Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 202__ р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності 122 «Комп’ютерні науки»

на тему: «Кросплатформений застосунок «е-Студент»»

Студентка групи КН-41	Аліна ЛИТВИНЧУК	_____ (підпис)
Керівник роботи	Надія ГАВРИШКІВ	_____ (підпис)
Консультанти: з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Оксана СИРОТЮК	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 202_ р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студентці Литвинчук Аліні Петрівні

(прізвище, ім’я та по-батькові студента)

1. Тема роботи Кросплатформений застосунок «е-Студент»

затверджена наказом по коледжу від “25” листопада 2024 р., №253а-н

2. Термін здачі студентом завершеної роботи “__” червня 2025 р.

3. Вихідні дані до роботи аналіз існуючих рішень для автоматизації навчального процесу

4. Перелік питань, які повинні бути розроблені:

а) основна частина: аналіз предметної області та постановка завдань; проєктування системи; реалізація та тестування системи

б) техніко-економічне: обґрунтування аналіз ринку збутку продукту; розрахункова частина; обґрунтування доцільності створення системи

5. Перелік графічного матеріалу: UML-діаграма варіантів використання, структурна схема взаємодії модулів, логічна модель бази даних, діаграма послідовності авторизації, діаграма роботи сервера

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
З техніко-економічного обґрунтування	Любов МЕЛЕНЧУК		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1	Вибір теми кваліфікаційної роботи	20.11.2024	24.11.2024
2	Аналіз програмних рішень	30.11.2024	18.12.2024
3	Вивчення технологій реалізації роботи	19.12.2024	30.12.2024
4	Робота над структурою програмного продукту, розробка функціональних вимог.	02.01.2025	28.01.2025
5	Встановлення та налаштування середовища реалізації	29.01.2025	31.01.2025
6	Проєктування програмного засобу (функціоналу, інтерфейсу)	02.02.2025	18.02.2025
7	Реалізація та налаштування програмного засобу	19.02.2025	28.05.2025
8	Тестування та налагодження програмного продукту	10.05.2025	30.05.2025
9	Опрацювання економічного розділу кваліфікаційної роботи	19.05.2025	28.05.2025
10	Оформлення пояснювальної записки.	02.06.2025	12.06.2025
11	Попередній захист кваліфікаційної роботи	13.06.2025	13.06.2025
12	Підготовка до захисту кваліфікаційної роботи	14.06.2025	23.06.2025
13	Захист кваліфікаційної роботи	24.06.2025	

Дата видачі “__” _____ 202_ р. Керівник _____/ Гавришків Н.Г.

Завдання прийняв до виконання _____/ Литвинчук А.

Реферат

Кваліфікаційна робота. Кросплатформений застосунок «е-Студент». Литвинчук Аліна Петрівна. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. 93 сторінки, рисунків – 38, додатків – 22, джерел – 14.

Об’єкт дослідження – системи автоматизації навчального процесу.

Метою роботи є створення функціональної системи автоматизації процесів навчання в Галицькому фаховому коледжі такими засобами: серверна частина з використанням мови C++ та фреймворку Pistache та кросплатформна розробка клієнту Flutter з використанням мови програмування Dart, середовищем розробки Visual Studio Code та реляційною базою даних MariaDB.

Кросплатформений застосунок «е-Студент» має забезпечити перегляд студентами особистого розкладу занять, оцінок і відвідуваності, а також отримання сповіщень про події та зміни в навчальному процесі. Викладачі повинні мати змогу фіксувати присутність студентів на заняттях і надсилати повідомлення. Методисти, у свою чергу, повинні формувати звіти по відвідуваності та здійснювати розсилку сповіщень для викладачів і студентів свого відділення. Крім функціональної частини, необхідно створити інтерфейс, адаптований до мобільних пристроїв на базі Android та iOS, з урахуванням вимог зручності, доступності та сучасних принципів дизайну.

Результатом розробки є кросплатформений застосунок «е-Студент», який повністю готовий до використання у Галицькому фаховому коледжі.

КРОПЛАТФОРМЕНИЙ ЗАСТОСУНОК, API, FLUTTER, JWT, C++, PISTACHE, MARIADB, JSON, КЛІЄНТ-СЕРВЕР.

Abstract

Qualification work. Cross-platform application «e-Student». Lutvynchuk Alina Petrivna. Halitskyi Vocational College named after Vyacheslav Chornovil, Department of Computer Technologies. 93 pages, 38 figures, 22 appendices, 14 references.

Object of research – systems for automating the educational process.

The aim of the work is to develop a functional system for automating educational processes at the Halitskyi Vocational College using the following technologies: a server-side component built with the C++ language and the Pistache framework, and a cross-platform client developed with Flutter using the Dart programming language, the Visual Studio Code development environment, and the MariaDB relational database.

The cross-platform application "e-Student" is designed to provide students with access to their personal class schedules, grades, and attendance records, as well as notifications about events and changes in the academic schedule. Teachers should be able to record student attendance and send messages. Methodologists, in turn, should be able to generate attendance reports and send announcements to teachers and students within their department. In addition to the functional part, it is necessary to design a user interface adapted for mobile devices running Android and iOS, considering usability requirements, accessibility, and modern design principles.

As a result of the development, a cross-platform application "e-Student" was created, which is fully ready for implementation at the Halitskyi Vocational College.

CROSS-PLATFORM APPLICATION, API, FLUTTER, JWT, C++, PISTACHE, MARIADB, JSON, CLIENT-SERVER.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЄСВ – Єдиний соціальний внесок

ЗП – Заробітна плата

ЗУНУ – Західноукраїнський національний університет

ОС – Операційна система

ПДФО – Податок на доходи фізичних осіб

ПК – Персональний комп’ютер

СУБД – Система управління базами даних

API – Application Programming Interface

FCM – Firebase Cloud Messaging

FK – Foreign Key

GPLv3 – GNU General Public License Version 3

HTTP – Hypertext Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

JWT – JSON Web Token

MIME – Multipurpose Internet Mail Extensions

PK – Primary Key

RAII – Resource Acquisition Is Initialization

REST API – Representational State Transfer Application Programming Interface

UI – User Interface

UML – Use-case diagram

UUID – Universally Unique Identifier

ВСТУП

В теперішній час цифрові технології мають значний вплив на організацію освітнього процесу. Швидкий розвиток інформаційних систем, мобільних пристроїв та інтернет-сервісів створює нові умови для покращення методів комунікації та взаємодії між усіма учасниками освітнього середовища – студентами, викладачами та адміністрацією навчальних закладів. Водночас традиційні форми управління навчальним процесом залишаються громіздкими та малоефективними, особливо в умовах дистанційного або гібридного навчання.

Сьогодні здобувачі освіти очікують оперативного доступу до розкладу, оцінок, сповіщень та навчальних ресурсів без необхідності звертатися до паперових носіїв або окремих кабінетів. Викладачам, своєю чергою, необхідні інструменти для швидкого обліку відвідуваності, оцінювання та звітності, а адміністрації – централізоване бачення освітнього процесу. Відсутність єдиного інтерактивного рішення створює інформаційні затримки та складності у комунікації.

Більшість існуючих рішень не є адаптованими до специфіки українських вишів або мають обмежену функціональність, не охоплюючи всіх потреб навчального закладу. Саме тому актуальною є розробка кросплатформеного застосунку, що забезпечить цілісну взаємодію між учасниками освітнього процесу та надасть необхідні інструменти у зручному форматі.

Метою цієї роботи є створення та впровадження застосунку «e-Студент», який дозволяє студентам переглядати розклад, оцінки, відвідуваність і сповіщення, а також забезпечує викладачам і адміністрації зручні засоби обліку та звітності. Реалізація цього проєкту сприятиме цифровізації освіти, підвищенню прозорості навчального процесу та стане поштовхом створення єдиного інформаційного простору в межах навчального закладу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВАДАНЯ

1.1 Опис предметної області

В сучасному світі де процвітає студентоцентроване навчання існує потреба створення платформи для автоматизації та полегшення навчального процесу для всіх його учасників, а особливо здобувачів освіти. Навчальний процес у закладах фахової передвищої освіти є складною системою взаємодії між студентами, викладачами та адміністрацією, що потребує ефективної координації і автоматизації обміну інформацією.

Освітні заклади стикаються з багатьма викликами в організації навчального процесу, наприклад оперативне інформування про зміни в розкладі занять, ефективний облік відвідуваності студентів та інші проблем які потребують вирішення.

Кросплатформений застосунок «е-Студент» буде використовуватися в Галицькому фаховому коледжі імені В’ячеслава Чорновола та покликаний допомогти організувати навчальний процес зручніше для всіх його учасників. Система враховуватиме організаційні особливості навчання у фахових коледжах, а також потреби різних видів користувачів.

Впровадження такого застосунку дозволить створити єдине інформаційне середовище для навчального закладу та буде сприяти підвищенню якості освітньої діяльності. Це особливо важливо в контексті діджиталізації багатьох життєвих процесів та переходу до більш гнучких форм організації навчального процесу.

1.2 Аналіз наявних рішень

Сьогодні спостерігається тенденція до створення численних застосунків, спрямованих на автоматизацію повсякденних завдань сучасного життя. Наприклад, застосунок «е-Тернопіль», розроблений для жителів Тернополя, дозволяє зручно перевіряти баланс карти тернополянина, переглядати розклад

громадського транспорту, дізнаватися про наявність електропостачання, спілкуватися з місцевою владою та користуватися багатьма іншими сервісами. Основна перевага полягає в об’єднанні всього необхідного функціоналу в одному зручному інтерфейсі.

При дослідженні аналогів варто звертати увагу на різні критерії, такі як інтерфейс, функціонал та безпека, що є ключовими критеріями при аналізі застосунків.

Застосунок «е-Тернопіль» має версію як для мобільних пристроїв, так і для ПК. Проте комп'ютерна версія це лише сайт, який не забезпечує повного функціоналу мобільної версії. Натомість веб-версія надає доступ лише до порталу мешканця, де користувачі мають змогу отримувати інформацію про комунальні послуги, переглядати сповіщення та подавати заявки на різні види послуг. Повний спектр можливостей доступний лише у мобільному застосунку.

У застосунку приємний та інтуїтивно зрозумілий дизайн, присутня можливість увімкнути темну тему. Головна сторінка дає можливість налаштувати свою стрічку сервісів у верхній частині екрану. Також відображаються новини області (рис. 1.1).

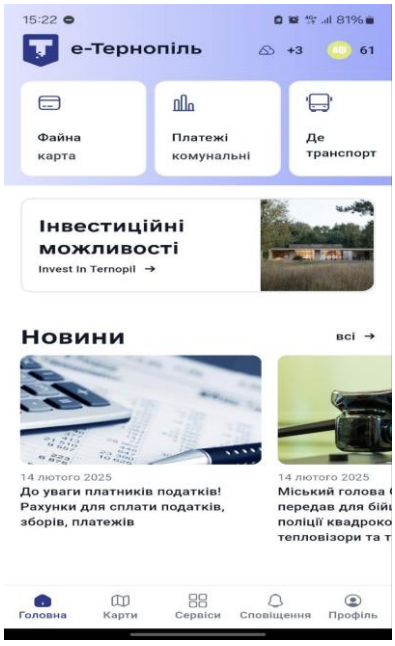

Рисунок 1.1 – Головна сторінка «е-Тернопіль»

В застосунку є окремі сторінки з інтерактивними картами, де можна переглянути важливу інформацію про міську інфраструктуру (рис. 1.2). Зокрема, можна дізнатися де є електропостачання, де розташовані точки з питною водою, укриття, медичні заклади, пункти незламності тощо. Карти оновлюються в режимі реального часу, що дозволяє мешканцям швидко отримувати актуальну інформацію.

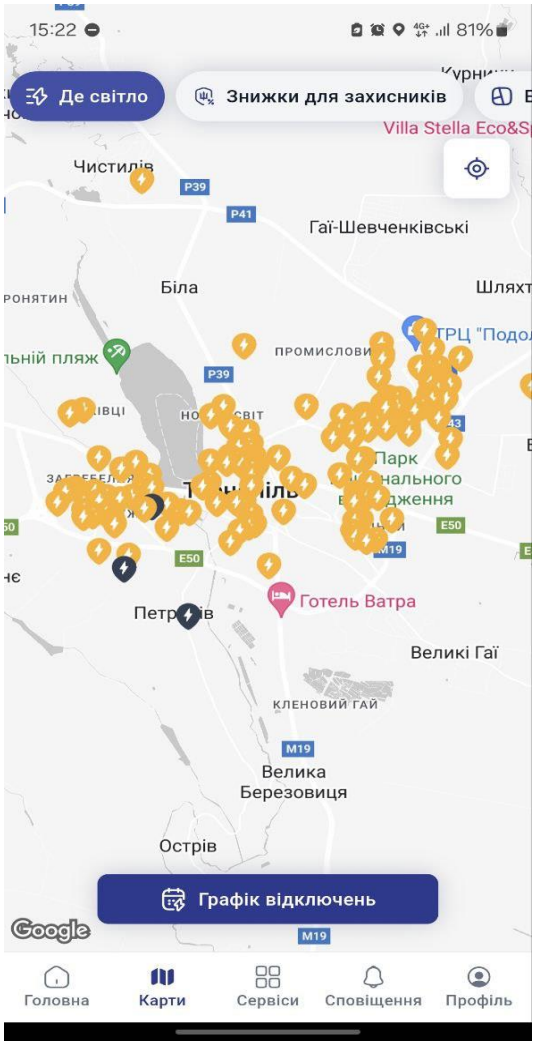

Рисунок 1.2 – Сторінка з картами

Застосунок має в собі великий функціонал, який постійно розширюється та доповнюється новими опціями. Для зручної навігації присутня сторінка де перелічено усі доступні сервіси. Всі вони структуровані за спільними категоріями, що дозволяє швидко знаходити потрібні функції (рис. 1.3).

