ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка навчальної платформи для школи "Школа східних мов та культури" мовою JavaScript»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим МОГИЛЬНИК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Максим МОГИЛЬНИК

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Могильнику Максиму Руслановичу

1. Тема кваліфікаційної роботи: «Розробка навчальної платформи для школи "Школа східних мов та культури" мовою JavaScript»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Тимур ДОВЖЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вхідні дані до кваліфікаційної роботи:
   3.1 Інструменти розробки програмного забезпечення
   3.2 Інструменти обробки та візуалізації даних
   3.3 Методи обробки та візуалізації даних
   3.4 Офіційна документація, пов’язана з розробкою
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1. Аналіз предметної області
   4.3. Визначення вимог до реалізації
   4.4. Визначення засобів реалізації
   4.5. Розробка платформи відповідно до визначених засобів та вимог

5. Перелік графічного матеріалу:

5.1 Аналіз аналогів
5.2 Вимоги до програмного забезпечення
5.3 Програмні засоби реалізації
5.4 Діаграма варіантів використання
5.5 Мапа сайту для студента
5.6 Мапа сайту для викладача
5.7 Діаграма послідовності виконання першого завдання студента
5.8 Схема бази даних
5.9 Діаграма розгортання
5.10 Екранні форми
5.11 Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Планування архітектури проєкту	14.03-20.03.2024	
4	Розробка back-end частини проєкту	21.03-27.03.2024	
5	Розробка front-end чатини проєкту	28.03-03.04.2024	
6	Перевірка працездатності, виправлення помилок	04.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Максим МОГИЛЬНИК
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 1 табл., 1 рис., 11 джерел.

Мета роботи – покращення продуктивності освітнього процесу мовної школи.

Об'єкт дослідження – процес вивчення іноземних мов східної групи.

Предмет дослідження – програмне забезпечення для викладачів та студентів мовної школи.

Короткий зміст роботи: В роботі проаналізовано алгоритми та методи для зберігання та передачі текстових та графічних файлів. Проаналізовано бібліотеки та фреймворки для створення клієнт-серверної архітектури. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: завантаження текстових документів, завантаження та перегляд графічних документів, перегляд заздалегідь зазначених дат оплати за навчання, перегляд списку додаткової літератури, оцінювання робіт учнів викладачами, перегляд успішності студентів. В роботі використано фреймворк Express.JS для створення серверу, бібліотеку Mongoose для роботи з MongoDB базою даних, бібліотеку React для створення користувацького інтерфейсу та візуалізації даних.

Сферою використання застосунку є організація роботи зі студентами в процесі вивчення іноземних мов східної групи.

КЛЮЧОВІ СЛОВА: EXPRESS.JS, MONGODB, MONGOOSE, ФАЙЛИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

JSON - JavaScript Object Notation

BSON - Binary JavaScript Object Notation

UML - Unified Modeling Language

JSX - JavaScript Syntax eXtension

XML - EXtensible Markup Language

HTML - HyperText Markup Language

JS - JavaScript

БД - база даних

Front-end (Фронтенд) - користувацький інтерфейс системи на клієнтській стороні вебсайту чи додатку

Back-end (Бекенд) - бізнес-логіка веб-сайту чи додатку

JWT – JSON Web Token

JWKS – JSON Web Key Set

ВСТУП

Завдяки стрімкому розвитку інформаційних технологій з'являється можливість для автоматизації різноманітних процесів. В сучасному освітньому середовищі, де зростає кількість учнів та складність управління даними про них, актуальним стає знаходження ефективних рішень для обліку та керування інформацією про навчальний процес. Це вимагає розробки спеціалізованого веб-сервісу, який допоможе спростити процеси роботи з інформацією.

Метою цієї дипломної роботи є розробка веб-сервісу, який допоможе спростити та покращити продуктивність освітнього процесу мовної школи.

У бакалаврській роботі було виділено такі задачі:

1. Провести аналіз потреб та вимог до навчальної платформи "Школа східних мов та культури".

2. Дослідити існуючі програмні рішення для онлайн-навчання, визначити їх переваги та недоліки.

3. Сформулювати вимоги до програмного забезпечення з урахуванням потреб користувачів.

4. Провести огляд та аналіз IT-засобів для розробки програмного забезпечення.

5. Спроектувати та розробити веб-додаток для навчальної платформи.

Для розробки веб-сервісу було обрано мову програмування JavaScript, яка дозволяє створювати динамічні та інтерактивні веб-додатки. В якості інструменту розробки було обрано бібліотеку React, яка забезпечує високу швидкість та ефективність розробки. Додатки, створені з використанням React, легко підтримувати та масштабувати. Для розробки серверної частини було використано Node.js разом із фреймворком Express, що забезпечує асинхронність і високу продуктивність додатку. В якості БД було обрано MongoDB.

Результатом роботи є веб-сервіс, який налагоджує процес взаємодії між викладачами та студентами та забезпечує їм легкий та швидкий доступ до домашніх завдань.

1 АНАЛІЗ ПРОБЛЕМИ ТА ІСНУЮЧИХ ЗАСОБІВ ДЛЯ ЇЇ ВИРІШЕННЯ

1.1 Опис задачі (предметної галузі)

Процес вивчення іноземних мов, особливо східних, є складним та потребує комплексного підходу. Східні мови, такі як китайська, японська, корейська та інші, відрізняються від західних мов своєю граматичною структурою, фонетикою та ієрогліфічним письмом. Ці особливості створюють додаткові виклики для студентів та викладачів.

Основними проблемами, з якими стикаються школи та мовні курси, є:

1. Складність навчального матеріалу: Ієрогліфічні системи письма потребують більше часу для вивчення, що вимагає доступу до спеціалізованих навчальних матеріалів та методичних посібників.
2. Інтерактивність та доступність: Необхідність створення інтерактивного контенту для забезпечення більш ефективного навчального процесу, який включає мультимедійні ресурси.
3. Оцінювання знань: Відсутність систематичного підходу до оцінювання знань студентів, що ускладнює відслідковування їхнього прогресу.
4. Адміністративні завдання: Управління навчальними матеріалами, організація занять, контроль за оплатою навчання потребують багато часу та зусиль з боку адміністрації.

Для вирішення вищеозначених проблем необхідна спеціалізована навчальна платформа, яка повинна забезпечувати наступні функції:

1. Завантаження та збереження текстових та графічних документів.
2. Надання доступу до додаткових джерел інформації та літератури.
3. Оцінювання та відстежування прогресу
4. Модулі для керування навчальними групами та розкладом занять.
5. Інструменти для відстежування та контролю оплат за навчання.

Для "Школи східних мов та культури" розробка спеціалізованої платформи є критично важливою через наступні причини:

Специфіка навчального процесу: Вивчення східних мов вимагає спеціалізованих інструментів для роботи з ієрогліфами, що відрізняються від алфавітних систем.

1. Потреба в індивідуальному підході:

Кожен студент потребує індивідуального підходу, що можливо реалізувати через адаптивні навчальні програми та інтерактивні завдання.

2. Різноманітність навчальних матеріалів:

Платформа повинна забезпечувати доступ до широкого спектру навчальних матеріалів.

3. Зручність комунікації:

Необхідність забезпечення ефективної комунікації між студентами та викладачами через інструмент для обміну домашніми завданнями та їх коментування.

Таким чином, розробка навчальної платформи для "Школи східних мов та культури" має на меті створення ефективного інструменту, який полегшить процес навчання та викладання східних мов, забезпечить зручність управління навчальним процесом та сприятиме досягненню високих результатів у навчанні.

1.2 Опис та порівняння можливих аналогів з розробленою платформою

Розробка спеціалізованої навчальної платформи для школи "Школа східних мов та культури" вимагає ретельного аналізу існуючих програмних продуктів, які можуть бути використані для реалізації цієї задачі. Розглянемо такі платформи як: Mystat та Moodle LMS. Оцінка проводиться з точки зору їх можливостей, переваг, недоліків та відповідності специфічним вимогам саме мовної школи.

1.2.1 Mystat

Електронний щоденник Mystat - це система, що дозволяє студенту в режимі реального часу бачити свої оцінки і рейтинг, домашні завдання, розклад занять і іспитів, а також мати цілодобовий доступ до навчальних матеріалів. Система розроблена спеціально для IT STEP academy.

Переваги:

1. Вбудовані інструменти для управління завданнями та оцінками.
2. Можливість відслідковування успішності студентів.
3. Наявність вбудованого списку корисної літератури.

Недоліки:

1. Відсутність специфічних інструментів для роботи з ієрогліфами та іншими особливостями східних мов.
2. Відсутність модуля для новин про навчання та школу.

1.2.2 Moodle LMS

Moodle LMS є однією з найпопулярніших систем управління навчанням, яка використовується в багатьох навчальних закладах по всьому світу.

Переваги:

1. Відкрите програмне забезпечення з можливістю налаштування інтерфейсу.
2. Велика кількість модулів і плагінів для розширення функціональності.
3. Підтримка різних форматів навчальних матеріалів, включаючи текст, відео та інтерактивні елементи.
4. Активна спільнота та регулярні оновлення.

Недоліки:

1. Висока складність налаштування та адміністрування.
2. Вимогливість до ресурсів сервера.

3. Відсутність інтеграції з системами для відслідковування оплат за навчання.

1.2.3 Платформа "Школа східних мов та культури"

Розробка власної платформи дозволить врахувати всі специфічні вимоги та потреби школи, що робить її найбільш адаптивним рішенням.

Переваги:

1. Можливість інтеграції всіх необхідних функцій в одному місці.
2. Підтримка роботи з ієрогліфами та іншими особливостями східних мов.
3. Інструменти для управління оплатою, новинами та бібліотекою додаткових джерел.

Недоліки:

1. Потреба у фахівцях для майбутнього обслуговування платформи.
2. Більш тривалий час реалізації порівняно з готовими рішеннями.

1.2.4 Порівняльна таблиця

Для наочного порівняння можливостей розглянутих програмних продуктів наведено порівняльну таблицю.

Таблиця 1.1

Функціональність	Mystat	Moodle LMS	Школа східних мов та культури
Новини, що стосуються обраного напрямку навчання та школи/ЗВО	Ні	Ні	Так
Наявна бібліотека додаткових джерел інформації	Так	Ні	Так
Модуль завантаження завдань	Так	Так	Так
Модуль перегляду успішності студента	Так	Так	Так
Модуль відслідковування оплати за навчання	Так	Ні	Так

Як видно з порівняльного аналізу, жодна з існуючих платформ не забезпечує повного спектру функціональності, необхідного для управління навчальним процесом у школі східних мов. Зокрема, ні Mystat, ні Moodle LMS не мають інтегрованих модулів для новин та відслідковування оплат за навчання, що є критично важливими для школи. Власна платформа дозволить врахувати всі специфічні потреби та забезпечить максимальну ефективність та зручність використання.

Таким чином, розробка платформи для "Школи східних мов та культури" є актуальною і необхідною для забезпечення повноцінного навчального процесу та ефективного управління ним.

1.3 Опис використаних ІТ-засобів (бібліотеки та фреймворки)

Для розробки навчальної платформи "Школа східних мов та культури" необхідно було обрати найбільш придатні бібліотеки та фреймворки. Ці інструменти повинні забезпечити високу продуктивність, надійність, масштабованість та зручність у використанні.

Бібліотека - це набір підготовлених та підтримуваних функцій, методів, класів або компонентів, які можна використовувати для вирішення конкретних завдань. Зазвичай бібліотеки створюються для надання певної функціональності у певній області. Вони можуть включати в себе різноманітні функції, такі як робота з рядками, обробка даних, маніпулювання зображеннями або робота з мережею.

Основними характеристиками бібліотек є:

1. Перевикористання коду: Бібліотека дозволяє використовувати готові функції та класи без необхідності повторного їх написання. Це дозволяє розробнику ефективно використовувати готові рішення та швидше розвивати програму.

2. Підтримка стандартів: Бібліотеки можуть бути стандартизованими та підтримуватися великою спільнотою розробників, що дозволяє використовувати надійні та перевірені рішення.

3. Розширення можливостей програми: Використання бібліотек дозволяє програмістам додавати нові функціональності без необхідності написання коду з нуля. Це дозволяє швидше реалізувати нові функції та покращення у програмі.

Фреймворк - це більш великий та комплексний засіб для розробки програмного забезпечення, який надає структуру та основу для побудови програм та додатків. Фреймворки включають в себе бібліотеки, стандарти, інструменти та шаблони проектування, які допомагають розробникам створювати програми швидше та ефективніше.

Основними характеристиками фреймворків є:

1. Структура програми: Фреймворк визначає структуру програми та надає скелет або каркас для розробки. Він визначає основні компоненти програми, такі як моделі даних, контролери та представлення, і встановлює правила та конвенції розробки.

2. Шаблони проектування: Фреймворк може включати в себе готові шаблони проектування, які допомагають розробникам ефективно вирішувати типові завдання та проблеми.

3. Стандартизація: Фреймворк може встановлювати стандарти та правила розробки, що дозволяє розробникам працювати в єдиному стандарті та забезпечує консистентність у програмі.

1.3.1 Front-end

Front-end відповідає за взаємодію з користувачем та відображення контенту навчальної платформи. У проекті front-end виконує такі завдання:

- Відображення курсів і матеріалів: Front-end повинен надати зручний інтерфейс для користувачів, щоб вони могли легко знаходити необхідну інформацію.
- Управління користувачами: Забезпечення можливості реєстрації, входу та керування профілем користувача.

- Взаємодія з матеріалами: Користувачі повинні мати можливість виконувати завдання та мати зворотній зв'язок у вигляді коментарів.

Для реалізації функціоналу front-end частини було обрано бібліотеку React. React - це бібліотека JavaScript для побудови користувацьких інтерфейсів. Вона дозволяє створювати багаторазові компоненти, спрощуючи розробку та підтримку складних додатків.

Переваги бібліотеки React:

1. Висока продуктивність: Віртуальна DOM (Document Object Model) забезпечує швидке оновлення інтерфейсу.
2. Компонентний підхід: Можливість створення модульних та багаторазових компонентів.
3. Широка підтримка спільноти: Велика кількість бібліотек та інструментів, розроблених спільнотою.
4. Легка інтеграція: Можливість інтеграції з іншими бібліотеками та фреймворками.

1.3.2 Back-end

Back-end відповідає за обробку запитів користувачів, управління даними та взаємодію з базою даних. Для створення серверної частини платформи було обрано Express.JS.

Express.JS - це мінімалістичний фреймворк для Node.js, який дозволяє створювати веб-додатки та API. Він відомий своєю простотою використання, що робить його доступним для розробників будь-якого рівня. Простота вивчення та використання дозволяє розробникам швидко створювати потужні веб-додатки. Окрім того, Express.JS є дуже масштабованим, що дозволяє розробникам створювати як невеликі, так і великі проекти без втрати продуктивності. Велика кількість плагінів і модулів, доступних для Express.JS, дозволяє розширювати його функціональність.

