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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система аналітики успішності студентів. Спец-частина: розробка Backend-частини системи»

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

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

_____ Святослав КАПЛЮЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Святослав КАПЛЮЧЕНКО

Керівник: Ігор ЦАПРО
викладач
Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Замрій І.В.

« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Каплюченку Святославу Романовичу

1. Тема кваліфікаційної роботи: «Система аналітики успішності студентів. Спец-частина: розробка Backend-частини системи»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Ігор ЦАПРО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про архітектуру клієнт-серверних застосунків, принципи побудови RESTful API, документація до фреймворку FastAPI та бібліотеки Pandas, опис роботи з нереляційними базами даних MongoDB, підходи до реалізації JWT-авторизації та розмежування прав доступу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі систем аналітики успішності студентів та визначення вимог до серверної частини.
2. Огляд та обґрунтування вибору технологій для реалізації серверної частини системи.

3. Проєктування архітектури серверної частини системи аналітики успішності студентів.
4. Реалізація та тестування серверної частини системи.

5. Перелік графічного матеріалу: презентація
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма послідовності створення аналітики.
6. Структура бази даних.
7. Архітектура застосунку.
8. Діаграма компонентів серверної логіки.
9. Діаграма класів авторизації.
10. Мапи веб-застосунку.
11. Тестування серверної частини.
12. Екранні форми.
13. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Аналіз предметної галузі та визначення вимог	07.03-14.03.2026	
4	Огляд і обґрунтування вибору технологій	15.03-21.03.2026	
5	Проєктування архітектури серверної частини	21.03-31.03.2026	
6	Реалізація серверної частини системи	01.04-20.04.2026	
7	Тестування та налагодження	21.04-30.04.2026	
8	Оформлення роботи: вступ, висновки, реферат	01.05-07.05.2026	
9	Розробка демонстраційних матеріалів	07.05-10.05.2026	
10	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Святослав КАПЛЮЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ЦАПРО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 стор., 2 табл., 26 рис., 17 джерел.

Метою роботи є спрощення процесу аналізу успішності студентів шляхом розробки веб-застосунку для збереження, обробки та візуалізації освітніх даних.

Об’єктом дослідження є процес аналізу успішності студентів на основі збереження, обробки та статистичного аналізу освітніх даних.

Предметом дослідження є методи агрегації даних, нереляційні бази даних та технології створення REST API з використанням FastAPI, MongoDB та Pandas.

У роботі розглянуто особливості систем контролю академічної успішності студентів та проведено аналіз існуючих програмних аналогів, зокрема Canvas LMS, Moodle та Blackboard Predict. Виконано порівняння їх функціональних можливостей, технічних особливостей, переваг і недоліків.

У процесі розробки обґрунтовано вибір технологічного стеку, до якого входять мова програмування Python, веб-фреймворк FastAPI, система керування базами даних MongoDB Atlas, бібліотека Pandas, технологія контейнеризації Docker та хмарна платформа Render.

У роботі спроєктовано архітектуру серверної частини системи, структуру бази даних, діаграми класів та схему розподілу прав доступу користувачів. Реалізовано REST API із JWT-авторизацією та розмежуванням ролей адміністратора і викладача. Також реалізовано CRUD-модулі для роботи зі студентами, предметами та оцінками, механізм масового імпорту студентів із CSV-файлів, а також аналітичний модуль для розрахунку середнього балу та визначення студентів групи ризику.

Розроблена система призначена для автоматизації обліку та аналізу академічної діяльності студентів у закладах вищої освіти. Актуальність теми обумовлена необхідністю автоматизації процесів обробки освітніх даних,

формування статистичних звітів та своєчасного виявлення студентів із низькими показниками успішності.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ПРОГРАМНА ІНЖЕНЕРІЯ, АНАЛІЗ УСПІШНОСТІ СТУДЕНТІВ, ОБРОБКА ОСВІТНІХ ДАНИХ, АНАЛІТИКА ОСВІТНІХ ДАНИХ, СИСТЕМА МОНІТОРИНГУ СТУДЕНТІВ, REST API, JWT-АВТЕНТИФІКАЦІЯ, PYTHON, FASTAPI, MONGODB, PANDAS, DOCKER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - Програмне забезпечення

API - Application Programming Interface (інтерфейс прикладного програмування)

REST - Representational State Transfer (передача репрезентативного стану)

HTTP - Hypertext Transfer Protocol (протокол передачі гіпертексту)

JWT - JSON Web Token (веб-токен на основі JSON)

JSON - JavaScript Object Notation (формат обміну даними)

ODM - Object-Document Mapping (об'єктно-документне відображення)

CRUD - Create, Read, Update, Delete (базові операції з даними)

NoSQL - Not Only SQL (нереляційні бази даних)

CORS - Cross-Origin Resource Sharing (спільне використання ресурсів між різними джерелами)

CSV - Comma-Separated Values (значення, розділені комами)

IDE - Integrated Development Environment (інтегроване середовище розробки)

ЗВО - Заклад вищої освіти

ВСТУП

Актуальність теми. Заклади вищої освіти щоденно накопичують значний обсяг інформації про результати навчальної діяльності студентів. Традиційні способи обробки таких даних часто потребують ручного формування звітів, що ускладнює оперативний аналіз успішності та своєчасне виявлення студентів із низькими навчальними показниками. У зв'язку з цим виникає потреба у використанні сучасних інформаційних систем, здатних автоматизувати процеси збору, збереження та аналізу освітніх даних.

Використання веб-технологій, хмарних баз даних та засобів аналітичної обробки інформації дозволяє спростити моніторинг успішності студентів, прискорити формування статистичних звітів та підвищити ефективність роботи адміністрації навчального закладу. Особливо актуальним є завдання своєчасного виявлення студентів, які мають низькі показники успішності та можуть належати до групи ризику.

Об'єктом дослідження є процес аналізу успішності студентів на основі збереження, обробки та статистичного аналізу освітніх даних.

Предметом дослідження є технології, алгоритми агрегації, інструменти створення back-end частин систем аналітики освітніх даних.

Метою роботи є спрощення процесу аналізу успішності студентів за допомогою впровадження комплексної системи для безпечного збереження, агрегації та візуалізації освітніх даних.

Для досягнення поставленої мети у роботі проведено аналіз предметної галузі та існуючих програмних аналогів, визначено функціональні й нефункціональні вимоги до системи, обґрунтовано вибір технологій для реалізації серверної частини, спроєктовано архітектуру застосунку та структуру бази даних, реалізовано REST API, механізм авторизації користувачів і модуль аналітичної обробки даних, а також проведено тестування функціональності системи.

Методи дослідження. У роботі використано методи аналізу предметної галузі, методи проєктування клієнт-серверних систем, технології створення REST API, методи обробки та агрегації даних, а також засоби тестування програмного забезпечення.

Наукова новизна роботи полягає у розробці серверної частини системи аналізу успішності студентів, яка поєднує централізоване збереження освітніх даних, автоматизовану обробку статистичних показників та механізм визначення студентів групи ризику на основі аналізу середнього балу.

Практичне значення одержаних результатів полягає у можливості використання розробленої системи для автоматизації процесів обліку та аналізу успішності студентів у закладах вищої освіти. Реалізована система дозволяє централізовано зберігати освітні дані, формувати статистичні звіти та оперативно виявляти студентів із низькими показниками успішності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Основні особливості проблеми відстеження академічної успішності студентів

Контроль успішності студентів є важливим елементом управління освітнім процесом у закладах вищої освіти. Вчасне отримання правильної інформації про академічні досягнення дозволяє керівництву та викладачам приймати обґрунтовані рішення щодо організації навчання та надання індивідуальної допомоги студентам.

У традиційному вигляді оцінки записуються в паперові або електронні журнали, де викладачі фіксують бали за певні завдання. Деканат збирає інформацію з різних кафедр та створює загальний звіт. Такий підхід потребує значних витрат часу, може містити помилки, що зроблені людиною, і не дозволяє проводити аналіз в режимі реального часу.

Аналіз освітніх даних є одним із важливих напрямів сучасних інформаційних технологій у сфері освіти. Основною метою такого аналізу є виявлення закономірностей у навчальній діяльності студентів, оцінка рівня успішності та формування статистичних показників на основі накопичених даних. Використання засобів автоматизованої обробки інформації дозволяє оперативно визначати студентів із низькими результатами навчання, формувати аналітичні звіти та спрощувати процес моніторингу освітнього процесу.

Головна проблема сучасних підходів полягає в тому, що немає автоматичної системи для виявлення групи ризику - студентів, рівень успішності яких може свідчити про ймовірність відрахування. За даними різних джерел, від 15 до 30 відсотків студентів перших та других курсів стикаються з проблемами навчання, але таких учнів часто виявляють досить пізно.

З точки зору інформаційних технологій, проблема відстеження успішності може бути вирішена за допомогою розробки автоматизованої системи, яка має виконувати наступні функції: зберігання освітньої інформації в одному місці; автоматичне обчислення загальних показників продуктивності; визначення студентів, які потрапляють до групи ризику; надання інтерфейсу у вигляді REST API для роботи з клієнтськими програмами; створення звітів для потреб адміністрації.

Архітектура клієнт-серверної програми має чітке розділення обов'язків між різними частинами. Сервер відповідає за виконання відповідних правил, зберігання інформації та надання API для взаємодії. Клієнт відображає дані та дозволяє користувачу взаємодіяти з системою. Головна функція аналітичного модуля полягає в виконанні математичних обчислень: для кожного учня розраховується середній бал, який порівнюється з лімітом в 60 балів. Під час реалізації цих розрахунків використовується бібліотека Pandas.

1.2 Аналіз існуючих програмних аналогів

Для визначення вимог до серверної частини системи аналітики успішності студентів було розглянуто існуючі програмні рішення, які використовуються для організації освітнього процесу, збереження навчальних даних та аналізу результатів навчання. Основну увагу приділено не лише наявності функцій електронного журналу чи візуалізації показників, а й технічним особливостям цих систем: способу збереження даних, складності впровадження, можливості інтеграції із зовнішніми сервісами, підтримці баз даних та розширюваності серверної частини.

Для аналізу було обрано три програмні аналоги: Canvas LMS, Moodle та Blackboard Predict. Canvas LMS є хмарною системою управління навчанням, яка забезпечує централізовану роботу з освітнім контентом і даними студентів. Moodle є поширеною LMS-платформою з відкритим кодом, яка може бути розгорнута у різних середовищах і підтримує значну кількість

розширень. Blackboard Predict орієнтована переважно на аналітичну обробку освітніх даних і прогнозування навчальних результатів студентів.

Порівняння цих систем дозволяє визначити їхні переваги та обмеження з точки зору побудови backend-частини системи, яка повинна забезпечувати надійне збереження даних, безпечну авторизацію користувачів, обробку запитів через API та формування аналітичних показників успішності студентів.

1.2.1 Canvas LMS

Canvas LMS - це хмарна система управління навчанням, яка використовується для організації освітнього процесу, розміщення навчальних матеріалів, проведення оцінювання та взаємодії між викладачами і студентами. Система працює за моделлю SaaS, тобто основна серверна інфраструктура підтримується постачальником платформи, а навчальний заклад отримує доступ до готового хмарного сервісу [1].

З точки зору серверної частини Canvas LMS має перевагу у вигляді централізованого збереження даних та готової інфраструктури. Користувачам не потрібно самостійно розгортати сервер, налаштовувати базу даних або підтримувати працездатність системи. Це знижує технічне навантаження на навчальний заклад і дозволяє швидше впровадити платформу в освітній процес.

Canvas LMS підтримує роботу з навчальними курсами, завданнями, оцінками, користувачами та статистичними показниками активності студентів. Дані, які накопичуються в системі, можуть використовуватися для формування звітів та аналізу залученості студентів до навчального процесу. Водночас значна частина внутрішньої логіки обробки даних залишається закритою для користувачів, оскільки система є пропрієтарною.

Недоліком Canvas LMS у контексті розробки власної системи аналітики є обмежена можливість зміни внутрішніх алгоритмів обробки даних. Хоча платформа має інструменти інтеграції із зовнішніми сервісами,

повна адаптація backend-логіки під конкретні потреби навчального закладу може бути складною. Крім того, використання хмарної моделі означає залежність від інфраструктури постачальника та умов ліцензування.

Таким чином, Canvas LMS є зручним готовим рішенням для організації навчального процесу, однак її використання не завжди дозволяє гнучко змінювати серверну логіку, структуру обробки освітніх даних та алгоритми аналітики відповідно до потреб конкретної системи.

1.2.2 Moodle

Moodle - це система управління навчанням з відкритим кодом, яка широко використовується у закладах освіти для організації дистанційного та змішаного навчання. Платформа підтримує створення курсів, розміщення навчальних матеріалів, проведення тестування, ведення журналу оцінок, облік активності студентів та формування звітів [2].

З технічної точки зору Moodle є більш гнучкою системою порівняно з багатьма закритими освітніми платформами, оскільки має відкритий код і підтримує розширення за допомогою плагінів. Це дозволяє навчальним закладам адаптувати систему під власні потреби, додавати нові функції та інтегрувати Moodle з іншими сервісами. Для збереження даних система може використовувати реляційні бази даних, зокрема MySQL, PostgreSQL, MariaDB та інші сумісні СКБД.

Перевагою Moodle є високий рівень розширюваності та наявність великої кількості готових модулів. Система може бути налаштована для різних сценаріїв навчання, а її відкрита архітектура дозволяє створювати додаткові компоненти для обробки освітніх даних. Це робить Moodle потужним інструментом для закладів освіти, які мають достатні технічні ресурси для адміністрування та підтримки платформи.

Водночас Moodle має і суттєві недоліки. Через велику кількість функцій система може бути складною у налаштуванні та супроводі. Для стабільної роботи Moodle потрібно правильно налаштувати серверне

середовище, базу даних, права доступу, резервне копіювання та продуктивність під час високого навантаження. Це підвищує поріг входження для навчальних закладів, які не мають окремої технічної команди.

У контексті backend-розробки Moodle є прикладом функціонально насиченої системи, однак її складність може бути надмірною для задачі вузько спрямованої аналітики успішності студентів. Саме тому для розроблюваної системи доцільним є створення легшої серверної частини, орієнтованої на конкретні задачі: роботу з даними студентів, предметів, оцінок, розрахунок статистичних показників та визначення студентів групи ризику.

1.2.3 Blackboard Predict

Blackboard Predict є спеціалізованим аналітичним рішенням, орієнтованим на прогнозування академічної успішності студентів та виявлення потенційних ризиків у навчальному процесі. На відміну від класичних LMS-платформ, ця система більше зосереджена не на організації навчальних курсів, а на обробці освітніх даних і формуванні аналітичних висновків [3].

Основою роботи Blackboard Predict є використання накопичених навчальних даних, зокрема оцінок, активності студентів, історичних показників успішності та інших параметрів, які можуть впливати на результат навчання. На основі цих даних система визначає студентів, які можуть мати проблеми з успішністю, і надає адміністрації або викладачам інформацію для своєчасного реагування.

З точки зору backend-логіки Blackboard Predict є найближчим аналогом до теми розроблюваної системи, оскільки також працює з аналітикою освітніх даних. Її перевагою є орієнтація на прогнозування та підтримку управлінських рішень. Такий підхід дозволяє використовувати освітні дані не лише для збереження результатів навчання, а й для виявлення закономірностей та потенційних проблем.

Водночас система має обмеження, пов’язані із закритістю платформи, складністю впровадження та залежністю від попередньо підготовлених даних. Для ефективної роботи Blackboard Predict необхідна наявність достатнього обсягу якісних історичних даних, а також інтеграція з іншими інформаційними системами навчального закладу. Крім того, користувачі мають обмежений доступ до внутрішніх механізмів обробки даних і не можуть повністю адаптувати алгоритми під власні потреби.

Таким чином, Blackboard Predict є прикладом спеціалізованої системи освітньої аналітики, однак її використання може бути складним для невеликих навчальних закладів або окремих проєктів через закриту архітектуру, потребу у підготовлених даних та обмежену можливість зміни внутрішньої логіки системи.

1.3 Порівняльний аналіз аналогів

Проведений аналіз показав, що розглянуті програмні рішення мають різне призначення, архітектурні особливості та рівень придатності для задач аналітики успішності студентів. Canvas LMS і Moodle є повноцінними системами управління навчанням, які охоплюють широкий спектр освітніх процесів. Blackboard Predict, на відміну від них, орієнтований переважно на аналітичну обробку даних та прогнозування результатів навчання.

Для порівняння було обрано параметри, які є важливими саме з точки зору розробки серверної частини системи: тип рішення, складність впровадження, гнучкість інтерфейсу, розширюваність та підтримувані бази даних. Ці критерії дозволяють оцінити не лише функціональні можливості аналогів, а й технічні обмеження, які можуть впливати на подальшу обробку освітніх даних, інтеграцію з іншими системами та масштабування.

Порівняльний аналіз функціональних та технічних характеристик існуючих програмних аналогів наведено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих програмних аналогів

Критерії порівняння	**Canvas LMS**	**Moodle**	**Blackboard Predict**
Тип рішення	Хмарна LMS	Гібридна LMS	Система передбачення успішності
Складність впровадження	Середня (час на інтеграцію та навчання)	Висока (повне самостійне налаштування)	Висока (попередня робота вхідними даними)
Гнучкість UI	Висока (редактор тем)	Низька (застарілий інтерфейс)	Низька (лише налаштування фільтрів)
Розширюваність	Середня (зовнішні сервіси)	Висока (відкритий код)	Відсутня
Підтримувані БД	PostgreSQL	PostgreSQL, MySQL, Oracle	Інтегрована

За результатами порівняння можна зробити висновок, що кожна з розглянутих систем має як переваги, так і обмеження. Canvas LMS є зручним хмарним рішенням, однак його можливості зміни внутрішньої логіки обробки даних обмежені. Moodle має високий рівень розширюваності завдяки відкритому коду, проте потребує складнішого розгортання та адміністрування. Blackboard Predict орієнтований на аналітику та прогнозування, але є закритою системою і потребує попередньо підготовлених даних для ефективної роботи.

Отримані результати підтверджують доцільність розробки окремої серверної частини системи аналітики успішності студентів. Така система має бути простішою у впровадженні, ніж великі LMS-платформи, але водночас повинна забезпечувати централізоване збереження освітніх даних, обробку запитів через API, розмежування прав доступу, агрегацію оцінок, визначення студентів групи ризику та формування аналітичних звітів.

Таким чином, проведений аналіз підтвердив доцільність розробки окремої системи аналітики успішності студентів. На відміну від Canvas LMS і Moodle, які призначені для комплексного управління освітнім процесом, розроблювана система зосереджена на зберіганні та аналізі даних успішності. Це дозволяє реалізувати механізми автоматичного розрахунку середнього бала, виявлення студентів групи ризику, формування аналітичних звітів та надання доступу до даних через REST API, забезпечуючи при цьому меншу складність впровадження та супроводу системи. Саме ці висновки були використані під час формування вимог до програмного забезпечення та подальшого проєктування серверної частини системи.

1.4 Визначення вимог до програмного забезпечення

На основі вивчення особливостей обраної предметної області та порівняння з існуючими аналогами складено повний перелік функціональних та нефункціональних вимог, що стосуються серверної частини системи.

Функціональні вимоги:

1. Реєстрація та авторизація користувачів, включаючи викладачів та адміністраторів, з видачею JWT-токена та забезпеченням безпечного збереження паролів.
2. Виконання повного набору операцій CRUD для роботи зі студентською інформацією.
3. Масове завантаження даних студентів з файлу CSV та підтримка автоматичного створення облікових записів.
4. Виконання операцій CRUD для управління предметами та оцінками.
5. Формування аналітичної панелі із загальними статистичними показниками, зокрема кількістю студентів, середнім балом та відсотком студентів групи ризику.

6. Визначення студентів, що належать до групи ризику, на основі середнього балу нижче 60 балів.

7. Експорт аналітичних даних у формат CSV.

8. Розподіл прав доступу між роллю адміністратора та роллю викладача.

9. Підтримка механізму CORS для взаємодії з клієнтськими застосунками.

Нефункціональні вимоги:

1. Безпека: паролі зберігаються лише у вигляді хешів за алгоритмом bcrypt; авторизація виконується через JWT-токени, які мають обмежений строк дії.

2. Продуктивність: час відповіді на запит API має бути не більше 2 секунд за стандартним навантаженням.

3. Масштабованість: архітектура має забезпечувати можливість горизонтального масштабування зростанням кількості студентів.

4. Контейнеризація: застосунок має бути упакований у Docker-контейнер для спрощення розгортання та перенесення між середовищами.

