МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Вебзастосунок розкладу занять учасників освітнього процесу

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Ілля КРИКАЛОВ**

«__» _____ 2026 р.

Керівник роботи

канд. техн. наук,
доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«_____» _____ 2026 р.

ЗАВДАННЯ
на кваліфікаційну бакалаврську роботу здобувача
Крикалова Іллі

1. Тема кваліфікаційної роботи «Вебзастосунок розкладу занять учасників освітнього процесу» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від « 26 » грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «____» _____ 2026 р.
3. Очікуваним результатом є створений вебзастосунок розкладу занять учасників освітнього процесу, що забезпечує децентралізоване управління розкладом, персоналізований доступ для здобувачів вищої освіти та адміністраторів, а також зручний інтерфейс для перегляду, створення та редагування розкладу з будь-якого пристрою.

4. Перелік питань, що підлягають розробці:

– провести аналіз предметної області та огляд існуючих програмних аналогів і сервісів для управління розкладом занять, визначити їхні переваги та недоліки;

– визначити функціональні та нефункціональні вимоги до системи, розробити сценарії використання для різних суб’єктів (староста, студент, викладач, адміністратор);

– спроєктувати архітектуру системи (обґрунтувати вибір клієнтської та серверної частин, підходи до масштабування, зберігання даних і безпеки);

– спроєктувати логічну модель бази даних у вигляді ER-діаграми та розробити фізичну модель із необхідними таблицями й обмеженнями цілісності;

– розробити структуру бази даних для зберігання інформації про розклад (групи, викладачі, аудиторії, дисципліни);

– реалізувати серверну частину з використанням сучасних технологій (REST API, інтеграція з базою даних з ORM, контейнеризація);

– реалізувати клієнтську частину у вигляді адаптивного та інтуїтивно зрозумілого вебінтерфейсу з інструментами перегляду, фільтрації та редагування розкладу;

– провести тестування розробленої системи та оцінити її продуктивність і відповідність визначеним вимогам.

5. Презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «25» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок розкладу занять учасників освітнього процесу**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	15.12.2025	25.12.2025	*Виконано*
2.	Огляд літератури за темою роботи	26.12.2025	29.01.2026	*Виконано*
3.	Складання календарного плану КБР	02.02.2026	03.02.2026	*Виконано*
4.	Складання вимог: визначення ролей користувачів (студент, викладач, адміністратор) та аналіз існуючих платформ для управління розкладом занять	04.02.2026	11.02.2026	*Виконано*
5.	Розробка проєктних рішень та вибір стеку технологій для клієнтської та серверної частин вебзастосунку	12.02.2026	01.03.2026	*Виконано*
6.	Моделювання та конструювання ПЗ: проєктування бази даних, архітектури REST API та інтерфейсу користувача	02.03.2026	27.03.2026	*Виконано*
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування	30.03.2026	30.04.2026	*Виконано*
8.	Відгук керівника КБР	04.05.2026	06.05.2026	*Виконано*
9.	Оформлення КБР та презентації	07.05.2026	20.05.2026	*Виконано*
10.	Попередній захист	27.05.2026	27.05.2026	*Виконано*
11.	Рецензування	01.05.2026	10.05.2026	*Виконано*
12.	Завершення оформлення КБР та презентації	10.05.2026	11.05.2026	*Виконано*
13.	Захист кваліфікаційної роботи			

Здобувач _____ **Ілля КРИКАЛОВ**

«03» лютого 2026 р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«03» лютого 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок розкладу занять учасників освітнього процесу»

Здобувач 409 гр.: Крикалов Ілля

Керівник: канд. техн. наук, доцент Давиденко Євген

Актуальність роботи полягає у необхідності автоматизації процесів планування та управління розкладом занять, підвищенні зручності доступу до навчальної інформації та зменшенні людського фактору в рутинних задачах організації освітнього процесу.

Метою є проєктування та розробка вебзастосунку розкладу занять для автоматизації формування, редагування та перегляду розкладу учасниками освітнього процесу у закладах освіти.

Об’єктом кваліфікаційної роботи є процес організації, планування та надання інформації про розклад занять учасників освітнього процесу в закладі вищої освіти.

Предметом роботи є засоби розробки клієнт-серверного вебзастосунку для автоматизованого управління розкладом занять.

Кваліфікаційна бакалаврська робота складається зі вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету і проведено огляд поставлених завдань, зазначено предмет та об’єкт роботи, а також розкрито практичне значення створюваного вебзастосунку.

У першому розділі проведено системний аналіз предметної області, описано структурно-функціональні особливості процесу управління розкладом у закладі вищої освіти та виконано огляд і порівняльний аналіз трьох програмних аналогів.

У другому розділі проведено аналіз сучасного стану інструментарію та підходів до розробки вебзастосунків на основі двох наукових публікацій категорії Б, а також сформовано специфікацію вимог до програмного забезпечення що охоплює шість основних функцій системи та нефункціональні вимоги.

У третьому розділі виконано проєктування та моделювання програмного забезпечення: побудовано ER-діаграму бази даних, діаграму варіантів

використання, UML-моделі ключових процесів системи, обґрунтовано вибір технологій розробки та описано інтерфейс користувача.

У четвертому розділі описано програмну реалізацію ключових компонентів системи, проведено модульне тестування серверної частини з охопленням 401 тест-кейсу, розроблено керівництво користувача та описано мобільний застосунок на основі WebView з підтримкою push-сповіщень.

У висновках узагальнено результати виконаної роботи та підтверджено виконання поставлених завдань.

КБР викладена на 68 сторінок, вона містить 4 розділи, 26 ілюстрацій, 2 таблиці, 12 джерел в переліку посилань.

Ключові слова: *вебзастосунок, розклад занять, освітній процес, управління розкладом, REST API, база даних, клієнт-серверна архітектура, React, Django, PostgreSQL.*

ABSTRACT

of the Bachelor's Thesis

"Schedule web application for participants of the educational process"

Student of group 409: Krykalov Illia

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor

Davydenko Yevhen

The relevance of this work lies in the need to automate the processes of planning and managing class schedules, improving the convenience of access to educational information, and reducing the human factor in routine tasks of organizing the educational process.

The objective is to design and develop a web application for class schedules to automate the creation, editing, and viewing of schedules by participants in the educational process at educational institutions.

The subject of this thesis is the process of organizing, planning, and providing information about the class schedule of participants in the educational process at a higher education institution.

The subject of the work is the means of developing a client-server web application for automated class schedule management.

The qualification work consists of an introduction, 4 chapters, conclusions and a list of references.

The introduction substantiates the relevance of the chosen topic, defines the aim and provides an overview of the set tasks, states the subject and object of the work, and reveals the practical significance of the web application being created.

The first chapter presents a systematic analysis of the subject area, describes the structural and functional features of the schedule management process at a higher education institution, and provides a review and comparative analysis of three software analogues.

The second chapter analyzes the current state of tools and approaches to web application development based on two Category B scientific publications, and forms a

software requirements specification covering six main system functions and non-functional requirements.

The third chapter covers the design and modeling of the software: an ER diagram of the database, a use case diagram, UML models of key system processes are constructed, the choice of development technologies is justified, and the user interface is described.

The fourth chapter describes the software implementation of key system components, conducts unit testing of the server-side with coverage of 401 test cases, develops a user guide, and describes the mobile application based on WebView technology with push notification support.

The conclusions summarize the results of the completed work and confirm the fulfillment of the set tasks.

The bachelor's thesis is presented on 68 pages, containing 4 chapters, 26 illustrations, 2 tables, and 12 sources in the list of references.

Keywords: *web application, class schedule, educational process, schedule management, REST API, database, client-server architecture, React, Django, PostgreSQL.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ЗВО – заклад вищої освіти

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ЧНУ – Чорноморський національний університет імені Петра Могили

API – Application Programming Interface

DI – Dependency Injection

ER – Entity-Relationship

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JWT – JSON Web Token

ORM – Object-Relational Mapping

RBAC – Role-Based Access Control

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

UML – Unified Modeling Language

UUID – Universally Unique Identifier

ВСТУП

Актуальність обраної теми полягає в необхідності автоматизації процесів планування та управління розкладом занять в умовах зростаючої кількості учасників освітнього процесу, різноманітності форматів навчання (очного, дистанційного, змішаного) та підвищених вимог до оперативності й доступності навчальної інформації. Існуючі рішення – зокрема, статичні таблиці у форматі Excel, розміщені на хмарних сховищах, або застарілі інформаційні системи університетів – не забезпечують належного рівня зручності, персоналізації та своєчасного оновлення даних. Здобувачі вищої освіти, викладачі та адміністративний персонал змушені витрачати час на пошук актуального розкладу, ручне відстеження змін та узгодження аудиторного фонду, що знижує ефективність організації навчального процесу в цілому.

Метою роботи є проєктування та розробка вебзастосунку для автоматизації формування, редагування та перегляду розкладу занять учасників освітнього процесу, що забезпечить зручний, швидкий і персоналізований доступ до навчальної інформації.

Для досягнення мети необхідно вирішити наступні **завдання**:

– провести аналіз предметної області та огляд існуючих програмних аналогів систем управління розкладом, визначити їхні переваги та недоліки;

– визначити функціональні та нефункціональні вимоги до системи, розробити сценарії використання для різних суб’єктів (студент, викладач, староста, адміністратор);

– спроєктувати логічну модель бази даних у вигляді ER-діаграми та розробити фізичну модель із необхідними таблицями й обмеженнями цілісності;

– спроєктувати архітектуру системи та обґрунтувати вибір технологічного стеку для клієнтської та серверної частин;

– реалізувати серверну частину вебзастосунку з використанням REST API та забезпечити інтеграцію з базою даних через ORM;

– реалізувати клієнтську частину у вигляді адаптивного та інтуїтивно зрозумілого вебінтерфейсу з функціями перегляду, фільтрації та редагування розкладу;

– провести тестування розробленої системи та оцінити її продуктивність і відповідність визначеним вимогам.

Об’єктом кваліфікаційної роботи є процес організації, планування та надання інформації про розклад занять учасників освітнього процесу в закладі вищої освіти.

Предметом роботи є засоби розробки клієнт-серверного вебзастосунку для автоматизованого управління розкладом занять.

Практичне значення полягає у створенні вебзастосунку, який забезпечить децентралізоване управління розкладом занять з розмежуванням прав між старостами, менеджерами та адміністраторами, та надасть зручний, швидкий і персоналізований доступ до актуальної навчальної інформації з будь-якого пристрою для всіх учасників освітнього процесу. Розроблена система дозволить автоматизувати формування та редагування розкладу, скоротити час на його пошук і узгодження, а також підвищить ефективність роботи здобувачів вищої овіти, адміністративного персоналу і викладачів закладу вищої освіти.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1 Розкриття об’єкта і предмета дослідження

Сучасний заклад вищої освіти є складною організаційною структурою, в якій навчальний процес одночасно охоплює тисячі здобувачів вищої освіти, сотні викладачів та сотні навчальних дисциплін. Ефективна організація такого процесу неможлива без чіткої системи планування й координації навчальних занять у часі та просторі. Розклад занять виступає інтегруючим інструментом, що поєднує всі ключові ресурси освітнього процесу – учасників, навчально-методичне забезпечення та аудиторний фонд – у єдину функціональну систему.

Об’єктом дослідження кваліфікаційної роботи є процес організації, планування та надання інформації про розклад занять учасників освітнього процесу у закладі вищої освіти. Цей процес охоплює формування розкладу адміністративним персоналом, його зберігання у структурованому вигляді, оперативне внесення змін у відповідь на зміни навантаження викладачів або реструктуризацію груп, а також надання учасникам освітнього процесу актуальної інформації про заняття.

Предметом дослідження є засоби розробки клієнт-серверного вебзастосунку для автоматизованого управління розкладом занять, що включають архітектурні підходи до побудови SPA-застосунків, технології організації серверної логіки на базі REST API, методи проєктування реляційних баз даних для зберігання розкладу та підходи до забезпечення розмежування прав доступу залежно від ролі користувача.

Доцільність розробки спеціалізованого вебзастосунку обумовлена низкою чинників, які обмежують ефективність наявних рішень в українських ЗВО. Найпоширеніший підхід до публікації розкладу – розміщення статичних таблиць у форматі Excel на хмарних сховищах на кшталт Google Drive – вимагає від користувача послідовно завантажувати файл, шукати свій факультет, групу й конкретний день, що суперечить принципам зручності взаємодії з інформацією на

мобільних пристроях. Окрім того, такий підхід унеможливлює оперативне інформування про зміни у розкладі.

1.2 Опис структурно-функціональних особливостей об’єкта дослідження

Процес управління розкладом занять у ЗВО має багаторівневу структуру, що відображає організаційну ієрархію навчального закладу та різноманіття ролей учасників освітнього процесу. Аналіз структурно-функціональних особливостей цього процесу дозволяє визначити вимоги до програмного забезпечення, що автоматизує його.

Учасники процесу становлять чотири основні категорії, кожна з яких має власні інформаційні потреби та повноваження. Здобувачі вищої освіти потребують швидкого доступу до актуального розкладу своєї групи з урахуванням підгрупи (А або Б)та типу навчального тижня (над рискою або під рискою). Викладачі мають отримувати персоналізований розклад власних занять із зазначенням груп, аудиторій та часових слотів. Старости академічних груп виконують посередницьку роль між адміністрацією та групою, потребуючи можливості оперативно вносити зміни до розкладу власної групи. Адміністративний персонал та профільні менеджери (менеджери клієнтів, факультетів, викладачів, дисциплін, аудиторій, груп та розкладу) забезпечують повне, або часткове керування системою.

Інформаційні сутності розкладу включають взаємопов’язаний набір об’єктів. Кожне заняття характеризується **дисципліною**, що викладається; **викладачем**, який її проводить; **аудиторією**, в якій відбувається заняття; **часовим слотом**, що визначається днем тижня (понеділок – п’ятниця), порядковим номером пари (1 – 6) та типом тижня (над рискою / під рискою); **типом заняття** (лекція / практика). Окремі категорії об’єктів – **групи, факультети, користувачі** – формують довідкову основу системи, з якою пов’язані заняття через зовнішні ключі.

Часова структура навчального процесу у ЗВО має циклічний характер із двотижневою періодичністю. У ЧНУ імені Петра Могили навчальний тиждень поділяється на тиждень над рискою та тиждень під рискою, що дозволяє рознести різні навчальні дисципліни у часі за умов обмеженого аудиторного фонду. Деякі

академічні групи додатково діляться на підгрупи (А та Б), що проводять окремі заняття у різних аудиторіях паралельно. Ця особливість унеможливлює використання спрощених моделей розкладу, що передбачають єдине заняття на групу у певному часовому слоті.

Функціональні процеси управління розкладом охоплюють як операції створення та модифікації розкладу адміністративним персоналом, так і операції доступу до інформації з боку студентів і викладачів. Процеси модифікації потребують суворого контролю прав доступу, оскільки внесення помилкових даних може призвести до зриву занять. Процеси доступу до інформації, навпаки, мають бути максимально швидкими та зручними і не повинні вимагати авторизації для базового перегляду розкладу.

Технологічні особливості процесу полягають у необхідності забезпечення одночасного доступу великої кількості користувачів до системи у пікові періоди (початок навчального семестру, періоди іспитів), у вимозі адаптивності інтерфейсу до різних типів пристроїв (ноутбук\комп’ютер, планшет, смартфон), а також у необхідності оперативного відображення змін розкладу всім зацікавленим сторонам без затримок.

Проблеми поточного підходу до публікації розкладу у ЧНУ імені Петра Могили. На сьогоднішній день у Чорноморському національному університеті імені Петра Могили актуальний розклад занять формується адміністративним персоналом у вигляді Excel-файлів, які розміщуються у спільному хмарному сховищі Google Drive. Такий підхід має низку суттєвих недоліків, що знижують ефективність роботи з розкладом для всіх категорій учасників освітнього процесу.

По-перше, для отримання актуальної інформації про власні заняття користувачеві необхідно послідовно виконати декілька кроків: відкрити Google Drive у браузері, знайти потрібну директорію з файлами розкладу, завантажити або відкрити Excel-файл відповідних курсів, груп або форми навчання (денної чи заочної), у разі використання смартфону – відкрити його у відповідному програмному забезпеченні, перейти на аркуш свого курсу, знайти власну групу та зорієнтуватися у двовимірній таблиці з розкладом, додатково враховуючи тип

тижня (парний / непарний) та власну підгрупу (А або Б). Цей процес займає значно більше часу, ніж отримання інформації через спеціалізований вебзастосунок із пошуком за номером групи, та особливо ускладнюється на мобільних пристроях, де таблиці Excel часто відображаються некоректно через велику ширину розкладу та малий розмір екрана.

По-друге, Excel-файли є статичним форматом подачі інформації, який не передбачає можливостей фільтрації за типом тижня (над рискою / під рискою) або підгрупою. Користувач вимушений візуально аналізувати весь розклад, відокремлюючи актуальні для нього заняття від нерелевантних, що збільшує ймовірність помилки.

По-третє, відсутній механізм оперативного інформування користувачів про внесення змін до розкладу. Якщо адміністративний персонал оновлює файл у Google Drive, користувачі дізнаються про це лише за умови повторного завантаження/перегляду файлу. Це призводить до ситуацій, коли студенти приходять на заняття за застарілим розкладом, орієнтуючись на минулу копію файлу.

По-четверте, Excel-формат не підтримує побудови розкладу за викладачем – інформація структурована виключно за групами. Викладачам, які прагнуть переглянути власне навантаження за тиждень, доводиться вручну переглядати розклади всіх груп, у яких вони викладають, і складати загальну картину самостійно.

Зазначені недоліки демонструють обмеженість поточного підходу та обґрунтовують доцільність розробки спеціалізованого вебзастосунку, орієнтованого на швидкий доступ до персоналізованої інформації з підтримкою фільтрації за різними критеріями.

1.3 Огляд та аналіз сучасного стану програмних рішень у предметній області

З метою виявлення кращих практик і слабких місць наявних рішень було проведено аналіз трьох програмних систем управління розкладом, що наявні в українських ЗВО: Розклад КПІ, АСУ ПДАУ Розклад та Розклад МНАУ. Результати такого аналізу дозволяють визначити функціональні можливості, які доцільно перейняти при розробці власної системи, а також виявити обмеження, які потрібно усунути у новому рішенні. Системи аналізувалися за такими критеріями: архітектура, мова реалізації, основні функціональні можливості, переваги та недоліки.

1.3.1 Аналіз системи Розклад КПІ

Система Розклад КПІ [1], розроблена Конструкторським бюро інформаційних систем НТУУ «КПІ ім. Ігоря Сікорського», є однією з найбільш сучасних та функціонально розвинених систем розкладу серед українських ЗВО. Система реалізована на базі Node.js та TypeScript у клієнт-серверній архітектурі.

