ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

РОЗРОБКА СИСТЕМИ РЕЙТИНГУВАННЯ СТУДЕНТІВ СПЕЦІАЛЬНОСТІ «КОМП'ЮТЕРНІ НАУКИ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Микитенко Віталій Іванович
_____«____» _____202_ р.
(підпис)

Науковий керівник доц., к.ф.-м.н. Черненко Оксана Олексіївна
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2024

РЕФЕРАТ

Записка: 79 с., 7 рис., 1 таблиця, 1 додаток, 20 джерел.

СИСТЕМА РЕЙТИНГУВАННЯ, СТУДЕНТИ, PYTHON, FLASK, REACT

Об’єкт розробки – програмна реалізація системи рейтингування студентів спеціальності «Комп’ютерні науки».

Мета роботи – розробка системи рейтингування студентів, яка дозволяє користувачам авторизуватися за допомогою PIN-коду, додавати інформацію про студентів, їхні оцінки, створювати рейтинги студентів та переглядати дані у зручному інтерфейсі, використовуючи мову програмування Python та фреймворки Flask і React.

Методи дослідження – методи програмування на мові Python з використанням фреймворку Flask для бекенду, бібліотеки SQLAlchemy для роботи з базою даних, фреймворку React для фронтенду, а також методи тестування та налагодження веб-додатків.

Розроблено веб-додаток, який дозволяє користувачам авторизуватися за допомогою PIN-коду, додавати та редагувати інформацію про студентів, додавати оцінки для студентів, створювати рейтинги на основі середнього бала та переглядати інформацію у зручному форматі. Система також надає можливість сортування та фільтрації даних для покращення зручності користувачів.

Було проведено огляд існуючих систем рейтингування студентів, аналіз їх функціоналу та інтерфейсів. На основі аналізу визначено ключові вимоги до функціональності та дизайну власного додатку. Відбулося проектування архітектури додатку, його реалізація, тестування та оптимізація.

Результати роботи включають розроблену програму з документацією, що містить інструкції для користувачів та рекомендації щодо подальшого розвитку додатку. Висвітлено позитивні та негативні аспекти використання розробленої системи, а також запропоновано можливі шляхи її вдосконалення.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Flask	Фреймворк для розробки веб-додатків на Python
SQLite	Легка вбудована база даних, яка використовується для збереження даних додатку
HTML	Мова розмітки гіпертексту, використовується для створення веб-сторінок
CSS	Мова стилізації веб-документів
JavaScript	Мова програмування, яка використовується для додавання інтерактивності на веб-сторінках
SQLAlchemy	ORM-бібліотека для Python, яка полегшує роботу з базами даних
Jinja2	Шаблонізатор для Python, використовується для динамічного створення HTML
Bootstrap	Фреймворк для швидкої розробки адаптивних веб-дизайнів
API	Інтерфейс програмування додатків, дозволяє взаємодію між різними програмами
CRUD	Аббревіатура, що означає створення (Create), читання (Read), оновлення (Update), видалення (Delete) даних
CSRF	Захист від підробки міжсайтових запитів, метод безпеки для захисту від атак на веб-додатки
JWT	JSON Web Token, стандарт для створення токенів доступу, що дозволяють безпечний обмін інформацією
CLI	Командний інтерфейс користувача, інтерфейс для взаємодії з додатком через командний рядок
ORM	Об'єктно-реляційне відображення, технологія, яка дозволяє взаємодіяти з базою даних за допомогою об'єктів
MVC	Архітектурний шаблон Model-View-Controller, який розділяє додаток на три взаємодіючі

	компоненти: модель, уявлення та контролер
HTTP	Протокол передачі гіпертексту, основний протокол для передачі даних у веб
REST	Стиль архітектури для створення веб-сервісів, що використовує HTTP
SSL/TLS	Протоколи для забезпечення захищеної передачі даних через Інтернет
JSON	Формат обміну даними, який є зручним для читання людиною та машинної обробки
Blueprint	Функція Flask для організації додатку в модулі та створення масштабованої структури проекту
Alembic	Інструмент для міграцій баз даних у проєктах, що використовують SQLAlchemy
WSGI	Веб-серверний шлюзовий інтерфейс, стандарт для взаємодії веб-серверів з веб-додатками у Python
React	Бібліотека JavaScript для побудови користувацьких інтерфейсів
JSX	Розширення синтаксису JavaScript, що використовується з React для опису вигляду UI-компонентів
Node.js	Середовище виконання JavaScript, що дозволяє виконувати код JavaScript на сервері
NPM	Менеджер пакетів для JavaScript, який зазвичай використовується з Node.js
Axios	Бібліотека для виконання HTTP-запитів з браузера та Node.js
Redux	Бібліотека JavaScript для управління станом додатків

ВСТУП

У сучасному світі цифрових технологій та Інтернету існує велика потреба у зручних та ефективних системах для управління освітнім процесом та оцінюванням студентів. Системи рейтингування студентів відіграють ключову роль у забезпеченні прозорості та об'єктивності оцінювання, що є важливим елементом сучасної освітньої системи.

Актуальним є розробка систем рейтингування, орієнтованих на специфічні потреби навчальних закладів та студентів, які дозволяють автоматизувати процеси збору, обробки та аналізу даних про успішність студентів. Використання сучасних технологій, таких як Python, Flask та React, дозволяє створювати потужні та надійні рішення, що відповідають вимогам часу та забезпечують високу продуктивність та зручність у використанні.

Об’єктом розробки є система рейтингування студентів спеціальності «Комп’ютерні науки», яка включає функціонал авторизації користувачів за допомогою PIN-коду, додавання та редагування інформації про студентів та їхні оцінки, створення рейтингів на основі середнього бала, а також перегляд та аналіз даних у зручному інтерфейсі.

Предметом розробки є система рейтингування студентів, реалізована на базі фреймворку Flask з використанням мови програмування Python для бекенду та фреймворку React для фронтенду.

Метою роботи є розробка системи рейтингування студентів, яка забезпечує зручне та ефективне управління даними про успішність студентів, дозволяє автоматизувати процеси збору та аналізу оцінок, створювати рейтинги на основі різних критеріїв та забезпечує прозорість та об'єктивність оцінювання.

Методи дослідження включають аналіз існуючих систем рейтингування студентів, їх методів та інструментів, розробку структури проекту, використання бібліотек та фреймворків, проектування та реалізацію основного функціоналу, тестування додатку на різних браузерах, а також аналіз його впливу на зручність користувачів.

Робота складається з вступу, де визначено актуальність, мету, об'єкт і предмет дослідження; інформаційного огляду, який присвячений теоретичним основам розробки веб-додатків, інструментам та технологіям для розробки на Python, Flask та React, а також прикладам аналогічних систем рейтингування; теоретичної частини, що описує принципи розробки веб-додатків, особливості роботи з Flask та React, питання забезпечення безпеки даних; практичної частини, яка включає опис процесу розробки програмного забезпечення, структури проекту, використаних бібліотек та фреймворків, а також функціоналу програми; висновків та додатків.

Розробка системи рейтингування студентів на базі Flask та React дозволяє детально вивчити процеси організації та управління користувацькими даними, забезпечення безпеки даних, а також інтеграції різних інструментів для покращення взаємодії студентів та викладачів. Це сприяє створенню ефективних інструментів для оцінювання та аналізу успішності студентів, що підвищує загальну продуктивність та прозорість освітнього процесу.

1. ПОСТАНОВКА ЗАДАЧІ

Сучасні технології та Інтернет суттєво впливають на різні аспекти життя, включаючи освіту та управління освітніми процесами. В умовах швидкого розвитку цифрових технологій та зростаючих вимог до якості освіти, виникає необхідність у створенні ефективних систем для оцінювання та рейтингування студентів. Такі системи мають забезпечувати зручний доступ до даних, прозорість та об'єктивність оцінювання, а також можливість аналізу успішності студентів.

Системи рейтингування студентів дозволяють не лише забезпечити об'єктивне оцінювання, а й надавати зворотний зв'язок студентам, що сприяє покращенню їхньої успішності. Використання сучасних технологій, таких як Python, Flask та React, дозволяє створювати потужні, надійні та зручні у використанні системи, що відповідають сучасним вимогам.

Метою даної роботи є розробка системи рейтингування студентів спеціальності «Комп’ютерні науки», яка забезпечує зручне та ефективне управління даними про успішність студентів, дозволяє автоматизувати процеси збору та аналізу оцінок, створювати рейтинги на основі різних критеріїв, а також забезпечує прозорість та об'єктивність оцінювання.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Провести аналіз існуючих систем рейтингування студентів, визначити їхні переваги та недоліки.
2. Визначити вимоги до функціональності та дизайну власної системи рейтингування студентів.
3. Розробити архітектуру додатку, вибрати відповідні інструменти та технології для його реалізації.
4. Реалізувати функціональність додатку, яка включає:
    - Авторизацію користувачів за допомогою PIN-коду.
    - Додавання та редагування інформації про студентів.
    - Додавання та перегляд оцінок для студентів.
    - Створення рейтингів на основі середнього бала студентів.
    - Перегляд рейтингів та детальної інформації про студентів.

5. Забезпечити безпеку даних користувачів та захист від несанкціонованого доступу.
6. Провести тестування та налагодження додатку, перевірити його роботу на різних браузерах.
7. Розробити документацію, яка містить інструкції для користувачів та рекомендації щодо подальшого розвитку додатку.
8. Оцінити ефективність розробленого додатку, визначити його позитивні та негативні сторони, а також можливості для подальшого вдосконалення.

Таким чином, виконання цих задач сприятиме створенню ефективного інструменту для оцінювання та аналізу успішності студентів, що підвищить їх продуктивність та забезпечить прозорість освітнього процесу.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд існуючих систем рейтингування студентів

Існуючі системи рейтингування студентів відіграють важливу роль у сучасних освітніх закладах, оскільки вони забезпечують прозорість та об'єктивність оцінювання, а також сприяють підвищенню успішності студентів. У цьому розділі розглянемо деякі з найбільш популярних систем рейтингування, їхні особливості, переваги та недоліки.

Blackboard Learn

Blackboard Learn є однією з найбільш відомих систем управління навчанням, яка використовується у багатьох університетах та коледжах по всьому світу. Основні функції Blackboard Learn включають управління курсами, розміщення навчальних матеріалів, проведення онлайн-тестувань та оцінювання студентів.

Переваги:

- Інтеграція з різними освітніми інструментами.
- Підтримка різних методів оцінювання, включаючи тести, завдання та проекти.
- Широкий набір інструментів для аналізу успішності студентів.

Недоліки:

- Висока вартість ліцензії.
- Складний інтерфейс, який може вимагати значного часу для навчання.

Moodle

Moodle є відкритою системою управління навчанням, яка широко використовується у різних освітніх установах. Вона дозволяє створювати онлайн-курси, управляти навчальними матеріалами, проводити оцінювання та генерувати звіти про успішність студентів.

Переваги:

- Безкоштовна та відкрита платформа.
- Гнучкість у налаштуванні та адаптації до потреб конкретного навчального

закладу.

- Підтримка різних плагінів та модулів, які розширюють функціональність системи.

Недоліки:

- Потребує технічних знань для налаштування та підтримки.
- Може бути складною у використанні для новачків.

Canvas

Canvas є сучасною системою управління навчанням, яка набирає популярності завдяки своїй зручності та функціональності. Вона забезпечує управління курсами, проведення оцінювань, а також надає інструменти для комунікації між студентами та викладачами.

Переваги:

- Інтуїтивно зрозумілий інтерфейс.
- Підтримка мобільних додатків для доступу до системи з будь-якого пристрою.
- Можливість інтеграції з іншими освітніми інструментами та сервісами.

Недоліки:

- Обмежена безкоштовна версія.
- Деякі функції можуть вимагати додаткових налаштувань та адаптацій.

Google Classroom

Google Classroom є безкоштовною системою управління навчанням, яка інтегрується з іншими сервісами Google, такими як Google Docs, Google Drive та Google Calendar. Вона дозволяє створювати курси, розміщувати завдання, оцінювати студентів та надавати зворотний зв'язок.

Переваги:

- Безкоштовне використання для освітніх установ.
- Інтеграція з екосистемою Google.
- Простота у використанні та налаштуванні.

Недоліки:

- Обмежена функціональність порівняно з іншими системами управління

навчанням.

- Залежність від інтернет-з'єднання для доступу до більшості функцій.

Coursera та інші онлайн-платформи

Coursera та інші онлайн-платформи для навчання, такі як Udacity та edX, також включають системи рейтингування та оцінювання студентів. Ці платформи пропонують курси від провідних університетів та компаній, що дозволяє студентам отримувати знання та сертифікації у різних галузях.

Переваги:

- Доступ до курсів від провідних університетів та компаній.
- Можливість отримання сертифікатів та дипломів.
- Гнучкість у виборі часу та темпу навчання.

Недоліки:

- Вартість деяких курсів може бути високою.
- Обмежені можливості для персоналізації навчального процесу.

На основі аналізу існуючих систем рейтингування студентів можна зробити висновок, що кожна з них має свої переваги та недоліки. Вибір конкретної системи залежить від потреб та можливостей навчального закладу. Розробка власної системи рейтингування студентів на основі сучасних технологій, таких як Python, Flask та React, дозволяє врахувати всі переваги та уникнути недоліків існуючих рішень, створивши ефективний інструмент для управління успішністю студентів.

2.2. Порівняльний аналіз технологій для розробки веб-додатків

У сучасному світі існує безліч технологій для розробки веб-додатків. Кожна з них має свої переваги та недоліки, а вибір конкретної технології залежить від вимог проекту, кваліфікації розробників та інших факторів. У цьому розділі розглянемо деякі з найпопулярніших технологій для розробки веб-додатків та проведемо їх порівняльний аналіз.

Flask

Flask – це мікрофреймворк для розробки веб-додатків на мові програмування Python. Він є легким, гнучким та дозволяє швидко створювати прості веб-додатки. [1]

Переваги:

- Простота використання та налаштування.
- Легка вага та швидкість роботи.
- Велика кількість розширень для додавання функціональності.
- Гнучкість та можливість налаштування під конкретні потреби.