5. Документація: API має бути повністю задокументований та доступний через інтерфейс Swagger UI.

1.5 Інформаційна модель предметної галузі

У межах предметної галузі можна виділити декілька основних сутностей: користувач, студент, навчальний предмет та оцінка. Кожна із цих сутностей виконує окрему роль у процесі обліку та аналізу навчальних результатів.

Побудова інформаційної моделі дозволяє визначити, які саме дані повинні зберігатися в системі та яким чином вони пов’язані між собою. Для системи аналізу успішності студентів така модель є особливо важливою, оскільки подальша аналітична обробка залежить від коректного зв’язку між

студентами, навчальними дисциплінами та отриманими оцінками. Якщо ці сутності описані неправильно або між ними не визначено чітких зв’язків, система не зможе коректно розраховувати середній бал, формувати статистичні показники та визначати студентів групи ризику. Саме тому інформаційна модель є основою для подальшого проєктування структури бази даних і серверної логіки застосунку.

Інформаційна модель предметної галузі наведена на рисунку 1.1.

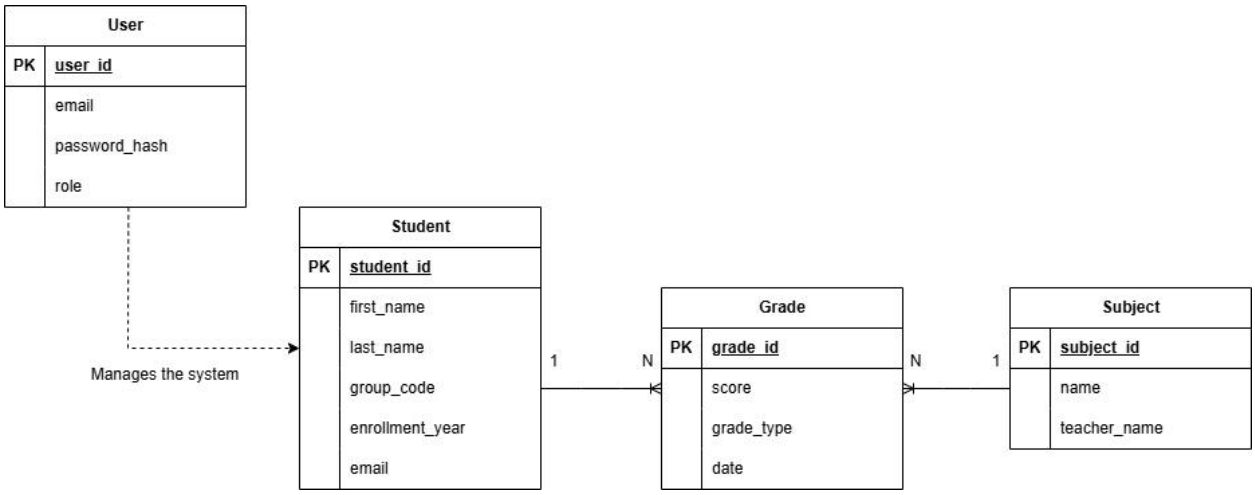


Рис. 1.1 Інформаційна модель предметної галузі системи аналізу успішності студентів

Сутність «Користувач» описує осіб, які взаємодіють із системою та мають доступ до її функціональних можливостей. До користувачів належать викладачі та працівники адміністрації навчального закладу. Для кожного користувача зберігаються облікові дані та інформація про рівень доступу до системи. Наявність ролей дозволяє розмежовувати доступ до функцій керування студентами, навчальними предметами та аналітичними даними.

Сутність «Студент» використовується для зберігання інформації про здобувачів освіти. До основних даних належать ім’я студента, академічна група, рік вступу та інші відомості, необхідні для обліку успішності. Інформація про студентів є основою для подальшого формування статистичних показників та проведення аналітичної обробки освітніх даних.

Сутність «Предмет» описує навчальні дисципліни, за якими здійснюється оцінювання студентів. Для кожного предмета зберігається

інформація про назву дисципліни та викладача, який проводить навчання. Використання окремої сутності для предметів дозволяє впорядкувати освітні дані та забезпечити коректний зв’язок між навчальними дисциплінами та результатами навчання студентів.

Сутність «Оцінка» є ключовим елементом інформаційної моделі предметної галузі, оскільки саме вона відображає результати навчальної діяльності студентів. Оцінка пов’язує конкретного студента з відповідним навчальним предметом та містить інформацію про отриманий бал, тип оцінювання та дату виставлення результату. На основі цих даних виконуються статистичні обчислення, визначається середній бал студентів та формуються аналітичні звіти.

Між основними сутностями предметної галузі існують відповідні зв’язки. Один студент може мати декілька оцінок з різних предметів, а один предмет може бути пов’язаний із результатами багатьох студентів. Така структура дозволяє виконувати подальший аналіз успішності, формувати статистичні показники та визначати студентів, які належать до групи ризику.

2 ЗАСОБИ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

2.1 Середовище розробки Cursor

Cursor є інтегрованим середовищем розробки, побудованим на основі Visual Studio Code. Середовище використовується для створення програмного забезпечення мовами Python, JavaScript, TypeScript та іншими мовами програмування. Однією з особливостей Cursor є вбудована підтримка інструментів штучного інтелекту для роботи з програмним кодом [4].

Середовище підтримує більшість розширень Visual Studio Code, що дозволяє використовувати знайомі інструменти для роботи з проєктом. Під час розробки серверної частини системи використовувались можливості підсвітки синтаксису, автозавершення коду, інтегрований термінал та засоби навігації між файлами проєкту.

Cursor використовувався як основне середовище розробки серверної частини системи аналітики успішності студентів. Інтеграція з інтерпретатором Python дозволяла запускати сервер FastAPI, виконувати команди Docker та працювати з системою контролю версій Git без потреби використання сторонніх програм.

Вбудовані засоби роботи зі штучним інтелектом використовувалися для автоматизації окремих етапів розробки, зокрема під час створення шаблонного коду, генерації структур моделей та перевірки окремих фрагментів програми. Це дозволило пришвидшити процес розробки та зменшити кількість типових помилок під час написання коду.

2.2 Мова програмування Python

Python є високорівневою мовою програмування загального призначення, яка широко використовується для веб-розробки, аналізу даних,

автоматизації процесів та створення серверних застосунків. Однією з основних переваг мови є простий синтаксис, що забезпечує зручність читання та підтримки програмного коду [5].

Мову Python було обрано для реалізації серверної частини системи аналітики успішності студентів через наявність великої кількості бібліотек для роботи з веб-технологіями та обробкою даних. Для реалізації REST API використовувався фреймворк FastAPI, а для аналітичної обробки даних - бібліотека Pandas. Також Python має засоби для взаємодії з нереляційними базами даних MongoDB через бібліотеки PyMongo та MongoEngine.

Важливою перевагою Python є підтримка асинхронного програмування через модуль asyncio. Це дозволяє серверу одночасно обробляти декілька мережевих запитів без блокування виконання програми, що є важливим для веб-застосунків із великою кількістю операцій введення-виведення.

Додатковою перевагою Python є велика кількість готових бібліотек та активна спільнота розробників. Це дозволяє використовувати перевірені рішення для реалізації окремих компонентів системи та скорочує час розробки програмного забезпечення.

Під час розробки застосунку також використовувались засоби типізації Python (Type Hints), які дозволяють зробити код більш зрозумілим та спрощують перевірку правильності типів даних. Це покращує підтримку проєкту та знижує ймовірність виникнення помилок під час роботи системи.

2.3 Веб-фреймворк FastAPI

FastAPI - це веб-фреймворк для створення REST API мовою Python, побудований на основі стандартів OpenAPI та JSON Schema [6 с. 27]. Фреймворк використовується для розробки серверних застосунків та забезпечує підтримку асинхронної обробки HTTP-запитів.

Однією з основних особливостей FastAPI є використання системи типізації Python для автоматичної перевірки вхідних даних. Для цього

застосовується бібліотека Pydantic, яка дозволяє описувати структуру запитів та відповідей у вигляді окремих моделей. Такий підхід спрощує перевірку коректності даних і зменшує кількість помилок під час взаємодії клієнта із сервером [7].

FastAPI автоматично генерує документацію API у форматах Swagger UI та ReDoc. Це дозволяє тестувати HTTP-запити без використання сторонніх програм та спрощує процес взаємодії між серверною і клієнтською частинами застосунку.

Фреймворк побудований на основі Starlette та підтримує асинхронне програмування через async/await. Завдяки цьому сервер може одночасно обробляти декілька запитів, що є важливим для веб-застосунків, які активно взаємодіють із базою даних або зовнішніми сервісами.

FastAPI було обрано для реалізації серверної частини системи аналітики успішності студентів через підтримку асинхронної архітектури, автоматичну генерацію документації API та зручну інтеграцію із засобами типізації Python. У межах проєкту фреймворк використовувався для реалізації REST API, маршрутизації HTTP-запитів, авторизації користувачів та перевірки вхідних даних.

Додатково у FastAPI використовувалась система ін’єкції залежностей, яка дозволяє централізовано реалізувати механізми перевірки JWT-токенів та розмежування прав доступу між адміністраторами і викладачами. Також у серверній частині було налаштовано підтримку CORS для забезпечення взаємодії між backend-частиною та клієнтським застосунком [8].

2.4 Система управління базами даних MongoDB Atlas

MongoDB є нереляційною системою управління базами даних, яка зберігає інформацію у вигляді документів формату BSON - бінарного представлення JSON [9]. На відміну від реляційних баз даних, MongoDB не

потребує жорстко визначеної структури таблиць, що забезпечує більшу гнучкість під час роботи з різними типами даних.

Для серверної частини системи аналітики успішності студентів було обрано MongoDB Atlas - хмарний сервіс для розгортання та адміністрування баз даних MongoDB [10]. Використання хмарної платформи дозволило уникнути необхідності самостійного налаштування серверної інфраструктури та забезпечило централізоване збереження даних.

MongoDB Atlas автоматично виконує резервне копіювання даних, моніторинг роботи системи та підтримує захищене підключення через TLS. Це дозволяє забезпечити безпечну передачу даних між серверною частиною застосунку та базою даних.

MongoDB було обрано через гнучкість структури документів та зручність роботи з JSON-подібними даними. Такий підхід добре підходить для веб-застосунків, у яких структура даних може змінюватися або розширюватися під час розвитку системи.

Для взаємодії з базою даних у проєкті використовувалась бібліотека MongoEngine, яка реалізує підхід Object Document Mapper (ODM) [11]. MongoEngine дозволяє працювати з документами MongoDB через класи Python, що спрощує створення моделей даних та перевірку їх коректності.

У межах системи було створено окремі колекції для користувачів, студентів, предметів та оцінок. Зв’язки між документами реалізовані за допомогою ReferenceField, що дозволяє пов’язувати оцінки з відповідними студентами та навчальними предметами.

2.5 Бібліотека аналітики Pandas

Pandas - це бібліотека Python для обробки та аналізу табличних даних [12]. Вона надає інструменти для роботи зі структурованою інформацією та широко використовується у задачах статистичного аналізу й обробки даних.

Основними структурами даних у Pandas є Series та DataFrame. Series використовується для збереження одновимірних даних, а DataFrame - для представлення інформації у вигляді таблиць із рядками та стовпцями. Саме DataFrame є основним інструментом для виконання аналітичних операцій.

У системі аналітики успішності студентів бібліотека Pandas використовується для обробки результатів навчання та формування статистичних показників. Дані про оцінки студентів завантажуються з MongoDB та перетворюються у структуру DataFrame для подальшого аналізу.

За допомогою методів groupby та mean виконується обчислення середнього балу студентів. Після цього система визначає студентів, середній бал яких є нижчим за встановлений поріг. На основі отриманих результатів формується список студентів групи ризику.

Додатково бібліотека Pandas використовується для формування звітів у форматі CSV. Метод to_csv() дозволяє експортувати структуровані дані у файл, який може бути відкритий у Microsoft Excel або інших табличних редакторах.

2.6 Технологія контейнеризації Docker

Docker є платформою для контейнеризації програмного забезпечення, яка дозволяє запускати застосунки разом із необхідними залежностями у відокремленому середовищі [13]. Використання контейнерів забезпечує однакову роботу програми на різних операційних системах та спрощує процес розгортання застосунків.

Для серверної частини системи аналітики успішності Docker використовувався для створення контейнера із FastAPI-застосунком та необхідними бібліотеками Python. Усі залежності проєкту були описані у файлі requirements.txt та автоматично встановлювались під час побудови Docker-образу.

Файл Dockerfile містив інструкції для створення контейнера, зокрема вибір базового Python-образу, копіювання файлів проєкту та запуск серверної частини застосунку. Такий підхід дозволив забезпечити стабільну роботу системи незалежно від локального середовища розробки.

Для локального тестування застосунку використовувався Docker Compose. Це дозволило одночасно запускати сервер FastAPI та базу даних MongoDB за допомогою одного конфігураційного файлу docker-compose.yml. Використання Docker Compose спростило процес налаштування середовища розробки та запуску системи.

2.7 Хмарна платформа Render

Render є хмарною платформою для розгортання веб-застосунків, серверів та інших мережевих сервісів [14]. Платформа підтримує автоматичне розгортання застосунків із GitHub-репозиторіїв та роботу з Docker-контейнерами.

Для розміщення серверної частини системи аналітики успішності студентів було обрано платформу Render через простоту налаштування та наявність безкоштовного тарифного плану. Це дозволило виконати розгортання застосунку без необхідності самостійного адміністрування серверної інфраструктури.

Render автоматично підтримує HTTPS-з’єднання, що забезпечує захищену передачу даних між клієнтською та серверною частинами системи. Також платформа дозволяє налаштовувати змінні середовища для безпечного збереження конфігураційних параметрів, зокрема рядка підключення до MongoDB Atlas та секретного ключа JWT.

Інтеграція Render із GitHub дозволила автоматизувати процес оновлення серверної частини системи. Після внесення змін до репозиторію платформа автоматично створювала нову версію Docker-контейнера та виконувала повторне розгортання застосунку.

2.8 Система контролю версій Git та GitHub

Git є розподіленою системою контролю версій, яка використовується для відстеження змін у програмному коді та організації спільної роботи над проєктами [15]. GitHub є хмарною платформою для збереження Git-репозиторіїв та управління процесом розробки програмного забезпечення.

Під час створення серверної частини системи Git використовувався для контролю змін у вихідному коді проєкту. Кожна нова функція або виправлення помилок фіксувалися окремими комітами, що дозволяло відстежувати історію розробки та за потреби повертатися до попередніх версій коду.

GitHub використовувався як основне сховище репозиторію проєкту та забезпечував централізоване збереження вихідного коду. Використання хмарного репозиторію також дозволило інтегрувати систему контролю версій із платформою Render для автоматичного розгортання серверної частини застосунку.

Використання Git та GitHub дозволило спростити процес розробки, тестування та оновлення системи, а також забезпечило надійне збереження змін у проєкті.

Отже, вибір технологічного стеку серверної частини системи є обґрунтованим з урахуванням поставлених задач. Для реалізації проєкту використано середовище розробки Cursor, мову програмування Python, веб-фреймворк FastAPI, систему управління базами даних MongoDB Atlas, бібліотеку Pandas для аналітичної обробки даних, технологію контейнеризації Docker, хмарну платформу Render, а також Git і GitHub для контролю версій. Такий набір технологій забезпечує стабільну роботу backend-частини, обробку освітніх даних, виконання аналітичних розрахунків і можливість розгортання застосунку в хмарному середовищі.

3 ПРОЄКТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

У цьому розділі розглянуто проєктування серверної частини системи аналітики успішності студентів. Основну увагу приділено архітектурі застосунку, структурі бази даних, логіці взаємодії між програмними компонентами, модулю авторизації та розподілу прав доступу між користувачами.

Особливістю розроблюваної системи є поєднання класичного обліку навчальних даних із модулем аналітичної обробки, який дозволяє не лише зберігати інформацію про студентів, предмети та оцінки, а й виконувати агрегацію показників успішності, розраховувати середній бал та визначати студентів групи ризику. Саме така інтеграція облікової та аналітичної функціональності якісно відрізняє розроблюваний застосунок від більшості розглянутих аналогів, які переважно орієнтовані на ведення журналів або формування базових звітів.

3.1 Архітектура застосунку

Система аналізу успішності студентів спроєктована на основі клієнт-серверної архітектури з поділом на три основні рівні: клієнтський рівень, серверний рівень та рівень збереження даних. Такий підхід дозволяє розділити відповідальність між компонентами системи, спростити підтримку застосунку та забезпечити можливість подальшого розширення функціональності.

На рисунку 3.1 зображено загальну архітектуру застосунку, яка демонструє взаємодію між клієнтською частиною, серверною частиною, базою даних та зовнішніми сервісами розгортання.

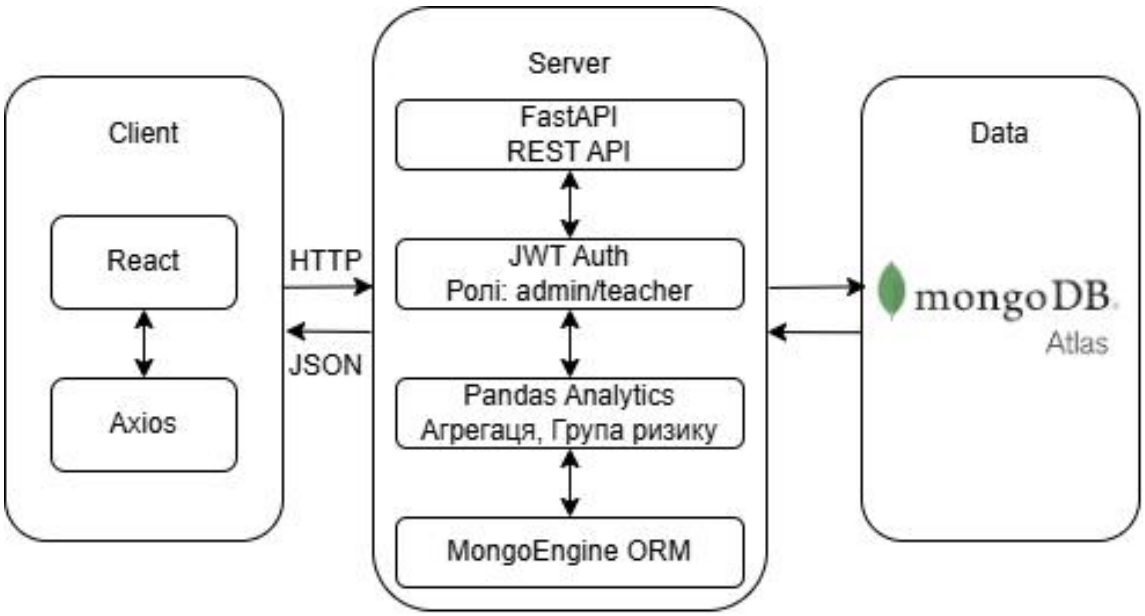


Рис. 3.1 Архітектура застосунку

Клієнтський рівень реалізовано як вебзастосунок, що взаємодіє із сервером через HTTP-запити до REST API. Користувач працює з інтерфейсом системи, надсилає запити на отримання або зміну даних, а також переглядає результати аналітичної обробки у вигляді таблиць і статистичних показників. Для захищених запитів використовується JWT-токен, який передається у заголовку Authorization.

Серверний рівень реалізовано на основі FastAPI. Він відповідає за обробку HTTP-запитів, перевірку вхідних даних, авторизацію користувачів, взаємодію з базою даних та виконання аналітичних операцій. У межах серверної частини виділено декілька логічних модулів: модуль роботи зі студентами, модуль роботи з предметами, модуль роботи з оцінками, модуль авторизації та аналітичний модуль.

Окрему роль у серверній архітектурі виконує аналітичний модуль. Його призначення полягає в обробці навчальних даних, розрахунку середнього балу студентів, формуванні загальних статистичних показників та визначенні студентів, які можуть належати до групи ризику. Саме наявність такого модуля є однією з ключових відмінностей системи від звичайних електронних журналів, де дані переважно лише зберігаються та відображаються без додаткової обробки.

Рівень збереження даних реалізовано з використанням MongoDB Atlas. У базі даних зберігаються відомості про користувачів, студентів, навчальні предмети та оцінки. Для взаємодії серверної частини з базою даних використовується MongoEngine ODM, що дозволяє описувати структуру документів у вигляді Python-класів та виконувати операції з даними на рівні моделей.