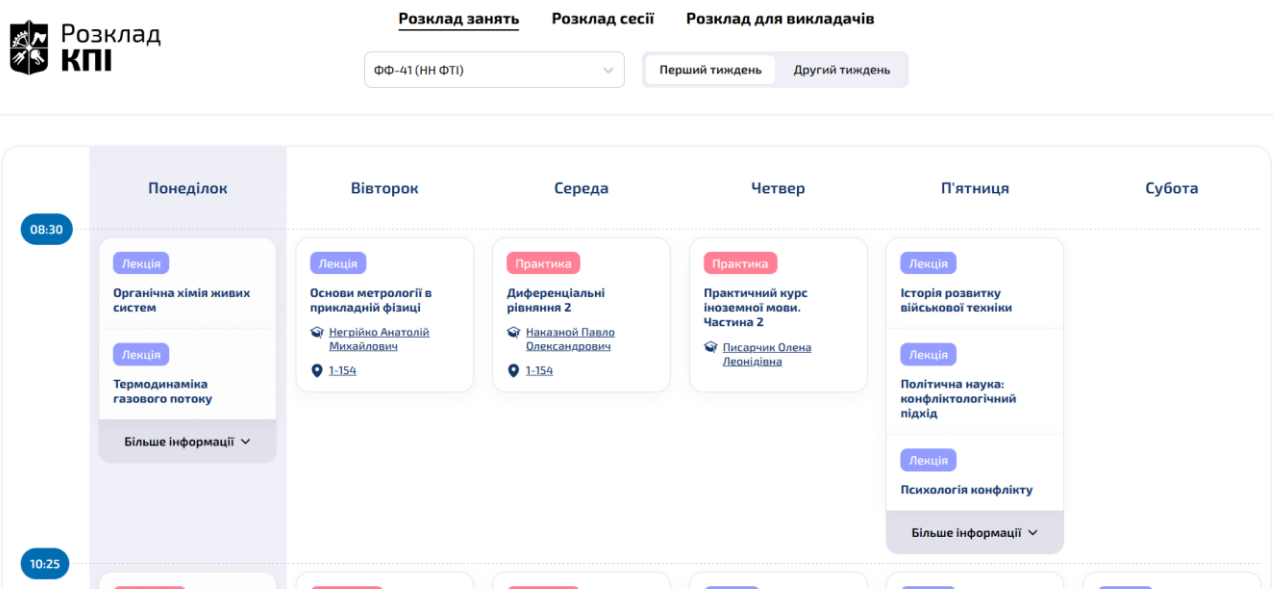


Рисунок 1.1 – Вебзастосунок Розклад КПІ

Основними функціональними можливостями Розклад КПІ є перегляд розкладу за групами, перегляд розкладу за викладачами, відображення розкладу

екзаменаційної сесії, підтримка двотижневого циклу занять (тиждень над рискою / під рискою) та відображення розкладу спеціальних занять.

До переваг системи слід віднести наявність окремого Telegram-бота, що дозволяє отримувати інформацію про розклад через месенджер; індикацію часу останнього оновлення даних; коректну адаптацію інтерфейсу під мобільні пристрої; доступність розкладу екзаменаційної сесії; регулярні оновлення функціоналу; сучасний та інтуїтивно зрозумілий інтерфейс.

Серед недоліків Розклад КПІ можна виділити надмірний обсяг таблиці з розкладом, що вимагає від користувача гортати сторінку для перегляду повної інформації, та відсутність офлайн-режиму, що робить систему повністю залежною від наявності інтернет-з'єднання.

1.3.2 Аналіз системи Розклад АСУ ПДАУ

Система Розклад АСУ ПДАУ [2] є частиною автоматизованої системи управління Полтавського державного аграрного університету. Реалізована з використанням ASP.NET (C#) та JavaScript у клієнт-серверній архітектурі.

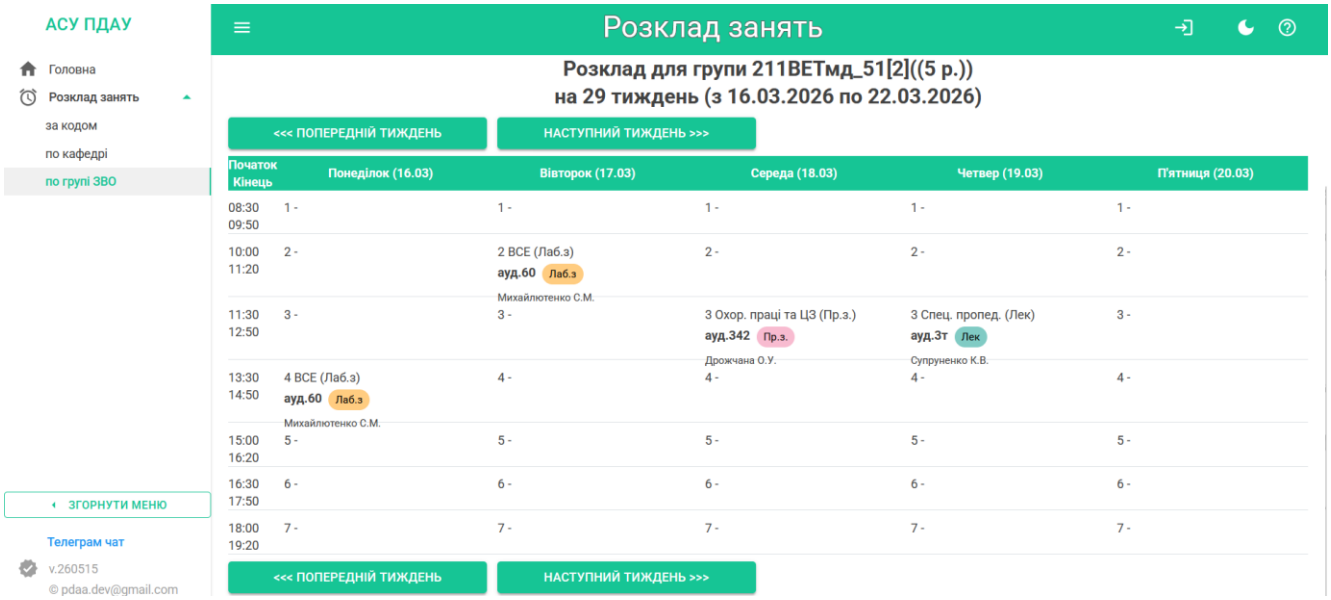


Рисунок 1.2 – Вебзастосунок Розклад АСУ ПДАУ

Функціональні можливості системи включають пошук розкладу за кодом дисципліни, перегляд розкладу за кафедрою, перегляд розкладу за групою, відображення розкладу на весь навчальний рік та пошук розкладу за викладачем.

Перевагами АСУ ПДАУ є можливість завантаження розкладу у вигляді Excel-файлу, що корисно для архівного зберігання та офлайн-роботи; можливість перегляду архівного розкладу (рік тому).

Недоліками системи є відсутність індикації часу останнього оновлення розкладу, що знижує довіру користувачів до актуальності даних; необхідність постійного інтернет-з'єднання; тривала відсутність функціональних оновлень системи; менш зручний та сучасний інтерфейс порівняно з аналогами.

1.3.3 Аналіз системи Розклад МНАУ

Система Розклад МНАУ [3], яка розроблена Віталієм Ендресом для Миколаївського національного аграрного університету, реалізована з використанням TypeScript та Vue.js у клієнт-серверній архітектурі.

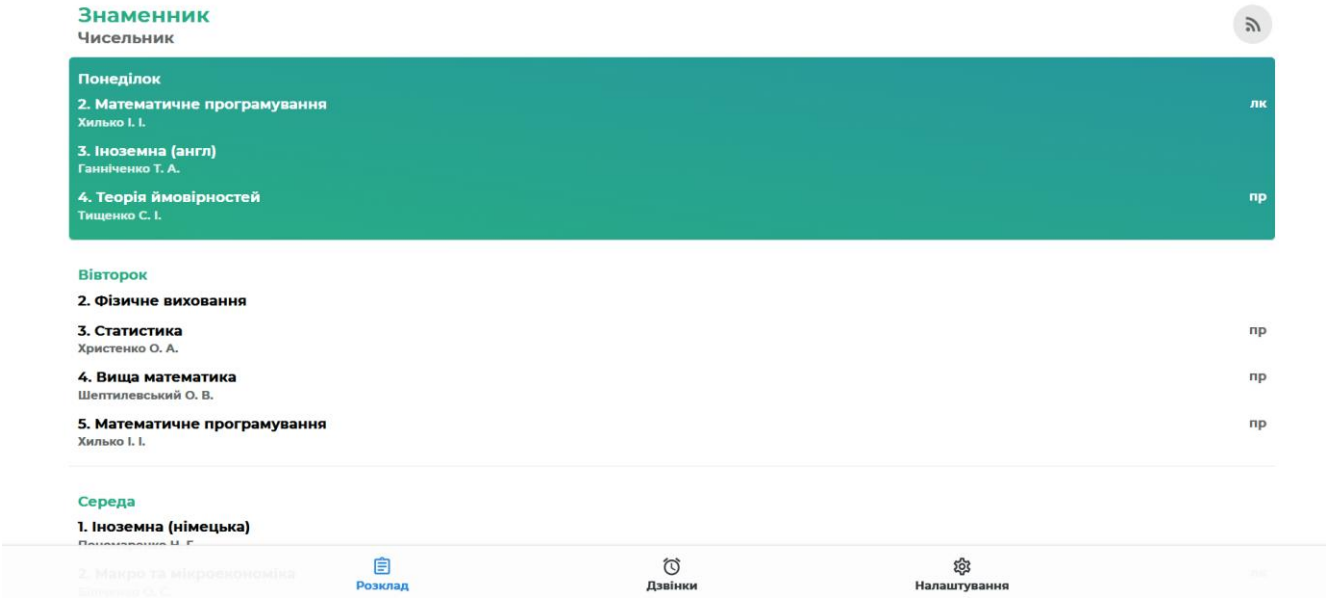


Рисунок 1.3 – Вебзастосунок Розклад МНАУ

Основні функції системи – перегляд розкладу за освітнім напрямом, перегляд розкладу за курсом, перегляд розкладу за викладачем, наявність окремого мобільного застосунку та відображення розкладу дзвінків.

Серед переваг – доступність розкладу дзвінків як окремого функціоналу; коректна адаптація під мобільні пристрої; наявність окремого мобільного застосунку; індикація відліку часу до завершення поточної пари; сучасний інтерфейс.

Недоліками є не до кінця інтуїтивна навігація; неповний перелік груп та викладачів у системі (відображаються не всі наявні в університеті групи та викладачі); необхідність постійного інтернет-з’єднання.

1.3.4 Порівняльний аналіз досліджених систем

Зведені результати аналізу функціональних можливостей розглянутих систем подано в таблиці 1.1.

Таблиця 1.1 – Порівняння характеристик аналогічних систем

Критерій / Система	Розклад КПІ	Розклад АСУ ПДАУ	Розклад МНАУ
Розклад за групою	+	+	+
Розклад за викладачем	+	+	+
Розклад за аудиторією	-	-	-
Розклад сесій	+	-	-
Адаптація під мобільні пристрої	+	-	+
Мобільний застосунок	-	-	+
Telegram-бот	+	+	-
Індикація часу оновлення	+	-	-
Розклад дзвінків	-	-	+
Динамічне формування розкладу	+	+	+
Розмежування прав доступу	-	-	-
Самостійне редагування старостою	-	-	-

Узагальнення результатів аналізу дозволяє зробити **висновок**, що всі три розглянуті системи реалізують подібну за концепцією модель – пасивне відображення розкладу, який формується та оновлюється централізовано на стороні університетських інформаційних систем. Студенти та викладачі виступають у ролі споживачів готової інформації без можливості безпосередньо взаємодіяти з даними розкладу. Така модель є виправданою для більшості ЗВО, оскільки забезпечує цілісність та контрольованість даних, проте вона має суттєві обмеження.

Основним обмеженням наявних систем є відсутність механізмів децентралізованого внесення оперативних змін до розкладу авторизованими учасниками освітнього процесу. У реальній практиці навчальної діяльності часто виникають ситуації, що потребують швидкої реакції – заміна викладача, перенесення заняття до іншої аудиторії, додавання консультації. У наявних системах такі зміни доходять до користувачів із затримкою, оскільки потребують проходження повного адміністративного циклу: повідомлення відповідального деканату, внесення змін до центральної інформаційної системи університету, очікування наступного циклу синхронізації даних. Цей процес може займати від кількох годин до кількох днів.

Другим обмеженням є відсутність у розглянутих системах функціоналу самостійного редагування розкладу старостами академічних груп. Староста є представником групи, який оперативно інформований про реальні зміни у навчальному процесі та має можливість швидко відобразити їх у розкладі за умови наявності відповідного інструменту. Жодна з трьох досліджуваних систем такого інструменту не пропонує.

Третім обмеженням є невелика різноманітність форматів подачі інформації для кінцевих користувачів. Серед трьох розглянутих систем мобільний застосунок наявний лише в одній (Розклад МНАУ), а інтеграція з Telegram-ботом – у двох (Розклад КПІ та Розклад АСУ ПДАУ). Це обмежує канали отримання інформації для користувачів із різними сценаріями використання.

Водночас аналіз дозволив виділити **успішні рішення**, які доцільно врахувати при проєктуванні розроблюваної системи: формування розкладу як для груп, так і

для викладачів (характерно для всіх трьох систем); адаптивний інтерфейс для мобільних пристроїв (Розклад КПІ, Розклад МНАУ); наявність окремого мобільного застосунку (Розклад МНАУ); інтеграція з Telegram-ботом для оперативного інформування (Розклад КПІ, Розклад АСУ ПДАУ); індикація часу останнього оновлення (Розклад КПІ); відображення розкладу дзвінків (Розклад МНАУ).

Окремої уваги заслуговує наявність мобільного застосунку як форми подачі інформації про розклад. Серед розглянутих систем такий застосунок реалізовано лише у Розкладі МНАУ. Мобільний застосунок надає користувачам ряд переваг, що недосяжні у вебверсії систем-аналогів.

Перевагою мобільного застосунку є можливість часткового або повного функціонування в офлайн-режимі завдяки локальному кешуванню даних розкладу на пристрої користувача. Користувач може отримати інформацію про власні заняття навіть за відсутності інтернет-з’єднання – у транспорті, у віддалених від точок Wi-Fi приміщеннях університету або в умовах нестабільного зв’язку. Ця особливість суттєво підвищує надійність доступу до важливої інформації.

Слід зазначити, що дві з трьох розглянутих систем (Розклад КПІ та Розклад АСУ ПДАУ) частково компенсують відсутність власного мобільного застосунку інтеграцією з месенджером Telegram – через спеціалізованих ботів. Такий підхід дає користувачам можливість отримувати інформацію через звичний канал комунікації без необхідності встановлення додаткового програмного забезпечення, проте обмежений за функціональними можливостями порівняно з повноцінним мобільним застосунком.

У контексті розроблюваної системи передбачається реалізація мобільного застосунку, побудованого за принципом WebView – тобто такого, що використовує адаптовану вебверсію застосунку як основу інтерфейсу. Такий підхід дозволяє забезпечити користувачів мобільним застосунком з нативною інтеграцією у систему пристрою (іконка на робочому екрані, повноекранний режим без елементів браузера, push-сповіщення про зміни у розкладі та нагадування про початок занять, доступ до інших системних можливостей) без необхідності окремої розробки

нативного мобільного інтерфейсу. Це обумовлює потребу у повноцінній адаптації вебінтерфейсу до мобільних пристроїв із застосуванням принципів responsive design, що забезпечить однакову якість користувацького досвіду як у звичайному браузері, так і в межах WebView мобільного застосунку. Push-сповіщення, своєю чергою, вирішують одне з ключових обмежень систем-аналогів, виявлених у ході аналізу – відсутність механізму оперативного інформування користувачів про зміни в розкладі.

На основі результатів аналізу **визначено ключові вимоги до розроблюваної системи**, що мають вигідно виділити її серед наявних рішень. Перш за все, на відміну від проаналізованих систем, що реалізують виключно перегляд розкладу, розроблюваний вебзастосунок передбачає наявність функціоналу редагування розкладу безпосередньо у системі. Це обумовлює потребу в багаторівневій моделі розмежування прав доступу, яка включатиме гостьовий перегляд для незареєстрованих користувачів, обмежені права редагування для старост у межах їхніх власних груп, диференційовані права для семи категорій менеджерів (клієнтів, факультетів, викладачів, дисциплін, аудиторій, груп та розкладу) та повний доступ для адміністратора. Крім того, система передбачатиме підтримку ділення груп на підгрупи (А / Б) із можливістю задавати окремі заняття для кожної підгрупи.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено системний аналіз предметної області автоматизації управління розкладом занять у закладі вищої освіти та визначено теоретико-методологічну основу для подальшого проєктування програмного забезпечення.

Розкрито об’єкт і предмет дослідження. Доведено доцільність розробки спеціалізованого програмного рішення з огляду на обмеженість поточного підходу до публікації розкладу у ЧНУ імені Петра Могили.

Описано структурно-функціональні особливості об’єкта дослідження: визначено чотири категорії учасників процесу (студенти, викладачі, старости, адміністративний персонал та менеджери), охарактеризовано основні інформаційні

сутності розкладу (заняття, дисципліни, викладачі, аудиторії, часові слоти, групи), розкрито часову структуру навчального процесу з двотижневою періодичністю та можливим діленням груп на підгрупи. Визначено технологічні особливості процесу, що формують вимоги до архітектури та продуктивності системи.

Проведено огляд та аналіз трьох програмних рішень управління розкладом, що використовуються в українських ЗВО: Розклад КПІ, Розклад АСУ ПДАУ та Розклад МНАУ.

На основі результатів аналізу обґрунтовано позиціонування розроблюваного вебзастосунку як системи нової парадигми – з децентралізованим оперативним внесенням змін авторизованими користувачами та багаторівневою моделлю розмежування прав доступу, що включатиме дев’ять ролей (гість, староста, адміністратор та сім профільних менеджерів).

Результатом проведеної у розділі роботи є обґрунтування актуальності розробки та визначення вихідних вимог до програмного забезпечення, що становить основу для подальшого моделювання предметної області та формування специфікації вимог у наступному розділі.

2 АНАЛІЗ ПІДХОДІВ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБЗАСТОСУНКУ

2.1 Аналіз сучасного стану інструментарію, моделей та методів розробки вебзастосунків

Розробка сучасного клієнт-серверного вебзастосунку для управління розкладом занять потребує обґрунтованого вибору архітектурних рішень, програмних інструментів та методів реалізації, що відповідають актуальним тенденціям розвитку галузі веброзробки. У цьому підрозділі проведено аналіз сучасного стану інструментарію та архітектурних підходів на основі двох наукових публікацій, розміщених у фахових виданнях України категорії Б, та виконано зіставлення виявлених тенденцій галузі з технічним вибором, прийнятим для реалізації розроблюваної системи.

2.1.1 Аналіз публікації щодо архітектурних підходів до розробки масштабованих вебзастосунків

У статті Скляренко О., Савченка Я., Литвиненка Л. та Сушинського О. «Архітектурні підходи до розробки масштабованих вебзастосунків» [4], опублікованій у фаховому виданні «Кібербезпека: освіта, наука, техніка» (категорія Б) у 2024 році, досліджено сучасні методи та технології створення масштабованих вебзастосунків, потреба в яких постійно зростає через збільшення обсягів даних та кількості користувачів.

