

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ АКАДЕМІЧНИМИ
РЕЙТИНГАМИ ТА ДОСЯГНЕННЯМИ СТУДЕНТІВ»**

Виконав студент групи ПЗ-22-1
ШУЛЬГА Максим Ігорович

Керівник роботи:
БАКАЛЮК Тетяна Анатоліївна

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ШУЛЬГА Максим Ігорович

Керівник роботи: ВАКАЛЮК Тетяна Анатоліївна

Тема роботи: «Вебзастосунок для управління академічними рейтингами та досягненнями студентів», затверджена наказом університету від **«27» лютого 2026 р. № 92/с.**

Вихідні дані для роботи: предметом дослідження є вебтехнології та методи автоматизованого розрахунку й подання академічних рейтингів і досягнень студентів, а об'єктом є процеси обліку та оцінювання академічної успішності й досягнень студентів закладу вищої освіти.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	3.03.2026	3.03.2026
2	Вакалюк Т.А.	3.03.2026	3.03.2026
3	Вакалюк Т.А.	3.03.2026	3.03.2026

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра «Вебзастосунок для управління академічними рейтингами та досягненнями студентів» складається з 73 сторінок, 12 рисунків, 13 таблиць, 41 джерела та 2 додатків.

Об'єктом дослідження є процеси обліку та оцінювання академічної успішності й досягнень студентів закладу вищої освіти.

Предметом дослідження є вебтехнології та методи автоматизованого розрахунку й подання академічних рейтингів і досягнень студентів.

Мета роботи – проектування та розробка вебзастосунку для автоматизації обліку, розрахунку та відображення академічних рейтингів і досягнень студентів закладу вищої освіти.

Методи досліджень – методи системного аналізу для виявлення вимог; методи об'єктно-орієнтованого аналізу й проектування з використанням нотації UML; методи реляційного моделювання баз даних; методи структурного й

У роботі виконано аналіз предметної області та задач, що підлягають автоматизації; проведено порівняльний аналіз програмних аналогів – систем управління навчанням, університетських деканатських систем та електронних таблиць; обґрунтовано вибір клієнт-серверної архітектури з принципом REST та шаровою структурою серверної частини; обґрунтовано вибір технологічного стеку – мови програмування TypeScript, серверного середовища Node.js з фреймворком Express 5, ORM Prisma 7, реляційної СУБД PostgreSQL 16, клієнтського фреймворка React 18 із збиральником Vite 6, контейнеризації засобами Docker.

Спроектовано об'єктно-орієнтовану структуру системи з трьома акторами та визначено сценарії використання; розроблено структуру бази даних з 11 таблиць та зв'язками між ними; реалізовано алгоритм розрахунку академічного

					ІПЗ.КР.Б – 121 – 26 – ПЗ						
Змн.	Арк.	№ докум.		Дата							
Розроб.		Шульга М.І			Вебзастосунок для управління академічними рейтингами та досягненнями студентів			Літ.	Арк.	Аркушів	
Перевір.		Вакалюк Т.А.								1	70
Керівник								ФІКТ Гр. ІПЗ-22-1			
Зав. каф.											

рейтингу студента на основі середнього балу з масштабуванням до 5-бальної шкали GPA; реалізовано серверну частину з REST API на понад 40 ендпоінтів, підсистемою автентифікації на базі JWT та refresh-токенів; реалізовано клієнтську частину як односторінковий вебзастосунок з трьома різними панелями для трьох ролей користувачів.

Розроблений програмний комплекс AcademicRate може бути впроваджений у навчальний процес закладу вищої освіти як інструмент автоматизації обліку успішності та формування академічних рейтингів студентів.

Ключові слова: ВЕБЗАСТОСУНОК, АКАДЕМІЧНИЙ РЕЙТИНГ, СИСТЕМА УПРАВЛІННЯ НАВЧАННЯМ, TYPESCRIPT, NODE.JS, EXPRESS, REACT, POSTGRESQL, REST API, JWT, PRISMA ORM, DOCKER.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				1
Змн.	Арк.	№ докум.	Підпис	Дата		

ABSTRACT

Explanatory note to the Bachelor's qualification thesis: 73 pages, 12 figures, 13 tables, 41 references, 2 appendix.

The object of research is the processes of accounting, assessment, and evaluation of academic performance and achievements of students in a higher education institution.

The subject of research is methods and web technologies for automating the calculation and presentation of academic rankings and student achievements.

The purpose of the work is to design and develop a web application for automating the recording, calculation, and visualization of academic rankings and achievements of students in a higher education institution.

Research methods include system analysis for requirements identification; object-oriented analysis and design using UML notation; relational database modeling; and structural and functional software testing.

The thesis includes an analysis of the subject area and the tasks to be automated; a comparative analysis of existing software solutions, including learning management systems, university dean's office systems, and electronic spreadsheets; justification of the client-server architecture based on REST principles and a layered server-side structure; and justification of the selected technology stack: the TypeScript programming language, the Node.js server environment with the Express 5 framework, the Prisma 7 ORM, the PostgreSQL 16 relational database management system, the React 18 client framework with the Vite 6 build tool, and Docker containerization.

An object-oriented system structure with three actors was designed and use cases were defined. A database structure of 11 tables with their relationships was developed. An academic ranking calculation algorithm based on the student's average grade scaled to a 5-point GPA scale was implemented. The server side was developed as a REST API with more than 40 endpoints and an authentication subsystem based on JWT and refresh tokens. The client side was implemented as a single-page web application with three separate dashboards for three user roles.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				2
Змн.	Арк.	№ докум.	Підпис	Дата		

The developed software system AcademicRate can be integrated into the educational process of a higher education institution as a tool for automating academic performance management and generating student academic rankings.

Keywords: WEB APPLICATION, ACADEMIC RANKING, LEARNING MANAGEMENT SYSTEM, TYPESCRIPT, NODE.JS, EXPRESS, REACT, POSTGRESQL, REST API, JWT, PRISMA ORM, DOCKER.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				3
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	8
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	10
1.1 Актуальність теми та постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту	16
1.3 Вибір архітектури програмного продукту.....	20
1.4 Обґрунтування вибору інструментальних засобів.....	22
Висновки до першого розділу.....	27
РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕЙТИНГАМИ ТА ДОСЯГНЕННЯМИ СТУДЕНТІВ	28
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи (UML)	28
2.2 Розробка та проєктування бази даних системи.....	35
2.3 Проєктування та реалізація алгоритмів роботи системи	39
2.4 Реалізація вебзастосунку клієнтська та серверна частини.....	42
Висновки до другого розділу	53
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ	55
3.1 Порядок встановлення та налаштування параметрів системи.....	55
3.2 Структура інтерфейсу та порядок роботи з вебзастосунком.....	57
3.3 Тестування роботи програмного продукту.....	64
Висновки до третього розділу.....	68
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	72

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ	76
ДОДАТОК А.....	76
ДОДАТОК Б	83

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface – інтерфейс програмування застосунків
CRUD	Create, Read, Update, Delete – чотири базові операції з даними
DOM	Document Object Model – об’єктна модель документа
DTO	Data Transfer Object – об’єкт передачі даних між шарами застосунку
ECTS	European Credit Transfer and Accumulation System – Європейська кредитно-трансферна та накопичувальна система.
GPA	Grade Point Average – середньозважений бал успішності (0–5)
HTTP	Hypertext Transfer Protocol – протокол передачі гіпертексту
HTTPS	Hypertext Transfer Protocol Secure – захищений варіант протоколу HTTP
IDE	Integrated Development Environment – інтегроване середовище розробки
JSON	JavaScript Object Notation – формат обміну даними
JWT	JSON Web Token – стандарт відкритих токенів доступу
LMS	Learning Management System – система управління навчанням
ORM	Object-Relational Mapping – об’єктно-реляційне відображення
RBAC	Role-Based Access Control – контроль доступу на основі ролей
REST	Representational State Transfer – архітектурний стиль побудови API
SPA	Single Page Application – односторінковий вебзастосунок
SQL	Structured Query Language – мова структурованих запитів
UML	Unified Modeling Language – уніфікована мова моделювання
URL	Uniform Resource Locator – уніфікований локатор ресурсу

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

СУБД

Система управління базами даних

ЗВО

Заклад вищої освіти

ПЗ

Програмне забезпечення

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Цифровізація освітнього процесу є одним із пріоритетних напрямів розвитку закладів вищої освіти України. Нині облік успішності студентів, розрахунок академічних рейтингів та фіксація позанавчальних досягнень значною мірою здійснюються засобами, що не були призначені для цього: паперовими відомостями, розрізненими таблицями Microsoft Excel, локальними документами на комп'ютерах окремих працівників. Такий підхід має суттєві недоліки – високу трудомісткість ручного перенесення й опрацювання даних, ризик арифметичних помилок при розрахунку рейтингу, відсутність єдиного джерела достовірної інформації, непрозорість методики ранжування для самих студентів та складність формування зведених рейтингових списків за кількома групами одночасно.

Академічний рейтинг безпосередньо впливає на розподіл стипендій, надання місць у гуртожитку, можливості участі в програмах академічної мобільності та міжнародному обміні, тому коректність і прозорість його обчислення мають практичне значення для кожного учасника освітнього процесу. Окремою проблемою є облік досягнень, що не зводяться до оцінок у заліковій книжці: перемог в олімпіадах і конкурсах, наукових публікацій, отриманих сертифікатів, участі в проєктах. Ці дані, як правило, взагалі не систематизуються централізовано в більшості ЗВО.

Сучасні вебтехнології надають можливість побудувати централізовану систему, доступну через браузер для всіх категорій користувачів, з розмежуванням прав доступу, автоматичним розрахунком рейтингу за затвердженою методикою та наочним поданням результатів. Проте наявні рішення або є надлишково складними, зорієнтованими на повний цикл управління навчанням, або не враховують специфіку методики розрахунку рейтингу конкретного закладу освіти. Це обумовлює актуальність розробки спеціалізованого вебзастосунку.

Мета і завдання дослідження. Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку для автоматизації обліку, розрахунку

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

та відображення академічних рейтингів і досягнень студентів закладу вищої освіти.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- провести аналіз предметної області та визначити перелік процесів, що підлягають автоматизації;
- виконати порівняльний аналіз існуючих програмних аналогів та визначити їх функціональні обмеження;
- обґрунтувати вибір архітектури вебзастосунку, технологічного стеку та вимоги до апаратного забезпечення;
- розробити варіанти використання системи для різних категорій користувачів засобами UML;
- спроектувати структуру бази даних та об'єктну структуру програмного комплексу;
- спроектувати та реалізувати алгоритми роботи системи, зокрема алгоритм розрахунку академічного рейтингу;
- реалізувати серверну та клієнтську частини застосунку;
- розробити інструкцію встановлення, налаштування та порядку роботи з вебзастосунком;
- розробити та виконати тести для перевірки коректності роботи системи.

Об'єкт дослідження – процеси обліку та оцінювання академічної успішності й досягнень студентів закладу вищої освіти.

Предмет дослідження – вебтехнології та методи автоматизованого розрахунку й подання академічних рейтингів і досягнень студентів.

Методи дослідження. Під час виконання роботи використано методи системного аналізу для виявлення вимог до програмного продукту; методи об'єктно-орієнтованого аналізу й проектування з використанням нотації UML; методи реляційного моделювання баз даних; методи проектування програмного забезпечення на основі шарової архітектури; методи структурного й функціонального тестування програмного забезпечення.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми та постановка задачі

Цифровізація освітнього процесу є одним зі стратегічних напрямів розвитку системи вищої освіти України [32], закріплених у Законі України «Про вищу освіту» [1] та підзаконних нормативних актах щодо організації освітнього процесу [2]. Перехід закладів вищої освіти на кредитно-трансферну систему організації навчання (ECTS) [6] і пов'язана з нею необхідність систематичного кількісного оцінювання результатів навчання істотно підвищують вимоги до повноти, достовірності та оперативності опрацювання даних про академічну діяльність здобувачів. У цих умовах автоматизація обліку успішності та формування академічних рейтингів перетворюється з допоміжної на одну з базових інформаційних задач закладу освіти.

Сучасний заклад вищої освіти оперує значним обсягом інформації про академічну діяльність студентів. До цієї інформації належать оцінки, отримані за окремі види робіт у межах кожного навчального предмета, результати підсумкового контролю – заліків, диференційованих заліків, екзаменів, курсових робіт, а також позанавчальні досягнення – участь у наукових і творчих конкурсах, олімпіадах, конференціях, наявність публікацій, сертифікатів про підвищення кваліфікації, перемог у спортивних змаганнях. Цей масив даних має виражену реляційну природу: він описує сукупність взаємопов'язаних сутностей (студент, група, кафедра, предмет, викладач, оцінка) і потребує засобів, що забезпечують цілісність, несуперечливість і захищеність збережених відомостей.

Академічний рейтинг студента є інтегральним кількісним показником, що узагальнює результати навчальної діяльності за певний період і дає змогу впорядкувати (ранжувати) здобувачів за рівнем успішності. У вітчизняній практиці методику обчислення рейтингу та порядок його використання визначають внутрішні нормативні документи закладу освіти, зокрема положення про організацію освітнього процесу та порядок формування

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

рейтингу успішності для призначення академічних стипендій [5]. Таким чином, рейтинг є не лише довідковим показником, а й юридично значущою підставою для прийняття адміністративних рішень, що зумовлює підвищені вимоги до точності та прозорості його розрахунку.

На основі сукупності цих даних обчислюється академічний рейтинг студента, який використовується для розв'язання низки практичних завдань: ранжування студентів у межах академічної групи, потоку, курсу та факультету; розподіл стипендіальних виплат, у тому числі підвищених і соціальних стипендій; розподіл місць у гуртожитку; прийняття рішень щодо рекомендації студента до програм академічної мобільності, наукових стажувань і стипендіальних програм за межами ЗВО. З огляду на безпосередній вплив рейтингу на матеріальне забезпечення та освітні можливості здобувача похибки в його обчисленні або непрозорість методики ранжування здатні спричинити не лише організаційні, а й репутаційні та правові наслідки для закладу освіти.

У значній кількості закладів вищої освіти України дотепер ці процеси автоматизовано лише частково або не автоматизовано взагалі. Типовою практикою є ведення обліку оцінок викладачами у власних електронних таблицях формату Microsoft Excel або у паперових журналах. Зведення даних у єдиний рейтинговий список виконується кураторами груп та працівниками деканатів вручну, шляхом перенесення цифрових значень з різних джерел в одну таблицю та обчислення підсумкових показників за формулами Excel. Такий спосіб організації обліку фактично відтворює модель «острівної автоматизації», за якої окремі ділянки роботи автоматизовано ізолювано, без єдиного сховища даних та узгодженого програмного інтерфейсу між ними.

Такий підхід має низку очевидних недоліків. Передусім він характеризується високою трудомісткістю. На формування рейтингового списку для однієї групи зі складу 25–30 студентів за результатами семестру з 6–8 предметами працівник деканату витрачає від двох до чотирьох робочих годин. По-друге, велика частка ручної роботи призводить до високого ризику арифметичних помилок та помилок копіювання. По-третє, відсутність єдиного

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

централізованого джерела інформації призводить до неузгодженостей між різними версіями таблиць, що використовуються різними працівниками. По-четверте, для самих студентів методика обчислення рейтингу залишається непрозорою – вони не мають змоги в будь-який момент часу побачити власну позицію та зрозуміти, які саме оцінки на неї вплинули.

З погляду теорії інформаційних систем [10] перелічені недоліки є наслідком порушення фундаментальних принципів організації даних – відсутності єдиного джерела істини (single source of truth), браку механізмів контролю цілісності та обмеження доступу, а також неможливості відстеження історії змін. Дублювання даних у численних локальних копіях закономірно призводить до їх розсинхронізації, а відсутність рольового розмежування доступу – до ризику несанкціонованого або помилкового редагування. Окрім того, електронні таблиці погано масштабуються на обсяги даних усього закладу освіти та не передбачають одночасної багатокористувацької роботи з гарантуванням узгодженості результату.

Окремою проблемою є облік позанавчальних досягнень. Якщо оцінки в межах навчальних дисциплін формально зафіксовано в заліково-екзаменаційних відомостях, то відомості про участь у конкурсах, отримані сертифікати, опубліковані статті часто фіксуються лише в довідках, що зберігаються особисто у студента або у профільної кафедри. Систематизованої електронної бази таких досягнень у переважній більшості ЗВО не існує. Це утруднює формування об'єктивного загального уявлення про студента – зокрема при прийнятті рішень щодо рекомендації на конкретні стипендіальні програми чи інші заохочення.

Систематизований електронний облік позанавчальних досягнень фактично формує цифрове портфоліо здобувача, що набуває практичної цінності не лише в межах закладу освіти, а й під час працевлаштування, вступу на наступні рівні вищої освіти та участі в конкурсних відборах. Збереження підтверджувальних документів (скан-копій дипломів, сертифікатів, публікацій) у структурованому вигляді з можливістю модерації забезпечує верифікованість

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

таких відомостей, що принципово відрізняє їх від несистематизованих паперових довідок.

Розвиток вебтехнологій та підходів до обміну знаннями в мережі [41] створює передумови для побудови подібних систем. Сучасні вебтехнології [9] надають можливість усунути перелічені недоліки шляхом побудови централізованої інформаційної системи з багатокористувацьким доступом через браузер, розмежуванням прав на основі ролей, автоматичним розрахунком рейтингу за затвердженою методикою та наочним поданням результатів. Водночас наявні на ринку готові рішення або є надлишково складними та зорієнтованими на повний цикл управління навчанням, або не враховують специфіку методики розрахунку рейтингу конкретного закладу освіти. Сукупність зазначених чинників обумовлює актуальність розробки спеціалізованого вебзастосунку для управління академічними рейтингами та досягненнями студентів.

Аналіз перелічених проблем дає підстави сформулювати загальну задачу автоматизації як побудову централізованої інформаційної системи, що має забезпечити:

- єдине електронне сховище даних про академічну успішність та досягнення студентів усього закладу освіти;
- розмежований доступ до даних на основі трьох ролей користувачів – студент, викладач, адміністратор – з чіткими повноваженнями кожної ролі;
- автоматичне обчислення академічного рейтингу за єдиною методикою з можливістю порівняння студентів у межах групи, курсу, кафедри та закладу загалом;
- оперативне сповіщення студентів про нові оцінки та зміну їхньої рейтингової позиції;
- формування зведених звітів за різними категоріями (групами, кафедрами, періодами часу) у форматі, придатному для подальшого опрацювання, з дотриманням вимог законодавства про інформацію [3];

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

– ведення обліку позанавчальних досягнень студентів зі збереженням підтверджувальних документів.

Виходячи зі сформульованих вимог, поставлено задачу проєктування та розробки вебзастосунку з робочою назвою AcademicRate. Вибір вебтехнологій обґрунтовано тим [8] , що вони не висувають вимог до встановлення додаткового програмного забезпечення на пристроях користувачів, забезпечують одночасний доступ до системи з будь-якого пристрою, обладнаного сучасним браузером, та дають змогу централізовано оновлювати застосунок без втручання у клієнтське обладнання. На відміну від настільних застосунків, що потребують інсталяції та оновлення на кожному робочому місці, вебзастосунок встановлюється й супроводжується централізовано на сервері, що суттєво знижує сукупну вартість володіння системою та спрощує її адміністрування.

Систему має бути реалізовано з підтримкою трьох ролей користувачів. Студент має право переглядати власні оцінки, особистий рейтинг, позицію в рейтингу групи, потоку та кафедри, а також додавати дані про власні досягнення з можливістю їх подальшого підтвердження адміністратором. Викладач має право переглядати склад груп, для яких він викладає закріплені за ним предмети, та виставляти оцінки студентам за результатами виконаних робіт. Адміністратор виконує організаційні дії – створює та редагує облікові записи користувачів, академічні групи, кафедри, перелік предметів, призначає викладачам предмети та групи, формує системні звіти, контролює коректність даних. Така трирівнева рольова модель відповідає принципу контролю доступу на основі ролей (RBAC) і забезпечує надання кожному користувачеві мінімально необхідного для виконання його функцій обсягу повноважень.