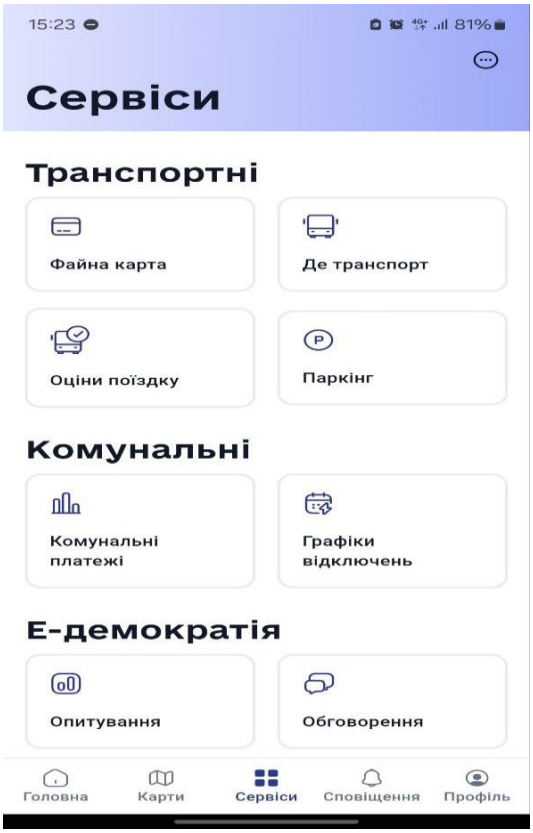

Рисунок 1.3 – Сторінка сервісів

Також, що зручно, в застосунку можна переглянути сповіщення що були отримані за час використання програми та відфільтрувати їх на «всі», «загальні» та «персональні».

Застосунок використовує сучасні безпекові заходи для того, щоб забезпечити максимальний рівень безпеки громадян особливо в період війни, проте інформації про конкретні рішення не було знайдено в відкритих джерелах.

Загалом проаналізувавши цей застосунок можна сказати що це є хорошим рішенням для автоматизації багатьох процесів, та зручною альтернативою застарілим способам отримання міських послуг.

Застосунок «Мрія» це нова розробка міністерства освіти для загальноосвітніх шкіл України який надає можливість цифрового обліку учнів, перегляду особистого розкладу і успішності для школярів, електронний журнал і домашні завдання для викладачів та можливість слідкувати за

прогресом своєї дитини, бачити її розклад та можливість спілкування з вчителями у приватних чатах. Також для всіх є доступ до бібліотеки контенту з різноманітними навчальними відео для саморозвитку дітей та педагогів.

Застосунок має схожий задум до «е-Студент», але розроблений він з розрахунком на використання лише в школах, а не у вищих та передвищих навчальних закладах, що є значним мінусом платформи.

«Мрія» має сучасний дизайн із цікавими анімаціями, інтуїтивну систему вкладок та вичерпний функціонал.

Нижче було проаналізовано функції для учнів та вчителів.

На першій сторінці для дітей відображається особистий розклад та домашні завдання (рис. 1.4).

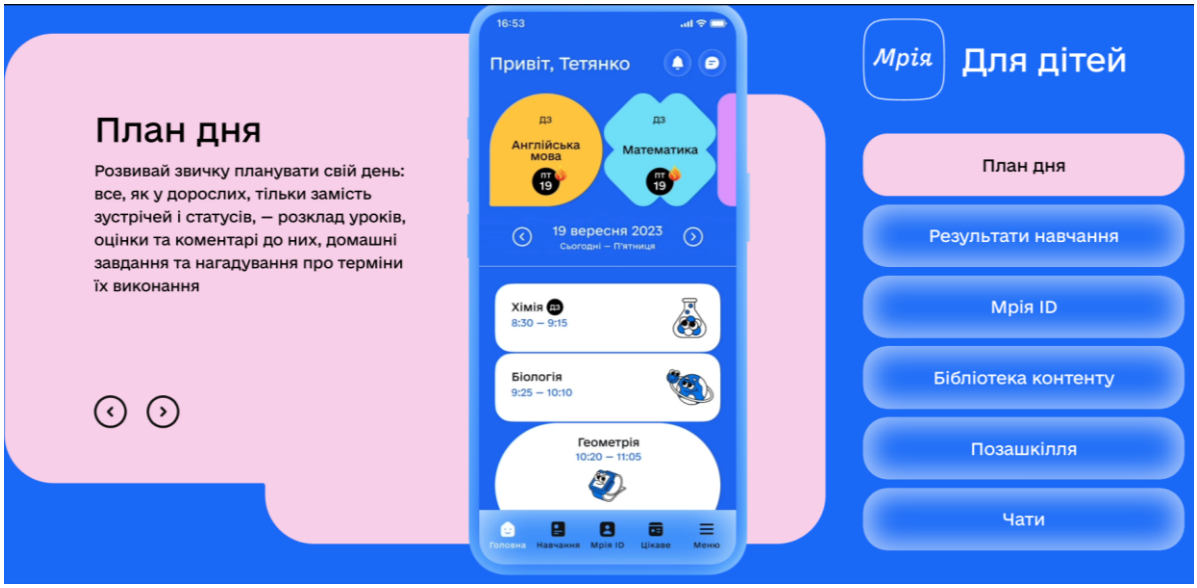

Рисунок 1.4 – План дня

Дизайн використовує яскраві кольори, що дозволяє візуально виокремити певні блоки застосунку. Інтерфейс інтуїтивно зрозумілий, проте дещо перевантажений, що може заплутати дітей молодшого віку та стати мінусом. Також блок розкладу містить обмежену інформацію про урок.

Сторінка «Результати навчання» дає можливість зручного перегляду статистики та моніторингу освітнього процесу за різними категоріями як «Результати», «Завдання» та «Розклад» (рис. 1.5).

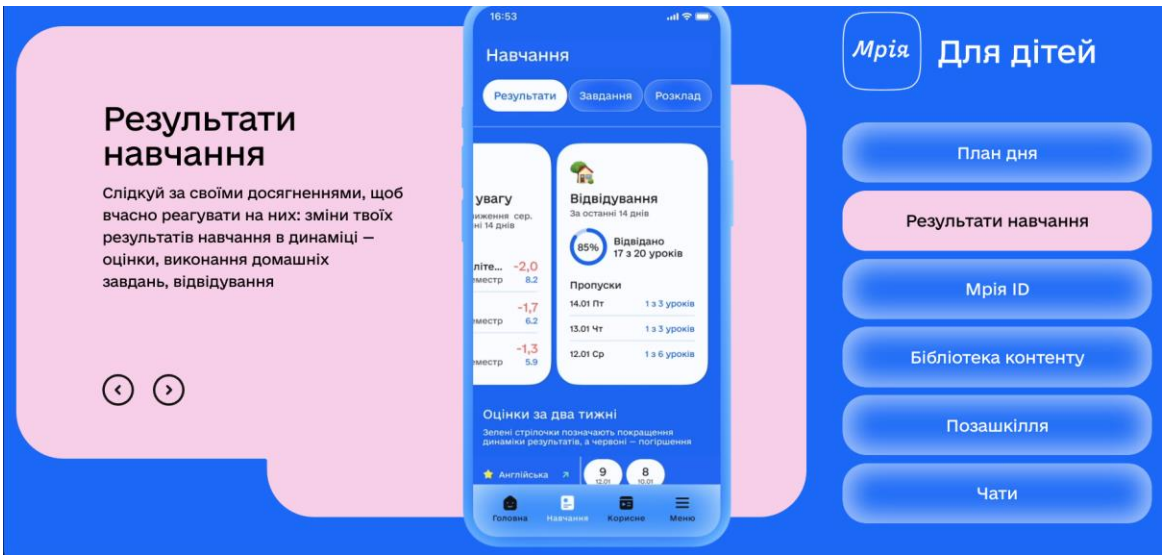

Рисунок 1.5 – Результати навчання

Ця сторінка надає детальну інформацію про успішність учня в структурованій формі. Наявна інформація про оцінки, середні бали із різних предметів та відображається інформація про те чи зменшилися вони, чи ні за останні два тижні. Проте, як і на попередній сторінці, інтерфейс перевантажений.

Для викладачів є можливість провести облік відвідуваності учнів на уроках та виставити оцінки (рис. 1.6).

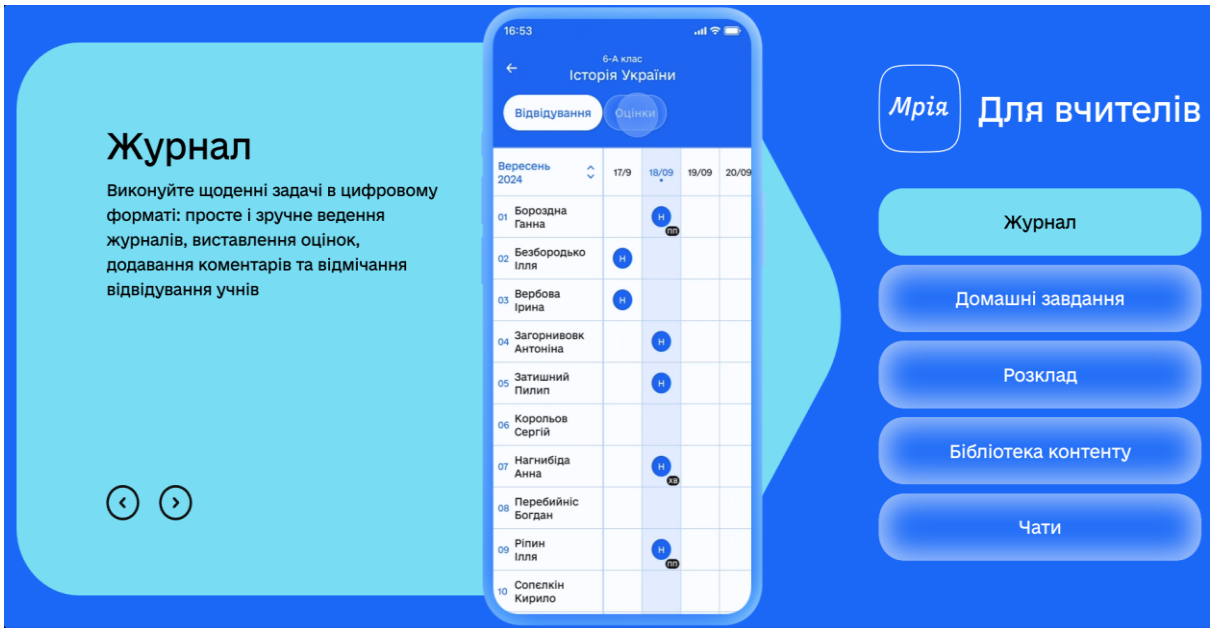

Рисунок 1.6 – Журнал для вчителів

На сторінці є зручний вибір з усіх класів, яким викладає певний вчитель, і зручно розділено відвідуваність та оцінки, можна позначити чи був пропущений урок за поважною, чи неповажною причиною. Інтерфейс та розташування елементів зручне, проте якщо клас великий доведеться багато прокручувати вниз.

«Мрія» також має веб версію із розширеним функціоналом для викладачів, що є хорошим доповненням для мобільного застосунку та дає можливість зручніше виконувати дії з розкладом та заповненням журналу.

На сайті застосунку вказано що «Мрія» не зберігає особисті дані на своїх серверах та не передає їх третім особам [1]. Застосунок використовує сучасні стандарти для шифрування персональних даних користувачів, та після їх передачі із захищеного державного реєстру зберігаються безпосередньо на особистому пристрої. Такі заходи безпеки є важливими для платформи що оперує великими масивами приватної інформації про викладачів та дітей.

Найближчим аналогом до створюваного застосунку є «Smart Університет» [2] який запроваджено в ЗУНУ. Було оглянуто функціонал, що представлено на сайті платформи, тому що для реєстрації акаунту необхідно бути студентом університету.