Автори визначають масштабованість як здатність системи ефективно обробляти зростаюче навантаження за рахунок додавання обчислювальних ресурсів без шкоди продуктивності. Зазначається, що недотримання принципів масштабованості на ранніх етапах проєктування призводить до уповільнення роботи системи, збоїв та втрати користувачів у міру зростання навантаження. Це робить проєктування масштабованої архітектури пріоритетним завданням інженерів програмного забезпечення вже на початковому етапі розробки.

У публікації проаналізовано та порівняно ключові архітектурні підходи, що дозволяють забезпечити масштабованість сучасних вебзастосунків:

– мікросервісна архітектура як підхід до проєктування системи у вигляді набору слабко зв’язаних незалежно розгортуваних сервісів;

– використання хмарних сервісів для динамічного виділення ресурсів відповідно до поточного навантаження;

– кешування часто запитуваних даних з метою скорочення часу відповіді системи та зменшення навантаження на основну базу даних;

– балансування навантаження для рівномірного розподілу запитів між декількома екземплярами серверних застосунків;

– асинхронні черги повідомлень для обробки тривалих операцій у фоновому режимі без блокування основного потоку обслуговування користувачів.

Автори ілюструють ефективність зазначених підходів на прикладах масштабних промислових систем – Netflix, Spotify, Facebook, Amazon та LinkedIn – та наголошують на важливості комплексного, а не ізольованого застосування архітектурних рішень. Ключовим висновком публікації є те, що для систем середнього масштабу, які працюють з обмеженою аудиторією, повний перехід на мікросервісну архітектуру не є обов’язковим і може призвести до надмірного ускладнення системи. Водночас впровадження кешування, балансування навантаження та моніторингу продуктивності автори вважають критично важливим незалежно від обраної архітектурної моделі.

Зіставлення виявлених у публікації тенденцій з архітектурними рішеннями розроблюваного вебзастосунку розкладу занять демонструє їхню відповідність сучасному стану галузі. Розроблюваний застосунок реалізує клієнт-серверну архітектуру з чітким розмежуванням SPA-клієнта (React + TypeScript) та REST API-сервера (Django + Django-Ninja), що відповідає принципу слабкого зв’язування компонентів. Серверна частина має модульну структуру з виділенням окремих доменів та чотирирівневою архітектурою Handler, UseCase, Service, ORM всередині кожного домену, що забезпечує незалежність компонентів і спрощує подальше масштабування. Кешування реалізовано за допомогою Redis, через бібліотеку django-redis, що дозволяє скорочувати час відповіді для повторних запитів до часто запитуваних даних розкладу, а при зміні даних кеш автоматично

інвалідується. Балансування навантаження та проксіювання запитів забезпечує Nginx, який розподіляє трафік між екземплярами застосунку та обслуговує статичні файли. Контейнеризація на базі Docker та Docker Compose уніфікує середовища розробки і розгортання, дозволяючи горизонтально масштабувати сервіс шляхом збільшення кількості контейнерів без змін у структурі бази даних. Вибір Django-Ninja замість повноцінної мікросервісної архітектури обґрунтовується масштабом системи – застосунок розраховано на обслуговування одного університету, що відповідає рекомендаціям публікації щодо уникнення надмірного ускладнення архітектури для систем середнього рівня.

2.1.2 Аналіз публікації щодо багаторівневого захисту даних у вебзастосунках

У статті Белоуса Р. та Клименкова О. «Багаторівневий захист даних у Laravel-додатках» [5], опублікованій у фаховому виданні «Кібербезпека: освіта, наука, техніка» (категорія Б) у 2026 році, досліджено проблему побудови ефективної моделі захисту даних у сучасних вебзастосунках в умовах зростаючої кількості кіберзагроз.

Автори систематизують вбудовані механізми захисту фреймворку Laravel та виявляють їхні обмеження у високонавантажених середовищах. Базова конфігурація захисту, що включає ORM із параметризованими запитами, екранування шаблонів, CSRF-токени та авторизацію через middleware, нейтралізує близько двох третин загроз, однак є недостатньою для протидії brute force та DoS-атакам.

На основі проведених експериментальних досліджень із використанням Apache Benchmark та Siege автори порівняли два алгоритми хешування паролів. Алгоритм bcrypt забезпечує середній час відповіді близько 72 мс, тоді як Argon2id – 95 мс при навантаженні в 1000 паралельних запитів. Для захисту API авторами порівнювались механізми Sanctum (82 мс, ~12 000 запитів/хв) та Passport на базі OAuth 2.0 (108 мс, ~9 100 запитів/хв). Застосування rate limiting із параметром 60

запитів за хвилину забезпечило блокування 89% спроб brute force-атак без суттєвих накладних витрат.

Ключовим висновком публікації є доцільність комбінованого підходу до захисту: поєднання стійкого алгоритму хешування паролів, механізму токенізованої автентифікації та rate limiting дозволяє досягти балансу між рівнем безпеки та продуктивністю системи.

Зіставлення результатів публікації з технічними рішеннями розроблюваного вебзастосунку розкладу занять свідчить про відповідність реалізованих підходів актуальному стану галузі. Хоча розроблюваний застосунок побудовано на стеку Python/Django замість PHP/Laravel, фундаментальні принципи захисту є незалежними від конкретного фреймворку. Для хешування паролів у системі застосовується алгоритм bcrypt із використанням бібліотеки bcrypt, що відповідає базовому рівню захисту, описаному у публікації. Захист від SQL-ін’єкцій забезпечується через ORM Django з параметризованими запитами – аналогічно до підходу на основі Eloquent ORM у Laravel. Захист від XSS-атак реалізується автоматичним екрануванням у React на клієнтській частині. Для захисту від CSRF Django використовує механізм CSRF-токенів, причому клієнтська частина передає токен у заголовку X-CSRFToken при кожному запиті, що змінює стан. Rate limiting реалізовано на двох рівнях: на рівні Django-Ninja (10 запитів/с для анонімних користувачів, 50 запитів/с для автентифікованих) та на рівні Nginx (окрема auth-зона з обмеженням 5 запитів/хв для ендпоінтів автентифікації). Автентифікацію реалізовано через JWT-токени з алгоритмом підпису HS256 за допомогою бібліотеки PyJWT – підхід, що за функціональними характеристиками відповідає Sanctum у Laravel і є оптимальним для SPA-клієнтів. Усі виданий JWT-токени зберігаються в таблиці IssuedJwtToken для підтримки механізму відкликання, що усуває відомий недолік stateless-токенів і не розглядається у публікації. Усі з’єднання між клієнтом та сервером захищені протоколом HTTPS з TLS-сертифікатами Let’s Encrypt, а заголовки безпеки (X-Frame-Options: DENY, X-Content-Type-Options: nosniff, Referrer-Policy: same-origin) налаштовано на рівні Nginx.

2.2 Специфікація вимог до програмного забезпечення

2.3.1 Призначення та межі проєкту

Призначення системи. Вебзастосунок призначений для автоматизації процесу організації, планування та надання інформації про розклад занять учасникам освітнього процесу у Чорноморському національному університеті імені Петра Могили. Система реалізує децентралізовану модель формування розкладу, диференційований доступ для різних категорій користувачів та оперативне відображення змін розкладу.

Погодження, ухвалені у програмній документації. Специфікацію розроблено на основі аналізу предметної області, проведеного у розділі 1, та результатів дослідження сучасних архітектурних підходів до розробки масштабованих вебзастосунків, представлених у підрозділі 2.1. Ухвалено використання сучасних вебтехнологій та клієнт-серверної архітектури з SPA на клієнтській стороні. Визначено перелік основних функцій, що мають бути реалізовані у межах кваліфікаційної бакалаврської роботи.

Межі проєкту. Проєкт охоплює розробку клієнтської частини вебзастосунку (SPA на React), серверної інфраструктури для обробки запитів (REST API на Django-Ninja) та проєктування реляційної бази даних для зберігання даних розкладу (PostgreSQL). Клієнтська частина реалізується з повною підтримкою адаптивного дизайну для мобільних пристроїв, що забезпечує коректне відображення та зручну взаємодію на екранах будь-якого розміру і є основою для мобільного застосунку на базі WebView. Проєкт не охоплює розробку нативних мобільних застосунків для iOS та Android, інтеграцію із зовнішніми системами управління навчальним процесом ЗВО та реалізацію Telegram-бота.

2.3.2 Загальний опис

Сфера застосування. Вебзастосунок використовується учасниками освітнього процесу у закладі вищої освіти Чорноморському національному університеті імені Петра Могили. Цільовою аудиторією є здобувачі вищої освіти, викладачі, старости академічних груп та адміністративний персонал.

Характеристики користувачів.

– Гість – незареєстрований відвідувач, що має базові навички роботи з вебзастосунками та доступ до Інтернету з персонального комп’ютера, ноутбука, планшета або смартфона.

– Староста – авторизований студент із розширеними правами на редагування розкладу власної групи.

– Менеджер – відповідальна особа, що має доступ до управління окремими сегментами даних відповідно до своєї спеціалізації (один з семи типів менеджерів).

– Адміністратор – відповідальна особа з повним доступом до всіх функцій системи.

Загальна структура та склад системи.

– Клієнтська частина – SPA, побудований на React з використанням TypeScript та Vite;

– серверна частина – REST API-сервер на Python з використанням фреймворків Django та Django-Ninja;

– база даних – PostgreSQL для зберігання структурованих даних розкладу;

– кеш – Redis для кешування часто запитуваних даних;

– вебсервер – Nginx для обслуговування статичних файлів та проксіювання запитів;

– контейнеризація – Docker та Docker Compose для уніфікації середовищ розробки і розгортання.

Загальні обмеження. Функціонування системи потребує наявності стабільного інтернет-з’єднання у користувача (за винятком частково функціональних можливостей мобільного застосунку). Для адміністрування

системи необхідне обладнання з достатнім обсягом оперативної пам’яті та обчислювальних ресурсів для забезпечення продуктивної роботи з 1000 і більше одночасними користувачами.

2.3.3 Функції системи

Автентифікація та авторизація користувачів. Система забезпечує вхід користувачів за електронною поштою та паролем, генерацію JWT-токенів для подальших запитів та керування активними токенами.

Вхідна інформація: електронна пошта, пароль, ідентифікатор пристрою.

Вихідна інформація: токен доступу (access token), токен оновлення (refresh token у HttpOnly cookie).

Функціональні вимоги:

– хешування паролів алгоритмом bcrypt;

– зберігання виданих токенів у базі даних для підтримки механізму відкликання;

– видалення токенів при виході з системи.

Перегляд розкладу занять. Користувач (у тому числі неавторизований) може переглянути розклад будь-якої групи або активного викладача з фільтрацією за типом тижня та підгрупою.

Вхідна інформація: номер групи або ім’я викладача, тип тижня (над/під рискою), підгрупа (А/Б).

Вихідна інформація: перелік занять із зазначенням дисципліни, викладача, аудиторії, часового слоту та типу заняття.

Функціональні вимоги:

– формування розкладу викладача динамічно на основі даних розкладу груп;

– відображення поточного заняття та дня тижня на основі серверного часу.

Управління розкладом старостою. Авторизований староста може додавати, редагувати та видаляти заняття у розкладі виключно власної групи.

Вхідна інформація: параметри заняття (дисципліна, викладач, аудиторія, часовий слот, тип, підгрупа).

Вихідна інформація: оновлений розклад групи.

Функціональні вимоги:

– обмеження прав редагування виключно групою, у якій користувач призначений старостою;

– автоматичне оновлення мітки часу останньої зміни розкладу групи.

Управління розкладом менеджером та адміністратором. Менеджер розкладу або адміністратор може додавати, редагувати та видаляти заняття у розкладі будь-якої групи.

Вхідна інформація: номер групи, параметри заняття (дисципліна, викладач, аудиторія, часовий слот, тип, підгрупа).

Вихідна інформація: оновлений розклад групи.

Функціональні вимоги:

– можливість призначення одного заняття декільком групам одночасно (потокові лекції);

– контроль унікальності комбінації дисципліна–викладач–аудиторія–часовий слот–тип.

Управління довідниковими даними. Профільні менеджери та адміністратор можуть створювати, редагувати та деактивувати записи у довідниках системи: факультети, групи, викладачі, дисципліни, аудиторії.

Вхідна інформація: дані відповідного об’єкта (назва або ім’я, номер або код, та інші параметри).

Вихідна інформація: оновлений стан довідника.

Функціональні вимоги:

– реалізація м’якого видалення (soft delete) через поле is_active для збереження історичної цілісності даних;

– доступ до управління кожним довідником надається лише відповідному профільному менеджеру.

Управління обліковими записами користувачів. Адміністратор або менеджер клієнтів може створювати облікові записи, призначати ролі та редагувати дані користувачів.

Вхідна інформація: дані користувача (ПІБ, електронна пошта, пароль, ролі).

Вихідна інформація: створений або оновлений обліковий запис.

Функціональні вимоги:

– підтримка призначення декількох ролей одному користувачу одночасно;

– зміна пароля користувача адміністратором без знання поточного пароля.

2.3.4 Вимоги до інформаційного забезпечення

Джерела та зміст вхідної інформації. Основними джерелами вхідної інформації є дані, що вводяться авторизованими користувачами (старостами, менеджерами, адміністратором) через клієнтський інтерфейс системи. Вхідними даними є інформація про дисципліни, викладачів, аудиторії, групи, факультети та заняття, що формують розклад.

Нормативно-довідкова інформація. Система оперує наступними довідковими даними:

– довідник розкладу дзвінків;

– довідник днів тижня (понеділок – п’ятниця);

– довідник порядкових номерів пар (1 – 6);

– довідник типів тижнів (парний / непарний);

– довідник посад викладачів (професор, доцент, старший викладач, викладач, аспірант);

– довідник типів занять (лекція, практика);

– довідник підгруп (А, Б).

Вимоги до способів організації, збереження та ведення інформації.

– Дані зберігаються у реляційній базі даних PostgreSQL з реалізацією моделей через ORM Django;

– передбачено щоденне автоматичне резервне копіювання бази даних із збереженням архівних копій за останні 30 днів;

– конфіденційні дані (паролі користувачів) зберігаються у хешованому вигляді з використанням алгоритму Bcrypt;

– кешування часто запитуваних даних реалізується через Redis із автоматичною інвалідацією кешу при зміні відповідних даних у базі.

2.3.5 Вимоги до технічного забезпечення

Система розгортається на сервері з операційною системою сімейства GNU/Linux (рекомендовано Ubuntu 22.04 LTS або новіша версія) з мінімальними технічними характеристиками: процесор з 4 і більше ядрами, оперативна пам’ять обсягом 8 ГБ і більше, твердотільний накопичувач обсягом 50 ГБ і більше, мережне підключення зі швидкістю 100 Мбіт/с і більше. Користувачі взаємодіють із системою через персональні комп’ютери, ноутбуки, планшети або смартфони, що мають доступ до Інтернету та сучасний веббраузер.

2.3.6 Вимоги до програмного забезпечення

Архітектура програмної системи. Система реалізує клієнт-серверну архітектуру з чітким розмежуванням клієнтської (Single Page Application) та серверної (REST API) частин. Серверна частина має модульну структуру з виділенням окремих модулів для управління розкладом, користувачами, довідковими даними та автентифікацією.

Системне програмне забезпечення. Сервер працює під управлінням операційної системи Ubuntu 22.04 LTS. Вебсервер – Nginx, що виконує функції reverse proxy, балансування навантаження та обслуговування статичних файлів. Контейнеризація компонентів реалізується через Docker із оркестрацією Docker Compose.

Мережне програмне забезпечення. Усі запити між клієнтом та сервером передаються через захищений протокол HTTPS із застосуванням сертифіката SSL/TLS. Підтримуються сучасні веббраузери: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Opera.

Програмне забезпечення ведення інформаційної бази. Як основна СКБД використовується PostgreSQL версії 14 або новішої. Для кешування часто запитуваних даних застосовується Redis версії 7 або новішої.

Мова та технологія розробки. Серверна частина реалізується на мові програмування Python версії 3.10 або новішої з використанням фреймворків Django (як основа для роботи з ORM, міграціями та адмін-панеллю) та Django-Ninja (для побудови REST API). Клієнтська частина реалізується на мові TypeScript з використанням бібліотеки React та збиральника Vite.

2.3.7 Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача. Адаптивний дизайн із коректним відображенням на пристроях з роздільною здатністю від 320×480 до 2560×1440. Інтерфейс підтримує українську мову як основну.

Апаратний інтерфейс. Не потребує спеціалізованого обладнання; працює на стандартних ПК, ноутбуках, планшетах та смартфонах.

Програмний інтерфейс. Взаємодія між клієнтською та серверною частинами здійснюється через REST API з використанням стандартних HTTP-методів (GET, POST, PATCH, DELETE). Документація API генерується автоматично у форматі OpenAPI 3.0 засобами Django-Ninja.

Комунікаційний протокол. Усі запити передаються виключно через захищений протокол HTTPS із застосуванням сертифікатів SSL/TLS.

2.3.8 Властивості програмного забезпечення

Доступність. Цільовий показник доступності – 99,9% (не більше 8 годин 45 хвилин простою на рік).

Супроводжуваність. Модульна структура коду з чітким розмежуванням рівнів; документування API засобами OpenAPI; використання системи контролю версій Git.

Переносимість. Контейнеризація через Docker забезпечує розгортання на будь-якій інфраструктурі з підтримкою Docker. Кросбраузерна сумісність клієнтської частини.

Продуктивність. Час відповіді сервера на типовий запит не повинен перевищувати 200 мс при нормальному навантаженні. Система має забезпечувати одночасну роботу 1000 і більше користувачів у пікові періоди.

Надійність. Щоденне резервне копіювання з можливістю відновлення за останні 30 діб. Автоматичний перезапуск контейнерів при збоях.

Безпека. Захист від SQL-ін’єкцій через ORM з параметризованими запитами; захист від XSS через автоматичне екранування React; захист від CSRF через механізм токенів Django; автентифікація через JWT з підтримкою відкликання токенів; rate limiting на рівні Nginx та Django-Ninja.

2.3.9 Інші вимоги

Логування критичних дій користувачів (зміни розкладу, управління обліковими записами) для забезпечення можливості аудиту. Передбачена можливість горизонтального масштабування системи шляхом додавання екземплярів backend-сервісу без змін у структурі бази даних.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено аналіз сучасного стану інструментарію та підходів до розробки вебзастосунків на основі публікацій у фахових виданнях категорії Б, а також сформовано специфікацію вимог до програмного забезпечення розроблюваної системи.