Сформульовано такі функціональні вимоги до програмного продукту:

- автентифікація користувачів засобами електронної пошти та пароля з використанням токенів JWT;

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

- автоматичний розрахунок академічного рейтингу за визначеною методикою у вигляді середнього зваженого балу з масштабуванням до 5-бальної шкали GPA;
- ведення довідників академічних груп, кафедр, навчальних предметів та користувачів з підтримкою стандартних операцій CRUD;
- підсистема виставлення оцінок з вибором типу роботи (лекційна, лабораторна, контрольна, екзаменаційна, додаткова);
- підсистема обліку позанавчальних досягнень студентів з можливістю прикріплення скан-копій підтверджувальних документів;
- формування рейтингових списків для груп, кафедр та закладу загалом з можливістю фільтрації та експорту даних у формат, придатний для подальшого опрацювання;
- аналітична панель адміністратора з графічним поданням статистики системи;
- підтримка адаптивного інтерфейсу для роботи на пристроях різних розмірів.

До нефункціональних вимог до системи віднесено такі: час відповіді сервера на типові запити не повинен перевищувати 500 мс; одночасну роботу в системі мають здатні підтримати не менше 200 користувачів без помітного зниження швидкодії; інтерфейс має відповідати загальноприйнятим практикам ергономіки та доступності; система має бути захищена від типових вебатак (SQL-ін'єкцій, XSS, CSRF, brute force); персональні дані користувачів мають бути захищені відповідно до вимог Закону України «Про захист персональних даних» від 01.06.2010 № 2297-VI; розгортання системи має бути здійснене за допомогою стандартних інструментів контейнеризації для забезпечення відтворюваності середовища.

Перелічені функціональні та нефункціональні вимоги становлять основу для подальшого порівняльного аналізу аналогів, вибору архітектури та технологій реалізації, що розглядаються в наступних підрозділах цього розділу. Сформульована постановка задачі дає змогу класифікувати розроблюваний

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

продукт як спеціалізовану предметно-орієнтовану інформаційну систему класу управління даними академічної успішності, що поєднує функції обліку, обчислення похідних показників та подання інформації різним категоріям користувачів.

1.2. Аналіз аналогів програмного продукту

Перед прийняттям остаточного рішення про розробку власного програмного продукту необхідно виконати аналіз наявних на ринку рішень, що мають подібну функціональність. Це дає змогу уникнути дублювання вже реалізованих ефективних рішень та виявити функціональні прогалини, які потребують власної розробки. Аналіз виконано за трьома основними категоріями програмних продуктів: системи управління навчанням загального призначення, університетські деканатські інформаційні системи та електронні таблиці.

До категорії LMS належать спеціалізовані програмні платформи, призначені для організації навчального процесу в електронному середовищі. Найвідомішими представниками цього класу є Moodle, Canvas, Blackboard Learn, Google Classroom. Розглянемо їхні особливості з точки зору поставленої задачі.

Платформа Moodle є найпоширенішою LMS у вітчизняних закладах вищої освіти. Це відкрите програмне забезпечення з можливістю самостійного розгортання, що знімає питання вартості ліцензій [33]. Moodle забезпечує створення електронних курсів, проведення тестувань, ведення журналу оцінок з кожного курсу. Проте з точки зору задач, поставлених у роботі, Moodle має суттєві обмеження: фокус системи зміщено на дистанційне навчання та автоматизацію тестування, тоді як облік підсумкових оцінок з усіх предметів та обчислення зведеного академічного рейтингу для цілей розподілу стипендій не є основним призначенням системи. Журнал оцінок ведеться в межах кожного курсу окремо, а формування рейтингового списку зі складу студентів кількох груп вимагає або додаткового програмування, або експорту даних і подальшого ручного опрацювання.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Canvas та Blackboard Learn є комерційними рішеннями з подібною до Moodle функціональністю. Висока вартість ліцензій робить їх непридатними для масового впровадження в українських ЗВО. Google Classroom, навпаки, є безкоштовним сервісом, але має значно обмеженіші функціональні можливості, ніж Moodle, і фактично не підходить для централізованого ведення академічного обліку всього закладу.

Сучасні дослідження демонструють потенціал аналітики навчальних даних для прогнозування успішності студентів [35], однак реалізація такої аналітики не є типовою функцією наявних LMS. Загальний висновок щодо систем LMS – вони побудовані навколо інших задач (організація навчального процесу, проведення занять, тестування), а функції обліку рейтингу й позанавчальних досягнень є для них вторинними або відсутніми.

До цієї категорії належать спеціалізовані інформаційні системи, призначені для автоматизації роботи деканатів та інших підрозділів управління навчальним процесом ЗВО. Історично такі системи реалізовувалися як настільні застосунки на технологіях 2000-х років (Delphi, Visual Basic, FoxPro), що ускладнювало доступ до них з різних пристроїв та для різних категорій користувачів. Сучасним представником цього класу є хмарна платформа Digital University, що впроваджена в низці українських закладів вищої освіти (зокрема в Національному авіаційному університеті та Національному аерокосмічному університеті «ХАІ») і реалізує концепцію єдиного електронного середовища закладу освіти [34].

Платформа Digital University надає комплекс функцій управління освітнім процесом: персональний електронний кабінет здобувача та викладача, перегляд складу академічної групи й розкладу занять, ведення електронних журналів оцінок, доступ до навчальних та організаційних документів, інтеграцію з університетською електронною поштою як єдиним засобом автентифікації й комунікації. На відміну від застарілих настільних деканатських систем, Digital University є вебзастосунком із сучасним адаптивним інтерфейсом, доступним з

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

будь-якого пристрою, що забезпечує одночасну роботу студентів, викладачів та адміністрації.

Водночас, з огляду на поставлені в роботі завдання, дана платформа має низку суттєвих обмежень. Вона є комплексним пропрієтарним програмним рішенням, тобто програмним забезпеченням із закритим вихідним кодом, права на використання, модифікацію та поширення якого належать розробнику або компанії-власнику. Платформа охоплює повний цикл управління освітнім процесом, включаючи адміністрування навчання, комунікацію, оцінювання та аналітику. Проте для розв'язання вузькоспеціалізованої задачі обліку успішності здобувачів освіти і формування академічного рейтингу такий функціонал є надлишковим, що ускладнює адаптацію системи до конкретних потреб закладу освіти та може призводити до неефективного використання ресурсів. Логіка обчислення рейтингу в таких платформах відповідає вбудованим у продукт алгоритмам, а адаптація під специфічну методику ранжування конкретного закладу освіти обмежена й залежить від постачальника. Структурований облік позанавчальних досягнень студентів з можливістю самостійного подання та модерації підтверджувальних документів не є профільною функцією таких систем.

Окремою особливістю готових університетських платформ є модель їх постачання: вони впроваджуються та супроводжуються централізовано постачальником на умовах передплати, що унеможливлює самостійне розгортання, модифікацію вихідного коду або безкоштовну адаптацію під потреби окремої кафедри. Таким чином, попри сучасний інтерфейс і веборієнтованість, такі рішення не задовольняють вимог щодо гнучкої адаптації методики розрахунку рейтингу та автономного впровадження.

Найбільш поширеним на практиці інструментом обліку успішності та обчислення рейтингу є електронні таблиці Microsoft Excel та Google Sheets. Це універсальний інструмент з широкими функціональними можливостями: введення даних у табличній формі, обчислення формул, побудова графіків, фільтрація й сортування. Перевагою цього підходу є нульова вартість

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

впровадження за наявності у працівників ліцензії на Microsoft Office (або безкоштовного Google Sheets) та відсутність необхідності в адаптації під методику закладу.

Однак ці переваги нівелюються низкою принципових недоліків. Кожен працівник веде власні таблиці локально, що призводить до розсинхронізації даних між версіями. Студенти доступу до таблиць не мають взагалі. Обчислення рейтингу засобами формул Excel вимагає високої кваліфікації працівника, а в разі помилок у формулах їх знайти доволі складно. Жодних механізмів захисту цілісності даних – будь-який працівник, що має доступ до файла, може випадково змінити чи видалити будь-які значення. Обсяг обчислюваних даних обмежений можливостями однієї робочої книги, що ускладнює одночасну роботу з даними кількох академічних років.

Підсумкове порівняння розглянутих категорій програмних продуктів за основними критеріями подано в табл. 1.1.

Таблиця 1.1

Порівняльний аналіз аналогів програмного продукту

Критерій	LMS (Moodle)	Деканатські	Excel	AcademicRate
Вебдоступ	Так	Часткова	Ні	Так
Доступ для студента	Обмежений	Рідко	Ні	Повний
Автообчислення рейтингу	Ні	Так	Часткове	Так
Облік позанавчальних досягнень	Ні	Ні	Ручний	Так
Розмежування ролей	Так	Так	Ні	Так
Адаптація під методику ЗВО	Складна	Платна	Просто	Так
Вартість впровадження	Низька	Висока	Нульова	Низька
Сучасний інтерфейс	Задовільний	Застарілий	Не застос.	Так

LMS-системи зорієнтовано на інші задачі, сучасні університетські платформи на кшталт Digital University є комплексними закритими рішеннями, надлишковими для вузької задачі та обмежено адаптовними під методику конкретного ЗВО; електронні таблиці не мають жодних механізмів централізації та розмежування ролей. Це обґрунтовує доцільність розробки власного спеціалізованого вебзастосунку, що поєднає сильні сторони перелічених категорій – централізованість, веборієнтованість, розмежування ролей, автоматичне обчислення рейтингу, облік позанавчальних досягнень – у єдиному продукті з можливістю автономного впровадження.

1.3 Вибір архітектури програмного продукту

На основі результатів аналізу аналогів сформульовано вимогу до архітектури системи: програмний продукт має бути вебзастосунком з підтримкою одночасної роботи багатьох користувачів через мережу. Це безпосередньо обумовлює вибір клієнт-серверної архітектури як базової архітектурної моделі системи.

Клієнт-серверна архітектура передбачає поділ програмного комплексу на дві логічно й фізично відокремлені частини: серверну, що зберігає дані та реалізує бізнес-логіку обробки запитів, і клієнтську, що забезпечує користувацький інтерфейс та надсилає запити до сервера. Між клієнтом і сервером встановлюється канал обміну даними, побудований на стандартних мережевих протоколах.

Для забезпечення взаємодії клієнта і сервера у вебзастосунку використано архітектурний стиль REST (Representational State Transfer), сформульований Роєм Філдінгом у 2000 році [17]. Цей стиль ґрунтується на таких принципах: відсутність стану на стороні сервера між запитами; чітке розмежування клієнта і сервера; кешованість відповідей; уніфікований інтерфейс; шарова організація; принцип «код за вимогою» (опціонально).

REST-API будується на стандартних методах протоколу HTTP: GET для запитів читання, POST для створення нових ресурсів, PUT і PATCH для оновлення, DELETE для видалення. Кожен ресурс ідентифікується унікальним

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

URI. Дані передаються переважно у форматі JSON, що є природним для веборієнтованих застосунків та легко обробляється як на стороні сервера, так і на стороні клієнта.

Для забезпечення гнучкості, тестопридатності та підтримуваності серверного коду його організовано за принципом шарової архітектури [18]. Порівняльні дослідження серверних фреймворків підтверджують доцільність такої організації коду [37]. Кожен шар має чітко визначену зону відповідальності та взаємодіє лише із сусідніми шарами через визначені інтерфейси. У серверній частині застосунку AcademicRate виділено такі шари:

- шар маршрутизації приймає вхідні HTTP-запити, виконує первинну валідацію формату даних та передає управління відповідному обробнику;
- шар контролерів реалізує конкретну логіку обробки запиту: викликає необхідні сервіси, формує відповідь у форматі JSON, встановлює HTTP-статуси;
- шар сервісів містить бізнес-логіку застосунку: розрахунок рейтингу, перевірки прав доступу, виконання операцій з кількома сутностями в межах транзакцій;
- шар доступу до даних реалізує операції з базою даних через ORM Prisma, інкапсулює специфіку конкретної СУБД.

Перевагою такої організації є можливість заміни реалізації окремого шару без необхідності переписувати всі решта. Наприклад, заміна СУБД PostgreSQL на іншу зачіпає лише шар доступу до даних. Заміна формату даних з JSON на XML – лише шар маршрутизації. Логіка обчислення рейтингу, реалізована в шарі сервісів, не залежить від цих змін.

Клієнтська частина вебзастосунку AcademicRate реалізована як односторінковий застосунок (SPA – Single Page Application). На відміну від традиційного підходу, коли при кожному переході користувача на нову сторінку браузер виконує повне перезавантаження HTML, у SPA завантаження HTML-каркаса виконується один раз, а далі весь зміст сторінки оновлюється засобами JavaScript на основі даних, отриманих від сервера через REST API.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

Переваги такого підходу – швидший відгук інтерфейсу на дії користувача (бо не потрібно перезавантажувати всю сторінку), менший обсяг даних, що передаються по мережі при переході між «сторінками», та можливість будувати багатий інтерактивний інтерфейс зі складною логікою на стороні клієнта. Недолік – більш складна реалізація початкового завантаження та необхідність застосування спеціальних інструментів для оптимізації роботи з пошуковими системами.

Структура клієнтського SPA побудована за компонентним підходом: інтерфейс розділено на ієрархію React-компонентів, кожен з яких відповідає за обмежену частину функціональності та має чітко визначений API взаємодії з батьківськими компонентами. Це забезпечує перевикористовуваність компонентів та простоту їх тестування.

Маршрутизація в межах SPA реалізована засобами бібліотеки react-router-dom, що дає змогу відображати в адресному рядку браузера логічний URL поточної сторінки попри відсутність фактичних перезавантажень. Це забезпечує очікувану поведінку кнопок навігації браузера (назад/вперед) та можливість обміну посиланнями на конкретні сторінки застосунку.

Між клієнтом і сервером взаємодія виконується засобами вбудованого в браузер інтерфейсу Fetch API [31], обгорнутого в утилітарний модуль для додавання токенів автентифікації, обробки помилок та автоматичного оновлення access-токена через refresh-токен у разі його закінчення.

1.4 Обґрунтування вибору інструментальних засобів

Вибір технологічного стеку для реалізації вебзастосунку є одним з найважливіших архітектурних рішень, що визначає не лише швидкість розробки, але й якість, продуктивність, безпеку та подальшу підтримуваність системи. У цьому підрозділі розглянуто обґрунтування вибору кожної складової стеку.

Основною мовою програмування як для серверної, так і для клієнтської частини обрано TypeScript [19] – статично типізовану надбудову над мовою JavaScript, розроблену компанією Microsoft з 2012 року. На відміну від

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

динамічно типізованого JavaScript, TypeScript вимагає вказівки типів змінних, параметрів функцій та значень, що повертаються; невідповідності типів виявляються на етапі компіляції, а не під час виконання.

Перевагами використання TypeScript у поточному проєкті є такі: виявлення значної частини потенційних помилок на етапі компіляції, що знижує ризик помилок у вже розгорнутій системі; самодокументування коду через явні оголошення типів; підтримка автодоповнення та рефакторингу в IDE; можливість використовувати одну й ту саму мову та один технологічний стек на серверній і клієнтській частинах. Останнє особливо важливо для проєкту, що виконується одним розробником: знижується когнітивне навантаження від перемикання між мовами та полегшується винесення спільних типів даних у спільні модулі.

У проєкті використано TypeScript версії 5.4, що містить усі сучасні можливості мови, включно з шаблонами, generic-типами, типами-помічниками, що залежать від рядкових літералів, та строгим режимом перевірки null/undefined значень. Конфігурація компілятора tsconfig.json налаштована в строгий режим (strict: true) для максимально жорсткої перевірки типів.

Як серверне середовище виконання обрано Node.js — платформу для запуску JavaScript/TypeScript-коду на стороні сервера, побудовану на високопродуктивному рушії V8 від Google Chrome. Node.js характеризується подієво-керованою однопотоковою архітектурою з неблокуючими операціями введення-виведення, що особливо ефективно для вебзастосунків з великою кількістю одночасних мережових з'єднань; порівняльні дослідження продуктивності серверних середовищ підтверджують конкурентоспроможність Node.js за цим показником [36].

На рівні Node.js [20] реалізовано лише низькорівневий HTTP-сервер. Для побудови вебзастосунку на цій основі використовується фреймворк Express [21] — найпоширеніший серверний фреймворк екосистеми Node.js. У проєкті обрано Express 5 — найновіша версія, що містить нативну підтримку

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

async/await у проміжному програмному забезпеченні (middleware), оновлену підсистему маршрутизації та поліпшення безпеки.

Express забезпечує концепцію проміжного програмного забезпечення (middleware), що дає змогу будувати ланцюжок обробників, через які послідовно проходить кожен HTTP-запит. Це зручний механізм для впровадження таких загальних функцій, як логування запитів, обробка помилок, перевірка автентифікації, обмеження частоти запитів, валідація вхідних даних. У проєкті AcademicRate реалізовано власне middleware для перевірки JWT-токенів та контролю ролей користувачів.

Для роботи з базою даних обрано ORM Prisma [22] – сучасну систему об’єктно-реляційного відображення для TypeScript та JavaScript. На відміну від традиційних ORM (TypeORM, Sequelize), Prisma використовує власну мову опису схеми бази (Prisma Schema Language), на основі якої автоматично генерує типобезпечний клієнт доступу до даних.

Переваги Prisma щодо традиційних ORM: повна типобезпека всіх запитів на рівні TypeScript-компілятора, що виключає помилки невідповідності типів полів моделей і параметрів запитів; декларативний опис схеми бази в одному файлі; автоматична генерація міграцій бази даних на основі змін схеми; підтримка кількох СУБД з єдиним API; вбудована підтримка транзакцій та пакетних операцій.

У проєкті використано Prisma версії 7. Файл schema.prisma містить опис усіх 11 моделей даних, їх зв’язків та індексів. При зміні цього файлу команда prisma migrate dev автоматично формує SQL-міграцію, описує її дельтою змін до попереднього стану та накатує її на базу даних.

Як систему управління базами даних обрано PostgreSQL – об’єктно-реляційну СУБД з відкритим вихідним кодом, що розвивається з 1986 року та станом на 2026 рік є одним з найпопулярніших реляційних рішень для серверних задач. PostgreSQL повністю підтримує стандарт ANSI SQL та містить значну кількість розширень, що виходять за межі стандарту.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Серед особливостей PostgreSQL, важливих для поточного проєкту, варто відзначити: повну підтримку ACID-транзакцій з різними рівнями ізоляції; підтримку численних типів даних, включно з JSON та JSONB для напівструктурованих даних; багатий механізм індексів (B-tree, hash, GIN, GiST, BRIN); підтримку віконних функцій (window functions), необхідних для розрахунку рейтингу студентів у межах груп; підтримку матеріалізованих представлень; засоби розширення засобами власних функцій мовами SQL, PL/pgSQL, PL/Python.

У проєкті використано PostgreSQL версії 16. Для розрахунку позицій студентів у рейтингових списках застосовується віконна функція DENSE_RANK(), що ефективно обчислюється засобами індексів навіть для значних обсягів даних.

Для побудови клієнтської частини як SPA обрано React [24] – бібліотеку для побудови користувацьких інтерфейсів від Meta. React організовано навколо концепції компонентів – функцій або класів, що повертають опис UI у формі віртуального DOM. Бібліотека самостійно ефективно синхронізує віртуальний DOM з реальним DOM браузера, оновлюючи лише ті елементи, що змінилися.

У проєкті використано React версії 18, який містить такі вдосконалення: автоматичне групування оновлень стану (batching); конкурентний режим відображення для пріоритетизації важливих оновлень; нові хуки для розв’язання поширених типів задач (useId, useTransition, useDeferredValue).

Як збиральник проєкту обрано Vite [25] – сучасний інструмент розробки фронтенда, побудований на основі ES-модулів та esbuild. На відміну від традиційних збиральників (Webpack, Parcel), Vite забезпечує миттєвий старт dev-сервера та надзвичайно швидке гаряче перезавантаження модулів (HMR – Hot Module Replacement) навіть на значних за обсягом проєктах. Виробничу збірку Vite виконує засобами Rollup, що дає змогу отримувати оптимізовані бандли мінімального розміру.

Для оформлення інтерфейсу використано бібліотеку компонентів Material-UI (MUI) версії 7 – реалізацію Material Design від Google для React.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

MUI надає готові компоненти (форми, таблиці, діалоги, кнопки), що дотримуються єдиного стилю та забезпечують підтримку темної теми, адаптивності та доступності з мінімумом зусиль з боку розробника. Для побудови графіків аналітики використано бібліотеку Recharts [27], що інтегрується з React за компонентним принципом і дає змогу декларативно описувати лінійні, стовпчастові й кругові діаграми.

Для забезпечення відтворюваності середовища розгортання та спрощення інсталяції в проєкті використано засоби контейнеризації Docker та оркестратор Docker Compose [28]. Контейнеризація передбачає упаковку застосунку разом з усіма його залежностями в незалежну від хост-системи одиницю – контейнер.