1.3.3 База даних

База даних відповідає за збереження та організацію інформації про користувачів, курси, файли та інші дані. Для зберігання та маніпулювання даними використовується MongoDB разом з бібліотекою Mongoose.

MongoDB є документо-орієнтованою базою даних NoSQL, яка забезпечує гнучкість та високу продуктивність. Однією з головних переваг MongoDB є можливість зберігання даних у форматі JSON-подібних документів, що дозволяє легко розуміти структуру даних та працювати з ними. Однак, внутрішньо MongoDB використовує BSON (Binary JSON) для збереження даних у бінарному форматі. BSON є бінарним представленням JSON-подібних документів, що дозволяє ефективно зберігати дані та працювати з ними у MongoDB. Використання BSON дозволяє MongoDB оптимізувати розмір документів та швидкість їх обробки, що робить базу даних більш ефективною для роботи з великими обсягами даних та завданнями, які вимагають швидкого доступу до інформації.

GridFS є доповненням до базової функціональності MongoDB і призначений для зберігання та управління великими файлами, які не можуть бути збережені в стандартному об'ємі одного документа MongoDB. Використання GridFS дозволяє розбити великі файли на менші частини, що полегшує їх збереження та обробку.

GridFS розділяє великі файли на частини, які зберігаються в колекції MongoDB, під назвою chunks. Кожен chunk (чанк) має фіксований розмір (зазвичай 255 кілобайт), за винятком останнього, який може бути меншого розміру.

Додатково, GridFS створює колекцію files, в якій зберігаються метадані файлів, такі як ім'я файлу, тип MIME, розмір тощо. Ця колекція дозволяє ефективно керувати великою кількістю файлів та отримувати доступ до них за допомогою запитів MongoDB.

Використання GridFS у MongoDB дозволяє зручно та ефективно зберігати великі файли, такі як відео, аудіо або файли зображень, та забезпечує швидкий доступ до них у додатку.

Mongoose є бібліотекою для Node.js, яка забезпечує об'єктно-документне моделювання (ODM) для MongoDB. Вона дозволяє розробникам визначати структуру даних та взаємозв'язки між колекціями в MongoDB за допомогою JavaScript-подібного синтаксису. Однією з ключових переваг Mongoose є можливість визначення схем даних, що дозволяє проводити валідацію даних перед збереженням у базу. Використання Mongoose спрощує взаємодію з MongoDB, що перетворює розробку додатків з використанням цієї бази даних на більш зручний та продуктивний процес.

2 ПРОЕКТУВАННЯ НАВЧАЛЬНОЇ ПЛАТФОРМИ

2.1. Проектування архітектури/структури системи

2.1.1 Загальна архітектура системи

Навчальна платформа для школи "Школа східних мов та культури" будується на основі клієнт-серверної архітектури. Ця архітектура забезпечує розділення функціональності між клієнтською та серверною частинами, що дозволяє досягти високої гнучкості, масштабованості та зручності в обслуговуванні системи.

2.1.2 Компонентна структура системи

Клієнтська частина реалізована за допомогою React і складається з наступних основних компонентів:

1. Компонент аутентифікації: Відповідає за вхід користувачів до системи.
2. Компонент груп: Відображає інформацію про групи, дозволяє адміністраторам створювати нові групи та керувати ними.
3. Компонент завдань: Дозволяє студентам завантажувати виконані завдання, а викладачам - переглядати та оцінювати їх.
4. Компонент успішності: Відображає успішність студентів та дозволяє їм переглядати свої оцінки.

Серверна частина побудована з використанням Node.js і Express та складається з наступних основних модулів:

1. Модуль аутентифікації:

Обробляє запити на реєстрацію та вхід користувачів, використовує JWT (JSON Web Tokens) для управління сесіями. Створені JWT-токени проходять перевірку за допомогою JWKS (JSON Web Key Set). Для перевірки використовується наступний процес:

- Створення/наявність JWKS-модуля: Модуль, який надає ключі для перевірки токенів.
- Створення/наявність модуля для видачі токенів: Модуль, який підписує токени.
- Отримання JWKS з JWKS-модуля: Система отримує ключі для перевірки підписів.
- Витяг JWT з заголовка запиту: Система отримує токен з заголовка авторизації запиту.
- Декодування JWT та отримання унікального параметра kid (ID ключа) з заголовка: Система знаходить ключ для перевірки підпису токена.
- Знаходження ключа перевірки підпису в JWKS з відповідним параметром kid: Система використовує відповідний ключ для перевірки підпису.
- Перевірка токена з використанням фільтрованих JWKS: Система перевіряє токен за допомогою відповідного ключа.

2. Модуль курсів:

Управляє створенням, редагуванням та видаленням курсів, взаємодіє з базою даних для зберігання інформації про курси.

3. Модуль завдань:

Обробляє завантаження завдань студентами та їх перевірку викладачами, використовує GridFS для зберігання файлів завдань.

4. Модуль успішності:

Управляє оцінками студентів, дозволяє викладачам вводити та редагувати оцінки, а студентам - переглядати свої результати.

2.1.3 Робота з базою даних

Для зберігання даних використовується MongoDB, що є документо-орієнтованою NoSQL базою даних, яка забезпечує гнучке зберігання інформації та

високу продуктивність при роботі з великими обсягами даних. У цій системі використовуються кілька колекцій, кожна з яких відповідає за зберігання специфічної інформації про користувачів, групи, завдання, файли та інші сутності. Далі описується, як ці колекції взаємодіють між собою та з іншими компонентами системи.

Основні принципи роботи з базою даних

1. Децентралізоване зберігання даних:

- Кожна сутність має свою власну колекцію, що дозволяє ефективно керувати даними та забезпечувати високу продуктивність запитів.

2. Використання ідентифікаторів:

- Для встановлення зв'язків між сутностями використовуються ідентифікатори (ObjectId), що дозволяє уникнути дублювання даних і спрощує структуру бази даних.

3. Зберігання великих файлів:

- Для зберігання великих файлів, таких як завдання студентів, використовується GridFS, який дозволяє розбивати файли на фрагменти (чанки) і зберігати їх у двох колекціях: task.bucket та task.chunks, а також file.bucket та file.chunks для інших файлів.

Взаємодія з базою даних

1. Реєстрація та аутентифікація користувачів:

- При реєстрації нового користувача (студента або співробітника) інформація про нього зберігається у відповідних колекціях (students або staff). Для аутентифікації використовується хеш пароля, який зберігається у полі passHash.

2. Управління групами:

- Адміністратори можуть створювати, редагувати та видаляти групи. Інформація про групи зберігається у колекції group, де кожна група містить список студентів, що до неї належать.

- Студенти можуть належати до однієї або кількох груп, що відображається через масив ObjectId у полі students колекції group.

3. Управління завданнями:

- Викладачі можуть створювати завдання для груп. Інформація про завдання зберігається у колекції task, де вказується група, до якої належить завдання, та співробітник, який його завантажив.
- Завдання можуть містити додаткові матеріали, які зберігаються у колекції file. Кожен файл може бути прив'язаний до конкретного завдання через поле task.

4. Завантаження та зберігання файлів:

- Студенти можуть завантажувати свої виконані завдання у вигляді файлів, які зберігаються за допомогою GridFS. Дані файлів зберігаються у колекції file, а самі файли розбиваються на чанки та зберігаються у file.bucket і file.chunks.
- Викладачі можуть оцінювати завдання студентів, додаючи оцінки та коментарі до файлів у колекції file.

5. Управління тарифами:

- Інформація про тарифні плани зберігається у колекції tariff, яка містить дані про вартість та інтервали оплати.
- Кожен студент має прив'язаний тарифний план, який визначається через поле tariff у колекції students.

Зв'язки між колекціями забезпечують інтеграцію даних і дозволяють системі ефективно працювати з різними сутностями:

- Один студент має лише один тарифний план, але один тарифний план може використовуватися багатьма студентами.
- Один співробітник може завантажувати багато завдань, але одне завдання завантажується лише одним співробітником.
- Одне завдання належить до однієї групи, але одна група може мати багато завдань.
- Один файл завантажується одним співробітником і може складатися з багатьох частин, зберігаючись у GridFS.

Така структура бази даних забезпечує гнучкість, ефективність та масштабованість системи, дозволяючи легко додавати нові функціональні можливості та інтегруватися з іншими системами.

2.1.4 Взаємодія компонентів системи

Взаємодія між компонентами системи забезпечує цілісність, узгодженість та ефективність роботи всіх частин системи. Компоненти системи включають користувачів (студентів, викладачів, адміністраторів), базу даних, систему управління файлами (GridFS), а також механізми автентифікації та авторизації. У цьому розділі розглядаються основні сценарії взаємодії компонентів системи.

Сценарій 1: Реєстрація користувача

Адміністратор реєструє нового користувача:

- Адміністратор вводить дані нового користувача (ім'я, електронна пошта, пароль, тарифний план) у систему.
- Система перевіряє введені дані на коректність і зберігає інформацію у відповідній колекції бази даних (students або staff).
- База даних підтверджує збереження даних.
- Система надсилає адміністратору підтвердження успішної реєстрації користувача.

Сценарій 2: Аутентифікація користувача

Користувач входить у систему:

- Користувач вводить свої облікові дані (електронна пошта та пароль).
- Система надсилає запит на перевірку облікових даних до бази даних.
- База даних повертає хеш пароля для введеної електронної пошти.
- Система порівнює введений пароль з хешем у базі даних.
- Якщо облікові дані правильні, система генерує JWT (JSON Web Token) та надсилає його користувачеві.
- Користувач отримує токен та використовує його для доступу до захищених ресурсів.

Сценарій 3: Створення групи

Викладач або адміністратор створює нову групу:

- Викладач/адміністратор вводить назву групи та вибирає студентів, які до неї належать.
- Система зберігає інформацію про групу у колекції group.
- База даних підтверджує збереження даних.
- Система надсилає підтвердження викладачеві/адміністратору про успішне створення групи.

Сценарій 4: Завантаження завдання

Викладач завантажує нове завдання:

- Викладач вибирає групу та завантажує файл із завданням.
- Система зберігає метадані завдання у колекції task та файл у GridFS.
- GridFS розбиває файл на чанки та зберігає їх у відповідних колекціях (task.bucket та task.chunks).
- База даних підтверджує збереження метаданих та файлу.
- Система надсилає підтвердження викладачеві про успішне завантаження завдання.

Сценарій 5: Виконання завдання студентом

Студент завантажує виконане завдання:

- Студент вибирає завдання та завантажує файл із виконаною роботою.
- Система зберігає метадані виконаного завдання у колекції file та файл у GridFS.
- GridFS розбиває файл на чанки та зберігає їх у відповідних колекціях (file.bucket та file.chunks).
- База даних підтверджує збереження метаданих та файлу.
- Система надсилає підтвердження студенту про успішне завантаження виконаного завдання.

Сценарій 6: Оцінювання завдання

Викладач оцінює завдання:

- Викладач вибирає виконане завдання та додає оцінку й коментар.
- Система оновлює метадані виконаного завдання у колекції file.
- База даних підтверджує оновлення даних.
- Система надсилає студенту сповіщення про оцінку завдання.

Сценарій 7: Перегляд оцінок

Студент переглядає свої оцінки:

- Студент запитує список своїх оцінок у системі.
- Система запитує відповідні дані з бази даних.
- База даних повертає інформацію про оцінки.
- Система відображає оцінки студенту.

Ці сценарії ілюструють основні процеси взаємодії між компонентами системи. Вони демонструють, як користувачі можуть реєструватися, входити в систему, створювати та завантажувати завдання, оцінювати роботи та переглядати оцінки, використовуючи ефективну взаємодію між інтерфейсом користувача, системою, базою даних та GridFS для зберігання великих файлів.

2.2 Моделювання вимог

Моделювання вимог є ключовим етапом у розробці програмного забезпечення, що забезпечує розуміння функціональних та нефункціональних вимог до системи. В цьому розділі розглянуто основні вимоги до системи управління навчальним процесом, що включають функціональні вимоги, нефункціональні вимоги та обмеження.

2.2.1 Функціональні вимоги

Функціональні вимоги визначають конкретні функції та завдання, які система повинна виконувати. Основні функціональні вимоги до системи управління навчальним процесом включають:

1. Реєстрація та аутентифікація користувачів:

- Реєстрація нових користувачів: Система повинна надавати можливість реєстрації для нових користувачів, включаючи студентів, викладачів та адміністраторів. При реєстрації необхідно ввести ім'я, адресу електронної пошти та пароль.
- Аутентифікація користувачів: Система повинна забезпечувати аутентифікацію користувачів за допомогою введення облікових даних (електронна пошта та пароль). Використовуються JSON Web Tokens (JWT) для аутентифікації, що забезпечує безпечну передачу даних. Правила створення JWT токенів відповідають специфікаціям, вказаним на iana.org.
- Верифікація токенів: Для перевірки JWT токенів використовується JSON Web Key Set (JWKS), що дозволяє клієнтам перевіряти підпис токена, використовуючи відкриті ключі, які JWKS надає.

2. Управління групами:

- Створення груп: Система повинна дозволяти адміністраторам створювати нові навчальні групи.
- Управління студентами в групах: Система повинна дозволяти додавати студентів до груп та видаляти їх за необхідності.

3. Управління завданнями:

- Створення завдань: Викладачі повинні мати можливість створювати нові завдання для своїх груп. Завдання повинні містити інформацію про дедлайн, коментарі та пов'язані файли.
- Завантаження та зберігання файлів: Система повинна дозволяти викладачам завантажувати файли із завданнями та зберігати їх у GridFS. Це дозволяє ефективно управляти великими файлами.
- Виконання завдань студентами: Студенти повинні мати можливість завантажувати виконані завдання до системи.

4. Оцінювання завдань:

- Оцінювання виконаних завдань: Викладачі повинні мати можливість оцінювати виконані студентами завдання та залишати коментарі. Оцінки та коментарі зберігаються у відповідних колекціях бази даних.
- Перегляд оцінок: Студенти повинні мати можливість переглядати свої оцінки та коментарі до виконаних завдань.

5. Перегляд оцінок:

- Студенти повинні мати можливість переглядати свої оцінки та коментарі до виконаних завдань.

6. Управління тарифними планами:

- Створення та редагування тарифів: Адміністратори повинні мати можливість створювати та редагувати тарифні плани, визначаючи вартість та інтервал оплати.
- Перегляд тарифів: Студенти повинні мати можливість переглядати свій тарифний план, включаючи вартість та дату наступного платежу.

