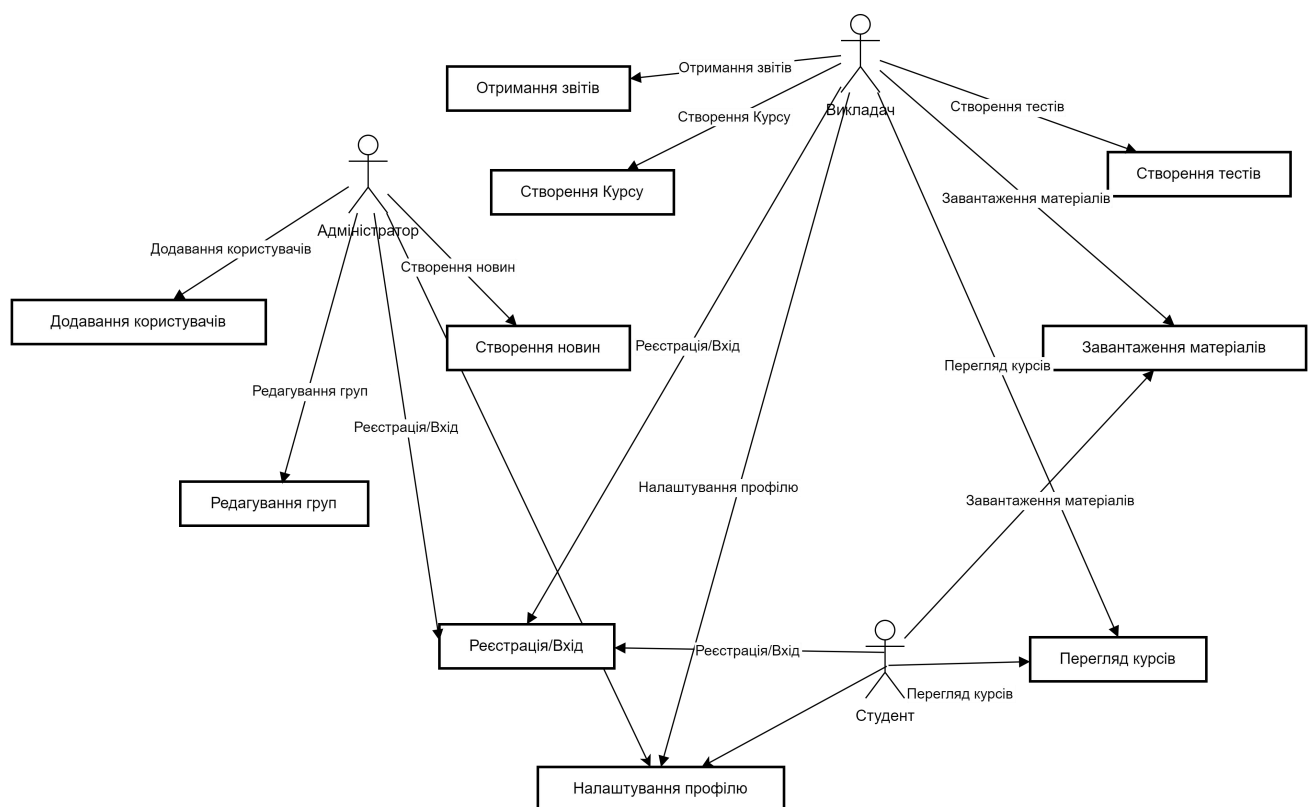


Рисунок 2.1. - Діаграма варіантів використання

Вплив онлайн-навчання на освіту є багатограним і дало як позитивні, так і трансформаційні результати. По-перше, онлайн-навчання значно розширило

доступ до освіти, руйнуючи географічні бар'єри та дозволяючи людям з різних верств суспільства продовжувати можливості навчання. Це призвело до більшої інклюзивності та рівності в освіті, що дозволило більшій кількості людей отримати доступ до знань та навичок, які раніше були недоступними [30].

Для наглядної взаємодії між елементами додатку було створено діаграму варіантів використання, зображено на рисунку 2.1.

Опис діаграми варіантів використання системи :

1. Реєстрація/Вхід (див. табл. 2.1)
2. Додавання користувачів (див. табл. 2.2)
3. Створення новин (див. табл. 2.3)
4. Редагування груп (див. табл. 2.4)
5. Створення Курсу (див. табл. 2.5)
6. Налаштування профілю (див. табл. 2.6)
7. Отримання звітів (див. табл. 2.7)
8. Перегляд курсів (див. табл. 2.8)
9. Завантаження матеріалів (див. табл. 2.9)

Таблиця 2.1

Опис прецеденту “Реєстрація/Вхід”

Назва варіанту використання	Основні актори	Короткий опис
Реєстрація/Вхід	Адміністратор, Викладач, Студент	Користувач може зареєструватися або ввійти в систему, використовуючи свої облікові дані.

Таблиця 2.2

Опис прецеденту “Додавання користувачів”

Назва варіанту використання	Основні актори	Короткий опис
Додавання користувачів	Адміністратор	Адміністратор може додавати нових користувачів до системи.

Таблиця 2.3

Опис прецеденту “Створення новин”

Назва варіанту використання	Основні актори	Короткий опис
Створення новин	Адміністратор	Адміністратор може створювати новини для користувачів системи.

Таблиця 2.4

Опис прецеденту “Редагування груп”

Назва варіанту використання	Основні актори	Короткий опис
Редагування груп	Адміністратор	Адміністратор може редагувати інформацію про групи користувачів.

Таблиця 2.5

Опис прецеденту “Створення Курсу”

Назва варіанту використання	Основні актори	Короткий опис
Створення Курсу	Викладач	Викладач може створювати нові курси.

Таблиця 2.6

Опис прецеденту “Отримання звітів”

Назва варіанту використання	Основні актори	Короткий опис
Отримання звітів	Викладач	Викладач може отримувати звіти про успішність студентів.

Таблиця 2.7

Опис прецеденту “Налаштування профілю”

Назва варіанту використання	Основні актори	Короткий опис
Налаштування профілю	Викладач	Викладач може налаштовувати свій профіль.

Таблиця 2.8

Опис прецеденту “Перегляд курсів”

Назва варіанту використання	Основні актори	Короткий опис
Перегляд курсів	Викладач, Студент	Користувач може переглядати доступні курси.

Таблиця 2.9

Опис прецеденту “Завантаження матеріалів”

Назва варіанту використання	Основні актори	Короткий опис
Перегляд курсів	Викладач, Студент	Користувач може переглядати доступні курси.

Основна мета розробки системи автоматизації онлайн-навчання полягає в створенні онлайн-середовища, яке дозволяє студентам ефективно проходити навчальні курси, а викладачам – організовувати та проводити ці курси. Платформа спрямована на забезпечення ефективної комунікації та взаємодії між викладачами та студентами, а також на надання зручного і надійного механізму для управління навчальним процесом.

Вимоги користувачів:

1. Внутрішній користувач – адміністратор:

1.1. Спілкування з викладачами та студентами.

- 1.2. Підтвердження створення курсів.
- 1.3. Перегляд заявок на створення нових курсів.
2. Внутрішній користувач – викладач:
 - 2.1. Авторизація.
 - 2.2. Перегляд та управління власними курсами.
 - 2.3. Перегляд статусу створених курсів.
 - 2.4. Створення нових курсів.
 - 2.5. Перегляд списку курсів.
 - 2.6. Перегляд статистики успішності студентів.
3. Внутрішній користувач – студент:
 - 3.1. Авторизація.
 - 3.2. Перегляд та проходження курсів.
 - 3.3. Виконання завдань та складання тестів.
 - 3.4. Перегляд власних результатів та успішності.

Функціональні вимоги:

1. В системі повинна бути представлена можливість авторизації користувачів.
2. Забезпечення анонімності звичайним користувачам.
3. Забезпечення спілкування користувачів.
4. Відображення результатів навчання.
5. Можливість збереження інформації про курси та успішність студентів.
6. Забезпечити викладачам переглядати статистику успішності студентів.

Не функціональні вимоги:

1. Сприйняття:
 - 1.1. Час, потрібний для освоєння вебсайту – від 3-х до 5-ти хвилин, для досвідчених менше 2-х хвилин.
 - 1.2. Час відповіді вебсайту на дії користувача не повинні перевищувати 3 секунд.

На рисунку 2.2 наведено діаграму класів, яка використовуються для його роботи.

User – клас, який відповідає за користувача:

- register() – відповідає за реєстрацію;
- login() – відповідає за авторизацію.

Administrator – клас, який відповідає за адміністратора:

- addUser() – відповідає за додавання користувача;
- editGroup() – відповідає за редагування групи;
- createNews() – відповідає за створення новин;
- viewReport() – відповідає за перегляд звітів.

Instructor – клас, який відповідає за викладача:

- createCourse() – відповідає за створення курсу;
- uploadMaterial() – відповідає за завантаження матеріалу;
- createTest() – відповідає за створення тесту;
- viewReport() – відповідає за перегляд звітів.

Student – клас, який відповідає за студента:

- viewCourses() – відповідає за перегляд курсів;
- downloadMaterial() – відповідає за завантаження матеріалу.

Course – клас, який відповідає за курси:

- addMaterial() – відповідає за додавання матеріалу;
- addTest() – відповідає за додавання тесту.

Material – клас, який відповідає за матеріали.

Test – клас, який відповідає за тести.

Group – клас, який відповідає за групи:

- editGroup() – відповідає за редагування групи.

News – клас, який відповідає за новини:

- createNews() – відповідає за створення новин.

Report – клас, який відповідає за звіти:

- viewReport() – відповідає за перегляд звітів.

2.2 Розробка бази даних системи

Для зберігання файлів та інформації в додатку використовується PostgreSQL, система управління базами даних, яка заснована на реляційній моделі. Крім того, після публікації завдань, вони зберігаються в блокчейні, де також зберігаються всі транзакції.

База даних системи складається з кількох таблиць:

1. Users – таблиця, яка містить інформацію про користувачів;
2. Educational Institutions – таблиця, що зберігає дані про освітні установи;
3. Groups – таблиця, яка містить інформацію про навчальні групи;
4. Academic Disciplines – таблиця, що зберігає дані про академічні дисципліни;
5. Tasks – таблиця, яка містить інформацію про завдання.