Перевагою такого підходу є те, що для розгортання системи на новому сервері достатньо встановити Docker та виконати команду `docker compose up` – усі необхідні залежності (Node.js, PostgreSQL, Nginx) автоматично завантажуються й налаштовуються в потрібних версіях. Це усуває класичну проблему «у мене працює, а на сервері – ні» через відмінності у версіях бібліотек чи конфігурації операційної системи.

Підсумковий перелік використаних у проєкті інструментальних засобів та технологій із зазначенням їхньої ролі наведено в табл. 1.2.

Таблиця 1.2

Технологічний стек проєкту AcademicRate

Технологія	Версія	Категорія	Призначення
TypeScript	5.4	Мова програмування	Основна мова розробки сервера й клієнта
Node.js	20 LTS	Серверне середовище	Виконання серверного коду
Express	5.0	Веб-фреймворк	Побудова REST API сервера
Prisma	7.0	ORM	Типобезпечний доступ до бази даних
PostgreSQL	16	СУБД	Реляційне сховище даних застосунку
React	18	Клієнтський фреймворк	Побудова інтерфейсу як SPA
Vite	6	Збиральник	Розробка та збірка клієнта

MUI (Material-UI)	7	UI-бібліотека	Готові компоненти Material Design
Recharts	2.x	Бібліотека графіків	Аналітичні діаграми
react-router-dom	6.x	Маршрутизація	Клієнтська маршрутизація SPA
jsonwebtoken	9.x	Автентифікація	Робота з JWT-токенами
bcryptjs	2.x	Криптографія	Хешування паролів користувачів
Vitest	1.x	Тестування	Модульні тести
Docker	24+	Контейнеризація	Упаковка та запуск сервісів
Nginx	1.25	Веб-сервер	Роздача статички, реверс-проксі

Висновки до першого розділу

У першому розділі виконано аналіз предметної області – автоматизації обліку академічної успішності й позанавчальних досягнень студентів закладу вищої освіти. Встановлено, що поточний стан автоматизації цих процесів у переважній більшості вітчизняних ЗВО є недостатнім: облік ведеться засобами електронних таблиць, доступ студентів до власних рейтингів обмежений або відсутній, систематизованої бази позанавчальних досягнень не існує.

На основі аналізу сформульовано перелік задач, що підлягають автоматизації, та функціональні й нефункціональні вимоги до системи. Виконано порівняльний аналіз трьох категорій програмних аналогів – LMS-систем, університетських деканатських систем та електронних таблиць; показано, що жодна з категорій не задовольняє всім сформульованим вимогам, що обґрунтовує доцільність розробки власного спеціалізованого вебзастосунку.

Обґрунтовано вибір клієнт-серверної архітектури з REST API та шаровою організацією серверної частини. Для клієнтської частини обрано підхід односторінкового застосунку (SPA) як такий, що забезпечує найкращий відгук інтерфейсу при роботі через мережу. Обґрунтовано вибір технологічного стеку: TypeScript як основна мова програмування; Node.js та Express як серверне середовище; Prisma як ORM; PostgreSQL як реляційну СУБД; React та Vite як клієнтський фреймворк і збиральник; Docker як засіб контейнеризації. Сформульовано вимоги до апаратного забезпечення серверних і клієнтських пристроїв.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕЙТИНГАМИ ТА ДОСЯГНЕННЯМИ СТУДЕНТІВ

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

Початковим етапом проєктування інформаційної системи є виявлення дійових осіб (акторів) системи та визначення сценаріїв їх взаємодії з системою – варіантів використання (use cases). Для цього застосовано засоби уніфікованої мови моделювання UML (Unified Modeling Language) відповідно до стандарту ISO/IEC 19505 [16]. Цей крок дає змогу формалізувати функціональні вимоги, виявлені у підрозділі 1.1, у структурованому вигляді, що становить основу для подальшої реалізації.

Перед детальним розглядом сценаріїв взаємодії доцільно навести узагальнену архітектурну схему системи, що обґрунтована в підрозділі 1.3. Схема відображає поділ програмного комплексу на клієнтську частину (SPA на React), серверну частину (REST API на Node.js з шаровою організацією) та реляційну базу даних PostgreSQL, а також напрями взаємодії між цими складовими. Узагальнену архітектурну схему системи AcademicRate наведено на рис. 2.1.

На рис. 2.1 відображено трирівневу клієнт-серверну організацію системи та потоки даних між її складовими. У лівій частині схеми показано трьох дійових осіб – студента, викладача та адміністратора, – які взаємодіють із системою виключно через вебоглядач. Оглядач звертається до системи захищеним з'єднанням за протоколом HTTPS, завдяки чому весь трафік між пристроєм користувача та сервером шифрується на транспортному рівні.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

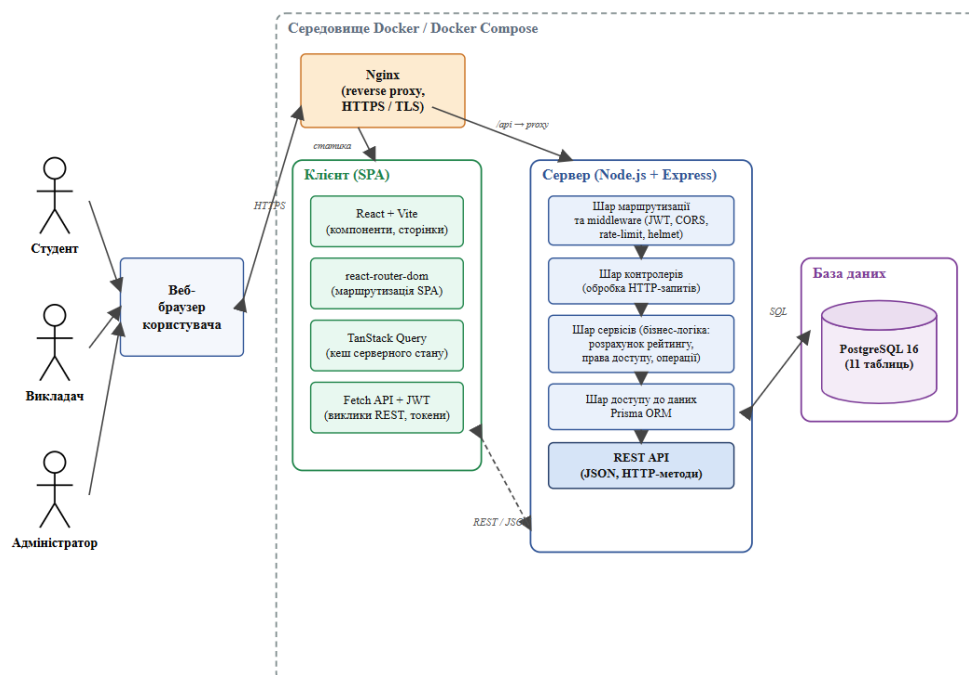


Рис. 2.1. Узагальнена архітектурна схема системи AcademicRate

Центральним елементом схеми є середовище контейнеризації Docker / Docker Compose (зображене зовнішнім штриховим прямокутником), у межах якого розгорнуто всі серверні складові системи як окремі контейнери. Точкою входу для всіх зовнішніх запитів є зворотний проксі-сервер Nginx, що завершує захищене з'єднання (HTTPS / TLS) та маршрутизує запити: статичні ресурси клієнтського застосунку він віддає безпосередньо, а запити до програмного інтерфейсу (за префіксом /api) перенаправляє (проксує) на серверний застосунок. Таке рішення приховує внутрішню топологію системи від зовнішньої мережі та дає змогу обслуговувати клієнт і сервер за єдиним доменним іменем.

Клієнтську частину (на схемі – блок «Клієнт (SPA)») реалізовано як односторінковий застосунок на основі React і збиральника Vite. У її складі виокремлено компоненти та сторінки інтерфейсу, підсистему клієнтської маршрутизації react-router-dom, бібліотеку керування серверним станом TanStack Query (що відповідає за кешування отриманих від сервера даних) та шар взаємодії з API на основі Fetch API, який додає до запитів токени JWT.

Стрілка з підписом «REST / JSON» між клієнтом і сервером відображає те, що обмін даними відбувається у форматі JSON через REST-інтерфейс.

Серверну частину (блок «Сервер (Node.js + Express)») організовано за шаровим принципом, що візуально передано вертикальною послідовністю шарів зі стрілками згори донизу: шар маршрутизації та проміжного програмного забезпечення (автентифікація за JWT, політика CORS, обмеження частоти запитів, заголовки безпеки helmet) приймає запит і передає його шару контролерів; шар контролерів опрацьовує HTTP-запит і делегує виконання шару сервісів, що містить бізнес-логіку (зокрема розрахунок рейтингу та перевірку прав доступу); шар сервісів звертається до шару доступу до даних, реалізованого засобами ORM Prisma. Нижній блок «REST API (JSON, HTTP-методи)» підкреслює, що зовнішнім контрактом сервера є саме REST-інтерфейс.

Праву частину схеми займає база даних – реляційна СУБД PostgreSQL 16, що містить 11 таблиць предметної області (підрозділ 2.2). Взаємодія серверного застосунку з базою даних позначена стрілкою з підписом «SQL»: шар доступу до даних Prisma формує параметризовані SQL-запити та повертає сервісам типізовані об'єкти. Таким чином, схема унаочнює ключовий архітектурний принцип системи – однонаправлений рух запиту від користувача через проксі-сервер і шари серверного застосунку до бази даних із зворотним поверненням відповіді тим самим шляхом, за якого кожен рівень виконує строго визначену функцію.

Сформульована задача автоматизації передбачає взаємодію з системою трьох категорій користувачів, кожна з яких має чітко визначені повноваження та інтереси. Виділено такі дійові особи системи:

– Студент – основний споживач інформації в системі; має право переглядати власні оцінки, академічний рейтинг та позицію в рейтингу групи, додавати дані про власні позанавчальні досягнення з прикріпленням підтверджувальних документів;

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

– Викладач – користувач системи, що виставляє оцінки студентам у межах закріплених за ним предметів; має право переглядати склад груп, виставляти, редагувати оцінки та формувати звіти;

– Адміністратор – користувач, що виконує організаційні дії в системі: ведення довідників (кафедр, груп, предметів, користувачів), призначення предметів викладачам, підтвердження позанавчальних досягнень студентів, формування системних звітів та переглядання аналітичних показників.

Анонімний користувач у системі не передбачений – будь-який доступ до даних вимагає попередньої автентифікації. Взаємодію трьох акторів з основними групами сценаріїв системи у графічному вигляді подано на діаграмі варіантів використання (рис. 2.2).



Рис. 2.2. Діаграма варіантів використання системи AcademicRate

На рис. 2.2 наведено діаграму варіантів використання, побудовану засобами нотації UML. Прямокутник з підписом «Вебзастосунок AcademicRate» позначає межу системи: усе, що розміщено всередині нього, є

функціями системи, а постаті ліворуч — це актори, тобто зовнішні стосовно системи дійові особи. Кольором умовно розрізнено зв'язки кожного актора з доступними йому варіантами використання, що полегшує сприйняття розмежування повноважень.

Актор «Студент» пов'язаний із варіантами використання перегляду власного профілю, перегляду рейтингу, перегляду оцінок та перегляду досягнень — тобто має доступ лише до перегляду власної академічної інформації. Актор «Викладач» пов'язаний з варіантами додавання оцінок, редагування оцінок, перегляду групи та перегляду звітів, що відповідає його ролі у процесі оцінювання. Актор «Адміністратор» має найширший набір повноважень — управління користувачами, управління групами, управління предметами, підтвердження досягнень та налаштування системи, що відповідає його організаційній ролі. Такий розподіл наочно ілюструє реалізовану в системі модель розмежування доступу на основі ролей (RBAC).

Окремо на діаграмі винесено службові варіанти використання «Вхід у систему» та «Вихід із системи», спільні для всіх акторів. Пунктирні стрілки з відношенням «include» поєднують функціональні варіанти використання зі службовим варіантом завершення сеансу, відображаючи те, що кожен сценарій роботи виконується в межах автентифікованого сеансу: доступ до будь-якої функції можливий лише після входу в систему («Вхід у систему»), а коректне завершення роботи передбачає вихід із неї («Вихід із системи»). Застосування відношення «include» дає змогу винести спільну для всіх сценаріїв поведінку в окремі багаторазово використовувані варіанти використання й уникнути її дублювання на діаграмі.

У сукупності діаграма охоплює чотирнадцять варіантів використання та відображає повний обсяг функціональних вимог у розрізі трьох категорій користувачів. Подальша деталізація окремих варіантів використання до рівня послідовності кроків наведена нижче на прикладі сценарію виставлення оцінки.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

Для кожного з визначених акторів сформульовано перелік варіантів використання – типових сценаріїв взаємодії з системою. Зведений перелік основних варіантів використання наведено в табл. 2.1.

Таблиця 2.1

Перелік основних варіантів використання системи

Актор	Варіант використання	Опис
Усі	Автентифікація	Вхід у систему за електронною поштою та паролем
Усі	Зміна пароля	Зміна власного пароля доступу до системи
Студент	Перегляд оцінок	Перегляд власних оцінок з усіх предметів
Студент	Перегляд рейтингу	Перегляд позиції в рейтингу групи та середнього балу
Студент	Додавання досягнення	Внесення інформації про позанавчальне досягнення зі скан-копією
Викладач	Виставлення оцінки	Виставлення оцінки конкретному студенту з вибором типу роботи
Викладач	Редагування оцінки	Зміна значення оцінки протягом 24 годин з моменту виставлення
Викладач	Статистика предмета	Розподіл оцінок, середній бал, динаміка успішності групи
Викладач	Експорт відомості	Вивантаження оцінок з предмета у форматі XLSX
Адмін.	Управління групами	Створення, редагування, видалення академічних груп
Адмін.	Управління кафедрами	CRUD-операції з довідником кафедр ЗВО
Адмін.	Управління предметами	CRUD-операції з предметами та призначення їх викладачам
Адмін.	Управління користувачами	Створення облікових записів усіх типів з імпортом з CSV
Адмін.	Підтвердження досягнення	Модерація позанавчальних досягнень, доданих студентами
Адмін.	Аналітика	Перегляд зведених показників роботи системи у графічному вигляді

Кожен з перелічених варіантів використання може бути деталізовано до рівня окремої послідовності кроків. Як приклад розглянемо варіант використання «Виставлення оцінки». Передумовою цього сценарію є те, що в системі є зареєстрований викладач, до облікового запису якого прив'язаний певний предмет та певна академічна група. Основний потік подій включає такі

кроки: викладач входить у систему під своїм обліковим записом; на головній сторінці обирає предмет, оцінки за яким потрібно виставити; на сторінці деталей предмета обирає академічну групу; у списку студентів натискає кнопку «Додати оцінку» поряд з потрібним прізвищем; у модальному вікні вказує числове значення оцінки за 100-бальною шкалою та обирає тип роботи з випадального списку; натискає кнопку «Зберегти». Постумовою сценарію є збереження нової оцінки в базі даних з фіксацією дати виставлення та автора виставлення, а також відображення цієї оцінки в особистому кабінеті студента.

Об'єктно-орієнтована структура серверної частини застосунку побудована за принципами шарової архітектури. Кожна основна сутність предметної області (User, Student, Teacher, Subject, Group, Department, Grade, Achievement, AcademicYear) представлена в коді як набір з трьох тісно пов'язаних артефактів:

- модель Prisma – опис сутності в файлі schema.prisma з зазначенням полів, типів даних, обмежень та зв'язків;
- сервіс – клас або модуль з функціями бізнес-логіки, що оперують з даною сутністю (наприклад, GradeService реалізує операції створення, редагування, видалення оцінок, обчислення середнього балу);
- контролер – модуль, що обробляє HTTP-запити, пов'язані з сутністю, та делегує бізнес-логіку відповідному сервісу.

Така структура забезпечує чітке розмежування зон відповідальності та полегшує супровід коду. Між класами існує обмежений набір зв'язків: контролер залежить від сервісу, сервіс – від PrismaClient (доступу до даних). Зворотних залежностей не передбачено, що виключає циклічні залежності та полегшує тестування.

Клієнтська частина побудована за принципом компонентної декомпозиції. Виділено три верхньорівневі шари:

- сторінки (pages) – компоненти, що відповідають за окремі екрани застосунку (StudentDashboard, TeacherDashboard, AdminDashboard, LoginPage, GradesPage тощо);

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

- блоки інтерфейсу (features) – складені компоненти, що реалізують конкретні функціональні блоки (форма виставлення оцінки, таблиця рейтингу, картка студента);
- базові компоненти (components) – універсальні компоненти, що використовуються в кількох місцях застосунку (модальні вікна, спінери, форми).

2.2 Розробка та проєктування бази даних системи

Проєктування бази даних є одним з ключових етапів розробки інформаційної системи [7], оскільки структура збереження даних суттєво впливає на функціональні можливості системи, продуктивність обробки запитів та простоту супроводу системи в майбутньому. У цьому підрозділі описано побудовану реляційну структуру бази даних застосунку AcademicRate.

На основі функціональних вимог системи та визначеного переліку варіантів використання виявлено основні сутності предметної області: користувач (з трьома ролями), студент, викладач, академічна група, кафедра, навчальний предмет, оцінка, тип роботи, позанавчальне досягнення, академічний рік, токен оновлення сесії.

Між цими сутностями встановлено такі зв'язки. Кожен користувач належить до однієї з трьох ролей. Студент є спеціалізацією користувача, прив'язаною до однієї академічної групи. Викладач є спеціалізацією користувача, прив'язаною до однієї кафедри (та може викладати в кількох групах). Академічна група належить до однієї кафедри. Навчальний предмет належить до однієї кафедри й вивчається кількома академічними групами (зв'язок «багато до багатьох»). Оцінка прив'язується одночасно до студента, предмета, викладача (автор виставлення) та має тип роботи. Позанавчальне досягнення прив'язане до студента і має статус підтвердження (підтверджене / непідтверджене).

На основі концептуальної моделі побудовано фізичну структуру з 11 таблиць у реляційній СУБД PostgreSQL. Перелік таблиць з коротким описом їх призначення наведено в табл. 2.2.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				35
Змн.	Арк.	№ докум.	Підпис	Дата		

Перелік таблиць бази даних системи AcademicRate

№	Назва таблиці	Призначення
1	users	Загальні дані про користувачів системи (П.І.Б., email, hashedPassword, role)
2	students	Розширені дані про студентів (зв'язок з користувачем, групою, рік вступу)
3	teachers	Розширені дані про викладачів (зв'язок з користувачем і кафедрою)
4	departments	Довідник кафедр ЗВО (назва, аббревіатура)
5	groups	Академічні групи (номер, рік початку навчання, кафедра)
6	subjects	Навчальні предмети (назва, кредити ECTS, кафедра, тип контролю)
7	subject_groups	Зв'язок «багато до багатьох» між предметами та групами
8	subject_teachers	Призначення викладачів на предмети
9	grades	Оцінки студентів (значення, тип роботи, дата, автор)
10	achievements	Позанавчальні досягнення студентів (тип, дата, статус)
11	refresh_tokens	Активні токени оновлення сесій користувачів

Розглянемо детальніше структуру основних таблиць. Таблиця users є базовою для всіх типів користувачів і містить такі поля: id (серійний первинний ключ), email (унікальне текстове поле для адреси електронної пошти), hashedPassword (текстове поле для зберігання хешу пароля), lastName, firstName, middleName (текстові поля для прізвища, імені та по батькові), role (текстове поле зі значеннями STUDENT, TEACHER, ADMIN), createdAt та updatedAt (часові мітки створення та останнього оновлення запису). Структуру полів таблиці users наведено в табл. 2.3.

Таблиця 2.3

Структура таблиці users

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор користувача
email	VARCHAR(255)	UNIQUE, NOT NULL	Адреса електронної пошти (логін)
hashedPassword	VARCHAR(255)	NOT NULL	Хеш пароля (bcrypt)
lastName	VARCHAR(100)	NOT NULL	Прізвище

firstName	VARCHAR(100)	NOT NULL	Ім'я
middleName	VARCHAR(100)	NULL	По батькові
role	VARCHAR(16)	ENUM	Роль (STUDENT / TEACHER / ADMIN)
createdAt	TIMESTAMP	DEFAULT NOW()	Дата створення запису
updatedAt	TIMESTAMP	DEFAULT NOW()	Дата останнього оновлення

Таблиці students та teachers містять додаткові поля, специфічні для відповідних типів користувачів. Поле userId є зовнішнім ключем на таблицю users, забезпечуючи зв'язок «один до одного» – кожному студенту й викладачеві відповідає рівно один загальний обліковий запис у таблиці users. Таблиця students додатково містить поле groupId (зовнішній ключ на таблицю groups) та admissionYear (рік вступу). Таблиця teachers містить поле departmentId (зовнішній ключ на таблицю departments).