Функціональні вимоги охоплюють основні функції системи, які забезпечують управління студентами, викладачами, завданнями та тарифами. Важливо, щоб ці функції працювали ефективно та безпечно, забезпечуючи користувачам зручний доступ до необхідної інформації та функціоналу.

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики системи, які стосуються її продуктивності, безпеки, надійності, масштабованості та юзабіліті. Вони є важливими для забезпечення якості роботи системи та задоволення потреб користувачів. Нижче детально розглянуто основні нефункціональні вимоги до системи управління навчальним процесом.

1. Продуктивність

1.1 Час відгуку:

Система повинна забезпечувати швидкий час відгуку на запити користувачів. Максимальний час відгуку для більшості операцій (завантаження сторінок, отримання даних) не повинен перевищувати 2 секунд.

Зберігання та отримання файлів у GridFS повинно здійснюватися ефективно, навіть для великих файлів. Максимальний час завантаження файлу розміром до 100 МБ не повинен перевищувати 10 секунд.

1.2 Пропускна здатність:

Система повинна бути здатна обробляти не менше 1000 одночасних запитів користувачів без значного зниження продуктивності.

2. Безпека

2.1Захист даних:

Паролі користувачів повинні зберігатися у хешованому вигляді з використанням сучасних алгоритмів хешування (наприклад, bcrypt).

JWT повинні використовуватися для аутентифікації та авторизації користувачів, що забезпечує безпечну передачу даних. Створення JWT токенів повинно відповідати специфікаціям, вказаним на iana.org.

2.2 Верифікація токенів:

Для перевірки JWT токенів використовується JSON Web Key Set (JWKS), що дозволяє клієнтам перевіряти підпис токена, використовуючи відкриті ключі, які JWKS надає.

2.3 Захист від атак:

Система повинна бути захищена від поширених веб-атак, таких як SQL-ін'єкції, XSS (міжсайтовий скриптинг), CSRF (міжсайтові запити підробки) тощо.

3. Надійність

3.1 Відновлюваність:

Система повинна бути стійкою до збоїв та забезпечувати відновлення після збоїв без втрати даних. Всі критичні операції повинні підтримувати транзакційність, щоб забезпечити цілісність даних.

Регулярне резервне копіювання даних повинно здійснюватися автоматично, щонайменше раз на добу.

3.2 Безперервність роботи:

Система повинна забезпечувати безперервність роботи з мінімальними періодами простою. Допустимий час простою не повинен перевищувати 1% часу роботи на рік.

4. Масштабованість

4.1 Горизонтальна масштабованість:

Система повинна бути спроектована з урахуванням можливості горизонтального масштабування, що дозволяє додавати нові сервери для обробки збільшеного навантаження.

4.2 Підтримка великих обсягів даних:

Система повинна бути здатна ефективно обробляти та зберігати великі обсяги даних, включаючи велику кількість користувачів, файлів та запитів.

5. Юзабіліті

5.1 Інтуїтивний інтерфейс:

Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким у використанні. Всі основні функції повинні бути доступні максимум у три кліки.

5.2 Довідкові матеріали:

Інструкції та довідкові матеріали повинні бути доступні для користувачів, включаючи навчальні відео, документи та відповіді на поширені питання.

6. Сумісність

6.1 Підтримка різних пристроїв:

Система повинна бути сумісною з різними пристроями, включаючи комп'ютери, планшети та смартфони. Веб-інтерфейс повинен бути адаптивним і коректно відображатися на різних екранах.

6.2 Підтримка різних браузерів:

Система повинна підтримувати різні веб-браузери, включаючи Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Функціональність системи повинна бути однаково доступною у всіх підтримуваних браузерах.

7. Підтримка великих файлів

7.1 Завантаження та зберігання файлів:

Система повинна підтримувати можливість завантаження та зберігання великих файлів, використовуючи GridFS для ефективного управління такими файлами. Розмір файлів може досягати декількох гігабайтів.

7.2 Фрагментація файлів:

Великі файли повинні автоматично розбиватися на частини (чанки) для зберігання, що забезпечує ефективне управління та доступ до файлів.

Нефункціональні вимоги є критично важливими для забезпечення надійної, безпечної, масштабованої та зручної у використанні системи. Виконання цих вимог допомагає забезпечити високу якість програмного забезпечення та задовольнити потреби кінцевих користувачів.

2.2.3 Обмеження

Обмеження визначають рамки та умови, в яких розробляється і функціонує система управління навчальним процесом. Вони можуть включати технічні, організаційні, правові та інші види обмежень, які можуть впливати на проектування, розробку, впровадження та експлуатацію системи.

Технічні обмеження

1. Обмеження бази даних:

- Модель даних: Використання MongoDB з певною структурою даних накладає обмеження на типи операцій, які можна ефективно виконувати. Наприклад, MongoDB є документно-орієнтованою базою даних, що робить її менш підходящою для складних реляційних запитів.
- Розмір даних: Хоча MongoDB може зберігати великі обсяги даних, є обмеження на розмір документів та колекцій. Зокрема, максимальний розмір документа в MongoDB становить 16 МБ, що вимагає використання GridFS для зберігання більших файлів.
- Обмеження GridFS: Хоча GridFS дозволяє зберігати великі файли, його використання додає складність у управлінні та доступі до файлів, що може вплинути на продуктивність системи при роботі з дуже великими обсягами даних.

2. Обмеження інфраструктури:

- Пропускна здатність мережі: Продуктивність системи може бути обмежена пропускною здатністю мережі, особливо при передачі великих файлів. Це може призвести до затримок у завантаженні та скачуванні файлів.
- Потужність серверів: Продуктивність та масштабованість системи залежать від потужності серверів, на яких вона розгорнута. Недостатня потужність може обмежити здатність системи обробляти велику кількість одночасних запитів.

3. Сумісність з іншими системами:

- Інтеграція: Система повинна бути здатна інтегруватися з існуючими освітніми платформами та сервісами, що може вимагати дотримання певних протоколів та стандартів. Це може обмежити вибір технологій та підходів при розробці.
- API: Використання зовнішніх API для авторизації та аутентифікації (наприклад, OAuth, JWKS) може вимагати дотримання специфічних вимог та обмежень, встановлених провайдерами цих API.

Організаційні обмеження

1. Часові обмеження:

- Терміни розробки: Проект має жорсткі терміни, які вимагають завершення певних етапів розробки до визначених дат. Це може обмежити можливості для детального тестування та оптимізації системи.
- Графік впровадження: Впровадження системи може бути заплановано на певні періоди, наприклад, між навчальними семестрами, що обмежує доступний час для налаштування та міграції даних.

Правові обмеження

1. Захист даних:

- Відповідність нормативним вимогам: Система повинна відповідати нормативним вимогам щодо захисту персональних даних, таким як GDPR (General Data Protection Regulation) у Європейському Союзі. Це вимагає реалізації певних функцій, таких як анонімізація даних, можливість видалення даних користувачів на їх запит та забезпечення безпеки зберігання та передачі даних.
- Політики конфіденційності: Необхідно забезпечити відповідність політикам конфіденційності, встановленим організацією або навчальним закладом, який використовує систему. Це може обмежити можливості для збору та аналізу певних типів даних.

2. Ліцензійні обмеження:

- Використання відкритого програмного забезпечення: Використання відкритого програмного забезпечення може підпадати під обмеження ліцензій, такі як GPL, MIT, Apache та інші, які визначають умови використання, модифікації та розповсюдження програмного коду.
- Патенти: Необхідно враховувати можливість порушення патентів при використанні певних технологій або алгоритмів, що може вимагати придбання ліцензій або зміни реалізації.

Обмеження проекту є важливими для розуміння контексту, в якому він реалізується, та для забезпечення відповідності всім вимогам і стандартам, що встановлені. Врахування цих обмежень на ранніх етапах проекту допомагає уникнути проблем у майбутньому та забезпечити успішну реалізацію системи.

2.3 Проектування структури бази даних

Проектування структури бази даних є важливим етапом розробки системи управління навчальним процесом для школи «Школа східних мов та культури». Цей розділ описує деталі створення та організації баз даних, що забезпечують ефективне зберігання та обробку даних.

2.3.1 Вибір бази даних

Для проекту обрано NoSQL базу даних MongoDB з використанням GridFS для зберігання великих файлів. MongoDB забезпечує високу продуктивність, гнучкість у зберіганні даних та можливість горизонтального масштабування, що є критичними факторами для системи.

2.3.2 Структура бази даних

Проектування структури баз даних включає створення різних колекцій, які будуть зберігати різні типи даних, що необхідні для функціонування системи.

1. Колекція: students

Ця колекція зберігає інформацію про студентів.

Поля:

- _id: ObjectId (унікальний ідентифікатор студента)
- name: String (ім'я студента)
- mail: String (адреса електронної пошти студента)
- passHash: String (хеш пароля студента)

- tariff: ObjectId (ідентифікатор тарифного плану студента)
- nextPayment: Date (дата наступного платежу студента)

2. Колекція: tariff

Ця колекція містить інформацію про тарифні плани.

Поля:

- _id: ObjectId (унікальний ідентифікатор тарифного плану)
- cost: Number (вартість тарифного плану)
- interval: Number (інтервал оплати тарифного плану)

3. Колекція: staff

Ця колекція зберігає інформацію про співробітників школи.

Поля:

- _id: ObjectId (унікальний ідентифікатор співробітника)
- name: String (ім'я співробітника)
- mail: String (адреса електронної пошти співробітника)
- passHash: String (хеш пароля співробітника)
- permission: String (права співробітника: адміністратор або викладач)

4. Колекція: group

Ця колекція зберігає інформацію про навчальні групи.

Поля:

- _id: ObjectId (унікальний ідентифікатор групи)
- name: String (назва групи)
- students: Array of ObjectId (масив ідентифікаторів студентів, які належать до групи)

5. Колекція: task

Ця колекція зберігає інформацію про завдання, які даються студентам.

Поля:

- _id: ObjectId (унікальний ідентифікатор завдання)
- group: ObjectId (ідентифікатор групи, до якої належить завдання)
- uploader: ObjectId (ідентифікатор співробітника, який завантажив завдання)

- deadline: Number (дата дедлайну завдання)
- comment: String (коментар до завдання)
- file: ObjectId (ідентифікатор файлу з завданням)

6. Колекція: file

Ця колекція зберігає інформацію про файли, які завантажуються до системи.

Поля:

- _id: ObjectId (унікальний ідентифікатор файлу)
- uploader: ObjectId (ідентифікатор співробітника, який завантажив файл)
- comment: String (коментар до файлу)
- grade: Number (оцінка виконаної роботи)
- task: ObjectId (ідентифікатор завдання, для якого надано відповідь)
- files: Array of ObjectId (ідентифікатори завантажених відповідей)

7. GridFS Колекції для завдань (task.bucket та task.chunks)

Ці колекції використовуються для зберігання великих файлів завдань.

Колекція: task.bucket

Поля:

- _id: ObjectId (унікальний ідентифікатор)
- length: Number (розмір файлу в байтах)
- chunkSize: Number (розмір одного чанку в байтах, за замовчуванням 255 кілобайт)
- uploadDate: Number (дата завантаження файлу)
- filename: String (назва файлу)
- contentType: String (тип MIME файлу)

Колекція: task.chunks

Поля:

- _id: ObjectId (унікальний ідентифікатор)
- files_id: ObjectId (ідентифікатор task.bucket документу, до якого належить чанк)
- n: Number (порядковий номер чанку)
- data: Binary (сама частина файлу у бінарному вигляді)

8. GridFS Колекції для файлів (file.bucket та file.chunks)

Ці колекції використовуються для зберігання великих файлів відповідей студентів.

Колекція: file.bucket

Поля:

- _id: ObjectId (унікальний ідентифікатор)
- length: Number (розмір файлу в байтах)
- chunkSize: Number (розмір одного чанку в байтах, за замовчуванням 255 кілобайт)
- uploadDate: Number (дата завантаження файлу)
- filename: String (назва файлу)
- contentType: String (тип MIME файлу)

Колекція: file.chunks

Поля:

- _id: ObjectId (унікальний ідентифікатор)
- files_id: ObjectId (ідентифікатор file.bucket документу, до якого належить чанк)
- n: Number (порядковий номер чанку)
- data: Binary (сама частина файлу у бінарному вигляді)

Для демонстрації вигляду бази даних загалом, наведено діаграму на рисунку 1:

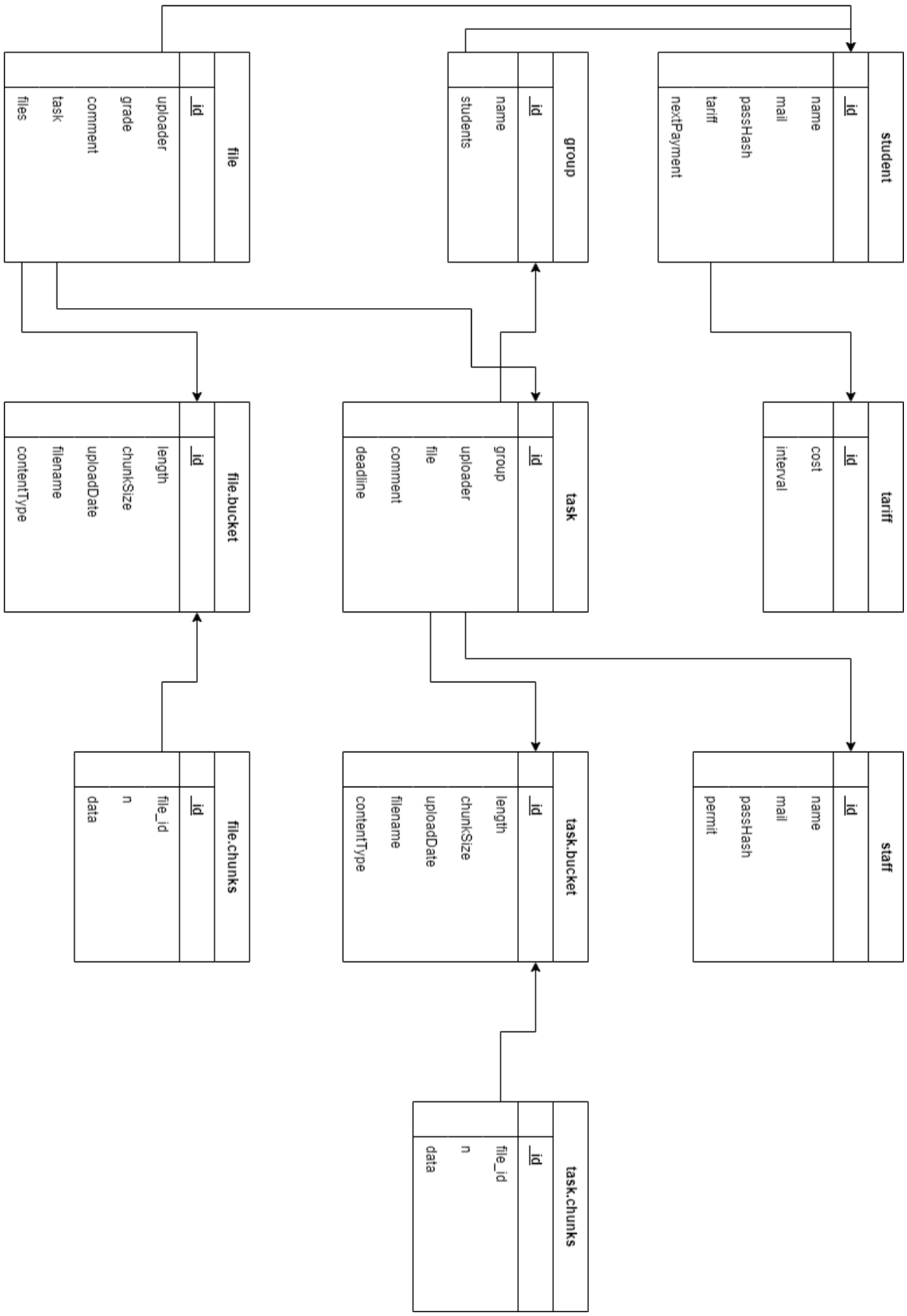


Рис 2.1. Схема бази даних

2.3.3 Зв'язки між колекціями

При проектуванні структури баз даних важливо враховувати зв'язки між колекціями, щоб забезпечити цілісність даних та ефективне виконання запитів.