Таблиця 2.10

Опис таблиці“Users”

Назва поля	Тип	Призначення
id	Int	Ідентифікатор користувача
email	String	Email користувача
password	String	Пароль користувача
firstName	String	Ім'я користувача
lastName	String	Прізвище користувача
lastLoginAt	Date	Дата останнього входу
role	Enum	Роль користувача (наприклад, Admin, Student, Teacher)
isVerifiedEmail	Boolean	Статус верифікації email
images	FileUser[]	Файли, пов'язані з користувачем
studentProfile	StudentProfile?	Профіль студента (якщо є)
teacherProfile	TeacherProfile?	Профіль викладача (якщо є)
adminProfile	AdminProfile?	Профіль адміністратора (якщо є)
institutions	UserInstitution[]	Пов'язані установи

Таблиця 2.11

Опис таблиці“Educational Institutions”

Назва поля	Тип	Призначення
id	Int	Ідентифікатор установи
logo	FileEdu[]	Логотипи установи
name	String	Назва установи
groups	Group[]	Пов'язані групи
users	UserInstitution[]	Пов'язані користувачі
news	News[]	Новини установи

Таблиця 2.12

Опис таблиці“Groups”

Назва поля	Тип	Призначення
id	Int	Ідентифікатор групи
name	String	Назва групи
curatorId	Int	Ідентифікатор куратора (викладача)
students	StudentGroup[]	Пов'язані студенти
lessons	Lesson[]	Пов'язані заняття
educationalInstitution	EducationalInstitution @relation(fields: [educationalInstitutionId], references: [id], onDelete: Cascade)	Пов'язана установа

Таблиця 2.13

Опис таблиці“Academic Disciplines”

Назва поля	Тип	Призначення
id	Int	Ідентифікатор дисципліни
image	FileDiscipline[]	Файли зображень дисципліни
color	String	Колір дисципліни (за замовчуванням: "білий")
name	String	Назва дисципліни

teachers	TeacherDiscipline[]	Пов'язані викладачі
students	StudentDiscipline[]	Пов'язані студенти
lessons	Lesson[]	Пов'язані заняття
modules	Module[]	Пов'язані модулі
details	String?	Деталі дисципліни
totalScore	String	Загальний бал (за замовчуванням: "середній")

Таблиця 2.14

Опис таблиці“Tasks”

Назва поля	Тип	Призначення
id	Int	Ідентифікатор завдання
name	String	Назва завдання
type	String	Тип завдання (за замовчуванням: "task")
test	Json?	Дані тесту
createDate	DateTime	Дата створення завдання
module	TaskModule[]	Пов'язані модулі
mark	Int	Бал завдання (за замовчуванням: 0)
isReview	Boolean	Статус рецензії (за замовчуванням: false)
maxMark	Int	Максимальний бал
files	FileTask[]	Пов'язані файли
students	StudentTask[]	Пов'язані студенти

2.3 Проектування та реалізація алгоритмів роботи системи

Система є зрозумілою для використання, забезпечує та виконує основні функції, які дають можливість користувачам легко працювати. Вона пропонує інтуїтивно зрозумілий інтерфейс, що дозволяє новим користувачам швидко орієнтуватися в процесах реєстрації та управління курсами.

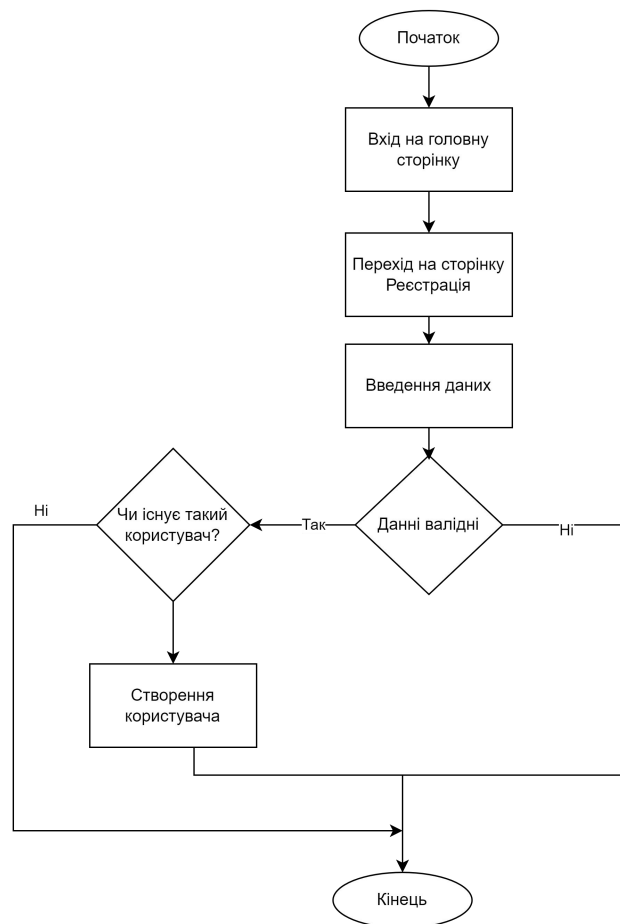


Рисунок 2.3. - Алгоритм реєстрації

Алгоритм реєстрації користувачів детально зображено на рисунку 2.3, де показані всі кроки від входу на головну сторінку до успішного створення нового користувача. Користувачі вводять свої дані, які перевіряються на валідність, а також перевіряється, чи існує вже такий користувач в системі. У разі відсутності помилок та існування користувача система створює новий обліковий запис.

Алгоритм управління курсами представлено на рисунку 2.4. Він ілюструє процес створення та виконання завдань викладачами і студентами. Викладач заходить на сторінку курсів, створює нове завдання та додає до нього студентів. Після цього завдання стає доступним для студентів, які можуть його виконати. Викладач потім перевіряє виконану роботу, повертає її для виправлень або оцінює. У разі валідних даних викладач виставляє оцінку, яку студент може переглянути. Такий детально прописаний алгоритм забезпечує чіткість та прозорість процесів навчання, полегшуючи роботу як викладачам, так і студентам.

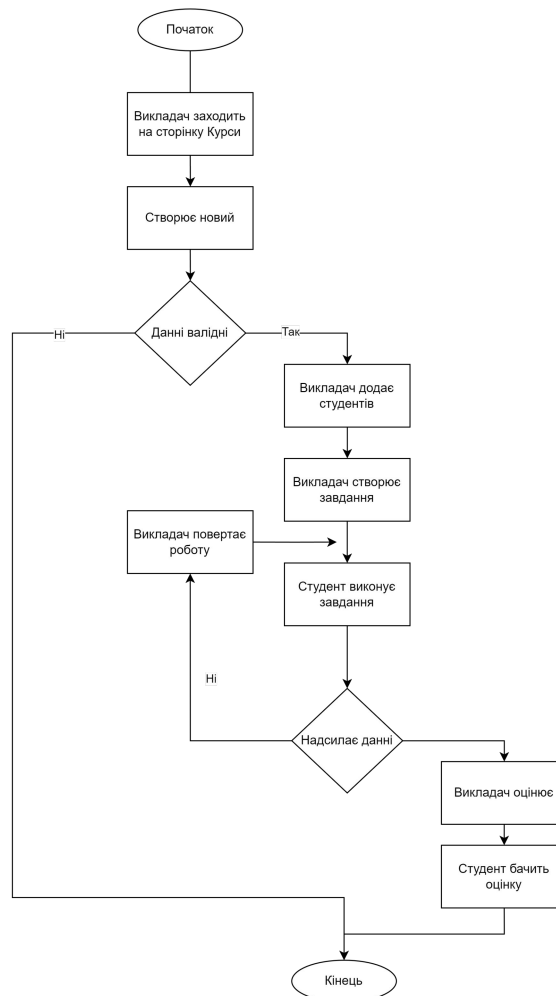


Рисунок 2.4. - Алгоритм управління курсами

2.4 Реалізація функціоналу системи автоматизації онлайн-навчання

Розглянемо основні функції для роботи платформи. Для цього користувачу необхідно зареєструватися.

```

import { Injectable } from '@nestjs/common';
import { randomBytes, pbkdf2Sync } from 'crypto';

Codeium: Refactor | Explain | Codiumate: Options | Test this class
@Injectable()
export class PasswordService {
  Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Options | Test this method
  getSalt() {
    return randomBytes(16).toString('hex');
  }
  Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Options | Test this method
  getHash(password: string, salt: string) {
    return pbkdf2Sync(password, salt, 1000, 64, 'sha512').toString('hex');
  }
}

```

Рисунок 2.5. - Hash функції

Для забезпечення захисту даних було створено функції для хешування даних, яка представлена на рисунку 2.5.

```
import { Injectable } from '@nestjs/common';
import { Response } from 'express';

Codeium: Refactor | Explain | Codiumate: Options | Test this class
@Injectable()
export class CookieService {
  static tokenKey = 'access-token';
  Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Op
  setToken(res: Response, token: string) {
    res.cookie(CookieService.tokenKey, token, {
      httpOnly: true,
      maxAge: 24 * 60 * 60 * 1000,
    });
  }
  Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Op
  removeToken(res: Response) {
    res.clearCookie(CookieService.tokenKey);
  }
}
```

Рисунок 2.6. - Token функції

В кінці авторизації генерується унікальний access-token, функцію що реалізую це представлено на рисунку 2.6.

```
async sendEmail(dto: CreateMailerDto) {
  const { from, recipients, subject, html } = dto;
  const transport = this.mailTransport();
  const options: Mail.Options = {
    from: from ?? {
      name: 'LMS',
      address: 'ipzk221_kvo@student.ztu.edu.ua',
    },
    to: recipients,
    subject,
    html,
  };
  try {
    const result = await transport.sendMail(options);
    return result;
  } catch (err) {
    throw Error('error send message');
  }
}
```

Рисунок 2.7. - Email функція

В процесі реєстрації користувач отримує email для верифікації пошти, функція представлена на рисунку 2.7.

```
import { useEffect, useState } from "react";
import { getFile } from "../api/files";

Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Options
export const useFileBlob = () => {
    const [file, setFile] = useState<Blob>();
    const [src, setSrc] = useState<string>("");
    Codeium: Refactor | Explain | Generate JSDoc | X
    const get = async (id: number) => {
        getFile(id).then((data) => setFile(data));
    };
    useEffect(() => {
        file && setSrc(URL.createObjectURL(file!));
    }, [file]);

    return { src, get, setFile };
}
```

Рисунок 2.8. - Hook useFileBlob

В роботі додатку використовуються hook useFileBlob для трансформація файлів отриманих з серверу, функція представлена на рисунку 2.8.

```
export function useSession() {
    const router = useRouter();
    const { initUser } = useUserStore();
    const { isLoading, isError, data } = useQuery("session", getSession);

    useEffect(() => {
        if (data) {
            initUser(data);
            console.log(data);
        }
    }, [data, initUser]);

    useEffect(() => {
        if (isError) {
            router.replace("/login");
        }
    }, [isError, router]);

    return { isError, isLoading };
}

Codeium: Refactor | Explain | Generate JSDoc | X | Codiumate: Options | Test this function
export function useResetSession() {
    const queryClient = useQueryClient();
    return () => queryClient.removeQueries();
}
```

Рисунок 2.9. - Hook useSession

Для реалізації сесій на клієнті створено hook useSession, який перевіряє token та реалізує перенесення на сторінку авторизації в разі помилки, функція представлена на рисунку 2.9.

```
import { diskStorage } from 'multer';
import { v4 as uuidv4 } from 'uuid';
import { extname } from 'path';

export const multerOptions = {
  storage: diskStorage({
    destination: __dirname + '/files/uploads/',
    Codeium: Refactor | Explain | Generate JSDoc | X
    filename: (req, file, cb) => {
      const uniqueSuffix = `${uuidv4()}${extname(file.originalname)}`;
      cb(null, uniqueSuffix);
    },
  }),
};
```

Рисунок 2.10. - Multer Options

Створено конфігурацію multer для зберігання файлів та створення унікальних імен, функція представлена на рисунку 2.10.

```
async singUp(authData: AuthUserBody, isVerified?: boolean, role?: string) {
  const user = await this.db.user.findFirst({ ...
});
  if (user) {
    throw new BadRequestException('User already exists');
  }
  const salt = this.passwordService.getSalt();
  const hash = this.passwordService.getHash(authData.password, salt);
  const newUserData = {
    firstName: authData.firstName,
    middleName: authData.middleName,
    lastName: authData.lastName,
    email: authData.email,
    hash,
    salt,
    role,
  };
  const newUser = await this.create(
    newUserData,
    +authData.educationalInstitution,
    isVerified || false,
  );
  console.log(authData.password);
  await this.mailer.sendEmail({ ...
});

  return {
    id: newUser.id,
    email: newUser.email,
    educationalInstitution: +authData.educationalInstitution,
  };
}
```

Рисунок 2.11. - Функція Реєстрації

Реєстрація виконує перевірку даних, наявність існуючих користувачів та використовує функції хешування, функція представлена на рисунку 2.11.

```

async signIn({ email, password, educationalInstitution }: SignInData) {
  const user = await this.db.user.findFirst({ ...
});

  if (!user) {
    throw new NotFoundException('User not found');
  }
  const hash = this.passwordService.getHash(password, user.salt);
  if (hash !== user.hash) {
    throw new UnauthorizedException('Password wrong');
  }
  console.log(user);
  const isVerifiedUser = await this.db.userInstitution.findFirst({ ...
});
  if (!isVerifiedUser) {
    throw new UnauthorizedException('Your request was denied ');
  }
  if (!isVerifiedUser.isVerified) {
    throw new UnauthorizedException(
      'You have not yet been added to the educational institution',
    );
  }
  if (user.role === 'admin') { ...
  }
  const accessToken = await this.jwtService.signAsync({
    id: user.id,
    email: user.email,
    role: user.role,
    educationalInstitution: isVerifiedUser.educationalInstitutionId,
  });
  return { accessToken };
}

```

Рисунок 2.12. - Функція Авторизації

Авторизація реалізує пошук користувачів, порівняння паролів, перевірка підтвердження запрошення до закладу освіти, , функція представлена на рисунку 2.12.