Структуру ключових полів таблиці grades, що зберігає виставлені студентам оцінки, наведено в табл. 2.4.

Таблиця 2.4

Структура таблиці grades

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор оцінки
studentId	INTEGER	FK→students.id	Студент, якому виставлено оцінку
subjectId	INTEGER	FK→subjects.id	Предмет, з якого виставлено оцінку
teacherId	INTEGER	FK→teachers.id	Викладач – автор виставлення
value	INTEGER	$0 \leq \text{value} \leq 100$	Значення оцінки за 100-бальною шкалою
workType	VARCHAR(32)	ENUM	Тип роботи (LECTURE, LAB, CONTROL тощо)
issuedAt	TIMESTAMP	DEFAULT NOW()	Дата та час виставлення оцінки
comment	TEXT	NULL	Необов'язковий коментар викладача

Облік позанавчальних досягнень студентів реалізовано таблицею achievements. Кожен запис фіксує тип і назву досягнення, його рівень, посилання на скан-копію підтверджувального документа та статус модерації,

що змінюється адміністратором. Структуру таблиці achievements наведено в табл. 2.5.

Таблиця 2.5

Структура таблиці achievements

Поле	Тип	Обмеження	Опис
id	SERIAL	PRIMARY KEY	Ідентифікатор досягнення
studentId	INTEGER	FK→students.id	Студент – власник досягнення
type	VARCHAR(32)	ENUM	Тип (олімпіада/конференція/конкурс/публікація/сертифікат)
title	VARCHAR(255)	NOT NULL	Назва досягнення
level	VARCHAR(32)	ENUM	Рівень (внутр./регіон./всєукр./міжнар.)
proofUrl	TEXT	NULL	Посилання на скан-копію документа
status	VARCHAR(16)	ENUM	Статус (PENDING / CONFIRMED / REJECTED)
issuedAt	DATE	NULL	Дата отримання досягнення
createdAt	TIMESTAMP	DEFAULT NOW()	Дата подання запису

На полях, що часто використовуються у фільтрації, побудовано індекси: на полі email таблиці users – для пришвидшення автентифікації; на парі полів (studentId, subjectId) таблиці grades – для швидкої вибірки всіх оцінок конкретного студента з певного предмета; на полі groupId таблиці students – для формування рейтингів. Зовнішні ключі сконфігуровано з опцією ON DELETE RESTRICT для запобігання випадковому видаленню сутностей, на які є посилання з інших таблиць.

До системи додано поняття академічного року для підтримки збереження історичних даних – оцінок та рейтингів за попередні роки – у систему додано поняття академічного року.

Кожна оцінка прив'язана до конкретного академічного року через дату виставлення issuedAt. При обчисленні поточного рейтингу враховуються лише оцінки поточного року, тоді як дані за попередні роки доступні в архівному розділі інтерфейсу.

Поняття академічного року реалізовано через довідник `academic_years` (рік початку, рік закінчення, статус), де поточний рік позначається відповідним статусом. При запуску нового академічного року адміністратор системи виконує операцію переходу – стара запис закривається, нова створюється, а студенти автоматично підвищуються на курс вище (з можливістю ручної корекції для тих, хто залишився на повторне навчання).

2.3 Проєктування та реалізація алгоритмів роботи системи

Серед усіх алгоритмів, реалізованих у системі `AcademicRate`, найбільш специфічним для предметної області та найскладнішим за логікою є алгоритм розрахунку академічного рейтингу студента. Цей алгоритм є центральним елементом системи й вимагає окремого обґрунтування. У підрозділі також розглянуто алгоритм автентифікації як приклад ключового алгоритму інфраструктурного характеру.

Академічний рейтинг студента – це числовий показник, що характеризує загальну успішність студента й використовується для його ранжування серед інших студентів. Залежно від політики конкретного закладу освіти існують різні методики обчислення рейтингу – за середнім арифметичним балом, за середнім зваженим балом з урахуванням кредитів ECTS, з нарахуванням додаткових балів за позанавчальні досягнення тощо.

У системі `AcademicRate` реалізовано методику розрахунку, що ґрунтується на середньому арифметичному значенні оцінок з усіх предметів за поточний академічний рік з масштабуванням до 5-бальної шкали GPA. Формально базовий рейтинг B обчислюється за формулою:

$$B = (\sum O_i) / n, \quad (2.1)$$

де O_i – значення i -ї оцінки за 100-бальною шкалою; n – загальна кількість оцінок студента в поточному академічному році.

Масштабований показник GPA обчислюється за формулою:

$$GPA = B \times 5 / 100 = B / 20. \quad (2.2)$$

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

Тобто оцінка 100 балів відповідає $GPA = 5,0$; оцінка 60 балів – $GPA = 3,0$; оцінка 50 балів – $GPA = 2,5$. Узагальнену блок-схему алгоритму обчислення рейтингу та ранжування студентів групи наведено на рис. 2.3.

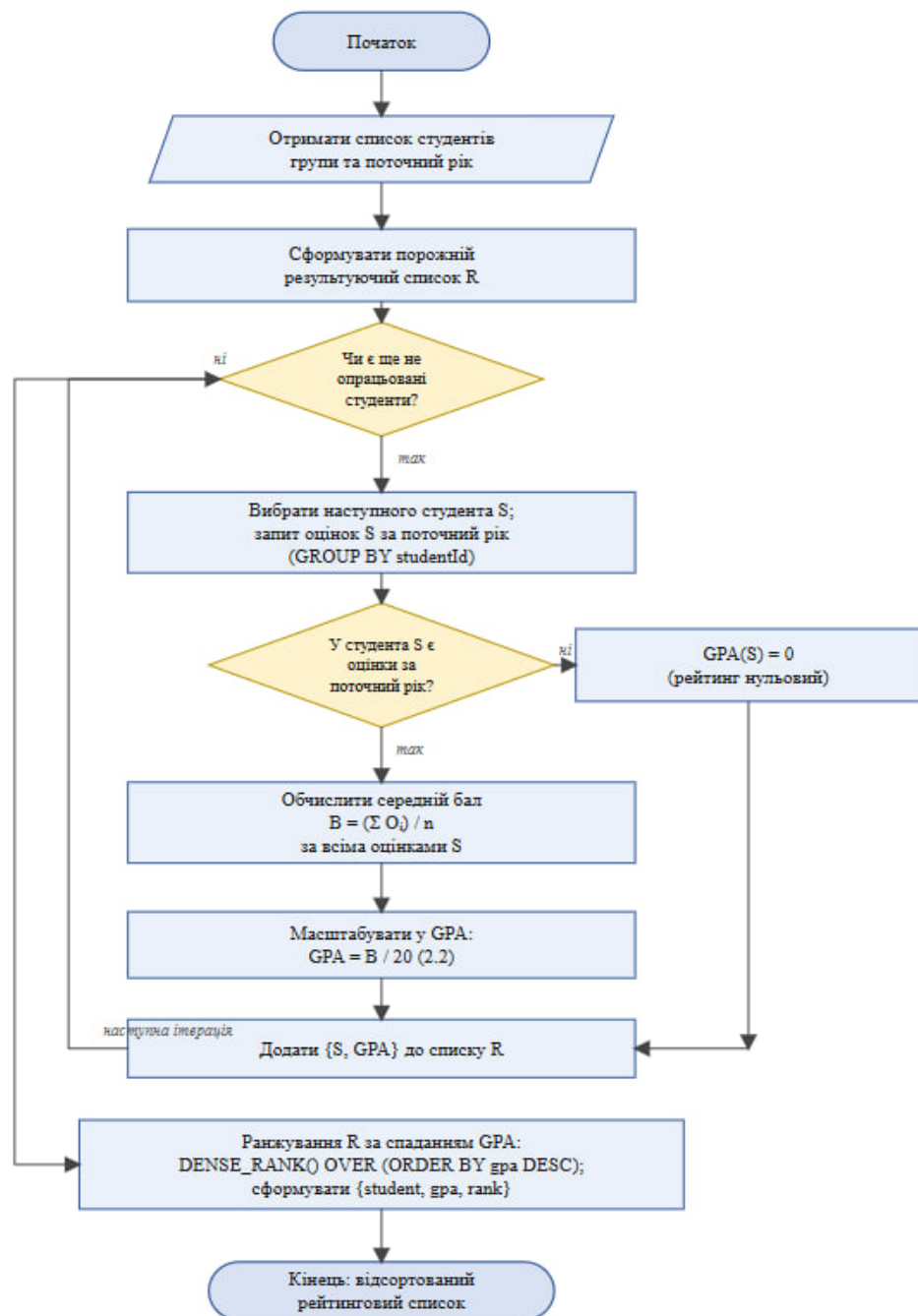


Рис. 2.3. Блок-схема алгоритму розрахунку академічного рейтингу

Розрахунок виконується у двох випадках: при відкритті панелі студента (для відображення власного рейтингу) та при відкритті панелі викладача чи адміністратора зі сторінки групи (для формування рейтингового списку всіх студентів групи). У другому випадку додатково виконується ранжування – присвоєння кожному студенту цілочисельної позиції в списку, упорядкованому

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

за спаданням GPA. Для ранжування використовується SQL-функція DENSE_RANK(), що ефективно реалізована в PostgreSQL засобами індексів [23].

Псевдокод алгоритму обчислення рейтингу групи має такий вигляд: для кожного студента групи виконується SQL-запит до таблиці grades із групуванням за полем studentId та обчисленням середнього значення поля value за поточний академічний рік. Отримані середні значення масштабуються в GPA за формулою:

$$GPA = B \times 5 / 100 = B / 20 \quad (2.2).$$

На наступному кроці засобами віконної функції DENSE_RANK() OVER (ORDER BY gra DESC) обчислюється позиція кожного студента в рейтингу. Кінцевий результат – впорядкований за позицією список об'єктів {student, gra, rank}.

Особливі випадки обробляються таким чином. Якщо у студента взагалі немає оцінок за поточний академічний рік (наприклад, тільки що зарахований), його рейтинг встановлюється як нульовий і його позиція в рейтинговому списку буде в кінці. Якщо у двох студентів однаковий GPA, вони отримують однакову позицію в рейтингу (саме така поведінка DENSE_RANK()).

Алгоритм автентифікації користувача в системі AcademicRate реалізований за двотокенною схемою на основі стандарту JSON Web Token [29]. При вході в систему користувач передає свої облікові дані (електронну пошту та пароль). Сервер виконує такі кроки: знаходить запис у таблиці users за вказаною електронною поштою; якщо запис не знайдено, повертає помилку 401; обчислює хеш переданого пароля з використанням збереженого хешу та порівнює; при невідповідності повертає помилку 401; при відповідності генерує два токени – access (термін дії 15 хвилин) та refresh (термін дії 7 днів); записує refresh-токен у таблицю refresh_tokens; повертає клієнту обидва токени.

Access-токен містить ідентифікатор користувача, його роль, час видачі та час закінчення, підписаний секретним ключем сервера за алгоритмом HMAC-

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

SHA256. Цей токен використовується для авторизації всіх наступних запитів – клієнт передає його у заголовку Authorization кожного запиту, що потребує автентифікації.

Двотокенні схеми автентифікації широко застосовуються в сучасних вебзастосунках для розмежування доступу [38]. Refresh-токен – це випадковий рядок, унікальний ідентифікатор активної сесії користувача. Він зберігається на сервері в таблиці refresh_tokens та передається клієнтові в HTTP-only cookie. При закінченні терміну дії access-токена клієнт самостійно надсилає запит /api/auth/refresh, передаючи в ньому refresh-токен; сервер перевіряє наявність токена в базі і його термін дії, генерує новий access-токен та повертає клієнтові. Це дає змогу не вимагати від користувача повторного введення пароля частіше, ніж раз на тиждень.

Алгоритм виставлення оцінки реалізовано як послідовність кроків з перевіркою прав доступу та валідацією на кожному кроці: верифікація access-токена; перевірка, що користувач має роль TEACHER (інакше - 403); перевірка, що зазначений предмет призначено цьому викладачеві (інакше 403); перевірка, що зазначений студент належить до однієї з груп, для яких призначено цей предмет (інакше 400); валідація значення оцінки ($0 \leq \text{value} \leq 100$); валідація типу роботи (значення з переліку) створення запису в таблиці grades з фіксацією поточного часу та ідентифікатора викладача: повернення створеного запису клієнту з HTTP-статусом 201.

2.4 Реалізація вебзастосунку клієнтська та серверна частини

Реалізація програмного комплексу AcademicRate виконана за принципом монорепозиторію (monorepo), а саме клієнтська і серверна частини, а також спільні модулі (типи даних, утиліти, конфігурація) розміщено в межах одного git-репозиторію. Це спрощує розробку, оскільки зміни, що зачіпають обидві частини, виконуються в одному коміті, а спільні типи даних автоматично оновлюються в обох частинах. Корінь репозиторію містить три робочі простори (workspaces) менеджера пакетів npm: /server (серверний застосунок), /client (клієнтський застосунок) та /shared (спільний код). Завдяки механізму npm

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

workspaces спільний пакет підключається до серверної та клієнтської частин як звичайна локальна залежність, без публікації в зовнішній реєстр і без копіювання файлів.

Ключовою перевагою монорепозиторію для проєкту, що використовує TypeScript на обох кінцях, є наскрізна типобезпека (end-to-end type safety). Типи даних, якими обмінюються клієнт і сервер (наприклад, структура об'єкта оцінки GradeDTO або відповіді на запит рейтингу RatingResponse), описуються один раз у пакеті /shared і використовуються як серверним кодом при формуванні відповідей, так і клієнтським кодом при їх опрацюванні. Внаслідок цього будь-яка зміна структури даних на сервері, не узгоджена з клієнтом, виявляється компілятором TypeScript на етапі складання, а не під час виконання у браузері користувача. Це суттєво знижує клас помилок, пов'язаних з невідповідністю контрактів даних між частинами системи.

Загальний цикл розробки організовано навколо системи контролю версій Git: кожна функціональна зміна оформлюється окремою гілкою (feature branch), проходить перевірку лінтером ESLint та автоматичними тестами, після чого зливається в основну гілку. Складання обох частин застосунку, запуск тестів та статичний аналіз коду уніфіковано через npm-скрипти кореневого пакета, що дає змогу виконувати їх однією командою як локально, так і в середовищі неперервної інтеграції.

Серверна частина застосунку розміщена в директорії /server монорепозиторію та реалізує шарову архітектуру. Внутрішня структура каталогів відображає поділ на шари та функціональні домени. Каталог /server/src/index.ts містить точку входу серверного застосунку з ініціалізацією екземпляра Express, реєстрацією глобального проміжного програмного забезпечення та запуском прослуховування порту. Файл /server/src/app.ts відповідає за конфігурацію застосунку Express окремо від запуску сервера, що дозволяє тестувати застосунок без фактичного відкриття мережевого порту. Каталог /server/src/routes/ містить модулі з визначенням маршрутів REST API, згрупованих за функціональними доменами (auth, users, students, teachers,

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

groups, subjects, grades, achievements, analytics). У каталозі `/server/src/controllers/` розміщені обробники HTTP-запитів з прийманням запитів, викликом відповідних сервісів та формуванням відповідей. Каталог `/server/src/services/` містить модулі з бізнес-логікою, а `/server/src/middleware/` – проміжне програмне забезпечення для автентифікації, авторизації за ролями, валідації, обробки помилок та обмеження частоти запитів. Каталог `/server/src/prisma/` забезпечує конфігурацію ORM Prisma, схему бази даних та модуль єдиного екземпляра PrismaClient, а `/server/src/utils/` містить допоміжні утиліти – валідатори, форматувачі та обгортки для роботи з JWT.

Розподіл відповідальності між шарами є строгим. Маршрут (route) лише пов'язує URI та HTTP-метод з певним обробником і ланцюжком проміжного програмного забезпечення. Контролер (controller) відповідає за роботу з протоколом HTTP, читає параметри запиту, тіло та заголовки, викликає сервіс і перетворює його результат на HTTP-відповідь з відповідним статусом. Сервіс (service) реалізує бізнес-логіку без посилань на об'єкти запиту чи відповіді HTTP, що робить його придатним для повторного використання та модульного тестування у відриві від мережевого рівня. Шар доступу до даних інкапсульовано в Prisma Client. Така дисципліна залежностей, за якої контролер залежить від сервісу, сервіс залежить від клієнта даних, але не навпаки, виключає циклічні залежності та відповідає принципу інверсії залежностей.

Як приклад розглянемо повний шлях обробки запиту на виставлення оцінки (POST `/api/grades`). Запит послідовно проходить такі стадії: глобальне проміжне ПЗ розбору тіла запиту (`express.json`) перетворює тіло JSON на об'єкт JavaScript; проміжне ПЗ автентифікації перевіряє access-токен і додає до запиту об'єкт поточного користувача; проміжне ПЗ авторизації перевіряє наявність у користувача ролі TEACHER; проміжне ПЗ валідації перевіряє структуру та типи полів тіла запиту за описаною схемою; контролер `gradeController.create` викликає сервіс `gradeService.createGrade`, передаючи йому перевірені дані та ідентифікатор викладача; сервіс виконує бізнес-перевірки (належність предмета викладачеві, належність студента до групи предмета), створює запис у таблиці

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

grades через Prisma Client і повертає створений об'єкт; контролер формує відповідь зі статусом 201 та тілом — створеним об'єктом оцінки. У разі помилки на будь-якій стадії управління передається централізованому обробнику помилок.

Кожен сервіс реалізовано як модуль з набором асинхронних функцій, що повертають Promise, оскільки всі операції з базою даних є асинхронними. Наприклад, файл /server/src/services/grade.service.ts містить функції createGrade, updateGrade, deleteGrade, getStudentGrades та calculateGroupRanking. Винесення бізнес-логіки в окремий шар сервісів дає змогу повторно використовувати ту саму функцію з різних контролерів (зокрема функція обчислення рейтингу викликається як з контролера панелі студента, так і з контролера панелі викладача) без дублювання коду, а також тестувати логіку, підставляючи у сервіс тестовий екземпляр клієнта даних.

Доступ до бази даних організовано через єдиний екземпляр Prisma Client, створюваний один раз при старті застосунку та імпортований усіма сервісами. Такий підхід (патерн «єдиний екземпляр», singleton) запобігає вичерпанню пулу з'єднань із базою даних, що сталося б у разі створення нового клієнта на кожен запит. Prisma Client автоматично керує пулом з'єднань, повторно використовуючи відкриті з'єднання між запитами.

Уся взаємодія клієнта із сервером відбувається через інтерфейс REST API. Загальна кількість REST-ендпоінтів сервера становить 47, згрупованих за дев'ятьма функціональними доменами. Перелік основних ендпоінтів з зазначенням HTTP-методу, шляху, рівня доступу та призначення наведено в табл. 2.6.

Таблиця 2.6

Основні REST-ендпоінти API сервера AcademicRate

Метод	URI	Роль	Опис
POST	/api/auth/login	Усі	Автентифікація користувача
POST	/api/auth/refresh	Усі	Оновлення access-токена

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

POST	/api/auth/logout	Усі	Завершення сесії
GET	/api/me	Усі	Отримання даних поточного користувача
GET	/api/students/:id/grades	Усі*	Список оцінок студента
GET	/api/students/:id/rating	Усі*	Поточний рейтинг студента
GET	/api/groups/:id/ranking	Усі*	Рейтинг групи
POST	/api/grades	TEACHER	Виставлення нової оцінки
PATCH	/api/grades/:id	TEACHER	Редагування оцінки
GET	/api/groups	ADMIN	Перегляд списку груп
POST	/api/groups	ADMIN	Створення нової групи
POST	/api/users	ADMIN	Створення нового користувача
GET	/api/analytics/overview	ADMIN	Загальна статистика системи

Примітка. «Усі*» означає доступність ендпоінта для всіх автентифікованих користувачів з додатковою перевіркою на рівень доступу: студент може отримати лише власні дані, викладач – дані студентів зі своїх груп, адміністратор – будь-які дані.