- Один студент може мати лише один тарифний план.
- Один тарифний план може використовуватися багатьма студентами.
- Один співробітник може завантажувати багато завдань.
- Одне завдання може мати лише одного співробітника, який його завантажив.
- Одне завдання може належати лише до однієї групи.
- Один файл може завантажуватися лише одним співробітником.
- Один файл може складатися з багатьох частин.
- Один студент може належати лише до однієї групи.
- Одна група може мати багато студентів.
- Одне завдання може мати багато матеріалів.
- Один курс може мати одного викладача.
- Один курс може мати багато матеріалів.

2.3.4 Резервне копіювання та відновлення даних

У рамках проекту використовуються найсучасніші методи резервного копіювання та відновлення даних з метою забезпечення безперебійної роботи системи. Регулярно проводиться створення резервних копій бази даних, що дозволяє зберегти інформацію в разі виникнення непередбачених ситуацій, таких як системні збої або кібератаки. Для мінімізації ризиків втрати даних резервні копії зберігаються в безпечних місцях з обмеженим доступом, а також проводиться регулярна перевірка їх цілісності та ефективності.

Зокрема, використовуються автоматизовані інструменти для регулярного створення та зберігання резервних копій, що дозволяє забезпечити ретельне дотримання графіку резервування без значних втрат часу або зусиль. При цьому

велика увага приділяється забезпеченню безпеки резервних копій, щоб уникнути можливості їх несанкціонованого доступу або зміни.

Крім того, для забезпечення ефективності процесу відновлення даних систематично проводяться тести та перевірки резервних копій. Це дозволяє не лише переконатися в правильності налаштувань резервного копіювання, але й відповідно реагувати на будь-які проблеми або неполадки в системі резервного копіювання до того, як вони стануть критичними.

2.3.5 Безпека даних

В проекті відводиться велика увага безпеці даних, оскільки вони є найціннішим активом системи. Для забезпечення цілісності, конфіденційності та доступності інформації використовуються різноманітні заходи та технології безпеки.

Шифрування даних є одним із основних методів захисту інформації, і в проекті використовуються сучасні алгоритми шифрування як для передачі, так і для зберігання даних. Це дозволяє захистити інформацію від несанкціонованого доступу навіть у випадку проникнення у систему.

Контроль доступу забезпечується за допомогою механізмів аутентифікації та авторизації, які обмежують доступ до даних лише авторизованим користувачам. При цьому реалізуються різні рівні доступу в залежності від ролі користувача, що дозволяє точно налаштувати права доступу та уникнути недозволених операцій з даними.

Для відстеження та аналізу дій користувачів в системі використовуються журнали аудиту. Вони фіксують всі дії, що виконуються в системі, що дозволяє оперативно виявляти та реагувати на можливі загрози безпеці даних. Такий підхід забезпечує не лише захист інформації, але й можливість аналізувати події та вдосконалювати систему безпеки на основі зібраної інформації.

2.4 Проектування інтерфейсу користувача

2.4.1 Вимоги юзабіліті

При проектуванні інтерфейсу користувача важливо враховувати різні аспекти, які забезпечують його зручність та ефективність. Нижче детально розглянуто кожну з цих вимог з конкретними прикладами:

1. Інтуїтивність:

Інтерфейс повинен бути легким у використанні та зрозумілим для користувача навіть без додаткового навчання. Наприклад:

- Меню навігації повинно мати зрозумілу структуру та найменування пунктів, що легко інтерпретуються користувачем.
- Кнопки та елементи керування мають мати зрозумілі підказки або піктограми, які передають їхнє призначення. Наприклад, кнопка "Вхід" або "Вихід".

2. Простота навігації:

Приклад: у системі головне меню знаходиться в лівій частині екрану і містить основні розділи, такі як "Література", "Завдання", тощо. Користувач може легко переходити між цими розділами, використовуючи меню, яке завжди доступне.

3. Чіткість та прозорість інформації:

Приклад: під час створення нового завдання, всі необхідні поля для заповнення (наприклад, назва завдання, дедлайн) чітко підписані і розташовані в логічному порядку. Це дозволяє користувачу швидко зрозуміти, яку інформацію він повинен ввести.

4. Інформативність елементів інтерфейсу:

Приклад: у списку груп користувач може бачити загальну кількість студентів у кожній групі. Це дозволяє швидко оцінити розмір групи без необхідності відкривати кожну групу окремо.

5. Мінімальна кількість дій для досягнення мети:

Приклад: для завантаження файлу з завданням користувачу потрібно всього лише кілька клацань мишею: обере потрібне завдання, натисне кнопку "Завантажити файл", обере файл зі свого пристрою та підтвердить завантаження.

6. Консистентність:

Приклад: у всіх формах інтерфейсу кнопки "Зберегти" та "Відмінити" розташовані у тому ж місці, що дозволяє користувачу легко знаходити їх, незалежно від контексту використання.

7. Сумісність з різними пристроями та браузерами:

Система повинна корректно відображатися на різних пристроях (комп'ютерах, планшетах, смартфонах) та працювати в усіх популярних веб-браузерах (Chrome, Firefox, Safari, Edge).

8. Доступ до будь-якої інформації максимум у три кліки:

Користувач повинен мати можливість знайти будь-яку інформацію або функціонал системи за максимум три кліки мишею чи дотиком екрану.

9. Можливість завантажувати великі за розміром файли:

Система повинна підтримувати завантаження та обробку файлів великого розміру забезпечуючи швидкий та безперебійний процес завантаження, навіть у випадку з великими обсягами даних. Такий функціонал дозволить користувачам комфортно використовувати систему навіть при роботі з великими файлами, наприклад, завданнями, які містять багато матеріалів або великі документи.

2.4.2 Сценарії виконання задач

Цей розділ описує основні сценарії взаємодії користувача з системою, включаючи послідовність дій та очікувані результати для кожної задачі.

1. Вхід у систему:
   - Користувач: Студент або викладач.
   - Дії: Введення ідентифікатора користувача та пароля.
   - Результат: Успішний вхід у систему, перехід на головну сторінку.

2. Перегляд груп:
   - Користувач: Викладач.
   - Дії: Обирає опцію "Групи" у головному меню.
   - Результат: Перегляд списку груп.
3. Завантаження завдання:
   - Користувач: Викладач.
   - Дії: Вибирає опцію "Створити завдання", обирає групу, завантажує файл завдання та встановлює дедлайн.
   - Результат: Створення нового завдання, яке студенти зможуть переглядати та завантажувати.
4. Завантаження відповіді на завдання:
   - Користувач: Студент.
   - Дії: Обирає завдання зі списку, завантажує свою відповідь.
   - Результат: Успішне завантаження відповіді, яка буде доступна для перегляду викладачем.
5. Перегляд успішності:
   - Користувач: Студент.
   - Дії: Обирає опцію "Успішність" у своєму профілі.
   - Результат: Перегляд списку оцінок та результатів виконаних завдань.
6. Перегляд та оцінювання відповіді на завдання:
   - Користувач: Викладач.
   - Дії: Вибирає завдання зі списку, переглядає завантажені відповіді, додає оцінку та коментарі.
   - Результат: Збереження оцінки та коментарів, доступність результатів для студентів.

Ці сценарії надають уявлення про типові дії користувачів у системі та відповідні очікувані результати після виконання кожної дії.

2.4.3 Вибір елементів UI

В цьому розділі розглянуто вибір елементів користувацького інтерфейсу (UI) з метою забезпечення зручності використання системи. Елементи UI повинні бути інтуїтивно зрозумілими та легкими у використанні для користувачів різного рівня експертизи. Нижче наведено опис основних елементів UI з прикладами коду, що використовують React компонентів та бібліотеку react-bootstrap:

1. Меню навігації:

Опис: Меню навігації розміщено у лівій частині інтерфейсу як виїзджаюче меню, яке розкривається при наведенні курсора. Воно містить основні розділи системи, такі як "Головна", "Групи", "Завдання" та "Успішність".

Призначення: Забезпечує швидкий доступ до основних функцій системи.

Приклад коду:

```
import React, { useState } from 'react';
import { Navbar, Nav, Container } from 'react-bootstrap';

const NavigationMenu = () => {
  const [expanded, setExpanded] = useState(false);

  return (
    <Navbar bg="light" expand="lg" expanded={expanded}>
      <Container>
        <Navbar.Brand href="#home">Школа східних мов та культури</Navbar.Brand>
        <Navbar.Toggle aria-controls="basic-navbar-nav" onClick={() =>
setExpanded(!expanded)} />
        <Navbar.Collapse id="basic-navbar-nav">
          <Nav className="me-auto">
            <Nav.Link href="#home">Головна</Nav.Link>
            <Nav.Link href="#groups">Групи</Nav.Link>
            <Nav.Link href="#tasks">Завдання</Nav.Link>
            <Nav.Link href="#performance">Успішність</Nav.Link>
          </Nav>
        </Navbar.Collapse>
      </Container>
    </Navbar>
  );
};
```

2. Форма входу:

Опис: Форма входу розташована на головній сторінці системи. Вона складається з полів для введення ідентифікатора користувача та пароля, а також кнопки "Увійти".

Призначення: Дозволяє користувачам ввійти у свої облікові записи для отримання доступу до функцій системи.

Приклад коду:

```
import React from 'react';
import { Form, Button } from 'react-bootstrap';

const LoginForm = () => {
  return (
    <Form>
      <Form.Group controlId="formBasicEmail">
        <Form.Label>Email address</Form.Label>
        <Form.Control type="email" placeholder="Enter email" />
      </Form.Group>

      <Form.Group controlId="formBasicPassword">
        <Form.Label>Password</Form.Label>
        <Form.Control type="password" placeholder="Password" />
      </Form.Group>

      <Button variant="primary" type="submit">
        Submit
      </Button>
    </Form>
  );
};
```

3. Список груп:

Опис: Відображає перелік доступних груп. Кожний елемент списку містить назву групи та додаткову інформацію.

Призначення: Дозволяє викладачам швидко знайти необхідну групу для перегляду або редагування.

Приклад коду:

```
import React from 'react';
import { ListGroup } from 'react-bootstrap';

const GroupList = ({ groups }) => {
  return (
    <ListGroup>
      {groups.map((group, index) => (
        <ListGroup.Item key={index}>{group.name}</ListGroup.Item>
      ))}
    </ListGroup>
  );
};
```

4. Форма завантаження завдання:

Опис: Форма, що дозволяє викладачам створювати нові завдання. Включає поля для вибору групи, завантаження файлу завдання та встановлення дедлайну.

Призначення: Надає можливість викладачам легко створювати та редагувати завдання для студентів.

Приклад коду:

```
import React from 'react';
import { Form, Button } from 'react-bootstrap';

const TaskUploadForm = () => {
  return (
    <Form>
      <Form.Group controlId="formBasicEmail">
        <Form.Label>Task Name</Form.Label>
        <Form.Control type="text" placeholder="Enter task name" />
      </Form.Group>

      <Form.Group controlId="formBasicPassword">
        <Form.Label>Task Description</Form.Label>
        <Form.Control as="textarea" rows={3} placeholder="Enter task description" />
      </Form.Group>

      <Form.Group controlId="formFile" className="mb-3">
        <Form.Label>Upload Task File</Form.Label>
        <Form.Control type="file" />
      </Form.Group>

      <Button variant="primary" type="submit">
        Submit
      </Button>
    </Form>  );};
```

5. Список завдань:

Опис: Перелік завдань для конкретної групи. Кожен елемент містить назву завдання, дедлайн та іншу інформацію.

Призначення: Дозволяє студентам швидко знаходити та завантажувати відповіді на завдання.

Приклад коду:

```
import React from 'react';
import { ListGroup } from 'react-bootstrap';

const TaskList = ({ tasks }) => {
  return (
    <ListGroup>
      {tasks.map((task, index) => (
        <ListGroup.Item key={index}>{task.name}</ListGroup.Item>
      ))}
    </ListGroup>
  );
};
```

6. Форма завантаження відповіді:

Опис: Форма, що дозволяє студентам завантажувати відповіді на завдання. Включає поле для завантаження файлу та кнопку "Завантажити".

Призначення: Надає студентам можливість легко завантажувати свої відповіді для подальшої перевірки викладачем.

Приклад коду:

```
import React from 'react';
import { Form, Button } from 'react-bootstrap';

const AnswerUploadForm = () => {
  return (
    <Form>
      <Form.Group controlId="formFile" className="mb-3">
        <Form.Label>Upload Answer File</Form.Label>
        <Form.Control type="file" />
      </Form.Group>

      <Button variant="primary" type="submit">
        Submit
      </Button>
    </Form>
  );
};
```

7. Список оцінок:

Опис: Відображає список оцінок студента за завдання. Кожен елемент містить назву завдання, оцінку та коментар викладача.

Призначення: Дозволяє студентам переглядати свої результати та отримані оцінки.

Приклад коду:

```
import React from 'react';
import { ListGroup } from 'react-bootstrap';

const GradeList = ({ grades }) => {
  return (
    <ListGroup>
      {grades.map((grade, index) => (
        <ListGroup.Item key={index}>Grade: {grade}</ListGroup.Item>
      ))}
    </ListGroup>
  );
};

export default GradeList;
```

Вибрані елементи UI мають бути зрозумілими, доступними та функціональними для всіх користувачів системи, щоб забезпечити комфортну взаємодію з нею.

