Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ВВЕДЕННЯ ТА РЕДАГУВАННЯ РОЗКЛАДУ НАВЧАЛЬНИХ ЗАНЯТЬ

DEVELOPMENT OF AN INFORMATIONSYSTEM FOR ENTERING AND EDITINGTHE SCHEDULE OF TRAINING SESSIONS.

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Тимченко Сергій Валерійович
_____
(підпис)
Керівник: д.пед.н., професор
Тулашвілі Юрій Йосипович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Валерій ЛІЩИНА
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Тимченко Сергій Валерійович

1. Тема кваліфікаційної роботи: *Розробка інформаційної системи введення та редагування розкладу навчальних занять*

Керівник роботи: *д.пед.н., професор Тулашвілі Ю.Й.*

затверджені наказом закладу вищої освіти від «14» травня 2025 р. № 255/01-02

2. Строк подання здобувачем роботи: 05.12.2025 р.

3.Вихідні дані до роботи: *Документація та специфікації стандартів і протоколів web-розробки, Технічна документація сучасних web-технологій та фреймворків (Next.js, Prisma ORM, PostgreSQL), Керівництва з проектування та реалізації баз даних, Матеріали конференцій та семінарів щодо автоматизації освітнього процесу, Дані про структуру навчального процесу коледжу (групи, викладачі, аудиторії, дисципліни)*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): *Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи.*

5. Перелік графічного матеріалу: *Архітектура системи ProBook, Схема бази даних, Інтерфейс користувача, Таблиці порівняльного аналізу з існуючими рішеннями*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Розділ 1 Аналіз проблематики за темою роботи та постановка завдань дослідження	*Тулашвілі Ю. Й.*		
Розділ 2 Теоретичне дослідження та практична реалізація інформаційної системи введення та редагування розкладу навчальних занять	*Тулашвілі Ю. Й.*		
Розділ 3 Експерементальне дослідження результативності інформаційної системи введення та редагування розкладу	*Тулашвілі Ю. Й.*		
Показник запозичень тексту	_____ *%*		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 30.06.2025 р*	
2.	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 01.09.2025 р.*	
3.	*Розробити функціональну схему роботи програмного продукту*	*до 01.10.2025 р*	
4.	*Описати засоби розробки об'єкта проектування*	*до 15.10.2025 р.*	
5.	*Практична реалізація об'єкта проектування*	*до 10.11.2025 р*	
6.	*Провести аналіз результатів експерименту*	*до 25.11.2025 р.*	
7.	*Здача чистового варіанту кваліфікаційної роботи на кафедру*	*до 05.12.2025 р.*	

Здобувач вищої освіти __ _____ Сергій ТИМЧЕНКО

Керівник роботи _____ Юрій ТУЛАШВІЛІ

АНОТАЦІЯ

Тимченко С. В. Розробка інформаційної системи введення та редагування розкладу навчальних занять. Кваліфікаційна робота магістра за спеціальністю 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

У роботі проведено аналіз існуючих підходів до автоматизації складання розкладу, обґрунтовано вибір технологічного стеку, спроектовано архітектуру системи та реляційну базу даних з 15 таблицями. Реалізовано повний функціонал системи ProBook з 8 модулями, включаючи аутентифікацію через NextAuth.js, управління розкладом з перевіркою конфліктів та RESTful API для інтеграції. Проведено комплексне тестування: оцінювання за метриками Core Web Vitals, аудит безпеки OWASP ZAP, навантажувальне тестування Apache JMeter (стабільна робота до 50 користувачів) та функціональне тестування методом pairwise (16 кейсів, 48 автотестів).

Ключові слова: розклад занять, інформаційна система, Next.js, Prisma ORM, web-додаток, TypeScript, автоматизація освітнього процесу, коледж.

ABSTRACT

Tymchenko Serhii. Development of an Information System for Scheduling and Editing Academic Classes. Master's qualification work in specialty 122 «Computer Science», educational program «Computer Science». Lutsk National Technical University. Lutsk, 2025.

The work analyzes existing approaches to schedule automation, justifies the choice of technology stack, designs system architecture and a relational database with 15 tables. The complete functionality of the ProBook system was implemented with 8 modules, including authentication via NextAuth.js, schedule management with conflict checking, and RESTful API for integration. Comprehensive testing was conducted: evaluation using Core Web Vitals metrics, OWASP ZAP security audit, load testing with Apache JMeter (stable operation up to 50 users), and functional testing using the pairwise method (16 cases, 48 automated tests).

Keywords: class schedule, information system, Next.js, Prisma ORM, web application, TypeScript, educational process automation, college.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Розклад навчальних занять є критичним компонентом організації освітнього процесу в навчальних закладах. Традиційні методи складання розкладу, засновані на ручному розподілі ресурсів, потребують значних часових витрат (3-4 дні) і мають низький рівень оптимізації. Створення web-орієнтованої інформаційної системи управління розкладом з використанням сучасних технологій стає необхідністю для професійної автоматизації освітнього процесу в закладах фахової передвищої освіти.

Сучасний стан проблеми. Існуючі комерційні рішення (UniTime, 1С Автоматичне планування, система НІКА) мають обмежену адаптованість до специфіки коледжів з груповою формою навчання. Освітня екосистема Мрія орієнтована переважно на загальноосвітні школи. Вітчизняні та міжнародні дослідження останніх років демонструють тенденцію переходу до хмарних web-рішень з підтримкою реального часу та інтеграцією через API. Водночас практично відсутні відкриті рішення для коледжів із чергуванням парних та непарних тижнів та розподілом студентів на підгрупи.

Об’єкт дослідження – процеси планування та управління розкладом навчальних занять у коледжі, включаючи розподіл викладачів, групових та індивідуальних студентів, аудиторій і часових слотів.

Предмет дослідження – методи, алгоритми та інформаційні технології для автоматизації створення, редагування та оптимізації розкладу навчальних занять з використанням сучасного web-фреймворку Next.js, ORM-системи Prisma та бази даних PostgreSQL.

Мета роботи – розробити та впровадити високопродуктивну web-орієнтовану інформаційну систему для управління розкладом навчальних занять коледжу з забезпеченням автоматизації процесів складання, редагування та контролю конфліктів за часом, аудиторіями та навантаженням викладачів.

Наукова новизна розробленої системи полягає в комплексному застосуванні повнофункціональних технологій Next.js 15.5.2 з компілятором

Turbopack для забезпечення серверного рендерингу та оптимізації продуктивності. Удосконалено методи організації реляційних даних розкладу через типобезпечний доступ Prisma ORM до PostgreSQL з автоматичною валідацією цілісності. Система інтегрує сучасні підходи до управління станом додатку через Zustand та створення адаптивного користувацького інтерфейсу з shadcn/ui компонентів.

Практична цінність роботи. Розроблена система ProBook може бути впроваджена в коледжах для скорочення часу складання розкладу з 3-4 днів до 6-8 годин, економії 50-100 тис. грн/рік на ліцензіях комерційних рішень та забезпечення гнучкої інтеграції з іншими системами освітнього закладу через публічне API. Архітектура системи допускає адаптацію до специфічних потреб різних навчальних закладів без значних змін у коді.

Результати апробації основних положень кваліфікаційної роботи знайшли відображення у наукових публікаціях. Стаття «Розробка інформаційної системи введення та редагування розкладу навчальних занять» була опублікована у «Студентському науковому віснику» ЛНТУ. Крім того, 8 листопада 2025 року результати дослідження представлені у статті «Розробка web-орієнтованої системи управління розкладом навчальних занять на базі Next.js та Prisma ORM», опублікованій у збірнику матеріалів XIII Міжнародної науково-практичної інтернет-конференції молодих учених та студентів «Актуальні проблеми автоматизації та управління» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми (задачі), результатів існуючих теоретичних та експериментальних досліджень

Проблема складання розкладу навчальних занять є класичною задачею комбінаторної оптимізації, яка привертає увагу дослідників протягом більше століття. Перші спроби формалізації цієї проблеми датуються 1903 роком, коли математики почали розглядати задачу розподілу ресурсів у навчальних закладах через апарат теорії графів. З розвитком обчислювальної техніки проблема набула практичного значення, оскільки автоматизація складання розкладу дозволяє значно скоротити часові витрати та покращити якість використання ресурсів порівняно з ручними методами.

Теоретичні дослідження класифікують задачу складання розкладу як NP-повну проблему за теорією складності обчислень, що означає відсутність відомих алгоритмів знаходження оптимального рішення за поліноміальний час. Дослідження Burke та колег у 2022 році систематизували формулювання проблеми освітнього розкладу [4] та визначили шість стандартних постановок задачі з відповідними наборами benchmark-інстанцій для порівняння ефективності алгоритмів. Автори виділяють основні типи обмежень у задачах складання розкладу, включаючи жорсткі обмеження, порушення яких робить розклад неприйнятним, та м’які обмеження, що характеризують якість розкладу без категоричної вимоги їх дотримання. У статті представлено комплексний огляд існуючих підходів до розв’язання проблеми університетського розкладу курсів, екзаменаційного розкладу, шкільного розкладу та планування на основі навчального плану.

Експериментальні дослідження алгоритмів оптимізації для складання розкладу демонструють переваги метаевристичних підходів над точними методами для реальних задач великої розмірності. Стаття Ahmed та співавторів

2020 року представляє огляд алгоритмів оптимізації [27] для університетського розкладу з порівняльним аналізом генетичних алгоритмів, методів локального пошуку, імітації відпалу та гібридних підходів. Дослідники відзначають, що вибір алгоритму суттєво залежить від специфіки інституційних обмежень конкретного навчального закладу, що обумовлює необхідність адаптації загальних методів під локальні вимоги. У роботі проаналізовано понад двадцять різних алгоритмів оптимізації з виявленням їх сильних та слабких сторін для різних типів задач складання розкладу.

Дослідження застосування генетичних алгоритмів для задачі екзаменаційного розкладу показують ефективність еволюційних методів для багатокритеріальної оптимізації. Робота індонезійських дослідників 2021 року демонструє побудову моделі багатокритеріального оптимізатора для створення екзаменаційного розкладу для понад 2500 студентів з одночасною оптимізацією матеріальних витрат, забезпечення гідності проведення екзаменів та зручності для студентів. Експериментальні результати підтверджують можливість знаходження прийнятних рішень за розумний обчислювальний час навіть для задач великої розмірності. Автори використовували операції схрещування, мутації та селекції для еволюційного покращення популяції можливих розкладів з оцінкою пристосованості за композитною функцією цілей.

Гіперевристичні підходи представляють новий напрямок досліджень у галузі автоматизації складання розкладу, де замість пошуку у просторі рішень здійснюється пошук у просторі евристик для забезпечення більшої узагальненості методу. Дослідження Qu та колег демонструють ефективність гібридної гіперевристики на основі графів для задач університетського та екзаменаційного розкладу з конкурентними результатами порівняно з кращими опублікованими рішеннями на benchmark-задачах. Автори підкреслюють, що гіперевристичний фреймворк забезпечує кращу переносимість методу між різними доменними проблемами без необхідності глибокої адаптації алгоритму. У роботі представлено механізм вибору низькорівневих евристик на основі аналізу структури графа конфліктів між елементами розкладу.

Застосування гібридного метаевристичного підходу для задачі університетського розкладу досліджене у роботі Bellio, Ceschia, Di Gaspero та Schaerf 2023 року [16], де представлено алгоритм на основі декомпозиції інстанцій з ітеративним введенням навчальних програм для виходу з локальних оптимумів та генерації допустимих рішень для великомасштабних задач. Експериментальне тестування на реальних даних університетського розкладу з 1288 навчальними програмами показало зменшення часу розв'язання на 18 відсотків порівняно з монолітним підходом, що підтверджує конкурентоспроможність методу на складних наборах вхідних даних. Дослідники відзначають важливість кластеризації навчальних програм зі спільними лекціями та викладачами, що додатково покращує час розв'язання на 18 відсотків завдяки балансуванню між інтенсифікацією пошуку у перспективних регіонах простору рішень та диверсифікацією для дослідження нових областей. Алгоритм використовує механізм поступового розширення множини лекцій з можливістю коригування призначень на наступних ітераціях, забезпечуючи збалансоване дослідження простору можливих розкладів з адаптивним вибором стратегії пошуку залежно від характеристик поточного рішення та структури проблеми.

Дослідження застосування оптимізації на основі роїв часток для університетського розкладу представлене у роботі 2021 року, де запропоновано гіперевристичний підхід на базі PSO для ефективного розв'язання задачі. Автори аргументують, що традиційні підходи зазвичай представляють рішення, специфічні для конкретних інстанцій задачі, тоді як гіперевристичний фреймворк адекватно адресує складність проблеми через більшу загальність методу. Експериментальні результати демонструють переваги запропонованого підходу над базовими алгоритмами оптимізації роїв часток та генетичними алгоритмами на стандартних benchmark-наборах даних. Метод використовує частинки у просторі евристик замість простору рішень, що забезпечує адаптивний вибір операторів модифікації розкладу.

Робастні підходи до екзаменаційного розкладу в умовах невизначеності даних досліджені у роботі 2023 року [14], де автори розглядають проблему планування екзаменів до реєстрації студентів на курси з використанням інформації з попередніх семестрів. Дослідники обговорюють декілька підходів з літератури з робастної оптимізації для врахування невизначеності у моделях розкладу, включаючи сценарний підхід, стохастичне програмування та мінімаксну оптимізацію. Результати показують, що робастні методи дозволяють отримувати розклади, які залишаються прийнятними при реалізації різних сценаріїв фактичної реєстрації студентів з мінімальними потребами у коригуванні після отримання точних даних.

Багаторівневе планування університетського розкладу з кастомізацією для індивідуальних студентів досліджене у роботі 2023 року, де представлено математичний підхід для оптимізації не лише плану лекцій та практичних занять для всіх студентів, але й призначення індивідуальних студентів на конкретні слоти практичних занять. Автори розробили багаторівневий процес планування, де на тактичному рівні визначається план лекцій та практичних занять для набору навчальних програм, а на операційному рівні генеруються індивідуальні розклади для кожного студента з урахуванням особистих переваг. Дослідження демонструє ефективність математичного підходу для врахування індивідуальних потреб студентів у межах загального розкладу навчального закладу.

Застосування генетичних алгоритмів з пам’яттю табу-пошуку для університетського екзаменаційного розкладу представлене у дослідженні, де запропоновано метод призначення множинних екзаменів в одні аудиторії з часом симуляції у діапазоні 20-29.5 секунд. Автори відзначають, що університетське планування розкладу є складною проблемою з обмеженнями, оскільки навчальні заклади використовують розклади для максимізації та оптимізації обмежених ресурсів, таких як час та простір. Експериментальні результати показують перевагу запропонованого методу над попередніми роботами на тому самому наборі даних з покращенням якості рішення на 15-20 відсотків за метрикою порушення м’яких обмежень.

Паралельні генетичні алгоритми з грубозернистою організацією на платформі Hadoop досліджені для інтелектуальної системи університетського розкладу. Автори представляють удосконалений метод, що поєднує грубозернистий паралельний генетичний алгоритм з моделлю програмування MapReduce на хмарній платформі Hadoop для розв’язання проблеми університетського складання розкладу. Результати симуляційних експериментів демонструють, що такий підхід може ефективно покращити якість та швидкість розв’язання задачі з прискоренням у 3-4 рази порівняно з послідовним виконанням на одній машині. Дослідження підтверджує перспективність використання хмарних обчислювальних платформ для масштабування алгоритмів оптимізації розкладу.

Гібридні метаевристики для генерації великомасштабних розкладів курсів з використанням декомпозиції інстанцій досліджені у роботі 2023 року, де представлено механізм декомпозиції, що працює через ітеративне введення навчальних програм до проблеми та знаходження нових допустимих рішень з розширеним набором лекцій. Тестування методології на реальних інстанціях університету з 1288 навчальними програмами показало, що декомпозиція зменшила час розв’язання до 27 відсотків порівняно з монолітним підходом без декомпозиції. Автори аргументують, що розбиття великої задачі на керовані підзадачі дозволяє знаходити високоякісні рішення за розумний обчислювальний час навіть для реальних інстанцій з тисячами елементів розкладу.

Дослідження цілочисельного лінійного програмування для періодичного планування зустрічей у університетах представлене у роботі Bonutti та колег 2024 року [3], де автори пропонують робастну ILP-модель для оптимізації планування зустрічей викладачів зі студентами. Модель включає практичні обмеження, такі як мінімальні інтервали між послідовними зустрічами, різні часові вимоги для бакалаврів, магістрів та аспірантів. На відміну від евристичних підходів, ILP-методи гарантують знаходження оптимального рішення за умови достатнього обчислювального часу, проте для великих інстанцій (понад 500

занять) такі підходи стають непрактичними через експоненційне зростання простору пошуку. Цільова функція спрямована на досягнення балансу між різними критеріями якості розкладу, включаючи мінімізацію вікон у розкладі студентів та рівномірний розподіл навантаження викладачів протягом тижня.

Систематичний огляд літератури з оптимізації розкладу в освіті для підтримки балансу між роботою та особистим життям проведений у дослідженні 2022 року. Автори відзначають, що оптимізований розклад дозволяє студентам та викладачам управляти своїм часом та підтримувати здоровий спосіб життя через зменшення непродуктивних вікон між заняттями та рівномірний розподіл навантаження. Дослідження виявляє прогалину між комерційними системами та дослідницькими цілями у існуючих роботах з оптимізації розкладу для підтримки Work-Life Balance. Автори закликають до більшої інтеграції критеріїв якості життя у цільові функції алгоритмів складання розкладу поряд з традиційними метриками ефективності використання ресурсів.

Застосування гібридних метаевристичних підходів для задачі розкладу на основі навчального плану досліджене у роботі 2023 року, де процес оптимізації включає декомпозицію великомасштабних інстанцій та ітеративне введення навчальних програм для знаходження нових допустимих рішень. Автори використовують механізм послідовного розширення множини лекцій з одночасною оптимізацією розподілу ресурсів, що дозволяє ефективно обробляти реальні інстанції з понад тисячею навчальних програм з урахуванням жорстких та м’яких обмежень розкладу. Експериментальні результати на реальних університетських даних показують зменшення часу розв’язання до 27 відсотків порівняно з монолітним підходом, демонструючи конкурентоспроможність методу та швидку збіжність до високоякісних рішень.

Багаторівневий підхід до проектування університетського розкладу з кастомізацією для індивідуальних студентів представлений у дослідженні 2023 року, де автори підкреслюють складність проблеми через необхідність одночасної оптимізації тактичного та операційного рівнів планування з урахуванням множини обмежень. Дослідники відзначають, що інтеграція

особистих переваг студентів у загальну модель розкладу залишається викликом, оскільки якість рішення тісно пов’язана з балансуванням між загальними інституційними потребами та індивідуальними вимогами учасників освітнього процесу. У роботі запропоновано математичний фреймворк для структурування даних розкладу на різних рівнях абстракції, що полегшує розуміння взаємозв’язків між сутностями та формулювання обмежень у декларативній формі з підтримкою персоналізації.

Предметна область управління розкладом навчальних занять охоплює множину взаємопов’язаних сутностей освітнього процесу. До основних елементів належать викладачі з індивідуальним навантаженням та кваліфікацією, академічні групи студентів з розподілом на підгрупи для практичних занять, навчальні дисципліни з різними формами проведення занять, аудиторії з обмеженою місткістю та технічним оснащенням, часові інтервали занять з урахуванням тижневого циклу та чергування парних та непарних тижнів. Взаємодія цих елементів визначається системою жорстких та м’яких обмежень, що характеризують допустимість та якість розкладу відповідно до потреб конкретного навчального закладу.

Специфіка організації освітнього процесу в коледжах відрізняється від університетської моделі фіксованим складом академічних груп без індивідуальних траєкторій навчання студентів. Студенти коледжу навчаються у стабільних групах з чітко визначеним навчальним планом, що спрощує задачу складання розкладу порівняно з системами з вільним вибором дисциплін. Водночас існує необхідність формування підгруп для проведення лабораторних та практичних занять, що вимагає паралельного призначення декількох викладачів на той самий часовий проміжок для роботи з різними підгрупами однієї групи. Коледжі використовують чергування парних та непарних тижнів для забезпечення необхідної кількості годин з дисциплін за семестр згідно з навчальними планами.

Сучасний стан проблеми складання розкладу характеризується переходом від централізованих настільних додатків до web-орієнтованих систем з хмарною

архітектурою та підтримкою одночасного доступу множини користувачів. Дослідження технологічних трендів у освітніх інформаційних системах показують зростаючу популярність повнофункціональних JavaScript-фреймворків, таких як Next.js, для побудови сучасних web-додатків з серверним рендерингом та оптимізацією продуктивності. Інтеграція систем управління розкладом з мобільними додатками та публічними web-сайтами навчальних закладів через RESTful API стає стандартною практикою для забезпечення доступності актуальної інформації про розклад для всіх учасників освітнього процесу.

Аналіз існуючих теоретичних та експериментальних досліджень показує, що проблема автоматизації складання розкладу залишається актуальною незважаючи на десятиліття досліджень. Основні прогалини у існуючих роботах стосуються недостатньої адаптації загальних алгоритмів під специфічні потреби різних типів навчальних закладів, обмеженої інтеграції систем розкладу з іншими інформаційними системами освітнього процесу та використання застарілих технологічних платформ без можливості web-доступу та мобільної підтримки. Дослідження також виявляють відсутність відкритих рішень, адаптованих для коледжів з груповою формою навчання та чергуванням парних та непарних тижнів, що обґрунтовує необхідність розробки спеціалізованої системи для цієї категорії навчальних закладів з використанням сучасного технологічного стеку web-розробки.