Висновки до другого розділу

В процесі проектування системи автоматизації онлайн-навчання були проаналізовані всі варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована структура системи. Проектування включало в себе моделювання структури проекту та взаємодії між його складовими частинами.

Також було спроектовано базу даних для зберігання значень, таких як результати тестів, навчальні матеріали, дані про користувачів (студентів, викладачів) та інші важливі дані, що будуть зберігатися на серверній частині додатку.

Був описаний алгоритм роботи з додатком та описані послідовні пункти варіантів його використання.

Основний функціонал додатку включає в себе створення курсів викладачами, реєстрацію студентів на курси, проведення онлайн-тестувань та зберігання результатів, а також інтеграцію з системами для відеоконференцій. Також були розглянуті основні функції по взаємодії користувачів зі системою, такі як перегляд навчальних матеріалів, участь у форумах та обговореннях, подання домашніх завдань та отримання зворотного зв'язку від викладачів.

Ці кроки забезпечують повний цикл навчального процесу від створення навчального контенту до оцінювання знань студентів, що дозволяє зробити процес навчання більш ефективним та зручним для всіх учасників.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ АВТОМАТИЗАЦІЇ ОНЛАЙН-НАВЧАННЯ

3.1 Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування платформи включає розгортання всіх її компонентів та конфігурацію кожного елемента окремо, а також налаштування загальної системи в цілому.

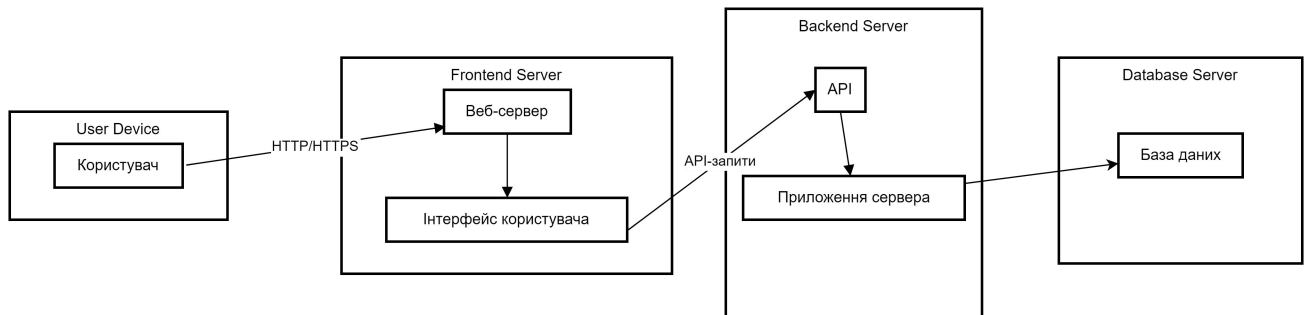


Рисунок 3.1. - Діаграма розгортання платформи

На рисунку 3.1 представлена схема архітектури розгортання платформи, яка складається з наступних компонентів:

User Device (Пристрій користувача):

- Це пристрій, який використовується кінцевим користувачем для доступу до платформи, наприклад, комп'ютер, планшет або смартфон.
- Користувач взаємодіє з платформою через веб-браузер або мобільний додаток.

Frontend Server (Фронтенд сервер):

- Веб-сервер: Приймає HTTP/HTTPS запити від пристрою користувача.
- Інтерфейс користувача: Відповідає за відображення інформації та взаємодію з користувачем через веб-браузер або мобільний додаток.
- Веб-сервер відправляє дані, отримані від користувача, до бекенд-сервера через API-запити.

Backend Server (Бекенд сервер):

- API: Приймає запити від фронтенд-сервера і обробляє їх.

- Прикладення сервера: Виконує основну логіку бізнес-процесів, обробляє дані і виконує взаємодію з базою даних.

- API-запити від фронтенд-сервера обробляються бекенд-сервером, який може виконувати різні операції, такі як автентифікація користувача, обробка даних та інші бізнес-логічні задачі.

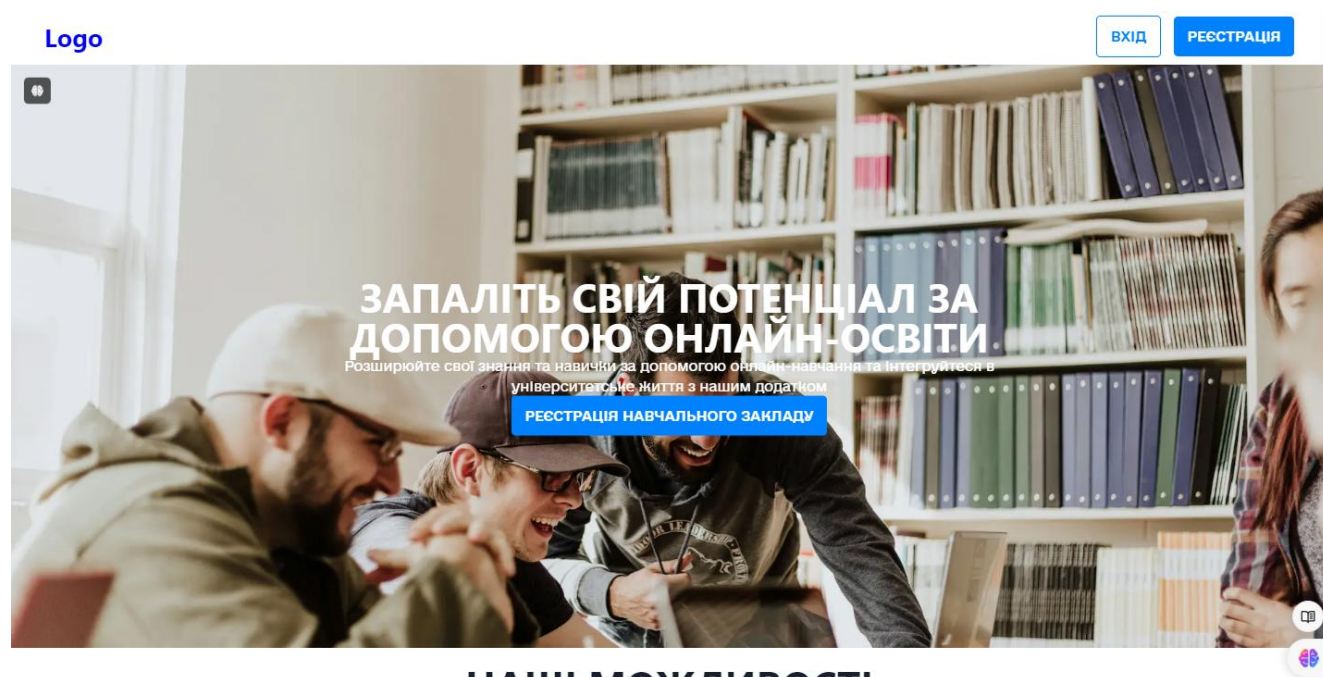
Database Server (Сервер баз даних):

- База даних: Зберігає всю необхідну інформацію для роботи платформи, таку як дані користувачів, контент, налаштування тощо.

- Бекенд-сервер взаємодіє з базою даних для збереження і отримання даних.

3.2 Структура інтерфейсу системи автоматизації онлайн-навчання.

Інтерфейс системи автоматизації онлайн-навчання розроблений таким чином, щоб забезпечити максимально зручний та інтуїтивно зрозумілий доступ до основних функцій як для викладачів, так і для студентів. Основною метою є спрощення процесу взаємодії з системою, що включає реєстрацію, авторизацію, управління навчальними матеріалами та взаємодію між учасниками навчального процесу.



НАШІ МОЖЛИВОСТІ

Рисунок 3.2. - Головна сторінка

На головній сторінці додатку, яку зображено на рисунку 3.2, представлено зображення, яке включає заголовок із закликом до дії, спрямованим на залучення користувачів до онлайн-освіти. У центральній частині сторінки розташоване велике зображення групи людей, що працюють разом, а під ним розміщено залик до реєстрації навчального закладу. У верхньому правому куті сторінки знаходяться кнопки "Вхід" та "Реєстрація", що дозволяють користувачам швидко перейти до відповідних розділів.

Рисунок 3.3. - Форма авторизація

Форма авторизації, яка зображена на рисунку 3.3, включає поля для введення електронної пошти, пароля та вибору навчального закладу з випадального списку. Користувач може ввести свої дані та натиснути кнопку "LOGIN", щоб увійти до системи. Для тих, хто ще не має облікового запису, під формою розміщено посилання на сторінку реєстрації.

Рисунок 3.4. - Форма Реєстрації

Форма реєстрації нового користувача, що представлена на рисунку 3.4, містить поля для введення прізвища, імені, по батькові, електронної пошти, пароля та вибору навчального закладу з випадającego списку. Після заповнення всіх необхідних даних користувач може натиснути кнопку "REGISTRATION" для завершення реєстрації. Для тих, хто вже має обліковий запис, під формою розміщено посилання на сторінку авторизації.

Рисунок 3.5. - Сторінка навчального закладу

Сторінка навчального закладу, що представлена на рисунку 3.5, надає доступ до новин та матеріалів, що стосуються конкретного навчального закладу. Вона також включає функціонал для створення нових інформаційних матеріалів, що дозволяє викладачам оперативно ділитися важливими новинами та оновленнями зі студентами.

Рисунок 3.6. - Форма створення новин

Форма створення новин, яка зображена на рисунку 3.6, і форма завантаження файлів є ключовими інструментами для додавання та управління контентом на платформі. Вони розроблені з урахуванням потреб користувачів, забезпечуючи простоту та зручність у використанні.