На сайті не надано інформації про повний функціонал кабінету студента, проте із наявної можна виділити, що ключовими аспектами є перегляд персоналізованого списку подій та розкладу занять, свої дисципліни, звіти успішності студента та комунікація із викладачами чи іншими здобувачами освіти (рис. 1.7).

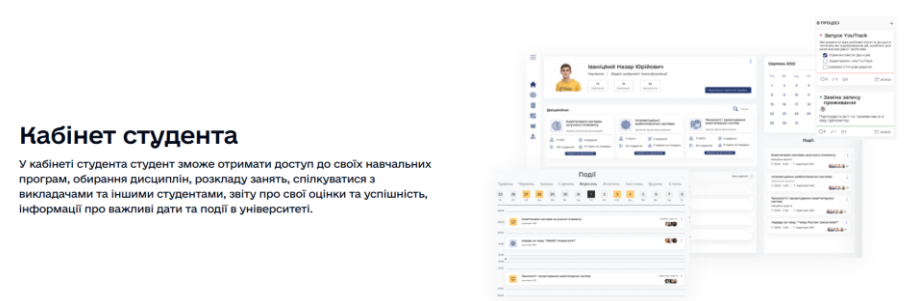

Рисунок 1.7 – Кабінет студента

Застосунок має приємний та сучасний дизайн виконаний у мінімалістичному стилі, оформлений у кольорах навчального закладу. Вся інформація слушно структурована. Зручним є календар на головному екрані, що надає легкий доступ до розкладу подій, навіть на далекі дати.

Для того, щоб користувачам було легше ознайомитися із повним функціоналом застосунку, на сайті можна знайти документацію, яка описує головні аспекти користування платформою.

Застосунок є кросплатформним, що є важливим аспектом зручності користування, забезпечуючи стабільну роботу на різних операційних системах та пристроях, включаючи комп'ютери, планшети та смартфони.

На жаль, у відкритих джерелах не було знайдено детальної інформації про конкретні заходи безпеки в застосунку «SMART Університет». Однак, враховуючи, що університет має кафедру кібербезпеки, можна зробити припущення, що розробка та підтримка платформи здійснюється з урахуванням сучасних стандартів інформаційної безпеки.

Під час проведення аналізу наявних рішень було виявлено позитивні та негативні аспекти, які стануть корисними при проєктуванні власного застосунку.

1.3 Постановка завдання

Після проведеного аналізу предметної області та наявних рішень було виділено основні вимоги до кінцевого продукту. Для того, щоб чітко окреслити межі проєкту та спланувати подальшу розробку необхідно визначити основні завдання та проблеми які вона має вирішувати.

Головним завданням, яке має вирішити створюваний застосунок – є спрощення організації навчального процесу для всіх його учасників. Застосунок має являти собою комплексну систему функціональних можливостей для викладачів, здобувачів освіти та адміністрації.

Застосунок має забезпечити перелік функціональних вимог:

– можливість зручного перегляду розкладу та внесення змін в нього;

– облік відвідуваності студентів;

– перегляд оцінок з практик, курсових проєктів та заліків;

– отримання сповіщень від адміністрації та викладачів.

Крім функціональних вимог варто також виділити нефункціональні:

– адаптивний та сучасний дизайн;

– інтуїтивно зрозумілий інтерфейс;

– кросплатформеність;

– наявність темної кольорової теми;

– швидкий відгук системи;

– безпечне збереження даних та їх шифрування при зберіганні;

– авторизація користувачів за різними рівнями доступу.

Система управління організацією навчання покликана об’єднати ключові функції освітнього процесу в єдиному зручному інтерфейсі, забезпечуючи ефективну взаємодію між студентами, викладачами та адміністрацією. Основна увага приділяється прозорості навчання, оперативності обміну інформацією та безпеці даних.

У результаті проведеного аналізу предметної області було визначено ключові проблеми, які постають перед учасниками освітнього процесу у фахових передвищих закладах освіти, а також сформульовано завдання, що має вирішити розроблювана система. На основі розглянутих аналогів було виявлено найбільш вдалі підходи до реалізації інтерфейсу, функціональності та захисту даних, що стануть у нагоді при створенні власного застосунку. Проведене дослідження дозволило чітко окреслити вимоги до майбутнього продукту як з функціонального, так і з нефункціонального боку. Визначено необхідність створення комплексної, зручної та безпечної системи, яка дозволить автоматизувати ключові процеси навчання та покращити взаємодію між студентами, викладачами і адміністрацією.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування архітектури системи «е-Студент»

Архітектура застосунку являє собою високорівневу структуру системи, яка описує її основні компоненти, взаємодію між ними, ключові можливості та принципи функціонування. Правильне проєктування дозволить створити маштабовану, надійну та просту для супроводу систему.

Для того щоб спроєктувати застосунок зручний для використання кінцевому користувачу необхідно проаналізувати варіанти використання майбутнього продукту та визначити типи користувачів (рис. 2.1).

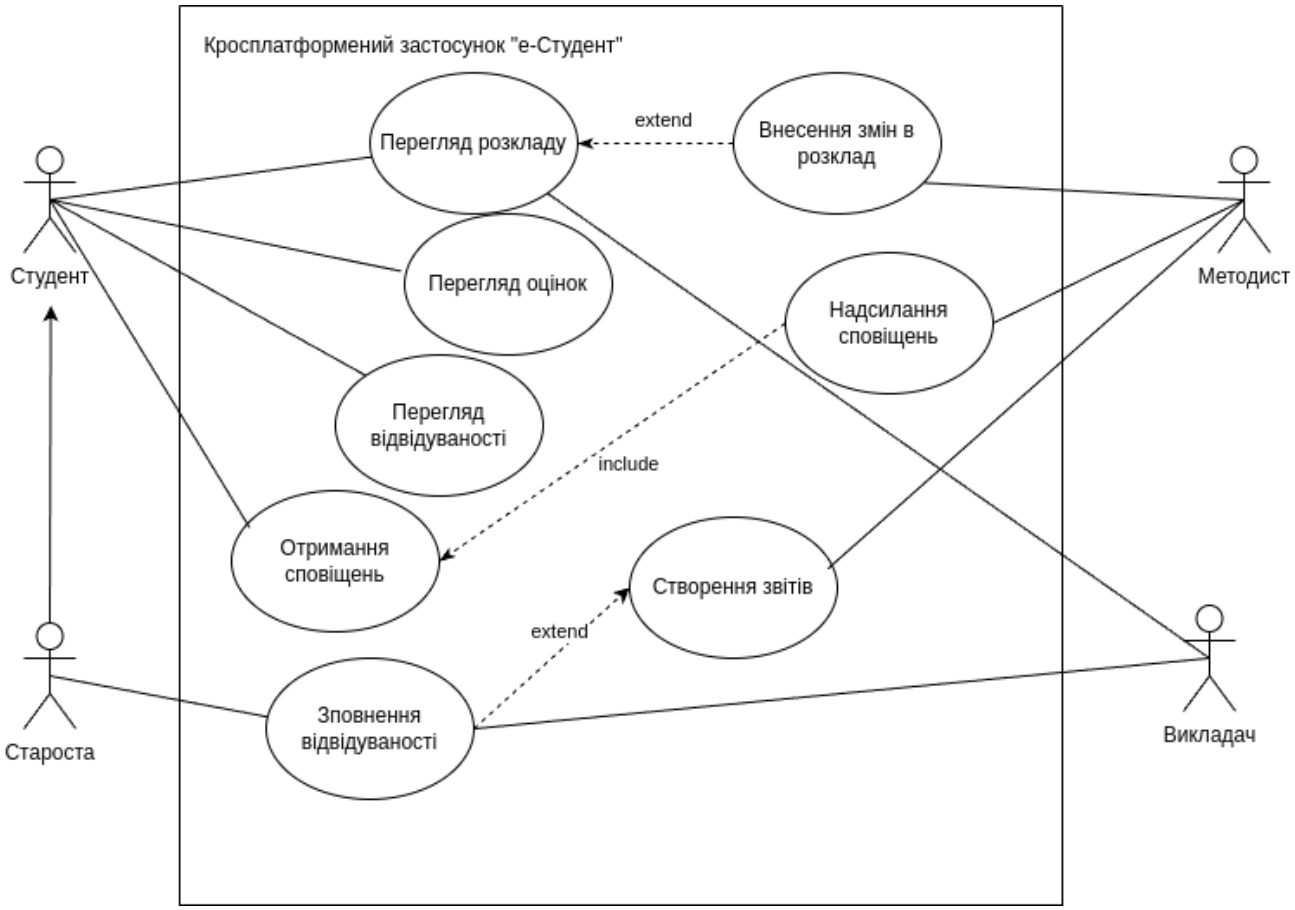

Рисунок 2.1 – Діаграма варіантів використання

В системі присутні 4 категорії користувачів: студент, староста, викладач та методист. Кожен з видів має свої права доступу та функціональні можливості в застосунку, що представлено в таблиці 2.1.

Таблиця 2.1 – Можливості різних видів користувачів в системі

Категорія користувачів	Можливості в системі
Студент	Має можливість переглядати свою особисту інформацію, персональний розклад занять, оцінки та відвідуваність, отримувати сповіщення про події та зміни в розкладі від адміністрації та методиста. Може переглядати історії сповіщень.
Староста	Має ті ж можливості що студент. Може заповнювати відвідуваність студентів та формувати по ній звіти.
Викладач	Може заповнювати присутність на своїх парах, переглядати персоналізований розклад та надсилати повідомлення студентам.
Методист	Може створювати звіти відвідуваності груп свого відділення, надсилати сповіщення про зміни в розкладі та події студентам і викладачам.

Для реалізації застосунку обрана клієнт-серверна архітектура, яка дозволить розділити навантаження між різними компонентами системи, та дасть можливість централізовано керувати даними та логікою застосунку.

Серверна частина системи реалізовуватиметься на основі архітектури REST API, що дозволить забезпечити стандартизований та ефективний спосіб обміну між клієнтом та віддаленим сервером.

REST API використовує стандартний протокол HTTP [3] для передачі даних, що забезпечує широку сумісність із різноманітними клієнтськими платформами. Взаємодія відбувається через набір чітко визначених HTTP-методів (GET, POST, PUT, DELETE та інші), кожен з яких має конкретне призначення та семантику. Схему робити REST API зображено на рисунку 2.2.

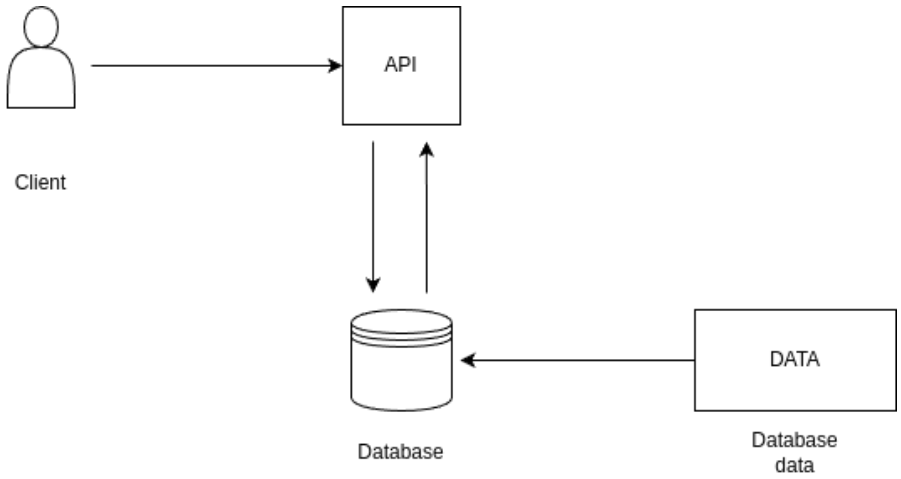

Рисунок 2.2 – Схема роботи REST API серверу

Для кращої структуризації, клієнтську частину буде розподілено за принципом модульної організації з чітким розділенням відповідальності між компонентами. На рисунку 2.3 зображено структурну схему взаємодії модулів між собою.

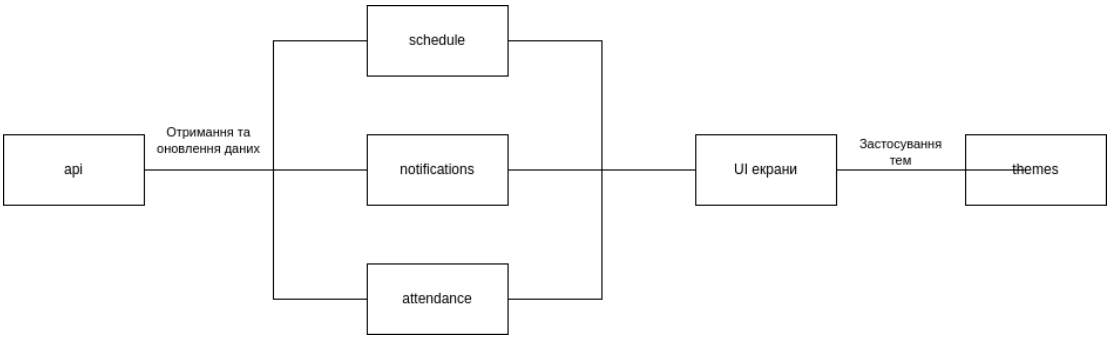

Рисунок 2.3 – Структурна схема взаємодію модулів системи

Клієнтська частина складатиметься із 6 модулів кожен із яких має свої завдання. API модуль відповідає за зв'язок із сервером, передачу та отримання даних для відображення в самому застосунку. Модулі «schedule», «notifications» та «attendance» є відповідальними за коректне відображення отриманої інформації з сервера в відповідних блоках.