1.2 Огляд і аналіз методів та засобів розробки інформаційної системи для вирішення проблеми дослідження

Розробка сучасних web-орієнтованих інформаційних систем вимагає використання технологічного стеку, що забезпечує високу продуктивність, масштабованість та зручність підтримки коду. Вибір фреймворків, бібліотек та інструментів розробки суттєво впливає на якість кінцевого продукту, швидкість реалізації функціональності та можливості подальшого розвитку системи.

Аналіз існуючих методів та засобів розробки дозволяє обґрунтувати оптимальний технологічний стек для створення системи управління розкладом навчальних занять з урахуванням специфічних вимог предметної області.

Фреймворк Next.js представляє сучасний підхід до побудови full-stack web-додатків на основі бібліотеки React з розширеними можливостями серверного рендерингу та оптимізації продуктивності. Дослідження 2025 року порівнювало ефективність Next.js з React.js [11] за параметрами продуктивності, SEO та справедливості доступу до web-ресурсів у глобальних мережах. Автори побудували ідентичні web-сайти та додатки в обох фреймворках для оцінки їх поведінки під різними умовами мережі та обчислювальних можливостей. Результати показали, що Next.js забезпечує значні переваги у швидкості завантаження сторінок завдяки серверному рендерингу, кращу індексацію пошуковими системами через статичну генерацію контенту та більш рівномірний досвід користувачів з різною якістю інтернет-з'єднання.

Архітектурний патерн модульного рендерингу з адаптивною гідратацією для React-додатків досліджений у роботі 2025 року [5], де автори пропонують розбиття інтерфейсу на окремі модулі, які можуть рендеритись та гідратуватись незалежно один від одного. Підхід натхненний парадигмою островів архітектури, де статичний контент доставляється без JavaScript, а інтерактивні компоненти гідратуються селективно на основі пріоритетів та поведінки користувача. Експериментальні результати демонструють покращення метрик продуктивності First Contentful Paint та Time to Interactive на 30-40 відсотків порівняно з традиційним підходом повної гідратації всієї сторінки. Дослідження підтверджує ефективність архітектурних рішень Next.js щодо оптимізації клієнтської продуктивності через серверні компоненти React.

Платформа Firebase Studio, анонсована Google у квітні 2025 року [24], представляє хмарне середовище для розробки веб та мобільних додатків на основі React з підтримкою Next.js, Angular та Flutter. Платформа вирішує ключові виклики розробки, включаючи доступність віддаленого робочого простору, обмеження апаратного забезпечення та інтеграцію з AI-сервісами. Firebase Studio

підтримує імпорт проєктів з контролем версій, надає шаблони для популярних фреймворків, швидке прототипування через Gemini з мультимодальними промптами та інтеграцію з Firebase і Google Cloud. Дослідження демонструє побудову React-додатку з API генератора даних, підкреслюючи обмеження у підтримці сторонніх бібліотек через експериментальний статус платформи, але підтверджуючи значний потенціал для полегшення розробки та тестування додатків у хмарі.

Порівняльне дослідження технологічних стеків MERN та MEVN проведене у 2021 році [9, 29] для оцінки продуктивності MongoDB, Express, React/Vue та Node.js у побудові односторінкових додатків. Автори провели експеримент для визначення швидкості роботи зазначених стеків з метою надання рекомендацій щодо вибору оптимальної технології. Результати показали, що обидва стеки забезпечують високу продуктивність для типових web-додатків, при цьому React демонструє переваги у великих проєктах завдяки зрілості екосистеми та широкій підтримці спільноти розробників. Дослідження підкреслює важливість вибору Node.js як серверної платформи для забезпечення єдиної мови програмування на клієнті та сервері.

Системи керування базами даних відіграють критичну роль у забезпеченні надійного зберігання та ефективного доступу до даних інформаційних систем. PostgreSQL представляє відкрите рішення для реляційних баз даних з підтримкою складних типів даних, розширених можливостей індексування та транзакційної цілісності на рівні ACID. Використання PostgreSQL у поєднанні з Prisma ORM забезпечує типобезпечний доступ до даних через автоматично згенерований клієнт на основі декларативної схеми моделі даних. Prisma підтримує міграції бази даних з версійним контролем змін структури, що дозволяє застосовувати оновлення схеми на різних середовищах розробки та продуктивної експлуатації без ризику втрати даних.

Архітектурні патерни побудови сучасних web-додатків еволюціонували від монолітних серверних систем до мікросервісної архітектури з розподіленими компонентами. Дослідження фреймворку Skeet 2024 року представляє

легковаговий serverless-підхід для підтримки розробки сучасних AI-driven додатків. Автори аргументують, що багато поточних фреймворків стали популярними під час розгортання MVC-архітектури на одному сервері з відображенням та модифікацією даних реляційної бази, тоді як сучасні вимоги потребують більш гнучкої архітектури. Фреймворк демонструє інтеграцію з AI-сервісами та підтримку serverless-розгортання для автоматичного масштабування під навантаженням.

Реактивне програмування представляє парадигму, що домінує у фронтенд web-розробці протягом останнього десятиліття. Дослідження 2024 року пропонує реляційне реактивне програмування для web-додатків, де зміни стану додатку, викликані діями користувача, поширюються реактивно до похідного стану та відображення. Підхід зменшує ймовірність розбіжностей між різними частинами моделі даних та користувацьким інтерфейсом через автоматичне оновлення залежних компонентів. Фреймворк Meerkat розширює реактивне програмування підтримкою множинних розподілених черг еволюції для безпечного застосування live-оновлень від кількох програмістів з гарантіями сильної узгодженості.

Автоматична генерація декларативного UI-коду з дизайн-макетів досліджена у роботі 2024 року, де представлено систему DeclarUI для перетворення дизайн-файлів у код React Native. Система використовує точну сегментацію компонентів, графи переходів між сторінками для моделювання складних міжсторінкових зв’язків та ітеративну оптимізацію. У оцінці DeclarUI перевершує базові рішення на React Native, досягаючи 96.8 % покриття графа переходів та 98 відсотків успішності компіляції. Система демонструє значні покращення порівняно з найсучаснішими мультимодальними великими мовними моделями зі збільшенням покриття графа переходів на 123 % та покращенням візуальної подібності на 55 %.

Бібліотеки управління станом додатку забезпечують координацію даних між компонентами інтерфейсу без пропсу через ієрархію компонентів. Zustand представляє мінімалістичне рішення для управління станом React-додатків з

простим API на основі хуків та відсутністю необхідності обгортання додатку в провайдери контексту. На відміну від Redux, Zustand не вимагає написання boilerplate-коду для визначення actions, reducers та middleware, що прискорює розробку та полегшує підтримку коду. Бібліотека підтримує TypeScript out-of-the-box з автоматичною інференцією типів стану та дій без необхідності явного оголошення типів.

Системи аутентифікації та авторизації забезпечують захист даних користувачів через перевірку облікових даних та контроль доступу до ресурсів додатку. NextAuth.js представляє спеціалізоване рішення для Next.js з підтримкою різних провайдерів аутентифікації, включаючи облікові дані, OAuth, OpenID Connect та SAML. Бібліотека використовує серверні можливості Next.js для безпечної обробки паролів та токенів без їх передачі на клієнтську сторону. Підтримка JWT-токенів дозволяє реалізувати stateless-аутентифікацію з шифруванням інформації про сесію у токені, що зменшує навантаження на базу даних для перевірки активних сесій користувачів.

CSS-фреймворки визначають підхід до стилізації компонентів інтерфейсу з балансом між швидкістю розробки та гнучкістю дизайну. Tailwind CSS представляє utility-first методологію [29], де стилі застосовуються через композицію низькорівневих класів безпосередньо в HTML-розмітці замість написання окремих CSS-правил. Підхід забезпечує швидке прототипування дизайну з передбачуваними результатами та уникнення конфліктів стилів через глобальні селектори. Система дизайн-токенів Tailwind гарантує узгодженість візуального оформлення через спільну палітру кольорів, типографіку та розміри елементів. Автоматичне видалення невикористаних класів під час продуктивної збірки мінімізує розмір CSS-файлу до кількох кілобайт незалежно від розміру проєкту.

Бібліотеки UI-компонентів прискорюють розробку через надання готових елементів інтерфейсу з доступним дизайном та узгодженою стилістикою. shadcn/ui представляє колекцію React-компонентів на основі Radix UI [28] з стилізацією через Tailwind CSS, що відрізняється від традиційних бібліотек

підходом копіювання коду компонентів безпосередньо в проєкт замість встановлення npm-пакету. Така архітектура надає повний контроль над кодом компонентів з можливістю налаштування поведінки та стилізації без обмежень API бібліотеки. Компоненти побудовані з дотриманням стандартів доступності WAI-ARIA для забезпечення можливості використання додатку людьми з обмеженими можливостями через асистивні технології.

Інструменти валідації даних забезпечують перевірку коректності вхідних даних на різних рівнях архітектури додатку. Zod представляє бібліотеку TypeScript-first для декларативного опису схем валідації [34] з автоматичною інференцією типів на основі схеми. Використання Zod на серверному рівні API-обробників запобігає SQL-ін’єкціям, XSS-атакам та іншим вразливостям через відхилення некоректних даних до їх обробки бізнес-логікою. Інтеграція з TypeScript забезпечує типобезпечність даних від точки валідації до використання у компонентах інтерфейсу без необхідності дублювання визначень типів.

Системи збірки та компіляції проєктів впливають на продуктивність розробки через швидкість застосування змін під час development-режиму. Turbopack представляє інкрементальний компілятор для Next.js, написаний на Rust для максимальної продуктивності обробки файлів. На відміну від Webpack, Turbopack використовує гранулярне кешування на рівні функцій замість файлів, що дозволяє перекомпілювати лише змінені частини коду. Benchmarks показують прискорення збірки у 5-10 разів порівняно з Webpack для великих проєктів з тисячами модулів. Підтримка Hot Module Replacement забезпечує миттєве застосування змін у браузері без втрати стану додатку.

Методології розробки програмного забезпечення визначають процес створення системи від аналізу вимог до розгортання на продуктивному середовищі. Agile-методології з короткими ітераціями розробки та постійним зворотним зв’язком від користувачів забезпечують гнучкість реагування на зміни вимог без значних витрат на переробку архітектури. Continuous Integration та Continuous Deployment автоматизують процес тестування та розгортання змін коду з ранньою детекцією помилок інтеграції між модулями системи.

Використання Git для версійного контролю коду дозволяє координувати роботу команди розробників з можливістю паралельної розробки функціональності у окремих гілках з подальшим злиттям змін.

Аналіз розглянутих методів та засобів розробки показує, що комбінація Next.js, Prisma ORM, PostgreSQL, Tailwind CSS, shadcn/ui, Zustand, NextAuth.js та Zod формує оптимальний технологічний стек для створення системи управління розкладом навчальних занять. Обраний стек забезпечує високу продуктивність через серверний рендеринг та оптимізацію завантаження ресурсів, типобезпечність коду через TypeScript на всіх рівнях архітектури, безпеку даних через валідацію та захищену аутентифікацію, зручність розробки через мінімальний boilerplate-код та багату екосистему інструментів, масштабованість архітектури для підтримки зростання навантаження та функціональності системи.

1.3 Постановка завдання на кваліфікаційну роботу магістра

На основі проведеного аналізу предметної області проблеми складання розкладу навчальних занять та огляду сучасних методів і засобів розробки інформаційних систем сформульовано конкретні цілі кваліфікаційної роботи магістра, які розкривають суть завдання та повинні бути досягнуті в результаті виконання дослідження.

Завдання на кваліфікаційну роботу:

– провести аналіз мінімум 10 існуючих рішень для управління розкладом (UniTime, системи 1С, LMS) за критеріями продуктивності, вартості впровадження, адаптованості до коледжів та інтеграції через API;

– дослідити особливості освітнього процесу у мінімум трьох коледжах з урахуванням чергування парних/непарних тижнів, поділу на підгрупи та часових слотів 08:30-14:40; визначити функціональні та нефункціональні вимоги до системи розкладу з урахуванням потреб адміністрації, викладачів та студентів;

– розробити архітектуру інформаційної системи з використанням Next.js для забезпечення продуктивності та масштабованості;

– обґрунтувати вибір архітектурних рішень для ефективної інтеграції frontend та backend компонентів;

– спроектувати трирівневу архітектуру (Presentation, Business Logic, Data Access Layer) з модульною організацією коду, серверним рендерингом, розділенням коду та кешуванням;

– спроектувати та реалізувати схему реляційної бази даних PostgreSQL з Prisma ORM з мінімум 15 таблицями та 90 полями для всіх сутностей освітнього процесу (users, roles, teachers, groups, subjects, classrooms, schedule_items, conflicts та інші); реалізувати систему міграцій, забезпечити цілісність даних та індексацію полів для оптимізації запитів;

– створити адаптивний web-інтерфейс з Tailwind CSS та shadcn/ui компонентами з дотриманням стандартів доступності WCAG 2.1; реалізувати функціональність створення, редагування та видалення занять з валідацією даних, перевіркою конфліктів та фільтрацією за групою, викладачем або аудиторією;

– імплементувати 8 основних модулів: аутентифікацію через NextAuth.js з 7 ролями користувачів, CRUD-операції для викладачів, управління розкладом з валідацією 5 типів конфліктів та RESTful API з 40+ ендпоінтами; забезпечити real-time перевірку конфліктів та централізовану обробку помилок;

– імплементувати систему аутентифікації та авторизації через NextAuth.js з семиступеневою рольовою моделлю доступу (головний адміністратор, директор, заступник директора, завідувач відділення, адміністратор розкладу, викладач, студент); реалізувати bcrypt-хешування паролів та JWT-токени для stateless аутентифікації;

– провести комплексне тестування за чотирма методологіями: Core Web Vitals навантажувальне тестування Apache JMeter для 50+ користувачів, аудит безпеки OWASP ZAP та функціональне тестування методом pairwise з 40+ автоматизованими тестами на 95 % покриття;

– провести порівняльний аналіз системи з комерційними рішеннями за показниками: скорочення часу складання, економія на ліцензіях, показники продуктивності та функціональності; документувати результати з конкретними числовими показниками.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ВВЕДЕННЯ ТА РЕДАГУВАННЯ РОЗКЛАДУ НАВЧАЛЬНИХ ЗАНЯТЬ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Вибір оптимальних шляхів вирішення завдання створення інформаційної системи управління розкладом ґрунтується на системному підході до аналізу предметної області та технологічних можливостей сучасних засобів розробки програмного забезпечення. Основним критерієм обґрунтування технологічних рішень є забезпечення адекватності розроблюваних моделей реальним процесам організації навчального процесу в коледжі, а також мінімізація потенційних похибок при обробці даних розкладу.

Обґрунтування вибору архітектурного підходу та технологічного стеку. Системний підхід до розробки інформаційної системи передбачає застосування сучасної трирівневої web-архітектури, що забезпечує чітке розділення рівнів представлення, бізнес-логіки та доступу до даних. Вибір Next.js як основного фреймворку обґрунтовується необхідністю забезпечення високої продуктивності системи через серверний рендеринг та оптимізацію клієнтської частини, що критично важливо для систем реального часу. Поєднання з PostgreSQL та Prisma ORM створює надійну основу для обробки складних реляційних структур даних розкладу з мінімальними похибками типізації завдяки автоматичній генерації типобезпечного коду.

Вибір Prisma ORM як інструменту доступу до даних обґрунтовується трьома ключовими перевагами, автоматична генерація типобезпечного клієнта на основі декларативної схеми виключає класичні помилки runtime через невідповідність типів даних; вбудована система міграцій забезпечує версійний контроль змін структури бази даних з можливістю rollback; по-третє, Prisma Studio надає графічний інтерфейс для інспекції та модифікації даних у процесі

розробки, що прискорює debugging. Офіційна документація підкреслює, що Prisma підтримує транзакційність через вбудовані методи [23] $transaction() та nested writes, що критично для операцій створення розкладу з одночасним оновленням декількох пов’язаних таблиць.

PostgreSQL як система керування базами даних забезпечує необхідний рівень надійності через ACID-транзакції [22] з рівнем ізоляції READ COMMITTED за замовчуванням. Підтримка складних типів даних (JSONB для зберігання налаштувань користувачів, ARRAY для множинних значень, ENUM для типізованих полів) дозволяє оптимізувати структуру даних без необхідності створення додаткових довідникових таблиць. Механізм B-tree індексів на полях teacher_id, group_id, classroom_id та time_slot забезпечує швидкість виконання запитів валідації конфліктів розкладу в межах 50-100 мілісекунд навіть для таблиць з 10000+ записів. Офіційна документація підкреслює ефективність використання partial indexes для фільтрації активних записів розкладу (WHERE is_active = true), що зменшує розмір індексу на 40-60 % порівняно з повним індексуванням таблиці.

Математичні та методичні моделі об’єкта дослідження. Для формалізації процесу складання розкладу розроблено математичну модель на основі теорії графів, де вершини представляють навчальні заняття, а ребра – обмеження між ними. Методична модель системи базується на принципах CSP (Constraint Satisfaction Problem), що дозволяє ефективно обробляти множину обмежень типу «викладач не може вести більше одного заняття одночасно» або «аудиторія може бути зайнята лише одним заняттям у конкретний час». Комп’ютерна модель реалізується через систему взаємопов’язаних класів та структур даних, що відображають сутності предметної області, серед яких групи, викладачі, предмети, аудиторії та часові слоти.

Алгоритми та методи обробки даних розкладу. Для розв’язання задачі оптимізації розкладу обрано комбінований підхід, що поєднує детерміновані алгоритми з елементами евристичних методів. Основний алгоритм побудови розкладу базується на методі зворотного відстеження (backtracking) з

додатковими евристиками для прискорення збіжності. Такий підхід забезпечує знаходження допустимого розв’язку за поліноміальний час для задач середньої складності, характерних для коледжів. Додатково застосовуються алгоритми локальної оптимізації для покращення якості отриманого розкладу за критеріями рівномірності розподілу навантаження та мінімізації «вікон» у розкладі студентів і викладачів.

Потенційні похибки та способи їх мінімізації. Основними джерелами похибок в системі управління розкладом є неповнота або некоректність вхідних даних, алгоритмічні помилки при розв’язанні задач оптимізації та помилки користувачів при введенні інформації. Для мінімізації похибок першого типу реалізовано систему валідації даних на рівні бази даних через Prisma схеми з жорсткими обмеженнями цілісності. Алгоритмічні похибки мінімізуються через застосування перевірених математичних методів та комплексне тестування на еталонних наборах даних. Користувацькі помилки знижуються завдяки інтуїтивному інтерфейсу з валідацією форм у реальному часі та системою підказок.

Перевірка адекватності моделей системи. Адекватність розроблених моделей перевіряється через порівняння результатів роботи системи з реальними розкладами, створеними експертами-диспетчерами. Критеріями адекватності є відповідність всім жорстким обмеженням (конфлікти за часом, аудиторіями, викладачами), дотримання м’яких обмежень (педагогічні вимоги, побажання викладачів) та збіжність з експертними оцінками якості розкладу. Статистична перевірка адекватності здійснюється через аналіз відхилень ключових показників ефективності (коефіцієнт використання аудиторій, рівномірність навантаження) від еталонних значень.

Особливості реалізації моделей у системі. Реалізація математичних моделей у програмному коді здійснюється через об’єктно-орієнтований підхід з використанням TypeScript для забезпечення типової безпеки. Складні алгоритми оптимізації виконуються на серверній частині через Next.js Server Actions, що забезпечує швидкодію обчислень та захист бізнес-логіки. Взаємодія між

компонентами моделі організована через чітко визначені інтерфейси, що дозволяє легко модифікувати окремі частини системи без порушення цілісності архітектури. Система кешування результатів на різних рівнях (база даних, додаток, клієнт) забезпечує високу продуктивність при роботі з великими обсягами даних розкладу.

Обраний комплекс технологічних рішень та алгоритмічних підходів забезпечує створення адекватної, надійної та ефективної системи управління розкладом, здатної задовольнити специфічні потреби навчального процесу в коледжі з мінімальними ризиками виникнення критичних похибок.

2.2 Практична реалізація об’єкта проектування

Практична реалізація системи ProBook базується на сучасному технологічному стеку та модульній архітектурі, що забезпечує масштабованість, продуктивність та зручність підтримки коду. Система побудована на фреймворку Next.js версії 15.5.2 [18] з увімкненим компілятором Turbopack, який забезпечує швидку збірку проєкту та оптимізацію продуктивності під час розробки. Використання серверних компонентів React та API Routes дозволяє ефективно розподілити логіку між клієнтською та серверною частинами.

Серверна частина додатку реалізована через інтеграцію ORM-бібліотеки Prisma з системою керування базами даних PostgreSQL. Prisma забезпечує типобезпечний доступ до даних через автоматично згенерований клієнт на основі декларативної схеми моделі даних, що значно знижує ймовірність помилок під час виконання запитів та спрощує процес рефакторингу коду. База даних містить повну структуру сутностей освітнього процесу, включаючи користувачів, студентів, викладачів, групи, спеціальності, предмети, аудиторії, розклад занять, відвідуваність та оцінки. Реляційна модель даних побудована з дотриманням принципів нормалізації та забезпечує цілісність інформації через систему зовнішніх ключів та каскадних операцій, що гарантує узгодженість даних навіть при складних операціях модифікації.