2.5 Забезпечення безпеки даних та конфіденційності.

У даному розділі детально описані заходи щодо захисту даних та забезпечення конфіденційності користувачів.

2.5.1 Захист паролів користувачів

Захист паролів користувачів є критично важливим аспектом забезпечення безпеки в системі. Далі розглянуто заходи, яких можна вжити для забезпечення безпеки паролів користувачів.

1. Хешування паролів: Паролі користувачів повинні бути збережені у вигляді хешу в базі даних, а не у чистому текстовому форматі. Хешування паролів забезпечує безпеку в разі, якщо база даних стане доступною зловмисникам. Зазвичай для хешування паролів використовуються криптографічні алгоритми, такі як bcrypt, які забезпечують додатковий рівень захисту.

2. Сіль (salt): Додавання солі до паролів перед хешуванням підвищує безпеку. Сіль - це випадкова послідовність, яка додається до паролю перед хешуванням. Вона запобігає використанню технік, таких як атаки за словником або рейнджами, і ускладнює процес розшифрування.

3. Криптографічна міцність: Важливо використовувати криптографічно міцні алгоритми хешування, такі як bcrypt. Це гарантує, що паролі залишаються надійними, навіть у випадку виявлення хешів.

4. Шифрування відкритих ключів (HTTPS): Під час передачі паролів через мережу важливо використовувати шифрування за допомогою протоколу HTTPS. Це захищає паролі від перехоплення зловмисниками під час їх передачі по мережі.

5. Політики паролів: Рекомендується встановлювати вимоги до паролів, такі як мінімальна довжина, використання різних типів символів, та періодична зміна паролів. Це зменшує ймовірність успішного використання атак за перебором.

6. Автоматичне блокування облікових записів: Застосунок може блокувати облікові записи користувачів після декількох невдалих спроб входу, щоб уникнути атак за перебором.

7. Моніторинг активності облікового запису: Система може вести журнал доступу та моніторити активність облікового запису, щоб виявити незвичайну або підозрілу активність.

Ці заходи забезпечують надійний захист паролів користувачів і допомагають зменшити ризики безпеки в системі.

Паролі користувачів повинні бути збережені у хешованому вигляді з використанням сучасних алгоритмів хешування, наприклад, bcrypt. Це забезпечить

високий рівень безпеки паролів, оскільки хешовані значення практично неможливо розшифрувати знову в текстовий формат.

Приклад використання бібліотеки bcrypt для хешування паролів у Node.js:

```
const bcrypt = require('bcrypt');

// Генерація хешу паролю
const saltRounds = 10;
const password = 'mypassword';
bcrypt.hash(password, saltRounds, function(err, hash) {
  // зберегти hash у базі даних
});

// Перевірка паролю
const plainTextPassword = 'mypassword';
bcrypt.compare(plainTextPassword, hash, function(err, result) {
  // result - true, якщо паролі співпадають
});
```

2.5.2 Використання JWT для аутентифікації та авторизації

Використання JWT для аутентифікації та авторизації є ключовим аспектом забезпечення безпеки веб-додатків. Детальніше:

1. JWT (JSON Web Tokens): JWT - це стандарт, який описує формат передачі безпечних інформаційних токенів у вигляді JSON-об'єктів. Токени JWT складаються з трьох частин: заголовка, вмісту (пейлоаду) та підпису. Підпис забезпечує перевірку цілісності даних та перевірку автентичності відправника.

2. Аутентифікація за допомогою JWT: Після успішної аутентифікації користувача (наприклад, з використанням логіну та пароля), сервер створює JWT токен і відправляє його клієнту. Цей токен містить інформацію про користувача та деякі метадані. При подальших запитах до сервера клієнт повинен включати цей токен у заголовку запиту для перевірки автентичності.

3. Перевірка токенів: Сервер перевіряє цілісність та автентичність токена, а також перевіряє, чи є він дійсним та чи має користувач відповідні права доступу. Після успішної перевірки сервер розуміє, що запит був відправлений автентичним користувачем.

4. Зберігання інформації у токенах: Токени JWT можуть містити корисну інформацію, таку як ідентифікатор користувача, роль користувача, строк дії токена тощо. Ця інформація дозволяє серверу приймати рішення щодо надання доступу до ресурсів.

5. Авторизація за допомогою JWT: Після успішної аутентифікації сервер може використовувати інформацію з токена для прийняття рішення щодо доступу користувача до певних ресурсів або дій. Наприклад, сервер може перевірити роль користувача та відповідні дозволи перед наданням доступу до конкретного ресурсу.

6. Забезпечення безпеки токенів: Токени JWT повинні бути збережені в безпечному місці та зашифровані, якщо вони містять конфіденційну інформацію. Також важливо встановити обмеження на строк дії токенів для запобігання можливим атакам.

JWT токени повинні використовуватися для аутентифікації та авторизації користувачів, що забезпечує безпечну передачу даних. Створення JWT токенів повинно відповідати специфікаціям, вказаним на iana.org.

Приклад створення JWT токену у Node.js з використанням бібліотеки jsonwebtoken:

```
const jwt = require('jsonwebtoken');

const payload = { username: 'user', isAdmin: true };
const secretKey = 'secretKey';
const options = { expiresIn: '1h' };

const token = jwt.sign(payload, secretKey, options);
// Повернути цей токен користувачеві
```

2.5.3 Верифікація токенів за допомогою JWKS

Верифікація токенів за допомогою JSON Web Key Set (JWKS) є важливим етапом в забезпеченні безпеки веб-додатків, особливо при використанні JWT для аутентифікації та авторизації. Розглянемо цей процес детальніше:

1. Що таке JWKS: JWKS - це набір відкритих ключів, які використовуються для перевірки цілісності та автентичності підписаних JWT токенів. JWKS зазвичай містить набір RSA або ECDSA ключів.

2. Публічні ключі: JWKS містить публічні ключі, які можуть бути використані для перевірки підпису токенів. Публічні ключі витягуються з JWKS та використовуються для верифікації токенів.

3. Перевірка підпису токенів: При отриманні JWT токена, сервер витягує ідентифікатор ключа (kid) з заголовка токена. За допомогою цього ідентифікатора, сервер шукає відповідний публічний ключ у JWKS.

4. Верифікація токенів: Після отримання публічного ключа, сервер використовує його для перевірки підпису токена. Якщо підпис вірний та токен дійсний, сервер приймає токен як автентичний.

5. Оновлення JWKS: JWKS може періодично оновлюватися, щоб забезпечити безпеку. Це може включати регулярне оновлення ключів або повний обмін JWKS.

6. Захист від атак: Використання JWKS допомагає уникнути атак, таких як атаки на середовище виконання, шляхом забезпечення безпеки процесу верифікації та перевірки підпису JWT токенів.

Використання JWKS для верифікації токенів дозволяє забезпечити безпеку та надійність механізму аутентифікації та авторизації у веб-додатках.

Приклад верифікації JWT токену з використанням JWKS у Node.js:

```
const jwksClient = require('jwks-rsa');
const jwt = require('jsonwebtoken');

const client = jwksClient({
  jwksUri: 'https://example.com/.well-known/jwks.json'
});

function getKey(header, callback) {
  client.getSigningKey(header.kid, function(err, key) {
    const signingKey = key.publicKey || key.rsaPublicKey;
    callback(null, signingKey);
  });
}

const options = {
```

```
  audience: 'your_audience',
  issuer: 'https://example.com/',
  algorithms: ['RS256']
};

const token = 'your_jwt_token';

jwt.verify(token, getKey, options, function(err, decoded) {
  console.log(decoded);
});
```

2.5.4 Захист від веб-атак

Захист від веб-атак включає в себе ряд заходів, спрямованих на запобігання та виявлення різних видів атак, таких як SQL-ін'єкції, XSS (міжсайтовий скриптинг), CSRF (міжсайтові запити підробки) та інші. Детальніше про ці заходи:

1. Захист від SQL-ін'єкцій:

Оскільки MongoDB використовує NoSQL підхід і не використовує SQL для взаємодії з даними, SQL-ін'єкції не є проблемою. Проте, незабезпечене введення даних може призвести до інших безпекових проблем, таких як NoSQL-ін'єкції.

Приклад коду:

```
const collection = db.collection('users');
const username = req.body.username; // Припустимо, що username введений користувачем

// Використання параметризованого запиту
const user = await collection.findOne({ username: username });
```

2. Захист від XSS:

Використання Content Security Policy (CSP) для обмеження джерел виконання скриптів на сторінках.

Екранування та екземплярність (escaping) виведених на сторінку даних, щоб уникнути виконання небезпечних скриптів.

Приклад коду:

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'">
```

3. Захист від CSRF:

Використання токенів CSRF, які вбудовані в кожний запит форми або AJAX, та перевірка їх при обробці запиту на сервері.

Використання SameSite cookies для обмеження передачі cookies в рамках міжсайтових запитів.

Приклад коду:

```
<form action="/submit-form" method="post">
  <input type="hidden" name="_csrf" value="<%= csrfToken %>">
  <!-- Решта полів форми -->
  <button type="submit">Відправити</button>
</form>
```

4. Захист від інших веб-атак:

Валідація всіх введених даних на стороні клієнта та сервера для виявлення та усунення недопустимих або потенційно небезпечних даних.

Використання HTTPS для захисту передачі даних між клієнтом і сервером, що дозволяє шифрувати дані та забезпечувати конфіденційність і цілісність.

Приклад коду:

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser('secret'));

app.get('/', (req, res) => {
  res.cookie('sessionID', '123', {
    sameSite: 'strict',
    httpOnly: true
  });
  res.send('Cookie встановлено');
});

app.listen(3000, () => {
  console.log('Сервер запущено на порту 3000');
});
```

5. Моніторинг і реагування:

Постійний моніторинг системи на виявлення підозрілих або несподіваних активностей.

Швидке реагування на виявлені загрози, включаючи блокування доступу та відновлення безпеки системи.

Приклад коду:

```
const { body, validationResult } = require('express-validator');

app.post('/register', [
  // Перевірка наявності та правильності введення email
  body('email').isEmail(),
  // Перевірка наявності та мінімальної довжини пароля
  body('password').isLength({ min: 5 })
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  // Реєстрація користувача
});
```

6. Постійне оновлення:

Постійне оновлення системи та всіх її компонентів для виправлення відомих уразливостей та захисту від нових атак.

3 ОПИС РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Структура проекту

Проект складається з двох основних частин: фронтенд (React) і бекенд (Node.js). Кожна частина має свої власні структури каталогів і файлів, що забезпечують зручність розробки та підтримки.

3.1.1 Фронтенд

Фронтенд частина реалізована за допомогою React і має наступну структуру:

- Компоненти: Всі UI-компоненти, включаючи форми, таблиці, кнопки та інші елементи інтерфейсу. Компоненти організовані у відповідних каталогах для забезпечення логічної структури проекту.
- Загальні компоненти (Common Components): Компоненти, що використовуються в різних частинах програми, такі як кнопки, форми, модальні вікна тощо.
- Функціональні компоненти (Functional Components): Компоненти, що відповідають за конкретну функціональність, наприклад, форма реєстрації, таблиця груп, форма редагування профілю тощо.
- Сторінки: Основні сторінки додатку, такі як вхід, реєстрація, особистий кабінет, управління групами і тарифами:
    - LoginPage: Сторінка входу до системи.
    - RegisterPage: Сторінка реєстрації нового користувача.
    - Dashboard: Головна сторінка особистого кабінету користувача.
    - GroupsPage: Сторінка управління навчальними групами.
    - TariffsPage: Сторінка управління тарифними планами.

- Маршрутизація (Routing): Відповідальна за навігацію між сторінками додатку.
- Використання бібліотеки React Router для визначення маршрутів та переходів між сторінками.
- Основні маршрути включають /login, /register, /dashboard, /groups, /tariffs.
- Стан (State Management): Використання Redux для управління станом додатку.
- Redux, структура включає ред'юсери (reducers), дії (actions) та сховище (store).
- Сервіси (Services): Логіка для взаємодії з бекендом через API запити.
- API Service: Модуль для здійснення HTTP-запитів до бекенд-сервера, включаючи методи для аутентифікації, отримання та відправки даних про групи, тарифи тощо.
- Auth Service: Модуль для обробки аутентифікації користувачів, включаючи збереження та видалення токенів, перевірку стану сесії користувача.
- Стилі (Styles): CSS/SCSS файли для стилізації компонентів.
- Global Styles: Глобальні стилі, що застосовуються до всіх компонентів.
- Component-specific Styles: Стилі, що застосовуються до конкретних компонентів.

Фронтенд частина розроблена таким чином, щоб забезпечити максимальну модульність та повторне використання компонентів, що сприяє зручності розробки та підтримки додатку.

3.1.2 Бекенд

Бекенд частина реалізована на Node.js з використанням Express.js і MongoDB через бібліотеку Mongoose. Основні компоненти включають:

- Маршрути (Routes): Визначення всіх API ендпоінтів для взаємодії з фронтендом.

- User Routes: Маршрути, що відповідають за реєстрацію, вхід, вихід та отримання інформації про користувачів.
- Group Routes: Маршрути для управління навчальними групами (створення, редагування, видалення, отримання списку груп).
- Tariff Routes: Маршрути для управління тарифними планами (створення, редагування, видалення, отримання списку тарифів).
- Контролери (Controllers): Логіка обробки запитів і формування відповідей.
- User Controller: Обробка запитів, пов'язаних з користувачами (реєстрація, аутентифікація, отримання профілю).
- Group Controller: Обробка запитів для управління групами (створення, оновлення, видалення груп).
- Tariff Controller: Обробка запитів для управління тарифними планами (створення, оновлення, видалення тарифів).
- Моделі (Models): Визначення структури даних і взаємодії з базою даних за допомогою Mongoose.
- User Model: Схема Mongoose для збереження інформації про користувачів (ім'я, електронна пошта, пароль, роль).
- Group Model: Схема для збереження інформації про групи (назва групи, список учасників).
- Tariff Model: Схема для збереження інформації про тарифні плани (назва тарифу, опис, ціна).
- Сервіси (Services): Допоміжні функції та бізнес-логіка.
- AuthService: Логіка для аутентифікації користувачів, включаючи генерацію та валідацію JWT токенів.
- EmailService: Логіка для надсилання електронних листів (підтвердження реєстрації, відновлення пароля).
- PaymentService: Логіка для обробки платежів (інтеграція з платіжними системами).