Компонент «themes» реалізує можливість вибору світлої або темної теми інтерфейсу. Модуль «UI екрани» об'єднує всі користувацькі інтерфейси та забезпечує єдиний стиль і логіку навігації між різними частинами застосунку.

Така модульна архітектура забезпечує чітке розділення відповідальності між компонентами системи, що спрощує розробку, тестування та подальше масштабування застосунку.

2.2 Вибір засобів реалізації

Для реалізації проєкту було обрано комплекс сучасних технологій, які можуть забезпечити якісну та функціональну систему в рамках клієнт-серверної архітектури.

Для клієнтської сторони обрано кросплатформенний фреймворк Flutter від Google, що дає можливість створювати високопродуктивні застосунки з нативним виглядом для різних платформ як наприклад IOS та Android, а також для Windows та Linux з використанням єдиної кодової бази [4]. Flutter використовує тільки одну мову програмування Dart, що спрощує розробку програмного забезпечення.

Серверна частина застосунку буде реалізована із використанням Pistache – високопродуктивного C++ фреймворку для створення веб-серверів та REST API [5]. Цей фреймворк надає об'єктно-орієнтований інтерфейс для роботи з HTTP запитами, керування маршрутизацією та відповідями. Використання мови C++ забезпечить високу продуктивність сервера та ефективне використання ресурсів.

Для зберігання даних обрано MariaDB, що є відкритою та ефективною системою керування реляційними базами даних. Вона забезпечує високу продуктивність, масштабованість та надійність, має в собі такі можливості як транзакції, індексацію та інші механізми роботи з даними.

Для реалізації системи сповіщень у застосунку обрана технологія Firebase Cloud Messaging (FCM) від Google. Вона дозволяє надсилати push-нотифікації користувачам навіть тоді, коли додаток перебуває у фоновому режимі або повністю закритий [6]. Завдяки інтеграції з Flutter клієнт отримує унікальний FCM-токен, який реєструється на сервері та прив’язується до

конкретного користувача. Серверна частина надсилає повідомлення безпосередньо до Firebase, який доставляє їх на пристрій користувача. Такий підхід забезпечує стабільну та масштабовану систему оповіщення, що не потребує постійного з'єднання з сервером та мінімізує навантаження на інфраструктуру.

Використання такого технологічного стеку дозволить створити ефективну, функціональну та швидку систему для зручного користування. Комбінація Flutter, MariaDB та REST API на базі Pistache C++ формує збалансоване рішення, що відповідає сучасним вимогам до продуктивності та зручності розробки.

2.3 Проєктування бази даних

Проєктування бази даних – це ключовий етап розробки будь-якого застосунку, особливо такого, який працює з великими обсягами інформації та вимагає швидкої взаємодії між численними користувачами. На цьому етапі визначається логічна структура даних, типи сутностей, їх атрибути, взаємозв’язки, а також правила забезпечення цілісності та узгодженості. Якісне проєктування суттєво впливає на продуктивність, масштабованість і надійність усієї системи.

База даних застосунку «е-Студент» призначена для зберігання ключових компонентів навчального процесу: інформації про розклад занять, зміни до нього, ролі та дані користувачів (студентів, викладачів, старост, методистів), відвідуваність студентів, а також систему сповіщень. Особлива увага приділяється структурованості та розмежуванню прав доступу, що дозволяє забезпечити коректну роботу залежно від ролі користувача.

У процесі проєктування було виділено основні сутності та встановлено зв’язки між ними, що дозволяє ефективно організувати та обробляти дані. Ця структура представлена у вигляді ER-діаграми на рисунку 2.4, що відображає загальну архітектуру моделі даних системи.

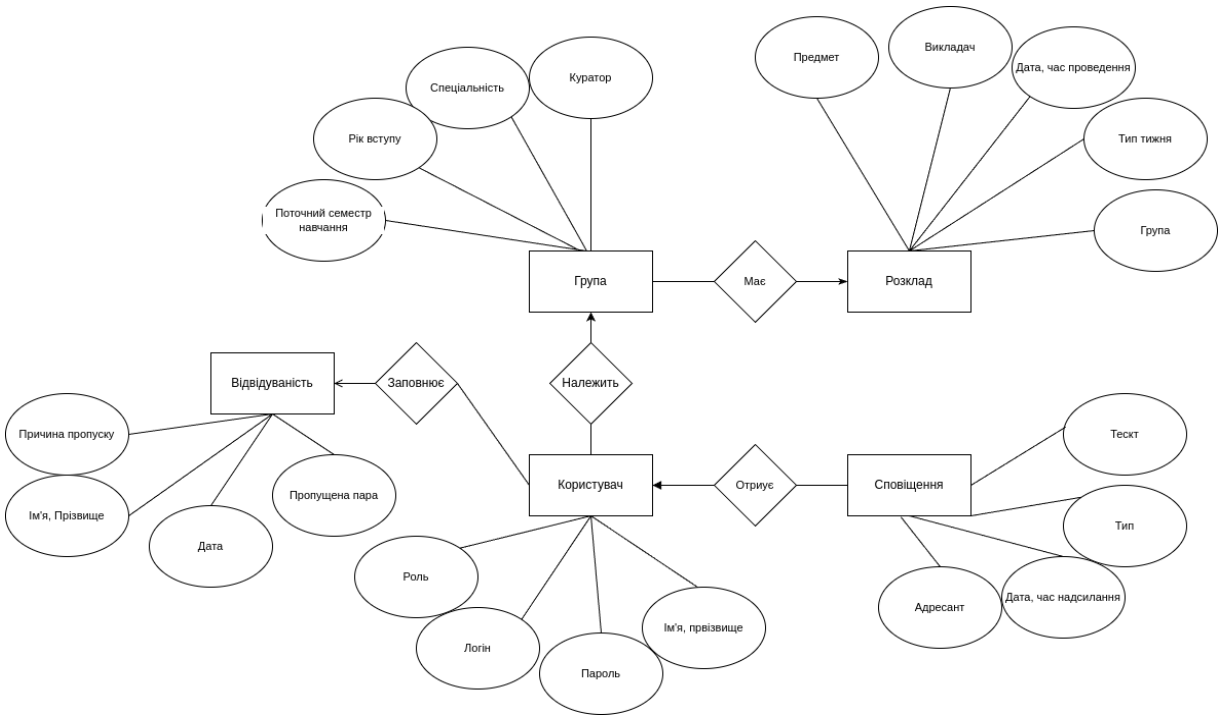

Рисунок 2.4 – ER-діаграма

Ця ER-модель представляє систему збереження інформації про необхідні навчальні процеси. Основними сутностями є користувач, група, розклад, відвідуваність та сповіщення. Користувачі належать до груп, кожна група має свій особистий розклад. Користувач може отримувати сповіщення та заповнювати відвідуваність.

В розробленій моделі присутні зв’язки багато до багатьох, які не підтримуються обраною системою керування реляційними базами даних [7], тому були виділені проміжні сутності. Для того щоб краще представити усі таблиці бази даних, їх було поділено на 6 підмодулів:

– користувачі, ролі, безпека;

– навчальні підрозділи, групи, підгрупи;

– періоди, тижні та навчальний час;

– розклад;

– відвідуваність;

– сповіщення.

Логічну модель підмодуля користувачів, ролей та безпеки зображено на рисунку 2.5.

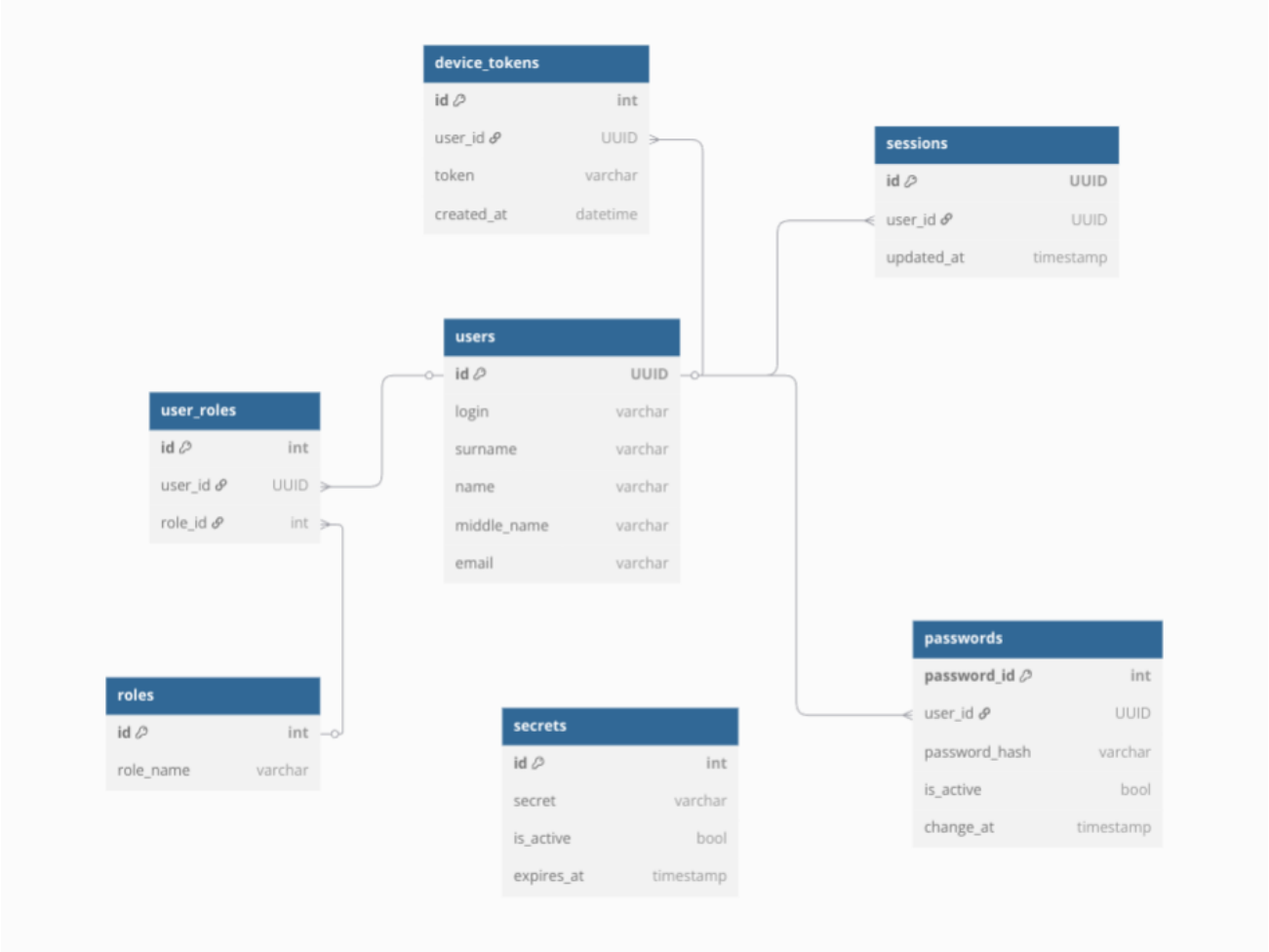

Рисунок 2.5 – Логічна модель бази даних модуля користувачів

Після визначення логічної структури, було розроблено фізичну структуру таблиць з детальним описом атрибутів. Для кожної таблиці визначено поля, їх типи даних, обмеження цілісності та зв'язки з іншими таблицями.

Таблиця «users» зберігає в собі інформацію про користувачів системи, вона містить такі атрибути як логін, ім’я, прізвище, по батькові та пошту. Для запису ID використовується тип даних UUID який є текстовим рядком, згенерованим складним алгоритмом, що гарантує його унікальність при великій кількості користувачів [8]. Також UUID є більш безпечним при використанні в публічних API, оскільки він не є послідовним і

передбачуваним, на відміну від звичайних цілочисельних ідентифікаторів, що значно ускладнює потенційні атаки на систему методом перебору. Повний опис таблиці «users» представлено в таблиці 2.2.