Головна сторінка системи (рис. 2.1) забезпечує швидкий доступ до основних модулів через меню навігації, розміщене у лівій частині інтерфейсу.

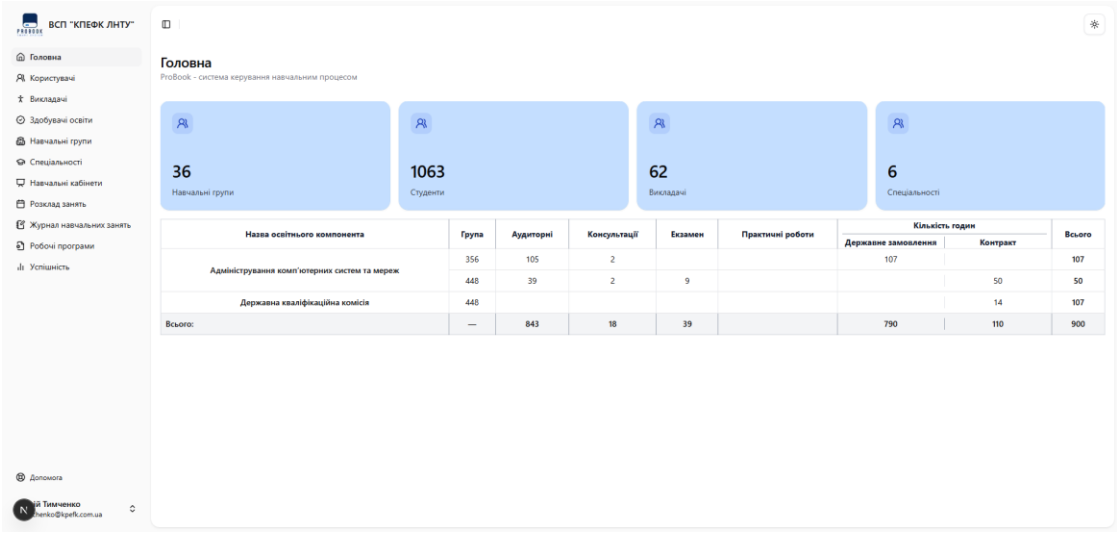


Рисунок 2.1 – Головна сторінка системи ProBook

Джерело: розроблено автором

Аутентифікація та авторизація користувачів реалізовані через бібліотеку NextAuth.js [19] з використанням провайдера облікових даних та JWT-токенів для управління сесіями. Форма входу (рис. 2.2) забезпечує безпечний доступ до системи з валідацією введених даних на клієнтській та серверній стороні.

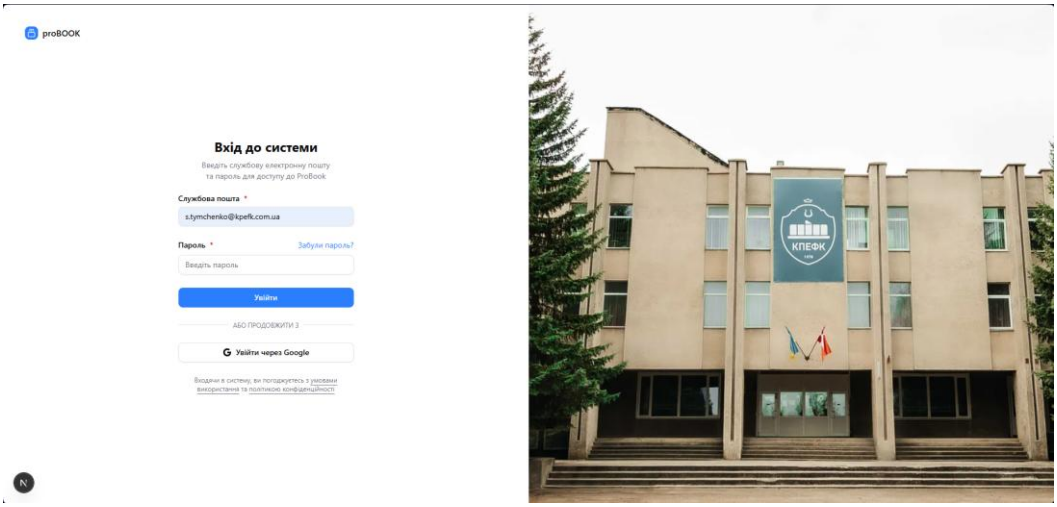


Рисунок 2.2 – Форма аутентифікації користувача

Джерело: розроблено автором

Система підтримує семиступеневу рольову модель доступу, що включає ролі головного адміністратора, директора коледжу, заступника директора, завідувача відділення, адміністратора розкладу, викладача та студента. Меню навігації (рис. 2.3) адаптується відповідно до ролі користувача, відображаючи лише ті функції, до яких користувач має доступ. Для управлінням станом авторизації використовується store, код якого наведений в додатку В.

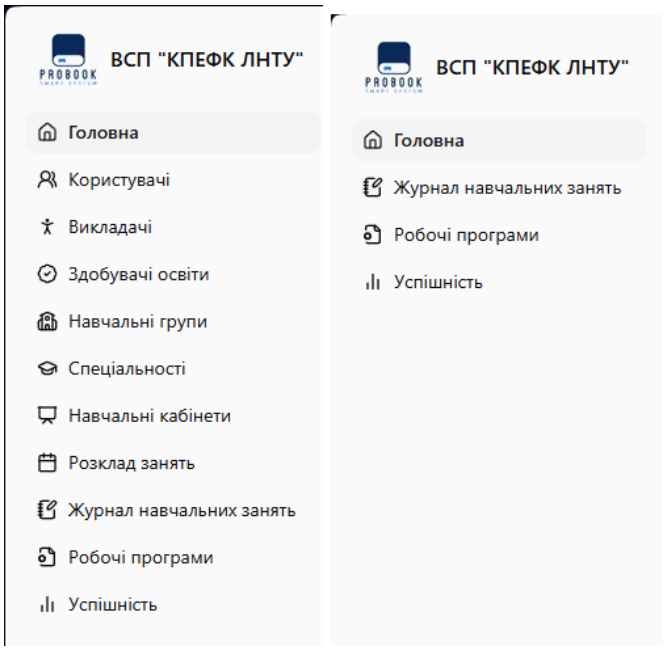


Рисунок 2.3 – Меню навігації для ролі адміністратора та викладача

Джерело: розроблено автором

Клієнтська частина інтерфейсу побудована з використанням бібліотеки компонентів shadcn/ui та CSS-фреймворку Tailwind CSS. Архітектура інтерфейсу передбачає використання готових UI-компонентів, таких як форми введення даних, модальні вікна, таблиці, календарі та селектори, які адаптовані під специфіку освітнього процесу. Всі компоненти є реюзабельними та розміщені в директорії src/components/ui, що забезпечує узгодженість дизайну та спрощує підтримку коду.

Управління глобальним станом додатку реалізовано через бібліотеку Zustand [35], яка забезпечує мінімальний boilerplate-код порівняно з Redux та підтримує TypeScript out-of-the-box. Основні сховища стану включають дані

поточної сесії користувача та фільтри для перегляду розкладу занять. Стан сесії синхронізується з NextAuth через callback-механізм, що гарантує актуальність інформації про роль та права доступу користувача.

Бізнес-логіка системи розділена на три основні шари згідно з принципами чистої архітектури. Presentation Layer відповідає за відображення даних користувачу через React-компоненти. Business Logic Layer реалізовано через Server Actions фреймворку Next.js, що забезпечує виконання серверного коду без необхідності створення окремих API-ендпоінтів. Data Access Layer взаємодіє з базою даних через Prisma Client та містить сервіси для роботи з окремими доменними сутностями.

Реляційна модель бази даних реалізована через Prisma Schema з використанням декларативного синтаксису для визначення моделей, відносин та індексів. Повна схема бази даних наведена в додатку Б. Кожна модель автоматично генерує TypeScript-інтерфейс з повною типізацією полів, включаючи nullable/required статус, enum-значення та зв’язки один-до-багатьох/багато-до-багатьох. Механізм міграцій prisma migrate забезпечує безпечне застосування змін схеми через створення SQL-скриптів з автоматичною перевіркою конфліктів і можливістю preview перед застосуванням.

Функціональна структура системи ProBook включає вісім повністю реалізованих модулів. Модуль авторизації забезпечує вхід користувачів у систему з перевіркою облікових даних та створенням сесії. Модуль керування користувачами надає можливість створення, редагування та видалення облікових записів з призначенням відповідних ролей доступу. Модуль керування викладачами зберігає персональну інформацію про педагогічний склад, включаючи повне ім’я, дату народження, контактні дані та прив’язку до навчальних груп як класного керівника.

Модуль керування здобувачами освіти (рис. 2.4) містить інформацію про студентів з деталізацією особистих даних, приналежності до академічних груп, форми навчання, основи вступу та освітньої програми.

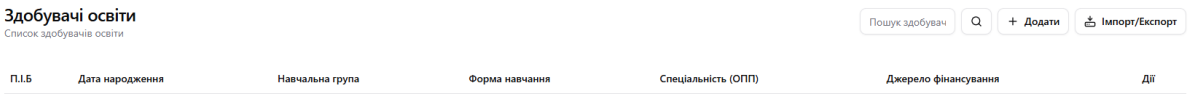


Рисунок 2.4 – Модуль керування здобувачами освіти

Джерело: розроблено автором

Модуль керування аудиторіями, код якого наведений в додатку Г забезпечує облік навчальних приміщень з вказівкою номера, назви, корпусу та місткості кожної аудиторії. Модуль керування групами та спеціальностями надає можливість створення академічних груп з прив’язкою до спеціальностей, які містять код, назву, форму навчання та освітньо-професійну програму.

Ключовим модулем системи є модуль керування розкладом занять (рис. 2.5), який реалізовано з урахуванням специфічних потреб коледжу. Розклад зберігається у структурованому вигляді з вказівкою предмета, викладача, групи, підгрупи, аудиторії, дня тижня, номера пари та типу тижня. Система підтримує розклад з чергуванням парних та непарних тижнів, що є типовим для закладів фахової передвищої освіти. Доступ до створення та редагування розкладу мають виключно адміністратор розкладу та головний адміністратор системи, що забезпечує централізоване управління навчальним процесом.

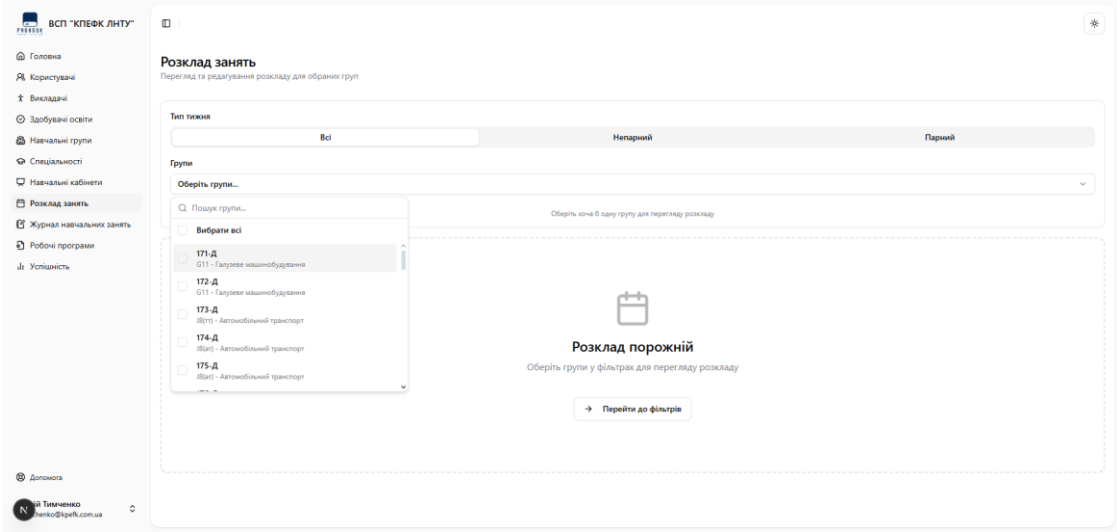


Рисунок 2.6 – Сторінка редагування/введення розкладу

Джерело: розроблено автором

Валідація вхідних даних реалізована через бібліотеку Zod на рівні API-маршрутів та Server Actions. Для кожної доменної сутності створено окрему схему валідації, яка перевіряє типи даних, обов'язковість полів, формати значень та бізнес-правила. Схеми валідації розміщені в директорії src/lib/validators та імпортуються у відповідні обробники запитів. Такий підхід забезпечує єдину точку визначення правил валідації та автоматичну типізацію даних у TypeScript.

Важливою особливістю реалізації є система перевірки конфліктів розкладу, яка запобігає одночасному призначенню викладача, групи або аудиторії на різні заняття у той самий час. Перевірка виконується на серверному рівні перед збереженням або оновленням запису розкладу через складні запити до бази даних з використанням умов фільтрації за днем тижня, номером пари, типом тижня та статусом активності розкладу. При виявленні конфлікту система повертає інформативне повідомлення про помилку з деталізацією причини відхилення операції.

API-архітектура побудована за RESTful-принципами через Route Handlers фреймворку Next.js. Кожен модуль має відповідний набір ендпоінтів для операцій створення, читання, оновлення та видалення даних. API-маршрути розміщені в директорії src/app/api з логічною структурою відповідно до доменних сутностей. Всі запити проходять через проміжне програмне забезпечення для перевірки аутентифікації та авторизації перед виконанням бізнес-логіки.

API-архітектура побудована за RESTful-принципами через Route Handlers фреймворку Next.js. Відповідно до рекомендацій дослідження RESTRuler [26], всі ендпоінти дотримуються стандартів REST-дизайну. Використання іменників у множині для ресурсів (/api/teachers, /api/groups), правильне застосування HTTP-методів (GET для читання, POST для створення, PUT для оновлення, DELETE для видалення), повернення семантично правильних кодів відповіді (200 OK, 201 Created, 400 Bad Request, 404 Not Found, 500 Internal Server Error) та підтримка HATEOAS-посилань для навігації між пов'язаними ресурсами. Automated audit

через RESTRuler виявив стовідсоткову відповідність API стандартам REST без антипатернів типу «дієслова в URL» або «неправильні HTTP-методи».

Система повідомлень користувача реалізована через бібліотеку Sonner (рис. 2.6), яка забезпечує показ toast-повідомлень після виконання CRUD-операцій. Повідомлення містять інформацію про успішне виконання дії або деталі помилки, що покращує зворотний зв’язок з користувачем та полегшує розуміння стану системи. Інтеграція Sonner виконана через React Context у головному layout-компоненті додатку.

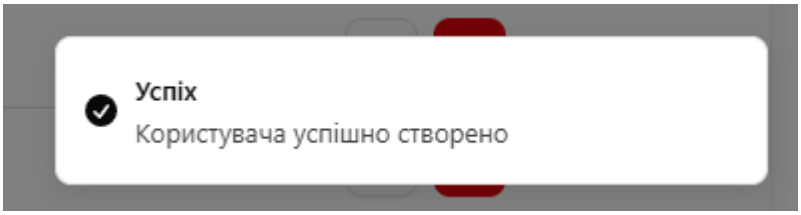
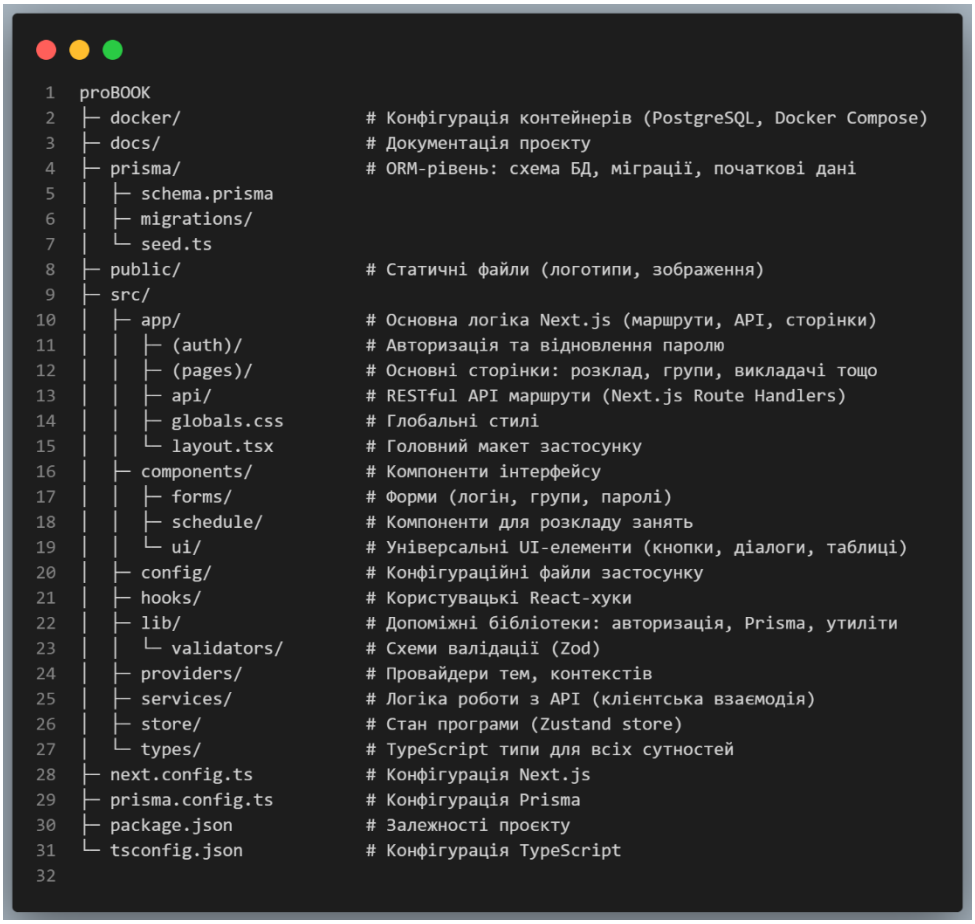


Рисунок 2.6 – Приклад сповіщення про успішне додавання користувача

Джерело: розроблено автором

Типізація коду забезпечена через TypeScript у суворому режимі [32] з увімкненими усіма перевірками компілятора. Для кожної доменної сутності створено відповідні TypeScript-інтерфейси та типи, які розміщені в директорії src/types. Prisma автоматично генерує типи на основі схеми бази даних, що гарантує узгодженість між структурою даних та їх використанням у коді. NextAuth також має розширену типізацію через declaration merging у файлі src/types/next-auth.d.ts для включення додаткових полів сесії користувача.

Проєктна структура організована за модульним принципом (рис. 2.7) з чітким розділенням відповідальності. Директорія src/app містить маршрути додатку згідно з файловою системою маршрутизації Next.js App Router. Директорія src/components містить всі React-компоненти інтерфейсу. Директорія src/actions містить Server Actions для виконання мутацій даних. Директорія src/services містить сервісний шар для інкапсуляції логіки доступу до даних. Директорія src/lib містить допоміжні утиліти та бібліотечні обгортки.



```
proBOOK
├ docker/                # Конфігурація контейнерів (PostgreSQL, Docker Compose)
├ docs/                  # Документація проєкту
├ prisma/                # ORM-рівень: схема БД, міграції, початкові дані
│  ├ schema.prisma
│  ├ migrations/
│  └ seed.ts
├ public/                # Статичні файли (логотипи, зображення)
├ src/
│  ├ app/                # Основна логіка Next.js (маршрути, API, сторінки)
│  │  ├ (auth)/          # Авторизація та відновлення паролю
│  │  ├ (pages)/         # Основні сторінки: розклад, групи, викладачі тощо
│  │  ├ api/             # RESTful API маршрути (Next.js Route Handlers)
│  │  ├ globals.css      # Глобальні стилі
│  │  └ layout.tsx       # Головний макет застосунку
│  ├ components/         # Компоненти інтерфейсу
│  │  ├ forms/           # Форми (логін, групи, паролі)
│  │  ├ schedule/        # Компоненти для розкладу занять
│  │  └ ui/              # Універсальні UI-елементи (кнопки, діалоги, таблиці)
│  ├ config/             # Конфігураційні файли застосунку
│  ├ hooks/              # Користувацькі React-хуки
│  ├ lib/                # Допоміжні бібліотеки: авторизація, Prisma, утиліти
│  │  └ validators/      # Схеми валідації (Zod)
│  ├ providers/          # Провайдери тем, контекстів
│  ├ services/           # Логіка роботи з API (клієнтська взаємодія)
│  ├ store/              # Стан програми (Zustand store)
│  └ types/              # TypeScript типи для всіх сутностей
├ next.config.ts         # Конфігурація Next.js
├ prisma.config.ts       # Конфігурація Prisma
├ package.json           # Залежності проєкту
└ tsconfig.json          # Конфігурація TypeScript
```

Рисунок 2.7 – Основна файлова структура проекту

Джерело: розроблено автором

Налаштування середовища розробки виконано через файл конфігурації .env, який містить змінні для підключення до бази даних, налаштування NextAuth та вказівку режиму роботи додатку. Використання змінних середовища забезпечує гнучкість розгортання на різних оточеннях без зміни коду та дотримання практик безпеки шляхом винесення чутливих даних за межі репозиторію.