3.2 Проєктування структури бази даних

Для збереження та обробки інформації у системі SDF використовується документоорієнтована база даних MongoDB. Вибір такої моделі зберігання обумовлений тим, що освітні дані можуть мати різну структуру та в майбутньому розширюватися новими полями без необхідності складної зміни всієї схеми бази даних.

База даних складається з чотирьох основних колекцій: User, Student, Subject та Grade. Така структура відповідає інформаційній моделі предметної галузі та забезпечує збереження даних, необхідних для обліку й аналізу успішності студентів.

Колекція User призначена для зберігання облікових записів користувачів системи. У ній містяться дані, необхідні для авторизації та розмежування прав доступу. Поле email використовується як унікальний ідентифікатор користувача під час входу до системи. Поле hashed_password зберігає пароль у хешованому вигляді, що дозволяє не зберігати відкриті паролі в базі даних. Поле role визначає роль користувача та використовується для обмеження доступу до окремих функцій системи.

Колекція Student використовується для зберігання інформації про студентів. Вона містить персональні та навчальні дані, зокрема ім’я, прізвище, код академічної групи, рік вступу, електронну адресу та кількість пропусків. Наявність окремої колекції студентів дозволяє централізовано

зберігати навчальну інформацію та використовувати її для подальшої аналітичної обробки.

Колекція Subject призначена для зберігання інформації про навчальні дисципліни. Для кожного предмета зберігається назва дисципліни та ім’я викладача. Винесення предметів в окрему колекцію дозволяє уникнути дублювання даних та забезпечує зручний зв’язок між навчальними дисциплінами й оцінками студентів.

Колекція Grade є основною для аналітичної обробки даних, оскільки саме вона містить результати навчальної діяльності студентів. У цій колекції зберігається посилання на студента, посилання на навчальний предмет, числове значення оцінки, тип оцінювання та дата створення запису. Завдяки зв’язкам Grade → Student та Grade → Subject система може аналізувати успішність як окремого студента, так і групи студентів у межах конкретних навчальних дисциплін.

Запропонована структура бази даних орієнтована не лише на збереження даних, а й на їх подальшу агрегацію. Це важливо для реалізації аналітичного модуля, який використовує оцінки студентів для обчислення середнього балу, формування статистики та визначення групи ризику.

На рисунку 3.2 графічно зображено структуру бази даних.

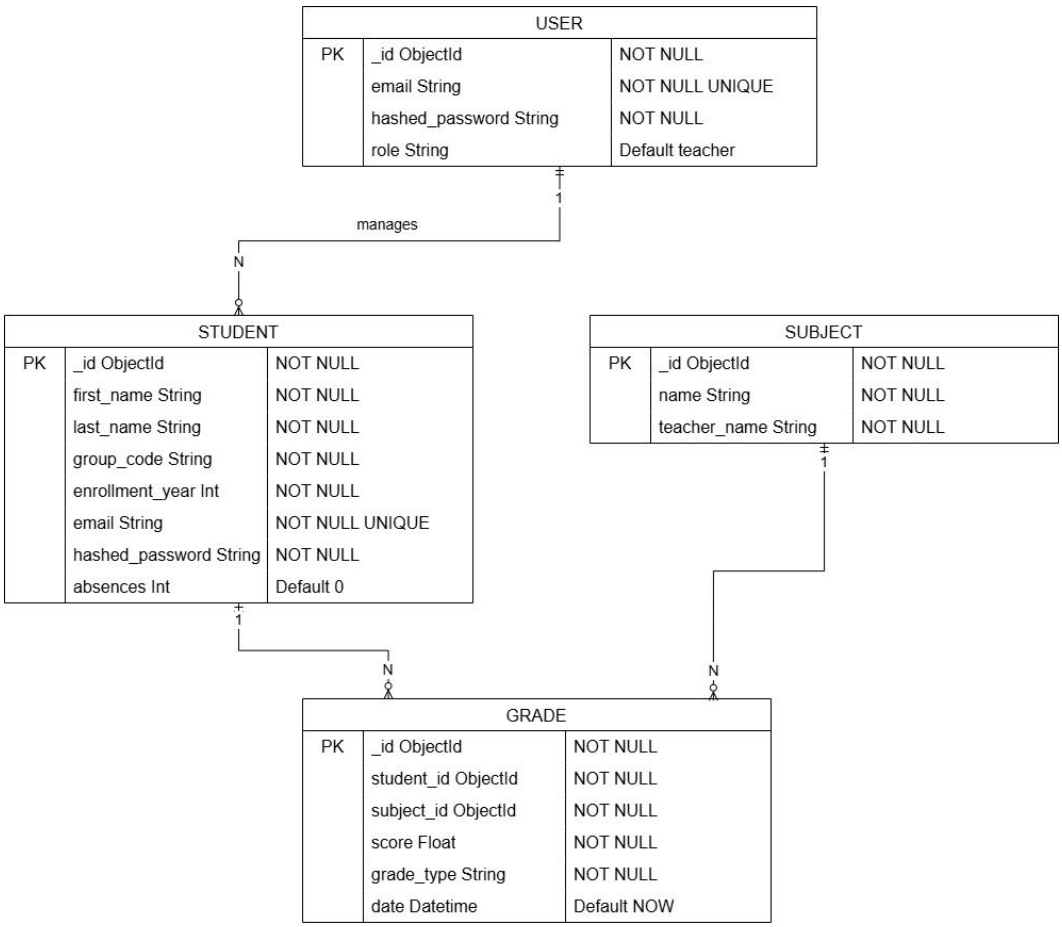


Рис. 3.2 Структура бази даних

3.3 Діаграма компонентів серверної логіки

Діаграма класів основної логіки відображає структуру програмних компонентів серверної частини та взаємозв’язки між ними.

У структурі серверної частини можна виділити три основні групи компонентів: моделі MongoEngine, схеми Pydantic та маршрутизатори API. Моделі MongoEngine відповідають за опис документів бази даних і використовуються для роботи з колекціями MongoDB. До них належать Student, Subject та Grade. Клас Student описує інформацію про студента, клас Subject - навчальну дисципліну, а клас Grade - оцінку студента з конкретного предмета.

Клас Grade є одним із ключових елементів основної логіки системи, оскільки саме через нього реалізується зв’язок між студентом, предметом та

результатом навчання. Надалі ці дані використовуються аналітичним модулем для розрахунку середнього балу та визначення студентів із низькими показниками успішності.

Pydantic-схеми використовуються для опису структури вхідних і вихідних даних API. Схеми типу Create визначають дані, які надходять від клієнта під час створення нового запису. Схеми типу Response описують формат відповіді сервера. Окремі схеми Update використовуються для часткового оновлення даних, коли користувач змінює лише окремі поля запису.

Маршрутизатори API на діаграмі подано як логічні групи ендпоінтів, що відповідають за окремі частини серверної логіки. У межах реалізації ці ендпоінти розміщені у файлі main.py та згруповані за призначенням: робота зі студентами, предметами, оцінками та аналітичними даними. Такий спосіб подання на діаграмі дозволяє наочно показати функціональне розділення серверної частини, навіть якщо фізично маршрути реалізовані в одному файлі.

StudentRouter на діаграмі відображає групу маршрутів для роботи зі студентами. До неї належать створення студента, отримання списку студентів, редагування, видалення та масове завантаження даних із CSV-файлу. SubjectRouter відображає групу маршрутів для створення та перегляду навчальних предметів. GradeRouter відповідає за додавання та отримання оцінок студентів. AnalyticsRouter відображає групу аналітичних маршрутів, які забезпечують формування статистичних показників, визначення студентів групи ризику та експорт аналітичних даних у CSV.

Наявність окремої логічної групи аналітичних маршрутів підкреслює прикладну новизну системи, оскільки застосунок не обмежується стандартними CRUD-операціями, а містить окремий рівень аналітичної обробки освітніх даних.

На рисунку 3.3 зображено основні класи, які використовуються для роботи з даними, перевірки вхідної інформації та обробки HTTP-запитів.

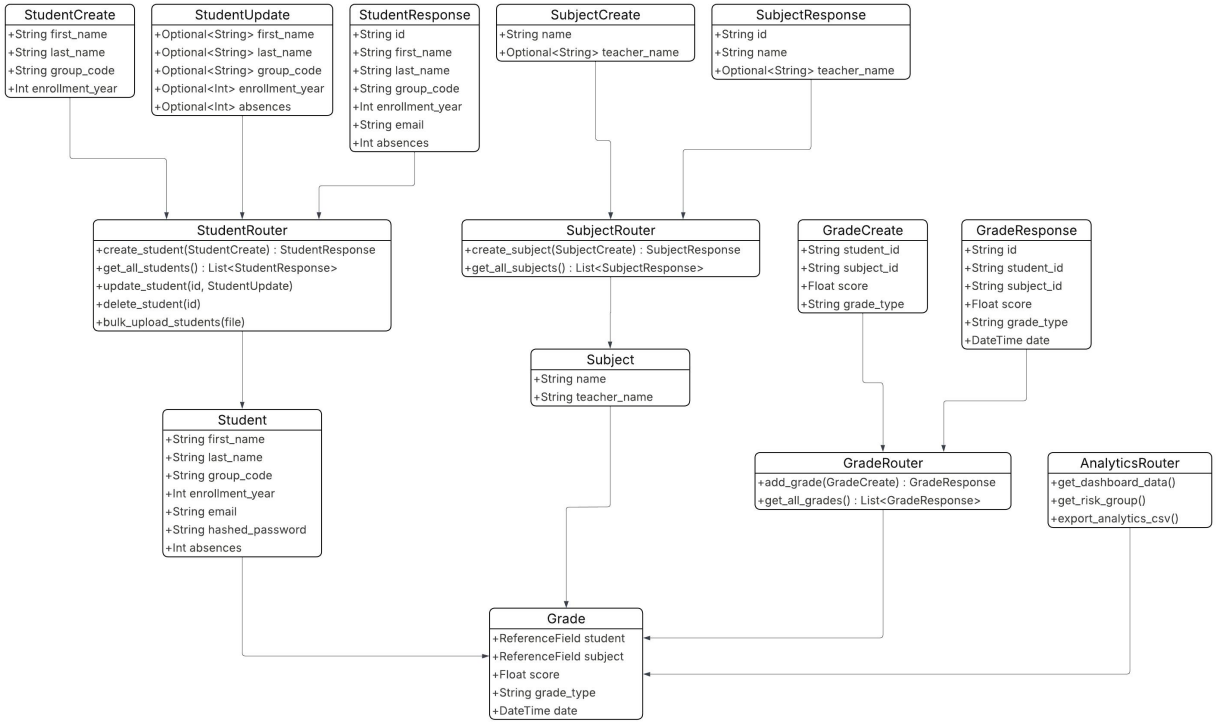


Рис. 3.3 Діаграма компонентів серверної логіки

3.4 Діаграма класів модуля авторизації

Модуль авторизації відповідає за реєстрацію користувачів, перевірку облікових даних, створення JWT-токенів та контроль доступу до захищених маршрутів. На рисунку 3.4 наведено діаграму класів модуля авторизації, яка демонструє основні компоненти цього модуля та зв’язки між ними.

Клас User є моделлю MongoEngine і використовується для зберігання облікових записів користувачів. Основними полями цієї моделі є email, hashed_password та role. Поле role дозволяє визначити рівень доступу користувача до функцій системи, що є основою для реалізації рольової моделі доступу.

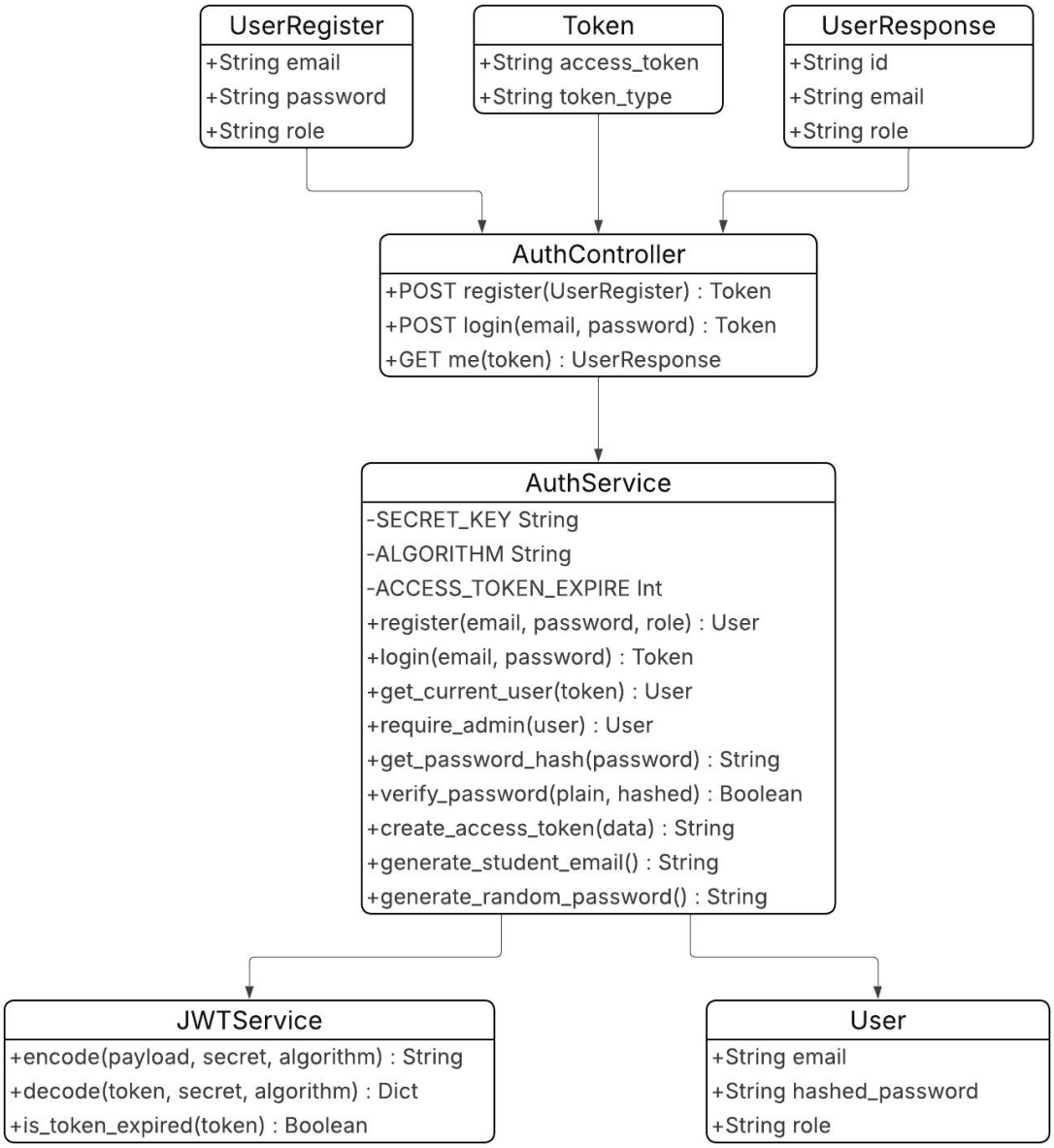


Рис. 3.4 Діаграма класів авторизації

Pydantic-схеми UserRegister, Token та UserResponse описують структуру даних, які використовуються під час взаємодії з API. Схема UserRegister визначає дані, які користувач передає під час реєстрації. Схема Token описує відповідь сервера після успішної авторизації. Схема UserResponse використовується для повернення інформації про поточного користувача без розкриття службових або конфіденційних даних.

AuthService містить набір функцій, які реалізують основну логіку авторизації. Функції register() та login() використовуються для створення нового користувача та входу до системи. Функція get_current_user()

реалізована як залежність FastAPI та виконує перевірку JWT-токена під час звернення до захищених маршрутів. Функція require_admin() перевіряє роль користувача та обмежує доступ до адміністративних функцій.

Для реалізації токенів використовується підхід JWT, який дозволяє передавати інформацію про користувача у вигляді підписаного токена [16]. У системі SDF токен використовується для підтвердження особи користувача під час кожного захищеного запиту. Такий підхід відповідає принципам побудови REST API та не потребує збереження стану сесії на сервері. Загальні принципи побудови OAuth2- та JWT-авторизації розглядаються в джерелі [17, с. 21].

AuthController реалізований як FastAPI-маршрутизатор і містить основні ендпоінти для роботи з авторизацією: реєстрацію користувача, вхід до системи та отримання інформації про поточного користувача. Така структура дозволяє відокремити логіку авторизації від інших модулів системи та спростити її подальшу підтримку.

Окрім структури класів, важливим елементом модуля авторизації є послідовність перевірки користувача під час входу до системи. У системі SDF використовується JWT-авторизація, яка забезпечує перевірку користувача та розмежування доступу до захищених ресурсів. Під час входу користувач передає електронну адресу та пароль, після чого сервер виконує пошук облікового запису в базі даних і перевіряє відповідність введеного пароля його хешованому значенню. У разі успішної перевірки backend формує JWT-токен, який клієнтська частина використовує під час подальших запитів до системи. Токен передається у заголовку Authorization, а сервер перевіряє його справжність і визначає роль користувача. Такий підхід дозволяє реалізувати захищений доступ до операцій зі студентами, предметами, оцінками та аналітичними даними без зберігання активних сесій на сервері.

3.5 Карта веб-застосунку та розподіл прав доступу

Карта веб-застосунку відображає структуру основних сторінок системи та доступність функцій для різних категорій користувачів. На рисунку 3.5 наведено карту вебзастосунку, яка демонструє розподіл доступу між неавторизованим користувачем, викладачем та адміністратором.

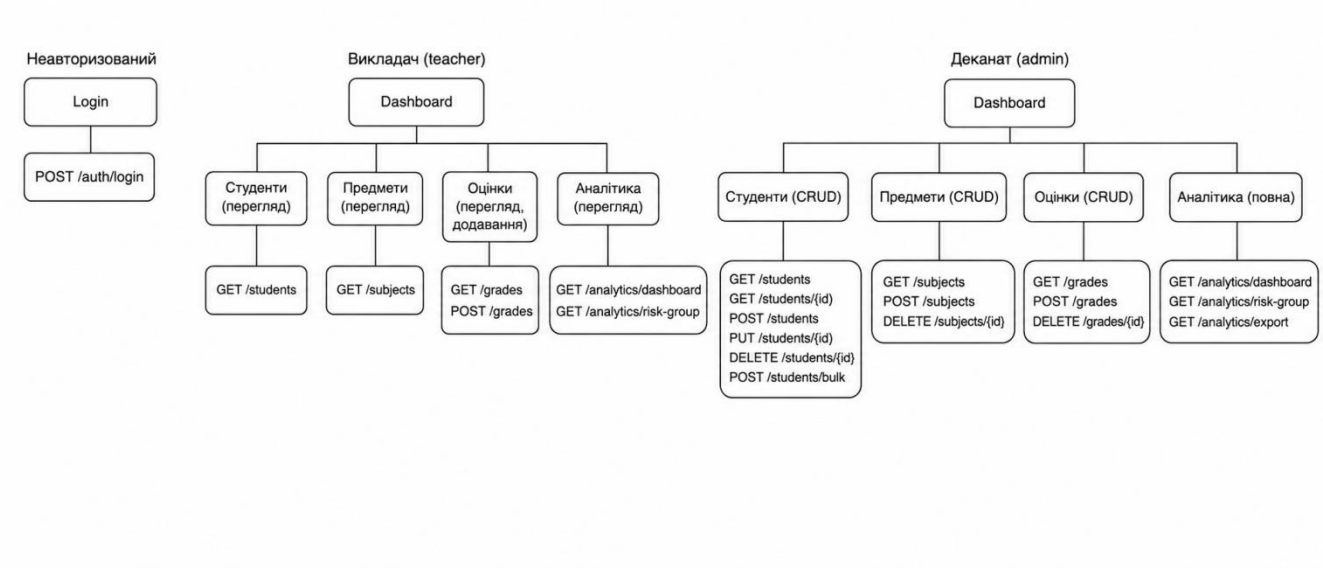


Рис. 3.5 Мапи веб-застосунку

Неавторизований користувач має доступ лише до сторінки входу. Після успішної авторизації система визначає роль користувача та відкриває доступ до відповідних функціональних можливостей.

Викладач має доступ до основних інформаційних розділів системи, зокрема до перегляду списку студентів, предметів, оцінок та аналітичних показників. Також викладач може працювати з оцінками студентів у межах доступного йому функціоналу. Такий рівень доступу орієнтований на щоденну роботу з навчальними результатами.

Адміністратор має розширені права доступу. Йому доступні операції створення, редагування та видалення студентів, предметів і оцінок, масове завантаження студентів із CSV-файлу, перегляд аналітичних даних та експорт звітів. Такий розподіл прав дозволяє відокремити адміністративні дії

від функцій викладача та зменшити ризик випадкової або несанкціонованої зміни критичних даних.