За результатами аналізу двох наукових публікацій визначено ключові тенденції сучасної веброзробки. Дослідження архітектурних підходів до побудови масштабованих вебзастосунків підтвердило доцільність застосування модульної

клієнт-серверної архітектури, кешування даних, балансування навантаження та контейнеризації компонентів системи для забезпечення масштабованості та надійності. Аналіз підходів до багаторівневого захисту даних у вебзастосунках обґрунтував необхідність комплексного поєднання механізмів хешування паролів, токенізованої автентифікації та rate limiting для досягнення балансу між рівнем безпеки та продуктивністю системи. Зіставлення виявлених тенденцій з технічними рішеннями розроблюваної системи показало їхню відповідність сучасному стану галузі.

Сформована специфікація вимог до програмного забезпечення визначає розроблювану систему як вебзастосунок з децентралізованою моделлю формування розкладу, де кожна академічна група самостійно управляє власним розкладом через старосту, а адміністративний персонал та профільні менеджери здійснюють загальний контроль. Специфікація охоплює шість основних функцій системи, вимоги до інформаційного, технічного та програмного забезпечення, вимоги до зовнішніх інтерфейсів та властивості програмного забезпечення.

Результатом проведеної у розділі роботи є обґрунтована основа для подальшого проєктування архітектури системи, проєктування бази даних та розробки графічних моделей у нотаціях UML та ER, що становитиме зміст третього розділу кваліфікаційної роботи.

3 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

3.1 Моделювання предметної області

3.1.1 Структура та зв’язки бази даних

Структура бази даних розроблюваного вебзастосунку складається з двох функціональних груп сутностей: домен автентифікації та управління користувачами, і домен академічного розкладу.

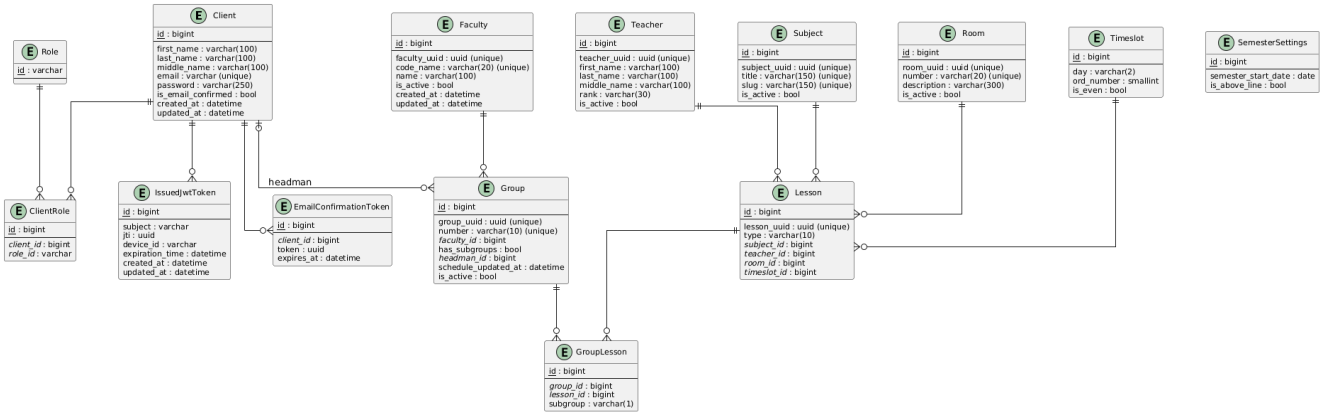


Рисунок 3.1 – ER-діаграма проєкту

Домен автентифікації та управління користувачами включає такі сутності. Сутність **Client** є центральною сутністю домену і зберігає дані облікового запису користувача:

– ідентифікатор;

– прізвище;

– ім’я;

– ім’я по батькові;

– електронну пошту (унікальну);

– хешований пароль;

– ознаку підтвердження електронної пошти;

– а також часові мітки створення та оновлення запису (вони наявні для всіх сутностей в системі, тому далі не будуть згадуватися).

Сутність **Role** містить перелік дозволених ролей системи і пов’язана з Client через проміжну таблицю відношенням M2M (Many-To-Many), що дозволяє призначати одному користувачу декілька ролей одночасно:

– ідентифікатор ролі (рядкове значення з визначеним переліком).

Сутність **IssuedJwtToken** зберігає метадані кожного виданого JWT-токена для підтримки механізму відкликання:

– електронну пошту суб’єкта;

– унікальний ідентифікатор токена (JTI – JWT ID);

– ідентифікатор пристрою;

– час закінчення дії токену.

Сутність **EmailConfirmationToken** зберігає токени підтвердження електронної пошти:

– зовнішній ключ на Client;

– значення токена (uuid);

– час закінчення дії токена.

Домен академічного розкладу є основним і включає вісім сутностей. Сутність **Faculty** зберігає дані факультету:

– унікальний UUID;

– скорочену назву (унікальну);

– повну назву;

– is_active (для функції «м’якого» видалення).

Сутність **Group** представляє академічну групу університету:

– унікальний UUID;

– номер групи(унікальний);

– зовнішній ключ на Faculty;

– ознаку наявності підгруп;

– посилання на старосту (зовнішній ключ на Client);

– мітку часу останньої зміни розкладу;

– is_active.

Сутність **Teacher** зберігає дані викладача:

– унікальний UUID;

– прізвище;

– ім’я;

– ім’я по батькові;

– посаду з визначеним переліком значень;

– is_active.

Сутність **Subject** представляє навчальну дисципліну:

– унікальний UUID;

– назву (унікальну);

– slug-поле для формування URL-сумісних ідентифікаторів (унікальне);

– is_active.

Сутність **Room** зберігає дані навчальної аудиторії:

– унікальний UUID;

– унікальний номер аудиторії;

– необов’язковий опис;

– is_active.

Сутність **Timeslot** визначає часовий слот у розкладі. Комбінація трьох полів є унікальною:

– день тижня;

– порядковий номер пари (1–6);

– ознаку парності тижня (is_even).

Сутність **Lesson** є атомарною одиницею розкладу і об’єднує чотири зовнішніх ключі та тип заняття. Унікальне обмеження на комбінацію всіх п’яти полів унеможливлює дублювання ідентичних занять:

– унікальний UUID;

– тип заняття (лекція або практика);

– зовнішній ключ на Subject;

– зовнішній ключ на Teacher;

– зовнішній ключ на Room;

– зовнішній ключ на Timeslot.

Сутність **GroupLesson** є проміжною таблицею відношення M2M між Group та Lesson і додатково зберігає атрибут підгрупи:

– зовнішній ключ на Group;

– зовнішній ключ на Lesson;

– підгрупу (А, Б або відсутня).

Сутність **SemesterSettings** є одиночним записом (singleton) і зберігає параметри поточного семестру, що використовуються для визначення парності навчального тижня:

– дату початку семестру;

– ознаку того, чи перший тиждень семестру є «над рискою» (is_above_line).

3.1.2 Варіанти використання системи

Діаграма варіантів використання відображає взаємодію між акторами системи та її функціями. У розроблюваному вебзастосунку визначено дев’ять ролей користувачів, кожна з яких має чітко окреслений набір повноважень.

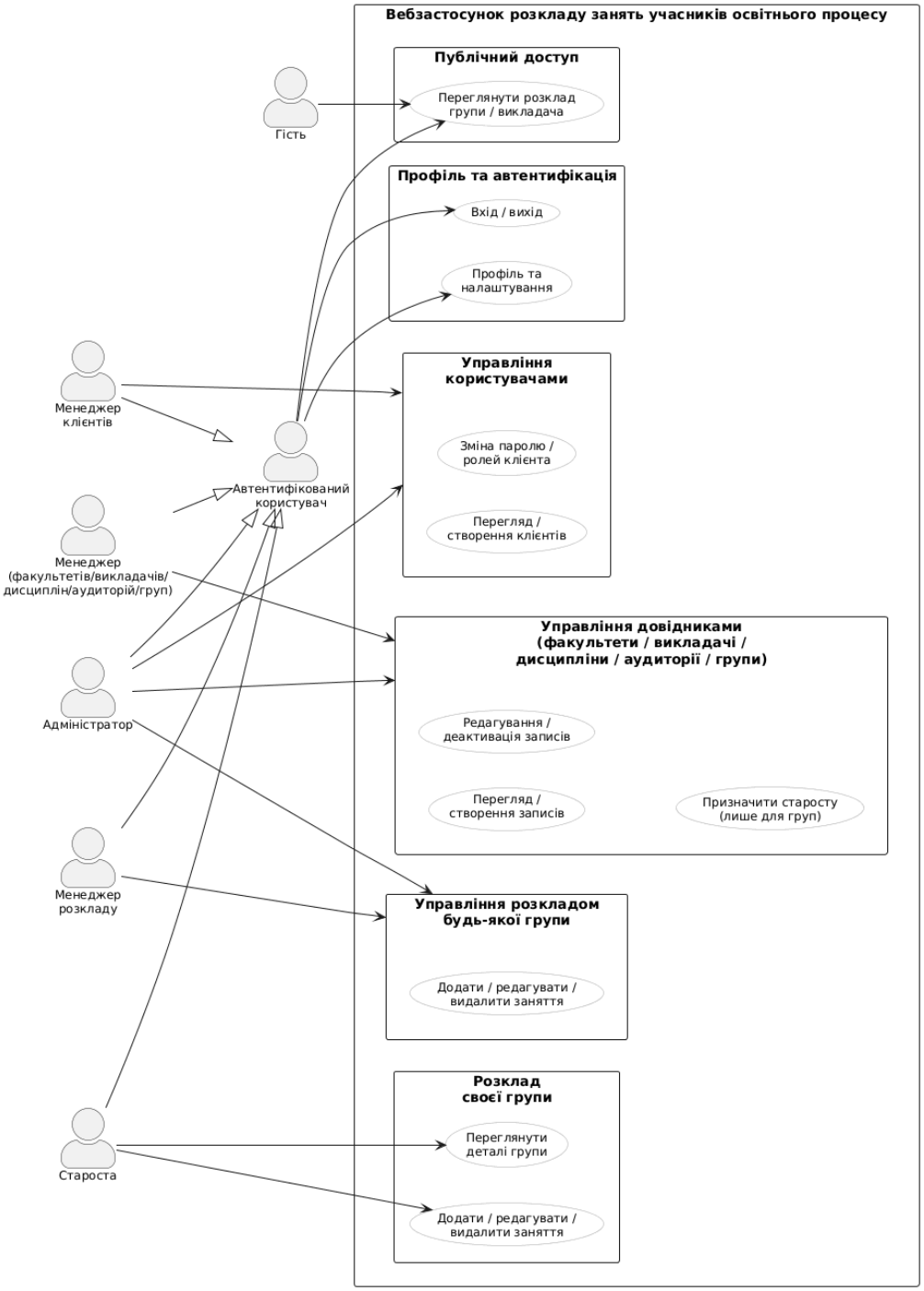


Рисунок 3.2 – Діаграма варіантів використання

Гість – незареєстрований користувач, який має доступ лише до публічної частини системи – перегляду розкладу груп та викладачів.

Автентифікований користувач – будь-який авторизований користувач системи незалежно від ролі. Може виконувати вхід та вихід із системи, переглядати і редагувати власний профіль (ПІБ, електронну пошту, пароль), а також переглядати розклад груп та викладачів.

Староста – авторизований студент, призначений старостою певної групи. Успадковує всі можливості автентифікованого користувача та додатково може переглядати деталі власної групи, додавати, редагувати та видаляти заняття у розкладі виключно своєї групи.

Менеджер клієнтів – відповідає за управління обліковими записами користувачів – перегляд списку користувачів, створення нових облікових записів, зміну паролів та ролей користувачів.

Менеджер факультетів – відповідає за управління довідником факультетів – перегляд, створення, редагування назви та скороченої назви, деактивацію факультетів.

Менеджер викладачів – відповідає за управління довідником викладачів – перегляд, створення, редагування імені та посади, деактивацію викладачів.

Менеджер дисциплін – відповідає за управління довідником навчальних дисциплін – перегляд, створення, редагування та деактивацію дисциплін.

Менеджер аудиторій – відповідає за управління довідником аудиторій – перегляд, створення, редагування та деактивацію аудиторій.

Менеджер груп – відповідає за управління довідником академічних груп – перегляд, створення, видалення груп та призначення старости групи.

Менеджер розкладу – відповідає за управління розкладом будь-якої групи – додавання, редагування та видалення занять у розкладі будь-якої групи університету.

Адміністратор – має повний доступ до всіх функцій системи.

3.2 Моделювання взаємодії та архітектури системи

3.2.1 Процес автентифікації та авторизації користувачів

Процес автентифікації та авторизації у розроблюваному вебзастосунку реалізовано на основі механізму JWT-токенів і складається з трьох взаємопов’язаних сценаріїв.

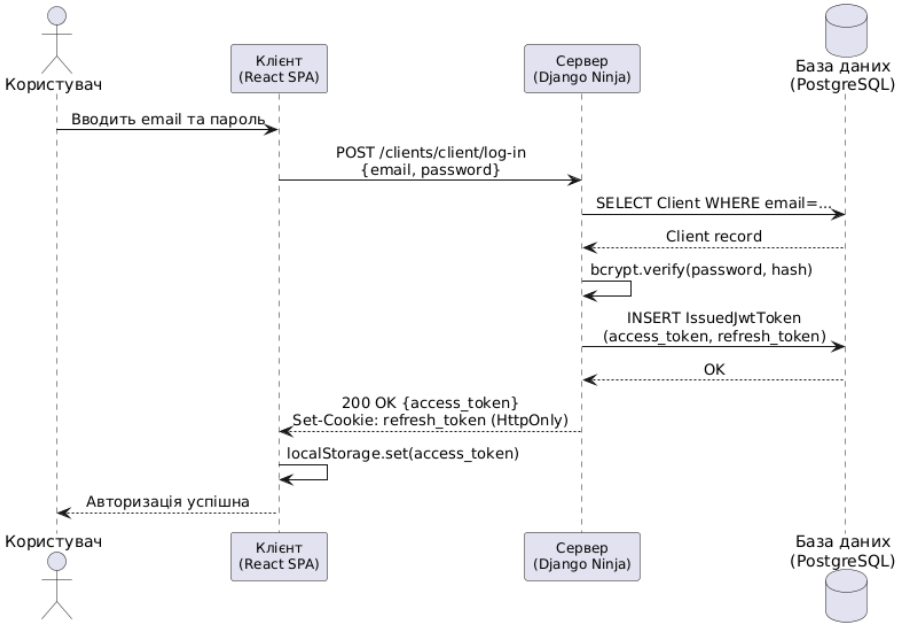


Рисунок 3.3 – Діаграма послідовності автентифікації

Автентифікація відбувається при вході користувача в систему. Користувач надсилає електронну пошту та пароль на ендпоінт POST /clients/client/log-in. Сервер отримує обліковий запис з бази даних та верифікує пароль за допомогою алгоритму bcrypt. У разі успіху генеруються два токени: короткостроковий токен доступу (access token), що повертається у тілі відповіді, та довгостроковий токен оновлення (refresh token), що встановлюється у HttpOnly cookie. Метадані обох токенів зберігаються в таблиці IssuedJwtToken. Клієнтська частина зберігає токен доступу в localStorage та використовує його у заголовку Authorization: Bearer для подальших запитів.

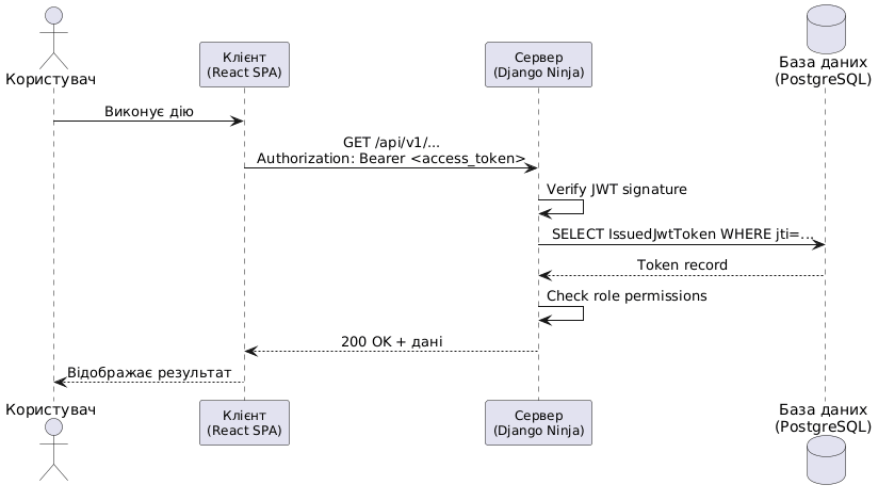


Рисунок 3.4 – Діаграма послідовності авторизації запиту

Авторизація запиту відбувається при кожному зверненні автентифікованого користувача до захищених ресурсів API. Сервер перевіряє підпис JWT-токена, наявність його запису в таблиці IssuedJwtToken та відповідність ролі користувача вимогам ендпоінту. Зберігання виданих токенів у базі даних усуває відомий недолік stateless-токенів – неможливість їх відкликання до закінчення терміну дії.

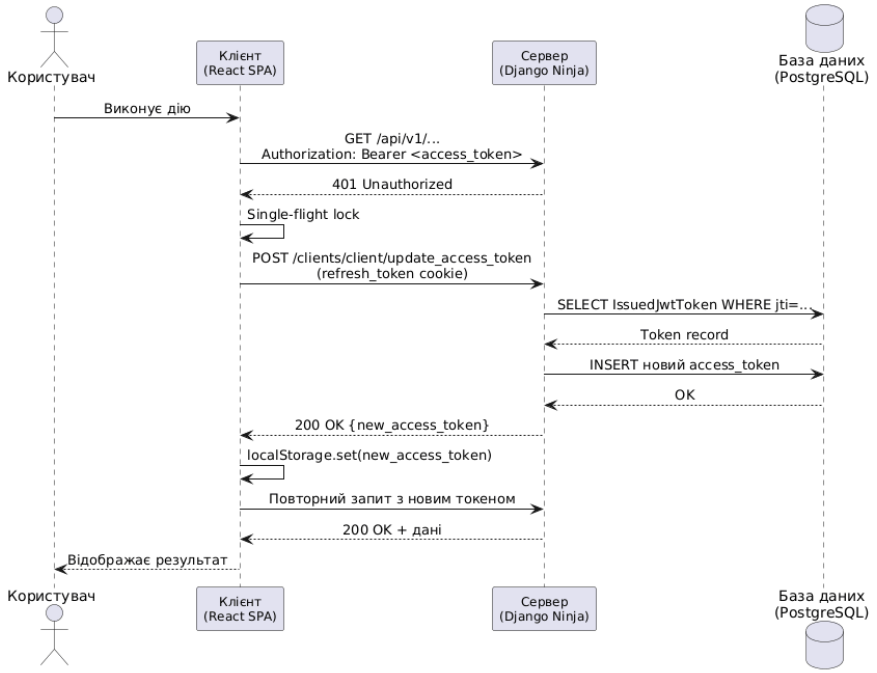


Рисунок 3.5 – Діаграма послідовності оновлення токена доступу

Оновлення токена доступу відбувається автоматично при отриманні відповіді з кодом 401. Клієнтська частина надсилає запит на ендпоінт POST /clients/client/update_access_token, передаючи refresh token через HttpOnly cookie. Для запобігання одночасному відправленню кількох запитів на оновлення при паралельних 401-відповідях реалізовано патерн single-flight – усі паралельні запити очікують результату одного спільного запиту. Після успішного оновлення оригінальний запит повторюється автоматично з новим токеном доступу.