Таблиця 2.2 – Атрибути таблиці «users»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	user_id	Первинний	UUID	Унікальний	NOT NULL, PK	Ідентифікатор користувача
login	login	-	varchar	-	NOT NULL, Унікальний	Логін
surname	surname	-	varchar	-	NOT NULL	Прізвище
name	name	-	varchar	-	NOT NULL	Ім’я
middle_name	middle_name	-	varchar	-	NULL	По батькові
email	email	-	varchar	-	NOT NULL, Унікальний	Електронна пошта

Для зберігання типів облікових записів користувачів створено таблицю «roles», яка зберігатиме в собі категорії учасників системи та забезпечуватиме можливість легкого масштабування в випадку необхідності додавання нових функціональних груп користувачів (табл. 2.3).

Таблиця 2.3 – Атрибути таблиці «roles»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	role_id	Первинний	int	> 0	NOT NULL, PK	ID ролі
role_name	role_name	-	varchar	-	NOT NULL	Назва ролі

Для зв’язку між таблицею «roles» та «users» створено проміжну сутність «user_roles, для уникнення зв’язку багато до багатьох.

Для надійного зберігання паролів користувачів створено таблиця «passwords» (табл. 2.4).

Таблиця 2.4 – Атрибути таблиці «passwords»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
password_id	password_id	Первинний	int	-	NOT NULL, PK	ID паролю
user_id	user_id	Зовнішній	UUID	-	NOT NULL, FK	Користувач
password_hash	password_hash	-	varchar	-	NOT NULL, Унікальний	Хеш паролю
is_active	is_active	-	bool	-	NOT NULL	Чи активний пароль
change_at	change_at	-	timestamp	-	NULL	Дата зміни паролю

Ця таблиця дозволяє зберігати зашифровані паролі та дає можливість організувати зручну обробку зміни паролів користувачами.

Таблиця «sessions» використовується для зберігання інформації про активні або останні сесії входу користувачів у систему (табл. 2.5). Вона служить важливим елементом у системі аутентифікації та авторизації, дозволяючи відстежувати, який користувач увійшов у систему, зберігати унікальний ідентифікатор сесії та контролювати тривалість активності користувача, використовуючи поле «updated_at».

Таблиця 2.5 – Атрибути таблиці «sessions»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	UUID	-	NOT NULL, PK	Ідентифікатор сесії
user_id	user_id	Зовнішній	UUID	-	NOT NULL, FK	Користувач сесії
updated_at	updated_at	-	timestamp	-	NOT NULL	Час оновлення сесії

Таблиця «secrets» використовується для зберігання криптографічних ключів (секретів), які застосовуються для створення, підпису або валідації токенів авторизації в системі. Це критично важлива таблиця для безпеки

механізму аутентифікації, оскільки дозволяє керувати життєвим циклом ключів: активувати, деактивувати, встановлювати терміни дії (табл. 2.6).

Таблиця 2.6 – Атрибути таблиці «secrets»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID секрету
secret	secret	-	varchar	-	NOT NULL	Секрет
is_active	is_active	-	bool	-	NOT NULL	Чи активний
expires_at	expires_at	-	timestamp	-	NULL	Дата закінчення дії

Для того щоб зберігати push-токени пристроїв, пов’язаних з користувачами для надсилання сповіщень створено таблицю «device_tokens», що представлена в таблиці 2.7.

Таблиця 2.7 – Атрибути таблиці «device_tokens»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID токена
user_id	user_id	Зовнішній	UUID	-	NOT NULL, FK	Користувач
token	token	-	varchar	-	NOT NULL	Токен пристрою
created_at	created_at	-	datetime	-	NOT NULL	Дата створення

Цей підмодуль дозволяє реалізувати повноцінну систему управління користувачами з підвищеним рівнем безпеки та гнучкістю налаштувань. Структура забезпечує надійну ідентифікацію користувачів через використання UUID, що унеможливлює передбачувані атаки на систему.

Для збереження інформації про навчальні підрозділи, групи, підгрупи створено підмодуль з основними і проміжними таблицями для уникнення зв’язків багато до багатьох (рис. 2.6).

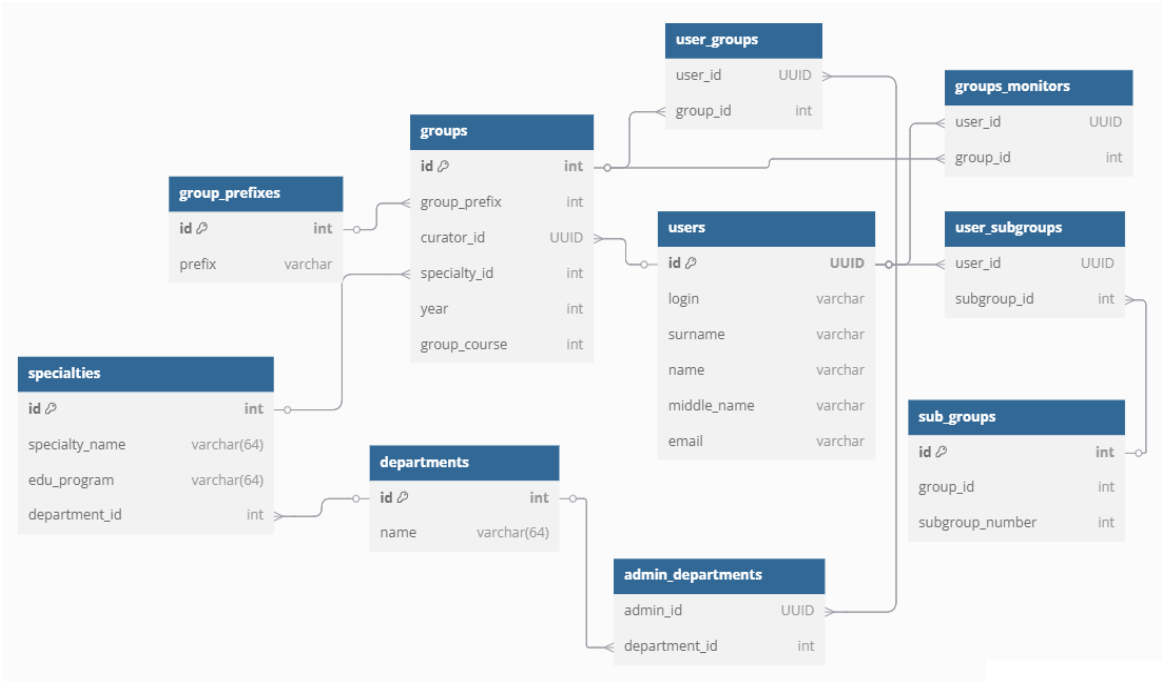

Рисунок 2.6 – Логічна модель бази даних модуля навчальні підрозділів, груп та підгруп

Сутність «groups» зберігає інформацію про групи, їх кураторів, рік вступу, спеціальність та префікс групи. Структура цієї таблиці забезпечує зберігання всіх необхідних даних про академічні групи з підтримкою цілісності даних через систему зовнішніх ключів, що зв'язують таблицю з іншими сутностями системи (табл. 2.8).

Таблиця 2.8 – Атрибути таблиці «groups»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	group_id	Первинний	int	> 0	NOT NULL, PK	ID академічної групи
group_prefix	group_prefix_id	Зовнішній	int	> 0	NOT NULL, FK → group_prefixes(id)	Префікс
group_course	group_course	-	int	1–6	NOT NULL	Курс
year	year	-	int	-	NOT NULL	Рік набору
curator_id	curator_id	Зовнішній	UUID	-	NULL, FK → users(id)	Куратор
specialty_id	specialty_id	Зовнішній	int	-	NOT NULL, FK → specialties(id)	Спеціальність

Для зберігання різних спеціальностей спроєктовано таблиці «departments» та «specialties».

Таблиця «departments» призначена для маштабованого зберігання інформації про відділення навчального закладу (табл. 2.9).

Таблиця 2.9 – Атрибути таблиці «departments»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID департаменту
name	name	-	varchar(64)	-	NOT NULL	Назва департаменту

Таблиця «specialties» зберігає інформацію про спеціальності, за якими ведеться підготовка здобувачів освіти. Вона містить унікальний ідентифікатор спеціальності, її назву та інформацію про відповідну освітню програму. Поле «department_id» є зовнішнім ключем, що пов'язує спеціальність з відділенням, до якого належить конкретна спеціальність. Ця таблиця є важливою частиною структури бази даних, оскільки вона визначає основні напрямки навчання в закладі освіти та дозволяє систематизувати інформацію про академічні групи, навчальні плани та інші аспекти освітнього процесу (табл. 2.10).

Таблиця 2.10 – Атрибути таблиці "specialties"

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	specialty_id	Первинний	int	> 0	NOT NULL, PK	ID спеціальності
specialty_name	specialty_name	-	varchar	-	NOT NULL	Назва спеціальності
edu_program	edu_program	-	varchar	-	NOT NULL	Освітня програма
department_id	department_id	Зовнішній	int	-	NOT NULL, FK departments(id)	Кафедра

Разом ці дві таблиці формують основу інформаційної моделі навчального процесу, дозволяючи зберігати та структурувати дані про відділення та спеціальності.

Для того щоб зберігати інформацію про те які студенти належать до яких груп створено проміжні таблиці «user_groups» та «user_subgroups». Також для того щоб поєднати методиста з відділенням якому вони належать та старост з їх групами створено таблиці «admin_departments» та «groups_monitors».

Для того щоб ефективно зберігати інформацію про періоди навчання, та початки семестру для відліку верхніх та нижніх тижнів створено модуль навчального часу (рис. 2.7).

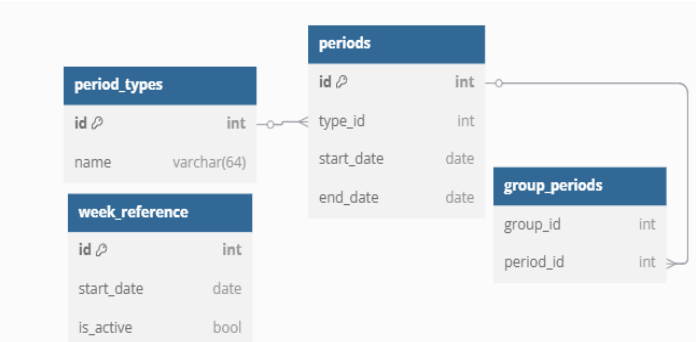

Рисунок 2.7 – Логічна модель бази даних модуля навчальних періодів

Для зберігання періодів навчальних семестрів, сесій та інше створено таблиці «periods» та «period_types».

Таблиця «periods» зберігає основну інформацію як початок та кінець певного періоду, та пов’язана з таблицею з «period_types» для визначення типу цього проміжку часу (табл. 2.11).

Таблиця 2.11 – Атрибути таблиці «periods»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID періоду
type_id	type_id	Зовнішній	int	-	NOT NULL, FK	Тип періоду
start_date	start_date	-	date	-	NOT NULL	Дата початку
end_date	end_date	-	date	-	NOT NULL	Дата завершення

Для того щоб пов’язати ці періоди із групами яким вони належать, створено таблицю «group_periods».

Для зручності обрахунку верхніх та нижніх тижнів створено таблицю «week_reference». Вона має в собі поле дати початку відліку та атрибут «is_active» для визначення чи ця дата ще дійсна (табл. 2.12).

Таблиця 2.12 – Атрибути таблиці «week_reference»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID тижня
start_date	start_date	-	date	-	NOT NULL	Початок тижня
is_active	is_active	-	bool	-	NOT NULL	Чи активний

Логічну модель розкладу занять зображено на рисунку 2.8.

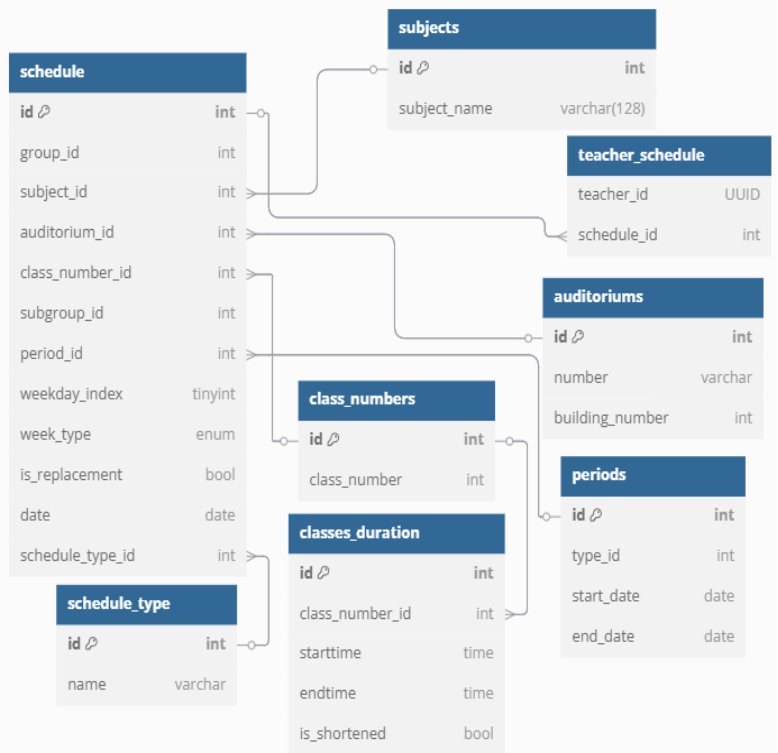

Рисунок 2.8 – Логічна модель бази даних модуля розкладу

Для масштабованого збереження усіх додаткових даних створені таблиці «schedule_type», «class_numbers», «classes_duration», «subjects» та «auditoriums». Ці таблиці поєднуються з основною сутністю «schedule» та забезпечують ефективне зберігання даних

Таблиця «schedule» зберігає інформацію про навчальні заняття в системі «е-Студент». Кожен запис містить унікальний ідентифікатор, посилання на предмет, групу, підгрупу, викладача та аудиторію. Часові параметри визначаються днем тижня, номером пари та ознакою верхнього/нижнього тижня. Додатково зберігається інформація про семестр та визначається чи пара є постійною чи це одноразова заміна, що дозволяє фільтрувати поточні заняття та відображати тимчасові зміни в розкладі (табл. 2.13).