Недоліки:

- Не включає багато вбудованих функцій, які є у більш важких фреймворках.
- Може бути менш підходящим для великих проектів з великою кількістю користувачів.

Django

Django – це потужний веб-фреймворк для Python, який включає багато вбудованих функцій, що дозволяють швидко розробляти складні веб-додатки.

Переваги:

- Велика кількість вбудованих функцій та інструментів.
- Висока продуктивність та безпека.
- Велика та активна спільнота розробників.
- Структурований підхід до розробки.

Недоліки:

- Більша складність у порівнянні з Flask.
- Менша гнучкість через велику кількість вбудованих функцій.

Express.js

Express.js – це мініфреймворк для Node.js, який дозволяє створювати веб-додатки та API. Він є одним з найпопулярніших фреймворків для JavaScript.

Переваги:

- Легка вага та висока продуктивність.
- Велика кількість доступних модулів та розширень.
- Простота у використанні та налаштуванні.

- Відмінна інтеграція з іншими інструментами JavaScript.

Недоліки:

- Менша структура у порівнянні з більшими фреймворками.
- Можуть виникати проблеми з продуктивністю у великих проектах.

Ruby on Rails

Ruby on Rails – це веб-фреймворк для мови програмування Ruby, який орієнтований на простоту та швидкість розробки.

Переваги:

- Висока продуктивність та швидкість розробки.
- Велика кількість вбудованих функцій та інструментів.
- Активна спільнота розробників.
- Гарна підтримка для тестування додатків.

Недоліки:

- Менш популярний у порівнянні з іншими фреймворками.
- Може бути менш продуктивним у великих проектах з великою кількістю користувачів.

Spring

Spring – це потужний фреймворк для розробки веб-додатків на мові програмування Java. Він включає багато вбудованих функцій для створення складних та масштабованих додатків.

Переваги:

- Висока продуктивність та масштабованість.
- Велика кількість вбудованих функцій та інструментів.
- Відмінна підтримка для корпоративних додатків.
- Велика спільнота розробників.

Недоліки:

- Велика складність та крива навчання.
- Важка вага у порівнянні з іншими фреймворками.

React

React – це бібліотека для розробки користувацьких інтерфейсів, створена

компанією Facebook. Вона дозволяє створювати динамічні та інтерактивні веб-додатки. [2]

Переваги:

- Висока продуктивність та швидкість роботи.
- Компонентний підхід до розробки.
- Велика та активна спільнота розробників.
- Підтримка для створення мобільних додатків за допомогою React Native.

Недоліки:

- Потребує додаткових бібліотек для повноцінної роботи.
- Може бути складним для новачків.

Angular

Angular – це фреймворк для розробки веб-додатків, створений компанією Google. Він орієнтований на створення масштабованих та продуктивних додатків.

Переваги:

- Висока продуктивність та масштабованість.
- Велика кількість вбудованих функцій та інструментів.
- Активна спільнота розробників.
- Підтримка для створення мобільних додатків за допомогою Angular.

Недоліки:

- Висока складність та крива навчання.
- Більша вага у порівнянні з іншими фреймворками.

На основі порівняльного аналізу технологій для розробки веб-додатків можна зробити висновок, що кожна з них має свої переваги та недоліки. Вибір конкретної технології залежить від вимог проекту, кваліфікації розробників та інших факторів. Для розробки системи рейтингування студентів спеціальності «Комп’ютерні науки» були обрані Flask для бекенду та React для фронтенду, оскільки вони забезпечують високу продуктивність, гнучкість та зручність у використанні, що дозволяє створити ефективний інструмент для управління даними про успішність студентів.

2.3 Огляд інструментів для розробки на Python та Flask

Flask є одним з найпопулярніших фреймворків для розробки веб-додатків на мові програмування Python. Його популярність зумовлена простотою використання, гнучкістю та можливістю швидкого створення прототипів. У цьому розділі ми розглянемо основні інструменти та бібліотеки, які використовуються разом з Flask для створення потужних і ефективних веб-додатків. [1]

Flask

Flask – це мікрофреймворк для розробки веб-додатків, який надає основні функції для створення серверних додатків без додаткового навантаження. Він легкий та розширюваний, що дозволяє розробникам додавати лише необхідні компоненти.

Основні переваги Flask:

- Простота у використанні.
- Гнучкість та можливість розширення за допомогою додаткових бібліотек.
- Відмінна документація та велика спільнота розробників.

Jinja2

Jinja2 – це потужний шаблонізатор для Python, який використовується у Flask для створення HTML-сторінок. Він дозволяє легко генерувати динамічний HTML-код, використовуючи шаблони з вбудованими змінними та логічними конструкціями.

Основні можливості Jinja2:

- Зручний синтаксис, схожий на Python.
- Підтримка фільтрів та макросів для спрощення коду.
- Можливість створення власних розширень.

SQLAlchemy

SQLAlchemy – це ORM (об'єктно-реляційне відображення) бібліотека для Python, яка спрощує роботу з базами даних. Вона дозволяє розробникам взаємодіяти з базою даних, використовуючи об'єктно-орієнтований підхід замість написання SQL-запитів.

Основні переваги SQLAlchemy:

- Абстрагування роботи з базами даних.
- Підтримка різних баз даних (SQLite, MySQL, PostgreSQL та інші).
- Потужні можливості для роботи з транзакціями та відношеннями між таблицями.

Alembic

Alembic – це інструмент для управління міграціями баз даних у проектах, які використовують SQLAlchemy. Він дозволяє легко відслідковувати зміни у схемі бази даних та застосовувати їх у різних середовищах.

Основні можливості Alembic:

- Створення та застосування міграцій баз даних.
- Автоматичне генерування міграцій на основі змін у моделях SQLAlchemy.
- Можливість керування версіями схеми бази даних.

Flask-Login

Flask-Login – це розширення для Flask, яке спрощує управління аутентифікацією та авторизацією користувачів. Воно забезпечує прості засоби для обробки сесій користувачів та захисту доступу до ресурсів додатку. [1]

Основні можливості Flask-Login:

- Просте управління сесіями користувачів.
- Підтримка зберігання інформації про входи користувачів.
- Можливість легкої інтеграції з іншими розширеннями для управління користувачами.

Flask-WTF

Flask-WTF – це розширення для інтеграції Flask з бібліотекою WTForms, яка використовується для створення та обробки веб-форм. Вона забезпечує зручні засоби для валідації даних форм та їхньої обробки.

Основні можливості Flask-WTF:

- Інтеграція з WTForms для створення та валідації форм.
- Підтримка CSRF-захисту для форм.
- Зручні засоби для роботи з файлами та завантаженням даних.

Flask-Migrate

Flask-Migrate – це розширення для Flask, яке забезпечує інтеграцію з Alembic для управління міграціями баз даних у проектах, що використовують Flask та SQLAlchemy. Воно дозволяє легко керувати змінами у схемі бази даних та застосовувати їх у різних середовищах.

Основні можливості Flask-Migrate:

- Інтеграція з Alembic для управління міграціями.
- Простий інтерфейс для створення та застосування міграцій.
- Підтримка автоматичного генерування міграцій.

Flask-CORS

Flask-CORS – це розширення для Flask, яке забезпечує підтримку Cross-Origin Resource Sharing (CORS) у веб-додатках. Воно дозволяє налаштовувати доступ до ресурсів додатку з інших доменів.

Основні можливості Flask-CORS:

- Просте налаштування CORS для Flask-додатків.
- Підтримка різних методів та заголовків для запитів.
- Гнучкість у налаштуванні політик доступу.

Висновок

Розробка веб-додатків на Python та Flask забезпечує високу продуктивність, гнучкість та зручність у використанні. Використання додаткових інструментів та розширень, таких як Jinja2, SQLAlchemy, Alembic, Flask-Login, Flask-WTF, Flask-Migrate та Flask-CORS, дозволяє створювати потужні та надійні веб-додатки, які відповідають сучасним вимогам та забезпечують високу якість користувацького досвіду.

2.4 Огляд інструментів для розробки на React

React є однією з найпопулярніших бібліотек для розробки користувацьких інтерфейсів. Його популярність обумовлена високою продуктивністю, гнучкістю та широкими можливостями для створення динамічних веб-додатків. У цьому розділі ми розглянемо основні інструменти та бібліотеки, які використовуються разом з

React для створення сучасних веб-додатків.

React

React – це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона дозволяє створювати багаторазові компоненти, що спрощує розробку та підтримку додатків. [3]

Основні переваги React:

- Компонентний підхід до розробки.
- Висока продуктивність завдяки використанню віртуального DOM.
- Велика спільнота розробників та багата екосистема.
- Можливість використання з іншими бібліотеками та фреймворками.

JSX

JSX – це синтаксичне розширення JavaScript, яке використовується в React для опису інтерфейсу. Воно дозволяє поєднувати JavaScript з HTML-подібним синтаксисом, що робить код більш читабельним та зрозумілим.

Основні можливості JSX:

- Зручний синтаксис для створення компонентів.
- Інтеграція з JavaScript.
- Підтримка умовного рендерингу та циклів.

Create React App

Create React App – це офіційний інструмент для швидкого створення проектів на базі React. Він забезпечує готову структуру проекту та налаштування середовища розробки, що дозволяє розробникам зосередитися на написанні коду.

Основні можливості Create React App:

- Швидке створення нових проектів.
- Готова конфігурація для розробки, тестування та збірки додатків.
- Підтримка сучасних можливостей JavaScript (ES6+).

Redux

Redux – це бібліотека для управління станом додатків, яка часто використовується разом з React. Вона дозволяє централізовано зберігати стан додатку та спрощує його керування.

Основні переваги Redux:

- Централізоване управління станом.
- Передбачувана поведінка додатку.
- Інструменти для дебагінгу та тестування.

React Router

React Router – це бібліотека для керування маршрутизацією у додатках на базі React. Вона дозволяє створювати багатосторінкові додатки з підтримкою навігації.

Основні можливості React Router:

- Декларативна маршрутизація.
- Підтримка динамічних маршрутів.
- Інтеграція з компонентами React.

Axios

Axios – це бібліотека для виконання HTTP-запитів, яка часто використовується у додатках на базі React для взаємодії з сервером. Вона підтримує обробку запитів та відповідей, а також роботу з обіцянками (promises).

Основні можливості Axios:

- Простий синтаксис для виконання HTTP-запитів.
- Підтримка обробки відповідей та помилок.
- Можливість налаштування запитів та використання інтерцепторів.

Material-UI

Material-UI – це бібліотека компонентів для React, яка реалізує принципи дизайну Material Design від Google. Вона забезпечує готові компоненти для створення стильних та зручних інтерфейсів.

Основні переваги Material-UI:

- Великий набір готових компонентів.
- Підтримка темізацій та стилів.
- Інтеграція з React.

Formik

Formik – це бібліотека для управління формами у додатках на базі React. Вона спрощує створення та валідацію форм, забезпечуючи зручний API для роботи з

ними.

Основні можливості Formik:

- Просте управління станом форм.
- Підтримка валідації та обробки помилок.
- Інтеграція з різними бібліотеками для валідації (наприклад, Yup).

Styled Components

Styled Components – це бібліотека для написання стилів у додатках на базі React, яка дозволяє використовувати CSS у JavaScript. Вона забезпечує динамічну зміну стилів та їхню прив'язку до компонентів.

Основні можливості Styled Components:

- Написання стилів у JavaScript.
- Підтримка темізації та динамічних стилів.
- Інтеграція з React.

Розробка веб-додатків на базі React забезпечує високу продуктивність, гнучкість та зручність у використанні. Використання додаткових інструментів та бібліотек, таких як JSX, Create React App, Redux, React Router, Axios, Material-UI, Formik та Styled Components, дозволяє створювати потужні та надійні веб-додатки, що відповідають сучасним вимогам та забезпечують високу якість користувацького досвіду.

2.5 Вибір бази даних для зберігання даних

Вибір бази даних є критично важливим етапом розробки будь-якого веб-додатку, оскільки від нього залежить ефективність зберігання, обробки та доступу до даних. У цьому розділі ми розглянемо декілька популярних баз даних, їхні переваги та недоліки, а також обґрунтуємо вибір конкретної бази даних для нашої системи рейтингування студентів.

SQLite

SQLite є вбудованою реляційною базою даних, яка не потребує окремого сервера для свого функціонування. Вона зберігає всі дані у одному файлі, що

робить її зручною для невеликих додатків або додатків, які потребують легкого розгортання.

Переваги:

- Простота налаштування та використання.
- Не потребує окремого сервера.
- Висока швидкість доступу до даних для невеликих обсягів.

Недоліки:

- Обмежені можливості для масштабування.
- Менша продуктивність для великих обсягів даних та високих навантажень.

MySQL

MySQL є однією з найпопулярніших реляційних баз даних, яка використовується у багатьох веб-додатках. Вона забезпечує високу продуктивність, надійність та підтримку різноманітних типів даних. [4]

Переваги:

- Висока продуктивність та надійність.
- Широкі можливості для масштабування.
- Велика спільнота розробників та багато інструментів для роботи.

Недоліки:

- Потребує налаштування та підтримки сервера.
- Більша складність у порівнянні з вбудованими базами даних.

PostgreSQL

PostgreSQL є потужною реляційною базою даних з відкритим вихідним кодом, яка забезпечує високу продуктивність, надійність та підтримку складних типів даних. Вона відома своєю відповідністю стандартам SQL та розширюваністю.

Переваги:

- Висока продуктивність та надійність.
- Підтримка складних типів даних та розширень.
- Велика спільнота розробників та багато інструментів для роботи.

Недоліки:

- Потребує налаштування та підтримки сервера.

- Більша складність у порівнянні з іншими реляційними базами даних.

MongoDB

MongoDB є нереляційною базою даних, яка зберігає дані у форматі JSON-подібних документів. Вона забезпечує високу гнучкість та масштабованість, що робить її популярною для додатків з великими обсягами даних та високими вимогами до продуктивності.