Проектування API виконано згідно з принципами REST: іменування ресурсів виконано іменниками у множині (students, grades, groups), а дії над ресурсами виражено HTTP-методами (GET — читання, POST — створення, PATCH — часткове оновлення, DELETE — видалення), а не дієсловами в шляху. Для ідентифікації конкретного ресурсу використовуються параметри шляху (наприклад, /api/grades/:id), а для фільтрації та посторінкового виведення колекцій — параметри рядка запиту (query parameters), зокрема ?page, ?limit, ?groupId. Семантика HTTP-статусів відповідей дотримана послідовно: 200 — успішне читання чи оновлення, 201 — успішне створення ресурсу, 400 — помилка валідації вхідних даних, 401 — відсутність або недійсність автентифікації, 403 — недостатність прав, 404 — ресурс не знайдено, 409 —

конфлікт (наприклад, спроба створити користувача з уже наявною електронною поштою), 429 — перевищення дозволеної частоти запитів.

Клієнтська частина застосунку розміщена в директорії `/client` монорепозиторію та реалізована як односторінковий застосунок (SPA) на бібліотеці React із застосуванням мови TypeScript. Структура каталогів відображає компонентну декомпозицію: `/client/src/main.tsx` — точка входу застосунку, що монтує кореневий компонент у DOM та обгортає його провайдерами контекстів; `/client/src/App.tsx` — кореневий компонент з визначенням маршрутизації; `/client/src/pages/` — компоненти сторінок (StudentDashboard, TeacherDashboard, AdminDashboard, LoginPage, NotFoundPage); `/client/src/features/` — функціональні блоки, що поєднують кілька компонентів та логіку (форма виставлення оцінки, таблиця рейтингу, картка досягнення); `/client/src/components/` — базові перевикористовувані компоненти інтерфейсу (кнопки, модальні вікна, індикатори завантаження, поля форм); `/client/src/api/` — модулі взаємодії з REST API сервера; `/client/src/hooks/` — кастомні React-хуки; `/client/src/context/` — React-контексти для глобального стану (зокрема контекст автентифікації); `/client/src/utils/` — утилітарні функції форматування та валідації.

Компоненти побудовано виключно як функціональні з використанням механізму хуків (hooks), що є сучасним підходом у React 18, який витіснив класові компоненти. Дані передаються від батьківських компонентів до дочірніх через властивості (props), а зворотний зв'язок реалізується через функції зворотного виклику, що передаються так само через властивості. Для кожного компонента засобами TypeScript описано інтерфейс його властивостей, що забезпечує перевірку коректності використання компонента на етапі компіляції.

Принциповим архітектурним рішенням клієнтської частини є розмежування двох видів стану – серверного та клієнтського. Серверний стан (server state) охоплює дані, першоджерелом яких є сервер, зокрема списки студентів, оцінки, рейтинги та довідники. Клієнтський стан (client state) описує

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

стан інтерфейсу без збереження на сервері, наприклад відкритість певного модального вікна, поточне значення поля форми, обрану вкладку. Змішування цих двох видів стану в єдиному сховищі є поширеною причиною ускладнення коду, тому в проєкті вони керуються різними засобами.

Для управління серверним станом використовується бібліотека TanStack Query. Вона автоматично кешує результати запитів за ключем запиту, керує станами завантаження та помилок, виконує повторне отримання даних у фоновому режимі при поверненні фокуса на вкладку, реалізує оптимістичні оновлення інтерфейсу та інвалідацію кешу після операцій зміни даних. Зокрема, після успішного виставлення оцінки виконується інвалідація кешу запитів списку оцінок студента та рейтингу групи, унаслідок чого інтерфейс автоматично відображає актуальні дані без ручного перезавантаження сторінки. Застосування TanStack Query дало змогу повністю відмовитися від ручного зберігання серверних даних у локальному стані компонентів та пов'язаного з цим коду синхронізації.

Для управління клієнтським станом використовуються вбудовані React-хуки `useState` та `useReducer` для локального стану окремих компонентів, а для глобального стану, що має бути доступним багатьом компонентам (наприклад, дані поточного автентифікованого користувача), застосовано механізм `React Context`. Контекст автентифікації `AuthContext` надає всім компонентам застосунку доступ до даних користувача, його ролі та функцій входу і виходу, без необхідності передавати ці дані через властивості крізь усю ієрархію компонентів (явище, відоме як «`props drilling`»).

Маршрутизацію в межах SPA реалізовано засобами бібліотеки `react-router-dom`. Визначено публічні маршрути (сторінка входу) та захищені маршрути, доступ до яких можливий лише для автентифікованих користувачів. Захист маршрутів реалізовано компонентом-обгорткою `ProtectedRoute`, що перевіряє наявність дійсної сесії і, за її відсутності, перенаправляє користувача на сторінку входу. Додатково реалізовано розмежування маршрутів за ролями: при спробі переходу за маршрутом, недоступним для поточної ролі (наприклад,

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

студент намагається відкрити адміністративну панель), користувач перенаправляється на доступну йому стартову сторінку.

Взаємодію з REST API сервера інкапсульовано в окремому шарі — модулях каталогу `/client/src/api/`. Усі HTTP-запити виконуються через єдиний клієнт на основі вбудованого браузерного інтерфейсу Fetch API, обгорнутого в утилітарний модуль. Цей модуль автоматично додає до кожного запиту заголовки авторизації з access-токеном, перехоплює відповіді зі статусом 401 і виконує прозоре оновлення токена через refresh-токен з повторенням початкового запиту, а також централізовано опрацьовує помилки. Завдяки спільним з сервером типам даних з пакета `/shared` відповіді API типізовано, тому опрацювання отриманих даних у компонентах також є типобезпечним.

Для оформлення інтерфейсу застосовано бібліотеку компонентів Material-UI (MUI), що надає готові компоненти, які відповідають принципам Material Design та забезпечують доступність і адаптивність. Адаптивність інтерфейсу досягається засобами системи сітки (Grid) та контрольних точок (breakpoints) MUI, завдяки чому застосунок коректно відображається як на широких моніторах, так і на екранах смартфонів. Реалізовано підтримку світлої та темної тем оформлення через механізм тем MUI; обрана тема зберігається між сеансами роботи. Аналітичні діаграми панелі адміністратора побудовано засобами бібліотеки Recharts, що інтегрується з React за компонентним принципом.

Забезпечення коректності даних реалізовано за принципом багаторівневої валідації, оскільки покладатися виключно на перевірки на стороні клієнта неприпустимо з міркувань безпеки — клієнтський код повністю контролюється користувачем і може бути обійдений. Тому валідацію реалізовано на трьох рівнях: клієнтському, серверному та рівні бази даних.

На клієнтському рівні валідація форм виконується безпосередньо під час введення та перед надсиланням запиту. Її призначення — підвищення зручності роботи (миттєвий зворотний зв'язок про некоректне значення без звернення до сервера), а не забезпечення безпеки. На цьому рівні перевіряються формат

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

електронної пошти, обов'язковість заповнення полів, належність числового значення оцінки до діапазону 0–100.

На серверному рівні валідація є обов'язковою та визначальною. Кожен ендпоінт, що приймає дані, має пов'язану з ним схему валідації, реалізовану засобами проміжного програмного забезпечення. Схема описує очікувану структуру тіла запиту, типи полів, обов'язковість, припустимі діапазони значень та перелік допустимих значень для перелічуваних полів (наприклад, тип роботи має належати множині {LECTURE, LAB, CONTROL, EXAM, EXTRA}). У разі невідповідності запит відхиляється зі статусом 400 та структурованим описом помилок, при цьому до бізнес-логіки некоректні дані не потрапляють. Окрім перевірки формату, на серверному рівні виконуються бізнес-перевірки, що потребують звернення до бази даних: належність предмета викладачеві, належність студента до групи, для якої викладається предмет, відсутність дубльованої електронної пошти при створенні користувача.

На рівні бази даних узгодженість гарантується обмеженнями цілісності, описаними у схемі Prisma та реалізованими засобами PostgreSQL: обмеженням унікальності (UNIQUE) для електронної пошти, обмеженнями зовнішніх ключів (FOREIGN KEY) для всіх зв'язків між таблицями, обмеженнями NOT NULL для обов'язкових полів. Цей рівень є останнім рубежем захисту цілісності: навіть у разі помилки в коді програми база даних не дозволить зберегти суперечливі дані. Операції, що змінюють кілька пов'язаних записів одночасно (наприклад, перехід на новий академічний рік), виконуються в межах транзакцій бази даних, що гарантує їх атомарність — або всі зміни застосовуються разом, або жодна.

Обробку помилок на сервері централізовано в спеціальному проміжному програмному забезпеченні (error-handling middleware), що реєструється останнім у ланцюжку Express. Усі помилки, що виникають у контролерах і сервісах, передаються до цього обробника, який визначає тип помилки та формує відповідну HTTP-відповідь. Для розрізнення категорій помилок

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

запроваджено ієрархію власних класів помилок: `ValidationError` (відповідає статусу 400), `UnauthorizedError` (401), `ForbiddenError` (403), `NotFoundError` (404), `ConflictError` (409). Такий підхід дає змогу в коді бізнес-логіки лаконічно сигналізувати про конкретну проблему викиданням відповідного виключення, а перетворення його на коректну HTTP-відповідь виконується централізовано в одному місці.

Відповідь про помилку має уніфіковану структуру JSON, що містить машинозчитуваний код помилки та зрозуміле користувачеві повідомлення. Важливим аспектом безпеки є те, що клієнтові не передаються внутрішні технічні подробиці (стек виклику, текст SQL-запиту, повідомлення СУБД), оскільки вони можуть бути використані зловмисником для розвідки структури системи. Натомість повний технічний опис помилки фіксується в журналі сервера.

Журналювання (logging) реалізовано на двох рівнях. Журнал доступу фіксує кожен HTTP-запит із зазначенням методу, шляху, коду відповіді та часу обробки, що дає змогу відстежувати навантаження та виявляти аномалії. Журнал помилок фіксує повну інформацію про виключення з рівнем серйозності. У режимі розробки журнали виводяться в консоль у читабельному форматі, а у production-конфігурації — у структурованому форматі JSON, придатному для подальшого опрацювання засобами систем збору журналів. Асинхронна неблокуюча модель Node.js забезпечує, що журналювання не блокує обробку запитів.

На етапі реалізації застосовано низку заходів для забезпечення безпеки системи відповідно до сучасних практик автентифікації та авторизації у вебзастосунках [39]. Перелік заходів сформовано з орієнтацією на типові класи вразливостей вебзастосунків, систематизовані у переліку OWASP Top 10, зокрема порушення контролю доступу, недоліки криптографічного захисту, ін'єкції та хибну конфігурацію безпеки.

Захист облікових даних. Усі паролі користувачів зберігаються лише у вигляді криптографічних хешів, обчислених засобами адаптивного алгоритму

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

bcrypt з фактором роботи 10 [40]. Застосування bcrypt замість звичайних геш-функцій (на кшталт SHA-256) є принциповим: bcrypt є навмисно повільним та містить вбудовану «сіль» (salt), що робить непрактичними атаки перебором за словником і за допомогою попередньо обчислених таблиць. У відкритому вигляді паролі не зберігаються та не передаються по мережі за умови використання HTTPS на транспортному рівні.

Контроль доступу. Авторизація реалізована за моделлю контролю доступу на основі ролей (RBAC) і застосовується на кожному захищеному ендпоінті у вигляді окремого шару проміжного програмного забезпечення. Перевіряється не лише факт автентифікації, а й достатність прав для конкретної дії, а також належність ресурсу користувачеві: студент не може отримати доступ до даних іншого студента навіть за наявності дійсного токена, оскільки сервіс додатково звіряє ідентифікатор власника ресурсу з ідентифікатором того, хто робить запит. Це унеможлиблює клас вразливостей, пов'язаних з небезпечними прямими посиланнями на об'єкти (IDOR).

Захист від ін'єкцій. Усі запити до бази даних виконуються через ORM Prisma, що автоматично параметризує SQL-запити і відокремлює дані від тексту запиту, унеможливлюючи SQL-ін'єкції. У коді відсутні прямі виклики `$queryRawUnsafe()` з конкатенацією користувацьких даних, що було перевірено під час код-ревію.

Захист від міжсайтового скриптингу та підробки запитів. Для запобігання атакам Cross-Site Scripting (XSS) використовується стандартне автоматичне екранування вмісту бібліотекою React при відображенні рядкових значень у DOM; небезпечний механізм `dangerouslySetInnerHTML` у коді не використовується. Cookie з refresh-токенами встановлено з атрибутами `HttpOnly`, `Secure` та `SameSite=Strict`, що захищає їх від несанкціонованого читання засобами JavaScript та від атак типу підробки міжсайтових запитів (CSRF). Крос-доменні запити обмежено політикою CORS, налаштованою на дозвіл звернень лише з довіреного походження (домену клієнтської частини), заданого змінною оточення.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Протидія автоматизованим атакам та хибній конфігурації. Для запобігання атакам повним перебором облікових даних реалізовано обмеження частоти запитів засобами проміжного ПЗ `express-rate-limit`: для ендпоінта автентифікації встановлено ліміт 5 спроб з однієї IP-адреси за 15-хвилинне вікно, після перевищення якого сервер відповідає статусом 429. Усі секрети (ключі підпису токенів, рядок підключення до бази даних) задаються через змінні оточення і не зберігаються у вихідному коді чи системі контролю версій. У `production`-конфігурації Nginx встановлено стандартні заголовки безпеки HTTP — `Strict-Transport-Security`, `X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy` та `Content-Security-Policy`, що зменшують поверхню атаки на рівні браузера.

Сукупність перелічених заходів утворює багаторівневий захист (*defense in depth*), за якого компрометація одного рівня не призводить до повного порушення безпеки системи. Реалізовані механізми покривають основні класи вразливостей, актуальні для вебзастосунків відповідного призначення, та відповідають нефункціональним вимогам щодо безпеки.

Висновки до другого розділу

У другому розділі виконано проєктування інформаційної системи. Визначено трьох дійових осіб системи (студент, викладач, адміністратор) та сформульовано основні варіанти використання для кожної з них загальною кількістю 14 сценаріїв. Описано об'єктно-орієнтовану структуру серверної частини з шаровою організацією (моделі, сервіси, контролери) та клієнтської частини з компонентною декомпозицією (сторінки, функціональні блоки, базові компоненти).

Спроектовано та реалізовано структуру бази даних, що складається з 11 таблиць і відображає всі основні сутності предметної області та зв'язки між ними. Описано фізичну структуру ключових таблиць (`users`, `grades`, `achievements`) із зазначенням типів даних, обмежень та індексів. Обґрунтовано рішення щодо використання віконних функцій PostgreSQL для ефективного формування рейтингових списків.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

Спроектовано основні алгоритми системи: алгоритм розрахунку академічного рейтингу за середнім балом з масштабуванням до 5-бальної шкали GPA; алгоритм автентифікації за двотокенною схемою JWT; алгоритм виставлення оцінки з багаторівневою перевіркою прав доступу.

Описано реалізацію серверної частини як набору з 47 REST-ендпоінтів у шаровій архітектурі (маршрути, контролери, сервіси, шар доступу до даних) з організацією коду за принципом монорепозиторію та наскрізною типобезпекою на основі спільних типів даних. Детально розглянуто повний шлях обробки запиту крізь ланцюжок проміжного програмного забезпечення. Описано реалізацію клієнтської частини як SPA на React 18 з розмежуванням серверного стану (TanStack Query) та клієнтського стану (React-хуки і Context), захищеною маршрутизацією за ролями та інкапсульованим шаром взаємодії з API.

Описано реалізацію багаторівневої валідації даних (на клієнті, сервері та рівні бази даних), централізованої обробки помилок із власною ієрархією класів виключень та журналювання. Систематизовано заходи із забезпечення безпеки системи з орієнтацією на типові класи вразливостей вебзастосунків: хешування паролів засобами bcrypt, контроль доступу за моделлю RBAC із перевіркою належності ресурсу, параметризовані SQL-запити через Prisma, автоматичне екранування в React, HTTP-only cookies для refresh-токенів, політику CORS, обмеження частоти запитів та заголовки безпеки HTTP, що в сукупності реалізують принцип багаторівневого захисту.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ

3.1 Порядок встановлення та налаштування параметрів системи

Серверна частина застосунку AcademicRate постачається у вигляді набору вихідних кодів у git-репозиторії [30] з конфігураційними файлами для контейнеризованого розгортання засобами Docker та Docker Compose [28]. Такий спосіб розгортання забезпечує відтворюваність середовища роботи та мінімізує імовірність помилок, пов'язаних з відмінностями у версіях залежностей між середовищем розробки й виробничим середовищем.

Перед розгортанням системи на цільовому сервері мають бути виконані такі підготовчі дії: встановлено операційну систему сімейства Linux (рекомендована Ubuntu Server 22.04 LTS або новіша); встановлено пакет Docker Engine версії 24 або новішої та плагін Docker Compose v2; виділено мінімум 4 ГБ оперативної пам'яті та 20 ГБ вільного дискового простору; забезпечено мережеве з'єднання з Інтернетом для першочергового завантаження Docker-образів.

Підтримуються також інші Linux-дистрибутиви (Debian, CentOS Stream, Fedora Server, Rocky Linux), однак конкретні команди встановлення Docker можуть відрізнятися. Для розгортання на віртуальних серверах хмарних провайдерів (DigitalOcean, AWS EC2, Hetzner Cloud) додаткових налаштувань не потрібно.

Розгортання у Windows-середовищі можливе через Docker Desktop, проте не рекомендується для виробничого використання через додаткові накладні витрати, пов'язані з роботою Docker через WSL2.

Основні параметри системи задаються через файл змінних оточення .env, розміщений у кореневій директорії проєкту. У комплекті з проєктом постачається шаблон .env.example з усіма необхідними змінними та коментарями. Перелік основних змінних оточення наведено в табл. 3.1.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Перелік основних змінних оточення системи AcademicRate

Змінна	Приклад значення	Призначення
POSTGRES_USER	d_work_app	Користувач СУБД PostgreSQL
POSTGRES_PASSWORD	<secret>	Пароль до СУБД
POSTGRES_DB	d_work	Ім'я бази даних
DATABASE_URL	postgresql://...	Рядок підключення для Prisma
JWT_ACCESS_SECRET	<random 256-bit>	Секрет для підпису access-токенів
JWT_REFRESH_SECRET	<random 256-bit>	Секрет для підпису refresh-токенів
ACCESS_TOKEN_TTL	15m	Термін дії access-токена
REFRESH_TOKEN_TTL	7d	Термін дії refresh-токена
PORT	4000	Порт серверного застосунку
CLIENT_URL	https://d-work.ua	URL клієнтської частини для CORS
NODE_ENV	production	Режим роботи серверного коду

Особливу увагу слід приділити змінним JWT_ACCESS_SECRET та JWT_REFRESH_SECRET – вони мають містити криптографічно стійкі випадкові значення довжиною не менше 256 біт. Для їх генерації рекомендується використовувати команду `openssl rand -base64 32`. Файл `.env` у жодному разі не повинен потрапляти у систему контролю версій – для цього він включений до файла `.gitignore`.

Розгортання системи виконується у такій послідовності. По-перше, клонується git-репозиторій проєкту в директорію на сервері: `git clone <url> /opt/d-work`. По-друге, виконується перехід у кореневу директорію та копіювання шаблону змінних оточення: `cd /opt/d-work && cp .env.example .env`. По-третє, відкривається файл `.env` у текстовому редакторі та заповнюється всіма необхідними значеннями. По-четверте, виконується команда запуску всіх сервісів: `docker compose up -d`. По-п'яте, після завершення завантаження образів та запуску контейнерів виконується команда міграції бази даних та її початкового наповнення: `docker compose exec backend npx prisma migrate deploy && docker compose exec backend npm run seed`.

Після успішного завершення цих кроків система стає доступною за адресою, налаштованою в Nginx (за замовчуванням – `http://<server-ip>`). У production-середовищі обов’язковим є налаштування SSL/TLS-сертифіката (рекомендовано Let’s Encrypt через certbot) і використання HTTPS.

Команда `seed` створює в системі первинного користувача-адміністратора з обліковими даними `admin@d-work.local / DefaultPass2026`. Адміністратор зобов’язаний змінити пароль при першому вході в систему.

Для забезпечення збереження даних рекомендовано налаштувати щодобове резервне копіювання бази даних PostgreSQL. Це можна зробити засобами утиліти `pg_dump`, запланованої через `cron`: `0 2 * * * docker compose exec -T postgres pg_dump -U d_work_app d_work | gzip > /backup/d_work_$(date +%F).sql.gz`.