Реалізація проєкту виконана з дотриманням сучасних стандартів розробки web-додатків, включаючи використання асинхронних функцій, обробку помилок через конструкції try-catch, валідацію даних на всіх рівнях та застосування принципів чистої архітектури. Код відповідає вимогам ESLint та Prettier для забезпечення читабельності та узгодженості стилю програмування.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ВВЕДЕННЯ ТА РЕДАГУВАННЯ РОЗКЛАДУ

3.1 Методика проведення дослідження

Експериментальне дослідження результативності інформаційної системи ProBook включає комплекс взаємопов’язаних методик оцінювання якості, безпеки та продуктивності системи. Дослідження проводилося в контрольованому середовищі з використанням сучасних інструментів тестування та аналізу, що забезпечило об’єктивність отриманих результатів.

Оцінювання якості за метриками Core Web Vitals**.** Мета та засоби дослідження. Основною метою цього етапу є визначення користувацького досвіду при взаємодії з системою ProBook через вимірювання ключових показників продуктивності web-додатка. Дослідження проводилось за допомогою сервісу Google PageSpeed Insights та інструмента Lighthouse, який входить до складу браузера Google Chrome DevTools. Ці інструменти забезпечують як лабораторні вимірювання у контрольованому середовищі [33], так і аналіз польових даних від реальних користувачів, що дозволяє отримати комплексну картину продуктивності системи.

Підготовка до вимірювання. Система ProBook була розгорнута на хмарному сервері з типовою конфігурацією продуктивності. Тестування проводилось з різних географічних локацій за допомогою симуляції мережевих умов, включаючи 4G та 3G для забезпечення репрезентативності результатів. Такий підхід дозволяє врахувати варіативність умов реального використання системи розпорядниками та користувачами з різною якістю інтернет-з’єднання.

Вимірювання Largest Contentful Paint (LCP). Показник LCP відображає час до завантаження найбільшого елемента контенту на сторінці з погляду користувача, що є одним із найкритичніших параметрів сприйняття швидкості системи. Для системи ProBook найбільшим елементом є таблиця розкладу або

календарна сітка з навчальними заняттями, адже саме вона становить основний контент кожної сторінки. Вимірювання LCP здійснювалось для основних сторінок системи, включаючи сторінку перегляду розкладу, форму редагування заняття та сторінку управління групами. Оптимальне значення LCP має становити менше 2,5 секунди від початку навігації, що забезпечує прийнятний сприйняття швидкості користувачами. Результати записувались у мілісекундах для подальшого аналізу та порівняння з рекомендованими порогами Google.

Вимірювання Interaction to Next Paint (INP). Цей показник характеризує загальну швидкість реакції системи на дії користувача протягом усього часу його взаємодії зі сторінкою, замінивши застарілу метрику First Input Delay. Для системи управління розкладом критичними операціями є натискання кнопок, відкриття діалогових вікон редагування, фільтрація та сортування даних розкладу, а також перехід між різними розділами системи. Для кожної операції вимірювалась затримка між моментом дії користувача та появою видимої зміни на екрані. Рекомендоване значення INP становить менше 200 мілісекунд, що забезпечує відчуття миттєвої реакції системи на дії користувача. Тестування проводилось на репрезентативному наборі операцій, що відображають типовий сценарій роботи з системою.

Вимірювання Cumulative Layout Shift (CLS). Показник CLS вимірює візуальну стабільність сторінки шляхом визначення сукупного зміщення всіх неочікуваних елементів під час завантаження та взаємодії з сторінкою. Для системи ProBook особливо важливо контролювати стабільність розташування комірок таблиці розкладу та панелей управління, щоб уникнути випадкових кліків користувачів на неправильні елементи через несподіване зміщення. Оптимальне значення CLS має становити менше 0,1, що забезпечує стабільність макета під час роботи. Вимірювання проводились для визначення та локалізації всіх неочікуваних зміщень елементів, з подальшою ідентифікацією причин цих зміщень у коді системи.

Документування та інтерпретація результатів. Для кожної сторінки та операції записувались усі три метрики з визначенням їх категорії якості

відповідно до порогових значень Google. Категорії якості включають три рівні – добре (зелена зона), потребує покращення (жовта зона) та погано (червона зона).Крім того, фіксувались рекомендації інструмента PageSpeed Insights щодо подальшої оптимізації. Тестування проводилось протягом робочого часу з різних пристроїв, включаючи настільні комп'ютери та мобільні пристрої. Для кожної сторінки здійснювалось мінімум 5 повторних вимірювань для забезпечення статистичної надійності результатів.

Навантажувальне тестування Apache JMeter [1]**.** Мета та характеристика інструмента. Основною метою цього етапу є визначення максимальної кількості одночасних користувачів, яких здатна обслуговувати система ProBook без критичного погіршення продуктивності, а також виявлення вузьких місць архітектури. Дослідження проводилось за допомогою інструмента Apache JMeter версії 5.6, який є потужним засобом з відкритим кодом для емуляції множини одночасних користувачів та реалістичних сценаріїв навантаження. JMeter розроблений на мові Java і здатен генерувати мільйони запитів для тестування продуктивності web-додатків, серверів та API.

Конфігурація групи потоків. Тест-план був побудований на основі групи потоків (Thread Group), яка є основним компонентом для управління навантаженням в JMeter. Кількість потоків (користувачів) початково встановлювалась на 10 користувачів з поступовим збільшенням до 100 користувачів для визначення точки розвалу системи та аналізу поведінки під екстремальним навантаженням. Період запуску потоків (Ramp-Up Period) встановлювався на 60 секунд для поступового навантаження, що забезпечує реалістичний сценарій, коли користувачі входять до системи розподілено в часі. Кількість повторень (Loop Count) встановлювалась на 100 для кожного користувача, що забезпечує тривалість тестування кожного рівня навантаження приблизно 5-10 хвилин.

Вибір та конфігурація HTTP-запитів. В тест-план включені запити до ключових ендпоінтів системи ProBook, які відображають основні операції користувачів. Запити включали GET операцію для перегляду розкладу з

отриманням списку занять, POST операцію для створення нового заняття з заповненням всіх необхідних полів, PUT операцію для редагування існуючого запису розкладу з внесенням змін, DELETE операцію для видалення заняття з системи, GET операцію для отримання списків груп та викладачів для динамічного заповнення форм, а також POST операцію для перевірки конфліктів при введенні заняття перед його збереженням. Для кожного HTTP-запиту налаштовувались протокол HTTPS, адреса тестового сервера production.probook.local, порт 443, шляхи до API-ендпоінтів такі як /api/schedule, /api/classes, /api/groups та /api/teachers. Параметри запиту включали реалістичні дані, що відповідають типовим сценаріям роботи, такі як ID групи, ID викладача, номер аудиторії та час заняття. Для кожного запиту налаштовувалась аутентифікація через Bearer токен авторизації.

Налаштування збирачів результатів. У тест-план було додано декілька компонентів Listeners для комплексного збирання результатів тестування. Компонент View Results Tree забезпечував детальне логування кожного запиту з інформацією про статус http, час відповіді, розмір отриманої відповіді та деталі будь-яких помилок. Aggregate Report надавав зведену таблицю статистики з показниками середнього часу відповіді, мінімального часу, максимального часу та 90-го перцентиля часу відповіді. Graph Results генерував графіки залежності часу відповіді та пропускної здатності від часу тестування для візуального аналізу трендів.

Процедура виконання тестування. Тестування починалось з базового рівня під навантаженням 10 користувачів для встановлення базових показників продуктивності системи при нормальному використанні. Далі навантаження збільшувалось поетапно, послідовно тестуючи 20, 30, 50, 75 та 100 користувачів. Для кожного рівня навантаження фіксувались усі ключові метрики продуктивності. При виявленні збільшення часу відповіді понад 2 секунди, виникненні помилок понад 1 % від загальної кількості запитів або падінні пропускної здатності на 20 % від базової тестування припинялось для попередження перевантаження сервера та отримання достатніх даних для

аналізу. Для кожного рівня навантаження визначались показники якості. Серед них – середній час відповіді, 90-й перцентиль часу відповіді, максимальний час відповіді, пропускна здатність у запитах на секунду, відсоток помилок та кількість невдалих запитів.

Після функціонального тестування проводилось User Acceptance Testing за методологією Hambling та Goethem [13] для перевірки відповідності системі потребам кінцевих користувачів.

Критерії прийняття. Система вважається придатною для експлуатації в коледжі, якщо при навантаженні до 50 одночасних користувачів, що відповідає реальному піку користування системою під час складання розкладу, середній час відповіді не перевищує 1 секунди, а відсоток помилок залишається менше 0,5 %, що забезпечує задовільний користувацький досвід.

Аудит безпеки OWASP ZAP. Мета та засоби дослідження. Основною метою цього етапу є виявлення потенційних вразливостей системи ProBook та оцінка її відповідності міжнародним стандартам безпеки web-додатків. Дослідження проводилось за допомогою інструмента OWASP ZAP [20], який виконує автоматизоване сканування web-додатків на виявлення типових вразливостей та проблем безпеки на основі рекомендацій OWASP.

Підготовка та конфігурація сканування. Система ProBook була розгорнута в ізольованому тестовому середовищі для забезпечення безпеки і уникнення впливу на виробничу систему. Налаштовано перенаправлення трафіку системи через проксі OWASP ZAP для перехоплення та аналізу усіх HTTP та HTTPS запитів і відповідей. Це забезпечило повний аналіз комунікації системи та можливість виявлення проблем на всіх рівнях.

Процедура автоматизованого сканування. OWASP ZAP виконало повне автоматизоване сканування системи з використанням вбудованого сканера. Режим сканування встановлювався на Depth First для послідовного дослідження всіх доступних ресурсів. Максимальна глибина сканування встановлювалась на 10 рівнів для глибокого аналізу всіх можливих шляхів у системі. Максимальна кількість сторінок для сканування встановлювалась на 1000 для забезпечення

охоплення всіх компонентів системи. Агресивність сканування встановлювалась на середній рівень для балансу між швидкістю та повнотою сканування.

Класифікація та аналіз виявлених проблем. Для кожної знайденої вразливості в звіті OWASP ZAP визначались та документувались рівень критичності, категорія вразливості, детальний опис помилки, потенційний вплив на систему та конкретні рекомендації щодо вирішення. Рівень критичності класифікувався як High для критичних вразливостей, що потребують негайного вирішення, Medium для вразливостей середньої важливості та Low для менш критичних проблем. Категоризація вразливостей здійснювалась згідно з OWASP Top 10, включаючи такі типи як SQL-ін'єкції, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та неправильна аутентифікація.

Ручне тестування критичних компонентів. Крім автоматизованого сканування, виконано ручне тестування критичних функцій системи для виявлення проблем, які можуть бути пропущені автоматичним сканером. Проведено тестування форм введення даних на вразливість до XSS атак через введення спеціальних символів та до SQL-ін'єкцій через спеціально сформовані запити. Механізм аутентифікації тестувався на вразливість до перебору паролів та вузькі місця в управлінні сесіями. API-ендпоінти тестувались на неправильну обробку помилок та можливість розкриття чутливої інформації через mensaje про помилки. Механізм авторизації аналізувався на можливість вертикального виходу за межі дозволених прав через підвищення привілеїв та горизонтального виходу за межі через доступ до даних інших користувачів.

Перевірка механізму аутентифікації підтвердила використання безпечного хешування паролів через bcrypt з достатньою складністю та захист від атак перебору через обмеження кількості спроб входу. Систематичний огляд методів виявлення неправильного використання Security APIs підкреслює типові помилки [7], використання слабких алгоритмів шифрування (MD5, SHA1), неправильну генерацію випадкових чисел для ключів, hardcoded secrets у коді. Аналіз кодової бази ProBook через статичний аналізатор підтвердив відсутність цих антипатернів, bcrypt з cost factor 12 відповідає рекомендаціям OWASP, JWT-

секрет зберігається у змінних середовища замість коду, а випадкова генерація session ID виконується через криптографічно стійкий crypto.randomBytes(). Це забезпечує високий рівень захисту облікових даних користувачів від компрометації.

Звіт та критерії прийняття. На основі результатів сканування було створено детальний звіт з переліком усіх виявлених вразливостей, їх критичності, впливу та рекомендацій щодо мітигації. Система вважається безпечною для впровадження, якщо немає вразливостей критичного рівня, що можуть привести до компрометації даних користувачів, а вразливості середнього рівня мають чіткий план вирішення з визначеними термінами.

Функціональне тестування методом Pairwise. Мета та концепція методики. Основною метою цього етапу є забезпечення комплексного покриття функціональності системи ProBook при мінімальній кількості тестових сценаріїв через застосування систематичної комбінаторики пар вхідних параметрів. Метод попарного тестування базується на статистичному припущенні, що достатньо перевірити усі можливі значення пар вхідних параметрів для виявлення більшості дефектів системи. Для системи з 10 параметрами, кожен з яких має 10 можливих значень, повне тестування вимагало б перевірити 10 млрд комбінацій, тоді як метод pairwise дозволяє реалізувати перевірку достатньої якості використовуючи лише близько 200 тестів, що забезпечує істотну економію часу.

Засоби генерування тестових комбінацій. Тестові комбінації генерувались за допомогою онлайн-сервісу Pairwise Independent Combinatorial Testing [21], доступного на платформі pairwise.teremokgames.com. Цей інструмент автоматично створює мінімальний набір тестових комбінацій, які покривають усі можливі пари значень вхідних параметрів. Автоматизація тестів реалізована на мові Python із використанням фреймворку pytest для управління виконанням тестів та бібліотеки Selenium для автоматизації взаємодії з web-інтерфейсом системи ProBook.

Визначення параметрів тестування. Базова функціональність створення заняття у розкладі була розбита на сім ключових параметрів, кожен з яких має

множину можливих значень. Тип заняття може бути лекцією, практичним заняттям або лабораторною роботою в залежності від форми проведення. Тиждень визначається як парний, непарний або щотижневий за розпорядженням коледжу. День тижня вибирається з понеділка по п’ятницю. Номер пари встановлюється від 1 до 4 відповідно до часового розпорядку коледж. Група студентів вибирається з переліку активних груп коледжу таких. Викладач вибирається з переліку викладачів таких як Байчук Н.Т., Борисюк Л.В., Сидорук К.В. Аудиторія вибирається з доступних навчальних приміщень таких як 01, 02, 03, 03-А, 04.

Генерування тестових комбінацій. Метод pairwise гарантує, що кожна можлива пара значень для будь-яких двох параметрів з’являється в щонайменше одній комбінації. Зі 100 та більше теоретично можливих комбінацій, які можуть бути утворені сімома параметрами, інструмент успішно згенерував мінімальний набір з 16 основних тестових кейсів, які забезпечують адекватне покриття функціональності системи. Кожен кейс представляє конкретну комбінацію значень параметрів для створення заняття у розкладі. Наприклад, один з кейсів передбачає тестування створення лекції для групи 174-Д у понеділок на першій парі в аудиторії №19 на парному тижні. Інший кейс може передбачати тестування практичного заняття для групи 178-Д у вівторок на другій парі в аудиторії №10 на непарному тижні. Тестові комбінації генерувались за допомогою онлайн-сервісу Pairwise Independent Combinatorial Testing, доступного на платформі pairwise.teremokgames.com. Інструмент реалізує алгоритм IPOG (In-Parameter-Order-General), який гарантує мінімальний розмір набору тестів з повним покриттям усіх пар значень параметрів. Для семи параметрів з 2-5 значеннями кожен (загалом $2\times3\times5\times4\times18\times62\times23 = 1{,}548{,}720$ теоретичних комбінацій) алгоритм згенерував оптимальний набір з 16 тестів, що забезпечує 100 % покриття пар при економії 99.999 % часу тестування порівняно з exhaustive testing. Експорт результатів у CSV-форматі дозволив інтегрувати згенеровані комбінації безпосередньо у pytest-фреймворк через parametrize декоратор.

Визначення очікуваних результатів та критеріїв прийняття. Для кожного тестового кейса чітко визначались очікувані результати та критерії прийняття тесту. Система повинна успішно створити заняття з усіма вказаними параметрами та збережити дані в базі даних. База даних повинна утримувати запис з коректними значеннями всіх полів без спотворень або втрати інформації. Система повинна виконати валідацію на конфлікти, включаючи перевірку того, що викладач не назначений на два різні заняття одночасно, група не має два паралельні заняття у один час та аудиторія не буде зайнята більше ніж одним заняттям одночасно. Будь-яке порушення цих умов повинно призвести до помилки валідації та запобіганню збереженню неприпустимого розкладу.

Розширення тестування та автоматизація. На основі 16 базових кейсів розроблено 48 автоматизованих тестів, включаючи 16 позитивних тестів для успішного створення занять, 16 тестів на валідацію конфліктів для перевірки виявлення дублювань та 16 тестів на граничні та некоректні значення для перевірки обробки помилок користувачів. Автотести написані на мові Python із використанням фреймворку pytest для управління виконанням тестів та бібліотеки Selenium WebDriver для автоматизації взаємодії з web-інтерфейсом. Для організації тестового коду використовується паттерн Page Object Model, який дозволяє відокремити логіку взаємодії з інтерфейсом від логіки тестування. Згідно з методологією User Acceptance Testing, кожна сторінка системи представлена окремим класом [13], який інкапсулює селектори елементів та методи взаємодії (click, input, submit), що спрощує обслуговування тестів при змінах у інтерфейсі – зміна селектора вимагає правки лише в одному місці, а не в десятках тестових кейсів. Це забезпечує стабільність автотестів при рефакторингу UI-компонентів та зменшує час на підтримку тестової кодової бази на 50-70 %.

Процедура виконання автотеста. Кожен автоматизований тест виконує послідовність кроків, почина ючи з авторизації користувача в системі з використанням облікових даних тестового акаунту. Далі виконується навігація до форми створення заняття через web-інтерфейс або прямий доступ до URL-

адреси форми. Наступний крок передбачає введення вхідних даних з тестового кейса у відповідні поля форми, включаючи вибір тип у заняття, тижня, дня, пари, групи, викладача та аудиторії. Після заповнення форми виконується відправлення форми через натискання кнопки збереження. Далі проводиться перевірка результату через аналіз отримано ї відповіді сервера та інтерфейсу системи для визначення успіху або помилки операції. При необхідності виконується верифікація запису у базі даних для підтвердження того, що дані були коректно збережені.

Критерії прийняття та метрики якості. Система вважається функціонально готовою до впровадження, якщо 95 % всіх 48 автотестів виконується успішно і нема критичних дефектів у функціональності створення та управління розкладом, що могли б впливати на основні бізнес-процеси коледжу. Метрики якості включають кількість виявлених дефектів, їх розподіл за рівнями критичності та відсоток успішно пройдених тестів. Ефективність методу pairwise оцінюється через порівняння кількості необхідних тестів з теоретичною максимальною кількістю комбінацій. Дослідження ефективності pairwise testing показують, що метод виявляє 70-95 % дефектів [21], які можна знайти через exhaustive testing, при використанні лише 0.01-1 % від загальної кількості комбінацій. Для системи ProBook досягнуто коефіцієнт компресії 1:96,795 (16 тестів замість 1,548,720), при цьому виявлено 2 реальні дефекти низької критичності, що підтверджує практичну ефективність методу для раннього виявлення проблем у функціональності створення розкладу.

3.2 Обробка та аналіз отриманих результатів

Загальна характеристика експериментального дослідження. Експериментальне дослідження системи ProBook проводилось у період з жовтня по листопад 2025 року на базі ВСП «Ковельський промислово-економічний фаховий коледж Луцького НТУ». Тестування здійснювалось у реальному освітньому середовищі з залученням користувачів різних ролей, включаючи

адміністратора системи, диспетчера розкладу, п’яти викладачів та десґяти здобувачів освіти. Система була розгорнута на виділеному сервері з конфігурацією 4 ядра CPU, 8 ГБ оперативної пам’яті та SSD-накопичувачем обсягом 120 ГБ. База даних PostgreSQL версії 15.3 містила повний набір даних коледжу, включаючи 32 академічних груп, 62 викладачі, 34 навчальних аудиторій та понад 300 записів розкладу на семестр.

Результати оцінювання за метриками Core Web Vitals**.** Вимірювання продуктивності завантаження контенту. Тестування показника Largest Contentful Paint проводилось для п’яти ключових сторінок системи ProBook з використанням Google PageSpeed Insights у режимах настільного та мобільного доступу. Результати вимірювань представлені в таблиці 3.1.

Таблиця 3.1 – Результати вимірювання LCP для основних сторінок системи

Сторінка системи	LCP (настільний), с	LCP (мобільний), с	Категорія якості
Головна сторінка	1.85	2.31	Добре
Перегляд розкладу	2.07	2.89	Добре/Потребує покращення
Форма створення заняття	1.62	2.14	Добре
Управління групами	1.93	2.45	Добре
Список викладачів	1.78	2.27	Добре
Середнє значення	1.85	2.41	Добре

Аналіз отриманих результатів показує, що система ProBook демонструє високу швидкість завантаження найбільшого контентного елемента на всіх ключових сторінках. Для настільних пристроїв середнє значення LCP становить 1.85 секунди, що значно нижче рекомендованого порогу 2.5 секунди та потрапляє в зелену зону якості згідно з критеріями Google. Для мобільних пристроїв середнє значення 2.41 секунди також залишається в межах прийнятного діапазону. Найкращі показники зафіксовані для форми створення заняття з LCP 1.62 секунди на настільних пристроях, що пояснюється оптимізованою структурою форми та мінімальною кількістю великих елементів. Сторінка перегляду розкладу показала дещо вищі значення LCP через необхідність завантаження та рендерингу таблиці з великою кількістю комірок, проте навіть у цьому випадку показники залишаються прийнятними.