Переваги:

- Висока гнучкість та масштабованість.
- Простота використання для додатків з нереляційною структурою даних.
- Швидкий доступ до великих обсягів даних.

Недоліки:

- Менша відповідність стандартам SQL.
- Потребує налаштування та підтримки сервера.

Вибір бази даних для системи рейтингування студентів

Враховуючи специфіку системи рейтингування студентів, було обрано базу даних PostgreSQL. Це рішення обумовлено наступними факторами:

1. **Висока продуктивність та надійність:** PostgreSQL забезпечує стабільну роботу навіть при високих навантаженнях та великих обсягах даних, що є важливим для системи рейтингування студентів.
2. **Підтримка складних типів даних:** PostgreSQL дозволяє працювати з різноманітними типами даних та підтримує розширення, що робить її гнучкою у налаштуванні під конкретні потреби додатку.
3. **Широкі можливості для масштабування:** PostgreSQL легко масштабується, що дозволяє забезпечити зростання додатку без значних витрат на зміну інфраструктури.
4. **Велика спільнота розробників:** Наявність великої спільноти розробників та багатьох інструментів для роботи з PostgreSQL забезпечує швидке вирішення проблем та доступ до великої кількості ресурсів.

Таким чином, PostgreSQL є оптимальним вибором для нашої системи рейтингування студентів, забезпечуючи високу продуктивність, надійність та

гнучкість у роботі з даними. [5]

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Принципи розробки систем рейтингування

Системи рейтингування студентів відіграють важливу роль в освітніх закладах, оскільки вони сприяють об'єктивному оцінюванню знань та навичок студентів, забезпечують зворотний зв'язок і стимулюють академічну мотивацію. Для розробки ефективної системи рейтингування необхідно дотримуватися певних принципів, які забезпечують її коректну роботу та зручність використання. У цьому розділі ми розглянемо основні принципи, які необхідно враховувати при розробці таких систем.

Прозорість і об'єктивність

Одним з головних принципів систем рейтингування є прозорість та об'єктивність оцінювання. Це означає, що всі критерії оцінювання повинні бути чітко визначені і доступні для студентів та викладачів. Кожен студент повинен розуміти, як саме формується його рейтинг і які показники впливають на кінцеву оцінку.

Основні аспекти:

- Відкриті критерії оцінювання.
- Документовані правила та процедури.
- Доступ до історії оцінок та рейтингів.

Точність і достовірність даних

Для забезпечення точності рейтингів необхідно використовувати достовірні дані, які регулярно оновлюються. Це включає правильне введення оцінок, своєчасне оновлення інформації про курси та студентів, а також перевірку даних на відповідність.

Основні аспекти:

- Регулярне оновлення даних.
- Використання надійних джерел інформації.
- Перевірка та валідація введених даних.

Масштабованість і гнучкість

Система рейтингування повинна бути масштабованою, щоб забезпечити обробку великої кількості даних без втрати продуктивності. Вона також повинна бути гнучкою, щоб дозволяти додавання нових функцій та змінення існуючих алгоритмів без значних зусиль.

Основні аспекти:

- Підтримка великих обсягів даних.
- Можливість додавання нових модулів та функцій.
- Легкість внесення змін до існуючих алгоритмів.

Безпека і конфіденційність

Забезпечення безпеки даних є критично важливим аспектом будь-якої системи рейтингування. Це включає захист даних від несанкціонованого доступу, а також дотримання правил конфіденційності при обробці персональної інформації студентів.

Основні аспекти:

- Захист даних від несанкціонованого доступу.
- Шифрування конфіденційної інформації.
- Дотримання нормативних вимог щодо захисту даних.

Зручність і доступність

Система повинна бути зручною у використанні як для студентів, так і для викладачів. Інтерфейс має бути інтуїтивно зрозумілим, а всі функції легко доступними. Крім того, важливо забезпечити доступ до системи з різних пристроїв, включаючи мобільні телефони та планшети.

Основні аспекти:

- Інтуїтивно зрозумілий інтерфейс.
- Доступ з різних пристроїв.
- Підтримка різних мов інтерфейсу.

Мотивація і стимулювання

Ефективна система рейтингування повинна не лише оцінювати успішність студентів, але й стимулювати їх до покращення результатів. [6] Це може включати надання зворотного зв'язку, рекомендацій щодо поліпшення, а також визнання та

заохочення досягнень.

Основні аспекти:

- Надання зворотного зв'язку.
- Рекомендації щодо поліпшення результатів.
- Заохочення та визнання досягнень.

Розробка системи рейтингування студентів вимагає дотримання ряду принципів, які забезпечують її ефективність, надійність та зручність у використанні. Прозорість і об'єктивність оцінювання, точність і достовірність даних, масштабованість і гнучкість, безпека і конфіденційність, зручність і доступність, а також мотивація і стимулювання є ключовими аспектами, які необхідно враховувати при розробці таких систем. Використання цих принципів дозволить створити систему, яка не лише оцінює успішність студентів, але й сприяє їхньому розвитку та покращенню результатів.

3.2 Особливості використання Flask для створення веб-додатків

Flask – це популярний мікрофреймворк для створення веб-додатків на мові програмування Python. Він відомий своєю простотою, гнучкістю та легкістю використання, що робить його ідеальним вибором як для новачків, так і для досвідчених розробників. У цьому розділі ми розглянемо особливості використання Flask для створення веб-додатків, його основні компоненти та переваги.

Легкість та простота використання

Flask відомий своєю мінімалістичною архітектурою, що дозволяє розробникам швидко розпочати роботу. Основний принцип Flask – "принеси лише те, що тобі потрібно", що означає, що розробник може додавати лише ті компоненти та розширення, які необхідні для конкретного проекту.

Основні особливості:

- Простий стартовий шаблон.
- Легка інтеграція з іншими бібліотеками та інструментами.
- Мінімалістична структура проекту.

Гнучкість та розширюваність

Flask забезпечує високу гнучкість, дозволяючи розробникам легко налаштовувати та розширювати додаток відповідно до потреб. Існує безліч розширень для Flask, які додають додаткову функціональність, таку як аутентифікація, валідація форм, робота з базами даних та інше.

Основні розширення:

- Flask-SQLAlchemy для роботи з базами даних.
- Flask-WTF для валідації форм.
- Flask-Login для аутентифікації користувачів.
- Flask-Migrate для управління міграціями баз даних.

Потужний механізм маршрутизації

Flask має потужний та гнучкий механізм маршрутизації, який дозволяє розробникам легко визначати URL-шляхи для різних функцій додатку. Використання декораторів дозволяє зручно пов'язувати URL зі специфічними функціями обробки запитів.

Приклад маршрутизації:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Flask!"

@app.route('/about')
def about():
    return "This is the about page."
```

Інтеграція з шаблонізатором Jinja2

Flask використовує шаблонізатор Jinja2, який дозволяє створювати динамічні HTML-сторінки з використанням шаблонів. Jinja2 підтримує різні функції, такі як змінні, цикли та умовні конструкції, що робить його потужним інструментом для створення веб-інтерфейсів.

Приклад використання Jinja2:

```
<!doctype html>
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>{{ heading }}</h1>
    <p>{{ content }}</p>
  </body>
</html>
```

Підтримка розширень для тестування

Flask забезпечує підтримку різних інструментів та бібліотек для тестування веб-додатків, що дозволяє розробникам писати юніт-тести та функціональні тести для перевірки коректності роботи додатку.

Основні інструменти для тестування:

- pytest-flask для написання тестів.
- Flask-Testing для розширених можливостей тестування.

Підтримка конфігурації та налаштувань

Flask дозволяє легко управляти конфігураціями та налаштуваннями додатку, використовуючи конфігураційні файли або змінні середовища. Це забезпечує гнучкість у налаштуванні додатку для різних середовищ (розробка, тестування, продакшн).

Приклад конфігурації:

```
app.config['DEBUG'] = True
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'
```

Flask є потужним та гнучким інструментом для створення веб-додатків, який забезпечує легкість використання та розширюваність. Основні особливості Flask включають простий стартовий шаблон, гнучкий механізм маршрутизації, інтеграцію з шаблонізатором Jinja2, підтримку розширень для роботи з базами даних та валідації форм, а також підтримку інструментів для тестування. Завдяки своїм особливостям, Flask дозволяє швидко створювати прототипи та масштабувати додатки відповідно до зростаючих вимог проекту. [6]

3.3 Особливості використання React для створення інтерфейсів користувача

React – це популярна бібліотека JavaScript, розроблена компанією Facebook, яка використовується для створення динамічних та інтерактивних користувацьких інтерфейсів. React дозволяє розробникам створювати багаторазові компоненти, що спрощує розробку та підтримку складних додатків. У цьому розділі ми розглянемо основні особливості використання React для створення інтерфейсів користувача.

Компонентний підхід

Однією з головних особливостей React є компонентний підхід до розробки. Всі елементи інтерфейсу поділяються на незалежні компоненти, кожен з яких відповідає за свою частину інтерфейсу. Це дозволяє легко повторно використовувати компоненти та підтримувати код.

Основні переваги компонентного підходу:

- Повторне використання коду.
- Зручність у підтримці та масштабуванні.
- Інкапсуляція логіки та стилів у компонентах.

Приклад компонента:

```
import React from 'react';

function Welcome(props) {
  return <h1>Привіт, {props.name}</h1>;
}
```

JSX – синтаксичне розширення JavaScript

JSX – це синтаксичне розширення JavaScript, яке дозволяє розробникам писати HTML-подібний код всередині JavaScript. JSX спрощує створення компонентів та робить код більш читабельним.

Основні особливості JSX:

- Легкий синтаксис для опису інтерфейсу.
- Інтеграція з JavaScript-логікою.
- Підтримка умовного рендерингу та циклів.

Приклад JSX:

```
const element = <h1>Привіт, світ!</h1>;
```

Віртуальний DOM

React використовує віртуальний DOM для підвищення продуктивності додатків. Віртуальний DOM – це легка копія реального DOM, яка оновлюється React при зміні стану компонентів. Це дозволяє мінімізувати кількість операцій з реальним DOM, що підвищує швидкість роботи додатків.

Основні переваги віртуального DOM:

- Швидкі оновлення інтерфейсу.
- Мінімізація змін у реальному DOM.
- Підвищення продуктивності додатків.

Одностороння прив'язка даних

React використовує односторонню прив'язку даних, що означає, що дані передаються з батьківських компонентів до дочірніх через властивості (props). Це забезпечує передбачувану поведінку додатків та спрощує налагодження.

Основні переваги односторонньої прив'язки даних:

- Легкість у відстеженні потоку даних.
- Передбачувана поведінка додатків.
- Спрощення налагодження та підтримки.

Управління станом компонентів

React дозволяє керувати станом компонентів, використовуючи спеціальний об'єкт state. Стан компонентів може змінюватися у відповідь на дії користувачів, що дозволяє створювати динамічні інтерфейси.

Основні особливості управління станом:

- Використання об'єкта state для зберігання стану компонентів.
- Оновлення стану за допомогою методу setState.
- Рендеринг компонентів у відповідь на зміни стану.

Приклад управління станом:

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
```

```
  this.state = { count: 0 };
 }

 increment = () => {
  this.setState({ count: this.state.count + 1 });
 };

 render() {
  return (
   <div>
    <p>Кількість: {this.state.count}</p>
    <button onClick={this.increment}>Збільшити</button>
   </div>
  );
 }
}
```

React Hooks

React Hooks – це функції, які дозволяють використовувати стан та інші можливості React у функціональних компонентах. Hooks спрощують управління станом та інші аспекти життєвого циклу компонентів.

Основні hooks:

- useState для управління станом.
- useEffect для побічних ефектів.
- useContext для використання контексту.

Приклад використання hooks:

```
import React, { useState, useEffect } from 'react';

function Counter() {
 const [count, setCount] = useState(0);

 useEffect(() => {
  document.title = `Кількість: ${count}`;
 }, [count]);

 return (
  <div>
```

```
    <p>Кількість: {count}</p>
    <button onClick={() => setCount(count + 1)}>Збільшити</button>
   </div>
  );
}
```

Інтеграція з іншими бібліотеками та інструментами

React легко інтегрується з іншими бібліотеками та інструментами, що дозволяє розширювати функціональність додатків. Це включає використання таких інструментів, як Redux для управління станом, React Router для маршрутизації, Axios для HTTP-запитів та багато інших.

Приклад інтеграції з Redux:

```
import React from 'react';
import { createStore } from 'redux';
import { Provider } from 'react-redux';
import rootReducer from './reducers';
import App from './App';

const store = createStore(rootReducer);

function Root() {
  return (
    <Provider store={store}>
      <App />
    </Provider>
  );
}
```

React є потужним інструментом для створення динамічних та інтерактивних користувацьких інтерфейсів. Основні особливості React включають компонентний підхід, використання JSX, віртуальний DOM, односторонню прив'язку даних, управління станом компонентів, React Hooks та легку інтеграцію з іншими бібліотеками та інструментами. Завдяки цим особливостям, React дозволяє створювати високопродуктивні та масштабовані веб-додатки, які забезпечують чудовий користувацький досвід.

3.4 Методи забезпечення безпеки даних у веб-додатках

Забезпечення безпеки даних є одним із ключових аспектів розробки веб-додатків. Враховуючи постійні загрози кібератак та витоків даних, розробники повинні впроваджувати ефективні методи для захисту інформації користувачів. У цьому розділі ми розглянемо основні методи забезпечення безпеки даних у веб-додатках, які допоможуть захистити додаток від найпоширеніших загроз.

Шифрування даних

Шифрування є одним з найефективніших методів захисту даних. Воно передбачає перетворення даних у формат, який може бути прочитаний лише за наявності спеціального ключа дешифрування. Шифрування забезпечує захист даних як під час передачі, так і при зберіганні.

Основні аспекти:

- **SSL/TLS:** Використання SSL/TLS для шифрування даних, що передаються між клієнтом та сервером.
- **Шифрування на рівні бази даних:** Шифрування чутливої інформації, такої як паролі користувачів, перед зберіганням у базі даних.