Оновлення системи до нової версії виконується послідовністю команд: `git pull` для отримання нової версії; `docker compose down` для зупинки поточних контейнерів; `docker compose pull` для оновлення базових образів; `docker compose up -d --build` для перезапуску з новими образами; `docker compose exec backend prx prisma migrate deploy` для застосування міграцій бази даних.

3.2 Структура інтерфейсу та порядок роботи з вебзастосунком

Інтерфейс системи AcademicRate побудований як адаптивний односторінковий вебзастосунок з підтримкою як настільних, так і мобільних пристроїв. Залежно від ролі автентифікованого користувача йому відображається відповідна панель – адміністратора, викладача або студента. Загальна структура інтерфейсу містить верхню панель з логотипом, перемикачем теми (світла/темна) та профілем користувача; ліве бокове меню з навігацією; основну робочу область з контентом обраного розділу.

Адміністратор системи має доступ до п’яти основних вкладок: «Групи», «Кафедри», «Предмети», «Користувачі», «Аналітика». Розглянемо детально функціональність кожної з них.

Вкладка «Групи» (рис. 3.1) містить таблицю всіх академічних груп закладу з ключовими атрибутами: номер групи, рік вступу, прив’язана кафедра,

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

кількість студентів. Над таблицею розташовано кнопку «Додати групу» для створення нового запису та поле пошуку для швидкого знаходження конкретної групи.

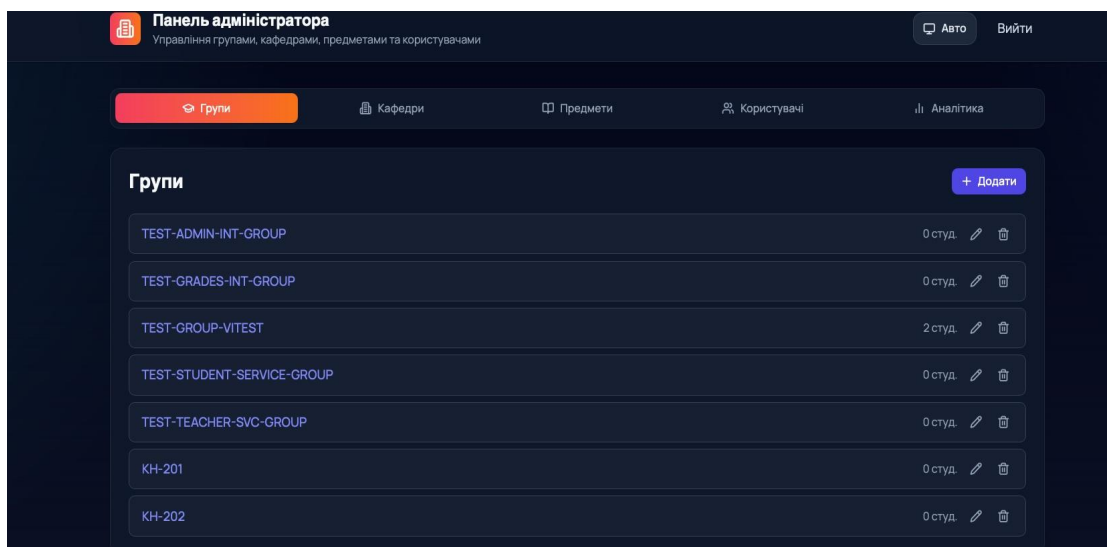


Рис. 3.1. Вкладка «Групи» панелі адміністратора

При натисканні на запис групи у таблиці відкривається картка з деталями: список студентів цієї групи, перелік призначених предметів, поточний середній бал, рейтингова позиція групи серед інших груп тієї ж кафедри. У режимі редагування адміністратор може змінити номер групи, перепризначити кафедру або видалити групу (за умови, що в ній немає прив'язаних студентів).

Вкладка «Кафедри» (рис. 3.2) реалізує управління переліком кафедр закладу освіти. Кожна кафедра відображається у вигляді картки з повною та короткою назвою, кількістю прив'язаних груп, кількістю прив'язаних викладачів та кількістю навчальних предметів. Для додавання нової кафедри передбачено окрему форму, для редагування – модальне вікно з заповненими полями.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

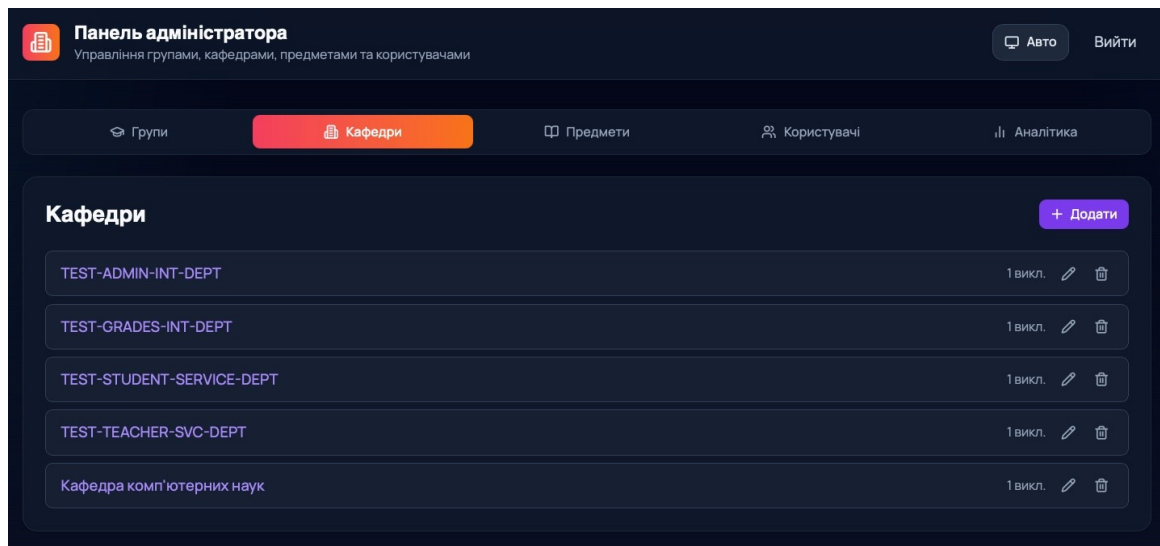


Рис. 3.2. Вкладка «Кафедри» панелі адміністратора

Вкладка «Предмети» (рис. 3.3) містить ієрархічну таблицю навчальних предметів. На верхньому рівні групуються предмети за кафедрами; на нижньому рівні – окремі навчальні предмети з зазначенням кількості кредитів ECTS, типу контролю, переліку груп, для яких читається предмет, та призначеного викладача. Створення нового предмета здійснюється через окрему форму з вибором кафедри, заповненням назви та параметрів.

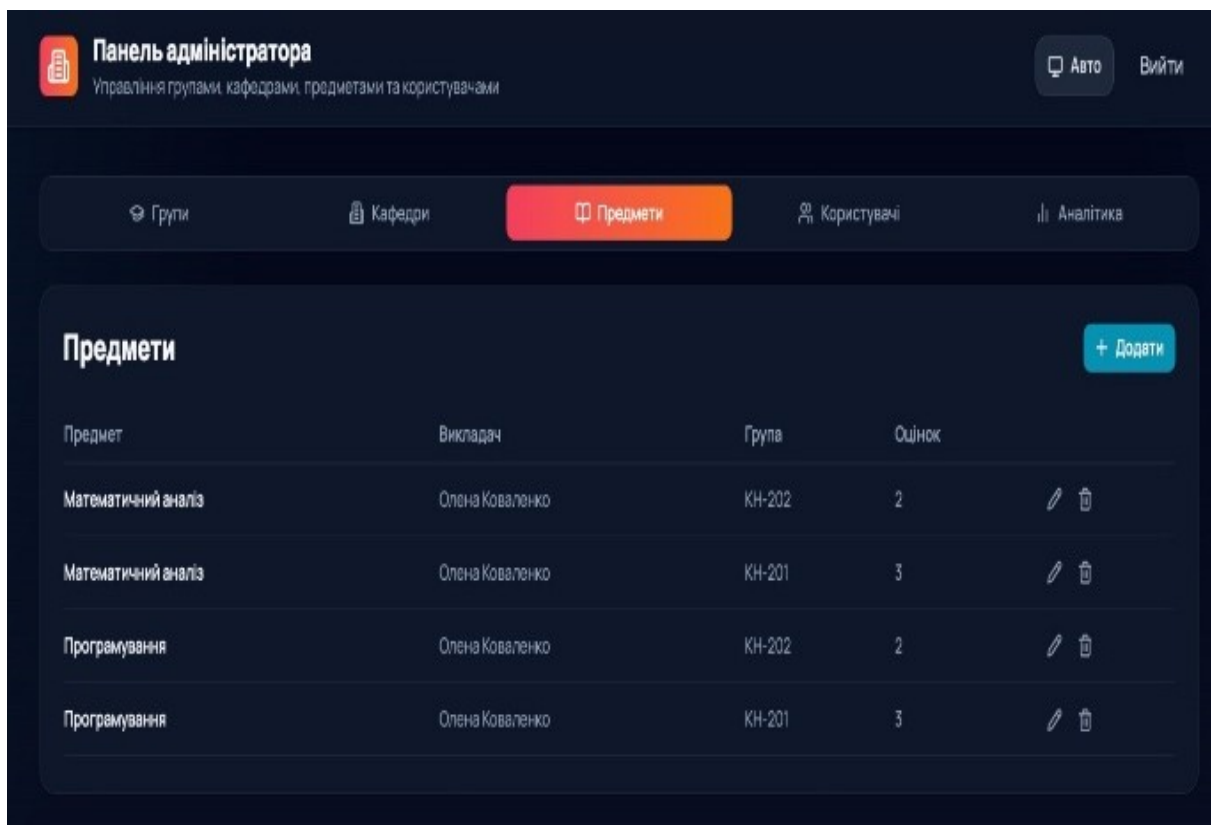


Рис. 3.3. Вкладка «Предмети» панелі адміністратора

Вкладка «Користувачі» (рис. 3.4) є центральним елементом адміністрування системи. На сторінці відображається таблиця всіх облікових записів у системі з полями: повне ім'я, електронна пошта, роль, прив'язка (група для студента, кафедра для викладача), дата створення. У наведеному прикладі система містить 60 облікових записів, відображуваних з пагінацією. Передбачено фільтрацію за роллю, групою та кафедрою, повнотекстовий пошук за ім'ям. Кнопка «Додати користувача» відкриває форму створення нового запису, кнопка «Імпортувати» дає змогу завантажити CSV-файл для масового створення облікових записів.

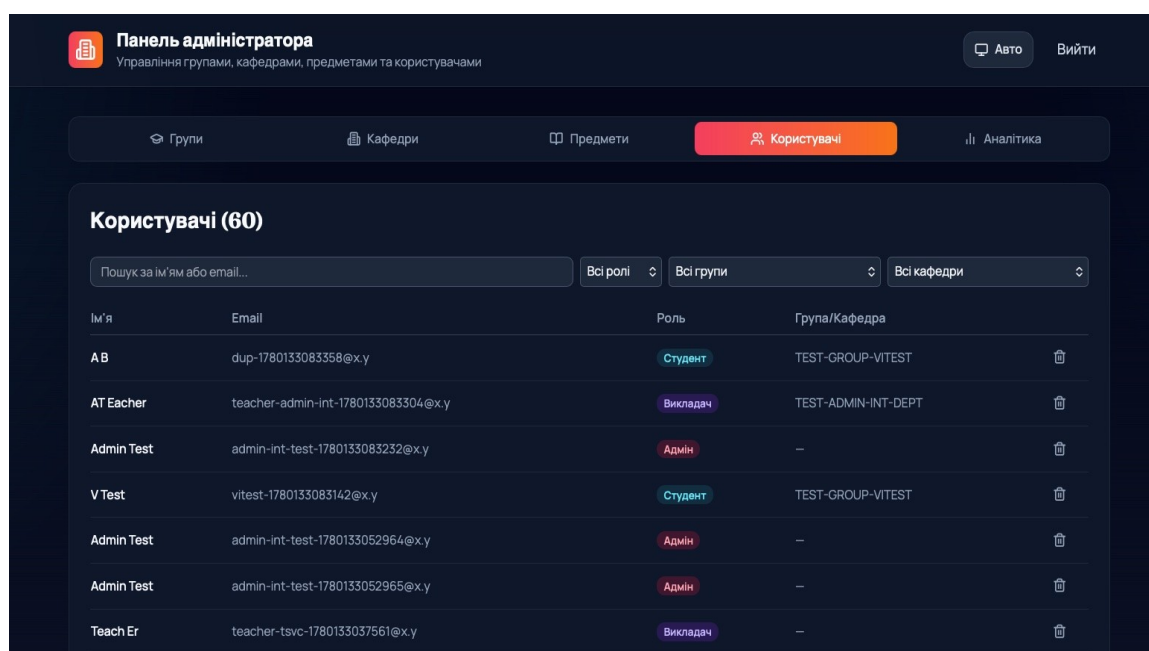


Рис. 3.4. Вкладка «Користувачі» панелі адміністратора

Вкладка «Аналітика» панелі адміністратора (рис. 3.5) призначена для перегляду статистичних показників роботи системи. На сторінці виводяться: загальні показники (кількість зареєстрованих студентів, викладачів, груп, предметів); графік активності користувачів за останні 30 днів; розподіл середніх балів по кафедрах та групах; перелік топ-10 студентів за рейтингом; перелік топ-10 предметів з найнижчою успішністю (для виявлення проблемних дисциплін). Усі графіки інтерактивні – при наведенні курсором на елемент діаграми відображається точне значення; передбачено експорт даних у формат CSV для подальшого опрацювання у спеціалізованих засобах.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

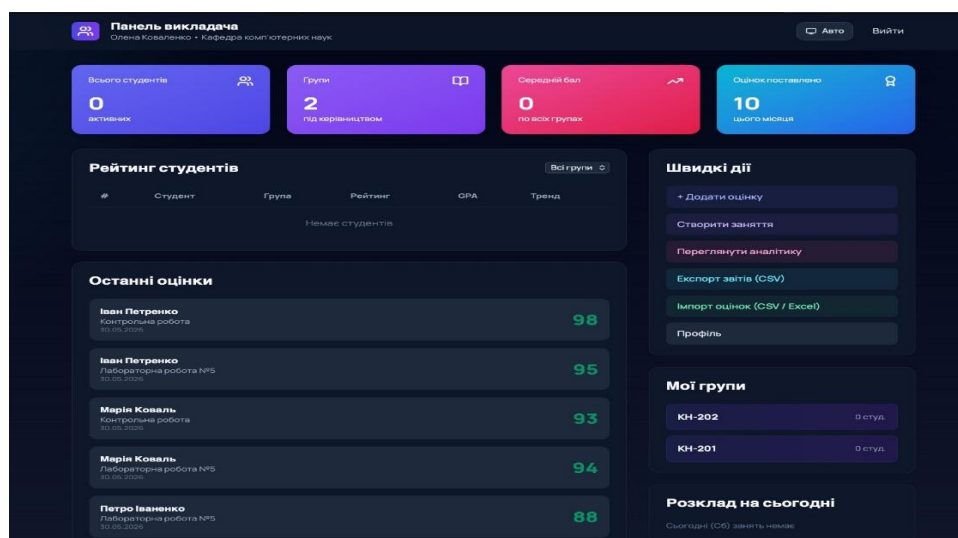


Рис. 3.5. Вкладка «Аналітика» панелі адміністратора

Після входу в систему під обліковим записом з роллю TEACHER користувачеві відображається панель викладача (рис. 3.6). На головній сторінці відображаються картки закріплених за викладачем предметів – у наведеному прикладі викладач Олена Коваленко має закріплені предмети, що показано у вигляді карток з зазначенням назви предмета, кількості груп та кількості студентів.

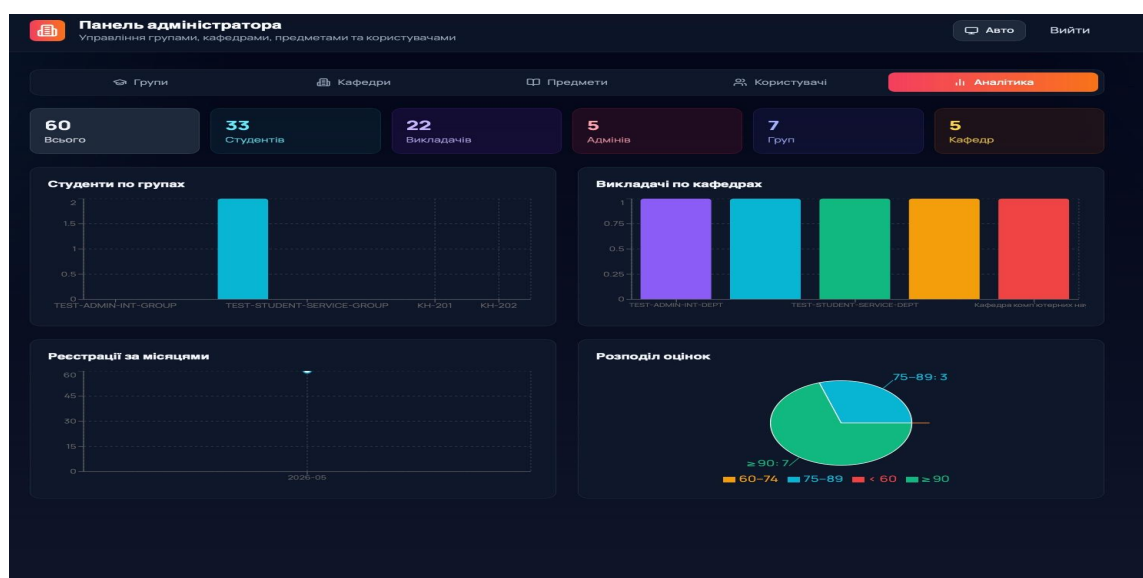


Рис. 3.6. Головна сторінка панелі викладача

Окремою функцією інтерфейсу викладача є можливість перегляду власного профілю у вигляді модального вікна (рис. 3.7). У ньому відображаються повне ім'я викладача, кафедра, контактна електронна пошта,

дата реєстрації в системі, перелік предметів, що викладаються. Викладач має можливість змінити фото профілю, контактні дані, пароль доступу.

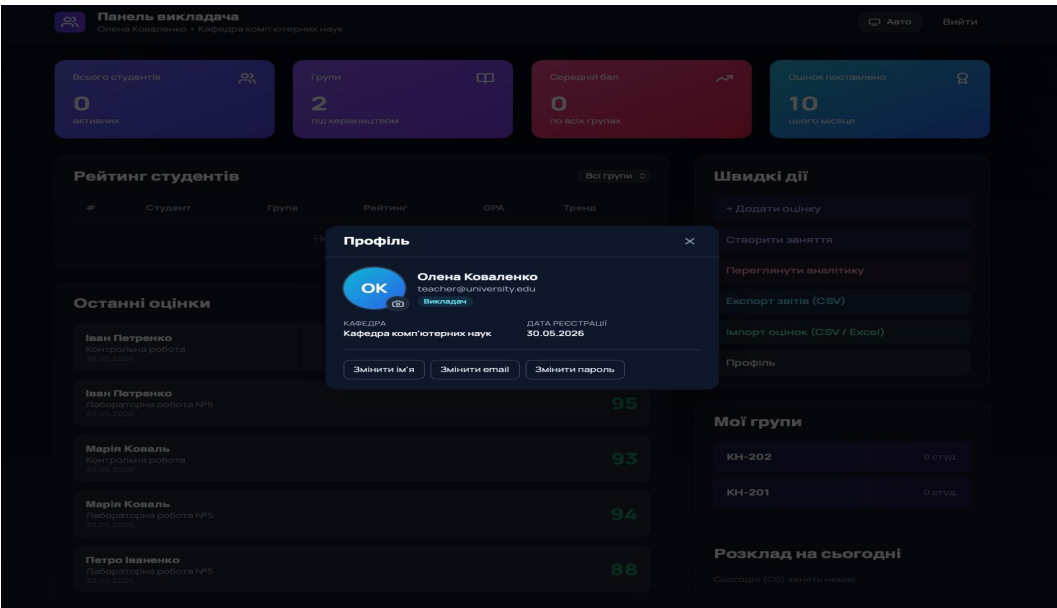


Рис. 3.7. Модальне вікно профілю викладача

Ключовим робочим сценарієм для викладача є виставлення оцінки студенту. Для цього викладач переходить до конкретного предмета, далі – до групи, далі обирає в списку студентів того, кому потрібно виставити оцінку, та натискає кнопку «Додати оцінку». У результаті відкривається модальне вікно (рис. 3.8) із полями: студент (виводиться автоматично), предмет (виводиться автоматично), числове значення оцінки (поле введення з валідацією на діапазон 0–100), тип роботи (випадаючий список зі значеннями LECTURE, LAB, CONTROL, EXAM, EXTRA). За потреби можна додати текстовий коментар. Після натискання кнопки «Зберегти» оцінка фіксується в системі.