- Мідлвери (Middlewares): Обробка запитів, включаючи аутентифікацію та обробку помилок.
- AuthMiddleware: Перевірка аутентифікації користувача за допомогою JWT токенів.
- ErrorHandlerMiddleware: Глобальна обробка помилок, що виникають під час виконання запитів.
- Конфігурація (Configuration): Налаштування проекту, включаючи змінні оточення та конфігураційні файли.
- config.js: Основні налаштування проекту (підключення до бази даних, секретні ключі для JWT, налаштування серверу).

Опис основних файлів та їх функцій:

- app.js: Основний файл додатку, де налаштовуються Express, підключаються маршрути, мідлвери та інші налаштування.
- server.js: Файл для запуску сервера, визначення порту та підключення до бази даних.
- config/config.js: Файл конфігурації, що містить налаштування для підключення до бази даних, секретні ключі та інші параметри.

Опис реалізації основних компонентів:

Маршрути (Routes)

- Файл userRoutes.js визначає ендпоінти для реєстрації, входу та отримання інформації про користувача.
- Файл groupRoutes.js містить маршрути для створення, редагування та видалення навчальних груп.
- Файл tariffRoutes.js забезпечує маршрути для управління тарифами.

Контролери (Controllers)

- userController.js містить функції для реєстрації, входу в систему та управління профілем користувача.
- groupController.js обробляє запити для управління навчальними групами.
- tariffController.js відповідає за запити, пов'язані з тарифними планами.

Моделі (Models)

- userModel.js визначає схему користувача з полями для імені, електронної пошти, пароля та ролі.
- groupModel.js визначає схему для групи, включаючи назву групи та список учасників.
- tariffModel.js визначає схему тарифу з полями для назви, опису та ціни.

Сервіси (Services)

- authService.js реалізує функції для генерації та валідації JWT токенів.
- emailService.js забезпечує функціонал для надсилання електронних листів.
- paymentService.js обробляє логіку платежів.

Мідлвери (Middlewares)

- authMiddleware.js перевіряє JWT токени для захищених маршрутів.
- errorHandlerMiddleware.js обробляє помилки, що виникають під час виконання запитів, і повертає відповідні повідомлення.

3.2 Функціональні компоненти

Функціональні компоненти забезпечують основну логіку роботи системи. Вони включають в себе компоненти для обробки аутентифікації користувачів, управління групами, тарифами та взаємодії з базою даних. Нижче наведено детальний опис кожного з функціональних компонентів.

3.2.1 Аутентифікація користувачів

Аутентифікація користувачів забезпечує безпечний доступ до системи. Основні функції включають реєстрацію нових користувачів, вхід до системи та управління сесіями користувачів.

Реєстрація: Користувач (адміністратор) заповнює форму реєстрації на фронтенді, яка містить поля для імені, електронної пошти та пароля. Після

відправки форми, дані надсилаються на бекенд, де вони обробляються в контролері userController.js. За допомогою userModel.js, новий користувач зберігається в базі даних MongoDB. Якщо все успішно, користувач отримує підтвердження на електронну пошту через emailService.js.

Приклад коду:

```
// userController.js

const User = require('../models/userModel');
const emailService = require('../services/emailService');

exports.registerUser = async (req, res) => {
  try {
    const { name, email, password } = req.body;

    // Create new user
    const newUser = new User({
      name,
      email,
      password
    });

    // Save user to database
    await newUser.save();

    // Send confirmation email
    await emailService.sendConfirmationEmail(email);

    res.status(201).json({ message: 'User registered successfully. Please check your email for confirmation.' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

Вхід до системи: Користувач заповнює форму входу, вводячи електронну пошту та пароль. Дані надсилаються на бекенд, де контролер userController.js перевіряє наявність користувача в базі даних за допомогою userModel.js і валідує пароль. При успішній аутентифікації, користувач отримує JWT токен, який використовується для доступу до захищених маршрутів.

Приклад коду:

```
// userController.js

const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const User = require('../models/userModel');

exports.loginUser = async (req, res) => {
   try {
      const { email, password } = req.body;

      // Find user by email
      const user = await User.findOne({ email });

      if (!user) {
         return res.status(401).json({ message: 'Invalid email or password' });
      }

      // Validate password
      const isPasswordValid = await bcrypt.compare(password, user.password);

      if (!isPasswordValid) {
         return res.status(401).json({ message: 'Invalid email or password' });
      }

      // Generate JWT token
      const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, { expiresIn: '1h'
});

      res.status(200).json({ token });
   } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Internal server error' });
   }
};
```

Управління сесіями: Використання JWT токенів для аутентифікації користувачів на захищених маршрутах. authMiddleware.js перевіряє токен і надає доступ до ресурсів, якщо токен дійсний.

Приклад коду:

```
// authMiddleware.js

const jwt = require('jsonwebtoken');

exports.authenticateUser = (req, res, next) => {
   const token = req.headers.authorization;

   if (!token) {
      return res.status(401).json({ message: 'Unauthorized' });
   }

   try {
      const decodedToken = jwt.verify(token, process.env.JWT_SECRET);
      req.userId = decodedToken.userId;
      next();
   } catch (error) {
      console.error(error);
      res.status(401).json({ message: 'Unauthorized' });
   }
};
```

3.2.2 Управління групами

Компонент управління групами дозволяє користувачам створювати, редагувати, видаляти та переглядати навчальні групи.

- Створення групи: Користувач заповнює форму створення групи, вказуючи назву та учасників групи. Дані надсилаються на бекенд, де контролер groupController.js створює нову групу за допомогою groupModel.js і зберігає її в базі даних.

Приклад коду:

```
// groupController.js

const Group = require('../models/groupModel');

exports.createGroup = async (req, res) => {
   try {
      const { name, members } = req.body;

      // Create new group
      const newGroup = new Group({
```

```
            name,
            members
        });

        // Save group to database
        await newGroup.save();

        res.status(201).json({ message: 'Group created successfully', group: newGroup });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal server error' });
    }
};
```

Редагування групи: Користувач може редагувати інформацію про групу. Дані змін надсилаються на бекенд, де контролер groupController.js оновлює інформацію в базі даних.

Приклад коду:

```
// groupController.js

const Group = require('../models/groupModel');

exports.editGroup = async (req, res) => {
    try {
        const { groupId } = req.params;
        const { name, members } = req.body;

        // Find group by id and update
        const updatedGroup = await Group.findByIdAndUpdate(groupId, { name, members }, { 
new: true });

        res.status(200).json({ message: 'Group updated successfully', group: updatedGroup });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal server error' });
    }
};
```

Видалення групи: Користувач може видалити групу. Запит на видалення надсилається на бекенд, де контролер groupController.js видаляє групу з бази даних.

Приклад коду:

```
// groupController.js

const Group = require('../models/groupModel');

exports.deleteGroup = async (req, res) => {
   try {
      const { groupId } = req.params;

      // Delete group by id
      await Group.findByIdAndDelete(groupId);

      res.status(200).json({ message: 'Group deleted successfully' });
   } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Internal server error' });
   }
};
```

Перегляд груп: Користувач може переглядати список усіх груп. Запит на отримання списку груп надсилається на бекенд, де контролер groupController.js отримує дані з бази даних і повертає їх на фронтенд.

Приклад коду:

```
// groupController.js

const Group = require('../models/groupModel');

exports.getAllGroups = async (req, res) => {
   try {
      // Retrieve all groups from database
      const groups = await Group.find();

      res.status(200).json(groups);
   } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Internal server error' });
   }
};
```

3.2.3 Управління тарифами

Компонент управління тарифами дозволяє адміністраторам створювати, редагувати та видаляти тарифні плани. Принцип роботи такий самий як і у випадку з групами тому прикладів не наведено.

- Створення тарифу: Адміністратор заповнює форму створення тарифного плану, вказуючи назву, опис та ціну. Дані надсилаються на бекенд, де контролер tariffController.js створює новий тариф за допомогою tariffModel.js і зберігає його в базі даних.
- Редагування тарифу: Адміністратор може редагувати інформацію про тариф. Дані змін надсилаються на бекенд, де контролер tariffController.js оновлює інформацію в базі даних.
- Видалення тарифу: Адміністратор може видалити тариф. Запит на видалення надсилається на бекенд, де контролер tariffController.js видаляє тариф з бази даних.
- Перегляд тарифів: Адміністратор може переглядати список усіх тарифних планів. Запит на отримання списку тарифів надсилається на бекенд, де контролер tariffController.js отримує дані з бази даних і повертає їх на фронтенд.

3.2.4 Взаємодія з базою даних

Взаємодія з базою даних реалізована за допомогою бібліотеки Mongoose, яка дозволяє легко працювати з MongoDB.

- Підключення до бази даних: У файлі config.js налаштовується підключення до бази даних MongoDB за допомогою Mongoose. При запуску сервера (файл server.js), встановлюється з'єднання з базою даних.

Приклад коду:

```
// config.js

const mongoose = require('mongoose');

const connectDB = async () => {
    try {
        await mongoose.connect('mongodb://localhost:27017/your_database_name', {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true,
            useFindAndModify: false
        });
        console.log('Connected to MongoDB');
```

```
    } catch (error) {
        console.error('Failed to connect to MongoDB:', error);
    }
};

module.exports = connectDB;
```

- Моделі даних: Всі дані у базі даних зберігаються у відповідних моделях (User, Group, Tariff), визначених у файлах userModel.js, groupModel.js та tariffModel.js.

Приклад коду:

```
// userModel.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    }
});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

- CRUD операції: Моделі Mongoose дозволяють виконувати всі необхідні CRUD операції (створення, читання, оновлення, видалення) для управління даними у базі.

Приклад коду:

```
// userController.js
const User = require('../models/userModel');

// Create new user
exports.createUser = async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const newUser = new User({ name, email, password });
    await newUser.save();
    res.status(201).json(newUser);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Failed to create user' });
  }
};

// Read all users
exports.getAllUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Failed to fetch users' });
  }
};

// Update user by ID
exports.updateUser = async (req, res) => {
  try {
    const { userId } = req.params;
    const { name, email, password } = req.body;
    const updatedUser = await User.findByIdAndUpdate(userId, { name, email, password }, { new: true });
    res.status(200).json(updatedUser);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Failed to update user' });
  }
};

// Delete user by ID
exports.deleteUser = async (req, res) => {
  try {
    const { userId } = req.params;
    await User.findByIdAndDelete(userId);
    res.status(200).json({ message: 'User deleted successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Failed to delete user' });  }};
```

3.3 Опис функціонування програми

Програма побудована з використанням React для фронтенд-частини та Node.js для бекенд-частини, що забезпечує високу ефективність та зручність в роботі з системою. Основні функції системи включають:

3.3.1 Реєстрація та аутентифікація користувачів

Студенти та співробітники мають можливість реєструватися та входити в систему через зручний веб-інтерфейс. Після успішної реєстрації або входу, вони отримують доступ до основних функцій системи.

Для забезпечення безпеки аутентифікації використовується JWT (JSON Web Token). Після входу користувача в систему, йому надається токен, який використовується для автентифікації при кожному запиті до сервера. Це дозволяє зберігати сесії без необхідності зберігання конкретних даних про користувача на сервері.

3.3.2 Управління навчальними групами

Адміністратори мають повний контроль над навчальними групами та можуть створювати, редагувати та видаляти їх через спеціальну адміністративну панель. Це дозволяє ефективно організовувати навчальний процес та керувати групами з урахуванням потреб користувачів.

Викладачі мають можливість додавати студентів до навчальних груп та видаляти їх за необхідності. Це забезпечує зручну та ефективну співпрацю між викладачами та студентами.

3.3.3 Управління тарифними планами

Адміністратори мають можливість створювати та редагувати тарифні плани через інтуїтивно зрозумілий інтерфейс. Це дозволяє швидко адаптувати систему до змінних потреб користувачів та ринкових умов.

Система автоматично розраховує дати наступних платежів для студентів на основі їх обраних тарифних планів. Це допомагає уникнути помилок та забезпечує вчасну оплату послуг, що надаються системою.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Обґрунтування використання обраних видів тестування програми

Для забезпечення надійності та якості розробленої навчальної платформи, було вирішено використовувати декілька видів тестування:

1. Модульне тестування: Воно дозволяє перевірити окремі компоненти системи на правильність їх роботи. Це тестування важливе для виявлення помилок на ранніх стадіях розробки, що знижує вартість їх виправлення.
2. Функціональне тестування: Цей вид тестування спрямований на перевірку функціональності програми відповідно до вимог, щоб забезпечити правильну роботу всіх функцій, заявлених у специфікації.
3. Системне тестування: Воно включає в себе тестування всієї системи як єдиного цілого для забезпечення відповідності всім функціональним та нефункціональним вимогам.

4.2 Тест-кейси з прикладами на Jest

Тест-кейс 1: Реєстрація нового користувача

Мета: Перевірити можливість успішної реєстрації нового користувача.

Кроки:

1. Відкрити сторінку реєстрації.
2. Ввести дійсні дані у поля форми (ім'я, електронна пошта, пароль).
3. Натиснути кнопку "Зареєструватися".

Очікуваний результат: Користувач успішно зареєстрований, відображається повідомлення про успішну реєстрацію.

Результат виконання: Реєстрація пройшла успішно, помилок не виявлено.

Код:

```
const request = require('supertest');
const app = require('../app');

describe('User Registration', () => {
  it('should register a new user successfully', async () => {
    const response = await request(app)
      .post('/register')
      .send({
        name: 'Test User',
        email: 'testuser@example.com',
        password: 'password123'
      });
    expect(response.statusCode).toBe(201);
    expect(response.body.message).toBe('Registration successful');
  });
});
```

Тест-кейс 2: Вхід в систему

Мета: Перевірити можливість входу в систему з правильними обліковими даними.

Кроки:

1. Відкрити сторінку входу.
2. Ввести дійсну електронну пошту та пароль.
3. Натиснути кнопку "Увійти".

Очікуваний результат: Користувач успішно входить в систему, відображається головна сторінка.

Результат виконання: Вхід в систему пройшов успішно, помилок не виявлено.

Код:

```
describe('User Login', () => {
  it('should log in successfully with correct credentials', async () => {
    const response = await request(app)
      .post('/login')
      .send({
        email: 'testuser@example.com',
        password: 'password123'
      });
    expect(response.statusCode).toBe(200);
    expect(response.body.message).toBe('Login successful');
  });
});
```

Тест-кейс 3: Створення нового курсу викладачем

Мета: Перевірити можливість створення нового курсу викладачем.

Кроки:

1. Увійти в систему як викладач.
2. Перейти до розділу "Курси".
3. Натиснути кнопку "Створити новий курс".
4. Заповнити форму створення курсу (назва, опис, матеріали).
5. Натиснути кнопку "Зберегти".

Очікуваний результат: Новий курс успішно створено, відображається у списку курсів.

Результат виконання: Курс успішно створено, помилок не виявлено.