Аутентифікація та авторизація

Аутентифікація та авторизація є критично важливими для захисту додатку від несанкціонованого доступу. Аутентифікація перевіряє особу користувача, тоді як авторизація визначає рівень доступу користувача до різних ресурсів додатку.

Основні методи:

- **JWT (JSON Web Tokens):** Використання JWT для безпечної передачі токенів аутентифікації.
- **OAuth:** Використання OAuth для забезпечення безпечного доступу до ресурсів додатку.
- **Двофакторна аутентифікація (2FA):** Додавання другого рівня перевірки для підвищення безпеки.

Захист від міжсайтових скриптів (XSS)

XSS-атаки передбачають впровадження шкідливого коду у веб-сторінки, які

потім виконуються на пристроях користувачів. Захист від XSS-атак включає валідацію та очищення введених даних.

Основні методи:

- **Валідація та фільтрація введених даних:** Перевірка та очищення всіх даних, що вводяться користувачами, перед їх обробкою.
- **Контекстуальне екранування:** Використання відповідних методів екранування для різних контекстів (HTML, JavaScript, CSS).

Захист від міжсайтових запитів підробки (CSRF)

CSRF-атаки використовують довірливі користувацькі сесії для виконання несанкціонованих дій від імені користувача. Захист від CSRF включає використання токенів, які перевіряються сервером для кожного запиту.

Основні методи:

- **CSRF-токени:** Додавання унікальних токенів до форм та запитів, які перевіряються сервером.
- **Перевірка HTTP-заголовків:** Перевірка заголовків Origin та Referer для виявлення підроблених запитів.

Захист даних на рівні бази даних

Захист бази даних є важливим аспектом безпеки веб-додатків. Це включає в себе налаштування прав доступу, шифрування даних та регулярне резервне копіювання.

Основні методи:

- **Налаштування прав доступу:** Обмеження доступу до бази даних лише для авторизованих користувачів та додатків.
- **Регулярні резервні копії:** Регулярне створення резервних копій даних для відновлення у разі їх втрати або пошкодження.
- **Моніторинг активності:** Відстеження та аналіз підозрілої активності у базі даних.

Захист від атак типу SQL Injection

SQL Injection атаки дозволяють зловмисникам виконувати несанкціоновані SQL-запити через введені дані. Захист від таких атак включає використання

параметризованих запитів та ORM (Object-Relational Mapping).

Основні методи:

- **Параметризовані запити:** Використання підготовлених запитів для передачі даних до бази даних.
- **ORM:** Використання ORM-бібліотек, таких як SQLAlchemy, для роботи з базою даних через об'єктно-орієнтовані моделі.

Безпека конфігурацій та налаштувань

Конфігурації та налаштування додатків можуть містити чутливу інформацію, таку як ключі API та паролі. Захист цих даних є важливим для загальної безпеки додатку.

Основні методи:

- **Використання змінних середовища:** Зберігання чутливих налаштувань у змінних середовища, а не у вихідному коді.
- **Шифрування конфігураційних файлів:** Шифрування конфігураційних файлів для захисту від несанкціонованого доступу.

Регулярне оновлення та патчинг

Регулярне оновлення додатків та бібліотек допомагає захистити додаток від відомих вразливостей. Важливо слідкувати за оновленнями та випускати патчі для усунення знайдених вразливостей. [7]

Основні методи:

- **Регулярне оновлення:** Встановлення останніх версій бібліотек та фреймворків.
- **Автоматизація процесу оновлення:** Використання інструментів для автоматичного відстеження та встановлення оновлень.

Забезпечення безпеки даних у веб-додатках вимагає комплексного підходу, який включає шифрування даних, аутентифікацію та авторизацію, захист від XSS та CSRF-атак, захист даних на рівні бази даних, захист від SQL Injection, безпеку конфігурацій та регулярне оновлення додатків. Дотримання цих методів допоможе забезпечити надійний захист даних користувачів та знизити ризики кіберзагроз.

3.5 Архітектурні рішення для веб-додатків

Вибір архітектури є одним з ключових рішень при розробці веб-додатків, оскільки він визначає, як компоненти додатку взаємодіятимуть між собою, як буде здійснюватися обробка даних, та як буде забезпечуватися масштабованість та продуктивність. У цьому розділі ми розглянемо основні архітектурні підходи для веб-додатків, їхні переваги та недоліки.

Монолітна архітектура

Монолітна архітектура передбачає розробку додатку як одного цілісного блоку, де всі компоненти взаємодіють безпосередньо. Це класичний підхід до розробки веб-додатків, який використовується багатьма традиційними додатками.

Переваги:

- Простота розробки та розгортання.
- Легше налагодження та тестування.
- Менше накладних витрат на зв'язок між компонентами.

Недоліки:

- Складнощі з масштабуванням окремих компонентів.
- Великий обсяг коду у одному проекті, що ускладнює підтримку.
- Можливі проблеми з надійністю та продуктивністю через взаємозалежності між компонентами.

Мікросервісна архітектура

Мікросервісна архітектура передбачає розбиття додатку на невеликі, незалежні сервіси, кожен з яких відповідає за виконання конкретних завдань. Кожен мікросервіс може бути розгорнутий, масштабований та оновлюваний незалежно від інших.

Переваги:

- Легкість у масштабуванні окремих компонентів.
- Підвищена надійність та стійкість до відмов.
- Можливість використання різних технологій та мов програмування для різних сервісів.

Недоліки:

- Складність управління розподіленою системою.
- Більші накладні витрати на зв'язок між сервісами.
- Складніша налагодження та тестування.

Архітектура з використанням API

Архітектура з використанням API передбачає чітке розділення між клієнтською та серверною частинами додатку, які взаємодіють через інтерфейси програмування додатків (API). Це дозволяє легко замінювати або оновлювати окремі компоненти без впливу на всю систему.

Переваги:

- Чітке розділення обов'язків між клієнтською та серверною частинами.
- Можливість легкої інтеграції з іншими системами та сервісами.
- Гнучкість у виборі технологій для клієнтської та серверної частин.

Недоліки:

- Потребує додаткових зусиль для забезпечення безпеки API.
- Можливі проблеми з продуктивністю через додаткові запити до API.
- Необхідність узгодження версій API між клієнтом та сервером.

Архітектура на основі MVC (Model-View-Controller)

MVC є одним з найбільш популярних архітектурних шаблонів для веб-додатків. Вона розділяє додаток на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Модель відповідає за роботу з даними, представлення – за відображення даних, а контролер – за обробку запитів користувачів.

Переваги:

- Чітке розділення обов'язків між компонентами.
- Підвищена зручність у підтримці та розширенні додатку.
- Зручність у тестуванні окремих компонентів.

Недоліки:

- Можливе збільшення складності через необхідність синхронізації між компонентами.

- Потребує досвіду у проектуванні архітектури для правильної реалізації.

Архітектура з використанням SPA (Single Page Application)

SPA передбачає створення веб-додатку, де вся навігація та оновлення контенту здійснюються без повного перезавантаження сторінки. Це досягається за допомогою JavaScript-фреймворків, таких як React, Angular або Vue.js.

Переваги:

- Швидка та плавна навігація між сторінками.
- Покращений користувацький досвід.
- Зниження навантаження на сервер завдяки меншим запитам.

Недоліки:

- Складність у SEO-оптимізації.
- Необхідність завантаження великого обсягу JavaScript-коду.
- Підвищені вимоги до безпеки та управління станом.

Архітектура на основі серверного рендерингу

Серверний рендеринг передбачає генерацію HTML-контенту на сервері перед його відправкою на клієнт. Це дозволяє знизити навантаження на клієнтську частину та покращити SEO-оптимізацію.

Переваги:

- Покращена SEO-оптимізація.
- Швидший час завантаження першої сторінки.
- Менше вимог до ресурсів клієнта.

Недоліки:

- Збільшення навантаження на сервер.
- Складність у реалізації інтерактивності.
- Обмежена гнучкість у порівнянні з SPA.

Вибір архітектури для веб-додатку залежить від багатьох факторів, таких як вимоги до продуктивності, масштабованості, зручності підтримки та специфічні потреби проекту. Монолітна архітектура підходить для невеликих проектів з обмеженими ресурсами, тоді як мікросервісна архітектура забезпечує гнучкість та масштабованість для великих систем. Використання API та архітектури MVC

дозволяє чітко розділити обов'язки між компонентами додатку, тоді як SPA та серверний рендеринг мають свої переваги у забезпеченні користувацького досвіду та SEO-оптимізації. Ретельний аналіз вимог та особливостей проекту допоможе обрати оптимальне архітектурне рішення для розробки веб-додатку.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Проектування системи рейтингування

Проектування системи рейтингування студентів включає кілька етапів, які забезпечують створення ефективного, зручного та безпечного додатку. У цьому розділі ми розглянемо основні етапи проектування системи, включаючи аналіз вимог, розробку архітектури, вибір технологій та інструментів, а також планування функціональності.

Аналіз вимог

Першим етапом проектування є аналіз вимог до системи. Це включає збір та аналіз вимог від користувачів та зацікавлених сторін, визначення ключових функцій системи та специфікацію технічних вимог.

Основні вимоги:

- Вхід користувачів за допомогою PIN-коду.
- Створення, редагування та видалення студентів.
- Додавання, редагування та видалення оцінок студентів.
- Перегляд списку студентів та їхніх оцінок.
- Створення рейтингів студентів за середніми балами.
- Забезпечення безпеки даних користувачів.

Розробка архітектури

На основі аналізу вимог розробляється архітектура системи. Вибір архітектури залежить від масштабу проекту, вимог до продуктивності, масштабованості та підтримки. Для даного проекту було обрано архітектуру з використанням Flask для серверної частини та React для клієнтської частини.

Основні компоненти архітектури:

- **Бекенд:** Flask для обробки запитів, управління базою даних та забезпечення аутентифікації користувачів.
- **Фронтенд:** React для створення динамічного інтерфейсу користувача.
- **База даних:** SQLite для зберігання даних про студентів та їхні оцінки.

Вибір технологій та інструментів

Для реалізації проекту було обрано наступні технології та інструменти:

Бекенд:

- **Flask:** Легкий веб-фреймворк для Python, який забезпечує зручну обробку HTTP-запитів та управління даними.
- **Flask-Login:** Розширення для забезпечення аутентифікації користувачів.
- **Flask-SQLAlchemy:** ORM-бібліотека для роботи з базою даних.
- **Flask-Migrate:** Інструмент для управління міграціями бази даних.

Фронтенд:

- **React:** Бібліотека для створення інтерфейсів користувача, яка дозволяє створювати багаторазові компоненти та динамічні інтерфейси.
- **Axios:** Бібліотека для виконання HTTP-запитів з клієнтської частини.

База даних:

- **SQLite:** Легка вбудована база даних, яка забезпечує зручне зберігання даних та не потребує налаштування серверної частини.

Планування функціональності

Наступним етапом є планування функціональності системи, що включає визначення основних модулів та функцій додатку. Основні модулі включають:

Модуль аутентифікації:

- Вхід користувачів за допомогою PIN-коду.
- Логування та розлогування користувачів.

Модуль управління студентами:

- Додавання нових студентів.
- Редагування даних студентів.
- Видалення студентів.

Модуль управління оцінками:

- Додавання нових оцінок для студентів.
- Редагування існуючих оцінок.
- Видалення оцінок.

Модуль перегляду даних:

- Перегляд списку студентів та їхніх оцінок.
- Створення рейтингів студентів за середніми балами.

Структура бази даних

Для зберігання даних було розроблено структуру бази даних, яка включає три основні таблиці: **users**, **students**, **grades**.

Таблиця users:

- **id:** Унікальний ідентифікатор користувача.
- **pin:** PIN-код для аутентифікації.

Таблиця students:

- **id:** Унікальний ідентифікатор студента.
- **first_name:** Ім'я студента.
- **last_name:** Прізвище студента.
- **middle_name:** По батькові студента.
- **group:** Група, до якої належить студент.

Таблиця grades:

- **id:** Унікальний ідентифікатор оцінки.
- **student_id:** Ідентифікатор студента.
- **course:** Назва курсу.
- **grade:** Оцінка.
- **semester:** Семестр, за який виставлена оцінка.

Проектування системи рейтингування студентів включає аналіз вимог, розробку архітектури, вибір технологій та інструментів, а також планування функціональності. Обрані технології, такі як Flask для бекенду та React для фронтенду, забезпечують гнучкість та зручність у розробці, а також високу продуктивність та масштабованість додатку. Правильне проектування структури бази даних забезпечує ефективне зберігання та обробку даних, необхідних для роботи системи рейтингування.

4.2. Реалізація системи

Реалізація системи рейтингування студентів включає створення серверної та клієнтської частин додатку. Серверна частина, побудована на Flask, відповідає за обробку запитів, управління даними та аутентифікацію користувачів. Клієнтська частина, створена на основі React, забезпечує динамічний та інтерактивний інтерфейс користувача. У цьому розділі ми розглянемо основні етапи реалізації системи.

Реалізація серверної частини (бекенду)

Серверна частина реалізована на Flask і включає наступні компоненти:

1. Налаштування проекту: Створення базової структури проекту та налаштування необхідних залежностей.