Logo				
NAME	EMAIL	ROLE	ACTIONS	
Віктор Олександрович Кучер	ipzk222_kvo_viktor@student.ztu.edu.ua	Select role ▾	Save	Delete
Віктор Олександрович Кучер	ipzk224_kvo_viktor@student.ztu.edu.ua	Select role ▾	Save	Delete
Віктор Олександрович Кучер	ipzk225_kvo_viktor@student.ztu.edu.ua	Select role ▾	Save	Delete

Рисунок 3.7. - Сторінка керування користувачами

Сторінка керування користувачами, яка зображена на рисунку 3.7, надає адміністратору можливість переглядати список користувачів, змінювати їх ролі та видаляти облікові записи. Інтерфейс включає поля для відображення імені користувача, його електронної пошти та випадаючий список для вибору ролі (Admin, Student, Teacher). Після вибору ролі користувач може натиснути кнопку "Save" для збереження змін або "Delete" для видалення облікового запису.

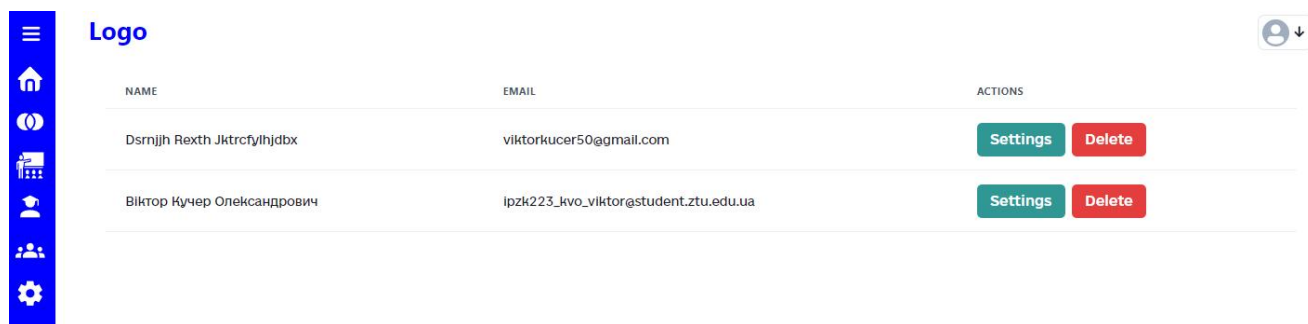


Рисунок 3.8. - Сторінка керування студентами

Сторінка керування студентами, яка зображена на рисунку 3.8, надає адміністратору можливість переглядати список зареєстрованих студентів, редагувати їхні дані та видаляти облікові записи. Вона складається з таблиці, яка містить такі стовпці: "Name" (Ім'я), "Email" (Електронна пошта) та "Actions" (Дії).

У стовпці "Name" відображаються імена студентів. Це дозволяє адміністратору легко ідентифікувати кожного студента за його іменем. У стовпці "Email" вказується електронна адреса студента, що також сприяє ідентифікації та забезпечує можливість зв'язку зі студентом через електронну пошту.

Стовпець "Actions" містить дві кнопки для кожного студента: "Settings" (Налаштування) та "Delete" (Видалити). Кнопка "Settings" дозволяє адміністратору редагувати інформацію про студента. При натисканні цієї кнопки відкривається форма налаштувань профілю, де можна змінити такі дані, як ім'я, по батькові, прізвище та групу, до якої належить студент. Кнопка "Delete" призначена для видалення облікового запису студента з системи. Після натискання цієї кнопки обліковий запис студента видаляється, і він більше не має доступу до системи.

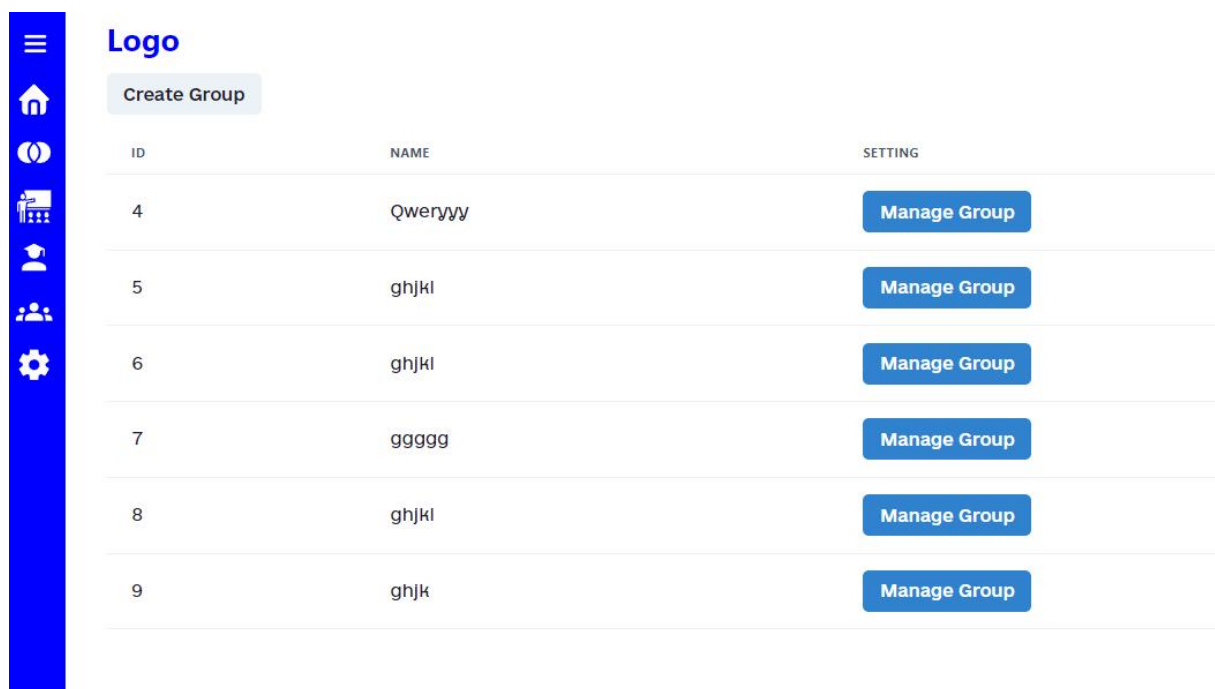


Рисунок 3.9. - Сторінка керування групами

Сторінка керування групами, яка зображена на рисунку 3.9, надає можливість адміністратору створювати та редагувати групи. Кожна група має унікальний ідентифікатор, ім'я та опцію для керування групою. Форма керування групою містить поля для введення назви групи, вибору куратора та студентів, а також кнопку "Save" для збереження змін або "Cancel" для скасування дії.

Manage Group

Group Name

Qwertyyy

Curator

Віктор Кучер Олександрович

Students

Select students

Cancel

Save

Рисунок 3.10. - Форма керування групою

Форма керування групою, яка зображена на рисунку 3.10, надає адміністратору можливість створювати та редагувати групи студентів. Вона

складається з декількох полів, які дозволяють налаштувати основні параметри групи.

Поле "Group Name" (Назва групи) призначене для введення назви групи. Адміністратор може задати унікальну назву для кожної групи, що допомагає легко ідентифікувати її серед інших.

Поле "Curator" (Куратор) дозволяє обрати куратора групи зі списку доступних викладачів. Це забезпечує призначення відповідальної особи за дану групу студентів.

Поле "Students" (Студенти) містить випадаючий список, який дозволяє обрати студентів для додавання до групи. Адміністратор може вибрати кілька студентів, які будуть належати до цієї групи.

Кнопка "Save" (Зберегти) призначена для збереження внесених змін. Після натискання цієї кнопки нова група або зміни до існуючої групи будуть збережені в системі.

Кнопка "Cancel" (Скасувати) дозволяє скасувати внесені зміни та повернутися до попереднього стану без збереження нових налаштувань.

Ця форма забезпечує зручний та інтуїтивний інтерфейс для адміністрування груп, дозволяючи ефективно управляти студентами та призначати відповідальних кураторів.

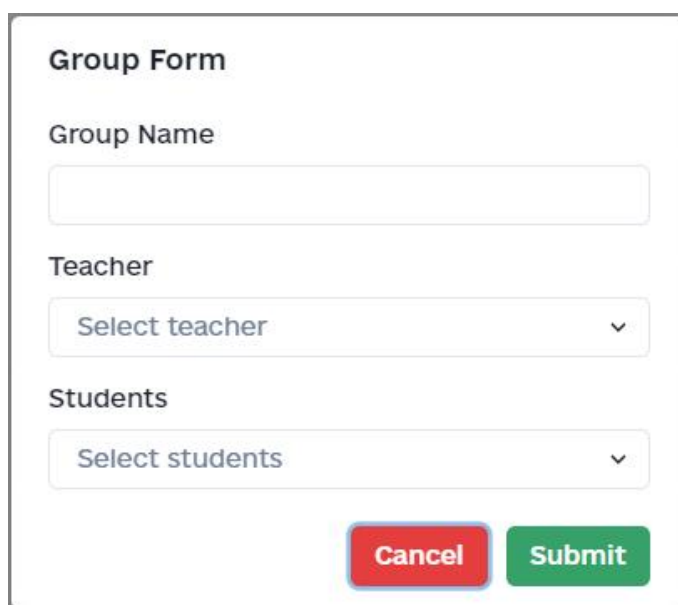


Рисунок 3.11. - Форма створення групи

Форма створення групи надає адміністратору можливість створювати нові групи студентів, налаштовуючи основні параметри групи. Вона складається з декількох полів, що дозволяють ввести необхідну інформацію, зображено на рисунку 3.11.

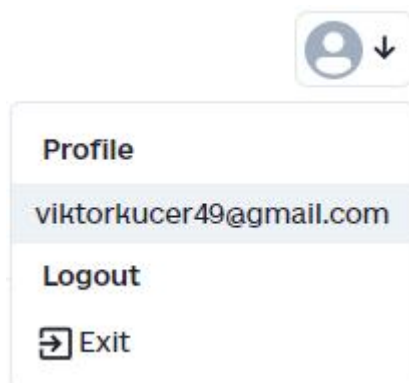


Рисунок 3.12. - Меню профілю користувача

Меню профілю користувача включає опції для переходу до профілю, виходу з системи та відображення електронної пошти користувача. Це забезпечує зручний доступ до особистих налаштувань та можливість швидко вийти з системи, зображено на рисунку 3.12.