Таблиця 2.13 – Атрибути таблиці «schedule»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	Ідентифікатор розкладу
group_id	group_id	Зовнішній	int	-	NOT NULL, FK	Група
subject_id	subject_id	Зовнішній	int	-	NOT NULL, FK	Предмет
auditorium_id	auditorium_id	Зовнішній	int	-	NOT NULL, FK	Аудиторія
class_number_id	class_number_id	Зовнішній	int	-	NOT NULL, FK	Номер пари
subgroup_id	subgroup_id	Зовнішній	int	-	NOT NULL, FK	Підгрупа
period_id	period_id	Зовнішній	int	-	NOT NULL, FK	Період
weekday_index	weekday_index	-	tinyint	0–6	NOT NULL	День тижня
week_type	week_type	-	enum	even/odd	NOT NULL	Тип тижня
is_replacement	is_replacement	-	bool	-	NOT NULL	Чи заміна
date	date	-	date	-	NULL	Дата конкретного заняття
schedule_type_id	schedule_type_id	Зовнішній	int	-	NOT NULL, FK	Тип розкладу

Логічну модель підмодуля відвідуваності зображено на рисунку 2.9.

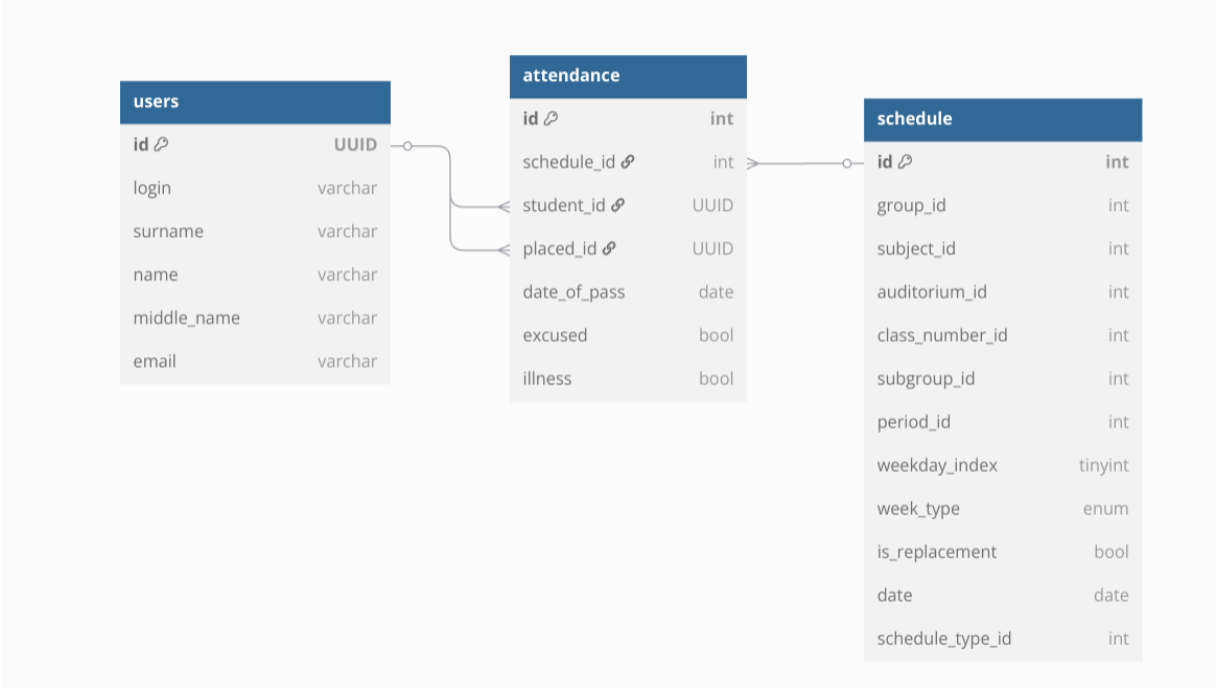

Рисунок 2.9 – Логічна модель бази даних модуля відвідуваності

Для зберігання відвідуваності студентів спроєктована таблиця «attendance», яка зберігає в собі інформацію про пару, на якій не був присутній студент (без або з поважної причини), дату та ID студента (табл. 2.14).

Таблиця 2.14 – Атрибути таблиці «attendance»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID запису відвідування
schedule_id	schedule_id	Зовнішній	int	-	NOT NULL, FK	Заняття
student_id	student_id	Зовнішній	UUID	-	NOT NULL, FK	Студент
placed_id	placed_id	Зовнішній	UUID	-	NOT NULL, FK	Особа, яка поставила
date_of_pass	date_of_pass	-	date	-	NOT NULL	Дата пропуску
excused	excused	-	bool	-	NOT NULL	Чи поважна причина
illness	illness	-	bool	-	NOT NULL	Чи через хворобу

Логічну модель підмодуля сповіщень зображено на рисунку 2.10.

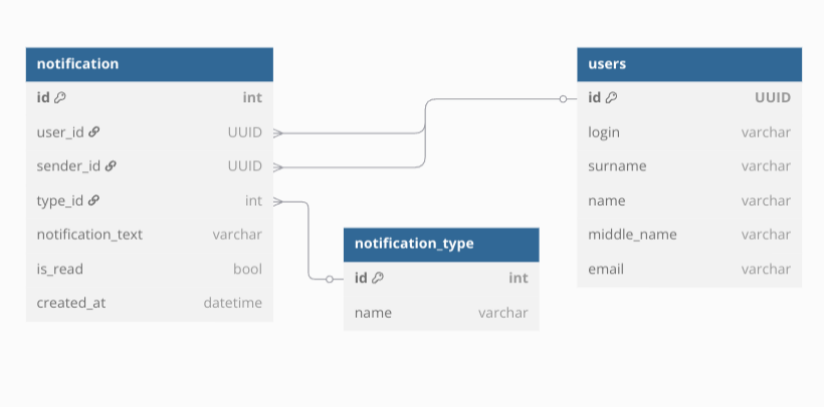

Рисунок 2.10 – Логічна модель бази даних модуля сповіщень

Таблиця «notification_type» створена для категоризації різних типів сповіщень в системі «e-Студент». Вона має простий дизайн з двома полями - первинним ключем «id» для унікальної ідентифікації кожного типу та полем «name», що містить назву типу сповіщення. Це дозволяє класифікувати повідомлення за призначенням, наприклад: зміни в розкладі, академічні оголошення, адміністративні повідомлення тощо (табл. 2.15).

Таблиця 2.15 – Атрибути таблиці «notification_type»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	> 0	NOT NULL, PK	ID типу сповіщення
name	name	-	varchar	-	NOT NULL	Назва типу сповіщення

Таблиця «notification» зберігає безпосередньо інформацію про окремі сповіщення в системі. Кожен запис має унікальний ідентифікатор, поле «user_id» для зв'язку з конкретним користувачем-отримувачем, «notification_type» для визначення категорії повідомлення, «text» із самим змістом сповіщення та «created_at», що фіксує дату і час створення повідомлення. Така структура забезпечує можливість надсилати персоналізовані сповіщення користувачам, відстежувати їх хронологію та організовувати за типами для зручного відображення в інтерфейсі (табл. 2.16).

Таблиця 2.16 – Атрибути таблиці «notification»

Атрибут	Умовне позначення	Ключ	Формат	Діапазон значень	Обмеження	Опис
id	id	Первинний	int	-	NOT NULL, PK	ID сповіщення
user_id	user_id	Зовнішній	UUID	-	NOT NULL, FK	Отримувач
sender_id	sender_id	Зовнішній	UUID	-	NOT NULL, FK	Відправник
type_id	type_id	Зовнішній	int	-	NOT NULL, FK	Тип сповіщення
notification_text	notification_text	-	varchar	-	NOT NULL	Текст сповіщення
is_read	is_read	-	bool	-	NOT NULL	Чи прочитано
created_at	created_at	-	datetime	-	NOT NULL	Дата створення

В результаті проєктування бази даних застосунку «е-Студент» було створено реляційну модель, що відповідає всім вимогам щодо зберігання та обробки інформації про навчальний процес. Розроблена структура включає таблиці для зберігання даних про користувачів, академічні групи, розклад занять, предмети, спеціальності та систему сповіщень, з чітко визначеними атрибутами та зв'язками між ними. Всі сутності нормалізовано для забезпечення цілісності даних та оптимізації їх зберігання, а для зв'язків типу багато-до-багатьох впроваджено проміжні таблиці. Використання зовнішніх ключів гарантує референційну цілісність бази даних та коректні взаємозв'язки між таблицями, а тип даних UUID для ідентифікаторів користувачів підвищує безпеку системи.

2.4 Проєктування серверної частини застосунку

Для забезпечення безпеки API в системі буде використано механізм JWT (JSON Web Tokens), який є відкритим стандартом (RFC 7519) для безпечної передачі даних між сторонами у вигляді JSON-об'єкта [9]. Цей стандарт

широко застосовується у сучасних веб-додатках та API для аутентифікації та авторизації користувачів, забезпечуючи при цьому високий рівень безпеки.

JWT токен складається з трьох частин: заголовка, корисного навантаження та підпису. В заголовку міститься інформація про тип токена та яким алгоритмом його шифрували, корисне навантаження містить в собі дані про користувача підпис гарантує що токен не був змінений після створення та забезпечує його цілісність.

У розроблюваній системі JWT буде використовуватися для захисту API-запитів. Після успішної автентифікації користувача сервер генеруватиме JWT-токен, після чого він передаватиметься клієнту, який повинен буде включати його в заголовок Authorization для всіх наступних запитів до захищених ресурсів API. При кожному запиті сервер перевірятиме валідність токена перед наданням доступу до захищених ресурсів.

Приклад використання JWT токена відображено в діаграмі послідовності на рисунку 2.11.

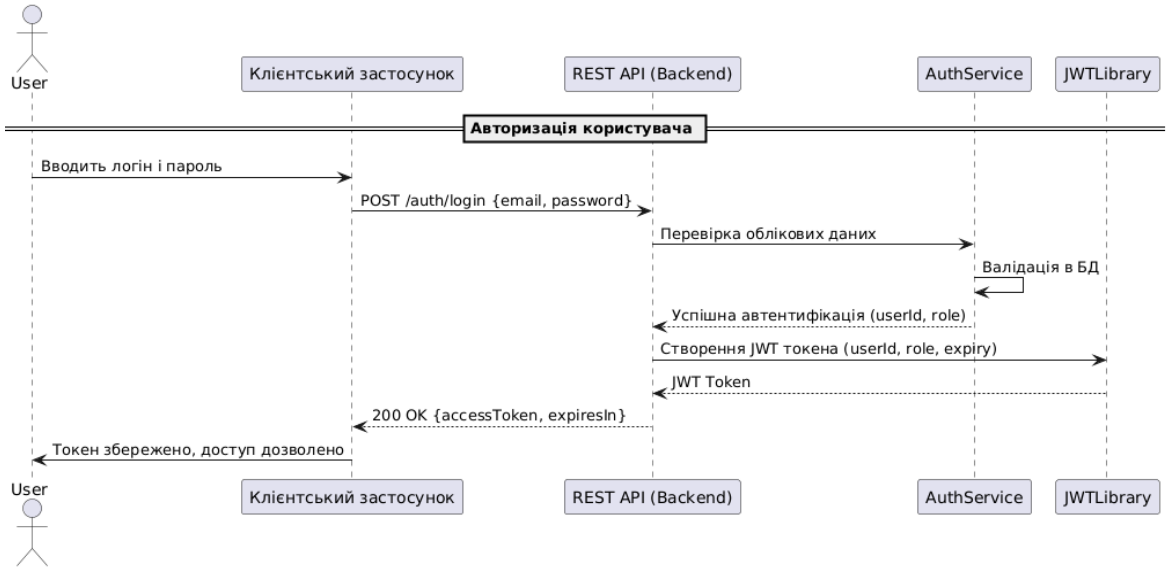

Рисунок 2.11 – Діаграма послідовності авторизації користувача

Схему обробки запитів буде організовано наступним чином: сервер отримуватиме HTTP-запит від клієнта, фреймворк Pistache оброблятиме цей запит і перенаправлятиме його до відповідного обробника згідно з

визначеними маршрутами. Обробник виконуватиме необхідну логіку, взаємодіятиме з базою даних MariaDB та формуватиме відповідь, яка буде відправлена клієнту у форматі JSON.