Основні файли та каталоги:

- **app/**: Каталог з основним кодом додатку.
- **app/__init__.py**: Ініціалізація додатку Flask.
- **app/models.py**: Моделі бази даних.
- **app/routes.py**: Маршрути та обробники запитів.
- **app/extensions.py**: Ініціалізація розширень Flask.
- **config.py**: Файл конфігурації додатку.

```
# app/__init__.py
from flask import Flask
from .extensions import db, login_manager
from .models import User, Student, Grade
from .routes import init_app

def create_app():
    app = Flask(__name__)
    app.config.from_object('config.Config')

    db.init_app(app)
    login_manager.init_app(app)

    with app.app_context():
        db.create_all()

    init_app(app)
```

```
    return app
```

2. Моделі бази даних: Створення моделей бази даних для зберігання даних про користувачів, студентів та їх оцінки.

```
# app/models.py
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin

db = SQLAlchemy()

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    pin = db.Column(db.String(100), unique=True, nullable=False)

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    middle_name = db.Column(db.String(100), nullable=False)
    group = db.Column(db.String(100), nullable=False)

class Grade(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('student.id'), nullable=False)
    course = db.Column(db.String(100), nullable=False)
    grade = db.Column(db.Float, nullable=False)
    semester = db.Column(db.Integer, nullable=False)

    student = db.relationship('Student', backref=db.backref('grades', lazy=True))
```

3. Аутентифікація та авторизація: Реалізація механізмів аутентифікації користувачів за допомогою PIN-коду та захисту маршрутів.

```
# app/routes.py
from flask import Blueprint, request, jsonify
from flask_login import login_user, logout_user, login_required, current_user
from .models import User, Student, Grade
from .extensions import db, login_manager

auth_bp = Blueprint('auth_bp', __name__)
```

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    pin = data['pin']
    user = User.query.filter_by(pin=pin).first()
    if user:
        login_user(user)
        return jsonify({'message': 'Logged in successfully'}), 200
    return jsonify({'message': 'Invalid PIN'}), 401

@auth_bp.route('/logout')
@login_required
def logout():
    logout_user()
    return jsonify({'message': 'Logged out successfully'}), 200

@auth_bp.route('/user', methods=['GET'])
@login_required
def get_user():
    return jsonify({'user': current_user.pin}), 200
```

4. Управління студентами та оцінками: Реалізація маршрутів для додавання, редагування та видалення студентів та їхніх оцінок.

```
# app/routes.py
students_bp = Blueprint('students_bp', __name__)

@students_bp.route('/students', methods=['POST'])
@login_required
def add_student():
    try:
        data = request.get_json()
        new_student = Student(
            first_name=data['first_name'],
            last_name=data['last_name'],
```

```
            middle_name=data['middle_name'],
            group=data['group']
        )
        db.session.add(new_student)
        db.session.commit()
        return jsonify({'message': 'Student added successfully'}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Error adding student', 'error': str(e)}), 500

@students_bp.route('/students/<int:student_id>', methods=['PUT'])
@login_required
def update_student(student_id):
    try:
        data = request.get_json()
        student = Student.query.get(student_id)
        if not student:
            return jsonify({'message': 'Студента не знайдено'}), 404

        student.first_name = data['first_name']
        student.last_name = data['last_name']
        student.middle_name = data['middle_name']
        student.group = data['group']

        db.session.commit()
        return jsonify({'message': 'Студента успішно оновлено'}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Помилка при оновленні студента', 'error': str(e)}), 500

@students_bp.route('/students/<int:student_id>', methods=['DELETE'])
@login_required
def delete_student(student_id):
    try:
        student = Student.query.get(student_id)
        if not student:
            return jsonify({'message': 'Студента не знайдено'}), 404
```

```
        db.session.delete(student)
        db.session.commit()
        return jsonify({'message': 'Студента успішно видалено'}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Помилка при видаленні студента', 'error': str(e)}), 500

@students_bp.route('/grades', methods=['POST'])
@login_required
def add_grade():
    try:
        data = request.get_json()
        new_grade = Grade(
            student_id=data['student_id'],
            course=data['course'],
            grade=data['grade'],
            semester=data['semester']
        )
        db.session.add(new_grade)
        db.session.commit()
        return jsonify({'message': 'Grade added successfully'}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Error adding grade', 'error': str(e)}), 500

@students_bp.route('/grades/<int:grade_id>', methods=['PUT'])
@login_required
def update_grade(grade_id):
    try:
        data = request.get_json()
        grade = Grade.query.get(grade_id)
        if not grade:
            return jsonify({'message': 'Оцінку не знайдено'}), 404

        grade.course = data['course']
        grade.grade = data['grade']
        grade.semester = data['semester']
```

```
        db.session.commit()
        return jsonify({'message': 'Оцінку успішно оновлено'}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Помилка при оновленні оцінки', 'error': str(e)}), 500

@students_bp.route('/grades/<int:grade_id>', methods=['DELETE'])
@login_required
def delete_grade(grade_id):
    try:
        grade = Grade.query.get(grade_id)
        if not grade:
            return jsonify({'message': 'Оцінку не знайдено'}), 404

        db.session.delete(grade)
        db.session.commit()
        return jsonify({'message': 'Оцінку успішно видалено'}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': 'Помилка при видаленні оцінки', 'error': str(e)}), 500
```

5. Створення рейтингів студентів: Реалізація маршрутів для створення рейтингів студентів на основі їх середніх балів.

```
# app/routes.py
rankings_bp = Blueprint('rankings_bp', __name__)

@rankings_bp.route('/rankings', methods=['GET'])
@login_required
def get_rankings():
    group = request.args.get('group')
    course = request.args.get('course')
    semester = request.args.get('semester')

    query = db.session.query(Student, Grade).join(Grade).filter(Grade.semester == semester)
    if group:
        query = query.filter(Student.group == group)
    if course:
        query = query.filter(Grade.course == course)
```

```
    results = query.all()
    student_grades = {}

    for student, grade in results:
        if student.id not in student_grades:
            student_grades[student.id] = {'name': f"{student.first_name} {student.last_name}", 'grades': []}
        student_grades[student.id]['grades'].append(grade.grade)

    top_students = sorted(student_grades.items(), key=lambda x: sum(x[1]['grades']) / len(x[1]['grades']), reverse=True)[:10]
    overall_students = sorted(student_grades.items(), key=lambda x: sum(x[1]['grades']) / len(x[1]['grades']), reverse=True)

    top_students = [{'student_id': student_id, 'name': data['name'], 'average_grade': sum(data['grades']) / len(data['grades'])} for student_id, data in top_students]
    overall_students = [{'student_id': student_id, 'name': data['name'], 'average_grade': sum(data['grades']) / len(data['grades'])} for student_id, data in overall_students]

    return jsonify({'top_students': top_students, 'overall_students': overall_students}), 200
```

Реалізація клієнтської частини (фронтенду)

Клієнтська частина реалізована на React і включає наступні компоненти:

1. Налаштування проекту: Створення базової структури проекту та налаштування необхідних залежностей.

```
npx create-react-app student-rating-system
cd student-rating-system
npm install axios react-router-dom
```

2. Компоненти для аутентифікації: Реалізація компонентів для входу користувачів за допомогою PIN-коду.

```
// src/components/Login.js
import React, { useState } from 'react';
import axios from 'axios';

const Login = () => {
  const [pin, setPin] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
```

```
    try {
      const response = await axios.post('http://localhost:5000/api/login', { pin });
      alert(response.data.message);
    } catch (error) {
      alert('Помилка при вході');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="pin">PIN-код:</label>
      <input
        type="password"
        id="pin"
        value={pin}
        onChange={(e) => setPin(e.target.value)}
      />
      <button type="submit">Увійти</button>
    </form>
  );
};

export default Login;
```

3. Компоненти для управління студентами та оцінками: Реалізація компонентів для додавання, редагування та видалення студентів та їхніх оцінок.

```
// src/components/StudentList.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const StudentList = () => {
  const [students, setStudents] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/students');
        setStudents(response.data);
      } catch (error) {
```

```
      alert('Помилка при отриманні студентів');
    }
  };
  fetchData();
 }, []);

 return (
  <div>
   <h2>Список студентів</h2>
   <ul>
    {students.map((student) => (
     <li key={student.id}>{student.first_name} {student.last_name} ({student.group})</li>
    ))}
   </ul>
  </div>
 );
};

export default StudentList;
```

4. Компоненти для створення рейтингів: Реалізація компонентів для створення рейтингів студентів на основі їх середніх балів.

```
// src/components/CreateRankings.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const CreateRankings = () => {
 const [criteria, setCriteria] = useState({ group: '', course: '', semester: '' });
 const [rankings, setRankings] = useState({ top_students: [], overall_students: [] });
 const [groups, setGroups] = useState([]);
 const [courses, setCourses] = useState([]);

 useEffect(() => {
  const fetchData = async () => {
   try {
    const groupsResponse = await axios.get('http://localhost:5000/api/groups');
    const coursesResponse = await axios.get('http://localhost:5000/api/courses');
    setGroups(groupsResponse.data);
    setCourses(coursesResponse.data);
```

```
    } catch (error) {
      alert('Помилка при отриманні даних');
    }
  };
  fetchData();
}, []);

const handleChange = (e) => {
  const { name, value } = e.target;
  setCriteria({ ...criteria, [name]: value });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.get('http://localhost:5000/api/rankings', { params: criteria });
    setRankings(response.data);
  } catch (error) {
    alert('Помилка при створенні рейтингу');
  }
};

return (
  <div>
    <h2>Створити рейтинги</h2>
    <form onSubmit={handleSubmit}>
      <label htmlFor="group">Група (необов'язково):</label>
      <select name="group" value={criteria.group} onChange={handleChange}>
        <option value="">Виберіть групу</option>
        {groups.map((group) => (
          <option key={group} value={group}>{group}</option>
        ))}
      </select>
      <label htmlFor="course">Курс (необов'язково):</label>
      <select name="course" value={criteria.course} onChange={handleChange}>
        <option value="">Виберіть курс</option>
        {courses.map((course) => (
          <option key={course} value={course}>{course}</option>
```

```
            ))}
          </select>
          <label htmlFor="semester">Семестр:</label>
          <input type="number" name="semester" value={criteria.semester} onChange={handleChange} required />
          <button type="submit">Створити рейтинги</button>
        </form>
        <div>
          <h3>Топ 10 студентів</h3>
          <ul>
            {rankings.top_students.map((student) => (
              <li key={student.student_id}>{student.name}: {student.average_grade.toFixed(2)}</li>
            ))}
          </ul>
        </div>
        <div>
          <h3>Загальний рейтинг</h3>
          <ul>
            {rankings.overall_students.map((student) => (
              <li key={student.student_id}>{student.name}: {student.average_grade.toFixed(2)}</li>
            ))}
          </ul>
        </div>
      </div>
    );
  };