3.2.2 Процес управління розкладом

Процес управління розкладом відображає децентралізовану модель формування розкладу занять. Розглянемо два основних сценарії: додавання заняття

старостою до розкладу власної групи та додавання заняття менеджером розкладу або адміністратором до розкладу будь-якої групи.

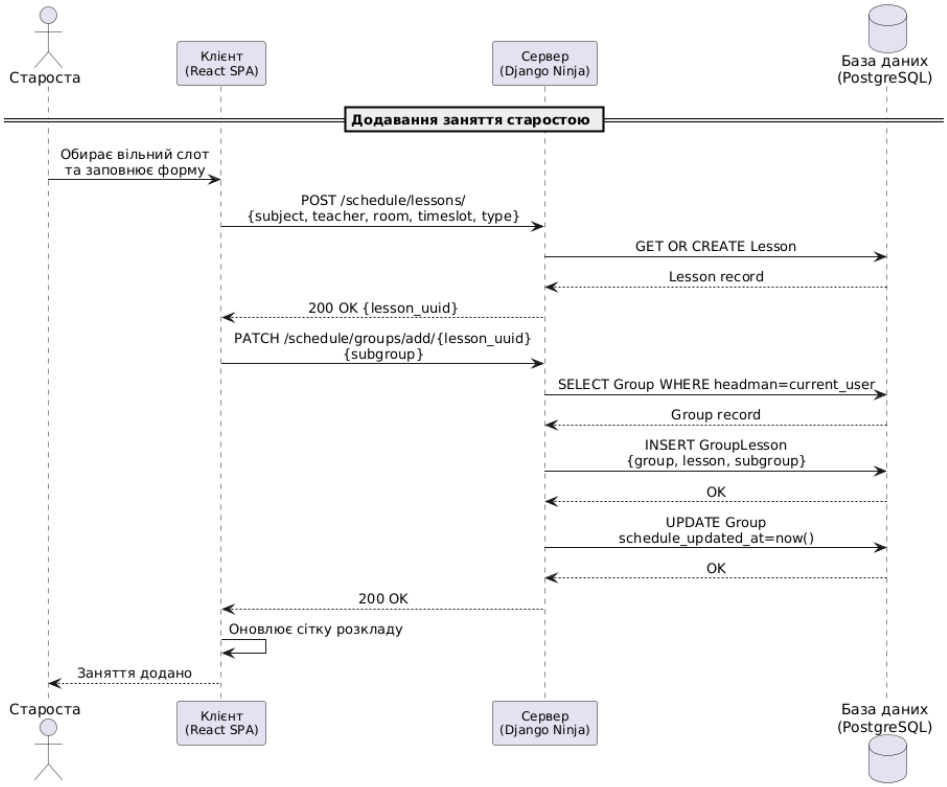


Рисунок 3.6 – Діаграма послідовності додавання заняття старостою

Додавання заняття старостою. Староста обирає вільний часовий слот та підгрупу у сітці розкладу своєї групи та заповнює форму заняття, вказуючи дисципліну, викладача, аудиторію, тип заняття. Клієнтська частина надсилає запит на ендпоінт POST /schedule/lessons/ для створення або отримання існуючого заняття з такими самими параметрами. Після цього надсилається запит PATCH /schedule/groups/add/{lesson_uuid} для прив’язки заняття до групи старости. Сервер перевіряє що користувач є старостою саме цієї групи, зберігає запис у таблиці GroupLesson та оновлює мітку schedule_updated_at групи. Клієнтська частина оновлює сітку розкладу.

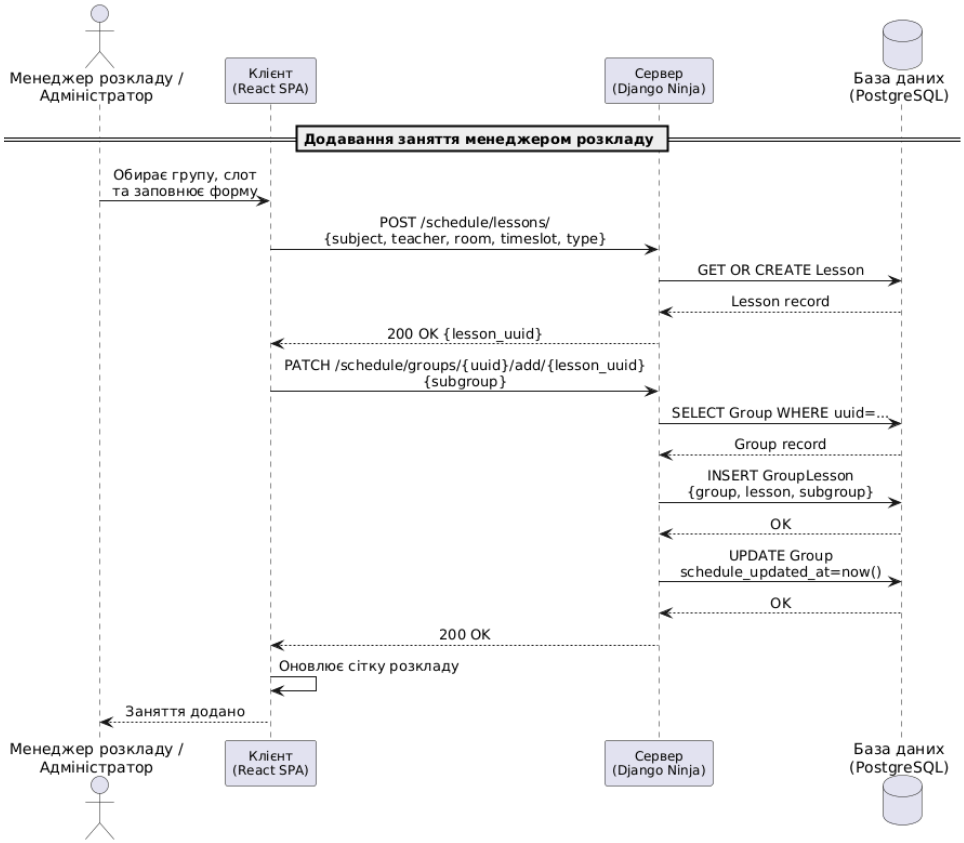


Рисунок 3.7 – Діаграма послідовності додавання заняття менеджером/адміністратором

Додавання заняття менеджером розкладу. Менеджер розкладу або адміністратор обирає групу через пошуковий компонент, після чого виконує аналогічні кроки. Запит надсилається на ендпоінт PATCH /schedule/groups/{uuid}/add/{lesson_uuid}, де uuid – ідентифікатор обраної групи. Сервер перевіряє наявність ролі schedule_manager або admin та виконує прив’язку заняття.

3.2.3 Алгоритм визначення поточного заняття

Для визначення поточного заняття система використовує серверний час та параметри семестру, що зберігаються в сутності SemesterSettings. Алгоритм складається з двох послідовних етапів: визначення типу поточного навчального тижня (парний/непарний) та визначення поточного часового слоту.

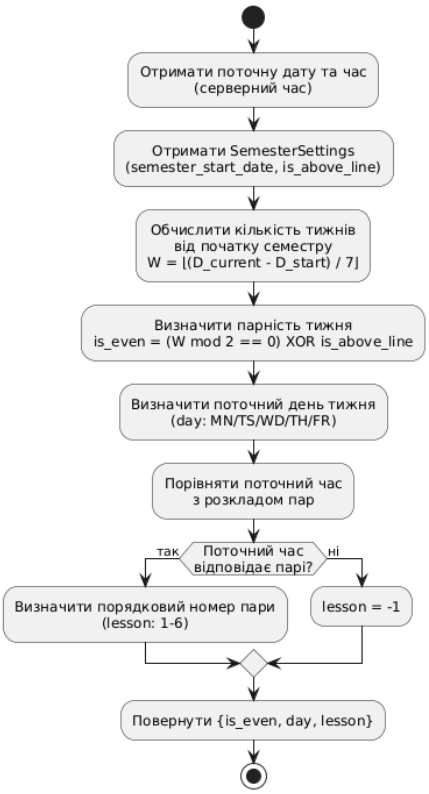


Рисунок 3.8 – Діаграма діяльності алгоритму визначення поточного заняття

Визначення типу тижня. Тип поточного навчального тижня обчислюється на основі дати початку семестру та ознаки is_above_line. Кількість тижнів від початку семестру обчислюється за формулою:

$$W = \left[\frac{D_{current} - D_{start}}{7}\right]$$

де $D_{current}$ – поточна дата;

D_{start} – дата початку семестру;

W – кількість повних тижнів від початку семестру.

Парність поточного тижня визначається як:

$$is\_even = (Wmod2 == 0) \oplus is\_above\_line$$

де $\oplus$ – операція XOR;

Якщо is_above_line має значення True, перший тиждень семестру вважається «над рискою» (непарним), і парність чергується відповідно.

Визначення поточного часового слоту. Після визначення типу тижня система порівнює поточний час із розкладом пар університету для визначення порядкового номера поточної пари. Ендпоінт GET /time/current повертає клієнту

об’єкт із трьома полями: тип тижня (is_even), поточний день тижня (day) та порядковий номер поточної пари (lesson). Клієнтська частина використовує ці дані для підсвічування поточного заняття в сітці розкладу.

3.2.4 Розгортання системи

Розгортання вебзастосунку реалізовано на основі контейнеризації з використанням Docker та Docker Compose. Діаграма розгортання (рис. 3.9) відображає фізичну структуру системи та взаємодію між її вузлами.

Система складається з п’яти Docker-контейнерів, що функціонують у межах єдиної мережі Docker Compose.

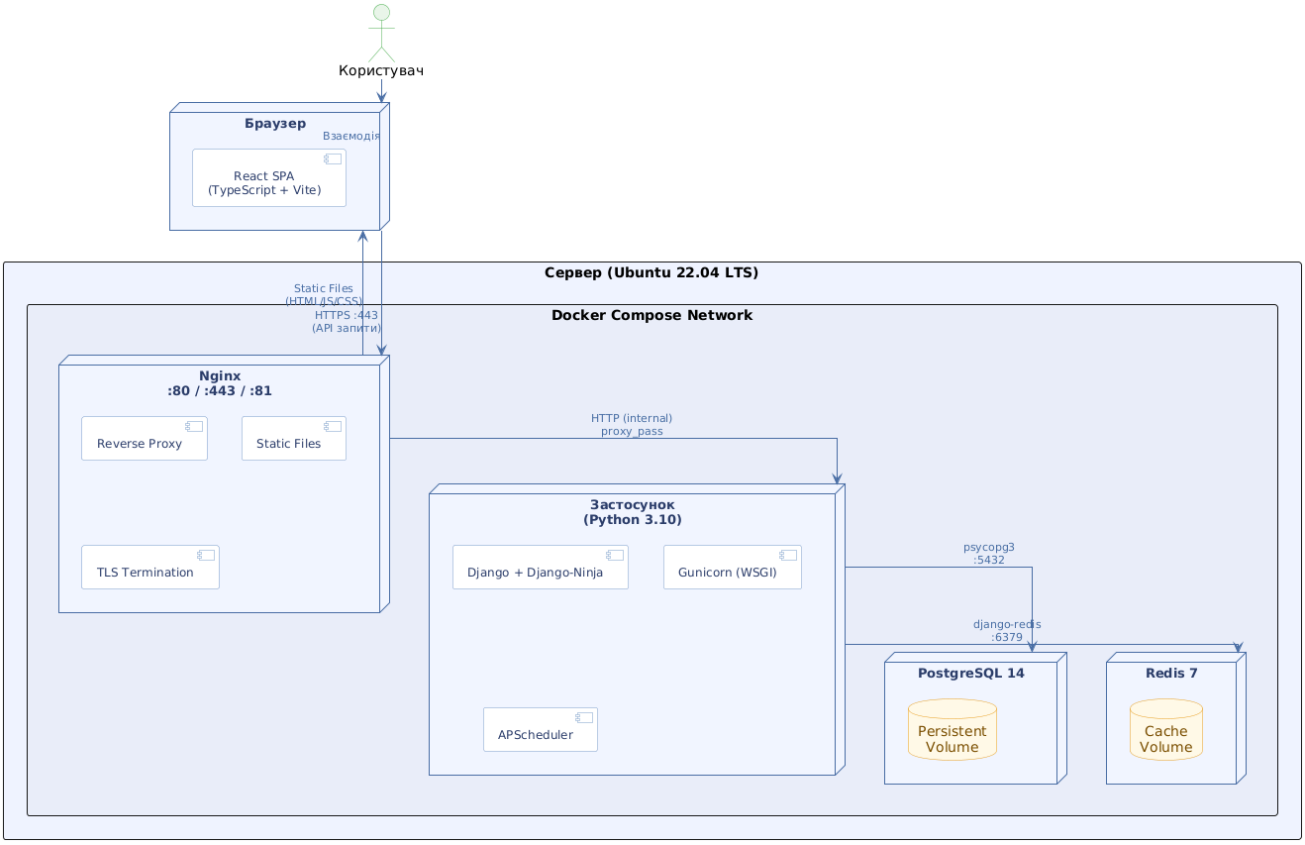


Рисунок 3.9 – Діаграма розгортання системи

Контейнер Nginx є точкою входу для всіх зовнішніх запитів. В продакшн-середовищі він обслуговує запити на портах 80 (HTTP) та 443 (HTTPS), автоматично перенаправляючи HTTP-трафік на HTTPS. Nginx виконує термінацію TLS, роздачу статичних файлів клієнтської частини та проксіювання API-запитів до

контейнера застосунку. Адміністративна панель Django доступна виключно через порт 81, який не є публічним.

Контейнер застосунку запускає серверну частину на основі Python 3.10 з використанням Gunicorn як WSGI-сервера в продакшн-середовищі. Після старту контейнер автоматично виконує міграції бази даних та запускає фоновий планувальник APScheduler для очищення прострочених JWT-токенів.

Контейнер PostgreSQL забезпечує зберігання всіх структурованих даних системи. Дані зберігаються у змонтованому Docker volume для збереження між перезапусками контейнера.

Контейнер Redis використовується як кеш для зберігання часто запитуваних даних розкладу.

3.3 Архітектура програмного забезпечення

Вибір технологій та інструментів розробки. При проєктуванні вебзастосунку розкладу занять здійснено обґрунтований вибір технологій для серверної частини, клієнтської частини та інфраструктури розгортання.

Серверна частина. Як основний фреймворк серверної частини обрано Django 5 [6] у поєднанні з Django-Ninja [7]. Django забезпечує зрілу екосистему, вбудований ORM, систему міграцій та адміністративну панель. Django-Ninja обрано замість Django REST Framework (DRF) з наступних міркувань:

- автоматична валідація запитів і відповідей на основі Pydantic v2;
- автоматична генерація OpenAPI-документації;
- краща продуктивність завдяки асинхронній підтримці;
- більш лаконічний синтаксис визначення ендпоінтів.

Для управління залежностями між компонентами системи використовується легковаговий IoC-контейнер punq, що реалізує патерн ін’єкції залежностей без надмірного ускладнення коду.

Як мову програмування серверної частини обрано Python завдяки широкій екосистемі бібліотек, високій читабельності коду та нативній підтримці

асинхронного програмування. Для хешування паролів використовується бібліотека bcrypt , для роботи з JWT-токенами – PyJWT з алгоритмом підпису HS256.

База даних та кешування. Як основну СКБД обрано PostgreSQL завдяки надійності, підтримці складних запитів, розвиненій системі обмежень цілісності та широкій підтримці з боку Django ORM. Для підключення використовується асинхронний драйвер psycopg3 з пулом з'єднань. Redis обрано як систему кешування для зберігання часто запитуваних даних розкладу, що дозволяє суттєво знизити навантаження на базу даних у пікові періоди.

Клієнтська частина. Як основну бібліотеку клієнтської частини обрано React [8] завдяки компонентній архітектурі, великій спільноті та широкій екосистемі. TypeScript обрано замість JavaScript для забезпечення статичної типізації, що суттєво знижує кількість помилок під час розробки та спрощує підтримку коду. Як інструмент збірки обрано Vite [9] завдяки значно швидшому часу запуску порівняно з Webpack та нативній підтримці ES-модулів.

Для управління станом застосунку обрано React Context API замість Redux, оскільки масштаб системи не потребує складного централізованого сховища стану.

Інфраструктура. Для контейнеризації обрано Docker та Docker Compose, що забезпечує уніфікацію середовищ розробки та розгортання. Nginx використовується як зворотний проксі-сервер завдяки високій продуктивності при обслуговуванні статичних файлів та гнучким можливостям налаштування rate limiting і заголовків безпеки. Gunicorn обрано як WSGI-сервер для продакшн-середовища.

3.4 Опис інтерфейсу користувача

Інтерфейс користувача розроблюваного вебзастосунку реалізовано з урахуванням принципів адаптивного дизайну та зручності використання для різних категорій користувачів. Клієнтська частина коректно відображається на пристроях з роздільною здатністю від 320×480 до 2560×1440 пікселів і адаптує свій макет залежно від розміру екрана.

Інтерфейс складається з таких основних екранів: сторінка перегляду розкладу групи, сторінка перегляду розкладу викладача, сторінка редагування розкладу (для старости та менеджера розкладу/адміністратора), сторінки управління довідниками та сторінка профілю користувача.

Інтерфейс перегляду розкладу. Панель керування розкладом (рис. 3.10) розташована над таблицею і містить пошуковий компонент для вибору групи або викладача, перемикач підгрупи (відображається лише для груп з підгрупами), перемикач типу тижня (парний/непарний) та індикатор часу останнього оновлення розкладу.