Розподіл прав доступу є важливим елементом проєктування системи, оскільки освітні дані мають різний рівень доступності для різних користувачів. У розроблюваній системі права доступу пов’язані не лише з переглядом інформації, а й з можливістю виконання аналітичних та адміністративних операцій.

3.6 Діаграма варіантів використання

Діаграма варіантів використання дозволяє формалізувати основні сценарії взаємодії користувачів із системою. На рисунку 3.6 наведено діаграму варіантів використання, яка показує функціональні можливості системи для основних ролей: викладача та адміністратора.

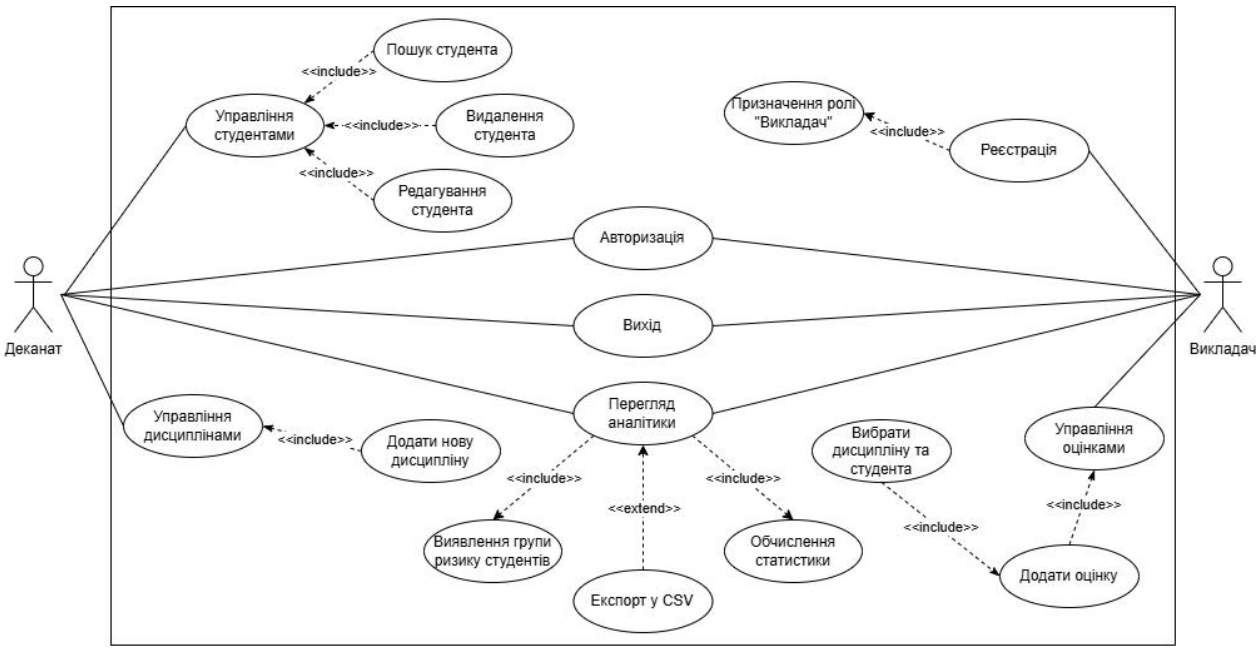


Рис. 3.6 Діаграма варіантів використання

Адміністратор виконує функції керування основними даними системи. До його можливостей належать авторизація, керування студентами, керування навчальними предметами, робота з оцінками, масове завантаження студентів, перегляд аналітичної інформації та експорт звітів. Адміністратор

також має доступ до функцій, які впливають на структуру даних системи, тому його роль передбачає найвищий рівень доступу.

Викладач має обмежений набір функцій, пов’язаний переважно з переглядом навчальних даних та роботою з оцінками. Він може авторизуватися в системі, переглядати інформацію про студентів і предмети, вносити або переглядати оцінки, а також отримувати аналітичні дані щодо успішності студентів.

Окрему увагу в діаграмі приділено аналітичним сценаріям використання. До них належать перегляд загальної статистики, визначення студентів групи ризику та формування звітів. Наявність таких сценаріїв демонструє, що система призначена не лише для обліку даних, а й для підтримки прийняття управлінських рішень на основі обробленої інформації.

3.7 Діаграма послідовностей застосунку

Діаграма послідовностей описує порядок взаємодії між користувачем, клієнтською частиною, сервером, базою даних та аналітичним модулем під час отримання статистичних даних. На рисунку 3.7 зображено послідовність дій, які виконуються під час запиту аналітичної інформації про успішність студентів.

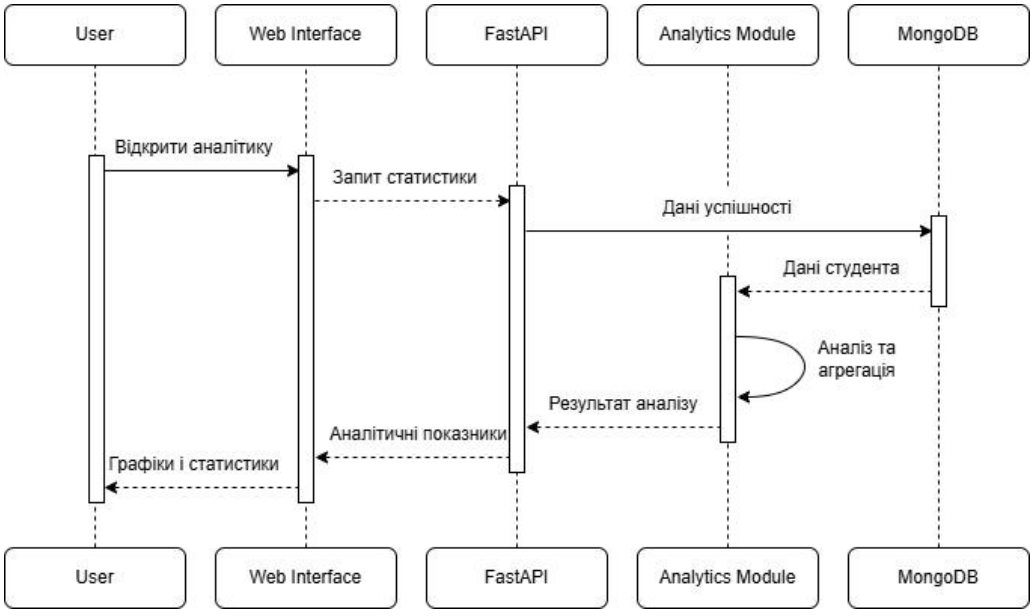


Рис. 3.7 Діаграма послідовностей застосунку

Процес починається з авторизації користувача у вебінтерфейсі. Після введення облікових даних клієнтська частина надсилає запит до серверної частини. Сервер перевіряє введені дані, звертається до бази даних та в разі успішної перевірки повертає користувачу JWT-токен.

Після авторизації користувач переходить до аналітичного розділу та надсилає запит на отримання статистичних показників. Сервер перевіряє токен доступу, визначає роль користувача та виконує запит до бази даних для отримання необхідної інформації про студентів, предмети й оцінки.

Отримані дані передаються до аналітичного модуля, де виконується їх обробка. На цьому етапі здійснюється групування оцінок за студентами, розрахунок середнього балу, визначення студентів із низькими показниками успішності та формування узагальнених статистичних даних.

Після завершення обробки сервер повертає результати клієнтській частині. Користувач бачить сформовані показники у вигляді таблиць, числових значень або графічних елементів інтерфейсу. Такий сценарій демонструє головну відмінність системи: користувач отримує не лише сирі дані з бази, а вже оброблену аналітичну інформацію, придатну для прийняття рішень.

Отже, у процесі проєктування серверної частини було визначено архітектуру системи, структуру бази даних, основні програмні модулі, механізм авторизації та взаємодію між клієнтською і серверною частинами. Запропонована структура дозволяє організувати збереження освітніх даних, виконувати їх аналітичну обробку та забезпечувати розмежування доступу між користувачами системи.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

4.1 Структура проєкту

Бекенд реалізовано як модульний проєкт на Python, що спрощує розробку, подальший супровід та розширення функціоналу системи. Точкою входу до програми є файл main.py. У ньому ініціалізується екземпляр FastAPI, налаштовується механізм CORS для коректної взаємодії з фронтенд-частиною, а також визначаються основні API-ендпоінти для роботи з даними про здобувачів освіти, навчальні дисципліни, результати навчання та аналітичні звіти.

Структура даних системи за допомогою бібліотеки MongoEngine визначена у файлі models.py. Цей документ встановлює схеми документів MongoDB, конкретизуючи моделі студентів, дисциплін, оцінок та облікових записів користувачів.

Файл schemas.py призначений для визначення валідаційних схем, що базуються на Pydantic. Ці схеми необхідні для верифікації коректності вхідних даних HTTP-запитів та форматування вихідних даних API.

Реалізація механізмів підтвердження особи та надання прав доступу користувачам закріплена за модулем auth.py. Тут реалізовано процес реєстрації, входу, створення JWT-токенів, перевірка повноважень та безпечне зберігання паролів через хешування.

Для ініціалізації або оновлення облікового запису адміністратора задіяний файл create_admin.py, тоді як seed_db.py служить для автоматичного наповнення сховища даних первинними (тестовими) записами, включаючи інформацію про студентів, курси та їх оцінки.

Список усіх бібліотек проекту зберігається у файлі requirements.txt. Процес контейнеризації забезпечується наявністю Dockerfile і docker-

compose.yml, що дозволяє уніфіковано та оперативно розгортати бекенд разом із базою у стандартизованому середовищі.

З'єднання з MongoDB встановлюється за викликом функції `mongoengine.connect()`. Архітектура має на увазі хмарне підключення до бази даних, що реалізується через передачу рядка URI, забезпечуючи єдину точку доступу до даних для серверної логіки.

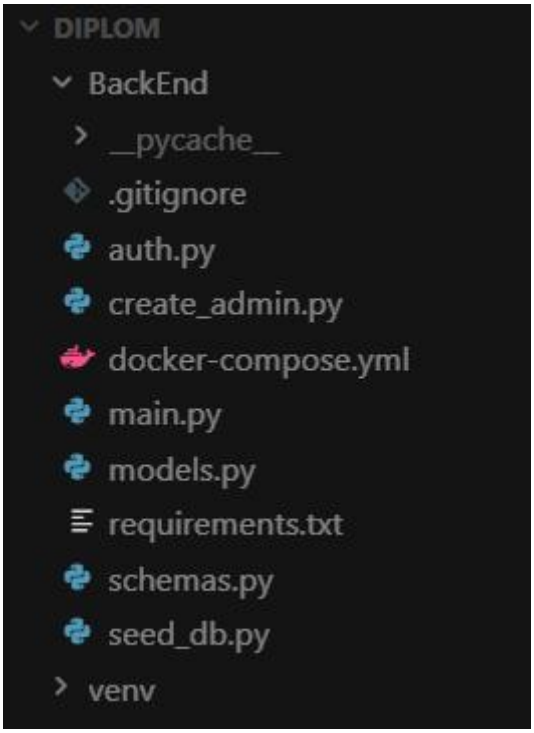


Рис. 4.1 Структура проєкту

4.2 Реалізація моделей даних

Схеми даних задаються у файлі models.py за допомогою ODM MongoEngine. Використання об'єктно-орієнтованого підходу значно спрощує роботу з базою даних MongoDB, оскільки кожна колекція описується як окремий клас на Python. Кожна така сутність реалізована як клас, що успадковує Document та визначає параметри відповідної колекції: типи полів, їх обов'язковість, дозволені значення та обмеження перевірки. Це забезпечує цілісність даних, мінімізуючи ризик помилок під час роботи системи.

Модель User використовується для облікових записів вчителя та адміністратора. Вона агрегує ключову інформацію, необхідну для аутентифікації користувачів у системі. Поле email є обов'язковим і має бути унікальним, що не допускає реєстрацію двох користувачів з однаковими адресами електронної пошти. Це дозволяє використовувати електронну пошту як єдиний ідентифікатор під час входу. Поле hashed_password зберігає лише хеш пароля, згенерований алгоритмом bcrypt, що гарантує, що пароль ніколи не зберігатиметься в базі даних у незашифрованому вигляді. Атрибут role вказує на рівень доступу користувача до функціональності системи та може приймати значення "admin" або "teacher", причому останнє встановлено за замовчуванням.

```
class User(Document):
    email = StringField(required=True, unique=True)
    hashed_password = StringField(required=True)
    role = StringField(choices=('admin', 'teacher'), default='teacher')
```

Рис. 4.2 Модель User

Модель Student призначена для запису особистої та освітньої інформації студента. Вона містить основну інформацію про предмет освітньої системи, зокрема, ім'я, прізвище, освітню групу та рік початку навчання. Поля first_name, last_name та group_code є обов'язковими текстовими полями, оскільки без них правильна ідентифікація студента в системі неможлива. Поле enrollment_year фіксує рік зарахування та сприяє подальшому аналізу академічних даних за курсами. Важливою особливістю моделі є те, що поля email та hashed_password генеруються автоматично, коли адміністратор створює обліковий запис студента. Пряма реєстрація студентів у поточній версії не передбачена. Поле absences використовується для запису кількості пропущених занять і за замовчуванням ініціалізується нулем.

```
class Student(Document):
    first_name = StringField(required=True)
    last_name = StringField(required=True)
    group_code = StringField(required=True)
    enrollment_year = IntField(default=2024)
    email = StringField(unique=True)
    hashed_password = StringField()
    role = StringField(default="student")
    absences = IntField(default=0)
```

Рис. 4.3 Модель Student

Модель Subject використовується для опису курсів, що пропонуються в рамках освітнього процесу. Вона містить обов'язкове поле name, яке зберігає назву дисципліни, а також необов'язкове поле teacher_name, яке містить ім'я викладача, відповідального за цей курс. Така структура дозволяє пов'язувати академічні предмети з оцінками, отриманими студентами, та надалі використовувати ці дані для аналізу успішності з окремих дисциплін.

```
class Subject(Document):
    name = StringField(required=True, unique=True)
    teacher_name = StringField()
```

Рис. 4.4 Модель Subject

Модель оцінювання є ключовим компонентом аналітичної частини системи, оскільки саме вона накопичує дані про академічні досягнення студентів. Модель використовує поля student та subject, реалізовані через ReferenceField, що дозволяє формувати зв'язки між записами в різних колекціях. У цьому випадку в базі даних зберігається лише ObjectId відповідних документів.

Модель Grade містить кількісне значення оцінки студента, а поле grade_type визначає тип контрольного заходу, наприклад, іспит, практична робота чи домашнє завдання. Поле date автоматично фіксує момент виставлення оцінки, що дозволяє відстежувати динаміку успішності студента з плином часу та виконувати подальший статистичний аналіз зібраних даних.

```
class Grade(Document):
    student = ReferenceField(Student, reverse_delete_rule=CASCADE, required=True)
    subject = ReferenceField(Subject, reverse_delete_rule=CASCADE, required=True)
    score = FloatField(required=True)
    grade_type = StringField(choices=('exam', 'homework', 'lab'), required=True)
    date = DateTimeField(default=datetime.datetime.now)
```

Рис. 4.5 Модель Grade

4.3 Реалізація модуля авторизації

У файлі auth.py реалізовано логіку авторизації користувачів. Цей модуль відповідає за вхід у систему, створення токенів і перевірку прав доступу. В цьому ж модулі знаходяться маршрути, пов'язані з реєстрацією користувачів та входом у систему. Для цього використовується APIRouter із префіксом /api/auth.

Для зберігання паролів у системі використовується bcrypt. Його застосування пов'язане з високим рівнем захисту, оскільки алгоритм спеціально уповільнений і стійкий до атак перебором. Пароль користувача не зберігається у відкритому вигляді. Спочатку він проходить хешування через функцію get_password_hash(), після чого записується в базу даних.

Під час авторизації користувач вводить email і пароль. Система перевіряє ці дані за допомогою pwd_context.verify(), яка порівнює введений пароль із хешем у базі даних. Якщо дані коректні, користувач отримує JWT-токен.

JWT-токен використовується для того, щоб не виконувати повторну авторизацію при кожному запиті. Після входу він передається на клієнтську сторону і додається до кожного запиту до захищених ендпоінтів. У токені міститься email користувача та час завершення його дії. У даній системі термін дії становить дві години, що зменшує ризики у разі його компрометації. Для підпису використовується алгоритм HS256 і секретний ключ SECRET_KEY.

Функція get_current_user() використовується для перевірки користувача під час роботи із захищеними ендпоінтами. Вона отримує токен із заголовка Authorization, перевіряє його підпис і термін дії, після чого виконується пошук користувача у базі даних. У випадку помилки сервер повертає відповідь 401 Unauthorized, що означає відсутність або недійсність токена.

Окремо реалізовано перевірку прав доступу адміністратора через require_admin(). Це дозволяє обмежити доступ до критичних функцій системи. Якщо роль користувача не відповідає значенню "admin", система повертає помилку 403 Forbidden.

Також у системі реалізовано автоматичне створення облікових даних для студентів. Це спрощує процес реєстрації нових користувачів. Email генерується у форматі stXXXXXXXX@duikt.edu.ua, а пароль створюється як випадкова комбінація літер і цифр довжиною вісім символів. Після генерації пароль одразу хешується та зберігається у базі даних у захищеному вигляді.

4.4 Реалізація маршрутизаторів API

У даній реалізації застосунку всі API-ендпоінти не винесені в окремий модуль маршрутизації, а розміщені безпосередньо у файлі main.py. Такий вибір планування був обміркований, адже він спрощує загальну конфігурацію проєкту та дає змогу динамічніше розробляти й валідувати програмні функції. Уся логіка опрацювання запитів сконцентрована в одній точці, що є вигідним на початкових етапах створення програми.

Сам застосунок збудовано із застосуванням програмного каркаса FastAPI. Для взаємодії з постійним сховищем даних використовується MongoDB у поєднанні з бібліотекою MongoEngine. Окрім того, у проєкті налаштовано механізм CORS, що уможливлює клієнтській частині надсилати запити з різнорідних джерел, включно з локальними середовищами та вже розгорнутими веб-додатками.

Ключові можливості цієї системи умовно розділені на декілька блоків: робота зі здобувачами освіти, навчальними дисциплінами, отриманими балами, проведення аналізу та перевірка робочого стану сервера. Кожен із цих сегментів відповідає за свою частину обробки даних та взаємодіє з відповідними моделями бази даних.

Діяльність, пов'язана зі студентами, є однією з центральних функцій системи. При ініціалізації нового студента відбувається автоматичне формування електронної пошти та початкового пароля. Цей пароль негайно криптографується за допомогою bcrypt і зберігається у сховищі даних вже у вигляді шифру. Це гарантує, що отримати його у незахищеному вигляді є неможливим.

Отримання переліку студентів реалізоване спеціальним запитом, який повертає всі наявні записи з бази даних. Для детального ознайомлення з інформацією про певного студента застосовується запит за його унікальним ідентифікатором. У випадку, якщо запис не може бути знайдений, система видає повідомлення про помилку, що забезпечує коректну обробку таких ситуацій.

Процедура оновлення відомостей про студента реалізована у формі часткової модифікації. Це означає, що змінюються лише ті атрибути, які були передані в запиті, тоді як решта залишається незмінною. Це є зручним, оскільки усуває необхідність передавати повний набір даних. Виключення запису про студента відбувається разом із усіма пов'язаними з ним оцінками завдяки попередньо налаштованому каскадному видаленню в базі даних.

Можливості додавання, редагування та усунення студентів обмежені певними користувачами. Для цього застосовується алгоритм перевірки ролі: виконувати ці дії дозволено лише особі з адміністративним статусом. Подібний підхід забезпечує обмеження доступу до операцій, що мають критичне значення.

У системі також передбачена функція пакетного внесення студентів через файл формату CSV. Після завантаження вміст файлу перевіряється на

наявність усіх необхідних стовпців. Якщо структура відповідає вимогам, кожен рядок опрацьовується окремо та вноситься до бази даних як новий навчальний суб'єкт. У разі виявлення невідповідності повертається відповідне сповіщення.

Робота з освітніми дисциплінами забезпечується окремими кінцевими точками. Доступні функції для створення нових предметів та отримання їхнього загального списку. Кожна дисципліна містить назву та ім’я відповідального викладача.