  export default CreateRankings;
```

Реалізація системи рейтингування студентів включає створення серверної частини на Flask та клієнтської частини на React. Серверна частина забезпечує обробку запитів, управління даними та аутентифікацію користувачів, тоді як клієнтська частина надає динамічний та інтерактивний інтерфейс для користувачів. Реалізовані функціональні можливості включають управління студентами та оцінками, створення рейтингів та забезпечення безпеки даних. Такий підхід дозволяє створити гнучку та масштабовану систему, яка відповідає вимогам сучасних веб-додатків.

4.3. Інструкція для користувача

Цей розділ містить детальні інструкції для користувачів щодо встановлення, налаштування та використання веб-додатку системи рейтингування студентів. Інструкція охоплює всі основні функції програми, включаючи вхід за допомогою PIN-коду, управління студентами та їхніми оцінками, створення рейтингів та перегляд інформації.

Встановлення та налаштування додатку

1. **Встановлення додатку:**
    - **Скачайте та розпакуйте проект:** Завантажте архів з кодом додатку з надійного джерела та розпакуйте його у зручному місці.
    - **Налаштування віртуального оточення:** Створіть віртуальне оточення за допомогою команди **python -m venv venv** та активуйте його:
        - **Для Windows: venv\Scripts\activate**
        - **Для Linux/Mac: source venv/bin/activate**
    - **Встановлення залежностей:** Встановіть необхідні залежності, виконавши команду: **pip install -r requirements.txt**.
    - **Ініціалізація бази даних:** Ініціалізуйте базу даних за допомогою команд:

flask db init
flask db migrate -m "Initial migration"
flask db upgrade

    - **Запуск додатку:** Запустіть додаток командою: **flask run**. Додаток буде доступний за адресою **http://127.0.0.1:5000**.

Використання додатку

2. **Вхід за допомогою PIN-коду:**
    - **Вхід у систему:** Відкрийте головну сторінку додатку. Введіть ваш PIN-код у відповідне поле та натисніть кнопку "Увійти". (див. рис. 4.1)

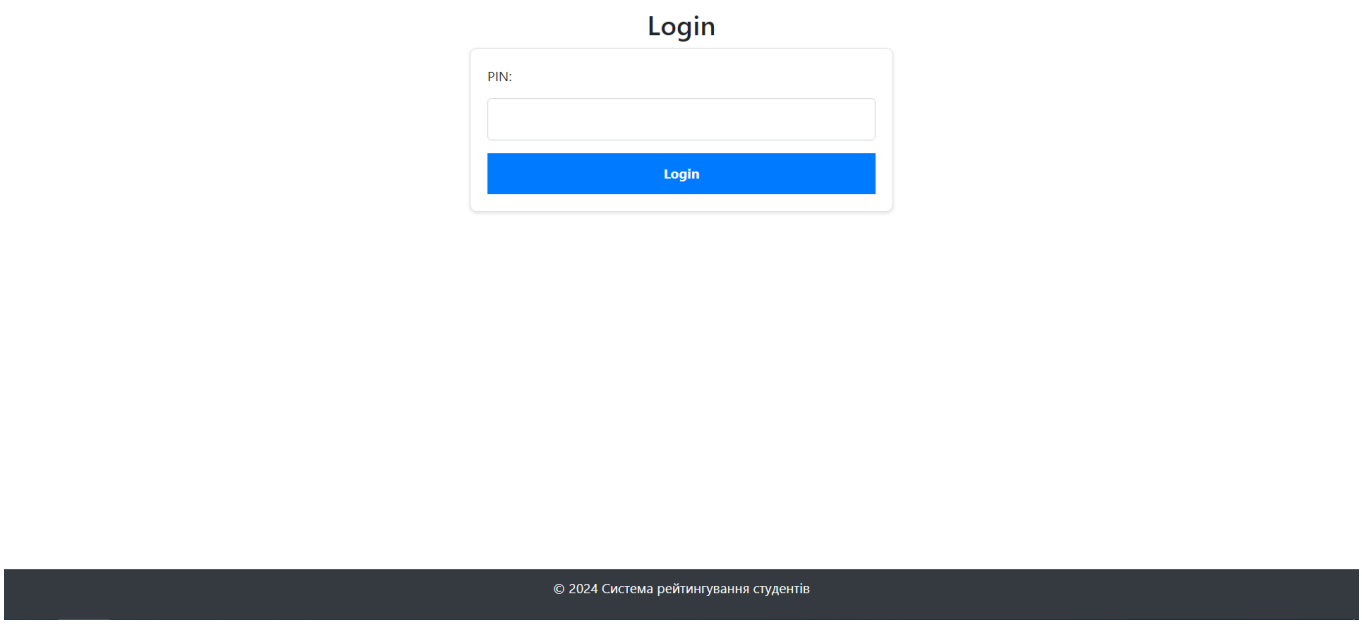

Рисунок 4.1 – Екран Логіну

3. **Управління студентами:**
    - **Додавання студента:** На головній сторінці або в меню навігації натисніть кнопку "Додати студента". Заповніть форму додавання студента, вказавши ім'я, прізвище, по батькові та групу. Натисніть кнопку "Додати". (див. рис. 4.2)

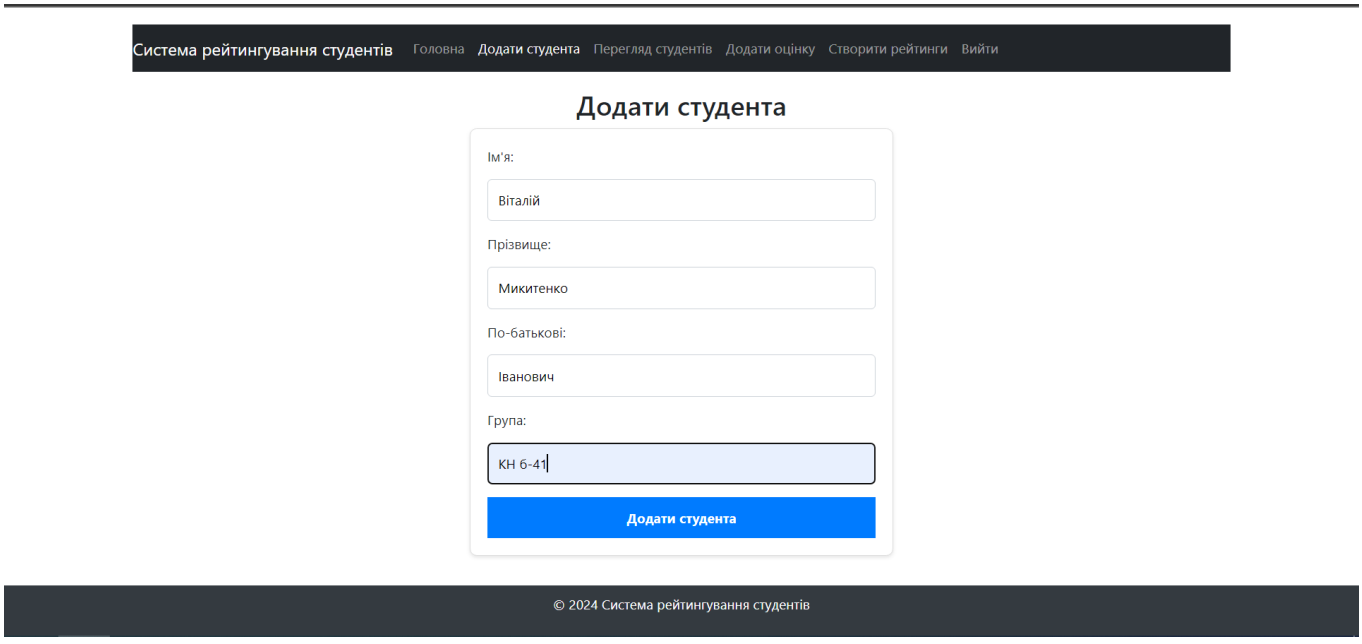

Рисунок 4.2 – Додати студента

- **Редагування студента:** На сторінці зі списком студентів натисніть кнопку "Редагувати" біля студента, якого ви хочете змінити. Внесіть необхідні зміни у форму та натисніть кнопку "Зберегти зміни". (див. рис. 4.3)

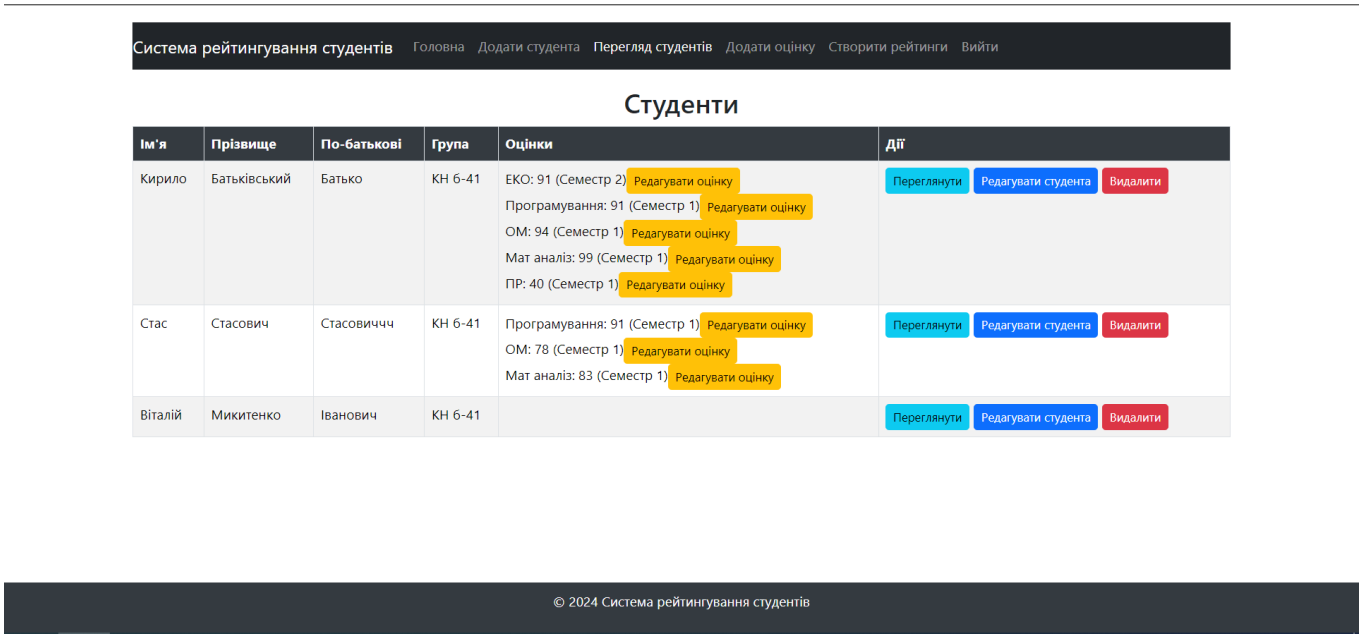

Рисунок 4.3 – Дії над списком

- **Видалення студента:** На сторінці зі списком студентів натисніть кнопку "Видалити" біля студента, якого ви хочете видалити. Підтвердіть видалення у спливаючому вікні. (див. рис. 4.3)

4. **Управління оцінками:**
   - **Додавання оцінки:** На сторінці профілю студента або в меню навігації натисніть кнопку "Додати оцінку". Заповніть форму додавання оцінки, вказавши курс, оцінку та семестр. Натисніть кнопку "Додати". (див. рис. 4.4)

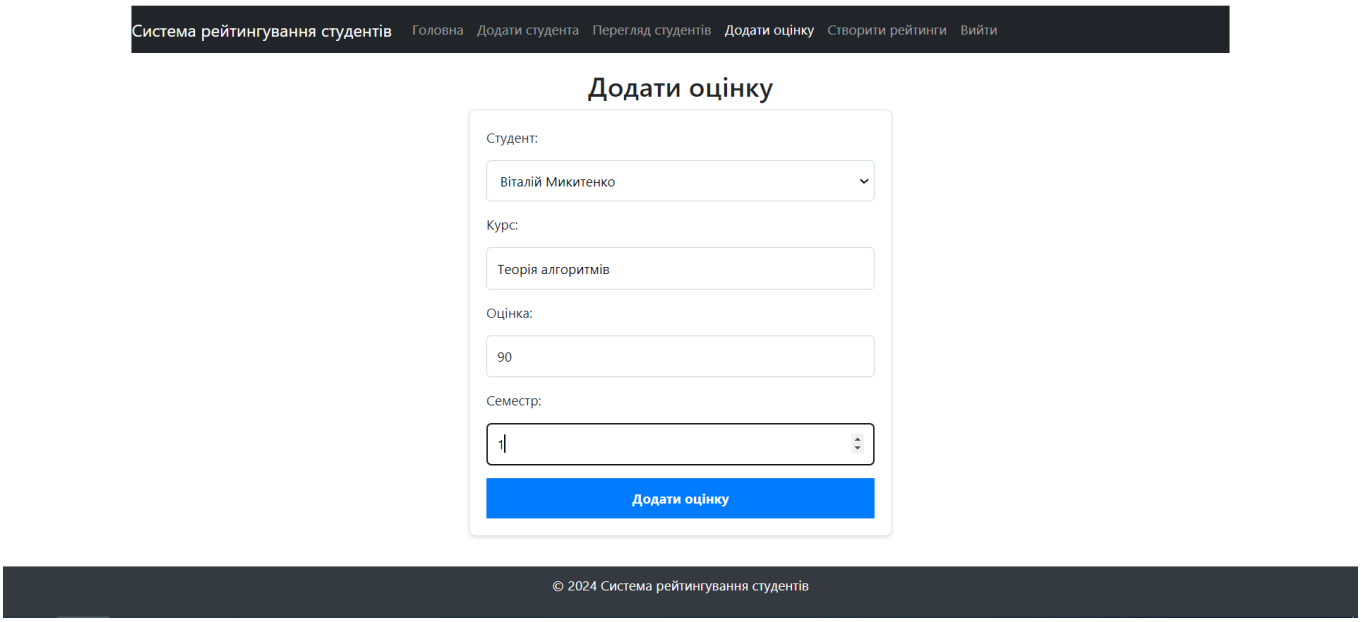

Рисунок 4.4 – Додати оцінку

- **Редагування оцінки:** На сторінці профілю студента натисніть кнопку "Редагувати" біля оцінки, яку ви хочете змінити. Внесіть необхідні зміни у форму та натисніть кнопку "Зберегти зміни". (див. рис. 4.3)
- **Видалення оцінки:** На сторінці профілю студента натисніть кнопку "Видалити" біля оцінки, яку ви хочете видалити. Підтвердіть видалення у спливаючому вікні. (див. рис. 4.3)

5. **Створення рейтингів:**
   - **Створення рейтингу:** На головній сторінці або в меню навігації натисніть кнопку "Створити рейтинг". Заповніть форму створення рейтингу, вказавши групу, курс та семестр. Натисніть кнопку "Створити". (див. рис. 4.5) Рейтинг буде створено на основі середніх балів студентів.

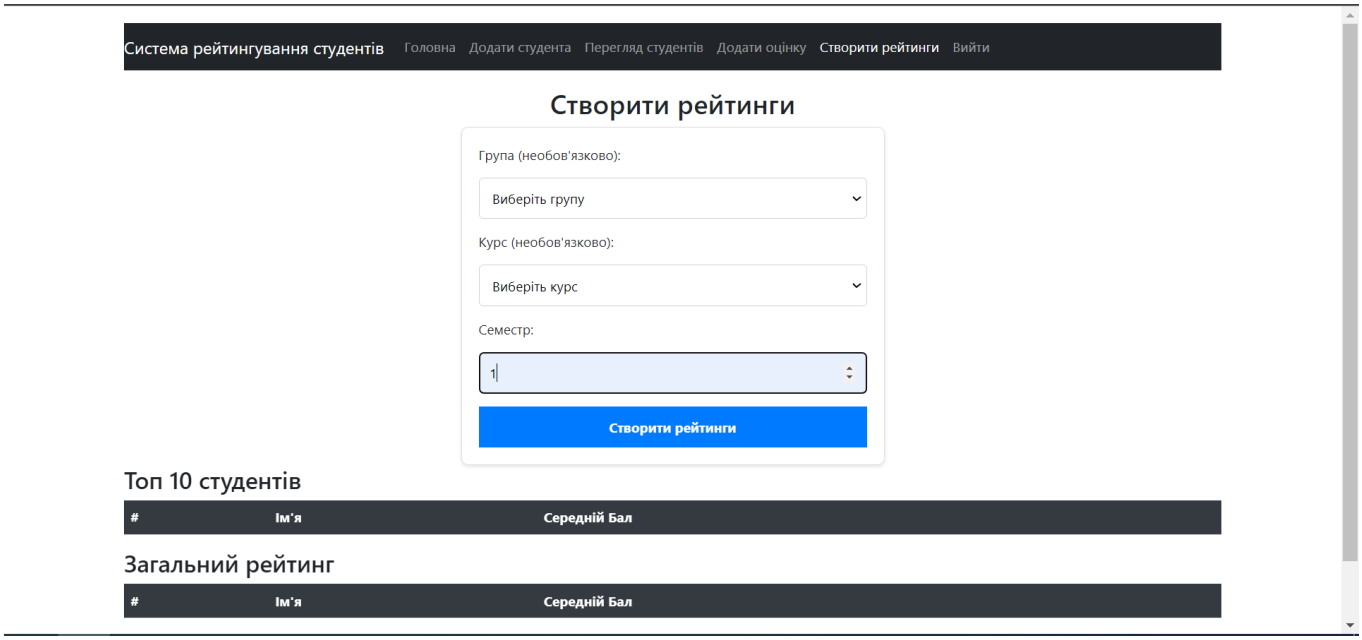

Рисунок 4.5 – Створити рейтинг

- **Перегляд рейтингу:** Після створення рейтингу на сторінці буде відображено список топ-10 студентів за середнім балом та загальний рейтинг студентів. (див. рис. 4.6)

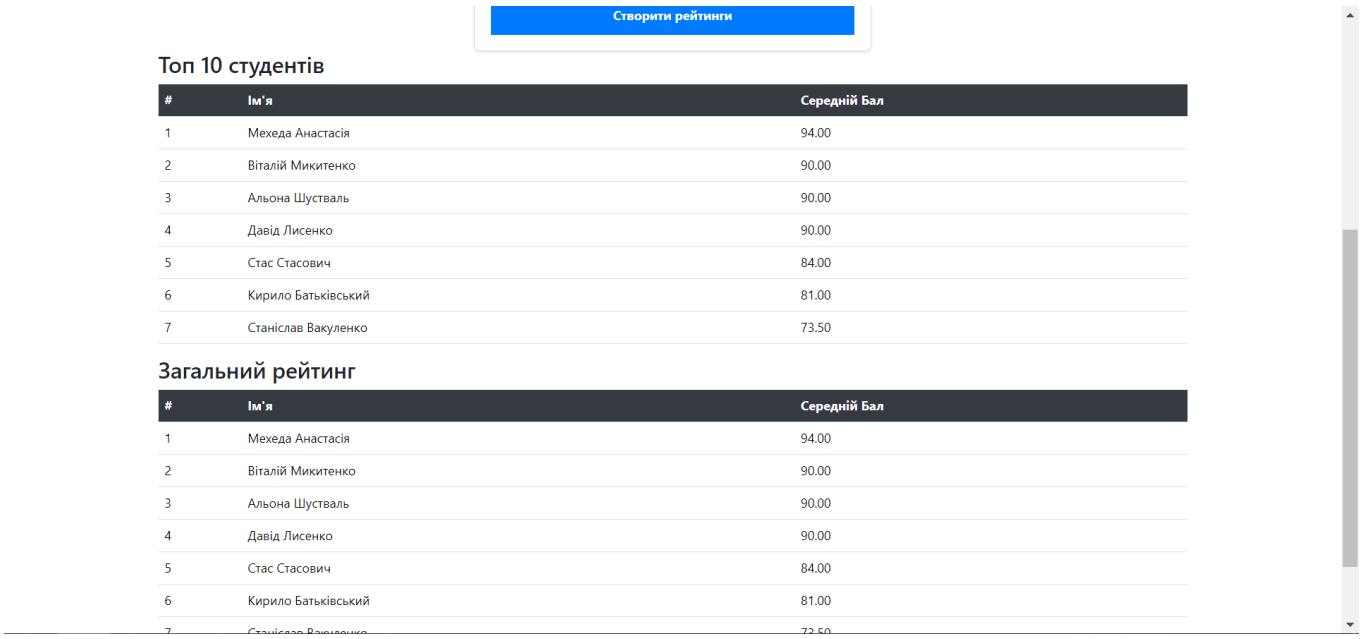

Рисунок 4.6 – Рейтинг студентів

6. **Перегляд інформації:**
   - **Перегляд студентів:** На головній сторінці натисніть кнопку "Список студентів" або перейдіть до відповідного розділу в меню навігації. Ви

побачите список усіх студентів та їхні середні бали. (див. рис. 4.3)

- **Перегляд профілю студента:** Натисніть на ім'я студента у списку для перегляду детальної інформації про нього та його оцінки. (див. рис. 4.7)

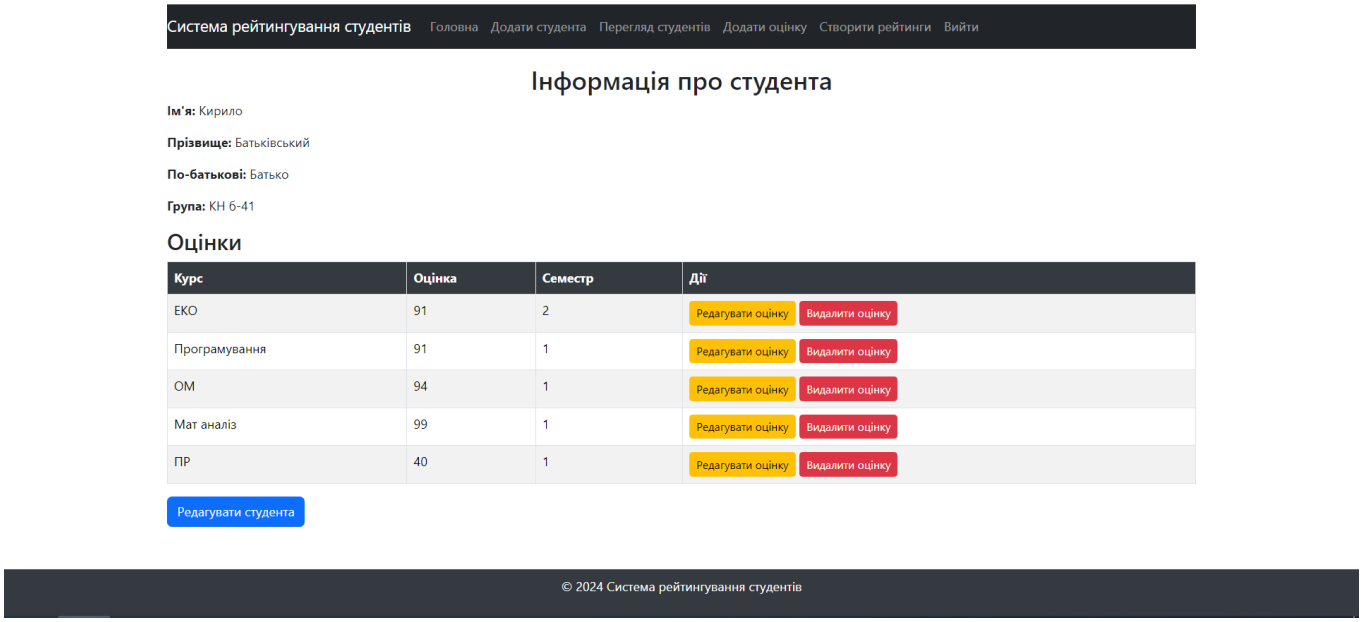

Рисунок 4.7 – Інформація про студента

Ця інструкція надає користувачам необхідні знання для встановлення, налаштування та використання системи рейтингування студентів. Дотримуючись вказаних кроків та рекомендацій, користувачі можуть ефективно управляти даними про студентів, їхні оцінки та створювати рейтинги для аналізу успішності студентів.

ВИСНОВКИ

Під час виконання дипломної роботи був детально розглянутий теоретичний матеріал з теми «Розробка системи рейтингування студентів спеціальності "Комп’ютерні науки" з використанням мови програмування Python та фреймворку Flask». Було проведено огляд сучасних методів та інструментів розробки веб-додатків, а також реалізовано систему рейтингування, що включає широкий спектр функцій для забезпечення ефективного обліку та аналізу успішності студентів.

Результати роботи:

1. Проведено інформаційний огляд літературних джерел та робіт з розробки веб-додатків на платформі Flask та інших аналогічних платформах.
2. Виділено позитивні та негативні сторони різних методів розробки веб-додатків та систем рейтингування.
3. Обґрунтовано вибір технологій для розробки веб-додатку, включаючи мову програмування Python, фреймворк Flask, базу даних SQLite, а також інструменти для фронтенд-розробки на React.
4. Розроблено веб-додаток системи рейтингування студентів, який включає функції реєстрації та автентифікації користувачів за допомогою PIN-коду, додавання, редагування та видалення студентів та їхніх оцінок, створення рейтингів за середнім балом.
5. Реалізовано та оптимізовано модулі для керування студентами та їхніми оцінками, що забезпечують ефективне зберігання та обробку даних у базі даних.
6. Створено інтерфейс користувача, який забезпечує зручний спосіб взаємодії з додатком, включаючи можливість додавання, редагування, видалення студентів та їхніх оцінок, а також створення рейтингів.
7. Проведено відладку та виправлення помилок у веб-додатку, використовуючи інструменти налагодження та тестування.
8. Проведено тестування веб-додатку для перевірки коректності його роботи, зручності використання та стійкості проти потенційних загроз.

9. Розроблено інструктаж для користувачів щодо встановлення, налаштування та використання системи рейтингування студентів.

Робота дозволила глибше зрозуміти принципи розробки веб-додатків на базі Flask та React, а також забезпечення безпеки даних у веб-додатках. Було розроблено практичний інструмент, який може бути використаний для підвищення ефективності обліку та аналізу успішності студентів, а також для навчальних та дослідницьких цілей у сфері розробки веб-додатків.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Miguel Grinberg. Flask Documentation. Доступно онлайн за адресою: https://flask.palletsprojects.com/en/2.0.x/
2. Krishna Rungta. Learn Flask Framework. Guru99, 2022. Доступно онлайн за адресою: https://www.guru99.com/flask-tutorial.html
3. Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018. 258 с.
4. Michael Herman. Mastering Flask Web Development. Packt Publishing, 2018. 400 с.
5. Robin Wieruch. The Road to React. 2020. Доступно онлайн за адресою: https://www.roadtoreact.com/
6. Dave Ceddia. Pure React: A step-by-step guide to mastering React. 2020. Доступно онлайн за адресою: https://daveceddia.com/pure-react/
7. Alex Banks, Eve Porcello. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020. 310 с.
8. Roy Derks. React Projects. Packt Publishing, 2020. 370 с.
9. Eric Matthes. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544 с.
10. William S. Vincent. Python Web Development with Flask. 2018. 200 с.
11. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, 2015. 792 с.
12. Jake VanderPlas. Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media, 2016. 548 с.
13. J. Burton Browning, Marty Alchin. Pro Python 3: Features and Tools for Professional Development. Apress, 2019. 468 с.
14. Brad Miller, David Ranum. Problem Solving with Algorithms and Data Structures Using Python. Franklin, Beedle & Associates Inc., 2011. 428 с.
15. Tony Gaddis. Starting Out with Python. Pearson, 2018. 744 с.
16. Mark Lutz. Learning Python, 5th Edition. O'Reilly Media, 2013. 1648 с.

17. Official React Documentation. Доступно онлайн за адресою: https://reactjs.org/docs/getting-started.html
18. SQLite Documentation. Доступно онлайн за адресою: https://sqlite.org/docs.html
19. Sean Larkin. Modern Front-End Development with React. Manning Publications, 2021. 375 с.
20. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2022. – 67 с. – 1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

AddGrade.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import Navbar from './Navbar';

const AddGrade = () => {
  const [students, setStudents] = useState([]);
  const [grade, setGrade] = useState({
    student_id: '',
    course: '',
    grade: '',
    semester: ''
  });

  useEffect(() => {
    const fetchStudents = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/students', {
          withCredentials: true,
        });
        setStudents(response.data);
      } catch (error) {
        alert('Помилка при отриманні студентів');
      }
    };

    fetchStudents();
  }, []);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setGrade({ ...grade, [name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:5000/api/grades', grade, {
        headers: {
          'Content-Type': 'application/json',
        },
        withCredentials: true,
      });
      alert('Оцінку додано успішно');