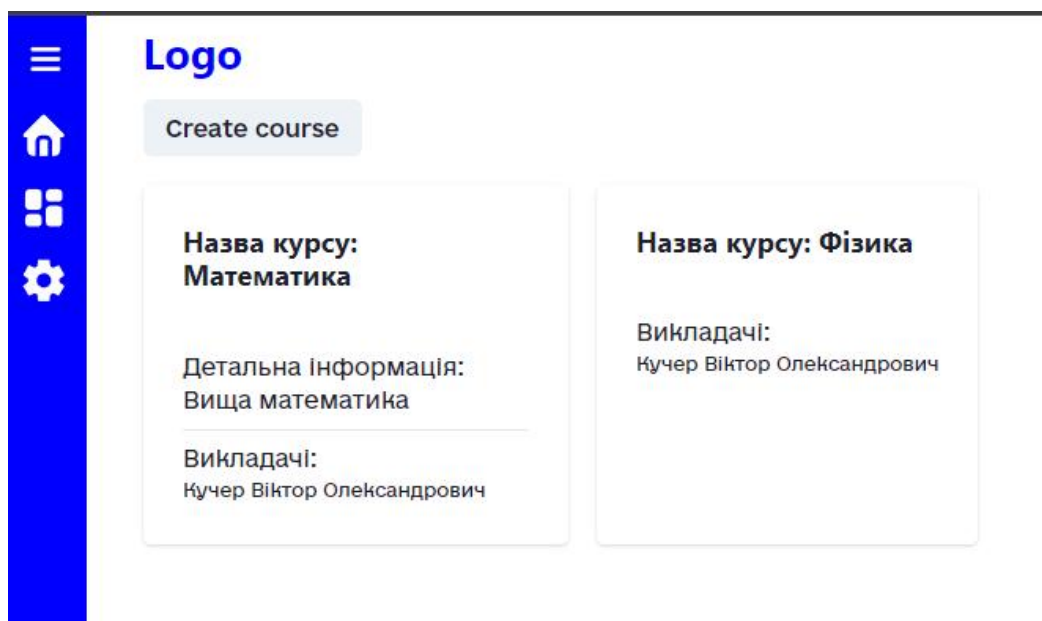


Рисунок 3.13. - Сторінка управління курсами

Сторінка управління курсами, зображено на рисунку 3.13, надає адміністраторам та викладачам можливість переглядати інформацію про доступні курси, створювати нові курси та керувати існуючими. Інтерфейс цієї сторінки

складається з декількох ключових елементів, які забезпечують ефективну роботу з курсами.

У верхній частині сторінки розташований логотип системи та кнопка "Create course" (Створити курс). Натискання цієї кнопки відкриває форму для створення нового курсу, де адміністратор або викладач можуть ввести назву курсу, додати детальну інформацію та призначити викладачів.

Центральна частина сторінки містить інформаційні картки, кожна з яких представляє окремий курс. На картках відображаються наступні дані:

Назва курсу: Назва кожного курсу, що допомагає ідентифікувати його серед інших.

Детальна інформація: Короткий опис або детальна інформація про курс, що надає додаткову інформацію про його зміст.

Викладачі: Список викладачів, які відповідають за проведення даного курсу.

Ліва частина сторінки містить вертикальне меню навігації, яке забезпечує швидкий доступ до різних розділів системи. Меню включає іконки для переходу до головної сторінки, сторінки курсів, сторінки управління користувачами, сторінки груп та налаштувань системи.

Ця сторінка надає інтуїтивно зрозумілий інтерфейс для управління курсами, дозволяючи адміністраторам та викладачам ефективно створювати, редагувати та переглядати курси, що сприяє покращенню організації навчального процесу.

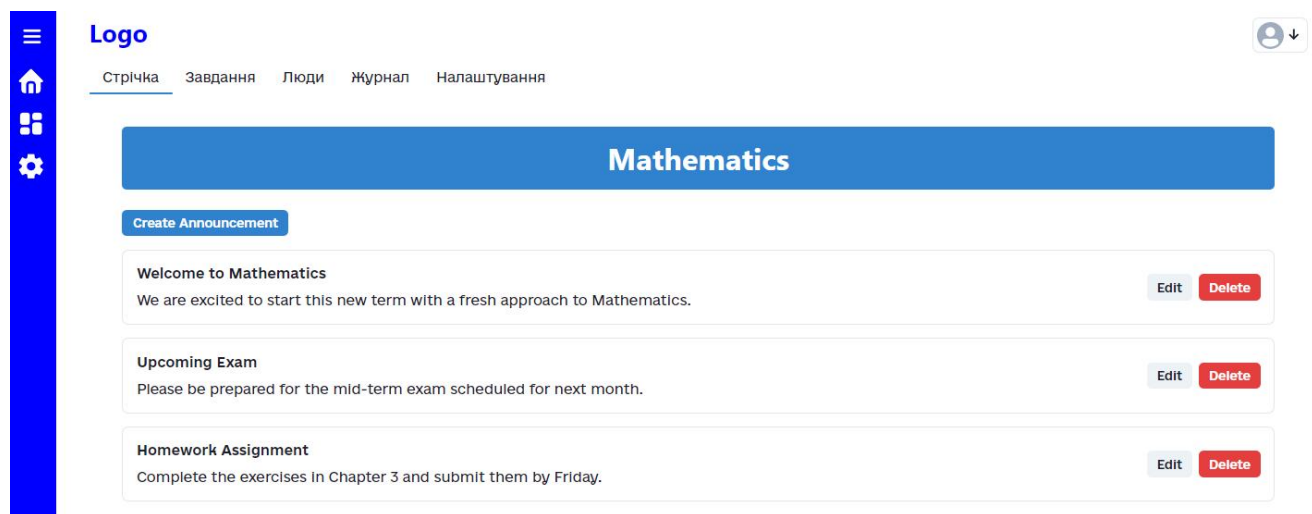


Рисунок 3.14. - Сторінка курсу

Сторінка курсу, зображено на рисунку 3.14, надає викладачам і студентам доступ до оголошень, завдань, списку людей, журналу та налаштувань курсу. Інтерфейс цієї сторінки складається з декількох основних елементів, що забезпечують зручність у використанні та ефективне управління курсом.

У верхній частині сторінки розташований логотип системи та горизонтальне меню навігації, яке включає такі розділи, як "Сторінка", "Завдання", "Люди", "Журнал" та "Налаштування". Це меню дозволяє користувачам швидко переходити між різними аспектами курсу.

Центральна частина сторінки відображає назву курсу у вигляді великого заголовка на синьому фоні. Під заголовком розташована кнопка "Create Announcement" (Створити оголошення), яка дозволяє викладачам додавати нові оголошення для студентів.

Нижче знаходиться список існуючих оголошень, кожне з яких включає заголовок та текст повідомлення. Для кожного оголошення передбачено дві кнопки: "Edit" (Редагувати) та "Delete" (Видалити). Кнопка "Edit" дозволяє викладачу редагувати текст оголошення, а кнопка "Delete" видаляти його.

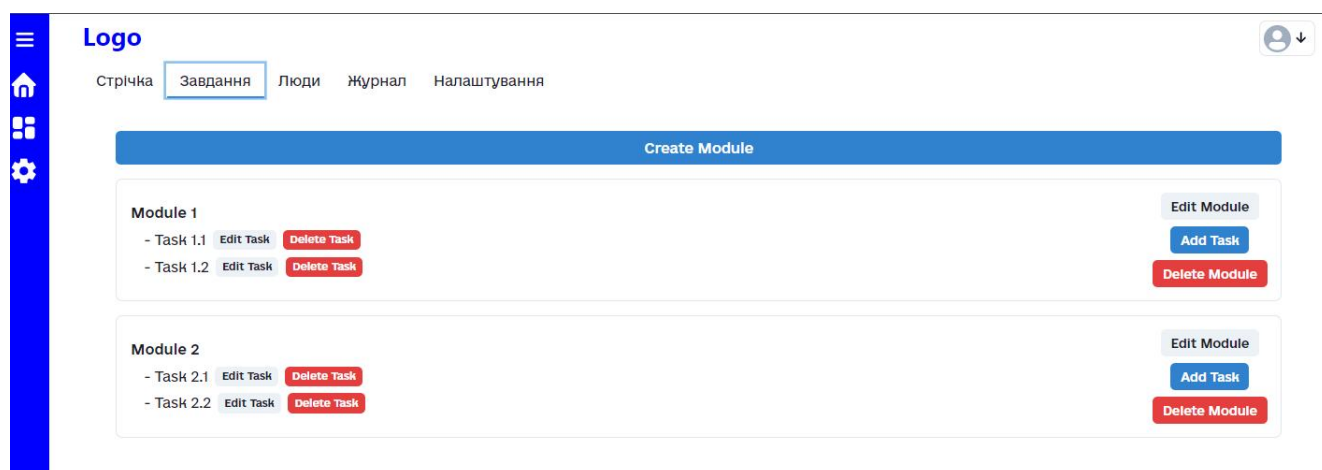


Рисунок 3.15. - Сторінка управління завданнями курсу

Сторінка управління завданнями курсу, зображено на рисунку 3.15, надає викладачам можливість створювати, редагувати та видаляти модулі та завдання в межах курсу. Інтерфейс цієї сторінки складається з декількох ключових елементів, що забезпечують зручність у використанні та ефективне управління навчальним процесом.

Logo

Стрічка

Завдання

Люди

Журнал

Налаштування

Generate Report

STUDENT	TASK 1	TASK 2	TASK 3	OVERALL GRADE
Student 1	85	90	78	84.33
Student 2	88	92	80	86.67
Student 3	75	85	90	83.33

Рисунок 3.16. - Сторінка журналу курсу

Сторінка журналу курсу, зображено на рисунку 3.16, надає викладачам можливість переглядати оцінки студентів за різні завдання та генерувати звіти про успішність. Інтерфейс цієї сторінки складається з декількох основних елементів, що забезпечують зручність у використанні та ефективне управління оцінюванням студентів.

☰ 🏠 ⚙️	Logo	Стрічка Завдання Люди Журнал Налаштування		👤 ↓
	Name			
	Image	Виберите файл Файл не выбран		
	Color	🌈 #FFFFFF HEX		
	Teachers	Select... ▼		
Students	Select... ▼			🧠

Рисунок 3.17. - Сторінка налаштувань курсу

Сторінка налаштувань курсу, зображено на рисунку 3.17, надає викладачам і адміністраторам можливість налаштовувати різні параметри курсу, що допомагає забезпечити індивідуальний підхід до його організації та проведення. Інтерфейс цієї сторінки складається з декількох основних елементів, що дозволяють зручно та ефективно змінювати налаштування курсу.

3.3 Тестування системи

Для тестування системи було використано Swagger та створено unit тести.



Рисунок 3.18. - Swagger інтерфейс

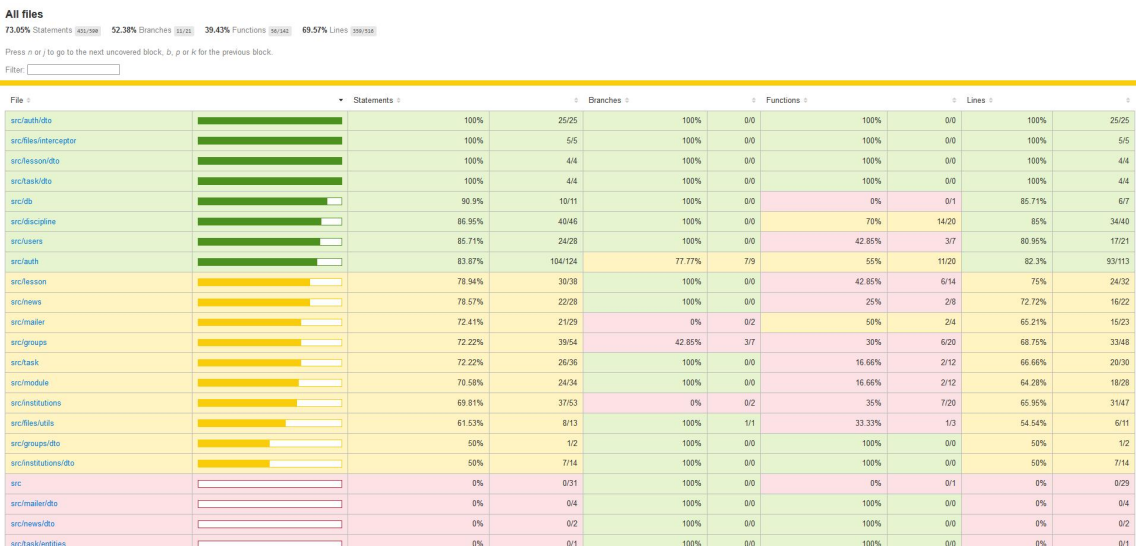


Рисунок 3.19. - Сторінка Unit тестів

На рисунку 3.18 зображено інтерфейс для візуалізації та тестування API. На рисунку 3.19 зображено сторінку, яка демонструє на скільки відсотків тести перевіряють код програми.