Оцінки слугують для встановлення зв'язку між здобувачами освіти та навчальними курсами. Перш ніж додати бал, система спершу верифікує наявність відповідного студента та предмета. Це допомагає уникнути формування нерелевантних записів. Також надається запит для отримання всіх наявних оцінок.

Модуль аналітики збудований із залученням бібліотеки Pandas. За її допомогою розраховуються середні показники балів у межах груп та окремих дисциплін, а також загальна кількість зафіксованих оцінок у системі.

Окремо реалізовано механізм ідентифікації груп підвищеного ризику. До цієї категорії потрапляють студенти, чий середній академічний показник не сягає позначки 60 балів. Це спрощує швидкий пошук тих, хто має проблеми з успішністю.

Крім того, у системі існує опція вивантаження аналітичних відомостей у формат CSV. Це зручно для подальшої роботи з даними або для формування офіційних звітів.

Для підтвердження функціональності системи впроваджено простий API-запит, який повертає поточний стан сервера. За його допомогою можна оперативно перевірити, що програмний інтерфейс працює належним чином.

4.5 Реалізація аналітичного модуля на базі Pandas

Аналітичний модуль серверної частини реалізовано у файлі main.py з використанням бібліотеки Pandas. Його призначення полягає в обробці даних про оцінки студентів, розрахунку статистичних показників, визначенні студентів групи ризику та формуванні звітів у форматі CSV.

Дані для аналітики отримуються з колекції Grade, у якій зберігаються оцінки студентів із посиланнями на відповідного студента та навчальний предмет. Після отримання записів із бази даних інформація перетворюється у структуру DataFrame, що дозволяє виконувати групування, агрегацію та фільтрацію даних засобами Pandas.

Ендпоінт GET /api/analytics/dashboard відповідає за формування загальних статистичних показників. У межах цього маршруту всі оцінки завантажуються з бази даних, після чого для кожного запису формується словник із кодом групи, назвою предмета, оцінкою та типом роботи. На основі сформованого списку створюється DataFrame, у якому виконується групування за академічною групою та навчальним предметом. Це дозволяє визначити середній бал за групами, середній бал за предметами та загальну кількість оцінок у системі. Реалізацію цього ендпоінта наведено на рисунку 4.6.

```python
@app.get("/api/analytics/dashboard", tags=["Analytics"])
async def get_dashboard_data():
    grades_db = Grade.objects()

    if not grades_db:
        return {"message": "Немає даних для аналізу"}

    data = []
    for g in grades_db:
        data.append({
            "group_code": g.student.group_code,
            "subject_name": g.subject.name,
            "score": g.score,
            "grad_type": g.grade_type
        })

    df = pd.DataFrame(data)
    avg_by_group = df.groupby("group_code")["score"].mean().round(2).to_dict()
    avg_by_subject =df.groupby("subject_name")["score"].mean().round(2).to_dict()
    total_grades =len(df)

    return {
        "total_grades_count": total_grades,
        "avarege_score_by_group": avg_by_group,
        "average_score_by_subject": avg_by_subject
    }
```

Рис. 4.6 Реалізація ендпоінта формування аналітичної панелі

Ендпоінт GET /api/analytics/risk-group використовується для визначення студентів, які належать до групи ризику. Для цього з бази даних отримуються оцінки, а до набору даних додаються ідентифікатор студента, повне ім’я, академічна група та значення оцінки. Після створення DataFrame виконується групування за студентом, ім’ям і групою, після чого розраховується середній бал кожного студента. Студенти, середній бал яких є нижчим за 60, відбираються за допомогою фільтрації та повертаються у відповіді API. Реалізацію цього алгоритму наведено на рисунку 4.7.

```python
@app.get("/api/analytics/risk-group", tags=["Analytics"])
async def get_risk_group():
    """
    Аналіз студентів, які знаходяться в групі ризику (середній бал нижче 60).
    Використовує Pandas для групування та фільтрації.
    """
    grades_db = Grade.objects()

    if not grades_db:
        return {"message": "Немає даних для аналізу"}

    data = []
    for g in grades_db:
        data.append({
            "student_id": str(g.student.id),
            "full_name": f"{g.student.first_name} {g.student.last_name}",
            "group_code": g.student.group_code,
            "score": g.score
        })

    df = pd.DataFrame(data)
    student_stats = df.groupby(["student_id", "full_name", "group_code"])["score"].mean().reset_index()
    student_stats["score"] = student_stats["score"].round(2)
    risk_df = student_stats[student_stats["score"] < 60]
    risk_list = risk_df.to_dict(orient="records")

    return {
        "risk_count": len(risk_list),
```

Рис. 4.7 Реалізація ендпоінта визначення студентів групи ризику

Ендпоінт GET /api/analytics/export відповідає за формування CSV-звіту з результатами навчання студентів. Спочатку система отримує всі оцінки з бази даних і формує структурований список, що містить ПІБ студента, групу, предмет, оцінку, тип роботи та дату виставлення оцінки. Після цього дані перетворюються у DataFrame і записуються у CSV-формат за допомогою методу to_csv(). Для передавання файлу клієнту використовується StreamingResponse, що дозволяє сформувати файл у пам’яті без його збереження на сервері. Реалізацію експорту аналітичних даних наведено на рисунку 4.8.

```python
@app.get("/api/analytics/export", tags=["Analytics"])
async def export_analytics_csv():
    """
    Генерує CSV-файл з усіма оцінками для деканату.
    """
    grades_db = Grade.objects()

    if not grades_db:
        return {"message": "Немає даних для експорту"}
    data = []
    for g in grades_db:
        data.append({
            "Студент": f"{g.student.first_name} {g.student.last_name}",
            "Група": g.student.group_code,
            "Предмет": g.subject.name,
            "Оцінка": g.score,
            "Тип роботи": g.grade_type,
            "Дата виставлення": g.date.strftime("%Y-%m-%d %H:%M")
        })

    df = pd.DataFrame(data)
    stream = io.StringIO()
    df.to_csv(stream, index=False, encoding='utf-8-sig', sep=';')
    response = StreamingResponse(iter([stream.getvalue()]), media_type="text/csv")
    response.headers["Content-Disposition"] = "attachment; filename=student_analytics_report.csv"

    return response
```

Рис. 4.8 Реалізація ендпоінта експорту аналітичних даних у CSV

Таким чином, аналітичний модуль забезпечує автоматизовану обробку освітніх даних, розрахунок середніх показників успішності, визначення студентів групи ризику та формування звітів. Використання Pandas дозволяє виконувати ці операції компактно та зручно, оскільки основні розрахунки реалізуються через групування, агрегацію та фільтрацію табличних даних.

4.6 Тестування функціональності застосунку

Тестування серверної частини системи виконувалося методом ручного функціонального тестування через інтерактивний інтерфейс Swagger UI, який автоматично формується фреймворком FastAPI. Такий підхід дозволив перевірити коректність роботи API без використання додаткових клієнтських програм, оскільки всі запити до ендпоінтів можна виконувати безпосередньо в браузері.

Метою тестування було підтвердження правильності роботи основних функціональних модулів системи: авторизації користувачів, розмежування прав доступу, CRUD-операцій для студентів, предметів та оцінок, аналітичного модуля, а також експорту звітів у форматі CSV. Окремо перевірялася реакція системи на некоректні дії користувача, зокрема введення неправильного пароля, звернення до захищеного маршруту без JWT-токена та запит до неіснуючого ресурсу.

На рисунку 4.9 наведено загальний вигляд Swagger UI з переліком доступних API-маршрутів серверної частини системи.

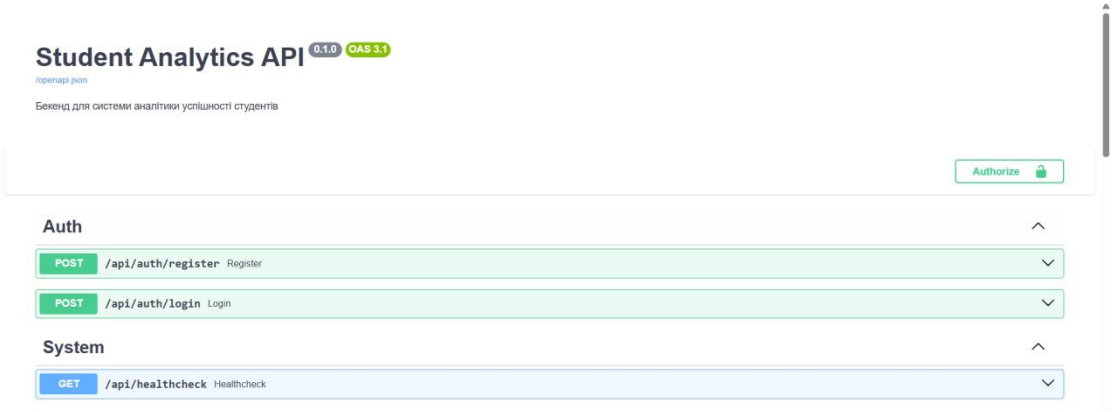


Рис. 4.9 Загальний вигляд Swagger UI

Перед виконанням запитів до захищених ендпоінтів проводилася авторизація користувача. Для цього через маршрут POST /api/auth/login передавалися email та пароль користувача. У разі успішної перевірки облікових даних сервер повертав JWT-токен, який надалі використовувався для доступу до захищених маршрутів API. Процес авторизації в Swagger UI наведено на рисунку 4.10.

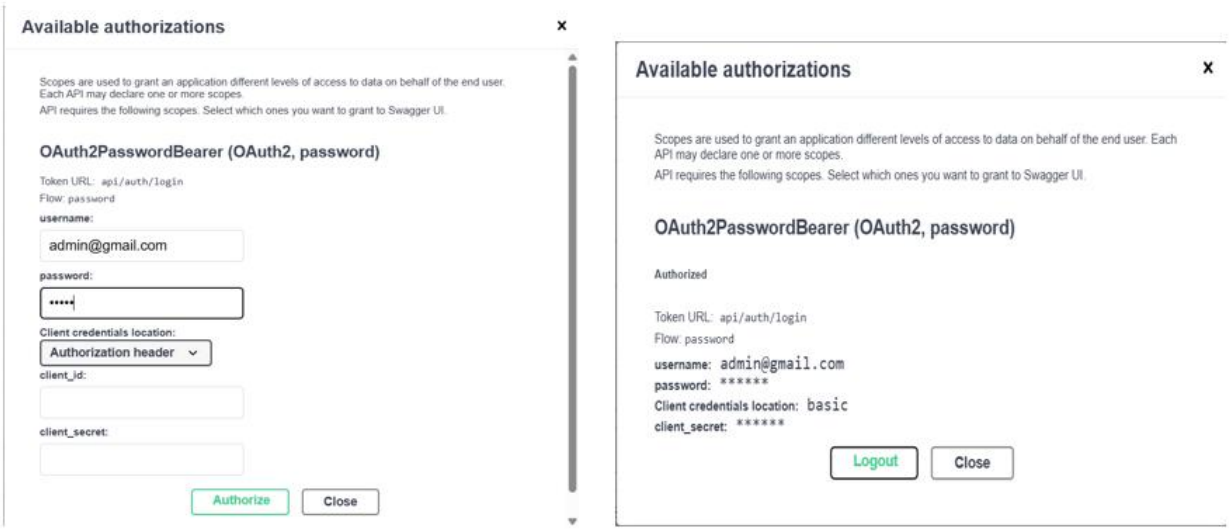


Рис. 4.10 Авторизація користувача через Swagger UI

Після отримання JWT-токена виконувалася перевірка основних CRUD-операцій. Для модуля студентів перевірялися запити на отримання списку студентів, створення нового студента, редагування наявного запису та видалення студента. Під час тестування контролювалося, чи правильно сервер приймає вхідні дані, чи створюється відповідний запис у базі даних та чи повертається коректна відповідь із відповідним HTTP-статусом.

На рисунку 4.11 показано приклад успішного отримання списку студентів за допомогою запиту GET /api/students.

```
[
  {
    "id": "69e390f099c88b1e51752ddf",
    "first_name": "string",
    "last_name": "string",
    "group_code": "string",
    "enrollment_year": 0,
    "email": "st25704787@duikt.edu.ua",
    "absences": 10
  },
  {
    "id": "69e390f099c88b1e51752de1",
    "first_name": "Орина",
    "last_name": "Штокало",
    "group_code": "TIP-43",
    "enrollment_year": 2024,
    "email": "st58143262@duikt.edu.ua",
    "absences": 33
  },
  {
    "id": "69e390f199c88b1e51752de2",
    "first_name": "Тарас",
    "last_name": "Скопенко",
    "group_code": "TIP-44",
    "enrollment_year": 2024,
    "email": "st71200008@duikt.edu.ua",
    "absences": 18
```


Рис. 4.11 Перевірка запиту отримання списку студентів

На рисунку 4.12 наведено приклад створення нового студента через запит POST /api/students. Під час виконання цього тесту перевірялося, чи створюється запис студента в базі даних, чи генеруються службові облікові дані та чи повертається відповідь із даними створеного студента.

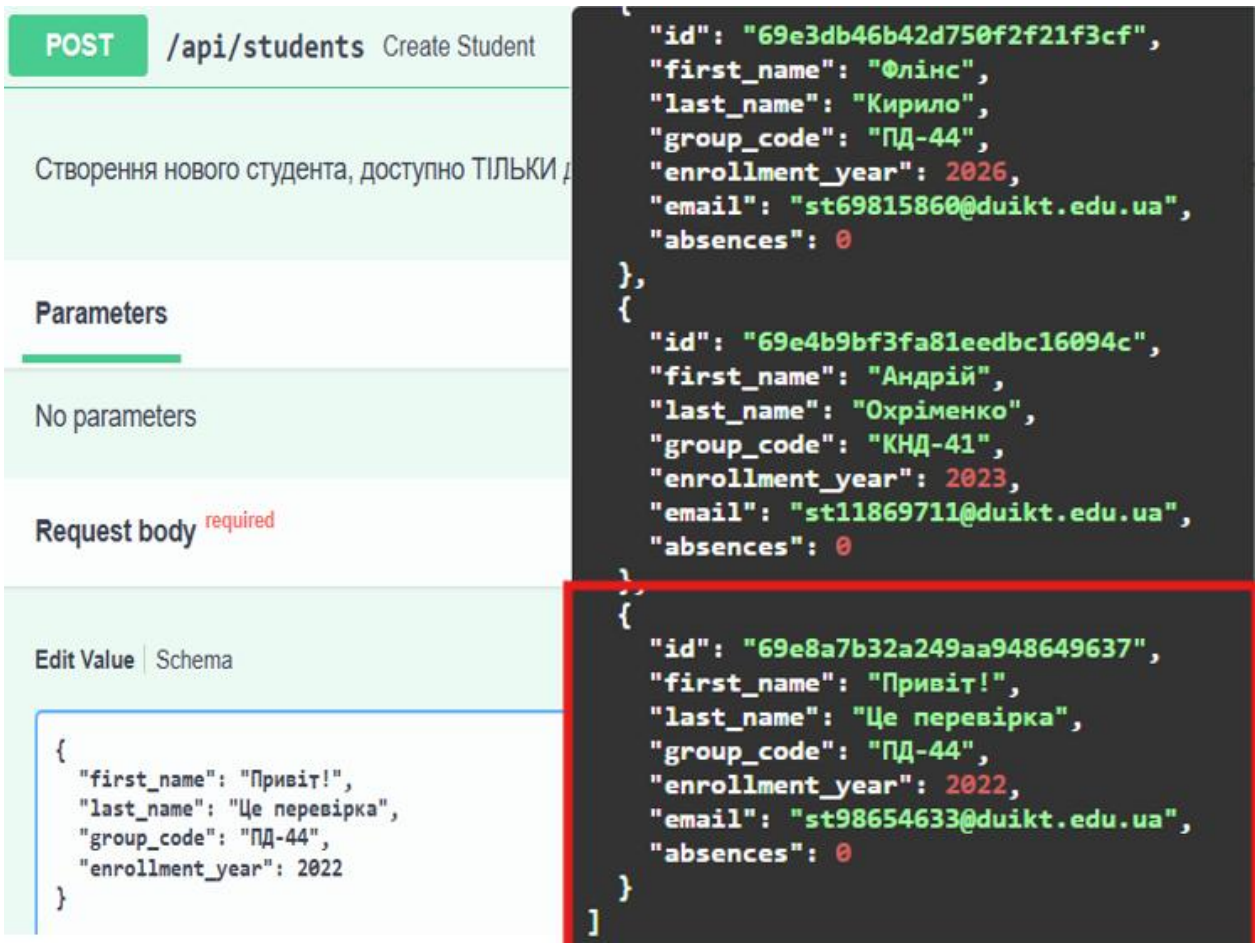


Рис. 4.12 Перевірка створення нового студента

Окремо перевірялася операція редагування даних студента. Для цього через запит PUT /api/students/{id} змінювалися окремі поля запису. Очікуваним результатом було оновлення лише тих даних, які були передані в тілі запиту, без зміни інших полів. Приклад перевірки редагування студента наведено на рисунку 4.13.

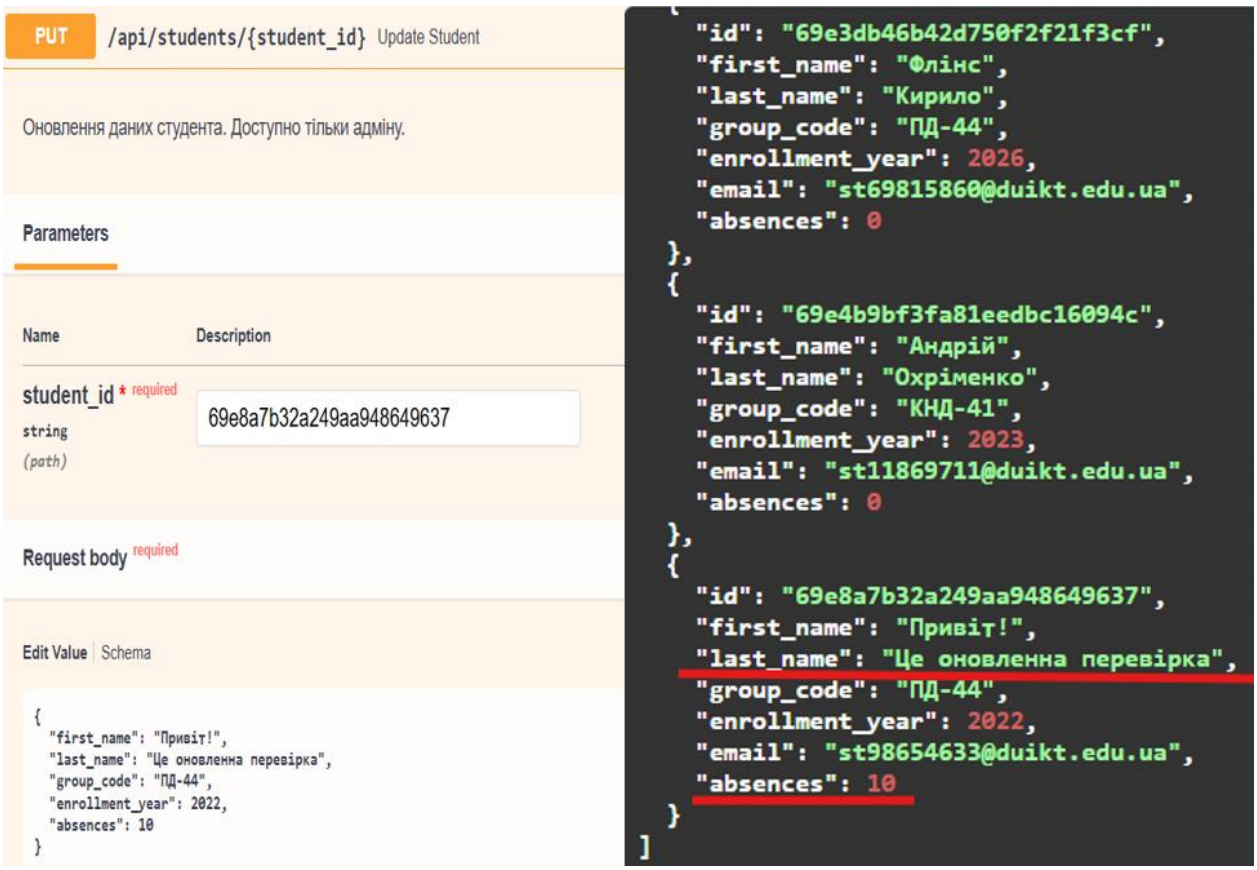


Рис. 4.13 Перевірка редагування даних студента

Також виконувалася перевірка видалення студента через запит DELETE /api/students/{id}. Після виконання запиту перевірялося, чи запис видаляється з бази даних та чи повертає сервер повідомлення про успішне виконання операції. Приклад тестування видалення студента наведено на рисунку 4.14.