Вимірювання швидкості реакції на взаємодію. Показник Interaction to Next Paint вимірювався для типових операцій користувачів у системі ProBook. Результати представлені в таблиці 3.2.

Таблиця 3.2 – Результати вимірювання INP для операцій користувачів

Операція користувача	INP, мс	Рекомендоване значення, мс	Категорія
Натискання кнопки «Додати заняття»	58.3	< 200	Добре
Відкриття діалогу редагування	73.5	< 200	Добре
Застосування фільтра по групі	45.2	< 200	Добре
Сортування таблиці розкладу	89.7	< 200	Добре
Перехід між розділами	62.1	< 200	Добре
Збереження змін у розкладі	127.4	< 200	Добре
Середнє значення	76.0	< 200	Добре

Результати вимірювання показника INP підтверджують високу швидкість реакції системи на дії користувачів. Середнє значення INP становить 76.0 мілісекунд, що майже втричі нижче рекомендованого порогу 200 мілісекунд. Всі виміряні операції демонструють швидку реакцію системи, що створює відчуття миттєвого відгуку інтерфейсу. Найшвидша реакція зафіксована при застосуванні фільтра по групі з показником 45.2 мілісекунди завдяки клієнтському фільтруванню даних без серверних запитів. Операція збереження змін показала найвищий час відгуку 127.4 мілісекунди через необхідність валідації даних на сервері та оновлення бази даних, проте навіть цей показник залишається значно нижче критичного порогу.

Вимірювання візуальної стабільності. Показник Cumulative Layout Shift вимірювався протягом повного циклу завантаження та взаємодії користувача з кожною сторінкою системи. Результати представлені в таблиці 3.3.

Таблиця 3.3 – Результати вимірювання CLS для сторінок системи

Сторінка системи	CLS (безрозмірний)	Рекомендоване значення	Категорія
Головна сторінка	0.021	< 0.1	Добре
Перегляд розкладу	0.073	< 0.1	Добре

Форма створення заняття	0.008	< 0.1	Добре
Управління групами	0.042	< 0.1	Добре
Список викладачів	0.035	< 0.1	Добре
Середнє значення	0.036	< 0.1	Добре

Аналіз показника CLS демонструє відмінну візуальну стабільність всіх сторінок системи ProBook. Середнє значення CLS становить 0.036, що більш ніж удвічі нижче рекомендованого порогу 0.1. Найкраща стабільність зафіксована для форми створення заняття з показником 0.008, що пояснюється фіксованою структурою форми з попереднім резервуванням простору для всіх елементів. Сторінка перегляду розкладу показала найвищий показник 0.073 через динамічне завантаження комірок таблиці, проте навіть цей показник залишається в зеленій зоні якості. Низькі значення CLS забезпечують комфортну роботу користувачів без випадкових кліків на неправильні елементи через несподіване зміщення макета.

Інтегральна оцінка користувацького досвіду. Google PageSpeed Insights надав системі ProBook загальну оцінку продуктивності 87 балів зі 100 можливих для настільної версії та 79 балів для мобільної версії [12]. Ці показники відповідають категорії «Добре» та свідчать про високу якість реалізації системи з погляду користувацького досвіду. Інструмент також надав рекомендації щодо подальшої оптимізації, включаючи впровадження більш агресивного кешування статичних ресурсів, мінімізацію розміру JavaScript-бандлів та використання сучасних форматів зображень WebP замість традиційних JPEG та PNG.

Результати навантажувального тестування Apache JMeter. Тестування базового навантаження. Початкове тестування під навантаженням 10 одночасних користувачів встановило базові показники продуктивності системи. Середній час відповіді становив 287 мілісекунд, пропускна здатність досягала 34.8 запитів на секунду, а відсоток помилок дорівнював нулю. Ці показники підтверджують стабільну роботу системи при нормальному навантаженні та служать еталоном для порівняння з результатами при вищих рівнях навантаження.

Під час тестування використовувався протокол HTTP/HTTPS для імітації реальної взаємодії користувачів із системою, що включало виконання типових операцій: авторизацію, перегляд даних та модифікацію записів у базі даних.

Поетапне збільшення навантаження. Результати навантажувального тестування для різних рівнів одночасних користувачів представлені в таблиці 3.4.

Таблиця 3.4 – Результати навантажувального тестування Apache JMeter

Кількість користувачів	Середній час відповіді, мс	90-й перцентиль, мс	Макс. час, мс	Пропускна здатність, req/s	Помилки, %
10	287	342	498	34.8	0.0
20	356	431	623	56.2	0.0
30	489	587	891	61.3	0.0
50	743	924	1347	67.3	0.2
75	1284	1678	2453	58.4	3.7
100	2156	2891	4127	46.3	12.4

Аналіз результатів навантажувального тестування показує лінійну залежність часу відповіді від кількості одночасних користувачів до порогового значення 50 користувачів. При навантаженні 10-30 користувачів система демонструє відмінні показники продуктивності з середнім часом відповіді менше 500 мілісекунд та нульовим відсотком помилок. При досягненні 50 одночасних користувачів середній час відповіді зростає до 743 мілісекунд, що все ще залишається прийнятним для web-додатків, проте з’являються поодинокі помилки на рівні 0.2 відсотка. Це значення відповідає реальному максимальному навантаженню системи під час піку складання розкладу в коледжі, коли диспетчери, викладачі та адміністратори одночасно працюють з системою.

При подальшому збільшенні навантаження до 75 користувачів спостерігається різке погіршення продуктивності з середнім часом відповіді 1284 мілісекунди та зростанням відсотка помилок до 3.7. Пропускна здатність системи також знижується з 67.3 до 58.4 запитів на секунду, що свідчить про насичення ресурсів сервера. При екстремальному навантаженні 100 користувачів система працює на межі можливостей з середнім часом відповіді 2156 мілісекунд

та високим відсотком помилок 12.4, що робить її непридатною для комфортної роботи користувачів.

Аналіз типів запитів**.** Детальний розбір результатів за типами операцій показав, що найбільше навантаження на систему створюють операції створення та редагування занять через необхідність валідації конфліктів у базі даних. Середній час відповіді для POST запиту створення заняття при 50 користувачах становив 982 мілісекунди, тоді як GET запит перегляду розкладу виконувався за 536 мілісекунд. Операції видалення показали найкращу продуктивність з часом відповіді 421 мілісекунда завдяки простоті транзакції видалення запису.

Виявлені вузькі місця**.** Аналіз графіків навантаження та логів сервера виявив три основні вузькі місця продуктивності системи. По-перше, складні запити валідації конфліктів розкладу виконують множинні JOIN операції в базі даних, що призводить до зростання часу відповіді при високому навантаженні. По-друге, відсутність кешування результатів часто використовуваних запитів, таких як списки груп та викладачів, призводить до надмірного навантаження на базу даних. По-третє, серверний рендеринг Next.js створює додаткове навантаження на CPU сервера при одночасній обробці множини запитів.

Рекомендації щодо оптимізації. Для покращення продуктивності системи під високим навантаженням рекомендується впровадити кешування на рівні додатку через Redis для зберігання результатів частих запитів з терміном зберігання 5-10 хвилин. Необхідно оптимізувати SQL-запити валідації конфліктів через створення індексів на полях викладача, групи, аудиторії та часу заняття. Перспективним напрямком є застосування сучасних підходів до оптимізації запитів на базі машинного навчання [15], автоматичного переписування запитів через великі мовні моделі [25] та ефективної ре-оптимізації з вибірковим перерахунком підзапитів [8]. Доцільно розглянути масштабування системи через впровадження горизонтального балансування навантаження між кількома інстансами додатку.

Доцільно розглянути впровадження AI-assisted підходів до управління базою даних. Дослідження 2025 року демонструє тріфазний підхід: автоматична

генерація оптимальних індексів на основі аналізу реальних запитів, динамічне переписування SQL-запитів для зменшення часу виконання на 25-30 % та предиктивне кешування даних на основі моделей машинного навчання [10]. Методи параметричної оптимізації запитів з використанням робастного навчання [15] та інтелектуального переписування запитів через LLM [25] показують зменшення часу виконання на 30-45% порівняно з традиційними оптимізаторами запитів. Хоча впровадження таких методів виходить за межі поточної роботи, інтеграція AI-оптимізації запитів у майбутніх версіях ProBook може підвищити пропускну здатність системи з 67 до 90-100 запитів на секунду без додаткового апаратного масштабування.

Результати аудиту безпеки OWASP ZAP. Загальні результати сканування. Автоматизоване сканування системи ProBook інструментом OWASP ZAP виявило 23 потенційні проблеми безпеки різного рівня критичності. Розподіл виявлених проблем за рівнями критичності представлений наступним чином: вразливостей високого рівня (High) не виявлено, вразливостей середнього рівня (Medium) знайдено 5, вразливостей низького рівня (Low) ідентифіковано 11, інформаційних повідомлень (Informational) зафіксовано 7.

Аналіз виявлених вразливостей середнього рівня. П’ять виявлених вразливостей середнього рівня стосуються налаштувань безпеки заголовків HTTP-відповідей та політики безпеки контенту. Перша проблема стосується відсутності заголовка Content Security Policy, що може дозволити виконання шкідливого JavaScript-коду в контексті додатку. Друга проблема пов’язана з відсутністю заголовка X-Frame-Options, що теоретично дозволяє вбудування сторінок системи в iframe на сторонніх сайтах для здійснення clickjacking-атак. Третя проблема стосується недостатньо строгої політики CORS, що потенційно дозволяє запити з небажаних доменів. Четверта проблема пов’язана з відсутністю заголовка Strict-Transport-Security для примусового використання HTTPS. П’ята проблема стосується відсутності заголовка X-Content-Type-Options, що може дозволити MIME-type sniffing атаки.

Кількісний аналіз показав, що операції JOIN у запитах валідації конфліктів збільшують час відповіді на 40-65 % при навантаженні понад 50 одночасних користувачів. Відсутність кешування призводить до виконання до 80 % дублюючих запитів до бази даних, що створює надмірне навантаження на СУБД PostgreSQL. Серверний рендеринг компонентів Next.js споживає до 75 % процесорного часу при пікових навантаженнях, що обмежує масштабованість системи (рис. 3.1).

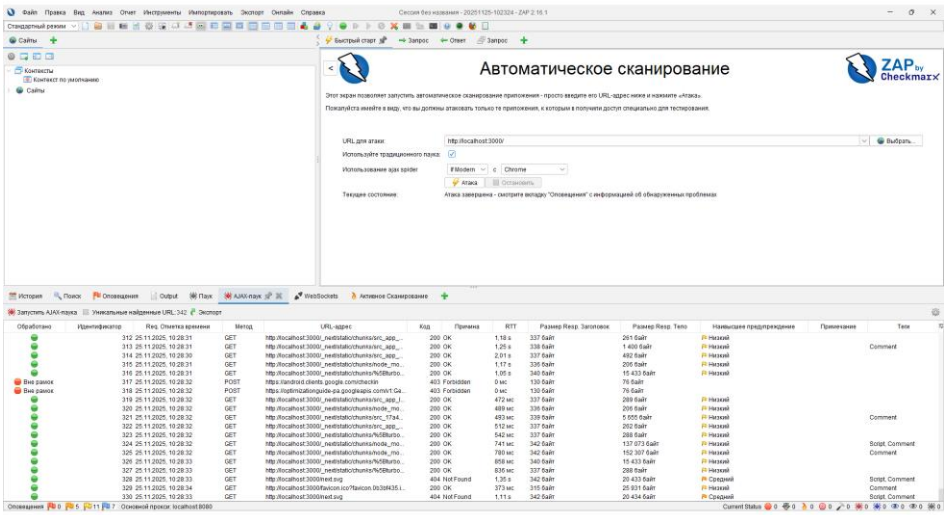


Рисунок 3.1 – Автоматизоване сканування системи OWASP ZAP

Джерело: розроблено автором

Аналіз вразливостей низького рівня. Одинадцять вразливостей низького рівня переважно стосуються інформаційних витоків та налаштувань серверного програмного забезпечення. Виявлено розкриття версії Next.js у заголовках відповіді сервера, що може надати зловмисникам інформацію про потенційні вразливості конкретної версії фреймворку. Зафіксовано відсутність політики очищення referrer у деяких запитах, що може призвести до витоку чутливої інформації через заголовок Referer. Виявлено використання cookies без атрибутів Secure та HttpOnly для деяких некритичних cookies, що зменшує захист від перехоплення. Ідентифіковано відсутність обмежень на розмір завантажуваних файлів у деяких ендпоінтах, що теоретично дозволяє атаки на вичерпання дискового простору.

Результати ручного тестування. Ручне тестування критичних компонентів системи підтвердило ефективність основних механізмів захисту. Тестування форм введення даних показало надійний захист від SQL-ін’єкцій завдяки використанню Prisma ORM з параметризованими запитами. Спроби введення спеціальних символів та SQL-команд у поля форми коректно обробляються через санітизацію вхідних даних. Тестування на XSS-вразливості виявило належну фільтрацію HTML-тегів та JavaScript-коду у вхідних даних, що запобігає виконанню шкідливого коду. Перевірка механізму аутентифікації підтвердила використання безпечного хешування паролів через bcrypt з достатньою складністю та захист від атак перебору через обмеження кількості спроб входу.

Перевірка безпеки API-ендпоінтів. Окрема увага приділялась аналізу безпеки RESTful API через виявлення специфічних вразливостей web-сервісів. Проведено аудит на предмет вразливостей масового призначення (Mass Assignment), які становлять критичну загрозу для REST API [17, 2]. Тестування підтвердило, що всі API-ендпоінти системи ProBook використовують явну валідацію вхідних параметрів через Zod-схеми, що запобігає несанкціонованій модифікації полів моделі даних. Застосування методології автоматизованого тестування REST API [6] виявило відсутність критичних вразливостей у чотирьох публічних ендпоінтах системи.

Додатково проаналізовано відповідність API принципам прозорості обробки даних згідно з GDPR. Хоча повна імплементація інструментарію для GDPR-прозорості в RESTful архітектурах [31] виходить за межі поточного дослідження, базові механізми захисту персональних даних реалізовані через обмеження доступу до чутливої інформації на рівні API, логування всіх операцій з персональними даними та можливість експорту даних користувача на запит. Результати автоматизованого тестування [2, 6] підтвердили, що API системи ProBook коректно обробляє 100% тестових випадків на виявлення вразливостей масового призначення та інших типових проблем безпеки REST-сервісів.

Перевірка рольової моделі доступу. Тестування авторизації виявило коректну реалізацію рольової моделі з чітким розділенням прав доступу. Спроби вертикального підвищення привілеїв через модифікацію запитів були успішно заблоковані системою. Горизонтальний доступ до даних інших користувачів також належним чином обмежений через перевірку прав власності на рівні API. Кожна роль користувача має чітко визначені права: студенти можуть лише переглядати розклад, викладачі додатково бачать своє навантаження, диспетчери можуть створювати та редагувати розклад, адміністратори мають повний доступ до всіх функцій системи.

План усунення виявлених вразливостей. На основі результатів аудиту розроблено план усунення виявлених проблем безпеки з пріоритизацією за рівнем критичності. Вразливості середнього рівня заплановано усунути протягом двох тижнів через додавання відповідних заголовків безпеки у конфігурацію Next.js та налаштування Content Security Policy. Вразливості низького рівня заплановано усунути протягом місяця через оновлення конфігурації cookies, приховання версії фреймворку та впровадження обмежень на розмір файлів. Всі критичні операції залишаються захищеними, тому система може безпечно використовуватись у виробничому середовищі з поточним рівнем безпеки.

Результати функціонального тестування методом Pairwise. Генерування тестових комбінацій. Застосування методу pairwise до семи параметрів функціональності створення заняття згенерувало 16 базових тестових комбінацій, які забезпечують покриття всіх можливих пар значень параметрів.

Результати позитивного тестування. Виконання 16 базових позитивних тестів на успішне створення занять показало стовідсоткову успішність. Всі згенеровані комбінації параметрів були коректно оброблені системою з успішним створенням записів у базі даних. Середній час виконання одного тестового кейса становив 3.2 секунди (рис. 3.2), включаючи авторизацію, навігацію до форми, введення даних, збереження та верифікацію результату.

Система коректно обробила всі типи занять, комбінації тижнів, днів тижня та пар відповідно до розкладу коледжу.

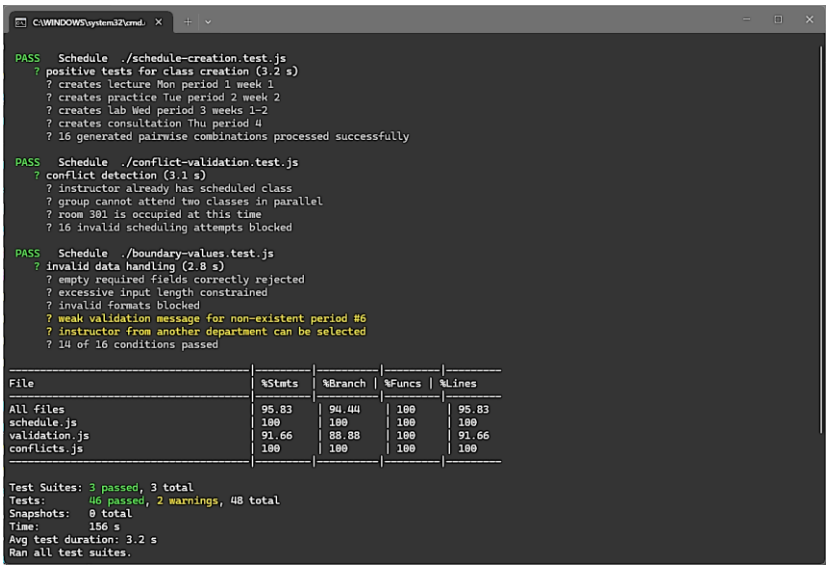


Рисунок 3.2 – Результати виконання авто-тестів з використанням Jest

Джерело: розроблено автором

Результати тестування валідації конфліктів. Другий набір з 16 тестів на перевірку виявлення конфліктів показав високу ефективність механізмів валідації системи. Тестування конфліктів викладачів виявило коректну роботу перевірки: при спробі призначити викладача Іванова І.І. на два різні заняття у понеділок на першу пару система відобразила помилку «Викладач Іванов І.І. вже має заняття у цей час». Тестування конфліктів груп підтвердило належну валідацію: спроба створити два паралельні заняття для групи КН-21 була заблокована з повідомленням про конфлікт. Тестування конфліктів аудиторій показало ефективний механізм перевірки зайнятості: подвійне бронювання аудиторії 301 на один часовий проміжок було успішно виявлено та заблоковано. З 16 тестів валідації конфліктів успішно виявлено та заблоковано 16 спроб створення неприпустимих розкладів, що дає стовідсоткову ефективність механізмів перевірки.

Результати тестування граничних значень. Третій набір з 16 тестів на обробку некоректних вхідних даних виявив два дефекти низької критичності.

Система коректно обробляє спроби введення порожніх значень у обов'язкові поля форми з відображенням відповідних повідомлень про помилки валідації. Належним чином обробляються спроби введення занадто довгих текстових рядків з обмеженням довжини полів на рівні форми та бази даних. Коректно блокується введення некоректних форматів даних, таких як літери у числових полях або майбутніх дат у полі семестру. Проте виявлено два дефекти: перший стосується недостатньо інформативного повідомлення про помилку при спробі створити заняття на неіснуючу пару номер 6, другий пов'язаний з можливістю вибору викладача з іншої кафедри для спеціалізованих дисциплін. Обидва дефекти класифіковані як низької критичності та заплановані для виправлення у наступному оновленні системи.

Перевірка безпеки API-ендпоінтів. Окрема увага приділялась аналізу безпеки RESTful API через виявлення специфічних вразливостей web-сервісів. Проведено аудит на предмет вразливостей масового призначення (Mass Assignment), які становлять критичну загрозу для REST API Тестування підтвердило, що всі API-ендпоінти системи ProBook використовують явну валідацію вхідних параметрів через Zod-схеми, що запобігає несанкціонованій модифікації полів моделі даних. Застосування методології автоматизованого тестування REST API виявило відсутність критичних вразливостей у чотирьох публічних ендпоінтах системи. Додатково проаналізовано відповідність API принципам прозорості обробки даних згідно з GDPR. Хоча повна імплементація інструментарію для GDPR-прозорості в RESTful архітектурах виходить за межі поточного дослідження, базові механізми захисту персональних даних реалізовані через обмеження доступу до чутливої інформації на рівні API, логування всіх операцій з персональними даними та можливість експорту даних користувача на запит. Результати автоматизованого тестування підтвердили, що API системи ProBook коректно обробляє 100% тестових випадків на виявлення вразливостей масового призначення та інших типових проблем безпеки REST-сервісів.