Висновки до третього розділу

У третьому розділі було детально розглянуто процес тестування системи автоматизації онлайн-навчання. Було використано два основних підходи до

тестування: інтерактивне тестування API за допомогою Swagger та модульне тестування (Unit тести).

Swagger забезпечив ефективний механізм для тестування API системи. Інтерактивна документація, створена за допомогою Swagger UI, дозволила розробникам та тестувальникам здійснювати запити та перевіряти відповіді API в реальному часі. Це сприяло швидкому виявленню та виправленню помилок, що покращило загальну якість і надійність системи.

Unit тести були розроблені для перевірки коректності роботи окремих компонентів системи. Модульні тести допомогли ізолювати та протестувати окремі функції та методи, що дозволило швидко ідентифікувати та локалізувати проблеми. Регулярне виконання Unit тестів забезпечувало стабільність системи при внесенні змін та оновленнях, що особливо важливо для підтримки високої якості коду.

Поєднання Swagger та Unit тестів забезпечило всебічне тестування системи автоматизації онлайн-навчання. Swagger дозволив перевіряти зовнішню взаємодію з API, тоді як Unit тести гарантували коректну роботу внутрішніх компонентів. Це поєднання методів тестування сприяло підвищенню якості та надійності системи, забезпечуючи її готовність до використання в реальних умовах.

Результати тестування показали, що система відповідає визначеним вимогам, забезпечує стабільну та коректну роботу, а також готова до подальшої інтеграції та використання в навчальному процесі.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано систему автоматизації онлайн-навчання, проведено аналіз предметної області та порівняно різні платформи, що використовуються для дистанційного навчання. На основі цього аналізу були визначені основні вимоги до системи автоматизації онлайн-навчання. У процесі дослідження була розроблена архітектура, підходи та основні технології реалізації вебдодатку.

Проектування системи включало аналіз варіантів використання програмного продукту та проектування структури, включаючи моделювання взаємодії між складовими частинами. Також було спроектовано базу даних для зберігання інформації про курси, студентів, викладачів та завдання, що зберігаються на серверній частині додатку.

Описано алгоритм роботи з додатком та послідовні пункти варіантів його використання. Розглянуто основний функціонал, включаючи створення курсів, реєстрацію користувачів, управління завданнями та взаємодію між студентами та викладачами.

Також була надана детальна схема розгортання системи автоматизації онлайн-навчання, описано взаємодію структурних елементів системи та наведено опис інтерфейсу розробленого програмного продукту для різних типів користувачів. Показано роботу системи оцінювання та журналу успішності для викладачів та студентів, що надає інформацію про статус виконання завдань, активність студентів та загальну успішність.

Архітектура системи побудована на базі сучасних вебтехнологій, що гарантує масштабованість та високу продуктивність. Інтеграція з інструментами для автоматизації тестування, такими як Swagger та Unit тести, забезпечує високу якість програмного забезпечення та його надійність. Swagger забезпечив зручність та ефективність тестування API, тоді як Unit тести гарантували коректну роботу окремих компонентів системи.

Додаток має інтуїтивний інтерфейс, який дозволяє користувачам легко створювати та управляти курсами, встановлювати завдання та оцінки, а також взаємодіяти один з одним. Користувачі можуть реєструватися в системі, проходити курси, виконувати завдання та спостерігати за своїм прогресом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Features [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.moodle.org/404/en/Features>.
2. BLACKBOARD LEARN [Електронний ресурс] – Режим доступу до ресурсу: <https://www.anthology.com/products/teaching-and-learning/learning-effectiveness/blackboard-learn>.
3. Teaching and Learning. To the Power of Canvas LMS. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.instructure.com/higher-education/products/canvas/canvas-lms>.
4. Google Classroom is the main address of Education, Training, and Study! [Електронний ресурс] – Режим доступу до ресурсу: <https://sites.google.com/view/classroom-workspace/>.
5. Edmodo [Електронний ресурс] – Режим доступу до ресурсу: <https://www.godaddy.com/>.
6. NestJS Documentation - Офіційна документація NestJS [Електронний ресурс]. – Режим доступу до ресурсу: <https://docs.nestjs.com>
7. ASP.NET MVC Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://dotnet.microsoft.com/en-us/apps/aspnet/mvc>.
8. NextJS Documentation - Офіційна документація NextJS [Електронний ресурс]. – Режим доступу до ресурсу: <https://nextjs.org/docs>
9. Server-side rendering [Електронний ресурс] – Режим доступу до ресурсу: <https://v17.angular.io/guide/ssr>.
10. HTML: HyperText Markup Language [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
11. React Documentation - Офіційна документація React [Електронний ресурс]. – Режим доступу до ресурсу: <https://reactjs.org/docs/getting-started.html>
12. TypeScript Documentation - Офіційна документація TypeScript [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>

13. Jest Documentation - Офіційна документація Jest [Електронний ресурс]. – Режим доступу до ресурсу: <https://jestjs.io/docs/en/getting-started>

14. Prisma Documentation - Офіційна документація Prisma [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.prisma.io/docs/>

15. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en>.

16. What is JavaScript? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/javascript/?nc1=h_ls.

17. Swagger Documentation - Офіційна документація Swagger [Електронний ресурс]. – Режим доступу до ресурсу: <https://swagger.io/docs/>

18. Zustand Documentation - Офіційна документація Zustand [Електронний ресурс]. – Режим доступу до ресурсу: <https://zustand.surge.sh/>

19. Chakra UI Documentation - Офіційна документація Chakra UI [Електронний ресурс]. – Режим доступу до ресурсу: <https://chakra-ui.com/docs>

20. Formik Documentation - Офіційна документація Formik [Електронний ресурс]. – Режим доступу до ресурсу: <https://formik.org/docs/overview>

21. PostgreSQL Documentation - Офіційна документація PostgreSQL [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>

22. What is an ORM – The Meaning of Object Relational Mapping Database Tools [Електронний ресурс] – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/>.

23. Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>.

24. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/docs>.

25. Docker: Accelerated Container Application Development [Електронний ресурс] – Режим доступу до ресурсу: <https://www.docker.com/>.

26. React Testing Library [Электронный ресурс] – Режим доступа до ресурсу:
<https://testing-library.com/docs/react-testing-library/intro/>.

27. TanStack Query [Электронный ресурс] – Режим доступа до ресурсу:
<https://tanstack.com/query/latest/docs/framework/react/overview>.

28. Що таке Axios? [Электронный ресурс] – Режим доступа до ресурсу:
<https://axios-http.com/uk/docs/intro>.

29. Multer [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.npmjs.com/package/multer>.

30. S. M. Huddar. Impacts Of Online Education Learning [Электронный ресурс]
/ S. M. Huddar, S. P. Chavarkar, B. U. Patil. – 2023. – Режим доступа до ресурсу:
https://www.researchgate.net/publication/377933739_Impacts_Of_Online_Education_Learning.

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						60

Лістинг коду

```

import {
  Controller,
  Post,
  Body,
  Res,
  HttpStatusCode,
  HttpStatus,
  Get,
  UseGuards,
  Query,
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthUserBody, SingInData, SingUpAdminBody } from './dto/user.dto';
import { ApiCreatedResponse, ApiOkResponse } from '@nestjs/swagger';
import { Response } from 'express';
import { CookieService } from './cookie.service';
import { AuthGuard } from './auth.guard';
import { SessionInfo } from './session-info.decorator';
import { GetSessionInfoDto } from './dto/session.dto';

@Controller('auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private cookieService: CookieService,
  ) {}

  @Post('sing-up')
  @ApiCreatedResponse()
  async singUp(@Body() createAuthDto: AuthUserBody) {
    await this.authService.singUp(createAuthDto);
    return { message: 'Registration successful' };
  }

  @Post('sing-up-admin')
  @ApiCreatedResponse()
  @HttpCode(HttpStatus.OK)
  async singUpAdmin(
    @Body() createAuthDto: SingUpAdminBody,
    @Res({ passthrough: true }) res: Response,
  ) {

```

```

    ) {
        const { accessToken } = await this.authService.singUpAdmin(createAuthDto);
        this.cookieService.setToken(res, accessToken);
    }

    @Post('sing-in')
    @ApiResponse()
    @HttpCode(HttpStatus.OK)
    async singIn(
        @Body() createAuthDto: SingInData,
        @Res({ passthrough: true }) res: Response,
    ) {
        const { accessToken } = await this.authService.signIn(createAuthDto);
        this.cookieService.setToken(res, accessToken);
    }

    @Post('sign-out')
    @ApiResponse()
    @HttpCode(HttpStatus.OK)
    @UseGuards(AuthGuard)
    signOut(@Res({ passthrough: true }) res: Response) {
        this.cookieService.removeToken(res);
        console.log(res.cookie);
    }

    @Get('verify?')
    async verify(
        @Query('id') id: string,
        @Res({ passthrough: true }) res: Response,
    ) {
        this.authService.verifiedEmail(+id);
        res.redirect('http://localhost:3000/login');
    }

    @Get('session')
    @ApiResponse({
        type: GetSessionInfoDto,
    })
    @UseGuards(AuthGuard)
    getSessionInfo(@SessionInfo() session: GetSessionInfoDto) {
        return session;
    }
}

```

```

import {
    CanActivate,
    ExecutionContext,
    Injectable,
    UnauthorizedException,
} from '@nestjs/common';
import { Request } from 'express';
import { CookieService } from '../cookie.service';
import { JwtService } from '@nestjs/jwt';

@Injectable()
export class AuthGuard implements CanActivate {
    constructor(private jwtService: JwtService) {}

    canActivate(context: ExecutionContext) {
        const req = context.switchToHttp().getRequest() as Request;
        const token = req.cookies[CookieService.tokenKey];
        if (!token) {
            throw new UnauthorizedException();
        }

        try {
            const sessionInfo = this.jwtService.verifyAsync(token, {
                secret: process.env.JWT_SECRET,
            });

            req['session'] = sessionInfo;
        } catch {
            throw new UnauthorizedException();
        }

        return true;
    }
}

import {
    BadRequestException,
    Injectable,
    NotFoundException,
    UnauthorizedException,
} from '@nestjs/common';
import { InstitutionsService } from '../institutions/institutions.service';
import { AuthUserBody, SingInData, SingUpAdminBody } from '../dto/user.dto';
import { MailerService } from '../mailer/mailer.service';

```

```

import { PasswordService } from './password.service';
import { JwtService } from '@nestjs/jwt';
import { DbService } from '../db/db.service';

@Injectable()
export class AuthService {
  constructor(
    private mailer: MailerService,
    private db: DbService,
    private institutionsService: InstitutionsService,
    private passwordService: PasswordService,
    private jwtService: JwtService,
  ) {}

  async singUp(authData: AuthUserBody, isVerified?: boolean, role?: string) {
    const user = await this.db.user.findFirst({
      where: {
        email: authData.email,
        institutions: {
          some: {
            educationalInstitution: { id: +authData.educationalInstitution },
          },
        },
      },
    });
    if (user) {
      throw new BadRequestException('User already exists');
    }
    const salt = this.passwordService.getSalt();
    const hash = this.passwordService.getHash(authData.password, salt);
    const newUserData = {
      firsName: authData.firsName,
      middleName: authData.middleName,
      lastName: authData.lastName,
      email: authData.email,
      hash,
      salt,
      role,
    };
    const newUser = await this.create(
      newUserData,
      +authData.educationalInstitution,
      isVerified || false,
  