API буде організовано навколо базового шляху /v1, що вказуватиме на версію API та забезпечуватиме можливість майбутніх оновлень без порушення сумісності. Основні запити до ресурсів системи буде спроєктовано наступним чином:

Службові ресурси:

– /ready — перевірятиме готовність сервера до роботи;

– /swagger.json та /swagger.yaml — надаватимуть документацію API у форматах Swagger (JSON та YAML відповідно).

Ресурси для аутентифікації (/auth):

– /register — обробляє запит на реєстрацію нового користувача (POST);

– /login — здійснює вхід користувача до системи (POST);

– /logout — виконує вихід користувача з системи (POST);

– /refresh — оновлює авторизаційний токен (POST).

Ресурси для роботи з користувачами (/users):

– /users — повертає список усіх користувачів системи (GET);

– /users/:id — повертає дані конкретного користувача (GET);

– /users/:id/grades — повертає список оцінок, що належать зазначеному користувачу (GET);

– /users/:id/grades/:grade_id — повертає детальну інформацію про конкретну оцінку (GET);

– /users/:id/schedule — повертає розклад занять конкретного користувача (GET);

– /users/:id/attendance — повертає дані про відвідуваність обраного користувача (GET);

– /users/:id/notifications — повертає перелік сповіщень користувача (GET);

– /users/:id/notifications/:notification_id — повертає інформацію про окреме сповіщення (GET).

Ресурси розкладу (/schedule):

– /schedule — повертає весь поточний розклад (GET);

– /schedule/week_type — повертає тип поточного тижня (верхній/нижній) (GET);

Ресурси викладача (/teacher):

– /teacher/:id/schedule — повертає розклад викладача (GET);

– /teacher/:id/attendance — повертає інформацію про відвідування занять викладача (GET).

Ресурси старост груп (/monitor):

– /monitor/:id/attendance — повертає дані про відвідування студентів у межах певної групи (GET);

– /monitor/:id/attendance_report — формує звіт про відвідування студентів групи (POST).

Ресурси методиста (/admin)

– /admin/attendance/report — обробляє формування загального звіту про відвідуваність певного переліку груп (POST);

– /admin/:id/department_info — надає інформацію про відділення, закріплене за конкретним методистом (GET).

Ресурси відвідуваності (/attendance):

– /attendance/add — додає новий запис про відвідування (POST);

– /attendance/update — оновлює дані про відвідування (POST);

– /attendance/delete — видаляє запис про відвідування (POST);

– /attendance/batch — обробляє масове оновлення або вставку записів про відвідування (POST).

Ресурси сповіщень (/notification):

– /notification/create — створює нове сповіщення в системі (POST).

Для обміну даними буде використано формат JSON (JavaScript Object Notation), що є стандартним, легким і зручним форматом структурованої передачі даних. JSON підтримується практично всіма мовами програмування та платформами, а також є читабельним, що спростить відлагодження.

У результаті проєктування серверної частини було сформовано загальну структуру архітектури взаємодії між клієнтською частиною та сервером, визначено основні принципи побудови програмного інтерфейсу для доступу до даних, а також обрано ефективні засоби захисту інформації під час обміну між компонентами системи.

2.5 Проєктування користувацького інтерфейсу

Інтуїтивний та привабливий інтерфейс є ключовим фактором в досягненні успіху сучасними застосунками. Якісний користувацький інтерфейс забезпечує позитивний досвід взаємодії з продуктом та підвищує лояльність користувачів, їх залученість та ефективність роботи з системою.

Під час проєктування UI-дизайну важливо створити простий в навігації інтерфейс з дотриманням принципів юзабіліті, як зрозумілість функціональних елементів, забезпечення зворотного зв’язку та відповідність очікуванням користувача. Також дизайн повинен бути узгодженим на всіх сторінках застосунку та мати спільну кольорову гаму, шрифти, компоненти та використовувати єдиний стиль. Важливо створити адаптивний дизайн який буде, який буде забезпечувати конкретне відображення на пристроях різних розмірів.

В контексті використання клієнт-серверної архітектури інтерфейс відповідає за відображення даних, що отримуються з сервера та відправку запитів на основі дій користувача.

Першим етапом проєктування інтерфейсу зазвичай є створення прототипу низької деталізації (wireframes). Вони визначають розміщення елементів на екрані та є чорно білими без використання кольорів.

Макет головної сторінки застосунку для мобільних пристроїв зображено на рисунку 2.12.

Рисунок 2.12 – Макет головної сторінки мобільної версії

В блоці розкладу розташовуватимуться картки із персоналізованим розкладом занять користувача на певний день. В ньому відображатиметься повна інформація про пари, чи це заміна, аудиторія, викладач та час проведення. Така структура буде інтуїтивною та зрозумілою для всіх користувачів.

Після створення макетів було сформовано кольорову палітру для темної та світлої теми і обрано основний, другорядний і акцентний шрифт (рис. 2.13).

Рисунок 2.13 – Кольорова палітра та шрифти

Створено макет високої деталізації із використанням обраної палітри для сторінки відвідуваності. Інтерфейс для старости групи та студента зображено на рисунку 2.14.

Рисунок 2.14 – Дизайн сторінки відвідуваності

При проєктуванні дизайну застосунку було взято до уваги недоліки та переваги проаналізованих аналогів в першому розділі, та створено мінімалістичний та функціональний дизайн для зручного користування всіма категоріями користувачів.

Таким чином, спроєктована архітектура застосунку «е-Студент» забезпечує комплексне вирішення завдань автоматизації навчального процесу через застосування сучасних технологічних рішень та принципів проєктування. Обрана клієнт-серверна архітектура з REST API дозволяє створити масштабовану та надійну систему, яка ефективно розділяє відповідальність між компонентами. Спроєктована структура бази даних дозволяє ефективно зберігати та обробляти всю необхідну інформацію про навчальний процес, забезпечуючи при цьому цілісність даних та безпеку, а створений користувацький інтерфейс відповідає сучасним принципам юзабіліті та забезпечує інтуїтивну взаємодію для всіх категорій користувачів системи, враховуючи їхні специфічні потреби та права доступу.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка серверної частини

Першим кроком у розробці застосунку стала реалізація серверної частини програми. На цьому етапі було забезпечено взаємодію з базою даних, реалізовано механізми аутентифікації користувачів, а також обробку ключових функціональних процесів, таких як керування користувачами, розклад занять, реєстрація відвідуваності та сповіщення.

Застосунок «е-Студент» використовує серверну частину у вигляді API, розгорнуту на базі інфраструктури Галицького фахового коледжу, який уже має функціональні можливості, що дозволяють реєстрацію нових користувачів, вхід, оновлення токенів аутентифікації та виведення оцінок користувача. У межах проєкту, його було розширено додатковими маршрутами, які реалізують функціональність, необхідну для роботи з розкладом, обробкою сповіщень та веденням обліку відвідуваності. Це дозволило інтегрувати нові можливості без порушення існуючої архітектури системи.

Сервер складається із великої кількості файлів, основними з яких є RestAPI.cpp та Database.cpp. Файл RestAPI.cpp реалізує основну логіку ініціалізації, запуску та опису REST API інтерфейсу сервера застосунку. Його структура чітко організована за функціональними блоками, що відповідають за різні етапи життєвого циклу HTTP-сервера та формування маршрутів. Файл Database.cpp реалізує набір утилітарних функцій для взаємодії з базою даних MariaDB. Він містить функції, що дозволяють виконувати типові запити до таблиць користувачів, сесій, сповіщень, відвідуваності, а також обробку JWT-секретів. Завдяки використанню RAII через Database::ConnectionGuard, забезпечується автоматичне керування з’єднаннями з базою даних.

У методі createDescription() файлу RestAPI.cpp відбувається формування структури всього REST API сервера. Спочатку задається основна інформація

про API: його версія, ліцензія (GPLv3) та формати MIME, які сервер приймає та повертає у відповідь. Далі оголошується базовий маршрут /v1, до якого додаються вкладені підшляхи для різних груп функціональності. Зокрема, це маршрути для авторизації (/auth) та роботи з користувачами (/users),

До маршруту /users було додано нові точки доступу для обробки запитів пов'язаних з інформацією про користувачів. Кожен з ресурсів прив'язаний до відповідної функції, що виконує передбачені маршрутом функції. Додавання точок доступу до маршруту /users наведено в лістингу 3.1.

Лістинг 3.1 – Додавання точок доступу до маршруту /users

```
auto usersPath = v1.path("/users");
usersPath.route(desc.get("/:id/schedule")).bind(&RestAPI::fetch_
user_shedule, this).produces(MIME(Application, Json));
usersPath.route(desc.get("/:id/attendance")).bind(&RestAPI::fetc
h_user_attendance, this).produces(MIME(Application, Json));
usersPath.route(desc.get("/:id/notifications")).bind(&RestAPI::f
etch_user_notifications, this).produces(MIME(Application, Json));
usersPath.route(desc.get("/:id/notifications/:notification_id"))
        .bind(&RestAPI::fetch_user_notification, this)
        .produces(MIME(Application, Json));
```

Функції мають певну структуру, якої було дотримано при написанні усіх методів класу RestAPI. Кожна процедура має приймати в себе запит та відправляти клієнту відповідь. Також на початку кожної функції має бути перевірка на те чи запит авторизований чи ні, та створення об'єкта-захисника (RAII-обгортки) для роботи з підключенням до бази даних із пулу з'єднань. Приклад реалізації структури функції наведено в лістингу 3.2.

Лістинг 3.2 – Структура функції

```
void RestAPI::function(const Rest::Request &request,
Http::ResponseWriter response)
{
    if (!checkAuthorize(request)) {
        response.send(Http::Code::Unauthorized, "Unauthorized");
        return;
    }
    Database::ConnectionGuard conn(*m_connPool);
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));
}
```

Для того щоб ефективніше працювати з датами всередині основних функцій, було створено допоміжні методи. Реалізація процедури для отримання індексу дня тижня за датою наведено в лістингу 3.3.

Лістинг 3.3. – Реалізація функції getWeekdayIndex

```
int getWeekdayIndex(const std::string &dateStr)
{
    std::tm t = {};
    std::istringstream ss(dateStr);
    ss >> std::get_time(&t, "%Y-%m-%d");
    if (ss.fail())
        return -1;
    std::time_t time = std::mktime(&t);
    if (time == -1)
        return -1;
    std::tm *local_t = std::localtime(&time);
    if (!local_t)
        return -1;
    return local_t->tm_wday;
}
```

Для того щоб отримувати індекси днів тижня в проміжку часу, створено функцію getWeekdaysInRange. Реалізація процедури для отримання індексів днів тижня в проміжку часу наведено в лістингу 3.4.

Лістинг 3.4 – Код функції getWeekdaysInRange

```
std::set<int> getWeekdaysInRange(const std::string &from, const
std::string &to)
{
    std::set<int> weekdays;
    std::chrono::time_point<std::chrono::system_clock,
std::chrono::days> date_from;
    std::chrono::time_point<std::chrono::system_clock,
std::chrono::days> date_to;
    std::istringstream in_from(from);
    in_from >> std::chrono::parse("%d.%m.%Y", date_from);
    std::istringstream in_to(to);
    in_to >> std::chrono::parse("%d.%m.%Y", date_to);
    for (auto day = date_from; day <= date_to; day +=
std::chrono::days{1}) {
        std::chrono::weekday wd(day);
        weekdays.insert(wd.c_encoding());
    }
    return weekdays;
}
```

Для того щоб ефективніше формувати запити до бази даних, було реалізовано функцію яка створює замінювачі для позиційних параметрів в залежності від кількості цих параметрів. Програмний код представлено в лістингу 3.5.

Лістинг 3.5 – Код функції generatePlaceholders

```
std::string generatePlaceholders(size_t count)
{
    std::string result;
    for (size_t i = 0; i < count; ++i) {
        if (!result.empty())
            result += ", ";
        result += "?";
    }
    return result;
}
```

Для того щоб ефективніше передавати дату в запитах до бази даних, було реалізовано функцію для отримання поточної дати. Код методу представлено в лістингу 3.6.

Лістинг 3.6 – Код функції getTodayDateString

```
std::string getTodayDateString()
{
    std::time_t now = std::time(nullptr);
    std::tm *now_tm = std::localtime(&now);
    char buf[11];
    std::strftime(buf, sizeof(buf), "%Y-%m-%d", now_tm);
    return std::string(buf);
}
```

Використання допоміжних функцій дозволяє уникнути дублювання логіки та підтримувати код у впорядкованому та читабельному вигляді.