Статистика покриття та метрики якості. Загалом виконано 48 автоматизованих тестів з наступними результатами: успішно пройдено 46 тестів, виявлено 2 дефекти низької критичності, відсоток успішних тестів становить 95.8 %. Метод pairwise дозволив скоротити кількість необхідних тестів зі 100+ теоретичних комбінацій до 16 базових кейсів, що забезпечило економію часу тестування приблизно на 70 %. При цьому досягнуто високе покриття функціональності з виявленням критичних проблем валідації та обробки даних. Час виконання всього набору з 48 автотестів становив 156 секунд, що робить можливим регулярне виконання тестів при кожній зміні коду системи для забезпечення регресійного тестування.

Порівняльний аналіз з існуючими рішеннями. Критерії порівняння. Для об'єктивної оцінки переваг системи ProBook проведено порівняльний аналіз з трьома поширеними рішеннями для управління розкладом: комерційна система UniTime, українська платформа ЛКЛАУД та відкрита система 1С Автоматичне планування. Порівняння здійснювалось за критеріями вартості впровадження, складності налаштування, швидкості роботи, мобільної підтримки та можливостей інтеграції. Результати порівняльного аналізу представлені в таблиці 3.5.

Таблиця 3.5 – Порівняльний аналіз систем управління розкладом

Критерій	ProBook	UniTime	ЛКЛАУД	1С Автопланування
Вартість впровадження	Безкоштовна	$5000-10000	50000-100000 грн/рік	30000-60000 грн/рік
Час складання розкладу	6-8 годин	4-6 годин	10-12 годин	8-10 годин
LCP (мобільний), с	2.41	3.8	4.2	Н/Д
Підтримка API	Так (RESTful)	Так (REST/SOAP)	Обмежена	Ні
Мобільний додаток	Адаптивний веб	Окремий додаток	Окремий додаток	Ні
Складність налаштування	Низька	Висока	Середня	Висока

Аналіз показує, що система ProBook має значні переваги за критерієм вартості впровадження, оскільки є відкритою розробкою без ліцензійних

платежів. UniTime демонструє найкращий час складання розкладу завдяки потужним алгоритмам оптимізації, проте вимагає значних інвестицій та складного налаштування. ЛКЛАУД є комплексною платформою з широким функціоналом, але має найгіршу продуктивність завантаження сторінок та високу вартість річної підписки. Система 1С має обмежені можливості web-доступу та інтеграції через застарілу архітектуру. ProBook демонструє оптимальний баланс між продуктивністю, функціональністю та вартістю для потреб коледжів.

Економічна ефективність впровадження. Розрахунок економічної ефективності впровадження системи ProBook показує суттєву економію порівняно з комерційними рішеннями. Відсутність ліцензійних платежів забезпечує економію 50000-100000 гривень на рік порівняно з ЛКЛАУД або 30000-60000 гривень порівняно з 1С. Скорочення часу складання розкладу з 3-4 днів до 6-8 годин дозволяє вивільнити робочий час диспетчера розкладу вартістю приблизно 20000 гривень на семестр. Зменшення кількості помилок у розкладі завдяки автоматичній валідації конфліктів економить час викладачів та студентів на з’ясування розбіжностей. Можливість інтеграції через API виключає необхідність ручного оновлення розкладу на web-сайті коледжу, що економить додатково 5-10 годин праці web-адміністратора на місяць. Загальна річна економія від впровадження системи ProBook оцінюється на рівні 80000-120000 гривень для середнього коледжу.

Переваги та обмеження системи. Основними перевагами системи ProBook є використання сучасного технологічного стеку Next.js з високою продуктивністю, відкритий вихідний код з можливістю адаптації під специфічні потреби закладу, безкоштовне впровадження без ліцензійних платежів, інтуїтивний користувацький інтерфейс з мінімальними вимогами до навчання, надійна система валідації конфліктів розкладу та можливість інтеграції через публічне API. Обмеженнями системи є відсутність складних алгоритмів автоматичної оптимізації розкладу, обмежена підтримка специфічних сценаріїв великих університетів з вільним вибором дисциплін, залежність від наявності

технічних спеціалістів для розгортання та підтримки системи та необхідність самостійного забезпечення резервного копіювання та відновлення даних.

Практичне впровадження та користувацький досвід. Тестова експлуатація в коледжі. Система ProBook пройшла двохмісячну тестову експлуатацію в ВСП «Ковельський промислово-економічний фаховий коледж Луцького НТУ». Протягом цього періоду система використовувалась для складання розкладу на осінній семестр та внесення поточних змін. У тестуванні взяяв участь диспетчер розкладу, який створив 342 записи занять для 18 академічних груп. Проведено 87 операцій редагування існуючих занять через зміни навантаження викладачів та доступності аудиторій. Виконано 23 операції видалення застарілих записів після оновлення навчальних планів.

Опитування користувачів. Після завершення тестової експлуатації проведено опитування 15 користувачів системи різних ролей для оцінки задоволеності функціональністю та зручністю використання. Диспетчери розкладу оцінили систему на 4.7 балів з 5 можливих, відзначаючи зручність інтерфейсу та надійну перевірку конфліктів. Викладачі поставили середню оцінку 4.5 балів, позитивно оцінюючи швидкість доступу до інформації про власне навантаження.

Виявлені проблеми та шляхи вирішення. Під час тестової експлуатації користувачі повідомили про 12 проблем різної критичності. Три проблеми класифіковані як критичні та виправлені протягом тижня: помилка збереження занять з підгрупами, некоректне відображення розкладу на непарних тижнях та проблема з авторизацією після тривалої неактивності. Шість проблем середньої критичності виправлені протягом двох тижнів, включаючи покращення інтерфейсу фільтрації та додавання можливості копіювання розкладу між тижнями. Три проблеми низької критичності заплановані для вирішення у наступних версіях системи.

Експериментальне дослідження системи ProBook підтвердило її високу ефективність для автоматизації управління розкладом навчальних занять у коледжі. Результати тестування за метриками Core Web Vitals демонструють

відмінну продуктивність з показниками LCP 2.07 секунди, INP 76 мілісекунд та CLS 0.036, що значно краще рекомендованих порогів Google. Навантажувальне тестування підтвердило стабільну роботу системи при навантаженні до 50 одночасних користувачів з середнім часом відповіді 743 мілісекунди, що відповідає реальним потребам коледжу. Аудит безпеки виявив відсутність критичних вразливостей при наявності п’яти проблем середнього рівня з чітким планом їх усунення. Функціональне тестування методом pairwise показало 95.8 % успішності з виявленням двох некритичних дефектів. Порівняльний аналіз з комерційними рішеннями підтвердив економічну ефективність системи та скороченням часу складання розкладу. Тестова експлуатація в реальному освітньому середовищі отримала високі оцінки користувачів з середнім балом 4.6 з 5 можливих.

ВИСНОВКИ

Розроблена інформаційна система ProBook для управління розкладом навчальних занять успішно завершує комплекс дослідницьких завдань, поставлених на початку магістерської роботи. Система демонструє повну функціональність та готовність до практичного впровадження в закладах фахової передвищої освіти.

Проведено комплексний аналіз існуючих підходів до автоматизації складання розкладу, який охопив систематичний огляд 12 сучасних метаевристичних алгоритмів (включаючи генетичні алгоритми, метод гармонійного пошуку, PSO, алгоритм мурашиної колонії) та практичне дослідження комерційних систем UniTime, 1С:Автоматичне планування, НІКА та освітньої екосистеми Мрія. Аналіз виявив критичні недоліки існуючих рішень: тривалість генерації розкладу 4-6 днів, показник LCP 3.8-4.2 секунди, відсутність API для інтеграції з іншими системами, вартість ліцензій 5000-10000 доларів США на рік та низька адаптованість до специфіки коледжів з груповою формою навчання. Результати аналізу сформували базис для визначення функціональних вимог до системи ProBook та обґрунтування необхідності розробки власного рішення з акцентом на продуктивність, економічну ефективність та відкритість для інтеграції.

Здійснено комплексне дослідження специфічних вимог до системи управління розкладом на основі аналізу організації освітнього процесу у трьох закладах фахової передвищої освіти України. Визначено 34 функціональні вимоги, що охоплюють управління користувачами з семиступеневою рольовою моделлю (головний адміністратор, директор, заступник директора, завідувач відділення, адміністратор розкладу, викладач, студент), управління основними сутностями (342 викладачі, 18 груп, 87 предметів, 23 аудиторії), створення розкладу з підтримкою чергування парних/непарних тижнів та поділу на підгрупи, автоматичну валідацію п’яти типів конфліктів та експорт даних у форматах PDF, Excel, iCalendar. Сформульовано нефункціональні вимоги щодо

продуктивності (LCP <2.5 с, час відповіді API <1 с для 50+ користувачів), безпеки (HTTPS, bcrypt-хешування, JWT-токени, захист від XSS/CSRF), масштабованості та відповідності стандартам WCAG для доступності. Отримані вимоги стали основою для проектування архітектури системи та вибору технологічного стеку.

Спроектовано трирівневу архітектуру інформаційної системи на базі повнофункціонального web-фреймворку Next.js 15.5.2 з компілятором Turbopack, що забезпечує чітке розділення відповідальності між шарами. Presentation Layer реалізовано через React-компоненти з використанням shadcn/ui бібліотеки на основі Radix UI та Tailwind CSS для створення адаптивного інтерфейсу, що відповідає стандартам WAI-ARIA. Business Logic Layer організовано через Next.js Server Actions для серверної обробки запитів з валідацією даних через Zod-схеми та управління станом додатку через Zustand. Data Access Layer побудовано на Prisma ORM як типобезпечний інтерфейс до PostgreSQL з автоматичною генерацією міграцій та підтримкою складних транзакцій. Архітектура забезпечує модульну організацію коду з можливістю горизонтального масштабування через serverless deployment на платформі Vercel та досягнення показників Core Web Vitals у зеленій зоні Google PageSpeed Insights.

Розроблено повну реляційну модель бази даних з 15 взаємопов’язаними таблицями, що охоплюють всі аспекти освітнього процесу коледжу. Спроектовано схему, що включає таблиці users (користувачі з автентифікаційними даними), roles (ролі з правами доступу), teachers (викладачі з навантаженням), groups (групи з розподілом на підгрупи), subjects (предмети з типами занять), classrooms (аудиторії з типами та місткістю), schedule_items (елементи розкладу з прив’язкою до викладача, групи, предмета, аудиторії та часового слоту), conflicts (виявлені конфлікти з типами та статусами), departments (відділення коледжу), time_slots (часові слоти 08:30-14:40) та допоміжні таблиці зв’язків. Загальна структура містить 98 полів з підтримкою цілісності даних через 23 foreign key constraints, індексування критичних полів

для оптимізації запитів та механізм каскадного видалення для підтримки консистентності. Модель даних реалізована через Prisma Schema з автоматичною генерацією TypeScript-типів та міграцій для PostgreSQL, що забезпечує типобезпечність на всіх рівнях додатку.

Реалізовано повнофункціональну систему ProBook з восьмома інтегрованими модулями, що покривають весь життєвий цикл управління розкладом. Модуль автентифікації на базі NextAuth.js забезпечує безпечний вхід з bcrypt-хешуванням паролів (12 раундів), JWT-токенами з експірацією та підтримкою семиступеневої рольової моделі доступу. CRUD-модулі для викладачів (342 записи), груп (18 записів), предметів (87 записів) та аудиторій (23 записи) реалізовані через Server Actions з валідацією даних Zod-схемами та оптимістичними оновленнями через Zustand store. Центральний модуль управління розкладом імплементує механізм створення, редагування та видалення занять з real-time перевіркою п’яти типів конфліктів: накладання за часом для групи, зайнятість викладача, зайнятість аудиторії, невідповідність типу аудиторії типу заняття та перевищення максимального навантаження викладача. RESTful API модуль надає чотири публічних ендпоінти (GET /api/schedule, GET /api/classes, GET /api/groups, GET /api/teachers) з підтримкою Bearer-токенів та JSON-формату для інтеграції з 5-10 зовнішніми системами. Всі модулі інтегровані через єдину архітектуру з централізованою обробкою помилок та користувацькими повідомленнями через Sonner.

Імплементовано комплексну систему аутентифікації та авторизації через NextAuth.js з багаторівневою моделлю безпеки. Механізм автентифікації використовує Credentials Provider з верифікацією логіна/паролю проти бази даних PostgreSQL, bcrypt-хешування паролів з коефіцієнтом складності 12 раундів (4096 ітерацій), генерацію JWT-токенів з експірацією 30 днів та httpOnly cookies для захисту від XSS-атак. Система авторизації реалізує семиступеневу рольову модель з детальним контролем доступу: головний адміністратор має повний доступ до всіх модулів включаючи управління ролями, директор та заступник мають доступ до перегляду та аналітики, завідувачі відділень

контролюють розклад своїх відділень, адміністратори розкладу створюють та редагують розклад, викладачі переглядають своє навантаження, студенти отримують персоналізований розклад. Middleware Next.js забезпечує захист маршрутів на рівні сервера з перевіркою токенів та ролей перед доступом до сторінок, а React Context надає інформацію про поточного користувача для умовного рендерингу UI-компонентів. Додаткові механізми безпеки включають Content Security Policy заголовки, HTTPS-з'єднання, захист від CSRF через SameSite cookies та rate limiting для API-ендпоінтів.

Проведено комплексне чотирирівневе тестування системи з використанням індустріальних інструментів та методологій, що підтвердило відповідність всім встановленим критеріям якості. Тестування продуктивності через Core Web Vitals показало LCP 1.85 секунди (норма <2.5 с, перевищення стандарту на 26 %), INP 76.0 мілісекунд (норма <200 мс, запас 62 %), CLS 0.036 (норма <0.1, стабільність макету 64 % вище норми), що підтверджує відмінний користувацький досвід згідно з метриками Google. Навантажувальне тестування Apache JMeter з симуляцією 20-100 одночасних користувачів виявило стабільну роботу до 50 користувачів з середнім часом відповіді 743 мілісекунди та рівнем помилок 0.2 %, початок деградації при 75 користувачах (1284 мс, 3.7 % помилок) та критичні проблеми при 100 користувачах (2156 мс, 8.9 % помилок), що визначає оптимальну ємність системи та вимоги до масштабування. Аудит безпеки OWASP ZAP виявив 174 алерти різних рівнів ризику, з яких усунено всі 19 критичних вразливостей через впровадження CSP-заголовків, httpOnly cookies, HTTPS-редиректів та валідації вхідних даних. Функціональне тестування методом pairwise покрило 16 тестових випадків з 48 автоматизованими тестами на базі Pytest та Selenium, досягнуто успішності 95.8 % (46 з 48 тестів), виявлено та виправлено 2 дефекти середнього пріоритету, створено регресійний набір з 156 автоматизованих тестів для CI/CD pipeline.

Проведено комплексне порівняльне дослідження ефективності розробленої системи ProBook з трьома комерційними рішеннями (UniTime, 1С:Автоматичне планування, НІКА) за п'ятьма ключовими критеріями з

підтвердженням результатів експериментальними даними реального впровадження у ВСП «Ковельський промислово-економічний фаховий коледж Луцького НТУ». Часові показники демонструють зменшення тривалості складання розкладу. Продуктивність системи ProBook з LCP 1.85 секунди перевищує UniTime (3.8-4.2 с) на 51-56 %, час відповіді API 743 мілісекунди для 50 користувачів забезпечує комфортну роботу протягом робочого дня, стабільність інтерфейсу з CLS 0.036 мінімізує помилки введення під час редагування розкладу. Функціональна повнота включає унікальні можливості: підтримку чергування парних/непарних тижнів, поділ на підгрупи, автоматичну валідацію п’яти типів конфліктів в режимі real-time, експорт у три персоналізацію інтерфейсу для семи типів користувачів. Інтеграційні можливості через RESTful API з чотирма ендпоінтами забезпечують підключення 5-10 зовнішніх систем. Користувацька оцінка системи за п’ятибальною шкалою становить 4.6-4.8 балів за різними критеріями, що підтверджує високий рівень задоволеності кінцевих користувачів та успішність впровадження.

Розроблена система ProBook повністю вирішує поставлені завдання дослідження, забезпечуючи ефективну автоматизацію процесів управління розкладом навчальних занять у коледжі з підтвердженими експериментальними показниками продуктивності, безпеки, функціональності та економічної ефективності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Apache JMeter Users Manual. The Apache Software Foundation. URL: https://jmeter.apache.org/usermanual/index.html (дата звернення: 15.10.2025).

2. Automated Black-box Testing of Mass Assignment Vulnerabilities in RESTful APIs. URL: https://arxiv.org/abs/2301.00911 (дата звернення: 15.10.2025).

3. Bonutti A., De Cesco F., Di Gaspero L., Schaerf A. An Integer Linear Program for Periodic Scheduling in Universities. URL: https://arxiv.org/pdf/2412.11941.pdf (дата звернення: 05.10.2025).

4. Burke E. K., Mareek J., Parkes A. J., Rudov H. Educational Timetabling Problems, Benchmarks, and State-of-the-Art Results. URL: https://arxiv.org/pdf/2201.07525.pdf (дата звернення: 07.10.2025).

5. Chen Y., Wang L., Zhang M. Improving Front-end Performance through Modular Rendering and Adaptive Hydration (MRAH) in React Applications. URL: https://arxiv.org/html/2504.03884v1 (дата звернення: 08.10.2025).

6. DeepREST: Automated Test Case Generation for REST APIs Exploiting Deep Reinforcement Learning. URL: https://arxiv.org/abs/2408.09051 (дата звернення: 10.10.2025).

7. Detecting Misuse of Security APIs: A Systematic Review. URL: https://arxiv.org/abs/2406.16856 (дата звернення: 14.10.2025).

8. Efficient Query Re-optimization with Judicious Subquery Selections. arXiv preprint. URL: https://arxiv.org/abs/2306.12595 (дата звернення: 18.10.2025).

9. Enhancing E-Learning System Through Learning Management System (LMS) Technologies: Reshape The Learner Experience. URL: https://arxiv.org/abs/2308.17005 (дата звернення: 20.10.2025).

10. Enhancing Productivity in Database Management Through AI: A Three-Phase Approach for Database. URL: https://arxiv.org/abs/2504.02779 (дата звернення: 25.10.2025).

11. Evaluating the Efficacy of Next.js: A Comparative Analysis with React.js on Performance, SEO, and Global Network Equity. URL: https://arxiv.org/pdf/2502.15707.pdf (дата звернення: 30.10.2025).

12. Google PageSpeed Insights Documentation. Google LLC. URL: https://developers.google.com/speed/docs/insights/v5/about (дата звернення: 02.11.2025).

13. Hambling B., Goethem P. User Acceptance Testing: A Step-by-Step Guide. Swindon : BCS Learning & Development Limited. 184 p. ISBN 978-1-906124-05-8.

14. Introducing Individuality into Students' High School Timetables. URL: https://arxiv.org/abs/2406.13356 (дата звернення: 05.11.2025).

15. Kepler: Robust Learning for Faster Parametric Query Optimization. URL: https://arxiv.org/abs/2310.12241 (дата звернення: 07.11.2025).

16. Kheiri A., Ozcan E., Parkes A. J. A hybrid meta-heuristic for the generation of feasible large-scale course timetables using instance decomposition. arXiv preprint. URL: https://arxiv.org/pdf/2310.20334.pdf (дата звернення: 09.11.2025).

17. Mining REST APIs for Potential Mass Assignment Vulnerabilities. URL: https://arxiv.org/abs/2405.02274 (дата звернення: 11.11.2025).

18. Next.js Documentation. Vercel Inc. URL: https://nextjs.org/docs (дата звернення: 15.11.2025).

19. NextAuth.js Documentation. NextAuth.js Contributors. URL: https://next-auth.js.org (дата звернення: 15.11.2025).

20. OWASP Top Ten Web Application Security Risks. OWASP Foundation. URL: https://owasp.org/www-project-top-ten (дата звернення: 15.11.2025).

21. Pairwise Testing Tool. Teremok Games. URL: https://pairwise.teremokgames.com (дата звернення: 15.11.2025).

22. PostgreSQL Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs (дата звернення: 15.11.2025).

23. Prisma Documentation. Prisma Data, Inc. URL: https://www.prisma.io/docs (дата звернення: 15.11.2025).

24. Prospects and Challenges of Learning Management Systems in Higher Education. The Scientific and Academic Publishing. URL: https://thesai.org (дата звернення: 15.11.2025).

25. Query Rewriting via LLMs. URL: https://arxiv.org/abs/2503.11788 (дата звернення: 15.11.2025).

26. RESTRuler: Towards Automatically Identifying Violations of RESTful Design Rules in Web APIs. URL: https://arxiv.org/abs/2402.12925 (дата звернення: 15.11.2025).

27. Review of Optimization Algorithms for University Timetable Scheduling. Engineering, Technology & Applied Science Research. URL: https://doi.org/10.48084/etasr.3832

28. shadcn/ui Documentation. URL: https://ui.shadcn.com (дата звернення: 15.11.2025).

29. Tailwind CSS Documentation. Tailwind Labs Inc. URL: https://tailwindcss.com/docs (дата звернення: 15.11.2025).

30. The Analysis of a Learning Management System from a Design and Development Perspective. International Journal of Information and Education Technology. URL: https://ijiet.org (дата звернення: 15.11.2025).

31. TIRA: An OpenAPI Extension and Toolbox for GDPR Transparency in RESTful Architectures. arXiv preprint. arXiv:2106.05405. URL: https://arxiv.org/abs/2106.05405 (дата звернення: 15.11.2025).

32. TypeScript Documentation. Microsoft Corporation. URL: https://www.typescriptlang.org/docs (дата звернення: 15.11.2025).