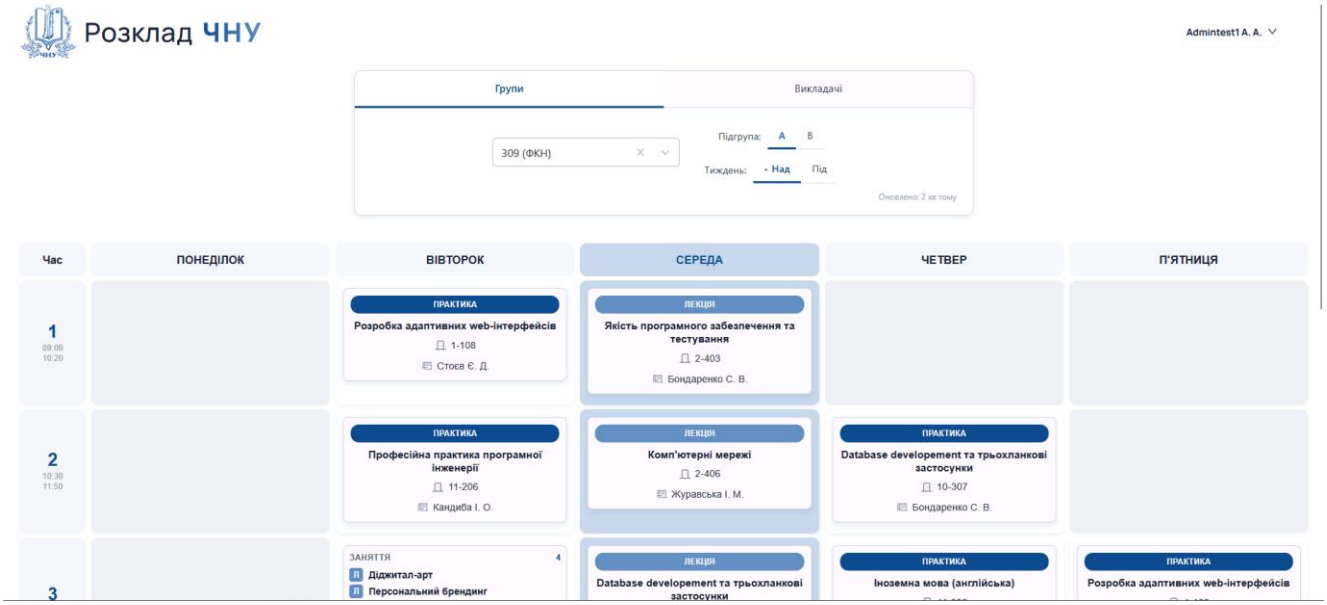


Рисунок 3.10 – Інтерфейс панелі керування розкладом

Таблиця розкладу є центральним елементом інтерфейсу та адаптується до трьох режимів відображення залежно від розміру екрана. На комп’ютері (рис. 3.11) відображається повна п’ятиколонкова сітка (один стовпець на кожен день тижня, один рядок на кожен часовий слот). На планшеті (рис. 3.12) використовується посторінковий режим з відображенням трьох днів одночасно (ПН-СР та СР-ПТ). На мобільному пристрої (рис. 3.13) відображається один день з перемиканням між днями через вкладки. Поточне заняття автоматично підсвічується зеленим в таблиці на основі серверного часу. У разі якщо група має декілька занять в один часовий слот усі вони відображаються у відповідній клітинці таблиці у вигляді стека карток.

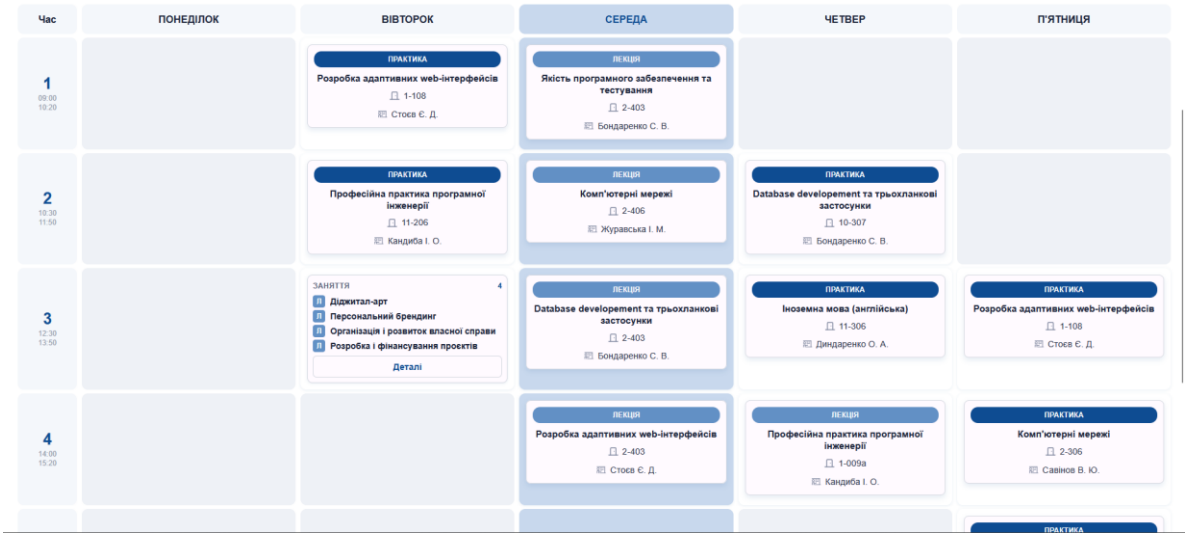


Рисунок 3.11 – Інтерфейс таблиці розкладу (Desktop View)

Рисунок 3.12 – Інтерфейс таблиці розкладу (Tablet View)

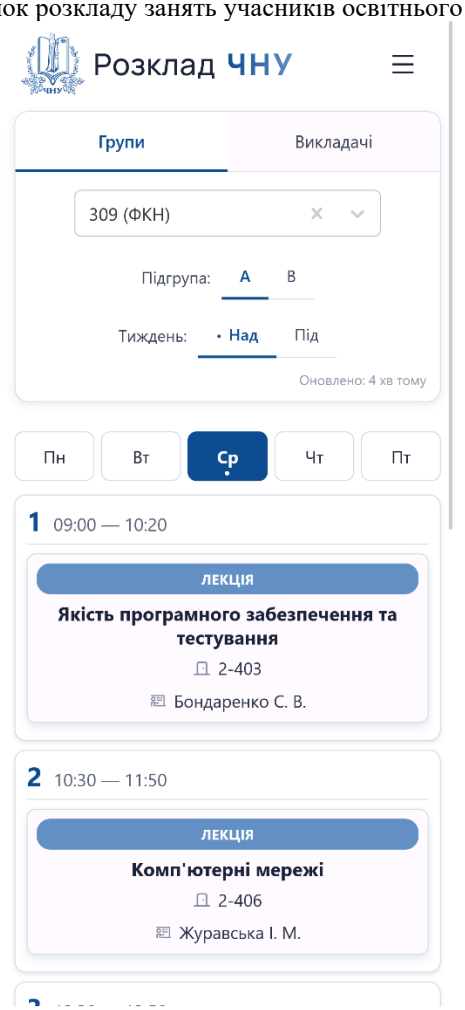


Рисунок 3.13 – Інтерфейс таблиці розкладу (Phone View)

На відміну від розкладу групи, розклад викладача (рис. 3.14) формується динамічно на основі всіх занять усіх груп до яких він прив’язаний.

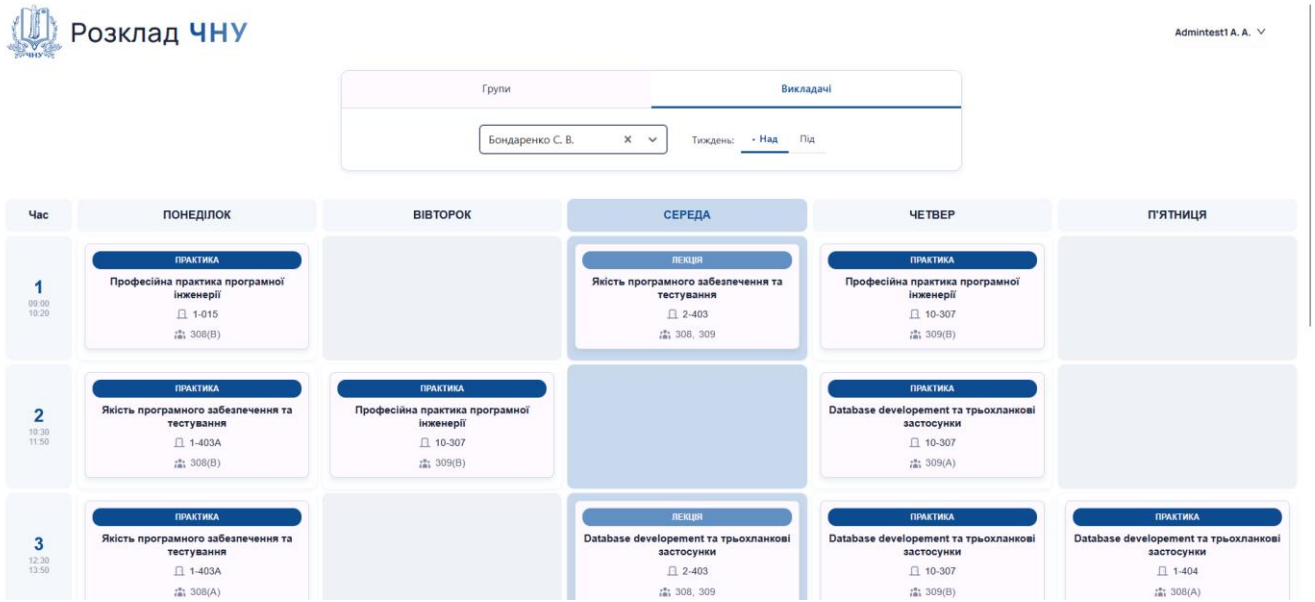


Рисунок 3.14 – Інтерфейс таблиці розкладу викладача

Інтерфейс редагування розкладу. Режим редагування активується для авторизованих користувачів з відповідними правами. У цьому режимі кожна порожня клітинка сітки містить кнопку додавання заняття, а існуючі заняття мають кнопку редагування (рис. 3.15). Додавання та редагування занять відбувається через модальне вікно з формою (рис. 3.16).

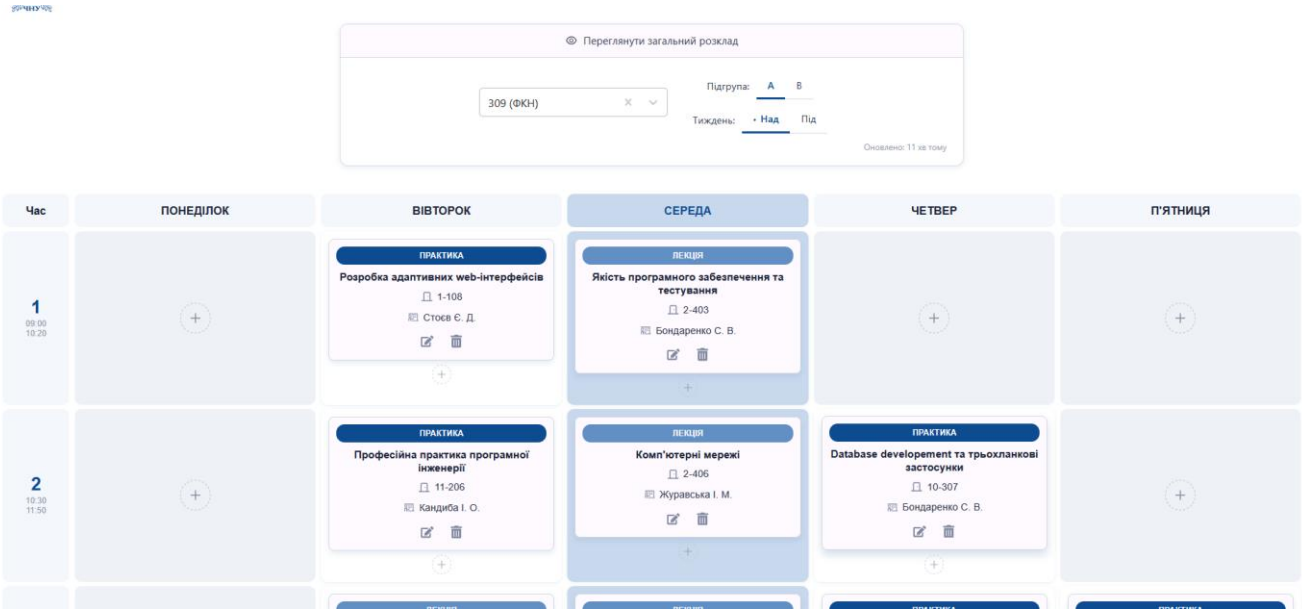


Рисунок 3.15 – Інтерфейс таблиці розкладу в режимі редагування (Desktop View)

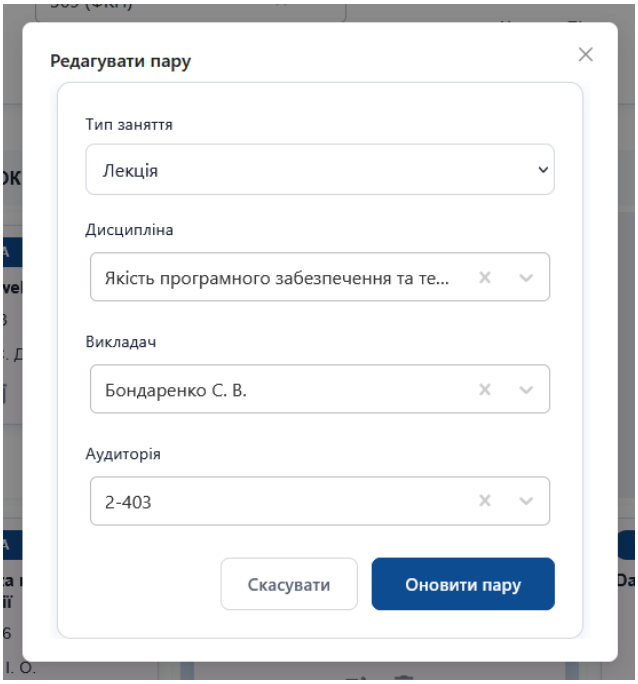


Рисунок 3.16 – Інтерфейс модального вікна з формою

Інтерфейс адміністративної панелі. Адміністративна панель містить окремі сторінки для управління кожним довідником системи. Кожна сторінка має єдину структуру: пошуковий рядок для фільтрації, список записів з посторінковою навігацією та кнопки дій для кожного запису.

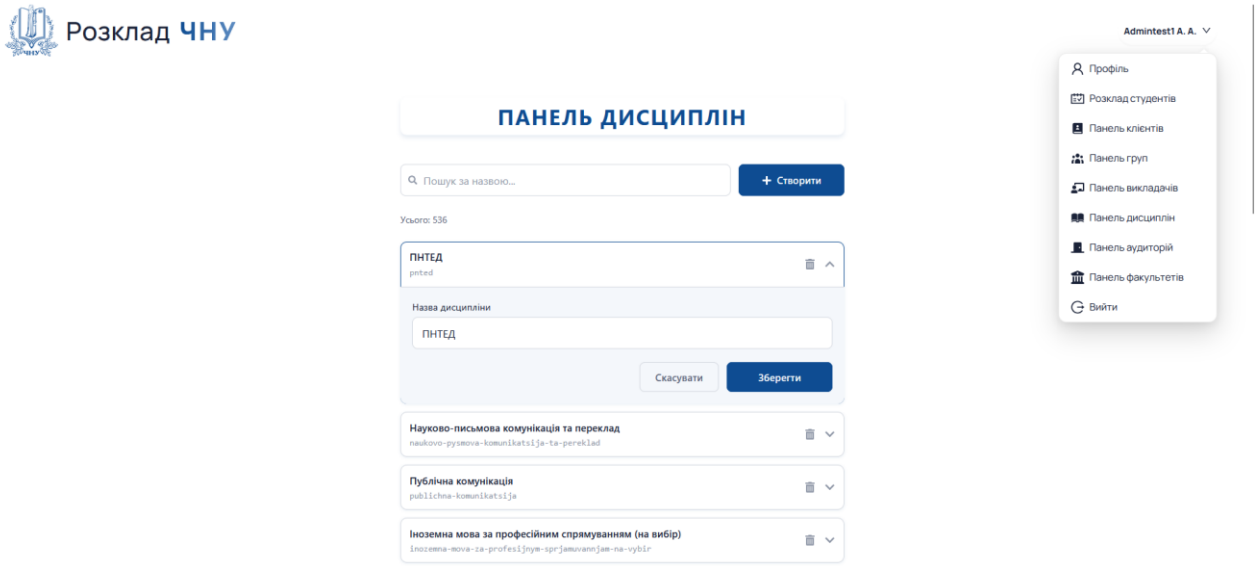


Рисунок 3.17 – Інтерфейс адміністративної панелі дисциплін

Уніфікована структура сторінок управління спрощує навчання нових користувачів та скорочує час на виконання адміністративних операцій.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи виконано проєктування та моделювання програмного забезпечення вебзастосунку розкладу занять учасників освітнього процесу.

Побудовано ER-діаграму бази даних та діаграму варіантів використання що відображає взаємодію дев’яти ролей користувачів з функціями системи. Розроблено UML-моделі ключових процесів: автентифікації та авторизації, управління розкладом, алгоритму визначення поточного заняття та розгортання системи.

Обґрунтовано вибір технологій розробки та описано інтерфейс користувача що включає адаптивну таблицю розкладу, інтерфейс редагування та адміністративну панель. Отримана проєктна документація є основою для реалізації програмного забезпечення у четвертому розділі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Реалізація програмного забезпечення

4.1.1 Реалізація формування розкладу групи

Формування розкладу групи реалізовано через двоетапний процес. На першому етапі створюється або отримується існуюче заняття (Lesson), на другому – заняття прив’язується до конкретної групи через проміжну таблицю GroupLesson.

Ключовим архітектурним рішенням є використання патерну get-or-create для сутності Lesson. Оскільки одне й те саме заняття може бути призначене декільком групам одночасно (наприклад потокова лекція), система не створює дублікати – замість цього повертається вже існуючий запис якщо комбінація дисципліна–викладач–аудиторія–часовий слот–тип вже присутня в базі даних. Це забезпечується унікальним обмеженням на рівні моделі та методом get_or_create Django ORM.

Після отримання заняття сервісний шар створює запис у таблиці GroupLesson що пов’язує групу з заняттям і додатково зберігає підгрупу (А, Б або відсутня). Унікальне обмеження на комбінацію (group, lesson, subgroup) унеможливлює дублювання призначень одного заняття для тієї самої групи та підгрупи. Після успішного збереження use-case оновлює поле schedule_updated_at групи поточним часом, що дозволяє клієнтській частині відображати час останньої зміни розкладу та інвалідує кеш Redis для відповідної групи.

4.1.2 Реалізація динамічного формування розкладу викладача

Розклад викладача не зберігається в базі даних як окрема сутність – він формується динамічно на основі даних таблиці GroupLesson. Це означає що система агрегує всі заняття всіх груп до яких прив’язаний викладач і збирає їх у єдиний розклад з урахуванням підгруп.

Центральним елементом реалізації є метод get_lessons_with_groups сервісного шару. Він отримує записи з таблиці GroupLesson з фільтрацією за

ідентифікатором викладача та типом тижня, після чого агрегує результати за допомогою двох словників-bucket: lesson_bucket що зберігає унікальні заняття та group_bucket що зберігає унікальні пари заняття-група. Для кожного заняття збираються всі групи що його відвідують, а для кожної групи – всі підгрупи. Такий підхід дозволяє обійти проблему дублювання рядків при JOIN-запитах коли одне заняття має кілька підгруп. Алгоритм динамічного формування розкладу викладача представлено у вигляді діаграми діяльності у додатку А.

Результат повертається у вигляді списку об’єктів LessonWithGroupsView – dataclass що містить сутність заняття та список груп з підгрупами. Це забезпечує чітке розмежування між шаром даних (ORM-модель) та шаром представлення (entity/view) відповідно до чотирирівневої архітектури системи. Результат виконання use-case кешується в Redis на півдня з автоматичною інвалідацією при зміні розкладу будь-якої групи викладача.