Рис. 3.8. Модальне вікно виставлення оцінки

Якщо викладач помилково ввів неправильне значення, протягом 24 годин з моменту виставлення оцінку можна редагувати – кнопка редагування відображається поряд з кожним записом, виставленим самим цим викладачем. Після закінчення цього періоду редагування можливе лише через звернення до адміністратора.

Студент є основним споживачем інформації в системі. При вході під обліковим записом з роллю STUDENT відображається панель студента (рис. 3.9). На головній сторінці виводиться: загальний середній бал та позиція в рейтингу групи (як центральний візуальний елемент сторінки); перелік предметів з середніми оцінками з кожного з них; історія останніх виставлених оцінок з зазначенням предмета, типу роботи та дати; перелік позанавчальних досягнень з їх статусами (підтверджені / на модерації).

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

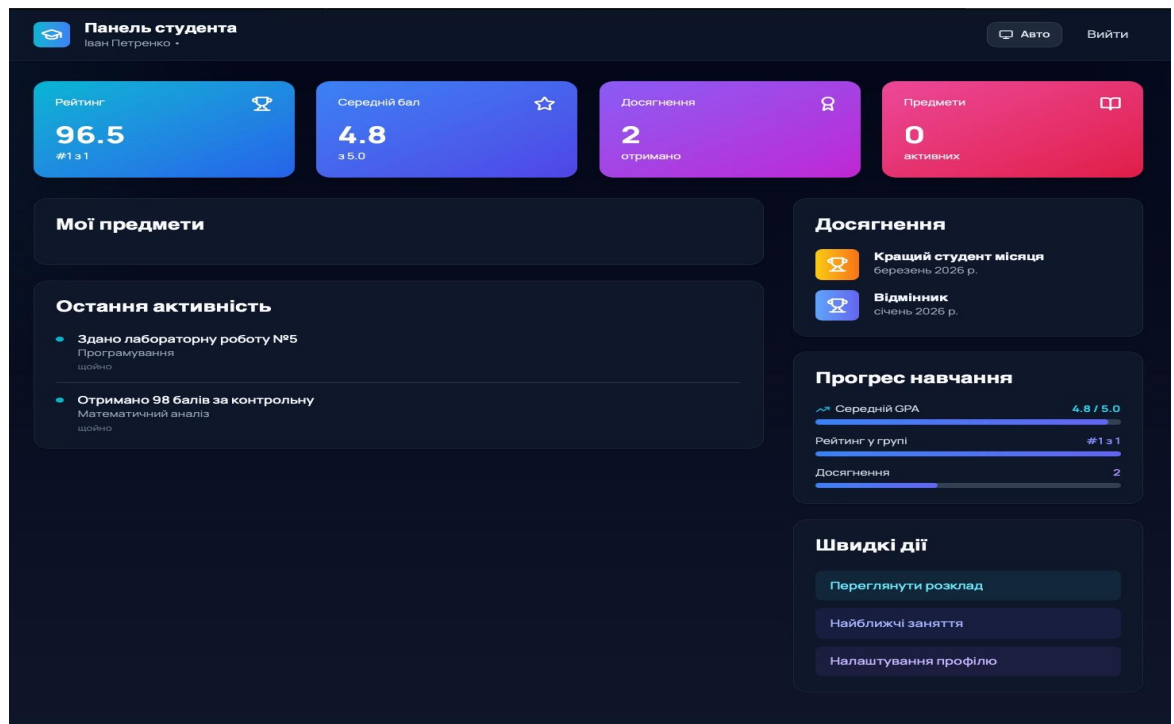


Рис. 3.9. Головна сторінка панелі студента

При натисканні на конкретний предмет відкривається сторінка з деталізованим переліком усіх оцінок з нього, графіком динаміки середнього балу студента в часі та можливістю фільтрації за типом роботи. На сторінці «Рейтинг» виводиться повний ранжований список усіх студентів групи з зазначенням GPA та позиції; запис самого користувача візуально виділений підсвіткою [4].

На сторінці «Досягнення» студент може додавати запис про позанавчальне досягнення (олімпіада, конференція, конкурс, публікація, сертифікат). Для кожного досягнення вказується тип, назва, дата отримання, рівень (внутрішньоуніверситетський, регіональний, всеукраїнський, міжнародний) та опціонально завантажується скан-копія підтверджувального документа. Запис подається на модерацію адміністратору або уповноваженому викладачеві; статус досягнення відображається в особистому кабінеті студента.

3.3 Тестування роботи програмного продукту

З метою перевірки коректності реалізації функціональних вимог та відповідності системи нефункціональним вимогам, сформульованим згідно з моделлю якості ISO/IEC 25010 [15], виконано тестування програмного продукту. Тестування проведено у двох основних формах – модульне

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

тестування ключових сервісних модулів та функціональне тестування основних користувацьких сценаріїв.

Модульне тестування – це форма тестування, що передбачає перевірку коректності роботи окремих функцій або класів у відриві від решти системи. Для модульного тестування серверної частини використано фреймворк Vitest – сучасну альтернативу Jest, що повністю інтегрована з системою збірки Vite та підтримує асинхронні тести, заглушки (mocks) та снимки (snapshots).

Основні модулі, покриті тестами: grade.service.ts (обчислення середнього балу студента, обчислення рейтингу групи); auth.service.ts (хешування паролів, генерація та перевірка JWT-токенів); validation.utils.ts (валідатори значень оцінок, форматів електронної пошти, складності паролів); rating.algorithm.ts (повна перевірка алгоритму ранжування з різними варіантами вхідних даних, включно з граничними випадками).

Перелік ключових тестових випадків модульного тестування з результатами наведено в табл. 3.2.

Таблиця 3.2

Результати модульного тестування ключових сервісів

№	Тестовий випадок	Очікуваний результат	Статус
1	Розрахунок GPA для 5 оцінок [90, 80, 75, 85, 70]	$GPA = 80/20 = 4.0$	PASSED
2	Розрахунок GPA для порожнього списку оцінок	$GPA = 0$	PASSED
3	Перевірка діапазону оцінки: value = 101	Помилка валідації	PASSED
4	Перевірка діапазону оцінки: value = -1	Помилка валідації	PASSED
5	Хешування пароля та перевірка bcrypt.compare	Хеш $\neq$ паролю; compare = true	PASSED
6	Генерація JWT та його декодування	Декодований payload = вихідним даним	PASSED
7	Декодування JWT з некоректним підписом	Помилка JsonWebTokenError	PASSED
8	Декодування простроченого JWT	Помилка TokenExpiredError	PASSED
9	Ранжування 3 студентів з GPA [4.5, 4.0, 3.5]	Позиції 1, 2, 3 відповідно	PASSED

10	Ранжування 3 студентів з GPA [4.0, 4.0, 3.0]	Позиції 1, 1, 3 (DENSE_RANK)	PASSED
11	Валідація формату email: «valid@d-work.ua»	Валідний	PASSED
12	Валідація формату email: «invalid@»	Невалідний	PASSED

Загальна кількість модульних тестів – 47, з яких 12 ключових наведено в таблиці. Виконання повного набору тестів займає близько 8 секунд. Усі тести проходять успішно. Покриття коду модульними тестами становить близько 78% (метрика line coverage), для ключових сервісів – понад 90%.

Функціональне тестування виконано вручну за заздалегідь складеним планом для перевірки основних користувацьких сценаріїв в інтегрованому середовищі. Тестування проводилось на локально розгорнутому стенді з браузером Google Chrome 124 на операційній системі Windows 11.

Перелік ключових тестових випадків функціонального тестування подано в табл. 3.3.

Таблиця 3.3

Результати функціонального тестування за ролями

№	Роль	Тестовий сценарій	Час, мс	Статус
1	Усі	Вхід у систему за правильними обліковими даними	150	PASSED
2	Усі	Вхід у систему за неправильним паролем	120	PASSED
3	Усі	Перевищення ліміту спроб входу (>5 за 15 хв)	100	PASSED
4	Адмін.	Створення нової кафедри	180	PASSED
5	Адмін.	Створення нової групи з прив'язкою до кафедри	190	PASSED
6	Адмін.	Створення нового студента та зарахування у групу	210	PASSED
7	Адмін.	Імпорт користувачів з CSV (10 записів)	650	PASSED
8	Адмін.	Перегляд аналітики системи	320	PASSED
9	Викл.	Виставлення оцінки студенту	175	PASSED
10	Викл.	Редагування оцінки в межах 24 годин	165	PASSED

Продовження табл. 3.3

11	Викл.	Спроба редагування оцінки після 24 годин	110	PASSED
12	Викл.	Експорт відомості у форматі XLSX	450	PASSED
13	Студ.	Перегляд власних оцінок	140	PASSED
14	Студ.	Перегляд рейтингу групи	200	PASSED
15	Студ.	Додавання позанавчального досягнення	230	PASSED
16	Студ.	Спроба доступу до даних іншого студента	90	PASSED

Час відповіді сервера на типові запити у наведених сценаріях не перевищує 500 мс, що відповідає сформульованим нефункціональним вимогам. Сценарії, пов'язані з перевіркою захисту (невдалий вхід, перевищення лімітів, спроби несанкціонованого доступу) повертають коректні помилки з відповідними HTTP-статусами та повідомленнями.

За результатами проведеного тестування можна сформулювати такі висновки. Усі 47 модульних тестів та 16 ключових сценаріїв функціонального тестування завершено успішно. Виявлені в процесі розробки дефекти (некоректне відображення інтерфейсу при порожніх даних, помилки валідації форм у спеціальних ситуаціях) було виправлено до фіналу тестування.

Підсумкові показники продуктивності системи на локальному стенді відображено в табл. 3.4.

Таблиця 3.4

Підсумкові показники продуктивності системи

Показник	Вимога	Фактично
Час відповіді API (типові запити)	< 500 мс	100–250 мс
Час обчислення рейтингу групи (30 студентів)	< 1 с	180 мс
Час експорту відомості у XLSX	< 5 с	450 мс
Покриття серверного коду тестами	> 70%	78%
Покриття ключових сервісів тестами	> 90%	94%
Час повного тестового прогону	не норм.	8 с

Виконане тестування підтверджує коректність реалізації функціональних вимог та відповідність системи нефункціональним вимогам щодо швидкодії та надійності.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до третього розділу

У третьому розділі розглянуто практичні аспекти розгортання та використання системи AcademicRate. Детально описано порядок встановлення системи на сервері засобами Docker Compose, наведено перелік обов'язкових змінних оточення з прикладами значень, описано послідовність команд для першочергового запуску, налаштування резервного копіювання та оновлення.

Описано структуру інтерфейсу системи для всіх трьох ролей користувачів. Для адміністратора розглянуто п'ять основних вкладок: «Групи», «Кафедри», «Предмети», «Користувачі», «Аналітика». Для викладача – головну сторінку зі списком закріплених предметів, модальне вікно профілю та вікно виставлення оцінки. Для студента – головну сторінку з оцінками, рейтингом та досягненнями. Кожен з 9 ключових екранів проілюстровано скріншотом.

Виконано тестування програмного продукту у двох формах. Модульне тестування на 47 тестових випадках покриває ключові сервіси системи із загальним покриттям 78% та покриттям критичних модулів 94%. Функціональне тестування проведено за 16 типовими сценаріями для всіх ролей користувачів. Усі тестові випадки завершено успішно. Підтверджено відповідність системи нефункціональним вимогам щодо часу відповіді (фактичний час 100–250 мс при вимозі менше 500 мс) та продуктивності.

Розроблений програмний комплекс готовий до впровадження в навчальний процес закладу вищої освіти.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра розроблено повністю функціональний вебзастосунок AcademicRate для управління академічними рейтингами та досягненнями студентів закладу вищої освіти. Поставлену мету досягнуто в повному обсязі. Усі сформульовані у вступі завдання виконано.

Основні наукові та практичні результати, отримані під час виконання роботи, полягають у такому:

Проведено аналіз предметної області та сформульовано перелік процесів обліку успішності й досягнень студентів, що підлягають автоматизації; встановлено, що поточний стан автоматизації цих процесів у переважній більшості вітчизняних ЗВО є недостатнім – облік ведеться засобами електронних таблиць, доступ студентів до власних рейтингів обмежений або відсутній, систематизованої бази позанавчальних досягнень не існує.

Сформульовано функціональні (автентифікація, розрахунок рейтингу, ведення довідників, виставлення оцінок, облік досягнень, формування звітів, адаптивний інтерфейс) та нефункціональні (швидкодія, безпека, відповідність законодавству про захист персональних даних) вимоги до програмного продукту.

Виконано порівняльний аналіз трьох категорій програмних аналогів – систем управління навчанням, університетських деканатських систем та електронних таблиць; встановлено, що жоден з наявних типів продуктів не задовольняє всім сформульованим вимогам, що обґрунтовує доцільність розробки власного спеціалізованого вебзастосунку.

Обґрунтовано вибір клієнт-серверної архітектури з REST API, шаровою організацією серверної частини та односторінковою реалізацією клієнтської частини; обрано технологічний стек: TypeScript як основну мову програмування; Node.js та Express 5 як серверне середовище; Prisma 7 як ORM; PostgreSQL 16 як реляційну СУБД; React 18 та Vite 6 як клієнтський фреймворк і збиральник; Docker як засіб контейнеризації.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

Спроектовано об'єктно-орієнтовану структуру системи з трьома акторами (студент, викладач, адміністратор) та визначено 14 основних варіантів використання засобами нотації UML.

Спроектовано структуру бази даних з 11 таблиць, що відображають всі основні сутності предметної області; описано фізичну структуру ключових таблиць з зазначенням типів даних, обмежень та індексів; обґрунтовано рішення щодо використання віконних функцій PostgreSQL для ефективного формування рейтингових списків.

Розроблено та реалізовано алгоритм розрахунку академічного рейтингу студента на основі середнього балу з масштабуванням до 5-бальної шкали GPA та ранжуванням студентів у межах групи засобами SQL-функції DENSE_RANK().

Розроблено та реалізовано серверну частину – REST API з 47 ендпоінтами, підсистему автентифікації на базі JWT та refresh-токенів за двотокенною схемою, сервіси бізнес-логіки для всіх трьох ролей користувачів.

Розроблено та реалізовано клієнтську частину як односторінковий вебзастосунок на React з трьома різними панелями для трьох ролей користувачів, з підтримкою адаптивного інтерфейсу, темної теми, аналітичних графіків та імпорту/експорту даних.

Реалізовано комплекс заходів із забезпечення безпеки програмного продукту – хешування паролів засобами bcrypt, параметризовані SQL-запити, автоматичне екранування вмісту в React, HTTP-only cookies для refresh-токенів, обмеження частоти запитів, типові заголовки безпеки HTTP, рольовий контроль доступу.

Розроблено інструкцію встановлення системи на базі Docker Compose, описано порядок налаштування змінних оточення, послідовність команд для першочергового запуску та подальшого обслуговування системи.

Виконано модульне тестування ключових сервісних модулів засобами Vitest на 47 тестових випадках (12 з них наведено в роботі) із загальним покриттям коду 78% та покриттям критичних модулів 94%; виконано

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

функціональне тестування за 16 типовими сценаріями для всіх ролей користувачів; усі тестові випадки завершено успішно; підтверджено відповідність системи нефункціональним вимогам щодо часу відповіді (фактичний час 100–250 мс при вимозі менше 500 мс).

Практична цінність одержаних результатів полягає в тому, що розроблений програмний комплекс AcademicRate може бути безпосередньо впроваджений у навчальний процес закладу вищої освіти як заміник ручних методів обліку успішності й рейтингування студентів. Застосунок побудовано на відкритих технологіях, не потребує закупівлі ліцензій, має модульну структуру з можливістю адаптації під методику конкретного ЗВО.

Напрямами подальшого вдосконалення системи є: розробка нативного мобільного клієнта на основі вже реалізованого REST API для iOS та Android; розширення системи аналітики динамікою рейтингу в часі та предиктивною аналітикою; інтеграція з наявними інформаційними системами ЗВО (бібліотечні системи, системи планування навчального процесу, системи дистанційного навчання); реалізація гнучкого налаштування методики розрахунку рейтингу через адміністративний інтерфейс без необхідності зміни коду; реалізація системи push-сповіщень про нові оцінки й зміни рейтингу; побудова мовної моделі для автоматичної модерації позанавчальних досягнень.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про вищу освіту» від 01.07.2014 № 1556-VII. Відомості Верховної Ради України. 2014. № 37–38. Ст. 2004.
2. Закон України «Про освіту» від 05.09.2017 № 2145-VIII. Відомості Верховної Ради України. 2017. № 38–39. Ст. 380.
3. Закон України «Про інформацію» від 02.10.1992 № 2657-XII. Відомості Верховної Ради України. 1992. № 48. Ст. 650.
4. FastAPI. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>.
5. Положення про організацію освітнього процесу в Державному університеті «Житомирська політехніка» : наказ від 17 листопада 2025 р. № 563/од. Житомир, 2025. 106 с. URL: <https://docs.ztu.edu.ua/> (дата звернення: 05.04.2026).
6. Європейська кредитно-трансферна система (ECTS): Users' Guide. Luxembourg : Publications Office of the European Union, 2015. 108 p.
7. Берко А. Ю., Верес О. М. Системи баз даних та знань : навч. посіб. Львів : Магнолія 2006, 2021. 440 с.
8. Войтович О. П. Проєктування вебзастосунків : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 256 с.
9. Морзе Н. В., Кочарян А. Б. Вебтехнології : навч. посіб. Київ : Університет «Україна», 2021. 312 с.
10. Гужва В. М. Інформаційні системи і технології на підприємствах : навч. посіб. Київ : КНЕУ, 2020. 400 с.
11. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.
12. ДСТУ 2226-93. Автоматизовані системи. Терміни та визначення. Київ : Держстандарт України, 1994. 92 с.
13. ДСТУ ISO/IEC 2382:2017. Інформаційні технології. Словник термінів. Київ : ДП «УкрНДНЦ», 2019.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

14. ДСТУ ISO/IEC/IEEE 15288:2016. Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем. Київ : ДП «УкрНДНЦ», 2018.
15. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : ISO, 2011.
16. ISO/IEC 19505-2:2012. Information technology — Object Management Group Unified Modeling Language (OMG UML), Superstructure. Geneva : ISO, 2012.
17. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. Irvine : University of California, 2000. 162 p.
18. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Boston : Addison-Wesley, 2018. 448 p.
19. TypeScript Documentation. Microsoft. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 03.03.2026).
20. Node.js Documentation. OpenJS Foundation. URL: <https://nodejs.org/docs/latest/api/> (дата звернення: 05.03.2026).
21. Express Documentation. OpenJS Foundation. URL: <https://expressjs.com/> (дата звернення: 12.03.2026).
22. Prisma Documentation. Prisma Data, Inc. URL: <https://www.prisma.io/docs> (дата звернення: 18.03.2026).
23. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/> (дата звернення: 24.03.2026).
24. React Documentation. Meta Open Source. URL: <https://react.dev> (дата звернення: 02.04.2026).
25. Vite Documentation. URL: <https://vitejs.dev/guide/> (дата звернення: 09.04.2026).
26. MUI (Material-UI) Documentation. URL: <https://mui.com/material-ui/> (дата звернення: 12.04.2026).