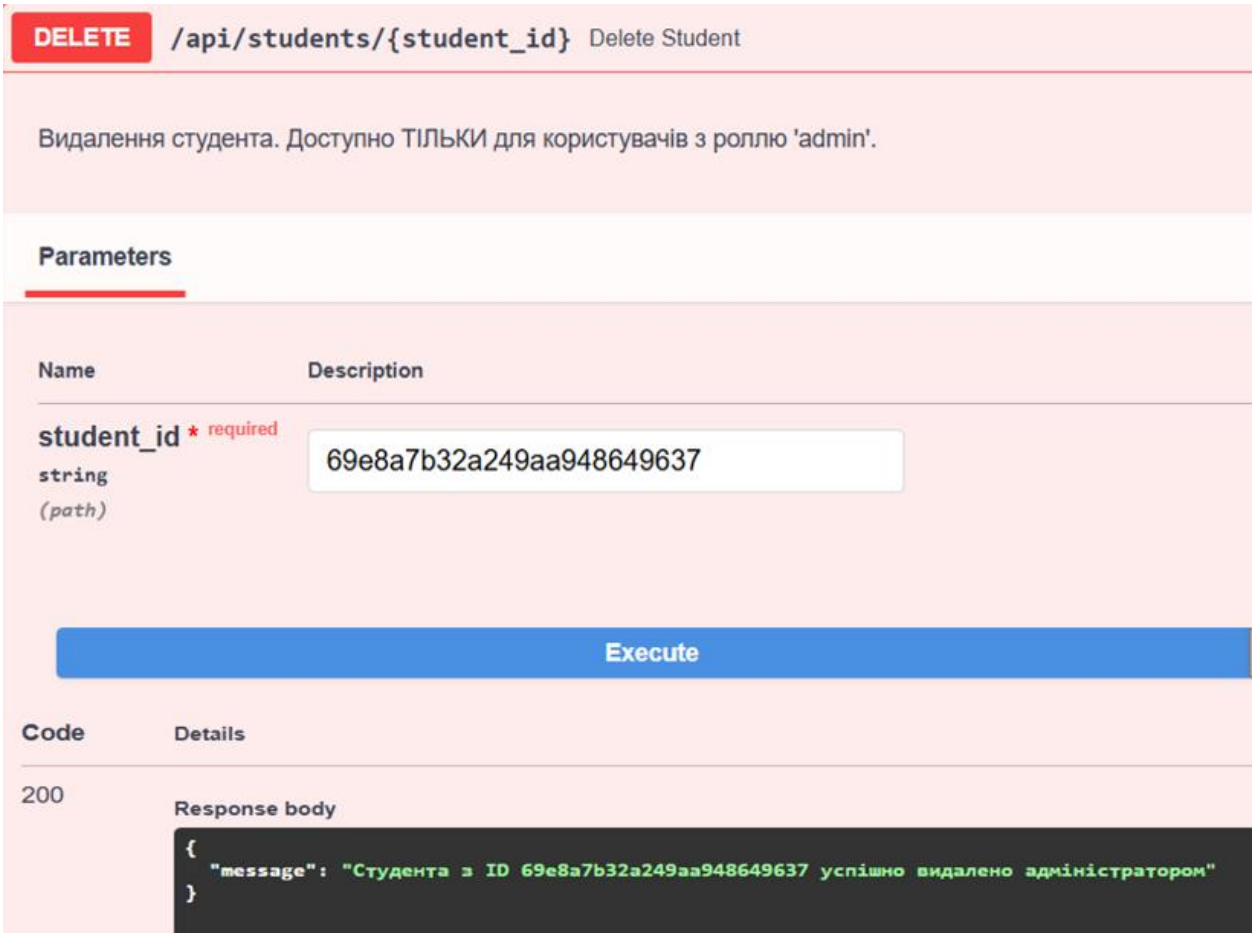


Рис. 4.14 Перевірка видалення студента

Для перевірки аналітичного модуля використовувалися маршрути GET /api/analytics/dashboard, GET /api/analytics/risk-group та GET /api/analytics/export. Під час тестування перевірялося, чи правильно система розраховує середній бал студентів, визначає студентів групи ризику за пороговим значенням 60 балів та формує CSV-звіт. Результати, отримані через API, порівнювалися з очікуваними значеннями, розрахованими на основі тестових даних.

На рисунку 4.15 наведено приклад відповіді аналітичного модуля із загальними статистичними показниками.

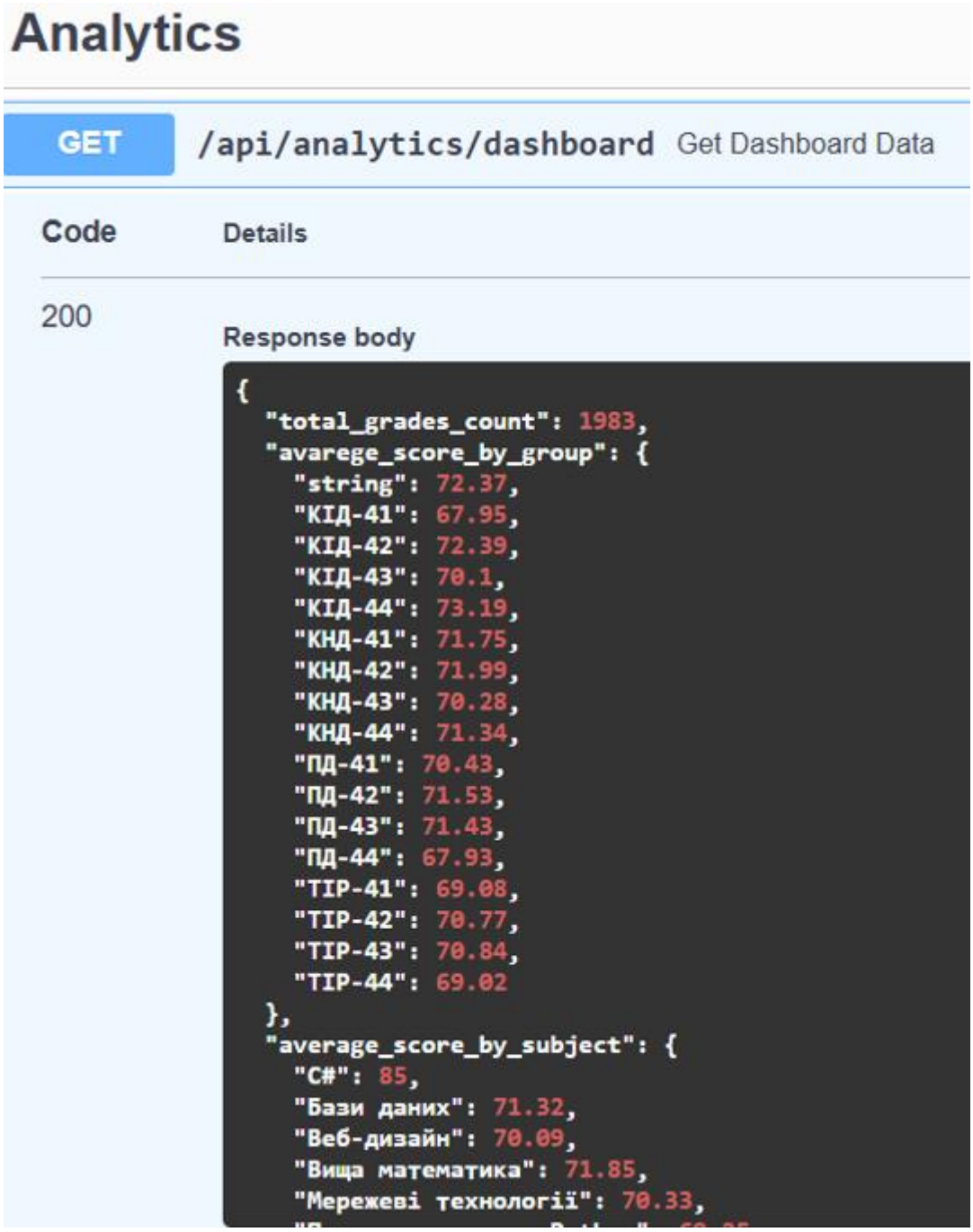


Рис. 4.15 Перевірка роботи аналітичного модуля

На рисунку 4.16 показано результат виконання запиту для отримання студентів групи ризику.

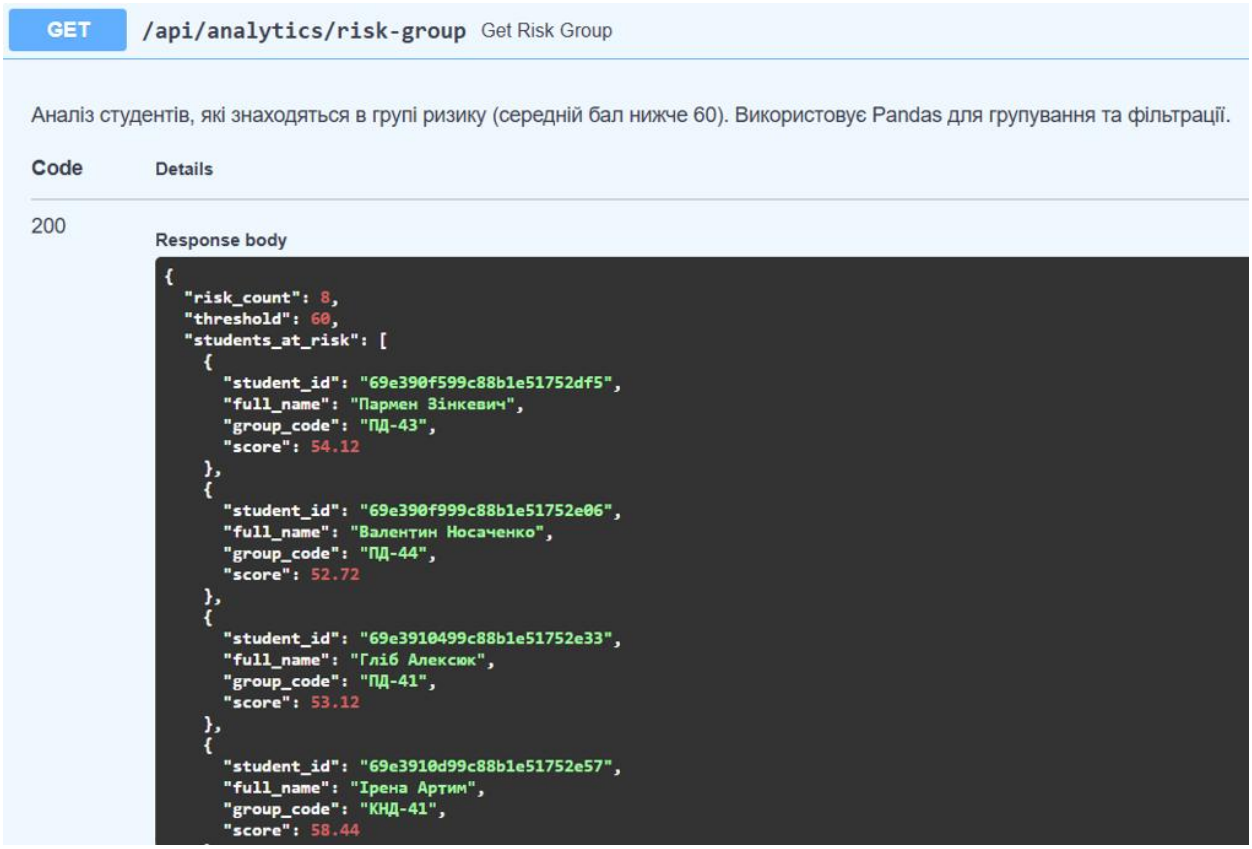


Рис. 4.16 Перевірка визначення студентів групи ризику

Для систематизації перевірки було сформовано набір тестових сценаріїв, наведений у таблиці 4.1.

Таблиця 4.1

Тестові сценарії перевірки функціональності API

№	Модуль	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Авторизація	Вхід користувача з правильними email і паролем	Сервер повертає JWT-токен і статус 200 OK	Виконано успішно
2	Авторизація	Вхід користувача з неправильним паролем	Сервер повертає помилку 401 Unauthorized	Виконано успішно
3	Авторизація	Запит до захищеного маршруту без токена	Сервер повертає помилку 401 Unauthorized	Виконано успішно
4	Студенти	Отримання списку студентів через GET /students	Сервер повертає список студентів	Виконано успішно
5	Студенти	Створення нового студента через POST /students	У базі даних створюється новий запис	Виконано успішно
6	Студенти	Редагування студента через PUT /students/{id}	Дані студента оновлюються	Виконано успішно
7	Студенти	Видалення студента через DELETE /students/{id}	Запис студента видаляється з бази даних	Виконано успішно
8	Предмети	Створення та перегляд навчальних предметів	Сервер коректно створює та повертає предмети	Виконано успішно
9	Оцінки	Додавання оцінки студенту	Оцінка зберігається та пов’язується зі студентом і предметом	Виконано успішно
10	Аналітика	Отримання загальної статистики через GET /analytics/dashboard	Сервер повертає середній бал і загальні показники	Виконано успішно
11	Аналітика	Отримання студентів групи ризику через GET /analytics/risk-group	Сервер повертає студентів із середнім балом нижче 60	Виконано успішно
12	Експорт	Формування CSV-звіту через GET /analytics/export	Сервер повертає CSV-файл для завантаження	Виконано успішно

ВИСНОВКИ

У результаті виконаної дипломної роботи розроблено серверну частину системи аналізу успішності студентів, яка реалізує безпечний REST-інтерфейс, забезпечує збереження даних у хмарній нереляційній базі даних та виконує аналітичну обробку показників успішності за допомогою бібліотеки Pandas.

1. Проведено аналіз предметної галузі систем моніторингу та аналізу академічної успішності студентів. Визначено основні проблеми традиційних підходів до обробки освітніх даних, зокрема значні витрати часу на ручне формування звітів, складність оперативного виявлення студентів групи ризику та обмежені можливості для швидкого отримання статистичних показників.

2. Проведено порівняльний аналіз існуючих програмних рішень, зокрема Canvas LMS, Moodle та Blackboard Predict. За результатами аналізу встановлено, що розглянуті системи здебільшого орієнтовані на ведення електронних журналів, організацію навчального процесу або формування базових звітів, проте мають обмежені можливості щодо аналітичної обробки освітніх даних, відкритої інтеграції через API та автоматизованого визначення студентів групи ризику.

3. Визначено функціональні та нефункціональні вимоги до серверної частини системи. До основних функціональних вимог належать керування даними студентів, навчальних предметів та оцінок, авторизація користувачів, розмежування прав доступу, масове завантаження даних із CSV-файлів, формування аналітичної панелі, визначення студентів групи ризику та експорт звітів. Серед нефункціональних вимог визначено безпеку, продуктивність, масштабованість, контейнеризацію та наявність документації API.

4. Обґрунтовано вибір засобів реалізації серверної частини системи. Для розробки було використано мову програмування Python, веб-фреймворк

FastAPI, хмарну базу даних MongoDB Atlas, бібліотеку Pandas, технологію контейнеризації Docker, хмарну платформу Render та систему контролю версій Git і GitHub. Обраний технологічний стек забезпечив можливість реалізації REST API, захищеної авторизації, збереження освітніх даних та їх подальшої аналітичної обробки.

5. Розроблена архітектура системи побудована за клієнт-серверним принципом і включає клієнтський рівень, серверну частину та рівень збереження даних. Спроєктовано структуру бази даних із чотирьох основних колекцій: User, Student, Subject та Grade. Також розроблено діаграми класів основної логіки, діаграму класів модуля авторизації, карту вебзастосунку з розподілом прав доступу, діаграму варіантів використання та діаграми послідовностей.

6. Реалізована серверна частина системи містить моделі даних на основі MongoEngine, REST API для роботи зі студентами, предметами та оцінками, механізм JWT-авторизації з хешуванням паролів за допомогою bcrypt, а також розмежування прав доступу між адміністратором і викладачем. Додатково реалізовано механізм масового завантаження студентів із CSV-файлу, що спрощує початкове наповнення системи даними.

7. Аналітичний модуль реалізовано з використанням бібліотеки Pandas. Він забезпечує обчислення середнього балу студентів, визначення студентів групи ризику за встановленим пороговим значенням, формування загальних статистичних показників та генерацію CSV-звітів. Саме поєднання обліку освітніх даних із їх автоматизованою аналітичною обробкою є основною якісною відмінністю розробленої системи від розглянутих аналогів.

8. Тестування функціональності системи виконано за допомогою інтерфейсу Swagger UI. У процесі тестування перевірено роботу основних API-маршрутів, механізму авторизації, операцій керування студентами, предметами та оцінками, а також функцій аналітичного модуля. Результати тестування підтвердили коректність роботи серверної частини системи та відповідність реалізованого функціоналу визначеним вимогам.

Розроблена система може використовуватися в закладах вищої освіти для автоматизації обліку та аналізу навчальних результатів студентів. Подальший розвиток системи може передбачати додавання прогнозування академічних ризиків, інтеграцію з іншими освітніми інформаційними системами та реалізацію механізму автоматичного надсилання повідомлень користувачам.

Кваліфікаційна робота пройшла апробацію. За результатами роботи було опубліковано тези доповідей:

1. Каплюченко С.Р., Цапро І.В. Архітектура БД для аналітики успішності студентів за допомогою MongoEngine. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії». 26 листопада 2025 року. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2025. С.144-145.

2. Каплюченко С.Р., Цапро І.В. Використання REST API та нереляційних баз даних для розробки серверної частини системи аналітики успішності студентів. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 23 квітня 2026 року. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2026. С.307-309.

ПЕРЕЛІК ПОСИЛАНЬ

1. Canvas LMS: сучасна платформа для управління навчанням у цифрову епоху. 7sky - Комфортні простори для навчання. URL: https://7sky.ua/blog/canvas-lms-a-modern-learning-management-platform-for-the-digital-age/

2. На головну | Moodle.org. Home | Moodle.org. URL: https://moodle.org/?lang=uk

3. Analytics for Learn. Redirect. URL: https://help.anthology.com/blackboard/instructor/en/analytics/analytics-for-learn.html?utm_source=chatgpt.com

4. The best way to code with AI. *Cursor*. URL: https://cursor.com/

5. Python 3.14 documentation. *Python documentation*. URL: https://docs.python.org/

6. Lubanovic B. Introducing python. O'Reilly Media, Incorporated, 2014. 253 с. URL: https://books.google.com.ua/books?hl=uk&lr=&id=WpHhEAAAQBAJ&oi=fnd&pg=PP1&dq=FastAPI&ots=2sW7lejPTv&sig=gm9tT5Cti8RkOabw0nVaEJeJrkg&redir_esc=y#v=onepage&q=FastAPI&f=false

7. DevDocs - FastAPI documentation. *DevDocs API Documentation*. URL: https://devdocs.io/fastapi/

8. Безпека - перші кроки - FastAPI. *FastAPI*. URL: https://fastapi.tiangolo.com/uk/tutorial/security/first-steps/

9. What is mongodb atlas? - atlas - mongodb docs. *MongoDB: The World's Leading Modern Data Platform | MongoDB*. URL: https://www.mongodb.com/docs/atlas/

10. Atlas database. *MongoDB: The World's Leading Modern Data Platform | MongoDB*. URL: https://www.mongodb.com/products/platform/atlas-database

11. MongoEngine docs - python object-document mapper for mongodb - documentation. *MongoEngine Docs - Python Object-Document Mapper for MongoDB - Documentation*. URL: https://docs.mongoengine.org/

12. Pandas documentation - pandas 3.0.2 documentation. *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/docs/

13. Docker Inc. Home. *Docker Documentation*. URL: https://docs.docker.com/

14. Render | The cloud for builders. *Render*. URL: https://render.com/

15. GitHub.com help documentation. *GitHub Docs*. URL: https://docs.github.com/

16. JSON web token introduction - jwt.io. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction

17. Peter Jones. Advanced microservice security: implementing oauth2 and JWT. Walzone Press, 2024. 270 c. URL: https://www.google.com.ua/books/edition/Advanced_Microservice_Security_Implement/ioo9EQAAQBAJ?hl=uk&gbpv=0

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Система аналітики успішності студентів. Спец-частина: розробка Backend-частини системи

Виконав студент 4 курсу
групи ПД-44
Каплюченко Святослав Романович
Керівник роботи
викладач кафедри ІПЗ Цапро Ігор Вікторович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу аналізу успішності студентів за допомогою впровадження комплексної системи для безпечного збереження, агрегації та візуалізації освітніх даних.