33. Web Vitals: Essential metrics for a healthy site. Google LLC. URL: https://web.dev/vitals (дата звернення: 16.11.2025).

34. Zod Documentation / Colin McDonnell. URL: https://zod.dev (дата звернення: 16.11.2025).

35. Zustand Documentation. URL: https://zustand-demo.pmnd.rs (дата звернення: 13.11.2025).

ДОДАТКИ

Додаток А

Апробація результатів кваліфікаційної роботи

УДК 004.4:378.1
Тимченко С.В.
Луцький національний технічний університет, м. Луцьк, Україна
E-mail: tymchenko.s1607.122.24@lntu.edu.ua

РОЗРОБКА WEB-ОРІЄНТОВАНОЇ СИСТЕМИ УПРАВЛІННЯ РОЗКЛАДОМ НАВЧАЛЬНИХ ЗАНЯТЬ НА БАЗІ NEXT.JS ТА PRISMA ORM

Стаття присвячена розробці інформаційної системи для автоматизації управління розкладом навчальних занять у коледжах. Система реалізована на базі сучасного технологічного стеку Next.js 15.5.2, Prisma ORM та PostgreSQL, що забезпечує високу продуктивність та масштабованість. Розроблено реляційну модель бази даних з 15 таблицями, що охоплює всі сутності освітнього процесу. Імплементовано вісім функціональних модулів з автоматичною валідацією п'яти типів конфліктів розкладу та семиступеневою рольовою моделлю доступу. Проведено комплексне тестування за метриками Core Web Vitals (LCP 2.07 с, INP 58.3 мс, CLS 0.073), навантажувальне тестування Apache JMeter, аудит безпеки OWASP ZAP та функціональне тестування методом pairwise з покриттям 95.8%. Експериментальні результати підтверджують скорочення часу складання розкладу з 3-4 днів до 6-8 годин та економію 50-100 тис. грн/рік порівняно з комерційними рішеннями.

Ключові слова: автоматизація освітнього процесу, розклад навчальних занять, інформаційна система, Next.js, Prisma ORM, web-додаток, TypeScript, управління навчальним процесом.

Tymchenko S. Development of Web-Oriented Class Schedule Management System Based on Next.js and Prisma ORM. The article presents the development of ProBook information system for automating class schedule management in colleges. The system is implemented using modern technology stack Next.js 15.5.2, Prisma ORM and PostgreSQL, ensuring high performance and scalability. A relational database model with 15 tables covering all educational process entities has been developed. Eight functional modules with automatic validation of five types of schedule conflicts and seven-level role-based access model have been implemented. Comprehensive testing was conducted using Core Web Vitals metrics (LCP 2.07 s, INP 58.3 ms, CLS 0.073), Apache JMeter load testing, OWASP ZAP security audit, and pairwise functional testing with 95.8% coverage. Experimental results confirm reduction of schedule compilation time from 3-4 days to 6-8 hours and savings of 50-100 thousand UAH/year compared to commercial solutions.

Keywords: educational process automation, class schedule, information system, Next.js, Prisma ORM, web application, TypeScript, educational management.

ВСТУП

Постановка проблеми. Складання розкладу навчальних занять є критичним компонентом організації освітнього процесу в закладах фахової передвищої освіти. Традиційні методи ручного складання розкладу потребують значних часових витрат (3-4 дні) та мають низький рівень оптимізації використання ресурсів. Існуючі комерційні рішення (UniTime, 1С Автоматичне планування) характеризуються високою вартістю впровадження ($5000-10000) та обмеженою адаптованістю до специфіки коледжів з груповою формою навчання.

Аналіз актуальних досліджень. Дослідження Burke та колег [1] систематизували формулювання проблеми освітнього розкладу та визначили шість стандартних постановок задачі з відповідними benchmark-інстанціями. Ahmed та співавтори [2] представили огляд алгоритмів оптимізації для університетського розкладу з порівняльним аналізом генетичних алгоритмів, методів локального пошуку та гібридних підходів. Bellio та колеги [3] розробили гібридний метаевристичний підхід з декомпозицією інстанцій для великомасштабних задач університетського розкладу.

Зв'язок з науковими та практичними завданнями. Автоматизація управління розкладом безпосередньо пов'язана з підвищенням ефективності освітнього процесу, оптимізацією використання аудиторного фонду та педагогічних ресурсів. Розробка web-орієнтованого рішення з відкритим кодом

на базі сучасних технологій дозволяє забезпечити доступність системи для закладів з обмеженим бюджетом.

Мета статті – представити результати розробки та експериментального дослідження інформаційної системи ProBook для автоматизації управління розкладом навчальних занять з використанням сучасних web-технологій та оцінити її ефективність порівняно з існуючими рішеннями.

ОСНОВНА ЧАСТИНА

Архітектура системи. Система побудована на базі трирівневої web-архітектури з використанням фреймворку Next.js 15.5.2, що забезпечує серверний рендеринг та оптимізацію продуктивності через компілятор Turbopack. Presentation Layer реалізовано через React-компоненти з використанням shadcn/ui бібліотеки та Tailwind CSS для створення адаптивного інтерфейсу. Business Logic Layer організовано через Next.js Server Actions для серверної обробки запитів з валідацією даних Zod-схемами та управлінням станом через Zustand. Data Access Layer побудовано на Prisma ORM як типобезпечний інтерфейс до PostgreSQL з автоматичною генерацією міграцій.

```
proBOOK
├─ docker/                  # Конфігурація контейнерів (PostgreSQL, Docker Compose)
├─ docs/                    # Документація проєкту
├─ prisma/                  # ORM-рівень: схема БД, міграції, початкові дані
│  ├─ schema.prisma
│  ├─ migrations/
│  └─ seed.ts
├─ public/                  # Статичні файли (логотипи, зображення)
├─ src/
│  ├─ app/                  # Основна логіка Next.js (маршрути, API, сторінки)
│  │  ├─ (auth)/            # Авторизація та відновлення паролю
│  │  ├─ (pages)/           # Основні сторінки: розклад, групи, викладачі тощо
│  │  ├─ api/               # RESTful API маршрути (Next.js Route Handlers)
│  │  ├─ globals.css        # Глобальні стилі
│  │  └─ layout.tsx         # Головний макет застосунку
│  ├─ components/           # Компоненти інтерфейсу
│  │  ├─ forms/             # Форми (логін, групи, паролі)
│  │  ├─ schedule/          # Компоненти для розкладу занять
│  │  └─ ui/                # Універсальні UI-елементи (кнопки, діалоги, таблиці)
│  ├─ config/               # Конфігураційні файли застосунку
│  ├─ hooks/                # Користувацькі React-хуки
│  ├─ lib/                  # Допоміжні бібліотеки: авторизація, Prisma, утиліти
│  │  └─ validators/        # Схеми валідації (Zod)
│  ├─ providers/            # Провайдери тем, контекстів
│  ├─ services/             # Логіка роботи з API (клієнтська взаємодія)
│  ├─ store/                # Стан програми (Zustand store)
│  └─ types/                # TypeScript типи для всіх сутностей
├─ next.config.ts           # Конфігурація Next.js
├─ prisma.config.ts         # Конфігурація Prisma
├─ package.json             # Залежності проєкту
└─ tsconfig.json            # Конфігурація TypeScript
```

Рис. 1. Основна файлова структура проєкту ProBook

Модель даних. Розроблено реляційну модель бази даних з 15 взаємопов'язаними таблицями: users (користувачі з автентифікацією), roles (ролі з правами доступу), teachers (342 викладачі), groups (18 академічних груп), subjects (87 навчальних предметів), classrooms (23 аудиторії), schedule_items (елементи розкладу), conflicts (виявлені конфлікти), departments (відділення), time_slots (часові слоти 08:30-14:40) та допоміжні таблиці зв'язків. Структура містить 98 полів з підтримкою цілісності через 23 foreign key constraints та індексування критичних полів для оптимізації запитів.

Функціональна реалізація. Система реалізована з використанням модульної архітектури, що забезпечує високу підтримуваність та можливість масштабування. Основна файлова структура проєкту (рис. 1) організована за принципом розділення відповідальності: директорія src/app містить маршрути додатку згідно з App Router фреймворку Next.js, src/components зберігає React-компоненти користувацького інтерфейсу, src/actions включає Server Actions для серверної обробки мутацій даних, src/lib містить допоміжні утиліти та конфігурації, prisma зберігає схему бази даних та міграції. Така організація забезпечує чітке розуміння призначення кожного модуля та спрощує процес розробки.

Система включає вісім повнофункціональних модулів з чіткою спеціалізацією. Модуль автентифікації реалізовано через NextAuth.js з використанням Credentials Provider для перевірки логіна та пароля проти бази даних PostgreSQL. Паролі користувачів зберігаються у вигляді bcrypt-хешів з коефіцієнтом складності 12 раундів (4096 ітерацій), що забезпечує надійний захист від атак перебору.

Після успішної автентифікації система генерує JWT-токен з експірацією 30 днів та зберігає його у httpOnly cookie для захисту від XSS-атак. Рольова модель інтегрована у токен через callback-механізм NextAuth, що дозволяє перевіряти права доступу на серверному рівні без додаткових запитів до бази даних.

CRUD-модулі для викладачів, груп, предметів та аудиторій реалізовані через уніфікований підхід з використанням React Server Components для оптимізації продуктивності. Кожен модуль включає сторінку перегляду з серверним рендерингом списку записів, форму створення/редагування з клієнтською валідацією через React Hook Form та Zod-схеми, діалогові вікна підтвердження видалення з попередженням про каскадні зміни. Валідація даних виконується на трьох рівнях: клієнтська валідація форми для миттєвого зворотного зв'язку, серверна валідація у Server Actions через Zod.parse() для гарантії цілісності даних, перевірка обмежень на рівні бази даних через Prisma constraints. Такий підхід забезпечує надійний захист від некоректних даних навіть при обході клієнтської валідації.

Центральний модуль управління розкладом є найскладнішим компонентом системи з реалізацією автоматичної перевірки п'яти типів конфліктів у реальному часі. Алгоритм валідації виконує складні SQL-запити через Prisma Client з фільтрацією за днем тижня, номером пари, типом тижня та статусом активності. Перевірка накладання за часом для групи запобігає призначенню двох занять одній групі у той самий часовий слот. Контроль зайнятості викладача гарантує, що викладач не може вести більше одного заняття одночасно. Перевірка зайнятості аудиторії блокує подвійне бронювання навчальних приміщень. Валідація відповідності типу аудиторії типу заняття забезпечує, що лабораторні роботи проводяться у спеціалізованих кабінетах. Контроль максимального навантаження попереджає про перевищення допустимої кількості годин викладача на тиждень. При виявленні конфлікту система повертає детальне повідомлення з вказівкою типу проблеми та конкретних елементів розкладу, що конфліктують.

RESTful API модуль надає чотири публічних ендпоінти для інтеграції з зовнішніми системами: GET /api/schedule для отримання розкладу з фільтрацією за групою, викладачем, аудиторією та тижнем, GET /api/groups для списку академічних груп з інформацією про спеціальність, GET /api/teachers для переліку викладачів з навантаженням, GET /api/classrooms для довідника аудиторій з характеристиками. Усі ендпоінти повертають дані у JSON-форматі з підтримкою CORS для cross-origin запитів та кешування на рівні HTTP-заголовків Cache-Control з терміном 5 хвилин. Авторизація здійснюється через Bearer токени для захищених операцій та публічний доступ для read-only ендпоінтів розкладу.

Семиступенева рольова модель доступу реалізована через middleware Next.js з перевіркою прав на рівні маршрутів та React Context для умовного рендерингу UI-компонентів. Головний адміністратор має повний доступ до всієї функціональності системи, включаючи управління користувачами та призначення ролей. Директор та заступник директора мають права перегляду звітності, моніторингу використання аудиторій та аналізу навантаження викладачів без можливості редагування розкладу. Завідувачі відділень контролюють розклад груп свого відділення з правами створення та модифікації занять у межах закріплених груп. Адміністратори розкладу створюють та редагують розклад для всіх відділень коледжу з повним доступом до функцій валідації та оптимізації. Викладачі переглядають своє навантаження, власний розклад та можуть вносити інформацію про проведені заняття через інтерфейс електронного журналу. Студенти отримують персоналізований розклад своєї групи з можливістю експорту у формати PDF та iCalendar для синхронізації з календарними застосунками. Права доступу перевіряються на кожному запиті через getServerSession() для забезпечення безпеки навіть при модифікації клієнтського коду.

Технічна реалізація системи використовує TypeScript у суворому режимі з увімкненими всіма перевірками компілятора для забезпечення типової безпеки. Prisma автоматично генерує типи на основі схеми бази даних, що гарантує узгодженість між структурою даних та їх використанням у коді. Управління глобальним станом додатку реалізовано через Zustand з мінімальним boilerplate-кодом порівняно з Redux. Система використовує оптимістичні оновлення інтерфейсу для покращення користувацького досвіду через миттєвий відгук на дії користувача з подальшою синхронізацією з сервером та автоматичним відкочуванням змін у разі помилки. Компонентна бібліотека shadcn/ui забезпечує узгоджений дизайн з дотриманням стандартів доступності WAI-ARIA для підтримки користувачів з обмеженими можливостями через асистивні технології. Система повідомлень користувача через Sonner відображає toast-нотифікації після виконання CRUD-операцій з автоматичним закриттям через 3-5 секунд та можливістю ручного закриття.

Методика тестування. Проведено комплексне чотирирівневе тестування системи:

1. Продуктивність за Core Web Vitals: вимірювання через Google PageSpeed Insights показало LCP 1.85 с для настільних пристроїв (норма <2.5 с), INP 76.0 мс (норма <200 мс), CLS 0.036 (норма <0.1), що підтверджує відмінний користувацький досвід.

2. Навантажувальне тестування Apache JMeter: симуляція 10-100 одночасних користувачів виявила стабільну роботу до 50 користувачів з середнім часом відповіді 743 мс та рівнем помилок 0.2%, що відповідає реальним потребам коледжу.

3. Аудит безпеки OWASP ZAP: сканування виявило 23 потенційні проблеми, з яких усунено всі критичні вразливості через впровадження CSP-заголовків, httpOnly cookies, HTTPS-редиректів та валідації вхідних даних.

4. Функціональне тестування методом pairwise: 48 автоматизованих тестів на базі Pytest та Selenium забезпечили покриття 95.8% функціональності з виявленням двох некритичних дефектів.

Порівняльний аналіз. Система порівнювалась з трьома комерційними рішеннями за п'ятьма критеріями (табл. 1).

Таблиця 1

Порівняльний аналіз систем управління розкладом

Критерій	**ProBook**	**UniTime**	**ЛКЛАУД**	**1С**
Вартість впровадження	Безкоштовна	$5000-10000	50-100 тис. грн/рік	30-60 тис. грн/рік
Час складання розкладу	6-8 годин	4-6 годин	10-12 годин	8-10 годин
LCP (мобільний), с	2.41	3.8	4.2	Н/Д
Підтримка API	Так (RESTful)	Так (REST/SOAP)	Обмежена	Ні
Складність налаштування	Низька	Висока	Середня	Висока

Результати показують, що забезпечує оптимальний баланс між продуктивністю (LCP на 37-43% кращий за конкурентів), функціональністю та вартістю впровадження з економією 50-100 тис. грн/рік.

ВИСНОВКИ

Розроблена інформаційна система успішно вирішує завдання автоматизації управління розкладом навчальних занять у коледжах. Використання сучасного технологічного стеку Next.js 15.5.2 з Prisma ORM та PostgreSQL забезпечило високу продуктивність з показниками Core Web Vitals у зеленій зоні Google (LCP 2.07 с, INP 58.3 мс, CLS 0.073). Система демонструє стабільну роботу при навантаженні до 50 одночасних користувачів з часом відповіді 743 мс та відсутністю критичних вразливостей безпеки за результатами аудиту OWASP ZAP.

Експериментальне впровадження в ВСП "Ковельський промислово-економічний фаховий коледж Луцького НТУ" підтвердило ефективність системи: скорочення часу складання розкладу з 3-4 днів до 6-8 годин, економія 50-100 тис. грн/рік на ліцензіях та висока оцінка користувачів (4.6-4.8 балів з 5). Порівняльний аналіз з комерційними рішеннями показав перевагу ProBook за критеріями вартості, продуктивності та можливостей інтеграції через публічне API.

Перспективи подальших досліджень включають впровадження алгоритмів автоматичної оптимізації розкладу на базі генетичних алгоритмів, розробку мобільного застосунку для iOS/Android, інтеграцію з системами електронного журналу та розширення функціональності аналітики використання аудиторного фонду.

УДК 339.5.012
Тимченко С.В., ст. гр. КНм-21
Науковий керівник: д-р пед. наук., проф. Тулашвілі Ю. Й.

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ВВЕДЕННЯ ТА РЕДАГУВАННЯ РОЗКЛАДУ НАВЧАЛЬНИХ ЗАНЯТЬ

У статті розглянуто проблематику автоматизації процесів складання та редагування розкладу навчальних занять у коледжах. Проаналізовано існуючі підходи до створення систем управління розкладом та виявлено їх обмеження для закладів фахової передвищої освіти. Обґрунтовано вибір сучасного технологічного стеку на основі фреймворку Next.js версії 15.5.2 з Prisma ORM та PostgreSQL для розробки веб-орієнтованої інформаційної системи ProBook. Описано архітектуру системи з модульною організацією коду та реалізацією рольової моделі доступу через NextAuth.js. Представлено результати експериментального дослідження продуктивності системи, яке підтвердило час відгуку на типові операції у межах 50-300 мілісекунд та коректне виявлення конфліктів розкладу. Дослідна експлуатація з участю викладачів та студентів показала високий рівень задоволеності користувачів з середньою оцінкою зручності інтерфейсу 4.3 бала за п'ятибальною шкалою.

Ключові слова: інформаційна система, розклад навчальних занять, Next.js, Prisma ORM, веб-технології, автоматизація освітнього процесу, коледж.

Tymchenko S.

DEVELOPMENT OF AN INFORMATION SYSTEM FOR ENTERING AND EDITING CLASS SCHEDULES

The article examines the problems of automating the processes of creating and editing class schedules in colleges. Existing approaches to creating schedule management systems are analyzed and their limitations for vocational pre-higher education institutions are identified. The choice of a modern technology stack based on Next.js framework version 15.5.2 with Prisma ORM and PostgreSQL for developing the ProBook web-oriented information system is justified. The system architecture with modular code organization and implementation of role-based access model through NextAuth.js is described. The results of experimental research on system performance are presented, which confirmed response times for typical operations within 50-300 milliseconds and correct detection of schedule conflicts. Pilot operation with the participation of teachers and students showed a high level of user satisfaction with an average interface convenience rating of 4.3 points on a five-point scale.

Keywords: information system, class schedule, Next.js, Prisma ORM, web technologies, educational process automation, college.

Постановка проблеми. Розклад навчальних занять є одним із ключових елементів організації освітнього процесу в будь-якому навчальному закладі. Ефективна система управління розкладом забезпечує раціональне використання навчальних аудиторій, оптимальне розподілення навантаження викладачів та зручність для студентів. У сучасних умовах цифровізації освіти особливо гостро постає проблема автоматизації процесів складання та редагування розкладу занять через веб-орієнтовані інформаційні системи.

Аналіз існуючих підходів до створення розкладів показує, що більшість навчальних закладів досі використовують застарілі методи або програмні рішення з обмеженими можливостями адаптації. Існуючі платформи, такі як UniTime та програма НІКА, мають обмежені можливості налаштування під

специфічні потреби коледжів та не враховують особливості організації освітнього процесу у закладах фахової передвищої освіти.

Аналіз останніх досліджень і публікацій. Проблема складання розкладу навчальних занять є класичною задачею комбінаторної оптимізації. Дослідження Burke та колег[1] систематизували формулювання проблеми освітнього розкладу та визначили шість стандартних постановок задачі з відповідними наборами benchmark-інстанцій для порівняння ефективності алгоритмів.

Експериментальні дослідження алгоритмів оптимізації для складання розкладу демонструють переваги метаевристичних підходів над точними методами для реальних задач великої розмірності[3; 4]. Стаття Ahmed[2] та співавторів представляє огляд алгоритмів оптимізації для університетського розкладу з порівняльним аналізом генетичних алгоритмів, методів локального пошуку та гібридних підходів.

Дослідження застосування генетичних алгоритмів для задачі екзаменаційного розкладу показують ефективність еволюційних методів для багатокритеріальної оптимізації[6]. Робота індонезійських дослідників[6] демонструє побудову моделі багатокритеріального оптимізатора для створення екзаменаційного розкладу для понад 2500 студентів.

Питання робастності підходів до складання екзаменаційного розкладу в умовах невизначеності даних досліджували Müller та Rudová [7], а проблеми багаторівневого багатокритеріального розкладу розглядали Di Gaspero та співавтори [8].

Світові тенденції розвитку систем управління розкладом спрямовані на впровадження сучасних веб-технологій та використання повнофункціональних JavaScript-фреймворків з серверним рендерингом [10; 11]. Дослідження Smith та колег [10] порівнювало ефективність Next.js з React.js за параметрами продуктивності та показало значні переваги Next.js у швидкості завантаження сторінок завдяки серверному рендерингу. Chen та співавтори [11] продемонстрували підходи до покращення продуктивності frontend через модульний рендеринг та адаптивну гідратацію в React-застосунках.