      setGrade({ student_id: '', course: '', grade: '', semester: '' });
    } catch (error) {
          console.error("Помилка при додаванні оцінки: ", error.response ? error.response.data :
```

```
error.message);
        alert('Помилка при додаванні оцінки');
      }
    };

    return (
      <div className="container">
        <Navbar />
        <h2>Додати оцінку</h2>
        <form onSubmit={handleSubmit}>
          <label htmlFor="student_id">Студент:</label>
          <select name="student_id" value={grade.student_id} onChange={handleChange} required>
            <option value="">Виберіть студента</option>
            {students.map(student => (
              <option key={student.id} value={student.id}>
                {student.first_name} {student.last_name}
              </option>
            ))}
          </select>
          <label htmlFor="course">Курс:</label>
          <input
            type="text"
            name="course"
            value={grade.course}
            onChange={handleChange}
            required
          />
          <label htmlFor="grade">Оцінка:</label>
          <input
            type="number"
            name="grade"
            value={grade.grade}
            onChange={handleChange}
            required
          />
          <label htmlFor="semester">Семестр:</label>
          <input
            type="number"
            name="semester"
            value={grade.semester}
            onChange={handleChange}
            required
          />
          <button type="submit">Додати оцінку</button>
        </form>
      </div>
    );
  };

  export default AddGrade;
```

```
AddStudent.js
import React, { useState } from 'react';
import axios from 'axios';
import Navbar from './Navbar';

const AddStudent = () => {
  const [student, setStudent] = useState({
    first_name: '',
    last_name: '',
    middle_name: '',
    group: ''
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setStudent({ ...student, [name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:5000/api/students', student, {
        headers: {
          'Content-Type': 'application/json',
        },
        withCredentials: true,
      });
      alert('Студента додано успішно');
      setStudent({ first_name: '', last_name: '', middle_name: '', group: '' });
    } catch (error) {
        console.error("Помилка при додаванні студента: ", error.response ? error.response.data :
error.message);
      alert('Помилка при додаванні студента');
    }
  };

  return (
    <div className="container">
      <Navbar />
      <h2>Додати студента</h2>
      <form onSubmit={handleSubmit}>
        <label htmlFor="first_name">Ім'я:</label>
        <input
          type="text"
          name="first_name"
          value={student.first_name}
          onChange={handleChange}
          required
        />
        <label htmlFor="last_name">Прізвище:</label>
```

```
        <input
          type="text"
          name="last_name"
          value={student.last_name}
          onChange={handleChange}
          required
        />
        <label htmlFor="middle_name">По-батькові:</label>
        <input
          type="text"
          name="middle_name"
          value={student.middle_name}
          onChange={handleChange}
          required
        />
        <label htmlFor="group">Група:</label>
        <input
          type="text"
          name="group"
          value={student.group}
          onChange={handleChange}
          required
        />
        <button type="submit">Додати студента</button>
      </form>
    </div>
  );
};

export default AddStudent;
CreateRankings.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import Navbar from './Navbar';

const CreateRankings = () => {
  const [criteria, setCriteria] = useState({
    group: '',
    course: '',
    semester: ''
  });
  const [rankings, setRankings] = useState({
    top_students: [],
    overall_students: []
  });
  const [groups, setGroups] = useState([]);
  const [courses, setCourses] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
```

```
    try {
      const groupsResponse = await axios.get('http://localhost:5000/api/groups', {
        withCredentials: true,
      });
      const coursesResponse = await axios.get('http://localhost:5000/api/courses', {
        withCredentials: true,
      });
      setGroups(groupsResponse.data);
      setCourses(coursesResponse.data);
    } catch (error) {
      alert('Помилка при отриманні даних');
    }
  };

  fetchData();
}, []);

const handleChange = (e) => {
  const { name, value } = e.target;
  setCriteria({ ...criteria, [name]: value });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.get('http://localhost:5000/api/rankings', {
      params: criteria,
      withCredentials: true,
    });
    setRankings(response.data);
  } catch (error) {
    alert('Помилка при створенні рейтингу');
  }
};

const formatGrade = (grade) => {
  return grade !== null && grade !== undefined ? grade.toFixed(2) : 'N/A';
};

return (
  <div className="container">
    <Navbar />
    <h2>Створити рейтинги</h2>
    <form onSubmit={handleSubmit}>
      <label htmlFor="group">Група (необов'язково):</label>
      <select name="group" value={criteria.group} onChange={handleChange}>
        <option value="">Виберіть групу</option>
        {groups.map(group => (
          <option key={group} value={group}>{group}</option>
        ))}
```

```
  </select>
  <label htmlFor="course">Курс (необов'язково):</label>
  <select name="course" value={criteria.course} onChange={handleChange}>
   <option value="">Виберіть курс</option>
   {courses.map(course => (
    <option key={course} value={course}>{course}</option>
   ))}
  </select>
  <label htmlFor="semester">Семестр:</label>
  <input
   type="number"
   name="semester"
   value={criteria.semester}
   onChange={handleChange}
   required
  />
  <button type="submit">Створити рейтинги</button>
 </form>
 <div>
  <h3>Топ 10 студентів</h3>
  <table className="table">
   <thead>
    <tr>
     <th>#</th>
     <th>Ім'я</th>
     <th>Середній Бал</th>
    </tr>
   </thead>
   <tbody>
    {rankings.top_students.map((student, index) => (
     <tr key={student.student_id}>
      <td>{index + 1}</td>
      <td>{student.name}</td>
      <td>{formatGrade(student.average_grade)}</td>
     </tr>
    ))}
   </tbody>
  </table>
 </div>
 <div>
  <h3>Загальний рейтинг</h3>
  <table className="table">
   <thead>
    <tr>
     <th>#</th>
     <th>Ім'я</th>
     <th>Середній Бал</th>
    </tr>
   </thead>
   <tbody>
```

```
          {rankings.overall_students.map((student, index) => (
            <tr key={student.student_id}>
              <td>{index + 1}</td>
              <td>{student.name}</td>
              <td>{formatGrade(student.average_grade)}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  </div>
 );
};

export default CreateRankings;
routes.py
from flask import Blueprint, request, jsonify
from flask_login import login_user, logout_user, login_required, current_user
from .models import User, Student, Course, Grade
from .extensions import db, login_manager
from sqlalchemy import func

auth_bp = Blueprint('auth_bp', __name__)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    pin = data['pin']
    user = User.query.filter_by(pin=pin).first()
    if user:
        login_user(user)
        return jsonify({'message': 'Logged in successfully'}), 200
    return jsonify({'message': 'Invalid PIN'}), 401

@auth_bp.route('/logout')
@login_required
def logout():
    logout_user()
    return jsonify({'message': 'Logged out successfully'}), 200

@auth_bp.route('/user', methods=['GET'])
@login_required
def get_user():
    return jsonify({'user': current_user.pin}), 200

students_bp = Blueprint('students_bp', __name__)
```

```
@students_bp.route('/students', methods=['POST'])
@login_required
def add_student():
    try:
        data = request.get_json()
        new_student = Student(
            first_name=data['first_name'],
            last_name=data['last_name'],
            middle_name=data['middle_name'],
            group=data['group']
        )
        db.session.add(new_student)
        db.session.commit()
        return jsonify({'message': 'Student added successfully'}), 201
    except Exception as e:
        db.session.rollback()
        print(f"Error adding student: {e}")
        return jsonify({'message': 'Error adding student', 'error': str(e)}), 500

@students_bp.route('/students', methods=['GET'])
@login_required
def get_students():
    students = Student.query.all()
    result = []
    for student in students:
        grades = Grade.query.filter_by(student_id=student.id).all()
         grades_list = [{'id': grade.id, 'course': Course.query.get(grade.course_id).name, 'grade': grade.grade,
'semester': grade.semester} for grade in grades]
        student_data = {
            'id': student.id,
            'first_name': student.first_name,
            'last_name': student.last_name,
            'middle_name': student.middle_name,
            'group': student.group,
            'grades': grades_list
        }
        result.append(student_data)
    return jsonify(result), 200

@students_bp.route('/students/<int:student_id>', methods=['GET'])
@login_required
def get_student(student_id):
    try:
        student = Student.query.get(student_id)
        if not student:
            return jsonify({'message': 'Студента не знайдено'}), 404

        grades = Grade.query.filter_by(student_id=student.id).all()
         grades_list = [{'id': grade.id, 'course': Course.query.get(grade.course_id).name, 'grade': grade.grade,
```

```
'semester': grade.semester} for grade in grades]

            student_data = {
                'id': student.id,
                'first_name': student.first_name,
                'last_name': student.last_name,
                'middle_name': student.middle_name,
                'group': student.group,
                'grades': grades_list
            }
            return jsonify(student_data), 200
        except Exception as e:
            print(f"Error fetching student: {e}")