Фінальним етапом є серіалізація результату у JSON-відповідь через схеми Django-Ninja. Клас LessonWithGroupsOutSchema перетворює об’єкт LessonWithGroupsView у типізовану схему відповіді, а TeacherLessonsOutSchema об’єднує дані викладача та список його занять в єдину відповідь. Метод from_view та from_views забезпечують явне перетворення між шарами без неявного маппінгу, що відповідає принципу явності архітектури системи. Таким чином повний ланцюжок перетворення даних має вигляд: ORM-модель – сутність (entity) – представлення (view) – схема (schema) – JSON-відповідь.

4.1.3 Реалізація автентифікації та авторизації

Реалізація автентифікації та авторизації у системі побудована на трьох взаємопов’язаних компонентах: сервісі хешування паролів, сервісі JWT-токенів та механізмі перевірки прав доступу.

Сервіс хешування паролів. Клас BcryptPasswordService реалізує два методи: hash_password для хешування пароля при створенні або оновленні облікового запису та verify_password для перевірки пароля при автентифікації. Обидва методи працюють з кодуванням UTF-8 та використовують бібліотеку bcrypt. Сіль

генерується автоматично при кожному хешуванні через bcrypt.gensalt(), що унеможливлює використання rainbow table атак.

Лістинг коду функції хешування паролів:

```
def hash_password(self, plain_password: str) -> str:
    hashed_password = bcrypt.hashpw(
        password=plain_password.encode(self.HASH_ENCODING),
        salt=bcrypt.gensalt(),
    )
    return hashed_password.decode(self.HASH_ENCODING)
```

Сервіс JWT-токенів. Клас JWTTokenService реалізує генерацію та декодування JWT-токенів з алгоритмом підпису HS256. Кожен токен містить стандартні claims (iss, sub, aud, jti, iat, nbf, exp) та кастомні (client_email, client_roles, device_id, token_type). Термін дії токенів конфігурується через змінні середовища ACCESS_TOKEN_EXP та REFRESH_TOKEN_EXP. Унікальний ідентифікатор токена (jti) генерується як UUID при кожному створенні токена і використовується для відкликання.

Лістинг коду функції створення JWT токену:

```
def __sign_jwt_token(self, token_type, subject, payload, ttl=None) -> str:
    current_timestamp = convert_to_timestamp(datetime.now(tz=timezone.utc))
    data = {
        self.ISSUER: "chmnu@auth_service",
        self.SUBJECT: subject,
        self.TOKEN_TYPE: token_type,
        self.JWT_ID: self.__generate_jti(),
        self.ISSUED_AT: current_timestamp,
        self.NOT_BEFORE: current_timestamp,
    }
    if ttl:
        data[self.EXPIRATION_TIME] = current_timestamp + int(ttl.total_seconds())
    data.update(payload)
    return self.__encode_jwt(payload=data)
```

Механізм перевірки прав доступу. Клас AuthCheck реалізує перевірку токена та ролей для кожного захищеного ендпоінту. При отриманні запиту він

декодує токен, перевіряє його тип (має бути ACCESS), перевіряє наявність jti в таблиці IssuedJwtToken для виявлення відкликаних токенів, після чого порівнює ролі користувача з дозволеними ролями ендпоінту. У разі успішної перевірки дані користувача (email, ролі, jti, device_id) прикріплюються до об’єкта запиту для використання в обробниках. Екземпляри JWTBearer створюються заздалегідь як константи для кожної ролі або комбінації ролей і застосовуються до ендпоінтів декларативно.

Лістинг коду констант ролей екземплярів JWTBearer:

```
jwt_auth_schedule_manager = JWTBearer(
   [ClientRole.SCHEDULE_MANAGER, ClientRole.ADMIN])
jwt_auth_headman = JWTBearer([ClientRole.HEADMAN])
```

4.1.4 Реалізація HTTP-клієнта та обробки помилок

Клієнтська частина взаємодіє з API через кастомний HTTP-wrapper реалізований без використання сторонніх бібліотек на основі нативного Fetch API. Така архітектура забезпечує повний контроль над процесом автентифікації, обробкою помилок та управлінням токенами.

HTTP-wrapper. Модуль fetchWrapper автоматично додає до кожного запиту заголовок Authorization: Bearer з токеном доступу зі сховища localStorage, заголовок X-CSRFToken зчитаний з cookie для захисту від CSRF-атак та параметр credentials: include для автоматичної передачі HttpOnly cookie з refresh token. Усі методи HTTP (GET, POST, PATCH, DELETE) реалізовані як тонкі обгортки над fetchWrapper що формують відповідний об’єкт запиту.

Патерн single-flight для оновлення токена. При отриманні відповіді з кодом 401 wrapper автоматично ініціює оновлення токена доступу. Ключовим архітектурним рішенням є змінна refreshInFlight що зберігає поточний Promise оновлення. Якщо кілька паралельних запитів отримують 401 одночасно – всі вони очікують результату одного спільного Promise замість того щоб надсилати окремі запити на оновлення. Після успішного оновлення оригінальний запит повторюється автоматично з новим токеном. У разі невдачі localStorage очищається і

диспетчеризується подія AUTH_EXPIRED_EVENT яку перехоплює AuthProvider для переведення застосунку у стан виходу з системи.

Лістинг коду функції створення JWT токену:

```
let refreshInFlight: Promise<string> | null = null;
const refreshAccessToken = (): Promise<string> => {
  if (refreshInFlight) return refreshInFlight;
  refreshInFlight = (async () => {
    try {
      const newToken = await updateAccessToken();
      localStorage.setItem("accessToken", newToken.data.access_token);
      return newToken.data.access_token;
    } catch {
      clearTokens();
      dispatchAuthExpired();
      throw new AuthError("Сесія завершена. Увійдіть знову.");
    } finally {
      refreshInFlight = null;
    }})();
  return refreshInFlight;};
```

Ієрархія помилок. Усі помилки API представлені типізованою ієрархією класів що успадковують від базового ApiCallError. Кожен клас відповідає конкретному HTTP-статусу: AuthError (401), ForbiddenError (403), NotFoundError (404), ConflictError (409), RateLimitError (429), ServerError (5xx) та NetworkError для відсутності з’єднання. Кожна помилка містить три поля: message для відображення користувачу, detail з деталями від бекенду та code – ідентифікатор у форматі SCREAMING_SNAKE_CASE що є стабільним між релізами і використовується для програмної обробки на клієнті.

Лістинг частини коду з безовим класом помилок та реалізацією на прикладі класу RateLimitError:

```
export class ApiCallError extends Error {
  public readonly status?: number;
  public readonly detail?: string;
  public readonly code?: string;}
```

```
export class RateLimitError extends ApiCallError {
   constructor(message = "Забагато спроб. Спробуйте пізніше.",
            detail?: string, code?: string) {
      super(message, 429, detail, code);}}
```

4.2 Тестування програмного забезпечення

4.2.1 Метод та організація тестування

Для тестування серверної частини вебзастосунку обрано метод модульного тестування (unit testing). Такий підхід дозволяє перевіряти кожен компонент системи ізольовано від інших, що забезпечує швидке виявлення помилок та спрощує локалізацію їх причин. Як інструмент тестування використовується фреймворк pytest у поєднанні з бібліотекою pytest-django для інтеграції з Django ORM та тестовою базою даних.

Тестові дані генеруються за допомогою бібліотеки factory-boy через набір фабрик моделей. Кожна фабрика автоматично генерує валідні тестові об’єкти з унікальними ідентифікаторами та коректними зв’язками між сутностями. Залежності між компонентами вирішуються через DI-контейнер punq, що дозволяє тестувати реальні реалізації сервісів та use-cases без використання mock-об’єктів. Кеш Redis очищається перед кожним тестом та після нього через pytest fixture. Тести виконуються в ізольованому середовищі з окремою тестовою базою даних що розгортається через Docker Compose конфігурацію docker_compose/test.yaml.

Тести організовано у три рівні відповідно до чотирирівневої архітектури системи: тести ORM-моделей, тести сервісного шару та тести use-cases. У цьому підрозділі наведено лише частину розроблених тестів що ілюструють ключові сценарії тестування – повний набір тестів містить 401 тест-кейс.

4.2.2 Тестування ORM-моделей

Тести ORM-моделей перевіряють коректність визначення моделей, обмежень цілісності та методів серіалізації. Для моделі GroupLesson перевіряється коректне створення запису з усіма обов’язковими полями, відображення об’єкта через

__str__, спрацювання унікального обмеження при спробі створення дублікату з однаковою підгрупою та з підгрупою None, а також можливість призначення одного заняття двом різним підгрупам однієї групи.

Лістинг коду функції модульного тесту ORM-моделі GroupLesson щодо створення дублікату з підгрупою None:

```
@pytest.mark.django_db
def test_group_lesson_unique_constraint_null_subgroup():
    group = GroupModelFactory.create()
    lesson = LessonModelFactory.create()
    GroupLessonModelFactory.create(group=group, lesson=lesson, subgroup=None)
    with pytest.raises(IntegrityError):
        GroupLessonModelFactory.create(group=group, lesson=lesson, subgroup=None)
```

4.2.3 Тестування сервісного шару

Тести сервісного шару перевіряють коректність реалізації бізнес-логіки доступу до даних. Для сервісу GroupLessonService перевіряються такі сценарії: повернення True методом check_lesson_belongs_to_any_group коли заняття прив’язане до групи, повернення False для заняття без групи, успішне видалення запису GroupLesson, генерація виключення GroupLessonDeleteError при спробі видалення неіснуючого запису, повернення списку підгруп методом get_subgroup_from_group_lesson та повернення None коли підгрупи відсутні.

Лістинг коду функції модульного тесту сервісного шару GroupLesson щодо перевірки роботи методу check_lesson_belongs_to_any_group:

```
@pytest.mark.django_db
def test_check_lesson_belongs_to_any_group_false_when_orphan(
        group_lesson_service: BaseGroupLessonService):
    lesson = LessonModelFactory()
    assert group_lesson_service.check_lesson_belongs_to_any_group(
        lesson_id=lesson.id) is False
```

4.2.4 Тестування use-cases

Тести use-cases перевіряють повний сценарій виконання бізнес-операцій включаючи валідацію вхідних даних, перевірку прав доступу та коректність змін у базі даних. Для кожного use-case тестуються як негативні сценарії (некоректні вхідні дані, порушення бізнес-правил) так і успішні сценарії.

Для AdminRemoveLessonFromGroupUseCase перевіряється генерація InvalidUuidFormatStringException при некоректному UUID, генерація GroupNotFoundException для неіснуючої групи, успішне видалення заняття з автоматичним видаленням осиротілого Lesson, а також збереження Lesson якщо він прив’язаний до інших груп.

Для HeadmanAddLessonToGroupUseCase перевіряється генерація ClientNotFoundException для неіснуючого користувача, ClientRoleNotMatchingWithRequiredException якщо користувач не є старостою, HeadmanNotAssignedToAnyGroup якщо старосту не призначено до групи, GroupWithoutSubgroupsInvalidSubgroupException при спробі призначити підгрупу групі без підгруп, GroupLessonAlreadyExists при дублікаті та успішне додавання заняття до групи.

4.2.5 Результати тестування

За результатами виконання повного набору тестів усі 401 тест-кейс виконано успішно. Час виконання повного набору тестів становить 55.04 секунди. Зведену інформацію про розподіл тестів та покриття коду наведено у таблиці 4.1.

Таблиця 4.1 – Результати модульного тестування

Компонент	Кількість тестів	Покриття
Моделі (clients)	14	89-100%
Моделі (schedule)	36	96-100%
Сервіси (clients)	20	88-99%
Сервіси (schedule)	87	92-100%
Use-cases (clients)	33	68-100%
Use-cases (schedule)	97	88-100%
Автентифікація	15	88-100%
Кеш	18	92-100%

Кінець таблиці 4.1

HTTP-частина (handlers, schemas, urls)	0	0%
Загалом	401	77%

Загальне покриття коду становить 77%. Нижчий показник загального покриття пояснюється тим що HTTP-частина (обробники запитів, схеми, маршрути) не покривається модульними тестами оскільки є тонким шаром без власної бізнес-логіки. Покриття бізнес-логіки системи – сервісного шару, use-cases та ORM-моделей – становить 88–100%, що свідчить про високу якість тестування ключових компонентів системи.

4.3 Керівництво користувача

4.3.1 Керівництво користувача вебзастосунку

Вебзастосунок розкладу занять доступний через будь-який сучасний веббраузер без необхідності встановлення додаткового програмного забезпечення. Інтерфейс адаптований для роботи на пристроях з різною роздільною здатністю екрана – персональних комп'ютерах, ноутбуках, планшетах та смартфонах.

Перегляд розкладу групи. Для перегляду розкладу академічної групи необхідно перейти на сторінку пошуку груп. У пошуковому полі слід ввести номер групи або його частину – система автоматично відобразить список відповідних груп. Після вибору групи відображається таблиця розкладу для поточного типу тижня. За допомогою перемикача типу тижня користувач може переключатися між тижнем під або над рискою. Для груп з підгрупами додатково відображається перемикач підгрупи що дозволяє переглядати розклад для підгрупи А або підгрупи Б. Поточний день та поточне заняття автоматично підсвічується в таблиці відповідно до серверного часу.

Перегляд розкладу викладача. Для перегляду розкладу викладача необхідно перейти на сторінку пошуку викладачів. У пошуковому полі слід ввести прізвище викладача або його частину. Після вибору викладача відображається

таблиця його розкладу сформована динамічно на основі розкладів усіх груп до яких прив'язаний викладач. Кожна картка заняття відображає назву дисципліни, номер аудиторії, назву групи та підгрупу.

Вхід у систему. Для отримання доступу до розширених функцій системи необхідно виконати автентифікацію. На сторінці входу слід ввести електронну пошту та пароль облікового запису і натиснути кнопку входу. Після успішної автентифікації користувач отримує доступ до функцій відповідно до призначених йому ролей. У верхній частині інтерфейсу відображається меню з доступними розділами.

Редагування розкладу старостою. Після авторизації з роллю старости користувач отримує доступ до сторінки редагування розкладу власної групи. Для додавання заняття необхідно натиснути кнопку у вільному часовому слоті таблиці – відкриється модальне вікно з формою. У формі слід обрати дисципліну, викладача, аудиторію та тип заняття з відповідних пошукових полів. За наявності підгруп додатково обирається підгрупа. Після заповнення форми та підтвердження заняття відображається у таблиці розкладу. Для редагування існуючого заняття необхідно натиснути кнопку редагування на картці заняття та внести необхідні зміни у формі. Для видалення заняття слід скористатися відповідною кнопкою на картці заняття та підтвердити дію.

Управління довідниками. Користувачі з роллю відповідного менеджера або адміністратора мають доступ до сторінок управління довідниками системи через адміністративне меню. Кожна сторінка управління містить пошукове поле для фільтрації записів, таблицю записів з посторінковою навігацією та кнопки створення, редагування і деактивації для кожного запису. Для створення нового запису слід натиснути кнопку створення та заповнити форму з відповідними даними.

Управління обліковими записами. Користувачі з роллю менеджера клієнтів або адміністратора мають доступ до сторінки управління обліковими записами. На цій сторінці можна переглядати список користувачів, створювати нові облікові записи, змінювати паролі та призначати ролі користувачам.

4.3.2 Мобільний застосунок

На основі розробленого вебзастосунку створено мобільний застосунок для платформ iOS та Android з використанням сервісу BuildNatively [10]. Сервіс BuildNatively дозволяє перетворити адаптивний вебзастосунок на нативний мобільний застосунок на основі технології WebView без необхідності розробки окремого нативного інтерфейсу. Застосунок відтворює повну функціональність вебверсії у середовищі нативного мобільного застосунку забезпечуючи користувачам такі переваги як іконка на робочому екрані пристрою, повноекранний режим відображення без елементів браузера та нативна інтеграція з операційною системою.

Мобільний застосунок буде доступний для встановлення через Apple App Store для пристроїв на iOS та через Google Play для пристроїв на Android. Інтерфейс застосунку повністю відповідає мобільній версії вебзастосунку з адаптивною таблицею розкладу що відображає один день з перемиканням між днями через вкладки. Функціональність застосунку охоплює перегляд розкладу груп та викладачів, автентифікацію та всі функції доступні авторизованим користувачам відповідно до їхніх ролей.

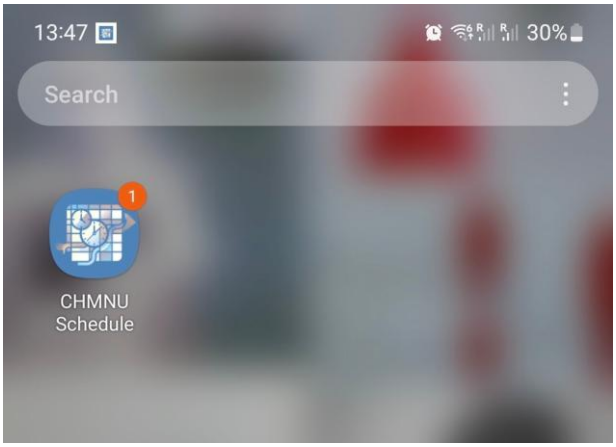
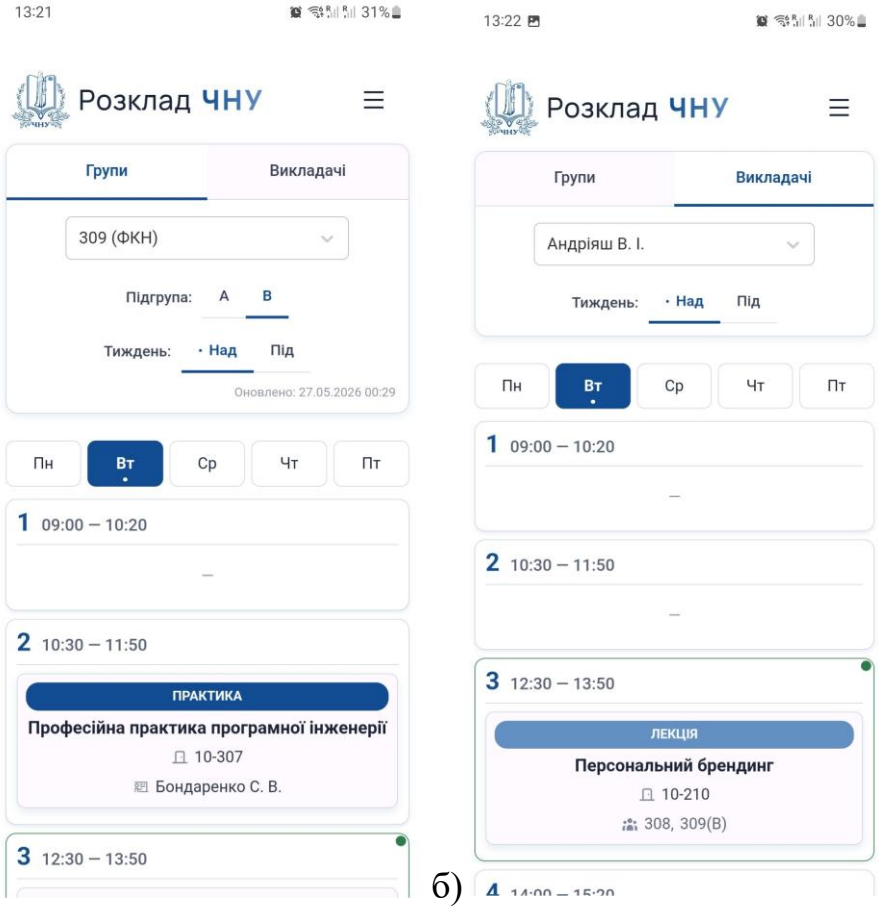


Рисунок 4.1 – Створений мобільний застосунок CHMNU Schedule встановлений на мобільний пристрій



а) б)

Рисунок 4.2 – а). Вигляд розкладу груп у мобільному застосунку; б). Вигляд розкладу викладачів у мобільному застосунку

Додатково у мобільному застосунку реалізовано функцію push-сповіщень через сервіс OneSignal [11,12]. Після встановлення застосунку користувач може підписатися на сповіщення про зміни у розкладі конкретної групи або викладача. При внесенні змін до розкладу групи – додаванні, редагуванні або видаленні заняття – підписані користувачі автоматично отримують push-сповіщення на свій мобільний пристрій.