Код:

```
describe('Course Creation', () => {
  it('should create a new course successfully', async () => {
    const response = await request(app)
      .post('/courses')
      .send({
        title: 'New Course',
        description: 'This is a new course',
        materials: 'Some materials'
      })
      .set('Authorization', 'Bearer instructorToken');
    expect(response.statusCode).toBe(201);
    expect(response.body.message).toBe('Course created successfully');
  });
});
```

Тест-кейс 4: Завантаження завдання викладачем

Мета: Перевірити можливість завантаження завдання викладачем.

Кроки:

1. Увійти в систему як викладач.
2. Перейти до розділу "Завдання".
3. Вибрати курс, для якого потрібно створити завдання.
4. Натиснути кнопку "Створити нове завдання".
5. Завантажити файл із завданням та вказати деталі завдання.
6. Натиснути кнопку "Зберегти".

Очікуваний результат: Завдання успішно завантажено, студенти можуть переглядати його.

Результат виконання: Завдання успішно завантажено, помилок не виявлено.

Код:

```
describe('Assignment Upload', () => {
  it('should upload an assignment successfully', async () => {
    const response = await request(app)
      .post('/assignments')
      .field('courseId', 'courseId123')
      .field('title', 'New Assignment')
      .field('description', 'This is a new assignment')
      .attach('file', 'path/to/assignment.pdf') // припустимо, що ви використовуєте multer для завантаження файлів
      .set('Authorization', 'Bearer instructorToken');
    expect(response.statusCode).toBe(201);
    expect(response.body.message).toBe('Assignment uploaded successfully');
  });
});
```

Тест-кейс 5: Виконання завдання студентом

Мета: Перевірити можливість завантаження виконаного завдання студентом.

Кроки:

1. Увійти в систему як студент.
2. Перейти до розділу "Завдання".
3. Вибрати завдання зі списку.
4. Завантажити файл із виконаним завданням.
5. Натиснути кнопку "Відправити".

Очікуваний результат: Виконане завдання успішно завантажено, викладач може переглядати його.

Результат виконання: Завдання успішно завантажено, помилок не виявлено.

Код:

```
describe('Assignment Submission', () => {
  it('should submit an assignment successfully', async () => {
    const response = await request(app)
      .post('/assignments/submit')
      .field('assignmentId', 'assignmentId123')
      .attach('file', 'path/to/completed_assignment.pdf')
      .set('Authorization', 'Bearer studentToken');
    expect(response.statusCode).toBe(200);
    expect(response.body.message).toBe('Assignment submitted successfully');  }); });
```

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи були досягнуті наступні результати:

1. Проведено аналіз предметної галузі, згідно теми дипломної роботи. Проаналізовано дві існуючі платформи для студентів: Mystat, Moodle LMS, виділено їхні основні переваги та недоліки.
2. Визначено функціональні та нефункціональні вимоги до платформи, за результатами аналізу предметної галузі та аналогів.
3. Проаналізовано існуючі засоби розробки web-застосунків та обрано наступні, як основні: бібліотека React та фреймворк Express, база даних MongoDB.
4. Спроектовано клієнт-серверну архітектуру платформи, що ділить застосунок на три частини: серверну, клієнтську та базу даних.
5. Розроблено навчальну платформу для студентів мовної школи із застосуванням компонентного підходу для розробки клієнтської частини та REST API для зв’язку між серверною та клієнтською частинами.
6. Проведено модульне, функціональне та системне тестування платформи із застосуванням фреймворку Jest. Результати тестування показали відсутність помилок у функціонуванні платформи.

Таким чином, виконання даної роботи дозволило створити ефективну систему управління навчальним процесом, яка відповідає сучасним вимогам та може бути успішно використана мовною школою для покращення організації та управління навчальними процесами. Розроблена система має потенціал для подальшого розширення.

Робота пройшла апробацію результатів дослідження на наступних конференціях: Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24.04.2024, Київ, ДУІКТ, Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.05.2024, Київ, ДУІКТ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Node.js документація [Електронний ресурс]. URL: https://nodejs.org/en
2. React документація [Електронний ресурс]. URL: https://react.dev/
3. Express.js документація [Електронний ресурс]. URL: https://expressjs.com/en/guide/routing.html
4. JWT сертифікація і документація [Електронний ресурс]. URL: https://jwt.io/
5. MongoDB документація [Електронний ресурс]. URL: https://www.mongodb.com/
6. Jest документація [Електронний ресурс]. URL: https://jestjs.io/uk/
7. Специфікації IANA [Електронний ресурс]. URL: https://www.iana.org
8. JSON Web Key Set (JWKS) [Електронний ресурс]. URL: https://auth0.com/docs/tokens/json-web-tokens/json-web-key-sets
9. Multer документація [Електронний ресурс]. URL: https://github.com/expressjs/multer
10. mongoose-paginate-v2 документація [Електронний ресурс]. URL https://www.npmjs.com/package/mongoose-paginate-v2
11. GridFS документація [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/core/gridfs/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка навчальної платформи для школи «Школа східних мов та культури" мовою JavaScript

Виконав студент 4 курсу

групи ПД-41
Могильник Максим Русланович
Керівник роботи

К.т.н., доцент кафедри ІПЗ Довженко Тимур Павлович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спрощення процесу вивчення іноземних мов східної групи за допомогою навчальної платформи мовою JavaScript.
- **Об'єкт дослідження** – процес вивчення іноземних мов східної групи.
- **Предмет дослідження** – програмне забезпечення для вивчення іноземних мов східної групи для школи «Школа східних мов та культури».

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих навчальних платформ, з метою визначення їх переваг і недоліків в умовах вивчення іноземних мов східної групи.
2. Визначити функціональні та нефункціональні вимоги до платформи.
3. Дослідити та вибрати відповідні засоби для розробки платформи. Це включає в себе вибір бібліотек та фреймворків для клієнтської та серверної частин, а також базу даних.
4. Спроектувати архітектуру платформи.
5. Розробити навчальну платформу на основі обраних інструментів і визначених вимог. Забезпечити реалізацію всіх функціональних можливостей.
6. Провести тестування розробленої платформи.

3

АНАЛІЗ АНАЛОГІВ

Характеристика застосунку	Mystat	Moodle LMS	Школа східних мов та культури
Новини, що стосуються обраного напрямку навчання та школи/ЗВО	-	-	+
Наявна бібліотека додаткових джерел інформації	+	-	+
Модуль завантаження завдань	+	+	+
Модуль перегляду успішності студента	+	+	+
Модуль відслідковування оплати за навчання	+	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Можливість взаємодії з завданнями (завантаження, коментування, оцінювання) в залежності від ролі користувача.
2. Можливість відстежувати успішність студента.
3. Перегляд списку додаткової літератури.
4. Можливість редагувати інформацію про користувача.
5. Перегляд дат оплати за навчання.
6. Адміністратори повинні мати можливість створювати та редагувати тарифні плани.

Нефункціональні вимоги

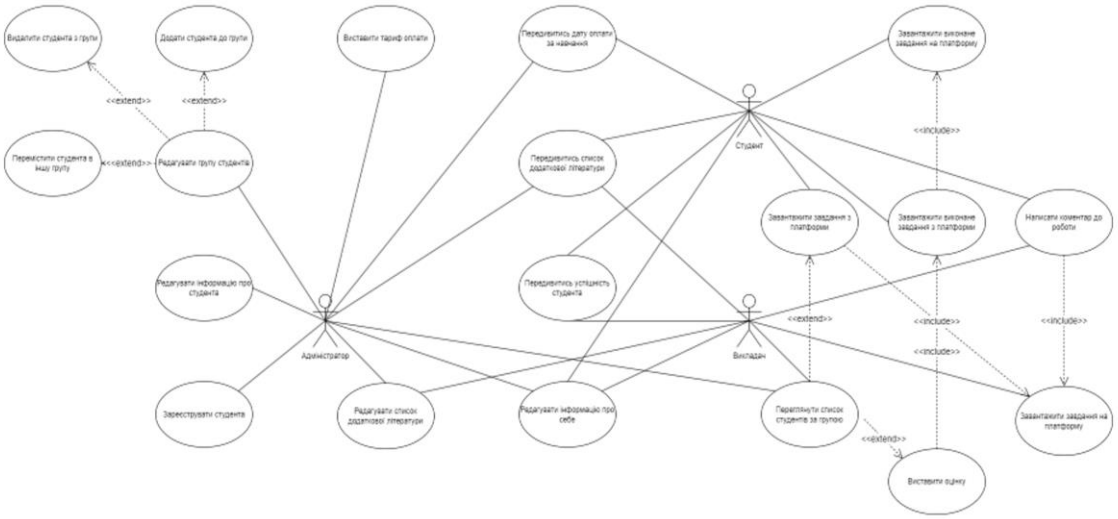
1. Сумісність з різними пристроями та браузерами (пристрої: ПК, ноутбуки, телефони; браузери: Google Chrome, Mozilla Firefox, Safari).
2. Доступ до будь-якої інформації максимум у три кліки.
3. Можливість завантажувати великі за розміром файли (до 100 Мб).
4. Формат використаних дат має бути: число.місяць.рік.
5. Можливість завантажувати і переглядати завдання у графічному форматі (фото: png, jpg, jpeg)

5

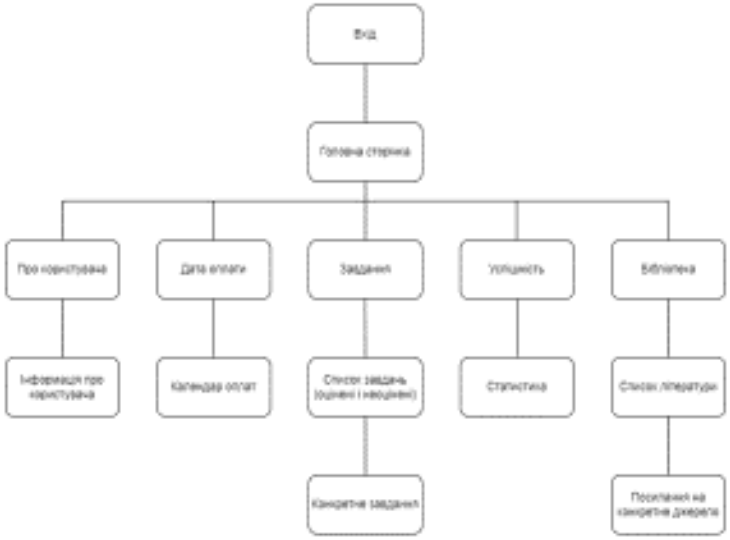
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

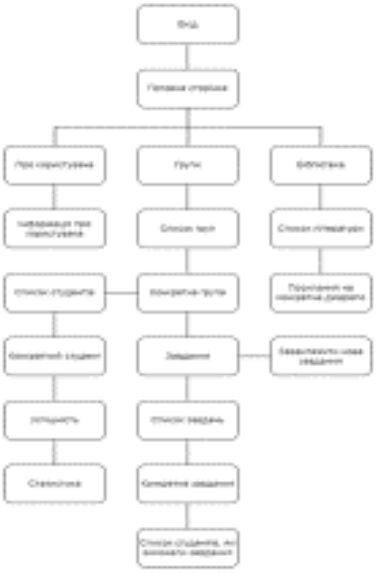
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



МАПА САЙТУ ДЛЯ СТУДЕНТА

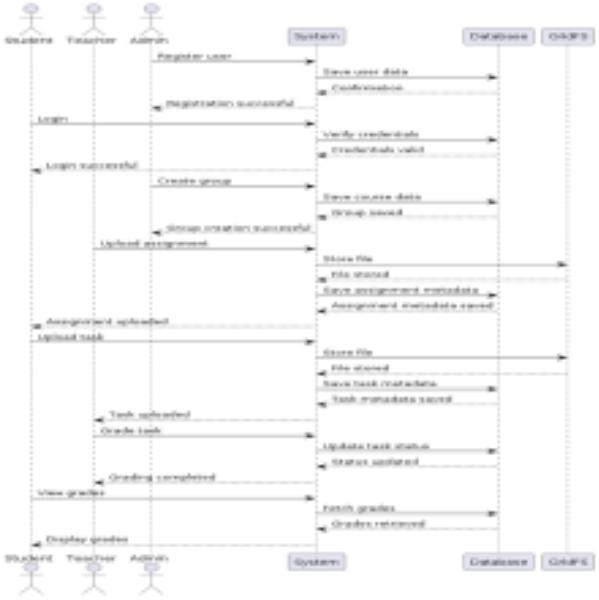


МАПА САЙТУ ДЛЯ ВИКЛАДАЧА



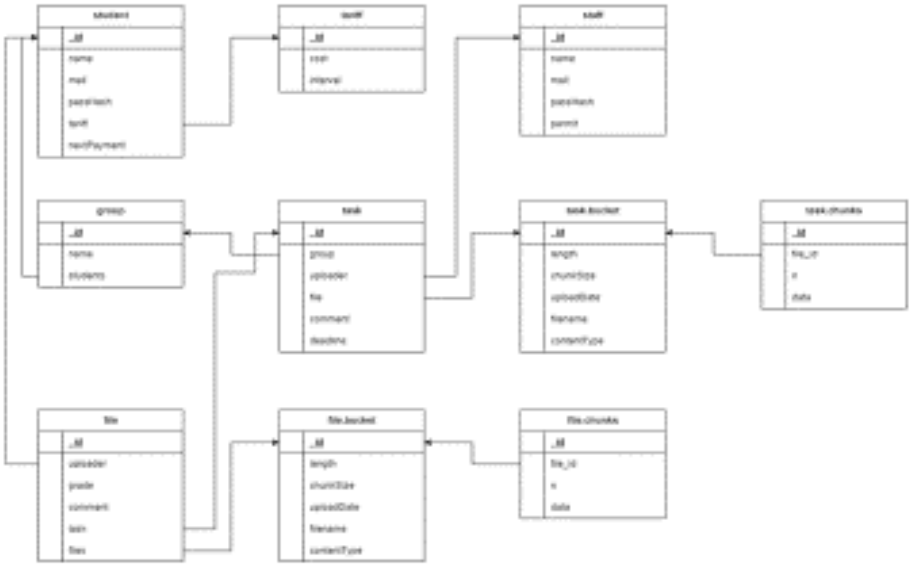
9

ДІАГРАМА ПОСЛІДОВНОСТІ ВИКОНАННЯ ПЕРШОГО ЗАВДАННЯ СТУДЕНТА



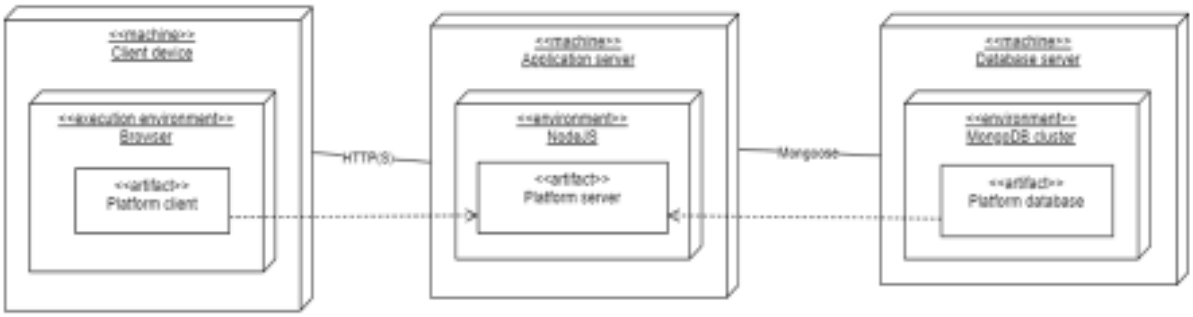
10

СХЕМА БАЗИ ДАНИХ



11

ДІАГРАМА РОЗГОРТАННЯ



12

ЕКРАННІ ФОРМИ

Вхід на платформу

13

ЕКРАННІ ФОРМИ

Головна сторінка на широкоформатному екрані

Головна сторінка на мобільному екрані

14

ЕКРАННІ ФОРМИ

Меню на широкоформатному екрані

Меню на мобільному екрані

15

ЕКРАННІ ФОРМИ

Сторінка з дз на широкоформатному екрані

Сторінка з дз на мобільному екрані

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Могильник М.Р., Довженко Т.П. Визначення вимог до навчальної платформи для школи "Школа східних мов та культури": Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, С. 49.
2. Могильник М.Р., Довженко Т.П. Визначення засобів реалізації навчальної платформи для школи "Школа східних мов та культури": Матеріали Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, С. 133.