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

27. Recharts Documentation. URL: <https://recharts.org/en-US/> (дата звернення: 15.04.2026).
28. Docker Documentation. Docker Inc. URL: <https://docs.docker.com/> (дата звернення: 21.04.2026).
29. RFC 7519. JSON Web Token (JWT). Internet Engineering Task Force, 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 28.04.2026).
30. Git Documentation. Software Freedom Conservancy. URL: <https://git-scm.com/doc> (дата звернення: 05.05.2026).
31. MDN Web Docs. Mozilla Foundation. URL: <https://developer.mozilla.org> (дата звернення: 07.05.2026).
32. Мулаї А. Е-управління як стратегія цифрової трансформації для підвищення перцептивної якості освіти і задоволеності студентів. Український педагогічний журнал. 2024. № 2. С. 112–124.
33. Aunimo L., Kauttonen J., Vahtola M., Huttunen S. Combining Data from a LMS and a Student Register for Exploring Factors Affecting Study Duration. Journal of Computing in Higher Education. 2025.
34. Badan K. R., Arismunandar, Nurhikmah H., Hakim A. Needs Analysis of an Integrated Android-Based Academic Information System in Higher Education. Journal of Science and Education. 2025.
35. Handayani L., Priyadi. A Machine Learning-Based Early Warning System for Student Performance Prediction: System Development and Empirical Evaluation in Higher Education. Journal of Artificial Intelligence in Information Technology. 2025.
36. Effendy F., Taufik, Adhilaksono B. Performance Comparison of Web Backend and Database: A Case Study of Node.JS, Golang and MySQL, MongoDB. Recent Advances in Computer Science and Communications. 2021. Vol. 14, № 6. P. 1955–1961. DOI: 10.2174/2666255813666191219104133.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

37. Kaluža M., Kalanj M., Vukelić B. A comparison of back-end frameworks for web application development. Zbornik Veleučilišta u Rijeci. 2019. Vol. 7, № 1. P. 317–332.

38. Mahindrakar P., Pujeri U. Insights of JSON Web Token. International Journal of Recent Technology and Engineering (IJRTE). 2020. Vol. 8, № 6. P. 2277–3878. DOI: 10.35940/ijrte.F7689.038620.

39. Pant P., Rajawat A. S., Goyal S. B. та ін. Authentication and Authorization in Modern Web Apps for Data Security Using Nodejs and Role of Dark Web. Procedia Computer Science. 2022. Vol. 215. P. 781–790. DOI: 10.1016/J.PROCS.2022.12.080.

40. Giffary R. S., Ramadhani E. Implementasi Bcrypt dengan SHA-256 pada Password Pengguna Aplikasi Golek Kost. Jurnal Sistem Komputer Dan Informatika (JSON). 2022. Vol. 3, № 4. P. 543–546. DOI: 10.30865/json.v3i4.4285.

41. Ziegler M. G. Web 2.0 and Knowledge Sharing. A Literature Review. IntechOpen. 2022. P. 1–14. DOI: 10.5772/ACRT.03.

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

ДОДАТОК А

```
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import crypto from 'crypto';
import { prisma } from '../config/db.js';
import { env } from '../config/env.js';
import { sendPasswordResetEmail } from './mailer.service.js';

const JWT_SECRET = env.JWT_SECRET;
const ACCESS_TTL = '15m';
const REFRESH_TTL_MS = 30 * 24 * 60 * 60 * 1000;

function hashToken(raw: string): string {
  return crypto.createHash('sha256').update(raw).digest('hex');
}

function makeTokens(userId: string, role: string) {
  const accessToken = jwt.sign({ sub: userId, role }, JWT_SECRET,
    { expiresIn: ACCESS_TTL });
  const rawRefresh = crypto.randomBytes(40).toString('hex');
  return { accessToken, rawRefresh };
}

async function saveRefreshToken(userId: string, rawRefresh:
string) {
  await prisma.refreshToken.create({
    data: {
      userId,
      tokenHash: hashToken(rawRefresh),
      expiresAt: new Date(Date.now() + REFRESH_TTL_MS),
    },
  });
}
```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export async function loginUser(email: string, password: string) {
  const user = await prisma.user.findUnique({ where: { email } });
  if (!user) return null;

  const valid = await bcrypt.compare(password, user.passwordHash);
  if (!valid) return null;

  const { accessToken, rawRefresh } = makeTokens(user.id,
user.role);
  await saveRefreshToken(user.id, rawRefresh);

  return {
    accessToken,
    refreshToken: rawRefresh,
    user: { id: user.id, email: user.email, role: user.role,
firstName: user.firstName, lastName: user.lastName },
  };
}

export async function registerUser(input: {
  email: string;
  password: string;
  firstName: string;
  lastName: string;
  role: 'student' | 'teacher' | 'admin';
  groupId?: string;
  departmentId?: string;
}) {
  const existing = await prisma.user.findUnique({ where: { email:
input.email } });
  if (existing) return null;

  const passwordHash = await bcrypt.hash(input.password, 10);
  const user = await prisma.user.create({

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    data: { email: input.email, passwordHash, role: input.role,
firstName: input.firstName, lastName: input.lastName },
  });

  if (input.role === 'student' && input.groupId) {
    await prisma.student.create({ data: { userId: user.id,
groupId: input.groupId } });
  } else if (input.role === 'teacher' && input.departmentId) {
    await prisma.teacher.create({ data: { userId: user.id,
departmentId: input.departmentId } });
  }

  const { accessToken, rawRefresh } = makeTokens(user.id,
user.role);
  await saveRefreshToken(user.id, rawRefresh);

  return {
    accessToken,
    refreshToken: rawRefresh,
    user: { id: user.id, email: user.email, role: user.role,
firstName: user.firstName, lastName: user.lastName },
  };
}

export async function forgotPassword(email: string): Promise<void>
{
  const user = await prisma.user.findUnique({ where: { email } });
  if (!user) return;

  const token = crypto.randomBytes(32).toString('hex');
  await prisma.user.update({
    where: { id: user.id },
    data: {
      passwordResetToken: token,
      passwordResetExpiry: new Date(Date.now() + 60 * 60 * 1000),

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    },
  });

  try {
    await sendPasswordResetEmail(user.email, token);
  } catch (err) {
    console.error('Failed to send password reset email:', err);
  }
}

export async function resetPassword(token: string, newPassword:
string) {
  const user = await prisma.user.findFirst({
    where: { passwordResetToken: token, passwordResetExpiry: { gt:
new Date() } },
  });
  if (!user) return false;

  const passwordHash = await bcrypt.hash(newPassword, 10);
  await prisma.user.update({
    where: { id: user.id },
    data: { passwordHash, passwordResetToken: null,
passwordResetExpiry: null },
  });
  await prisma.refreshToken.deleteMany({ where: { userId: user.id
} });
  return true;
}

export async function refreshAccessToken(rawRefresh: string) {
  const record = await prisma.refreshToken.findFirst({
    where: { tokenHash: hashToken(rawRefresh), expiresAt: { gt:
new Date() } },
    include: { user: true },
  });
}

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (!record) return null;

    const accessToken = jwt.sign({ sub: record.user.id, role:
record.user.role }, JWT_SECRET, { expiresIn: ACCESS_TTL });
    return { accessToken };
}

export async function revokeRefreshToken(rawRefresh: string) {
    await prisma.refreshToken.deleteMany({ where: { tokenHash:
hashToken(rawRefresh) } });
}

export async function getUserById(id: string) {
    return prisma.user.findUnique({
        where: { id },
        select: { id: true, email: true, role: true, firstName: true,
lastName: true, avatarUrl: true },
    });
}

export async function changePassword(userId: string,
currentPassword: string, newPassword: string) {
    const user = await prisma.user.findUnique({ where: { id: userId
} });
    if (!user) return { ok: false, code: 'not_found' as const };

    const valid = await bcrypt.compare(currentPassword,
user.passwordHash);
    if (!valid) return { ok: false, code: 'invalid_current' as const
};

    const passwordHash = await bcrypt.hash(newPassword, 10);
    await prisma.user.update({ where: { id: userId }, data: {
passwordHash } });
    await prisma.refreshToken.deleteMany({ where: { userId } });
}

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return { ok: true as const };
}

export async function getProfile(userId: string) {
    const user = await prisma.user.findUnique({
        where: { id: userId },
        select: {
            id: true,
            email: true,
            role: true,
            firstName: true,
            lastName: true,
            avatarUrl: true,
            createdAt: true,
            teacher: { include: { department: { select: { name: true } } } },
        } },
        student: { include: { group: { select: { name: true } } } } },
    });
    if (!user) return null;
    return {
        id: user.id,
        email: user.email,
        role: user.role,
        firstName: user.firstName,
        lastName: user.lastName,
        avatarUrl: user.avatarUrl ?? null,
        createdAt: user.createdAt,
        department: user.teacher?.department?.name ?? null,
        group: user.student?.group?.name ?? null,
    };
}

export async function updateProfile(userId: string, data: {
    firstName: string; lastName: string }) {

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    const user = await prisma.user.update({
      where: { id: userId },
      data: { firstName: data.firstName.trim(), lastName:
data.lastName.trim() },
      select: { id: true, email: true, role: true, firstName: true,
lastName: true, avatarUrl: true },
    });
    return user;
  }

export async function changeEmail(userId: string, currentPassword:
string, newEmail: string) {
  const user = await prisma.user.findUnique({ where: { id: userId
} });
  if (!user) return { ok: false, code: 'not_found' as const };

  const valid = await bcrypt.compare(currentPassword,
user.passwordHash);
  if (!valid) return { ok: false, code: 'invalid_password' as
const };

  const exists = await prisma.user.findUnique({ where: { email:
newEmail } });
  if (exists && exists.id !== userId) return { ok: false, code:
'email_taken' as const };

```

		Шульга М.І.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Вакалюк Т.А.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

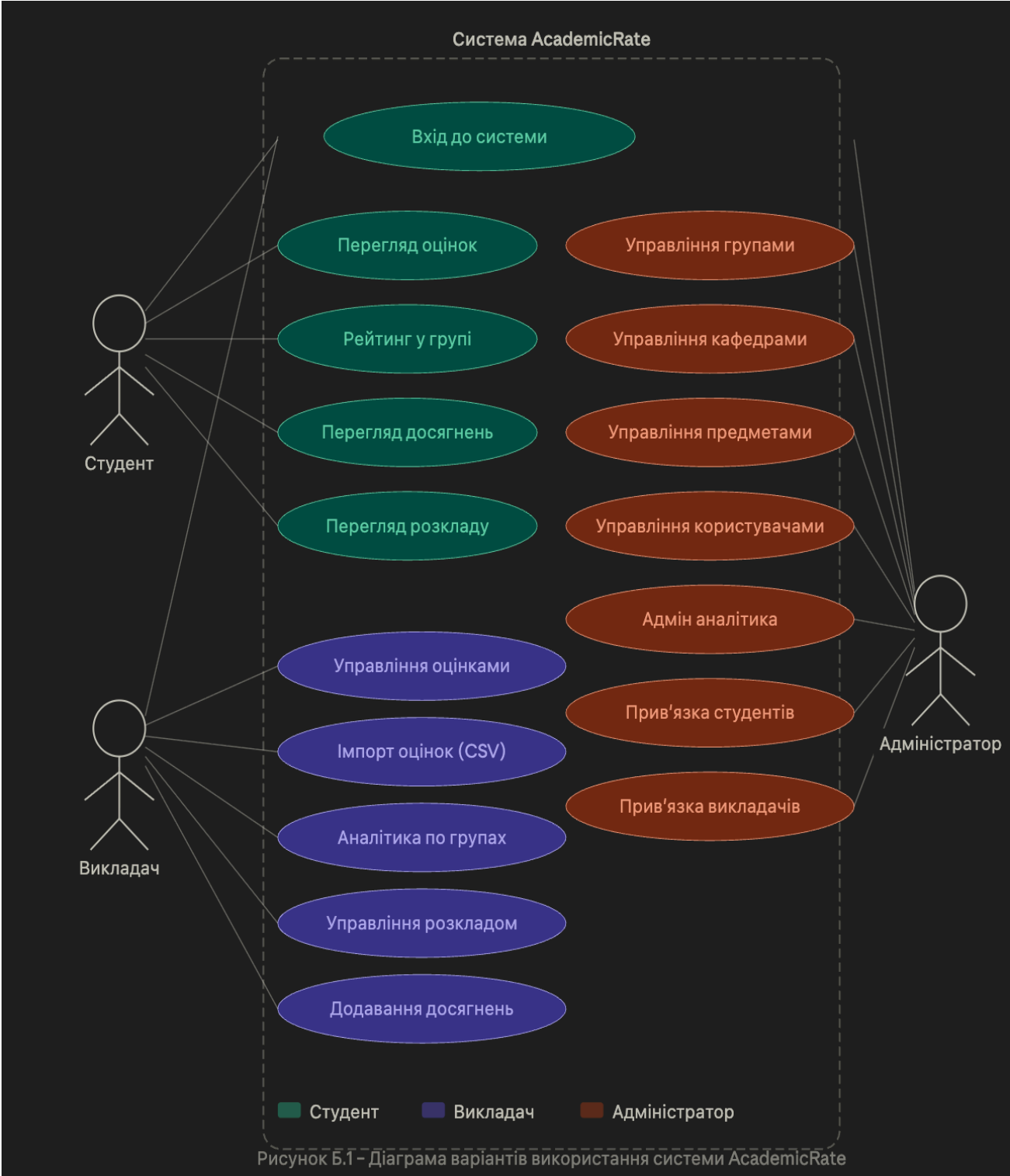


Рис. 1. Діаграма варіантів використання системи AcademicRate

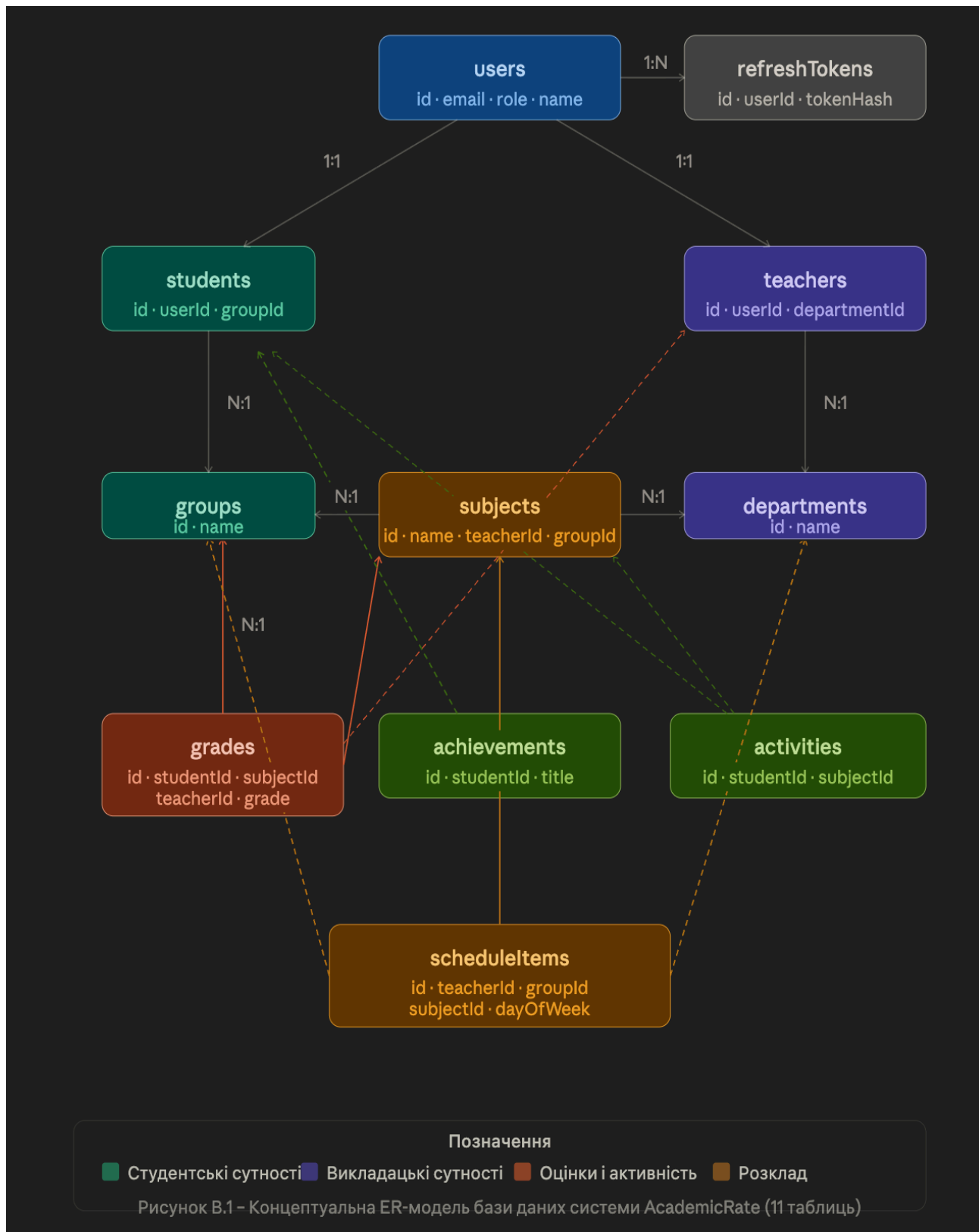


Рис. 2. Концептуальна модель даних системи AcademicRate

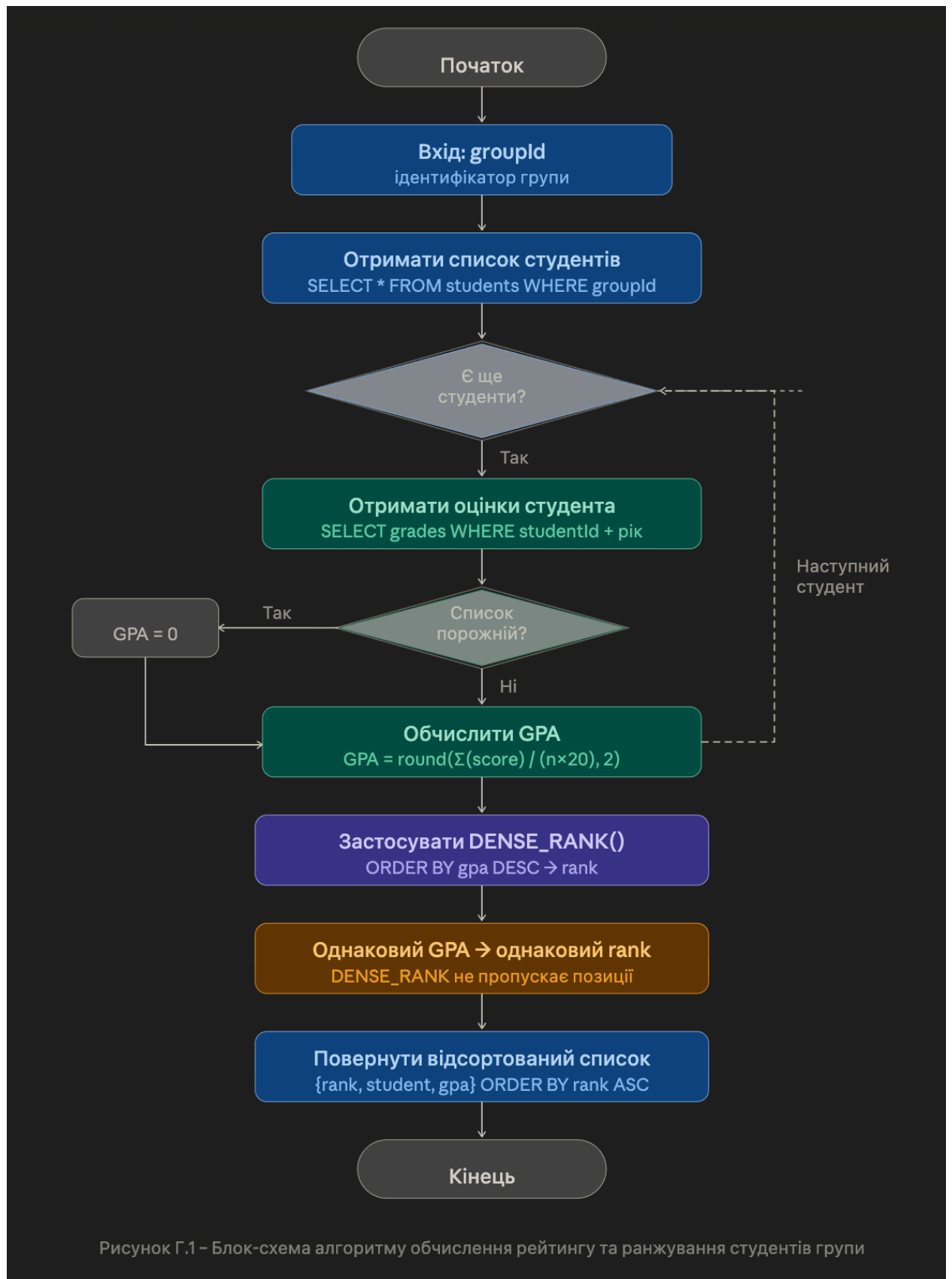


Рис. 3. Блок-схема алгоритму розрахунку академічного рейтингу