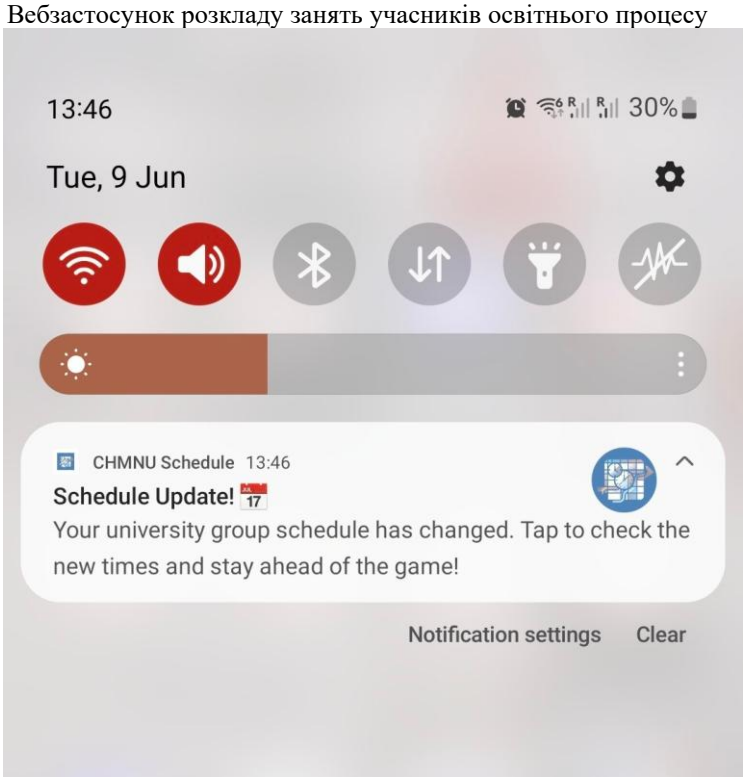


Рисунок 4.2 – Отримане push-сповіщення про зміну у розкладі групи до якої користувач підписався

Така функціональність вирішує одне з ключових обмежень наявних систем управління розкладом в українських ЗВО – відсутність механізму оперативного інформування учасників освітнього процесу про зміни у розкладі.

Висновки до розділу 4

Описано реалізацію формування розкладу групи та динамічного агрегування розкладу викладача, механізму автентифікації та авторизації на основі bcrypt і JWT з RBAC, а також HTTP-клієнта клієнтської частини з патерном single-flight та типізованою ієрархією помилок.

За результатами модульного тестування усі 401 тест-кейс виконано успішно із загальним покриттям коду 77% та покриттям бізнес-логіки 88–100%. Розроблено керівництво користувача для всіх категорій користувачів системи та описано мобільний застосунок на основі WebView з підтримкою push-сповіщень через OneSignal.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі розроблено вебзастосунок розкладу занять учасників освітнього процесу ЧНУ імені Петра Могили що реалізує децентралізовану модель формування розкладу з дев'ятьма ролями користувачів.

За результатами виконання роботи вирішено такі завдання:

– проаналізовано предметну область та три системи-аналоги (Розклад КПІ, АСУ ПДАУ, Розклад МНАУ) що використовуються в українських ЗВО; виявлено їхні ключові обмеження – відсутність децентралізованого редагування розкладу та розмежування прав доступу;

– проведено аналіз двох наукових публікацій категорії Б з питань архітектурних підходів до розробки масштабованих вебзастосунків та багаторівневого захисту даних; сформовано специфікацію вимог до програмного забезпечення що охоплює шість основних функцій системи та нефункціональні вимоги;

– спроєктовано архітектуру системи з чотирирівневою організацією Handler – UseCase – Service – ORM; побудовано ER-діаграму бази даних з чотирнадцятьма сутностями та UML-моделі ключових процесів системи; обґрунтовано вибір технологій розробки;

– реалізовано серверну частину на Django з Django-Ninja з дев'ятьма ролями користувачів на основі RBAC, JWT-автентифікацією з механізмом відкликання токенів, кешуванням через Redis та контейнеризацією через Docker;

– реалізовано децентралізовану модель формування розкладу де старости керують розкладом власної групи, а менеджери та адміністратор здійснюють загальний контроль; розклад викладача формується динамічно на основі агрегування даних розкладів груп;

– реалізовано клієнтську частину на React з TypeScript з адаптивним інтерфейсом для трьох типів пристроїв, фільтрацією за групою, підгрупою, викладачем та типом тижня, а також окремими режимами перегляду та редагування для різних категорій користувачів;

– проведено модульне тестування серверної частини що охопило 401 тест-кейс з покриттям бізнес-логіки 88–100%;

– розроблено мобільний застосунок для платформ iOS та Android на основі технології WebView з підтримкою push-сповіщень про зміни у розкладі через сервіс OneSignal.

Практичне значення роботи полягає у створенні та розгортанні вебзастосунку що знаходиться у відкритому доступі в мережі Інтернет і автоматизує процеси формування, редагування та перегляду розкладу занять учасників освітнього процесу ЧНУ імені Петра Могили (додаток Б). Децентралізована модель управління розкладом скорочує адміністративний цикл внесення змін та знижує навантаження на адміністративний персонал університету. Розроблений мобільний застосунок на основі WebView з підтримкою push-сповіщень забезпечує зручний доступ до розкладу з мобільних пристроїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Розклад КПІ. URL: https://schedule.kpi.ua (дата звернення: 25.05.2026)

2. Розклад занять АСУ ПДАУ. URL: https://asu.pdau.edu.ua/schedule (дата звернення: 25.05.2026)

3. Розклад занять Миколаївського НАУ. URL: https://rozklad.mnau.edu.ua (дата звернення: 25.05.2026)

4. Скляренко О., Савченко Я., Литвиненко Л., Сушинський О. Архітектурні підходи до розробки масштабованих веб-застосунків. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2024. Том 4, № 24. 341–350 с. URL: https://doi.org/10.28925/2663-4023.2024.24.341350 (дата звернення: 26.05.2026)

5. Белоус Р., Клименков О. Багаторівневий захист даних у Laravel-додатках. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2026. Том 4, № 32. 366–375 с. URL: https://doi.org/10.28925/2663-4023.2026.32.1075 (дата звернення: 26.05.2026)

6. Django Documentation. URL: https://docs.djangoproject.com/en/6.0 (Accessed: 02.05.2026)

7. Django Ninja – Fast Django REST Framework. URL: https://django-ninja.dev (Accessed: 02.05.2026)

8. React Documentation. URL: https://react.dev/learn (Accessed: 02.05.2026)

9. Vite Documentation. URL: https://vite.dev/guide/ (Accessed: 02.05.2026)

10. BuildNatively – Convert your website to mobile app. URL: https://buildnatively.com (Accessed: 09.06.2026)

11. BuildNatively Documentation. Push Notifications – OneSignal. URL: https://docs.buildnatively.com/guides/integration/push-notifications-onesignal (Accessed: 09.06.2026)

12. OneSignal Documentation. URL: https://documentation.onesignal.com/docs/en/home (Accessed: 09.06.2026)

ДОДАТОК А

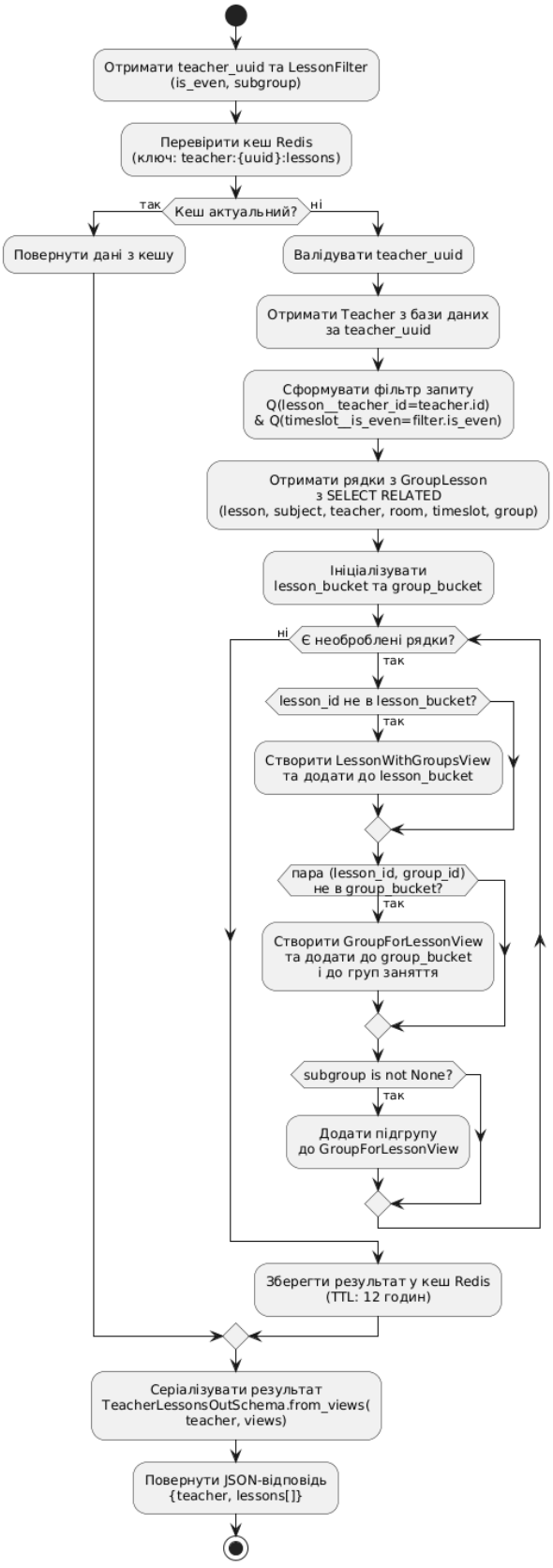


Рисунок А.1 – Діаграма діяльності алгоритму динамічного формування розкладу викладача

ДОДАТОК Б

Вебзастосунок розкладу занять учасників освітнього процесу ЧНУ імені Петра Могили розгорнуто та доступно за такими посиланнями:

Вебзастосунок (тимчасова URL-адреса): https://chmnu-schedule.mooo.com

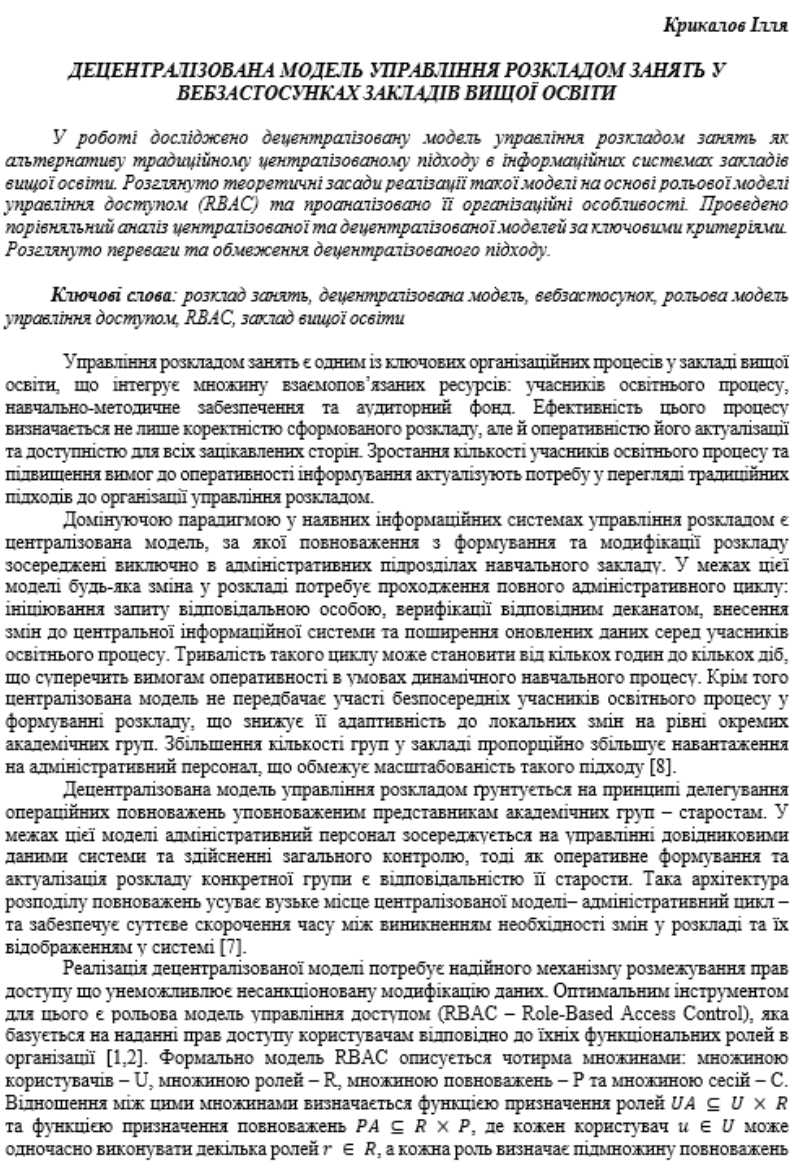
Репозиторій серверної частини (GitHub): https://github.com/uzineck/chmnu_schedule_app

Репозиторій клієнтської частини (GitHub): https://github.com/uzineck/chmnu_schedule_app_frontend

Мобільний застосунок (Google Play Store) (тимчасова URL-адреса): https://play.google.com/store/apps/details?id=com.chnnuschedule.app

ДОДАТОК В

Крикалов І. В., Давиденко Є. О. Децентралізована модель управління розкладом занять у вебзастосунках закладів вищої освіти. Ольвійський форум – 2026 : стратегії країн Причорноморського регіону в геополітичному просторі : XXII Міжнар. наук. конф. 29 черв. – 4 лип. 2026 р., м. Миколаїв : тези / ЧНУ ім. Петра Могили. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2026. С. xx-xx. URL: https://



Крикалов Ілля

ДЕЦЕНТРАЛІЗОВАНА МОДЕЛЬ УПРАВЛІННЯ РОЗКЛАДОМ ЗАНЯТЬ У ВЕБЗАСТОСУНКАХ ЗАКЛАДІВ ВИЩОЇ ОСВІТИ

У роботі досліджено децентралізовану модель управління розкладом занять як альтернативу традиційному централізованому підходу в інформаційних системах закладів вищої освіти. Розглянуто теоретичні засади реалізації такої моделі на основі рольової моделі управління доступом (RBAC) та проаналізовано її організаційні особливості. Проведено порівняльний аналіз централізованої та децентралізованої моделей за ключовими критеріями. Розглянуто переваги та обмеження децентралізованого підходу.

***Ключові слова**: розклад занять, децентралізована модель, вебзастосунок, рольова модель управління доступом, RBAC, заклад вищої освіти*

Управління розкладом занять є одним із ключових організаційних процесів у закладі вищої освіти, що інтегрує множину взаємопов'язаних ресурсів: учасників освітнього процесу, навчально-методичне забезпечення та аудиторний фонд. Ефективність цього процесу визначається не лише коректністю сформованого розкладу, але й оперативністю його актуалізації та доступністю для всіх зацікавлених сторін. Зростання кількості учасників освітнього процесу та підвищення вимог до оперативності інформування актуалізують потребу у перегляді традиційних підходів до організації управління розкладом.

Домінуючою парадигмою у наявних інформаційних системах управління розкладом є централізована модель, за якої повноваження з формування та модифікації розкладу зосереджені виключно в адміністративних підрозділах навчального закладу. У межах цієї моделі будь-яка зміна у розкладі потребує проходження повного адміністративного циклу: ініціювання запиту відповідальною особою, верифікації відповідним деканатом, внесення змін до центральної інформаційної системи та поширення оновлених даних серед учасників освітнього процесу. Тривалість такого циклу може становити від кількох годин до кількох діб, що суперечить вимогам оперативності в умовах динамічного навчального процесу. Крім того централізована модель не передбачає участі безпосередніх учасників освітнього процесу у формуванні розкладу, що знижує її адаптивність до локальних змін на рівні окремих академічних груп. Збільшення кількості груп у закладі пропорційно збільшує навантаження на адміністративний персонал, що обмежує масштабованість такого підходу [8].

Децентралізована модель управління розкладом грунтується на принципі делегування операційних повноважень уповноваженим представникам академічних груп – старостам. У межах цієї моделі адміністративний персонал зосереджується на управлінні довідниковими даними системи та здійсненні загального контролю, тоді як оперативне формування та актуалізація розкладу конкретної групи є відповідальністю її старости. Така архітектура розподілу повноважень усуває вузьке місце централізованої моделі– адміністративний цикл – та забезпечує суттєве скорочення часу між виникненням необхідності змін у розкладі та їх відображенням у системі [7].

Реалізація децентралізованої моделі потребує надійного механізму розмежування прав доступу що унеможливлює несанкціоновану модифікацію даних. Оптимальним інструментом для цього є рольова модель управління доступом (RBAC – Role-Based Access Control), яка базується на наданні прав доступу користувачам відповідно до їхніх функціональних ролей в організації [1,2]. Формально модель RBAC описується чотирма множинами: множиною користувачів – U, множиною ролей – R, множиною повноважень – P та множиною сесій – C. Відношення між цими множинами визначається функцією призначення ролей $UA \subseteq U \times R$ та функцією призначення повноважень $PA \subseteq R \times P$, де кожен користувач $u \in U$ може одночасно виконувати декілька ролей $r \in R$, а кожна роль визначає підмножину повноважень

Рисунок В.1 – Перша сторінка тези доповідей “ Децентралізована модель управління розкладом занять у вебзастосунках закладів вищої освіти ”