Об'єкт дослідження – процес аналізу успішності студентів на основі збереження, обробки та статистичного аналізу освітніх даних.

Предмет дослідження – технології, алгоритми агрегації, інструменти створення back-end частин систем аналітики освітніх даних.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі з проблеми цифровізації освітнього процесу та автоматизації аналітики навчальної діяльності студентів.
2. Провести аналіз існуючих інформаційних систем управління навчанням з функцією автоматичного створення аналітики успішності студентів, визначити їх переваги та недоліки.
3. Сформувати функціональні та нефункціональні вимоги до розроблюваної інформаційної системи.
4. Провести огляд засобів для розробки серверної частини системи аналітики успішності.
5. Спроєктувати архітектуру серверної частини системи (структуру API та бази даних).
6. Розробити back-end частину системи для автоматизованої агрегації даних, розрахунку показників успішності та візуалізації результатів на інтерактивних дашбордах.
7. Провести функціональне тестування розробленої інформаційної системи аналітики успішності студентів.

3

АНАЛІЗ АНАЛОГІВ

Параметр	Canvas LMS	Moodle	Blackboard Predict	"SDF"
Тип рішення	Хмарна LMS	Гібридна LMS	Система передбачення успішності	Веб-система аналітики та візуалізації даних
Складність впровадження	Середня (час на інтеграцію та навчання)	Висока (повне самостійне налаштування)	Висока (попередня робота з вхідними даними)	Низька (Docker-контейнеризація)
Гнучкість UI	Висока (редактор тем)	Низька (застарілий інтерфейс)	Низька (лише налаштування фільтрів)	Спеціалізована (під конкретні потреби)
Розширюваність	Середня (зовнішні сервіси)	Висока (відкритий код)	Відсутня	Висока
Підтримувані БД	PostgreSQL	PostgreSQL, MySQL, Oracle	Інтегрована	MongoDB

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

1. Управління даними студентів, предметів та оцінок (CRUD).
2. Автентифікація користувачів та розмежування прав доступу.
3. Математична агрегація та статистична обробка даних успішності.
4. Формування та експорт готових аналітичних звітів.

Нефункціональні вимоги:

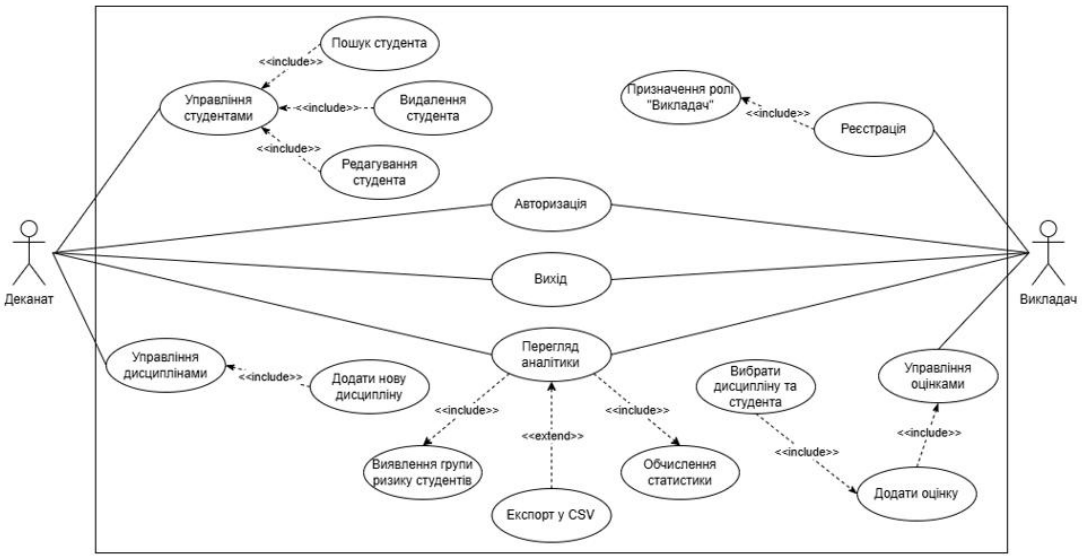
1. Безпека: захист маршрутів API (JWT-токени), хешування паролів (bcrypt), стійкість до NoSQL-ін'єкцій.
2. Продуктивність: висока швидкодія сервера (відповідь API < 200 мс).
3. Масштабованість: гнучка серверна архітектура з використанням нереляційної бази даних.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

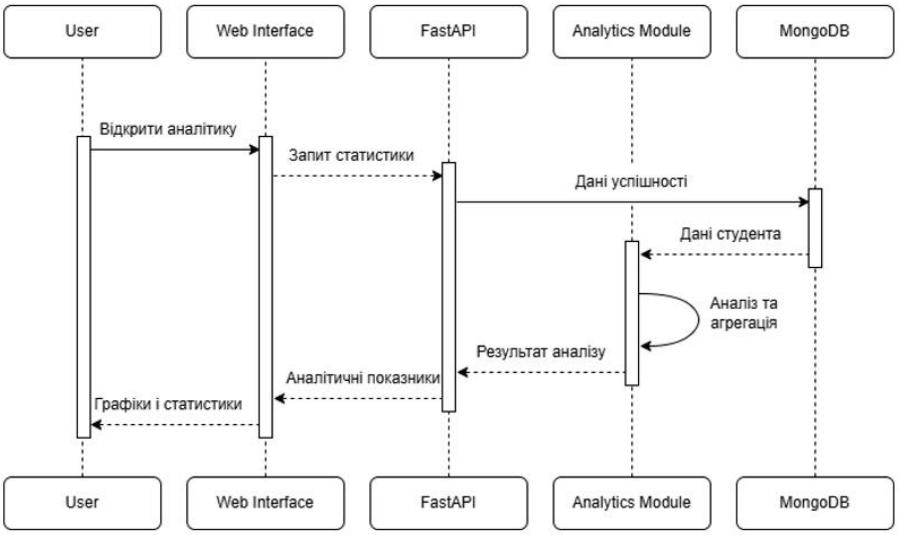
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



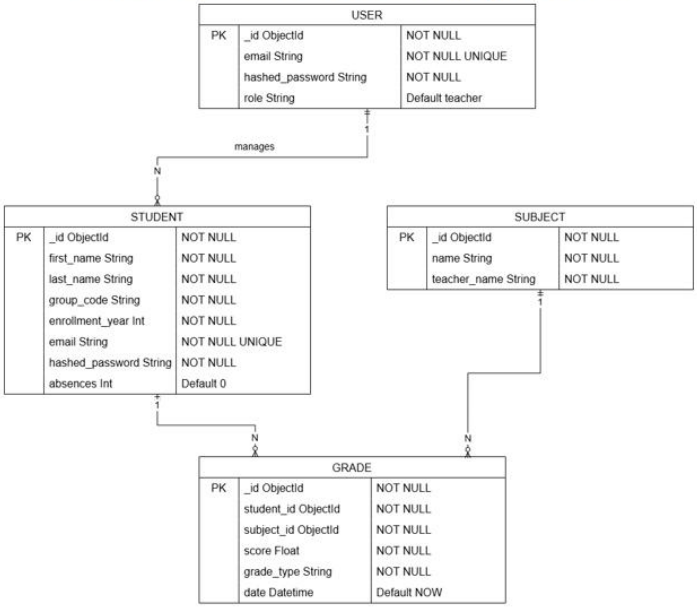
7

ДІАГРАМА ПОСЛІДОВНОСТІ СТВОРЕННЯ АНАЛІТИКИ



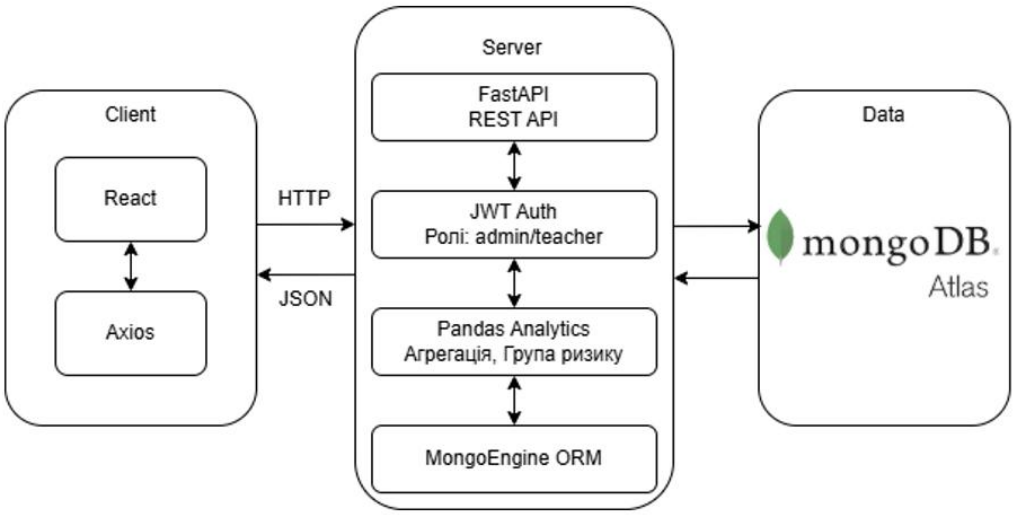
8

СТРУКТУРА БАЗИ ДАНИХ



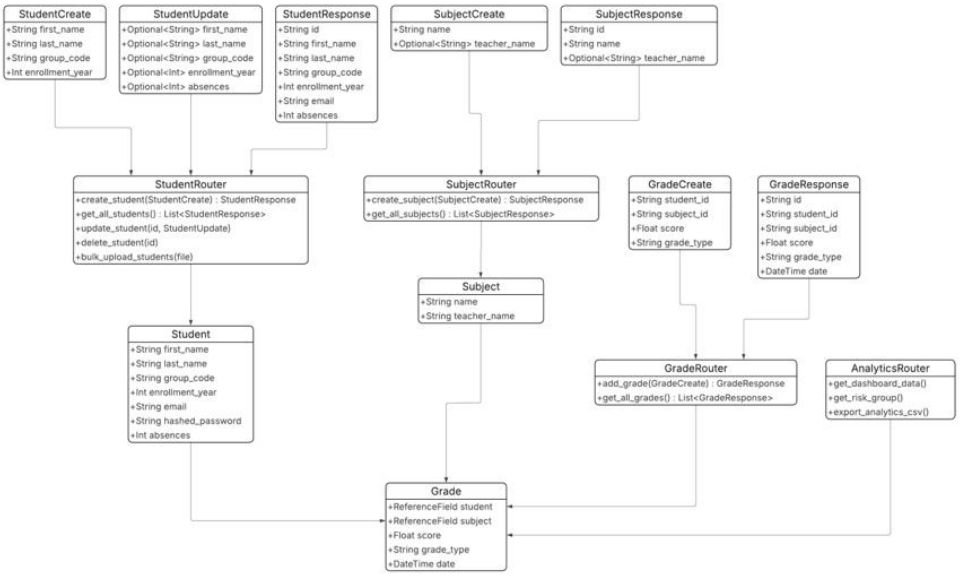
9

АРХІТЕКТУРА ЗАСТОСУНКУ



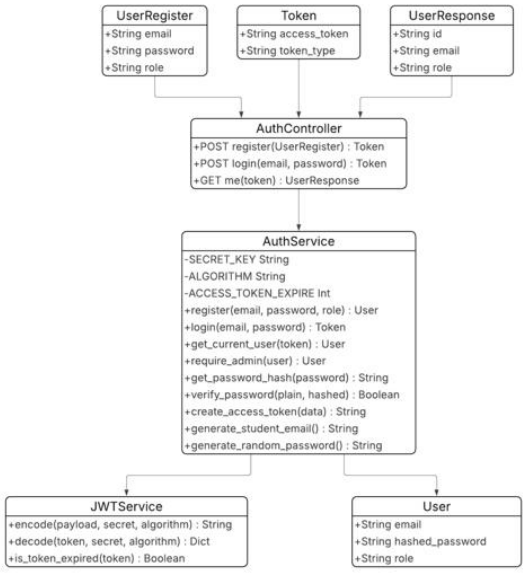
10

ДІАГРАМА КОМПОНЕНТІВ СЕРВЕРНОЇ ЛОГІКИ



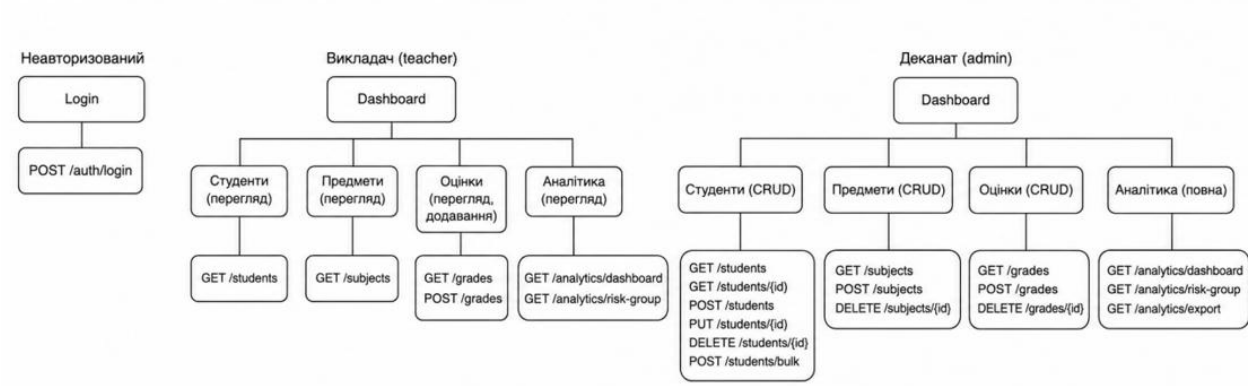
11

ДІАГРАМА КЛАСІВ АВТОРИЗАЦІЇ



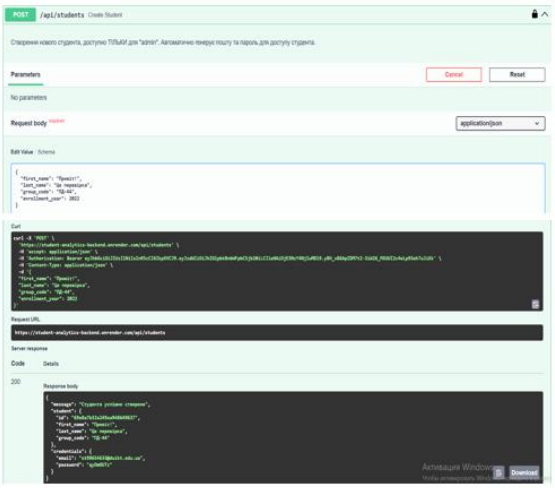
12

МАПИ ВЕБ-ЗАСТОСУНКУ



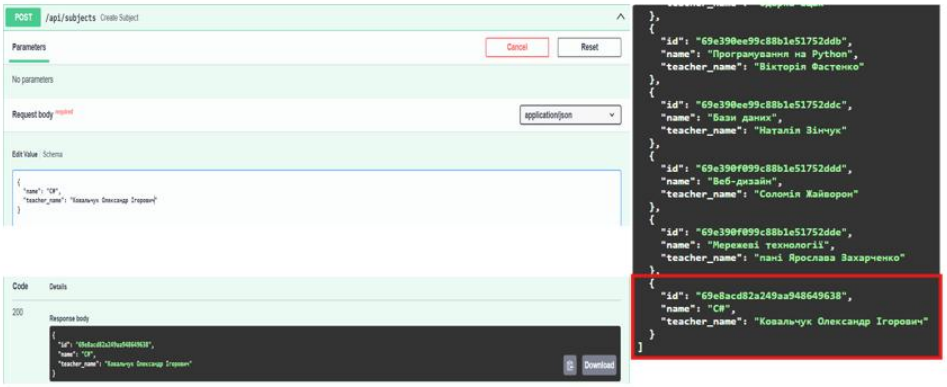
13

ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ



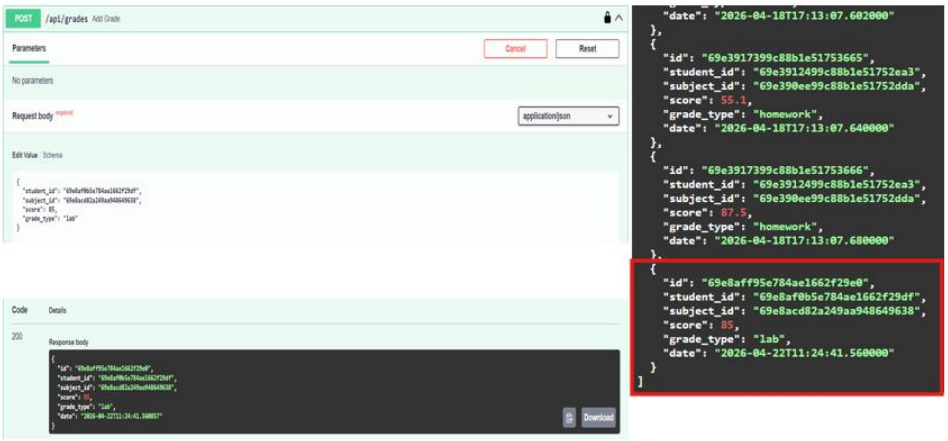
14

ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ



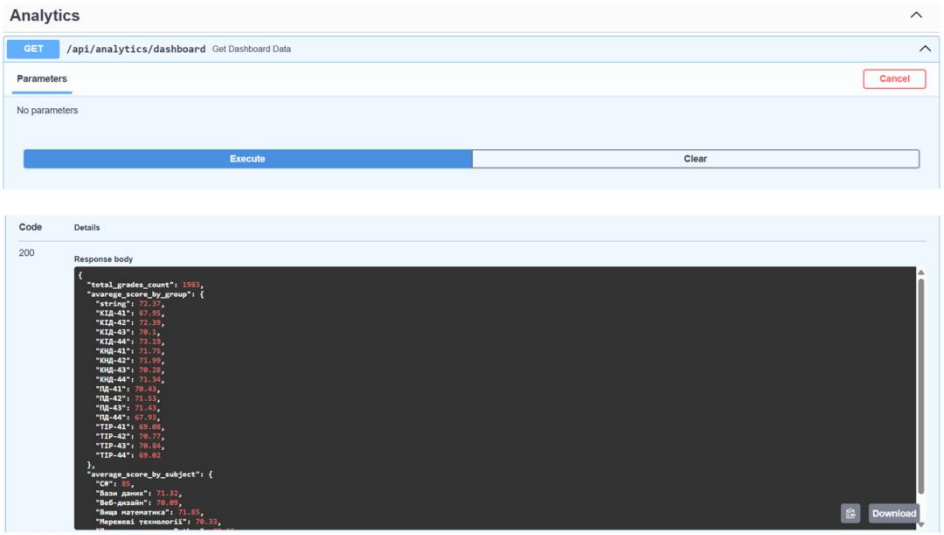
15

ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ

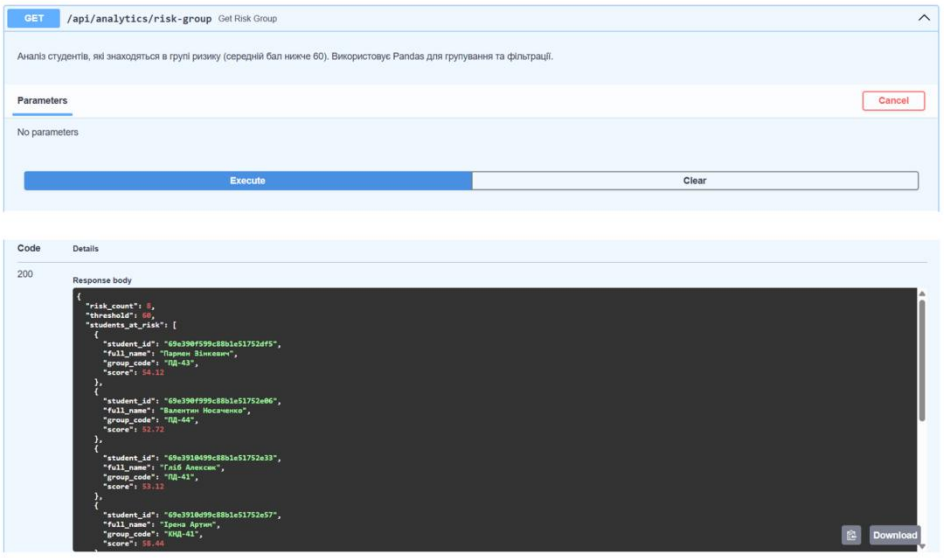


16

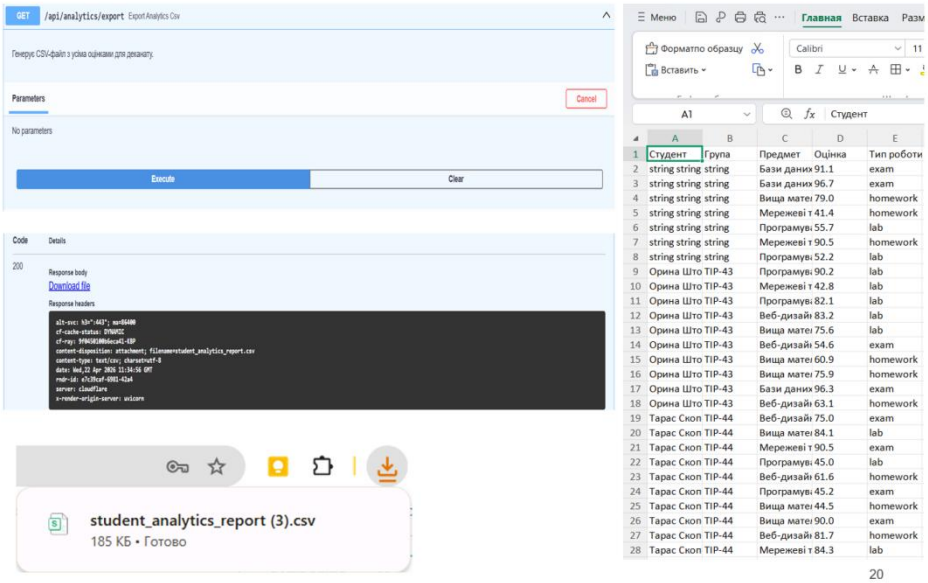
ЕКРАННІ ФОРМИ



ЕКРАННІ ФОРМИ



ЕКРАННІ ФОРМИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Каплюченко С.Р., Цапро І.В. Архітектура БД для аналітики успішності студентів за допомогою MongoEngine. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії». 26 листопада 2025 року. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2025. С.144-145.
2. Каплюченко С.Р., Цапро І.В. Використання REST API та нереляційних баз даних для розробки серверної частини системи аналітики успішності студентів. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 23 квітня 2026 року. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.: ДУІКТ, 2026. С.307-309.