```

```

    );
    console.log(authData.password);
    await this.mailer.sendEmail({
      recipients: 'ipzk221_kvo@student.ztu.edu.ua',
      subject: 'Hello in LMS!',
      html: `

Press <a href="http://localhost:3003/auth/verify?id=${newUser.id}">here</a> to verify
your email. Thanks</p>`,
    });

    return {
      id: newUser.id,
      email: newUser.email,
      educationalInstitution: +authData.educationalInstitution,
    };
  }
  async singUpAdmin({
    email,
    password,
    educationalInstitution,
    ...data
  }: SingUpAdminBody) {
    const institution = await this.institutionsService.findByName(
      educationalInstitution,
    );
    if (institution) {
      throw new BadRequestException({
        type: 'An educational institution with this name already exists',
      });
    }
    const newInstitution = await this.institutionsService.create({
      name: educationalInstitution,
    });
    const user = await this.singUp(
      {
        email,
        password,
        educationalInstitution: newInstitution.id,
        ...data,
      },
      true,
      'admin',
    );
    const accessToken = await this.jwtService.signAsync({


```

```

        id: user.id,
        email: user.email,
        role: 'admin',
        educationalInstitution: user.educationalInstitution,
    });

    return { accessToken };
}

async signIn({ email, password, educationalInstitution }: SignInData) {
    const user = await this.db.user.findFirst({
        where: {
            email,
            institutions: {
                some: { educationalInstitution: { id: +educationalInstitution } },
            },
        },
    });

    if (!user) {
        throw new NotFoundException('User not found');
    }

    const hash = this.passwordService.getHash(password, user.salt);
    if (hash !== user.hash) {
        throw new UnauthorizedException('Password wrong');
    }

    console.log(user);
    const isVerifiedUser = await this.db.userInstitution.findFirst({
        where: {
            userId: user.id,
            educationalInstitutionId: +educationalInstitution,
        },
    });

    if (!isVerifiedUser) {
        throw new UnauthorizedException('Your request was denied ');
    }

    if (!isVerifiedUser.isVerified) {
        throw new UnauthorizedException(
            'You have not yet been added to the educational institution',
        );
    }

    if (user.role === 'admin') {
        console.log(user);
        await this.db.adminProfile.upsert({

```

```

        where: {
            userId: user.id,
        },
        create: {
            userId: user.id,
        },
        update: {
            userId: user.id,
        },
    });
}

const accessToken = await this.jwtService.signAsync({
    id: user.id,
    email: user.email,
    role: user.role,
    educationalInstitution: isVerifiedUser.educationalInstitutionId,
});

return { accessToken };
}

verifiedEmail(id: number) {
    return this.db.user.update({
        where: {
            id,
        },
        data: {
            isVerifiedEmail: true,
        },
    });
}

create(
    data: {
        email: string;
        lastName: string;
        firstName: string;
        middleName: string;
        hash: string;
        salt: string;
    },
    educationalInstitution: number,
    isVerified: boolean,
) {
    return this.db.user.create({
        data: {

```

```

        ...data,
        institutions: {
            create: [
                {
                    isVerified,
                    educationalInstitution: {
                        connect: {
                            id: educationalInstitution,
                        },
                    },
                },
            ],
        },
    });
}
}

import { Injectable } from '@nestjs/common';
import { Response } from 'express';

@Injectable()
export class CookieService {
    static tokenKey = 'access-token';
    setToken(res: Response, token: string) {
        res.cookie(CookieService.tokenKey, token, {
            httpOnly: true,
            maxAge: 24 * 60 * 60 * 1000,
        });
    }
    removeToken(res: Response) {
        res.clearCookie(CookieService.tokenKey);
    }
}

import { Injectable } from '@nestjs/common';
import { randomBytes, pbkdf2Sync } from 'crypto';

@Injectable()
export class PasswordService {
    getSalt() {
        return randomBytes(16).toString('hex');
    }
    getHash(password: string, salt: string) {

```

```

        return pbkdf2Sync(password, salt, 1000, 64, 'sha512').toString('hex');
    }
}

import { Body, Controller, Get, Param, Patch, Post } from '@nestjs/common';
import { DisciplineService } from '../discipline.service';

@Controller('discipline')
export class DisciplineController {
    constructor(private readonly disciplineService: DisciplineService) {}

    @Post()
    create(
        @Body()
        createDisciplineDto: {
            name: string;
            teacher: string;
            details?: string;
        },
    ) {
        return this.disciplineService.create({
            name: createDisciplineDto.name,
            teacherId: +createDisciplineDto.teacher,
            details: createDisciplineDto.details,
        });
    }

    @Get('students/:id')
    findAllStudentDisciplines(@Param('id') id: string) {
        return this.disciplineService.findAllStudentDisciplines(+id);
    }

    @Get('teachers/:id')
    findAllTeacherDisciplines(@Param('id') id: string) {
        return this.disciplineService.findAllTeacherDisciplines(+id);
    }

    @Get('/:id')
    findOne(@Param('id') id: string) {
        return this.disciplineService.findDiscipline(+id);
    }

    @Patch('name/:id')
    update(@Param('id') id: string, @Body() { name }: { name: string }) {
        return this.disciplineService.updateName(+id, name);
    }
}

```

```

@Patch('students/:id')
updateStudents(
  @Param('id') id: string,
  @Body() { students }: { students: string[] },
) {
  return this.disciplineService.connectStudents(+id, students);
}

@Patch('teachers/:id')
updateTeachers(
  @Param('id') id: string,
  @Body() { teachers }: { teachers: string[] },
) {
  return this.disciplineService.connectTeacher(+id, teachers);
}

@Patch('modules/:id')
updateModules(
  @Param('id') id: string,
  @Body() { modules }: { modules: string[] },
) {
  return this.disciplineService.connectModules(+id, modules);
}
}

import { Module } from '@nestjs/common';
import { DisciplineService } from '../discipline.service';
import { DisciplineController } from '../discipline.controller';
import { DbModule } from '../db/db.module';

@Module({
  imports: [DbModule],
  controllers: [DisciplineController],
  providers: [DisciplineService],
})
export class DisciplineModule {}

import { Injectable } from '@nestjs/common';
import { DbService } from '../db/db.service';

@Injectable()
export class DisciplineService {
  constructor(private db: DbService) {}

  async create({
    name,

```

```

        teacherId,
        details,
    }: {
        name: string;
        details?: string;
        teacherId: number;
    }) {
        console.log(teacherId);
        const user = await this.db.user.findFirst({
            where: { id: teacherId, role: 'teacher' },
            select: { teacherProfile: { select: { id: true } } },
        });
        console.log(user);
        if (!user) {
            throw new Error('Teacher not found');
        }
        return await this.db.academicDiscipline.create({
            data: {
                name,
                details,
                teachers: {
                    create: {
                        teacher: {
                            connect: { id: user.teacherProfile?.id },
                        },
                    },
                },
            },
        });
    }

    findAllTeacherDisciplines(id: number) {
        return this.db.teacherDiscipline.findMany({
            where: {
                teacher: {
                    user: {
                        id,
                    },
                },
            },
            select: {
                discipline: {
                    select: {

```

					ИПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						71

```

        id: true,
        name: true,
        details: true,
        color: true,
        teachers: true,
        students: true,
        modules: true,
    },
},
});
}

async findAllStudentDisciplines(studentID: number) {
    const disciplines = await this.db.academicDiscipline.findMany({
        where: {
            students: {
                some: {
                    student: {
                        id: studentID,
                    },
                },
            },
        },
    });
    console.log(disciplines);
    return disciplines;
}

findDiscipline(id: number) {
    return this.db.academicDiscipline.findFirst({ where: { id } });
}

updateName(id: number, name: string) {
    return this.db.academicDiscipline.update({
        where: { id },
        data: { name },
    });
}

connectTeacher(id: number, teachers: string[]) {
    return this.db.academicDiscipline.update({
        where: { id },
        data: {

```

```

        teachers: {
            create: teachers.map((id) => ({ teacher: { connect: { id: +id } } })),
        },
    },
});
}

connectStudents(id: number, students: string[]) {
    return this.db.academicDiscipline.update({
        where: { id },
        data: {
            students: {
                create: students.map((id) => ({ student: { connect: { id: +id } } })),
            },
        },
    });
}

connectModules(id: number, modules: string[]) {
    return this.db.academicDiscipline.update({
        where: { id },
        data: {
            modules: {
                connect: modules.map((id) => ({ id: +id })),
            },
        },
    });
}

remove(id: number) {
    return this.db.academicDiscipline.delete({ where: { id } });
}
}

import { Controller, Get, Param, Res } from '@nestjs/common';
import { FilesService } from '../files.service';
import { Response } from 'express';

@Controller('files')
export class FilesController {
    constructor(private readonly filesService: FilesService) {}

    @Get('/:id')
    async getFile(@Param('id') id: string, @Res() res: Response) {

```

```

        console.log(id);

        const filePath = await this.filesService.getFile(+id);

        console.log(filePath);

        res.sendFile(filePath);
    }
}

import { Module } from '@nestjs/common';
import { FilesService } from './files.service';
import { FilesController } from './files.controller';
import { DbModule } from '../db/db.module';

@Module({
  imports: [DbModule],
  controllers: [FilesController],
  providers: [FilesService],
})
export class FilesModule {}

import { Injectable } from '@nestjs/common';
import { DbService } from '../db/db.service';

@Injectable()
export class FilesService {
  constructor(private db: DbService) {}

  async getFile(id: number) {
    const file = await this.db.file.findFirst({ where: { id } });
    const filePath = __dirname + '/uploads/' + file?.link;
    return filePath;
  }
}

import { Injectable } from '@nestjs/common';
import { CreateGroupDto } from '../dto/create-group.dto';
import { UpdateGroupDto } from '../dto/update-group.dto';
import { DbService } from '../db/db.service';

@Injectable()
export class GroupsService {
  constructor(private db: DbService) {}

  async create({ institution, curator, ...data }: CreateGroupDto) {
    const curatorExists = await this.db.teacherProfile.findUnique({
      where: { userId: curator },
    });

    const teacherProfile = curatorExists

```

```

    ? curatorExists
    : await this.db.teacherProfile.create({
      data: {
        userId: curator,
      },
    });
console.log(teacherProfile);

const validStudentIds: number[] = [];

for (const studentId of data.students) {
  const studentExists = await this.db.studentProfile.findUnique({
    where: { id: studentId },
  });
  if (studentExists) {
    validStudentIds.push(studentId);
  }
}

return this.db.group.create({
  data: {
    name: data.name,
    curator: { connect: { id: teacherProfile.id } },
    institution: { connect: { id: institution } },
    students: {
      create: validStudentIds.map((studentId) => ({
        student: { connect: { id: studentId } },
      })),
    },
  },
  select: {
    id: true,
    name: true,
    curator: { select: { id: true } },
  },
});
}

findAll(id: number) {
  return this.db.group.findMany({
    where: { institution: { id } },
    select: {
      id: true,