            return jsonify({'message': 'Помилка при отриманні студента', 'error': str(e)}), 500

    @students_bp.route('/grades', methods=['POST'])
    @login_required
    def add_grade():
        try:
            data = request.get_json()
            course = Course.query.filter_by(name=data['course']).first()
            if not course:
                course = Course(name=data['course'])
                db.session.add(course)
                db.session.commit()

            new_grade = Grade(
                student_id=data['student_id'],
                course_id=course.id,
                grade=data['grade'],
                semester=data['semester']
            )
            db.session.add(new_grade)
            db.session.commit()
            return jsonify({'message': 'Grade added successfully'}), 201
        except Exception as e:
            db.session.rollback()
            print(f"Error adding grade: {e}")
            return jsonify({'message': 'Error adding grade', 'error': str(e)}), 500

    @students_bp.route('/grades/<int:grade_id>', methods=['GET'])
    @login_required
    def get_grade(grade_id):
        try:
            grade = Grade.query.get(grade_id)
            if not grade:
                return jsonify({'message': 'Оцінку не знайдено'}), 404

            grade_data = {
                'id': grade.id,
```

```
            'student_id': grade.student_id,
            'course': Course.query.get(grade.course_id).name,
            'grade': grade.grade,
            'semester': grade.semester
        }
        return jsonify(grade_data), 200
    except Exception as e:
        print(f"Error fetching grade: {e}")
        return jsonify({'message': 'Помилка при отриманні оцінки', 'error': str(e)}), 500

@students_bp.route('/students/<int:student_id>', methods=['PUT'])
@login_required
def update_student(student_id):
    try:
        data = request.get_json()
        student = Student.query.get(student_id)
        if not student:
            return jsonify({'message': 'Студента не знайдено'}), 404

        student.first_name = data['first_name']
        student.last_name = data['last_name']
        student.middle_name = data['middle_name']
        student.group = data['group']

        db.session.commit()
        return jsonify({'message': 'Студента успішно оновлено'}), 200
    except Exception as e:
        db.session.rollback()
        print(f"Error updating student: {e}")
        return jsonify({'message': 'Помилка при оновленні студента', 'error': str(e)}), 500

@students_bp.route('/students/<int:student_id>', methods=['DELETE'])
@login_required
def delete_student(student_id):
    try:
        student = Student.query.get(student_id)
        if not student:
            return jsonify({'message': 'Студента не знайдено'}), 404

        grades = Grade.query.filter_by(student_id=student.id).all()
        for grade in grades:
            db.session.delete(grade)

        db.session.delete(student)
        db.session.commit()
        return jsonify({'message': 'Студента успішно видалено'}), 200
    except Exception as e:
        db.session.rollback()
        print(f"Error deleting student: {e}")
        return jsonify({'message': 'Помилка при видаленні студента', 'error': str(e)}), 500
```

```
@students_bp.route('/grades/<int:grade_id>', methods=['PUT'])
@login_required
def update_grade(grade_id):
    try:
        data = request.get_json()
        grade = Grade.query.get(grade_id)
        if not grade:
            return jsonify({'message': 'Оцінку не знайдено'}), 404

        course = Course.query.filter_by(name=data['course']).first()
        if not course:
            course = Course(name=data['course'])
            db.session.add(course)
            db.session.commit()

        grade.course_id = course.id
        grade.grade = data['grade']
        grade.semester = data['semester']

        db.session.commit()
        return jsonify({'message': 'Оцінку успішно оновлено'}), 200
    except Exception as e:
        db.session.rollback()
        print(f"Error updating grade: {e}")
        return jsonify({'message': 'Помилка при оновленні оцінки', 'error': str(e)}), 500

@students_bp.route('/grades/<int:grade_id>', methods=['DELETE'])
@login_required
def delete_grade(grade_id):
    try:
        grade = Grade.query.get(grade_id)
        if not grade:
            return jsonify({'message': 'Оцінку не знайдено'}), 404

        db.session.delete(grade)
        db.session.commit()
        return jsonify({'message': 'Оцінку успішно видалено'}), 200
    except Exception as e:
        db.session.rollback()
        print(f"Error deleting grade: {e}")
        return jsonify({'message': 'Помилка при видаленні оцінки', 'error': str(e)}), 500

rankings_bp = Blueprint('rankings_bp', __name__)

@rankings_bp.route('/rankings', methods=['GET'])
@login_required
def get_rankings():
    group = request.args.get('group')
    course_name = request.args.get('course')
```

```
        semester = request.args.get('semester')

        subquery = db.session.query(
            Grade.student_id,
            func.avg(Grade.grade).label('average_grade')
        ).join(Student, Student.id == Grade.student_id)

        if semester:
            subquery = subquery.filter(Grade.semester == semester)
        if group:
            subquery = subquery.filter(Student.group == group)
        if course_name:
                subquery = subquery.join(Course, Course.id == Grade.course_id).filter(Course.name ==
course_name)

        subquery = subquery.group_by(Grade.student_id).subquery()

        query = db.session.query(
            Student.id,
            Student.first_name,
            Student.last_name,
            func.coalesce(subquery.c.average_grade, 0).label('average_grade')
        ).outerjoin(subquery, Student.id == subquery.c.student_id)

        results = query.order_by(subquery.c.average_grade.desc()).all()

            top_students = [{'student_id': student.id, 'name': f"{student.first_name} {student.last_name}",
'average_grade': student.average_grade} for student in results[:10]]
            overall_students = [{'student_id': student.id, 'name': f"{student.first_name} {student.last_name}",
'average_grade': student.average_grade} for student in results]

        return jsonify({'top_students': top_students, 'overall_students': overall_students}), 200

    @students_bp.route('/groups', methods=['GET'])
    @login_required
    def get_groups():
        groups = db.session.query(Student.group).distinct().all()
        return jsonify([group[0] for group in groups]), 200

    @students_bp.route('/courses', methods=['GET'])
    @login_required
    def get_courses():
        courses = Course.query.all()
        return jsonify([course.name for course in courses]), 200

    @students_bp.route('/summary', methods=['GET'])
    @login_required
    def get_summary():
        try:
            num_students = Student.query.count()
```

```
        num_courses = Course.query.count()
                                    latest_grades = db.session.query(Grade, Student,
Course).join(Student).join(Course).order_by(Grade.id.desc()).limit(5).all()

        latest_grades_data = [
            {
                'student_name': f"{grade.Student.first_name} {grade.Student.last_name}",
                'course_name': grade.Course.name,
                'grade': grade.Grade.grade,
                'semester': grade.Grade.semester
            }
            for grade in latest_grades
        ]

        summary_data = {
            'num_students': num_students,
            'num_courses': num_courses,
            'latest_grades': latest_grades_data
        }

        return jsonify(summary_data), 200
    except Exception as e:
        print(f"Error fetching summary data: {e}")
        return jsonify({'message': 'Error fetching summary data', 'error': str(e)}), 500

def init_app(app):
    app.register_blueprint(auth_bp, url_prefix='/api')
    app.register_blueprint(students_bp, url_prefix='/api')
    app.register_blueprint(rankings_bp, url_prefix='/api')
models.py
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from .extensions import db

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    pin = db.Column(db.String(100), unique=True, nullable=False)

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    middle_name = db.Column(db.String(100), nullable=False)
    group = db.Column(db.String(100), nullable=False)

class Course(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True, nullable=False)

class Grade(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('student.id'), nullable=False)
    course_id = db.Column(db.Integer, db.ForeignKey('course.id'), nullable=False)
    grade = db.Column(db.Integer, nullable=False)
    semester = db.Column(db.Integer, nullable=False)

    student = db.relationship('Student', backref=db.backref('grades', lazy=True))
    course = db.relationship('Course', backref=db.backref('grades', lazy=True))
extensions.py
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

db = SQLAlchemy()
login_manager = LoginManager()
login_manager.login_view = 'auth_bp.login'
__init__.py
from flask import Flask
from flask_cors import CORS
from .extensions import db, login_manager

def create_app():
    app = Flask(__name__)
    app.config.from_object('config.Config')

    db.init_app(app)
    CORS(app, supports_credentials=True)  # Дозволити обмін куками
    login_manager.init_app(app)

    with app.app_context():
        from .models import User, Student, Course, Grade
        db.create_all()

        # Створення користувача з пін-кодом
        if User.query.filter_by(pin='1234').first() is None:
            user = User(pin='1234')
            db.session.add(user)
            db.session.commit()

    from .routes import init_app as routes_init_app
    routes_init_app(app)

    return app
config.py
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'your_secret_key'
    SQLALCHEMY_DATABASE_URI = 'sqlite:///students.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
Run.py
```

```
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True)
```