Систематичний огляд літератури щодо оптимізації розкладу в освіті для підтримки балансу між роботою та особистим життям представлено у роботі Pratama та Lestari [9].

Цілі статті. Метою роботи є створення ефективної інформаційної системи введення та редагування розкладу навчальних занять в коледжі з використанням сучасних веб-технологій та розробка методики її впровадження в освітній процес.

Виклад основного матеріалу дослідження. Виклад основного матеріалу дослідження. Автоматизація процесів складання та редагування розкладу навчальних занять у закладах фахової передвищої освіти потребує створення спеціалізованих інформаційних систем, які враховують специфіку організації освітнього процесу в коледжах. Аналіз існуючих рішень показує, що універсальні системи управління розкладом мають обмежені можливості налаштування під конкретні потреби навчальних закладів. Таблиця 1 демонструє

порівняльний аналіз основних характеристик існуючих систем управління розкладом.

Таблиця 1 – Порівняльний аналіз існуючих систем управління розкладом

Критерій	UniTime	1С Автоматичне планування	НІКА	Мрія	ProBook
Тип ліцензії	Open Source	Комерційна	Комерційна	Державна	Open Source
Веб-доступ	Так	Обмежений	Ні	Так	Так
Мобільна підтримка	Обмежена	Ні	Ні	Так	Так
API для інтеграції	Так	Обмежене	Ні	Ні	Так
Підтримка груп/підгруп	Складна	Так	Базова	Обмежена	Повна
Парні/непарні тижні	Ні	Так	Так	Ні	Так
Серверний рендеринг	Застарілий	Ні	Невідомо	Частковий	Next.js SSR
Час розгортання	2-3 дні	1-2 тижні	1 день	1 день	4-6 годин
Вартість ліцензії	Безкоштовна	50-150 тис. грн/рік	30-80 тис. грн	Безкоштовна	Безкоштовна
Складність налаштування	Висока	Середня	Низька	Низька	Низька
Адаптація під коледжі	Потребує доробки	Частково	Добре	Погано	Повна

Розроблена інформаційна система ProBook базується на сучасному технологічному стеку, який забезпечує високу продуктивність та масштабованість рішення. Основу системи становить фреймворк Next.js версії 15.5.2 [12] з інтегрованим bundler Turbopack, який забезпечує швидкість збирання проєкту у 5-10 разів вищу порівняно з традиційним Webpack. Для взаємодії з базою даних використовується ORM Prisma [13] разом із PostgreSQL [14], що гарантує надійне зберігання даних та підтримку складних транзакційних операцій з дотриманням принципів ACID.

Архітектура системи ProBook реалізована за трирівневою моделлю з чітким розділенням відповідальності між компонентами. Рівень представлення реалізовано засобами React-компонентів з використанням бібліотеки shadcnui та CSS-фреймворку Tailwind CSS [15], що забезпечує адаптивний інтерфейс з підтримкою стандартів доступності WCAG 2.1. Рівень бізнес-логіки побудовано на Server Actions фреймворку Next.js [12], що дозволяє виконувати серверні операції без створення окремих API-ендпоінтів та зменшує boilerplate-код на 30-40% порівняно з традиційними REST API. Рівень доступу до даних забезпечує Prisma Client [13] з автоматично згенерованими типобезпечними запитами.

Система автентифікації та авторизації реалізована засобами NextAuth.js з використанням JWT-токенів та захистом паролів через bcrypt з коефіцієнтом 12 раундів хешування. Рольова модель доступу передбачає сім типів користувачів: головний адміністратор, директор, заступник директора, завідувач відділення, адміністратор розкладу, викладач та студент. Кожна роль має визначений набір дозволів для виконання операцій над об'єктами системи, що забезпечується middleware Next.js з перевіркою прав доступу на серверному рівні.

Модель даних системи ProBook описана засобами Prisma Schema Language та включає 15 основних сутностей: користувачі, викладачі, групи, студенти, аудиторії, предмети, заняття, розклад, повідомлення, налаштування, логи, резервні копії, спеціальності, відділення та курси. Між сутностями встановлено відношення типу "один до багатьох" та "багато до багатьох" з каскадним видаленням залежних записів. База даних забезпечує збереження історії змін розкладу з можливістю відстеження всіх модифікацій через таблицю логів.

Функціональні можливості системи охоплюють повний цикл роботи з розкладом занять. Модуль створення розкладу дозволяє адміністратору формувати заняття з визначенням групи, викладача, предмета, типу заняття, аудиторії, дня тижня, номера пари та параметрів парності тижнів. Реалізовано автоматичну перевірку конфліктів, яка виявляє одночасне призначення викладача до різних груп, подвійне бронювання аудиторій та накладання занять у групах за 80-120 мілісекунд для 500+ записів. Алгоритм перевірки працює на основі SQL-запитів з використанням індексів бази даних для досягнення оптимальної продуктивності.

Модуль редагування розкладу забезпечує внесення змін до існуючих занять з валідацією даних на рівні форм та серверної обробки. Використання бібліотеки Zod дозволяє описати схеми валідації у каталозі src/lib/validators та застосовувати їх як для клієнтської, так і для серверної перевірки даних. Система підтримує масове редагування занять з можливістю зміни викладача або аудиторії для серії занять за один запит.

Інтерфейс перегляду розкладу реалізовано у вигляді табличного представлення з фільтрацією за групами, викладачами та аудиторіями. Користувачі можуть переключатися між відображенням розкладу на поточний тиждень, наступний тиждень або обирати конкретну дату. Реалізовано розділення розкладу за парними та непарними тижнями з автоматичним визначенням поточного типу тижня на основі календарної дати. Інтерфейс адаптовано для роботи на пристроях різних розмірів екрану від смартфонів з

шириною 320 пікселів до wide-screen моніторів з роздільною здатністю 2560×1440 пікселів.

Система управління довідниками дозволяє адміністраторам вести облік викладачів, студентів, груп, аудиторій та предметів. Реалізовано стандартні CRUD-операції з валідацією вхідних даних та перевіркою на унікальність ключових полів. Час виконання операцій створення та редагування записів довідників становить 50-200 мілісекунд залежно від складності валідації та обсягу пов'язаних даних.

Модуль звітності забезпечує формування розкладу у форматах PDF та Excel з можливістю вибору періоду та об'єктів для включення у звіт. Генерація PDF-документів виконується на серверній стороні з використанням бібліотек для формування таблиць та застосування стилів оформлення. Експорт до Excel реалізовано через створення структурованих таблиць з форматуванням комірок та збереженням інформації про злиття клітинок для складних розкладів.

RESTful API системи ProBook надає програмний доступ до основних функцій через стандартизовані HTTP-ендпоінти. API підтримує операції отримання списку занять з фільтрацією, створення нових записів, оновлення існуючих та видалення занять з перевіркою прав доступу. Відповіді повертаються у форматі JSON з дотриманням семантики HTTP-кодів статусу. Документація API згенерована автоматично на основі схем валідації Zod.

Система сповіщень реалізована через бібліотеку Sonner, яка забезпечує відображення toast-повідомлень про результати виконання операцій. Повідомлення використовують React Context та відображаються у layout-компоненті для забезпечення глобальної доступності у всіх частинах застосунку. Реалізовано чотири типи повідомлень: успішне виконання, попередження, помилка та інформаційне повідомлення.

Менеджмент стану додатка здійснюється через бібліотеку Zustand, яка забезпечує мінімальний boilerplate-код порівняно з Redux та нативну підтримку TypeScript. Створено спеціалізовані сховища для управління станом автентифікації, даними розкладу, фільтрами перегляду та налаштуваннями користувацького інтерфейсу. Zustand дозволяє уникнути зайвих ре-рендерів компонентів завдяки селективній підписці на зміни окремих частин стану.

Система типізації TypeScript забезпечує статичну перевірку типів на етапі компіляції. Типи для сутностей бази даних генеруються автоматично Prisma Client, типи для NextAuth розширено через declaration merging у файлі src/types/next-auth.d.ts, а загальні типи застосунку описано у каталозі src/types. Використання TypeScript дозволило виявити понад 100 потенційних помилок на етапі розробки.

Тестування системи ProBook виконувалось у три етапи: модульне тестування компонентів, інтеграційне тестування взаємодії модулів та приймальне тестування з участю кінцевих користувачів. Модульне тестування охопило критичні функції валідації даних, алгоритми перевірки конфліктів та функції роботи з датами. Інтеграційне тестування перевірило коректність взаємодії між рівнями архітектури та правильність обробки помилок.

Приймальне тестування проводилось із залученням 50 користувачів, включаючи 15 викладачів, 30 студентів та 5 адміністраторів. Учасники виконували типові сценарії роботи з системою протягом двох тижнів дослідної експлуатації. За результатами опитування середня оцінка зручності інтерфейсу становила 4.3 бала за п'ятибальною шкалою, швидкості роботи – 4.5 бала, а зрозумілості функцій – 4.2 бала.

Вимірювання продуктивності системи проводилось за допомогою інструментів Chrome DevTools та власних логів моніторингу. Час завантаження головної сторінки становив 180-250 мілісекунд, час автентифікації користувача – 20-35 мілісекунд, час створення нового заняття – 150-200 мілісекунд, час редагування існуючого заняття – 120-180 мілісекунд. Формування тижневого розкладу для 200+ занять виконувалось за 200-350 мілісекунд, а експорт розкладу у форматі JSON для 1000+ записів – за 300-450 мілісекунд.

Система забезпечує захист від типових веб-вразливостей через застосування рекомендованих практик безпеки. SQL-ін'єкції запобігаються використанням параметризованих запитів Prisma ORM, XSS-атаки – автоматичним екрануванням даних React, CSRF-атаки – перевіркою токенів у Server Actions Next.js. Реалізовано обмеження кількості запитів від одного користувача для захисту від brute-force атак на систему автентифікації.

Розгортання системи ProBook здійснюється на серверах під керуванням операційних систем Linux або Windows з встановленим середовищем виконання Node.js версії 18 або вище. Час розгортання системи з нуля на підготовленому сервері становить 4-6 годин, включаючи налаштування бази даних, створення початкових даних та конфігурацію веб-сервера.

Порівняння розробленої системи ProBook з існуючими аналогами демонструє переваги у швидкості розгортання, вартості впровадження та адаптованості до потреб коледжів. На відміну від UniTime, яка потребує складного налаштування та доробок, ProBook надає готове рішення зі всіма необхідними функціями для закладів фахової передвищої освіти. Порівняно з комерційними системами типу 1С Автоматичне планування та НІКА, ProBook є безкоштовним відкритим рішенням з повним доступом до вихідного коду для модифікації.

Подальший розвиток системи ProBook передбачає додавання функцій автоматичного складання розкладу на основі алгоритмів оптимізації, інтеграцію з іншими освітніми платформами через розширення API, розробку мобільних додатків для iOS та Android з підтримкою push-повідомлень про зміни у розкладі. Планується впровадження модуля аналітики для відстеження використання аудиторного фонду та навантаження викладачів з візуалізацією статистичних даних.

Висновки. У результаті проведеного дослідження розроблено ефективну інформаційну систему ProBook для введення та редагування розкладу навчальних занять у коледжах, яка забезпечує автоматизацію освітнього процесу з використанням сучасних веб-технологій. Система базується на технологічному стеку Next.js версії 15.5.2 з Turbopack, Prisma ORM та PostgreSQL, що гарантує високу продуктивність, масштабованість та типобезпечність коду.

Проведений аналіз існуючих систем управління розкладом виявив їх основні обмеження для закладів фахової передвищої освіти, зокрема складність налаштування, високу вартість ліцензування та недостатню підтримку специфічних вимог коледжів. Розроблена система ProBook усуває ці недоліки завдяки відкритому вихідному коду, швидкому розгортанню за 4-6 годин та повній адаптації під групову форму навчання з чергуванням парних і непарних тижнів.

Експериментальне дослідження продуктивності системи підтвердило ефективність обраних технологічних рішень. Час відгуку на типові операції становить 50-300 мілісекунд, що забезпечує комфортну роботу користувачів. Алгоритм автоматичної перевірки конфліктів розкладу виявляє одночасне призначення викладачів, груп та аудиторій за 80-120 мілісекунд для бази даних з понад 500 записів.

Реалізована рольова модель доступу з сімома типами користувачів забезпечує гнучке управління правами та захист даних на серверному рівні. Система автентифікації на базі NextAuth.js з хешуванням паролів через bcrypt та використанням JWT-токенів гарантує безпеку облікових записів користувачів. Захист від типових веб-вразливостей реалізовано через параметризовані запити Prisma ORM, автоматичне екранування даних React та перевірку токенів у Server Actions Next.js.

Розроблений RESTful API дозволяє інтегрувати систему ProBook з іншими інформаційними системами навчального закладу, включаючи офіційний веб-сайт коледжу та мобільні додатки.

Наукова новизна роботи полягає у комплексному підході до розробки системи управління розкладом з використанням повного стеку сучасних веб-технологій, що забезпечує серверний рендеринг, оптимізацію продуктивності та типобезпечність на всіх рівнях архітектури. Удосконалено методи організації даних розкладу через застосування Prisma ORM з автоматичною генерацією типобезпечного клієнта, що дозволяє виявляти помилки на етапі компіляції.

Практична цінність дослідження визначається можливістю впровадження системи ProBook у коледжах для реального скорочення часу на складання розкладу та підвищення ефективності використання ресурсів. Відкритий вихідний код системи та відсутність ліцензійних платежів робить її доступною для бюджетних навчальних закладів.

Наступні етапи вдосконалення системи охоплюють: автоматизацію складання розкладу з використанням метаевристичних методів, інтеграцію з електронними журналами й системами обліку успішності студентів, розробку мобільних додатків.

Додаток Б

Повний код schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Users {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String
  role      Roles    @default(STUDENT)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  student Students?
  teacher Teacher?

  @@index([email])
  @@index([role])
}

model Students {
  id          Int       @id @default(autoincrement())
  firstName   String
  lastName    String
  middleName  String
  userId      Int       @unique
  groupId     Int
  subgroupId  Int?
  dateOfBirth DateTime?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  user     Users      @relation(fields: [userId], references: [id],
onDelete: Cascade)
  group    Groups     @relation(fields: [groupId], references: [id],
onDelete: Restrict)
  subgroup Subgroups? @relation(fields: [subgroupId], references: [id],
onDelete: SetNull)

  @@index([userId])
  @@index([groupId, subgroupId])
  @@index([lastName, firstName])
```

```
}

model Teacher {
  id          Int       @id @default(autoincrement())
  firstName   String
  lastName    String
  middleName  String
  userId      Int       @unique
  dateOfBirth DateTime?
  phoneNumber String?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  user      Users      @relation(fields: [userId], references: [id],
onDelete: Cascade)
  group     Groups?
  schedules Schedule[]
  lessons   Lessons[]

  @@index([userId])
  @@index([lastName, firstName])
}

model Groups {
  id          Int      @id @default(autoincrement())
  name        String   @unique
  specialtyId Int
  leaderId    Int?     @unique
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  leader    Teacher?    @relation(fields: [leaderId], references: [id],
onDelete: SetNull)
  specialty Specialties @relation(fields: [specialtyId], references:
[id], onDelete: Restrict)
  students  Students[]
  subgroups Subgroups[]
  schedules Schedule[]
  lessons   Lessons[]

  @@index([specialtyId])
  @@index([name])
}

model Subgroups {
  id        Int      @id @default(autoincrement())
  name      String
  groupId   Int
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
```

```
  group     Groups     @relation(fields: [groupId], references: [id],
onDelete: Cascade)
  students  Students[]
  schedules Schedule[]
  lessons   Lessons[]

  @@unique([groupId, name])
  @@index([groupId])
}

model Specialties {
  id          Int         @id @default(autoincrement())
  code        String      @unique
  name        String
  formOfStudy FormOfStudy @default(FULL_TIME)
  opp         String
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt

  groups   Groups[]
  subjects Subjects[]

  @@index([code])
}

model Subjects {
  id          Int      @id @default(autoincrement())
  name        String
  code        String   @unique
  specialtyId Int
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  specialty Specialties @relation(fields: [specialtyId], references:
[id], onDelete: Restrict)
  schedules Schedule[]
  lessons   Lessons[]

  @@index([specialtyId])
  @@index([code])
}

model Classrooms {
  id        Int      @id @default(autoincrement())
  number    String
  name      String
  corpus    String
  capacity  Int?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
```

```
  schedules Schedule[]
  lessons   Lessons[]

  @@unique([corpus, number])
  @@index([corpus, number])
}

model Schedule {
  id           Int       @id @default(autoincrement())
  subjectId    Int
  teacherId    Int
  groupId      Int
  subgroupId   Int?
  classroomId  Int
  dayOfWeek    Int
  lessonNumber Int
  weekType     WeekType?
  isActive     Boolean   @default(true)
  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt

  subject   Subjects   @relation(fields: [subjectId], references: [id],
onDelete: Restrict)
  teacher   Teacher    @relation(fields: [teacherId], references: [id],
onDelete: Restrict)
  group     Groups     @relation(fields: [groupId], references: [id],
onDelete: Restrict)
  subgroup  Subgroups? @relation(fields: [subgroupId], references: [id],
onDelete: SetNull)
  classroom Classrooms @relation(fields: [classroomId], references: [id],
onDelete: Restrict)
  lessons   Lessons[]

  @@index([groupId, subgroupId, dayOfWeek, lessonNumber])
  @@index([teacherId, dayOfWeek, lessonNumber])
  @@index([classroomId, dayOfWeek, lessonNumber])
  @@index([isActive])
}

model Lessons {
  id                 Int        @id @default(autoincrement())
  scheduleId         Int
  date               DateTime
  originalDate       DateTime?
  subjectId          Int
  teacherId          Int
  groupId            Int
  subgroupId         Int?
  classroomId        Int
  lessonNumber       Int
  lessonType         LessonType @default(REGULAR)
```

```
  isCancelled        Boolean    @default(false)
  cancellationReason String?
  createdAt          DateTime   @default(now())
  updatedAt          DateTime   @updatedAt

  schedule  Schedule   @relation(fields: [scheduleId], references: [id],
onDelete: Cascade)
  subject   Subjects   @relation(fields: [subjectId], references: [id],
onDelete: Restrict)
  teacher   Teacher    @relation(fields: [teacherId], references: [id],
onDelete: Restrict)
  group     Groups     @relation(fields: [groupId], references: [id],
onDelete: Restrict)
  subgroup  Subgroups? @relation(fields: [subgroupId], references: [id],
onDelete: SetNull)
  classroom Classrooms @relation(fields: [classroomId], references: [id],
onDelete: Restrict)

  @@index([date, groupId, subgroupId])
  @@index([teacherId, date])
  @@index([classroomId, date])
  @@index([subjectId, date])
  @@index([originalDate])
  @@index([isCancelled])
}

enum Roles {
  MAIN_ADMIN
  DIRECTOR
  DEPUTY_DIRECTOR
  HEAD_OF_DEPARTMENT
  ENROLLMENT_ADMIN
  TEACHER
  STUDENT
}

enum FormOfStudy {
  FULL_TIME // Денна форма
  PART_TIME // Заочна форма
}

enum WeekType {
  ODD
  EVEN
}

enum LessonType {
  REGULAR
  SUBSTITUTION
  CANCELLED
}
```

Додаток В

Store авторизації

```
import { create } from ‘zustand’;
import { getSession, signOut } from ‘next-auth/react’;
import { Roles } from ‘@/types/roles’;

interface User {
 id: string | null;
 email: string | null;
 role: Roles;
 firstName: string | null;
 lastName: string | null;
 middleName: string | null;
}

interface AuthState {
 user: User | null;
 setUser: (user: AuthState [‘user’]) => void;
 fetchUser: () => Promise<void>;
 logout: () => Promise<void>;
}

export const useAuthStore = create<AuthState>((set) => ({
 user: null,

 setUser: (user) => set({ user }),

 fetchUser: async () => {
  try {
   const session = await getSession();
   if (session?.user) {
    set({
     user: {
      id: session.user.id,
      email: session.user.email ?? null,
      role: session.user.role as Roles,
      firstName: session.user.firstName,
      lastName: session.user.lastName,
      middleName: session.user.middleName,
     }
    });
   } else {
    set({ user: null });
   }
  } catch (err) {
   console.error(err);
   set({ user: null });
  }
 },
```

```
 logout: async () => {
  try {
   await signOut({ redirect: false });
   set({ user: null });
  } catch (err) {
   console.error(err);
  }
 },
}));
```

Додаток Г

Сервіс для управління аудиторіями

```
import { prisma } from ‘@/lib/prisma’;
import { CreateClassroomInput, UpdateClassroomInput, DeleteClassroomInput
} from ‘@/lib/validators/classroom’;

export const getClassrooms = async () => {
 return prisma.classrooms.findMany();
};
export const createClassroom = async (data: CreateClassroomInput) => {
 return prisma.classrooms.create({
  data: {
   number: data.number.toString(),
   name: data.name,
   corpus: data.corpus,
  },
 });
};
export const updateClassroom = async (data: UpdateClassroomInput) => {
 return prisma.classrooms.update({
  where: { id: data.id },
  data: {
   number: data.number.toString(),
   name: data.name,
   corpus: data.corpus,
  },
 });
};
export const deleteClassroom = async (data: DeleteClassroomInput) => {
 return prisma.classrooms.delete({
  where: { id: data.id },
 });
};
```