17

ВИСНОВКИ

1. Проведено аналіз предметної галузі, згідно теми дипломної роботи. Проаналізовано дві існуючі платформи для студентів: Mystat, Moodle LMS, виділено їхні основні переваги та недоліки.
2. Визначено функціональні та нефункціональні вимоги до платформи, за результатами аналізу предметної галузі та аналогів.
3. Проаналізовано існуючі засоби розробки web-застосунків та обрано наступні, як основні: бібліотека React та фреймворк Express, база даних MongoDB.
4. Спроектовано клієнт-серверну архітектуру платформи, що ділить застосунок на три частини: серверну, клієнтську та базу даних.
5. Розроблено навчальну платформу для студентів мовної школи із застосуванням компонентного підходу для розробки клієнтської частини та REST API для зв'язку між серверною та клієнтською частинами.
6. Проведено модульне, функціональне та системне тестування платформи із застосуванням фреймворку Jest. Результати тестування показали відсутність помилок у функціонуванні платформи.

18

ДОДАТОК Б

ЛІСТИНГ КОДУ ПРОГРАМИ

App.js:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import {
  createBrowserRouter,
  RouterProvider,
} from "react-router-dom";
import App from './App';
import reportWebVitals from './reportWebVitals';
import 'bootstrap/dist/css/bootstrap.min.css';
import Main from './Main';
import Homework from './studentComponents/Homework.js';
import Stat from './studentComponents/Stat.js';
import Literature from './studentComponents/Literature.js';
import Payment from './studentComponents/Payment.js';
import Profile from './Profile.js';
import TeacherHomework from './teacherComponents/TeacherHomework.js';

const router = createBrowserRouter([
  {
    path: "/",
    element: <App></App>,
  },
  {
    path: "/main",
    element: <Main></Main>,
    children: [
      {
        path: "homework",
        element: <Homework />,
```

```
      },
      {
        path: "stat",
        element: <Stat />,
      },
      {
        path: "literature",
        element: <Literature />,
      },
      {
        path: "payment",
        element: <Payment />,
      },
      {
        path: "profile",
        element: <Profile />,
      },
      {
        path: "/main/group1/tasks1",
        element: <TeacherHomework />
      }
    ]
  }
]);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
reportWebVitals();
```

Main.js:

```
import React from "react";
import "./Main.css";
import { Navbar, Container, Button} from "react-bootstrap";
import { FaSignOutAlt } from "react-icons/fa";
import SideNavBar from "./Sidebar";
import { useNavigate } from "react-router-dom";
import News from "./News.js";
import { Outlet, useLocation } from 'react-router-dom';

const Main = (props) => {
  const navigate = useNavigate();
  const location = useLocation();

  const handleExit = (e) => {
    navigate("/");
  };

  return (
    <div>
      <Navbar bg="dark" data-bs-theme="dark" className="navbar-custom">
        <Container className="d-flex justify-content-between">
          <div className="d-flex flex-grow-1 justify-content-center">
            <Navbar.Text className="navbar-brand-custom">
              Школа східних мов та культури
            </Navbar.Text>
          </div>
          <Button variant="outline-warning" onClick={handleExit}>
            <FaSignOutAlt /> Вийти
          </Button>
        </Container>
      </Navbar>
      <div className="all-content">
```

```
      <SideNavBar className="side-nav" />
      <div className="content-window">
        {location.pathname === '/main' ? <News /> : <Outlet/>}
      </div>
    </div>
  </div>
 );
};

export default Main;
```

Sidebar.js:

```
import React, { useState } from 'react';
import { Navbar, Nav, Image, Accordion, useAccordionButton, Card } from 'react-bootstrap';
import { Link } from 'react-router-dom';
import { FaHome, FaChartLine, FaTasks, FaChevronDown, FaChevronRight } from "react-icons/fa";
import { MdLocalLibrary } from "react-icons/md";
import { MdOutlinePayments } from "react-icons/md";
import { FaUserGroup } from "react-icons/fa6";
import './Sidebar.css';

const Sidebar = () => {
  const [expanded, setExpanded] = useState(false);
  const [activeEventKey, setActiveEventKey] = useState(null);
  const userName = localStorage.getItem("name"); // Replace with dynamic user data if available
  const userImage = "https://cdn-icons-png.flaticon.com/512/149/149071.png"; // Replace with the path
to the user's image

  const handleMouseEnter = () => setExpanded(true);
  const handleMouseLeave = () => {
    setExpanded(false);
    setActiveEventKey(null); // Collapse all accordion items
  };
```

```
 function CustomToggle({ children, eventKey }) {
  const decoratedOnClick = useAccordionButton(eventKey, () =>
   setActiveEventKey(activeEventKey === eventKey ? null : eventKey)
  );
  return (
   <div onClick={decoratedOnClick} style={{ cursor: 'pointer' }}>
    {children}
   </div>
  );
 }


 if(auth.header.permit === "Teacher"){
  return (
   <div className="sidebar-container" onMouseEnter={handleMouseEnter}
onMouseLeave={handleMouseLeave}>
    <div className={`sidebar ${expanded ? 'expanded' : ''}`}>
     <Navbar bg="dark" variant="dark" expand="lg" className="mr-auto flex-column">
      <div className="user-info text-center mt-3">
       <Image src={userImage} roundedCircle className="user-image" />
       <Navbar.Brand as={Link} to="/main/profile" className={`mt-2 user-name ${expanded ? 'd-
block' : 'd-none'}`}>
        {userName}
       </Navbar.Brand>
      </div>
      <Navbar.Text id="basic-navbar-nav">
       <Nav className="mr-auto flex-column mt-5">
        <Nav.Link as={Link} to="/main" className="nav-link-custom">
         <span className="icon"><FaHome /></span>
         <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Головна</span>
        </Nav.Link>
        <Nav.Link>
        <Accordion activeKey={activeEventKey}>
         <div>
```

```
            <CustomToggle eventKey="0">
              <span className="icon"><FaUserGroup /></span>
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Яп. вт, чт</span>
              <span className={`arrow-icon ${expanded ? " : 'hidden'}`}>
                {activeEventKey === "0" ? <FaChevronDown /> : <FaChevronRight />}
              </span>
            </CustomToggle>
            <Accordion.Collapse eventKey="0">
              <div>
                <Nav.Link as={Link} to="/main/group1/list1">Список студентів</Nav.Link>
                <Nav.Link as={Link} to="/main/group1/tasks1">Завдання</Nav.Link>
              </div>
            </Accordion.Collapse>
          </div>

          <div>
            <CustomToggle eventKey="1">
              <span className="icon"><FaUserGroup /></span>
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Кит. пн, ср,
пт</span>
              <span className={`arrow-icon ${expanded ? " : 'hidden'}`}>
                {activeEventKey === "1" ? <FaChevronDown /> : <FaChevronRight />}
              </span>
            </CustomToggle>
            <Accordion.Collapse eventKey="1">
              <div>
                <Nav.Link as={Link} to="/main/group2/list2">Список студентів</Nav.Link>
                <Nav.Link as={Link} to="/main/group2/tasks2">Завдання</Nav.Link>
              </div>
            </Accordion.Collapse>
          </div>
        </Accordion>
        </Nav.Link>
```

```
            <Nav.Link as={Link} to="/main/literature" className="nav-link-custom">
              <span className="icon"><MdLocalLibrary /></span>
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Література</span>
            </Nav.Link>
          </Nav>
        </Navbar.Text>
      </Navbar>
    </div>
  </div>
 );
}
 else{
  return (
   <div className="sidebar-container" onMouseEnter={handleMouseEnter} onMouseLeave={handleMouseLeave}>
    <div className={`sidebar ${expanded ? 'expanded' : ''}`}>
     <Navbar bg="dark" variant="dark" expand="lg" className="mr-auto flex-column">
      <div className="user-info text-center mt-3">
       <Image src={userImage} roundedCircle className="user-image" />
       <Navbar.Brand as={Link} to="/main/profile" className={`mt-2 user-name ${expanded ? 'd-block' : 'd-none'}`}>{userName}</Navbar.Brand>
      </div>
      <Navbar.Text id="basic-navbar-nav">
       <Nav className="mr-auto flex-column mt-5">
        <Nav.Link as={Link} to="/main" className="nav-link-custom">
         <span className="icon"><FaHome/></span>
         <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Головна</span>
        </Nav.Link>
        <Nav.Link as={Link} to="/main/stat" className="nav-link-custom">
         <span className="icon"><FaChartLine/></span>
         <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Успішність</span>
        </Nav.Link>
        <Nav.Link as={Link} to="/main/homework" className="nav-link-custom">
         <span className="icon"><FaTasks/></span>
```

```
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Домашні
завдання</span>
            </Nav.Link>
            <Nav.Link as={Link} to="/main/literature" className="nav-link-custom">
              <span className="icon"><MdLocalLibrary/></span>
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Література</span>
            </Nav.Link>
            <Nav.Link as={Link} to="/main/payment" className="nav-link-custom">
              <span className="icon"><MdOutlinePayments /></span>
              <span className={`nav-text ${expanded ? 'd-inline' : 'd-none'}`}>Оплата</span>
            </Nav.Link>
          </Nav>
        </Navbar.Text>
      </Navbar>
    </div>
  </div>
  );
 }

};

export default Sidebar;
```

Payment.js:

```
import React, { useState } from 'react';
import { Container, Row, Col } from 'react-bootstrap';
import { format, startOfMonth, endOfMonth, startOfWeek, endOfWeek, addMonths, isSameMonth,
isSameDay, addDays } from 'date-fns';
import './Payment.css';
import { MdKeyboardArrowLeft, MdKeyboardArrowRight } from "react-icons/md";

const paymentDates = [
 new Date('2024-06-02'),
 new Date('2024-07-07'),
```

```
  new Date('2024-08-04'),
  new Date('2024-09-08'),
  new Date('2024-10-06'),
];

const monthsInNominative = [
  'січень', 'лютий', 'березень', 'квітень', 'травень', 'червень',
  'липень', 'серпень', 'вересень', 'жовтень', 'листопад', 'грудень'
];

const daysInNominative = [
  'понеділок', 'вівторок', 'середа', 'четвер', 'п’ятниця', 'субота', 'неділя'
];

const CustomCalendar = () => {
  const [currentMonth, setCurrentMonth] = useState(new Date());
  const [selectedDate, setSelectedDate] = useState(null);

  const renderHeader = () => {
    const dateFormat = 'MMMM yyyy';
    const month = monthsInNominative[currentMonth.getMonth()];
    const year = currentMonth.getFullYear();

    return (
      <div className="header row flex-middle">
        <div className="col col-start">
          <div className="icon" onClick={prevMonth}>
            <MdKeyboardArrowLeft/>
          </div>
        </div>
        <div className="col col-center">
          <span>{`${month} ${year}`}</span>
        </div>
```

```
      <div className="col col-end">
        <div className="icon" onClick={nextMonth}>
          <MdKeyboardArrowRight/>
        </div>
      </div>
    </div>
  );
};

const renderDays = () => {
  const days = [];

  for (let i = 0; i < 7; i++) {
    days.push(
      <div className="col col-center" key={i}>
        {daysInNominative[i]}
      </div>
    );
  }

  return <div className="days row">{days}</div>;
};

const renderCells = () => {
  const monthStart = startOfMonth(currentMonth);
  const monthEnd = endOfMonth(monthStart);
  const startDate = startOfWeek(monthStart);
  const endDate = endOfWeek(monthEnd);

  const rows = [];
  let days = [];
  let day = startDate;
  let formattedDate = '';
```

```
  while (day <= endDate) {
    for (let i = 0; i < 7; i++) {
      formattedDate = format(day, 'd');
      const cloneDay = day;
      days.push(
        <div
          className={`col cell ${
            !isSameMonth(day, monthStart)
              ? 'disabled'
              : isSameDay(day, selectedDate)
              ? 'selected'
              : paymentDates.some(d => isSameDay(d, day))
              ? 'highlight'
              : ''
          }`}
          key={day}
          onClick={() => onDateClick(cloneDay)}
        >
          <span className="number">{formattedDate}</span>
        </div>
      );
      day = addDays(day, 1);
    }
    rows.push(
      <div className="row" key={day}>
        {days}
      </div>
    );
    days = [];
  }
  return <div className="body">{rows}</div>;
};
```

```
  const onDateClick = day => {
    setSelectedDate(day);
  };

  const nextMonth = () => {
    setCurrentMonth(addMonths(currentMonth, 1));
  };

  const prevMonth = () => {
    setCurrentMonth(addMonths(currentMonth, -1));
  };

  return (
    <Container fluid className="calendar-container d-flex justify-content-center align-items-center vh-100">
      <Row className="w-100">
        <Col className="d-flex justify-content-center">
          <div className="calendar">
            {renderHeader()}
            {renderDays()}
            {renderCells()}
          </div>
        </Col>
      </Row>
    </Container>
  );
};

export default CustomCalendar;
```