```

```

        name: true,
        curator: { select: { id: true } },
    },
});
}

findOne(id: number) {
    return this.db.group.findFirst({
        where: { id },
        select: {
            id: true,
            name: true,
            curator: { select: { id: true } },
            students: {
                select: {
                    student: {
                        select: {
                            id: true,
                        },
                    },
                },
            },
        },
    });
}

findCuratorGroups(curatorId: number, institutionId: number) {
    return this.db.group.findMany({
        where: { curator: { id: curatorId }, institution: { id: institutionId } },
    });
}

async update(id: number, data: UpdateGroupDto) {
    data.students &&
    (await this.db.studentGroup.createMany({
        data: data.students.map((student) => ({
            groupId: id,
            studentId: student,
        })),
    }));
    return this.db.group.update({
        where: { id },
        data: {

```

```

        name: data.name,
        ...(data?.curator && { curator: { connect: { id: data.curator } } })),
    },
  });
}

remove(id: number) {
  return this.db.group.delete({ where: { id } });
}

getStudentsGroup(id: number) {
  return this.db.studentGroup.findMany({
    where: { groupId: id },
    include: { student: true },
  });
}

getStudentsNoGroup(id: number) {
  return this.db.studentGroup.findMany({ where: { NOT: { groupId: id } } });
}
}

import { Module } from '@nestjsjs/common';
import { GroupsService } from './groups.service';
import { GroupsController } from './groups.controller';
import { DbModule } from '../db/db.module';

@Module({
  imports: [DbModule],
  controllers: [GroupsController],
  providers: [GroupsService],
})
export class GroupsModule {}

import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
} from '@nestjsjs/common';
import { GroupsService } from './groups.service';
import { UpdateGroupDto } from './dto/update-group.dto';

```

```

@Controller('group')
export class GroupsController {
    constructor(private readonly groupsService: GroupsService) {}

    @Post()
    create(
        @Body()
        createGroupDto: {
            name: string;
            teacher: string;
            educationalInstitution: string;
            students: string[];
        },
    ) {
        return this.groupsService.create({
            name: createGroupDto.name,
            curator: +createGroupDto.teacher,
            institution: +createGroupDto.educationalInstitution,
            students: createGroupDto.students.map((student) => +student),
        });
    }

    @Get('/all/:id')
    findAll(@Param('id') id: string) {
        return this.groupsService.findAll(+id);
    }

    @Get('/:id')
    findOne(@Param('id') id: string) {
        return this.groupsService.findOne(+id);
    }

    @Patch('/:id')
    update(@Param('id') id: string, @Body() updateGroupDto: UpdateGroupDto) {
        return this.groupsService.update(+id, updateGroupDto);
    }

    @Delete('/:id')
    remove(@Param('id') id: string) {
        return this.groupsService.remove(+id);
    }

    @Get('/students/:id')

```

					ИПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						78

```

    getStudents(@Param('id') id: string) {
        return this.groupsService.getStudentsGroup(+id);
    }

    @Get('/students-no-group/:id')
    getStudentsNoGroup(@Param('id') id: string) {
        return this.groupsService.getStudentsNoGroup(+id);
    }
}

import { Injectable } from '@nestjs/common';
import { DbService } from '../db/db.service';

@Injectable()
export class InstitutionsService {
    constructor(private db: DbService) {}

    findByName(name: string) {
        return this.db.educationalInstitution.findFirst({ where: { name } });
    }

    async getAll() {
        return await this.db.educationalInstitution.findMany({
            select: {
                id: true,
                name: true,
            },
        });
    }

    get(id: number) {
        return this.db.educationalInstitution.findFirst({
            where: { id },
            select: { id: true, name: true, logo: { select: { fileId: true } } },
        });
    }

    delete(id: number) {
        return this.db.educationalInstitution.delete({ where: { id } });
    }

    create(data: { name: string }) {
        return this.db.educationalInstitution.create({
            data: { ...data },
        });
    }

    getUsers(id: number, role?: string) {
        return this.db.user.findMany({
            where: {

```

```

        role: role || null,
        institutions: { some: { educationalInstitution: { id } } },
    },
    select: {
        id: true,
        email: true,
        lastName: true,
        firstName: true,
        middleName: true,
    },
});
}

async updateUserRole(id: number, role: string, institutionId: number) {
    const isVerifiedUser = await this.db.user.update({
        where: {
            id,
            institutions: { some: { educationalInstitutionId: institutionId } },
        },
        data: { role },
    });

    return await this.db.userInstitution.update({
        where: {
            userId_educationalInstitutionId: {
                userId: isVerifiedUser.id,
                educationalInstitutionId: institutionId,
            },
        },
        data: {
            isVerified: true,
        },
    });
}

deleteUser(id: number, institutionId: number) {
    return this.db.userInstitution.delete({
        where: {
            userId_educationalInstitutionId: {
                userId: id,
                educationalInstitutionId: institutionId,
            },
        },
    });
}
}

```

```

update({
  id,
  name,
  logo,
}): {
  id: number;
  name?: string;
  logo?: { link: string; type: string };
}) {
  const data = {};
  if (name) data['name'] = name;
  if (logo)
    data['logo'] = {
      create: {
        file: {
          create: logo,
        },
      },
    };
  return this.db.educationalInstitution.update({
    where: {
      id,
    },
    data,
  });
}
}

import { Injectable } from '@nestjs/common';
import * as nodemailer from 'nodemailer';
import { CreateMailerDto } from './dto/create-mailer.dto';
import Mail from 'nodemailer/lib/mailer';

@Injectable()
export class MailerService {
  mailTransport() {
    return nodemailer.createTransport({
      host: 'smtp.gmail.com',
      port: 587,
      secure: false,
      auth: {
        user: 'ipzk221_kvo@student.ztu.edu.ua',
        pass: 'hspg csgi aqjk wfve',
      },
    },

```

```

    });
  }
  async sendEmail(dto: CreateMailerDto) {
    const { from, recipients, subject, html } = dto;
    const transport = this.mailTransport();
    const options: Mail.Options = {
      from: from ?? {
        name: 'LMS',
        address: 'ipzk221_kvo@student.ztu.edu.ua',
      },
      to: recipients,
      subject,
      html,
    };
    try {
      const result = await transport.sendMail(options);
      return result;
    } catch (err) {
      throw Error('error send message');
    }
  }
}

import { AUTH_INSTANCE } from "../../constants/api";
import { SingInBody, SingUpAdminBody, SingUpBody } from "../interfaces";

export const singUp = async (data: SingUpBody) => {
  return AUTH_INSTANCE.post("sing-up", data);
};

export const singIn = (data: SingInBody) => {
  return AUTH_INSTANCE.post("sing-in", data);
};

export const singUpAdmin = async (data: SingUpAdminBody) => {
  return AUTH_INSTANCE.post("sing-up-admin", data);
};

import { FILES_INSTANCE } from "../../constants/api";

export const getFile = async (id: number) => {
  return (await (
    await FILES_INSTANCE.get(`/${id}`, {
      responseType: 'blob',
    })
  ).data as Blob);
}

```

```

}

import { GROUP_INSTANCE } from "../../constants/api";

type Group = {
  id: number;
  name: string;
  curator: number;
  students: number[];
};

export const getGroup = async (id: number): Promise<any> => {
  return await (
    await GROUP_INSTANCE.get(`${id}`)
  ).data;
};

export const getGroups = async (id: number): Promise<any[]> => {
  return await (
    await GROUP_INSTANCE.get(`all/${id}`)
  ).data;
};

export const createGroup = async (data: {
  name: string;
  teacher: string;
  educationalInstitution: string;
  students: string[];
}): Promise<any> => {
  return await (
    await GROUP_INSTANCE.post("", data)
  ).data;
};

export const getStudentsGroup = async (id: number): Promise<any> => {
  return await (
    await GROUP_INSTANCE.get(`students/${id}`)
  ).data;
};

export const getStudentsNoGroup = async (id: number): Promise<any> => {
  return await (
    await GROUP_INSTANCE.get(`students-no-group/${id}`)
  ).data;
};

import { NEWS_INSTANCE } from "../../constants/api"

```

```

export const createNews = async ({title, content, file, educationalInstitutionId, authorId}: { authorId:
string; title: string; content: string; file?: File, educationalInstitutionId: string }) => {
    const data = new FormData();
    data.append('title', title);
    data.append('content', content);
    data.append('educationalInstitutionId', educationalInstitutionId);
    data.append('authorId', authorId);
    if (file) {
        data.append('file', file);
    }
    return NEWS_INSTANCE.post("", data);
}

export const getNews = (id: number) => {
    return NEWS_INSTANCE.get(`/${id}`);
}

import { USER_INSTANCE, USERS_INSTANCE } from "../../constants/api";

export const getJoinRequests = async (
    id: number
): Promise<{
    id: number;
    image: string;
    lastName: string;
    firstName: string;
    middleName: string;
    email: string;
    isVerifiedEmail: boolean;
    isVerified: boolean;
}> => {
    return await (
        await USERS_INSTANCE.get(`join-requests/${id}`)
    ).data;
};

export const getTeachers = async (id: number): Promise<any> => {
    return await (
        await USERS_INSTANCE.get(`teachers/${id}`) as any
    ).data;
};

export const getStudents = async (id: number): Promise<any> => {
    return await (
        await USERS_INSTANCE.get(`students/${id}`)
    ).data as any;
};

```

```

export const getTeacher = async (id: number) => {
  return await (
    await USERS_INSTANCE.get(`teacher/${id}`)
  ).data as {
    id: number;
    image: string;
    lastName: string;
    firstName: string;
    middleName: string;
    email: string;};
}

export const getStudent = async (id: number): Promise<any> => {
  return await (
    await USERS_INSTANCE.get(`student/${id}`) as any
  ).data;
}

export const getUser = async (id: number): Promise<{id: number, image: string, lastName: string,
firstName: string, middleName: string, email: string}> => {
  return await (
    await USER_INSTANCE.get(`${id}`)
  ).data;
}

import { Box, Center, Flex } from "@chakra-ui/react";
import { Icon } from "@iconify/react";
const variants = {
  success: {
    icon: "mdi:success-bold",
    bgColor: "green",
  },
  delete: {
    icon: "material-symbols:delete",
    bgColor: "red",
  },
};

export const ActionButton = ({
  variant,
  onClick,
  icon,
  text,
  bgColor,

```

```

    color,
    width,
    height,
  }: {
    variant?: "success" | "delete";
    width?: string;
    height?: string;
    color?: string;
    onClick?: VoidFunction;
    icon?: string;
    text?: string;
    bgColor?: string;
  }) => {
    return (
      <Flex
        bgColor={
          bgColor || variants[variant as keyof typeof variants]?.bgColor || "black"
        }
        w={width || "40px"}
        h={height || "40px"}
        color={color || "white"}
        padding={2}
        gap={2}
        textAlign={"center"}
        borderRadius={5}
        justifyContent={"center"}
        alignContent={"center"}
        onClick={onClick}
      >
        <Center>
          <Box>{text}</Box>
          <Icon
            icon={icon || variants[variant as keyof typeof variants].icon || ""}
            width="30px"
            height="30px"
            style={{ color: color || "white" }}
          />
        </Center>
      </Flex>
    );
  };
import { Box } from "@chakra-ui/react";
import { motion } from "framer-motion";

```