ВИСНОВКИ

1. Проведено аналіз предметної галузі для з’ясування проблем автоматизації обліку та аналізу успішності студентів у закладах вищої освіти.
2. Проведено аналіз існуючих систем обліку успішності та ідентифіковано проблеми, пов'язані з відсутністю інструментів математичної агрегації даних та своєчасного виявлення студентів у групі ризику.
3. Сформовано вимоги до серверної частини системи аналітики успішності студентів та спроєктовано архітектуру застосунку.
4. Проаналізовано засоби розробки backend-частини системи та обґрунтовано вибір технологій FastAPI, MongoDB та Pandas.
5. Розроблено серверну частину системи аналітики успішності студентів з реалізацією FastAPI, JWT-авторизації з розмежуванням ролей адміністратора та викладача.
6. Реалізовано модуль аналітики на базі бібліотеки Pandas для розрахунку середнього балу, виявлення студентів у групі ризику та експорту звітів у форматі CSV.
7. Проведено розгортання застосунку у хмарному середовищі Render з використанням Docker-контейнеризації та підключенням до MongoDB Atlas.
8. Проведено функціональне тестування роботи серверної частини, підтверджено відповідність розробленого функціоналу вимогам.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

main.py

```
from fastapi import FastAPI, HTTPException, Depends, File, UploadFile
from fastapi.responses import StreamingResponse
import io
import pandas as pd
from mongoengine import connect, disconnect
from fastapi.middleware.cors import CORSMiddleware
from models import Student, Subject, Grade
from schemas import StudentCreate, StudentResponse, StudentUpdate, SubjectCreate, SubjectResponse, GradeCreate, GradeResponse
from auth import (
    auth_router,
    oauth2_scheme,
    require_admin,
    get_password_hash,
    generate_random_password,
    generate_student_email
)

app = FastAPI(
    title="Student Analytics API",
    description="Бекенд для системи аналітики успішності студентів"
)

origins = [
    "http://localhost:5173",
    "http://127.0.0.1:5173",
    "http://localhost:4173",
    "https://student-analytics-backend.onrender.com",
    "https://thesis-2026-xi.vercel.app"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(auth_router)

@app.on_event("startup")
async def startup_db_client():

connect(host="mongodb+srv://sviaticrrocer_db_user:KhhdFOGFIq98QMIx@cluster0.ncqipo7.mongodb.net/student_analytics?retryWrites=true&w=majority")
    print("Підключено до MongoDB")

@app.on_event("shutdown")
async def shutdown_db_client():
    disconnect()
    print("Відключено від MongoDB")

@app.get("/api/healthcheck", tags=["System"])
async def healthcheck():
    return {"status": "ok", "message": "FastAPI працює, інфраструктура готова"}


@app.post("/api/students", tags=["Students"])
async def create_student(student_data: StudentCreate, current_user = Depends(require_admin)):
    new_email = generate_student_email()
    raw_password = generate_random_password()
    hashed_pwd = get_password_hash(raw_password)

    new_student = Student(
        first_name=student_data.first_name,
        last_name=student_data.last_name,
        group_code=student_data.group_code,
        enrollment_year=student_data.enrollment_year,
        email=new_email,
```

```
            hashed_password=hashed_pwd,
            role="student"
        )
        new_student.save()

        return {
            "message": "Студента успішно створено",
            "student": {
                "id": str(new_student.id),
                "first_name": new_student.first_name,
                "last_name": new_student.last_name,
                "group_code": new_student.group_code
            },
            "credentials": {
                "email": new_email,
                "password": raw_password
            }
        }

    @app.get("/api/students",
response_model=list[StudentResponse],
tags=["Students"])
    async def get_all_students():
        students_db = Student.objects()

        result = []
        for s in students_db:
            result.append({
                "id": str(s.id),
                "first_name": s.first_name,
                "last_name": s.last_name,
                "group_code": s.group_code,
                "enrollment_year": s.enrollment_year,
                "email": s.email,
                "absences": getattr(s, 'absences', 0)
            })
        return result

    @app.put("/api/students/{student_id}",
tags=["Students"])
    async def update_student(
        student_id: str,
        student_data: StudentUpdate,
        current_user = Depends(require_admin)
    ):
        student = Student.objects(id=student_id).first()
        if not student:
            raise HTTPException(status_code=404,
detail="Студента не знайдено")

        update_data =
student_data.dict(exclude_unset=True)

        if not update_data:
            raise HTTPException(status_code=400,
detail="Не надано даних для оновлення")

        student.update(**update_data)
        student.reload()

        return {"message": "Дані студента оновлено
успішно", "student": student.to_json()}

    @app.delete("/api/students/{student_id}",
tags=["Students"])
    async def delete_student(student_id: str,
current_user = Depends(require_admin)):
        student = Student.objects(id=student_id).first()

        if not student:
            raise HTTPException(status_code=404,
detail="Студента не знайдено")

        student.delete()
        return {"message": f"Студента з ID {student_id}
успішно видалено адміністратором"}

    @app.post("/api/students/bulk-upload",
tags=["Students"])
    async def bulk_upload_students(file: UploadFile =
File(...)):
        content = await file.read()

        try:
```

```
            df = pd.read_csv(io.BytesIO(content),
sep=None, engine='python')

            required_columns = {"first_name",
"last_name", "group_code", "enrollment_year"}
            if not required_columns.issubset(df.columns):
                return {"error": f"Помилка! Файл має
містити такі колонки: {required_columns}"}

            added_count = 0
            for index, row in df.iterrows():
                new_student = Student(
                    first_name=str(row['first_name']),
                    last_name=str(row['last_name']),
                    group_code=str(row['group_code']),

enrollment_year=int(row['enrollment_year']) if
pd.notna(row['enrollment_year']) else 2024
                )
                new_student.save()
                added_count += 1

            return {"message": f"Супер! Успішно додано
{added_count} студентів до бази!"}

        except Exception as e:
            return {"error": f"Помилка обробки файлу:
{str(e)}"}

    @app.post("/api/subjects",
response_model=SubjectResponse, tags=["Subjects"])
    async def create_subject(subject_data:
SubjectCreate):
        new_subject = Subject(
            name=subject_data.name,
            teacher_name=subject_data.teacher_name
        )
        new_subject.save()
        return {
            "id": str(new_subject.id),
            "name": new_subject.name,
            "teacher_name": new_subject.teacher_name
        }

    @app.get("/api/subjects",
response_model=list[SubjectResponse],
tags=["Subjects"])
    async def get_all_subjects():
        subjects_db = Subject.objects()
        return [{"id": str(s.id), "name": s.name,
"teacher_name": s.teacher_name} for s in subjects_db]

    @app.post("/api/grades",
response_model=GradeResponse, tags=["Grades"])
    async def add_grade(grade_data: GradeCreate,
token: str = Depends(oauth2_scheme)):
        student =
Student.objects(id=grade_data.student_id).first()
        subject =
Subject.objects(id=grade_data.subject_id).first()

        if not student:
            raise HTTPException(status_code=404,
detail="Студента не знайдено")
        if not subject:
            raise HTTPException(status_code=404,
detail="Предмет не знайдено")

        new_grade = Grade(
            student=student,
            subject=subject,
            score=grade_data.score,
            grade_type=grade_data.grade_type
        )
        new_grade.save()

        return {
            "id": str(new_grade.id),
            "student_id": str(new_grade.student.id),
            "subject_id": str(new_grade.subject.id),
            "score": new_grade.score,
            "grade_type": new_grade.grade_type,
            "date": new_grade.date
        }
```

```
    @app.get("/api/grades",
response_model=list[GradeResponse], tags=["Grades"])
    async def get_all_grades():
        grades_db = Grade.objects()

        result = []
        for g in grades_db:
            result.append({
                "id": str(g.id),
                "student_id": str(g.student.id),
                "subject_id": str(g.subject.id),
                "score": g.score,
                "grade_type": g.grade_type,
                "date": g.date
            })
        return result

    @app.get("/api/analytics/dashboard",
tags=["Analytics"])
    async def get_dashboard_data():
        grades_db = Grade.objects()

        if not grades_db:
            return {"message": "Немає даних для
аналізу"}

        data = []
        for g in grades_db:
            data.append({
                "group_code": g.student.group_code,
                "subject_name": g.subject.name,
                "score": g.score,
                "grad_type": g.grade_type
            })

        df = pd.DataFrame(data)
        avg_by_group =
df.groupby("group_code")["score"].mean().round(2).to
_dict()
        avg_by_subject
=df.groupby("subject_name")["score"].mean().round(2)
.to_dict()
        total_grades =len(df)

        return {
            "total_grades_count": total_grades,
            "avarege_score_by_group": avg_by_group,
            "average_score_by_subject": avg_by_subject
        }

    @app.get("/api/analytics/risk-group",
tags=["Analytics"])
    async def get_risk_group():
        grades_db = Grade.objects()

        if not grades_db:
            return {"message": "Немає даних для
аналізу"}

        data = []
        for g in grades_db:
            data.append({
                "student_id": str(g.student.id),
                "full_name": f"{g.student.first_name}
{g.student.last_name}",
                "group_code": g.student.group_code,
                "score": g.score
            })

        df = pd.DataFrame(data)
        student_stats = df.groupby(["student_id",
"full_name",
"group_code"])["score"].mean().reset_index()
        student_stats["score"] =
student_stats["score"].round(2)
        risk_df = student_stats[student_stats["score"] <
60]
        risk_list = risk_df.to_dict(orient="records")

        return {
            "risk_count": len(risk_list),
```

```
        "threshold": 60,
        "students_at_risk": risk_list
    }

@app.get("/api/analytics/export", tags=["Analytics"])
async def export_analytics_csv():
    grades_db = Grade.objects()

    if not grades_db:
        return {"message": "Немає даних для експорту"}

    data = []
    for g in grades_db:
        data.append({
            "Студент": f"{g.student.first_name} {g.student.last_name}",
            "Група": g.student.group_code,
            "Предмет": g.subject.name,
            "Оцінка": g.score,
            "Тип роботи": g.grade_type,
            "Дата виставлення": g.date.strftime("%Y-%m-%d %H:%M")
        })

    df = pd.DataFrame(data)

    stream = io.StringIO()
    df.to_csv(stream, index=False, encoding='utf-8-sig', sep=';')

    response = StreamingResponse(iter([stream.getvalue()]), media_type="text/csv")
    response.headers["Content-Disposition"] = "attachment; filename=student_analytics_report.csv"

    return response
```

models.py

```
from mongoengine import Document, StringField, IntField, FloatField, DateTimeField, ReferenceField, CASCADE
import datetime

class Student(Document):
    first_name = StringField(required=True)
    last_name = StringField(required=True)
    group_code = StringField(required=True)
    enrollment_year = IntField(default=2024)
    email = StringField(unique=True)
    hashed_password = StringField()
    role = StringField(default="student")
    absences = IntField(default=0)

class Subject(Document):
    name = StringField(required=True, unique=True)
    teacher_name = StringField()

class Grade(Document):
    student = ReferenceField(Student, reverse_delete_rule=CASCADE, required=True)
    subject = ReferenceField(Subject, reverse_delete_rule=CASCADE, required=True)
    score = FloatField(required=True)
    grade_type = StringField(choices=('exam', 'homework', 'lab'), required=True)
    date = DateTimeField(default=datetime.datetime.now)

class User(Document):
    email = StringField(required=True, unique=True)
    hashed_password = StringField(required=True)
    role = StringField(choices=('admin', 'teacher'), default='teacher')
```

auth.py

```
import jwt
import datetime
import random
import string
from passlib.context import CryptContext
```

```
    from fastapi import APIRouter, Depends,
HTTPException
    from fastapi.security import
OAuth2PasswordBearer,
OAuth2PasswordRequestForm
    from pydantic import BaseModel
    from models import User

    SECRET_KEY =
"diploma_super_secret_key_2026"
    ALGORITHM = "HS256"

    pwd_context = CryptContext(schemes=["bcrypt"],
deprecated="auto")

    oauth2_scheme =
OAuth2PasswordBearer(tokenUrl="api/auth/login")
    auth_router = APIRouter(prefix="/api/auth",
tags=["Auth"])

    class UserRegister(BaseModel):
        email: str
        password: str

    def get_password_hash(password: str):
        return pwd_context.hash(password)

    def generate_random_password(length=8):
        characters = string.ascii_letters + string.digits
        return ''.join(random.choice(characters) for i in
range(length))

    def generate_student_email():
        random_numbers =
''.join(random.choices(string.digits, k=8))
        return f"st{random_numbers}@duikt.edu.ua"

    @auth_router.post("/register")
    async def register(user_data: UserRegister):
        if User.objects(email=user_data.email).first():
            raise HTTPException(status_code=400,
detail="Цей email вже зареєстровано")
        hashed_pw =
get_password_hash(user_data.password)
        new_user = User(email=user_data.email,
hashed_password=hashed_pw)
        new_user.save()
        return {"message": "Користувача успішно
створено!"}

    @auth_router.post("/login")
    async def login(form_data:
OAuth2PasswordRequestForm = Depends()):
        user =
User.objects(email=form_data.username).first()
        if not user or not
pwd_context.verify(form_data.password,
user.hashed_password):
            raise HTTPException(status_code=400,
detail="Невірний email або пароль")
        expire = datetime.datetime.utcnow() +
datetime.timedelta(hours=2)
        token = jwt.encode({"sub": user.email, "exp":
expire}, SECRET_KEY, algorithm=ALGORITHM)
        return {"access_token": token, "token_type":
"bearer"}

    def get_current_user(token: str =
Depends(oauth2_scheme)):
        try:
            payload = jwt.decode(token, SECRET_KEY,
algorithms=[ALGORITHM])
            email = payload.get("sub")
            if email is None:
                raise HTTPException(status_code=401,
detail="Недійсний токен")
        except Exception:
            raise HTTPException(status_code=401,
detail="Недійсний або прострочений токен")

        user = User.objects(email=email).first()
        if not user:
            raise HTTPException(status_code=401,
detail="Користувача не знайдено")
```

```
    return user

def require_admin(current_user: User =
Depends(get_current_user)):
    if current_user.role != 'admin':
        raise HTTPException(status_code=403,
detail="Доступ заборонено! Тільки для деканату
(admin).")
    return current_user
```

create_admin.py

```
from mongoengine import connect
from models import User
from auth import get_password_hash

connect(host="mongodb+srv://sviaticrrocer_db_use
r:KhhdFOGFIq98QMIx@cluster0.ncqipo7.mongodb.n
et/student_analytics?retryWrites=true&w=majority")

def create_super_admin():
    email = "admin@gmail.com"
    password = "admin"

    print("Шукаємо адміністратора в базі (в
таблиці User)...")
    admin_user = User.objects(email=email).first()

    if admin_user:
        admin_user.update(

set__hashed_password=get_password_hash(password),
            set__role="admin"
        )
        print(f"Адміністратор {email} вже існував.
Права та пароль успішно відновлено!")
    else:
        User(
            email=email,

hashed_password=get_password_hash(password),
            role="admin"
    ).save()
        print(f"Супер-адміністратор {email}
успішно створений у таблиці User!")

if __name__ == "__main__":
    create_super_admin()
```

schemas.py

```
from pydantic import BaseModel
from typing import Optional
from datetime import datetime

class StudentCreate(BaseModel):
    first_name: str
    last_name: str
    group_code: str
    enrollment_year: int = 2024

class StudentUpdate(BaseModel):
    first_name: Optional[str] = None
    last_name: Optional[str] = None
    group_code: Optional[str] = None
    enrollment_year: Optional[int] = None
    absences: Optional[int] = None

class StudentResponse(BaseModel):
    id: str
    first_name: str
    last_name: str
    group_code: str
    enrollment_year: int
    email: str
    absences: int

class SubjectCreate(BaseModel):
    name: str
    teacher_name: Optional[str] = None

class SubjectResponse(BaseModel):
    id: str
    name: str
    teacher_name: Optional[str]

class GradeCreate(BaseModel):
```

```
    student_id: str
    subject_id: str
    score: float
    grade_type: str

class GradeResponse(BaseModel):
    id: str
    student_id: str
    subject_id: str
    score: float
    grade_type: str
    date: datetime
```

seed_db.py

```
import random
from faker import Faker
from mongoengine import connect
from models import Student, Subject, Grade
from auth import get_password_hash, generate_student_email, generate_random_password

connect(
host="mongodb+srv://sviaticrrocer_db_user:KhhdFOGFIq98QMIx@cluster0.ncqipo7.mongodb.net/student_analytics?retryWrites=true&w=majority"
)

fake = Faker("uk_UA")


def seed_database():
    print("Очищуємо стару базу...")
    Student.objects().delete()
    Subject.objects().delete()
    Grade.objects().delete()

    print("Створюємо предмети...")
    subject_names = [
        "Вища математика",
        "Програмування на Python",
        "Бази даних",
        "Веб-дизайн",
        "Мережеві технології",
    ]

    subjects = []
    for name in subject_names:
        subj = Subject(name=name, teacher_name=fake.name()).save()
        subjects.append(subj)

    print("Формуємо групи...")
    departments = ["ТІР", "ПД", "КНД", "КІД"]
    groups = [f"{dep}-4{i}" for dep in departments for i in range(1, 5)]

    total_students = random.randint(150, 200)
    print(f"Зараховуємо {total_students} студентів...")

    students = []
    for i in range(total_students):
        email = generate_student_email()
        raw_password = generate_random_password()
        hashed_pwd = get_password_hash(raw_password)

        student = Student(
            first_name=fake.first_name(),
            last_name=fake.last_name(),
            group_code=groups[i % len(groups)],
            enrollment_year=2024,
            email=email,
            hashed_password=hashed_pwd,
            role="student",
            absences=random.randint(0, 50)
        ).save()
        students.append(student)

    print("Генеруємо випадкові оцінки (Журнал)...")
    grade_types = ['exam', 'homework', 'lab']
    total_grades = 0
```

```
        for student in students:
            num_grades = random.randint(5, 15)
            for _ in range(num_grades):
                Grade(
                    student=student,
                    subject=random.choice(subjects),
                    score=round(random.uniform(40.0,
100.0), 1),

grade_type=random.choice(grade_types)
                ).save()
                total_grades += 1

        print(f"Базу наповнено: {total_students}
студентів та {total_grades} оцінок!")


    if __name__ == "__main__":
        seed_database()
```