Реалізовано функцію, яка повертає персональний розклад користувача fetch_user_shedule. Цей метод приймає в себе як параметри в шляху запиту значення from та to, які потім в запиті обмежують період часу на який повертаються пари. Якщо параметру не було передано, функція повертає розклад на поточну пару. Код реалізації наведено в додатку А.

Відповідь формується в зручному форматі, де дані розкладу групуються по предметах, та формується додатковий список викладачів, із інформацією

про них, та додатково виводяться періоди в які входить діапазон вказаних дат в запиті.

Аналогічно реалізовується функція для отримання відвідуваності користувача за маршрутом «/v1/users/:id/attendance». Якщо в параметрі id вказано «me», функція розшифровує токен авторизації, визначає session_id та отримує фактичний user_id поточного користувача. Код отримання ідентифікатора користувача із сесії наведено в лістингу 3.7.

Лістинг 3.7 – Отримання ідентифікатора користувача

```
if (user_id == "me") {
    const auto auth_header = request.headers().tryGet<Http::Header::Authorization>();
if (!auth_header) {
  response.send(Http::Code::Unauthorized, "Unauthorized: Invalid token.\n");
  return;}
    const std::string session_id = jwt::decode(auth_header->value().substr(7)).get_payload_claim("session_id").as_string();
       const auto session_info = Database::getSession(conn, session_id);
if (!session_info.has_value()) {
  response.send(Http::Code::Not_Found, "NotFound: Invalid session");
   return;}
user_id = session_info->user_id;}
```

Метод отримує розширену інформацію про присутність студента на заняттях, що відбулись у межах поточного навчального періоду, та повертає структуровану відповідь у форматі JSON.

Також для користувача створено функцію отримання своїх сповіщень fetch_user_notifications. Вона повертає всі повідомлення, надіслані конкретному користувачу, із мінімальною деталізацією інформації про них, для виведення в списку. Код реалізації наведено в додатку Б.

Для того щоб отримувати детальну інформацію про окреме сповіщення реалізовано функцію fetch_user_notification, яка параметром приймає ідентифікатор сповіщення, деталі якого потрібно отримати, та повертає уже з

повними даними про відправника, текст, та час. Код реалізації наведено в додатку В.

Було створено маршрут /schedule, до якого додано відповідні точки доступу для обробки запитів, що стосуються даних розкладу. Кожна з цих точок доступу асоційована з окремою функцією, яка виконує відповідну логіку згідно з призначенням маршруту.

Метод fetch_schedule повертає загальний розклад занять (незалежно від користувача) за вказаною датою або діапазоном дат. Під час запиту за шляхом «v1/schedule», до нього можна додавати або параметри from та to, з якими буде виводитися розклад за обраний період, або передавати лише один параметр date, з яким буде отримано заняття на конкретну дату. Якщо не передавати жодних параметрів, дата буде визначена за поточною. Код реалізації наведено в додатку Г.

Також в цій точці доступу реалізована функція для отримання типу тижня (верхній/нижній) за датою. Метод is_upper_week повертає одне значення – «0» якщо тиждень нижній та «1» якщо верхній. Код реалізації наведено в лістингу 3.8.

Лістинг 3.8 – Код функції is_upper_week

```
std::string date = request.query().get("date").value_or("");
std::string formatted_date;
int weekType = -1;
if (date.empty()) {date = getTodayDateString();}
formatted_date = convertDateToSqlFormat(date);
std::unique_ptr<sql::PreparedStatement>                stmt(conn-
>prepareStatement("SELECT get_week_type(?) AS week_type"));
stmt->setString(1, formatted_date);
std::unique_ptr<sql::ResultSet> rs(stmt->executeQuery());
if  (rs->next()  &&  !rs->isNull("week_type"))  {weekType  =  rs-
>getInt("week_type");}
if (weekType == 0 || weekType == 1) {nlohmann::json jsonResponse
=     {{"week_type",     weekType}};response.send(Http::Code::Ok,
jsonResponse.dump(2), MIME(Application, Json));}
else {response.send(Http::Code::Bad_Request, "Invalid date or no
active reference found");}
```

Для методів викладача було створено маршрут /teacher, до якого додано відповідні точки доступу для обробки запитів, що стосуються викладача. Кожна з цих точок доступу асоційована з окремою функцією, яка виконує відповідну логіку згідно з призначенням маршруту.

В цій гілці шляхів, передбачена функція отримання персонального розкладу викладача методом fetch_teachers_shedule у заданому діапазоні дат. Вона також як попередні може приймати в себе параметри from та to, або жодного, в разі чого вона передає у відповідь розклад на поточну дату. Код реалізації наведено в додатку Ґ.

У відповідь на запит функції fetch_teachers_shedule сервер формує та повертає JSON-структуру, яка містить розклад занять для викладача. Окремими масивами виводяться дані про періоди та викладача, після чого виводиться масив предметів, всередині яких знаходяться дані про пари з цих дисциплін.

Також, для викладача реалізовано функцію fetch_lesson_attendance яка відповідає за отримання інформації про відвідуваність занять викладача на конкретну дату. Ключовою перевагою функції є гнучка обробка різних типів занять (заміна чи регулярна пара), з підвантаженням не лише відвідуваності, а й повного списку студентів у групі, та усіх пар в цей день для викладача, навіть якщо для них ще немає записів про відсутність – що особливо корисно для UI редагування пропусків викладачем. Код реалізації наведено в додатку Д.

Для методів старости було створено маршрут /monitor, до якого додано відповідні точки доступу для обробки запитів, що стосуються старости. Кожна з цих точок доступу асоційована з окремою функцією, яка виконує відповідну логіку згідно з призначенням маршруту.

Маршрут старости має в собі шляхи для опрацювання відвідуваності групи. Функція fetch_group_attendance працює за схожим принципом як для викладача: шлях перевіряє авторизацію користувача (повертає 401 якщо не авторизовано), визначає його групу через таблицю «users_groups», отримує

список студентів цієї групи, формує SQL-запит до таблиці «schedule» з приєднанням «attendance» для вибірки пар у заданому періоді (або поточній даті), створює об'єкт schedule_map з інформацією про кожну пару включно з масивом відміток про відсутність студентів, та повертає JSON-відповідь що містить масив пар з відвідуваністю (schedule_array), список студентів групи (students_array) та деталі навчального періоду (period_details).

Також, передбачено функцію form_group_attendance_report для створення JSON статистики пропусків студентів групи за заданий місяць та обрані періоди. Під час створення запиту до цього вузла, необхідно також передати JSON-структуру з вказаними періодами, за якими буде відбуватися підрахунок.

Для того щоб динамічно генерувати SQL-запит для підрахунку пропусків по кожному періоду окремо, застосовується цикл, який формує частину запиту з умовними підрахунками «COUNT(CASE WHEN ...)» для кожного інтервалу дат, вказаного у вхідних даних. Код представлено в лістингу 3.9.

Лістинг 3.9 – Динамічна генерація SQL-запиту

```
period_columns += ", COUNT(CASE WHEN a.date_of_pass BETWEEN ?
AND ? THEN 1 END) AS total_period_" + std::to_string(i) +
                  ", COUNT(CASE WHEN a.date_of_pass BETWEEN ?
AND ? AND a.excused = 1 THEN 1 END) AS excused_period_" +
                  std::to_string(i) +
                  ", COUNT(CASE WHEN a.date_of_pass BETWEEN ?
AND ? AND a.illness = 1 THEN 1 END) AS illness_period_" +
                  std::to_string(i);
```

Повний код реалізації наведено в додатку Е.

Для методів методиста було створено маршрут /admin, до якого додано відповідні точки доступу для обробки запитів, що стосуються методиста.

Для методиста реалізовано схожу функцію для створення звіту відвідуваності як для старости. Функція form_attendance_report виконує генерацію зведеного звіту про відвідуваність студентів для вибраних груп за вказаний місяць. Вона приймає у запиті список ідентифікаторів груп та місяць

у форматі «YYYY-MM», після чого формує SQL-запит з динамічною кількістю параметрів для кожної групи. Запит підраховує загальну кількість пропусків та кількість поважних (виправданих) причин для кожного студента. Результати групуються за групами, а у відповідь повертається структурований JSON-звіт з інформацією по кожному студенту.

Також щоб отримувати дані про відповідне відділення до якого належить методист, створена функція fetch_department_info, яка реалізує логіку отримання ієрархічної інформації про структурні підрозділи, спеціальності та групи, що належать до певного адміністратора. Повний код реалізації наведено в додатку Є.

Для того щоб робити різні дії з відвідуваністю (додавання, оновлення, видалення), було створено маршрут /attendance, до якого додано відповідні точки доступу для обробки запитів, що стосуються відвідуваності. Кожна з цих точок доступу асоційована з окремою функцією, яка виконує відповідну логіку згідно з призначенням маршруту.

В класі Database було створено відповідні функції для редагування даних в БД. Код функції insertAttendance представлено в лістингу 3.10.

Лістинг 3.10 – Функція insertAttendance

```
std::optional<std::string>
insertAttendance(std::shared_ptr<sql::Connection>    conn,    int
schedule_id,  const  std::string  &placed_id,const  std::string
&student_id, const std::string &date_of_pass, bool excused,bool
illness){std::unique_ptr<sql::PreparedStatement>       stmt(conn-
>prepareStatement("INSERT    INTO    attendance    (schedule_id,
placed_id, student_id, date_of_pass, excused, illness) ""VALUES
(?, ?, ?, ?, ?, ?) RETURNING id;"));
    stmt->setInt(1, schedule_id);
    stmt->setString(2, placed_id);
    stmt->setString(3, student_id);
    stmt->setString(4, date_of_pass);
    stmt->setBoolean(5, excused);
    stmt->setBoolean(6, illness);
    std::unique_ptr<sql::ResultSet> rs(stmt->executeQuery());
    if (!rs->next()) {
        return std::nullopt;}
    return std::string{rs->getString(1).c_str()};}
```

Використання цієї функції в класі RestAPI наведено в лістингу 3.11.

Лістинг 3.11 – Реалізація функції add_attendance

```
void RestAPI::add_attendance(const Pistache::Rest::Request
&request, Pistache::Http::ResponseWriter response)
{
    if (!checkAuthorize(request)) {
        response.send(Http::Code::Unauthorized, "Unauthorized");
        return;}
    auto json = nlohmann::json::parse(request.body(), nullptr,
false);
    if (json.is_discarded() || !json.contains("schedule_id") ||
!json.contains("student_id") || !json.contains("placed_id")) {
        response.send(Http::Code::Bad_Request, "Missing required
fields");return;}
    int schedule_id = json["schedule_id"];
    std::string placed_id = json["placed_id"];
    std::string student_id = json["student_id"];
    std::string date = json.value("date", getTodayDateString());
    bool excused = json.value("excused", false);
    bool illness = json.value("illness", false);
    Database::ConnectionGuard conn(*m_connPool);
    auto id = Database::insertAttendance(conn, schedule_id,
placed_id, student_id, date, excused, illness);
    if (!id.has_value()) {
        response.send(Http::Code::Internal_Server_Error, "Failed
to insert attendance");
        return;
    }response.send(Http::Code::Created, id.value());}
```

Дана функція приймає в себе в запиті JSON файл із даними які необхідно додати в базу даних, та метод з класу Database обробляє їх та додає.

Аналогічно було реалізовано функції для оновлення та видалення записів про відвідуваність.

Для методів керування сповіщеннями було створено маршрут /notification, до якого додано відповідні точки доступу для обробки запитів, що стосуються сповіщень. Кожна з цих точок доступу асоційована з окремою функцією, яка виконує відповідну логіку згідно з призначенням маршруту.

Для реалізації механізму push-сповіщень через FCM на сервері було створено універсальну функцію sendPushNotification. Вона виконує HTTPS POST-запит до FCM API за допомогою бібліотеки cpr і дозволяє надсилати повідомлення на пристрій користувача за вказаним токеном.

Перед викликом цієї функції виконується кілька попередніх кроків: спочатку з бази даних зчитується FCM-токен користувача з таблиці «device_tokens» за його «user_id», після чого формується вміст повідомлення.

Функція інтегрована в логіку REST API: одразу після створення нового запису у таблиці notifications, з відповідним текстом повідомлення та типом, сервер викликає sendPushNotification, автоматично ініціюючи надсилання повідомлення на пристрій користувача. Код реалізації функції sendPushNotification представлено в лістингу 3.12.

Лістинг 3.12 – Код функції sendPushNotification

```
void sendPushNotification(const std::string& deviceToken, const
std::string& title, const std::string& body) {
    std::string serverKey = "SERVER_KEY";
    cpr::Response r = cpr::Post(
        cpr::Url{"https://fcm.googleapis.com/fcm/send"},
        cpr::Header{
            {"Authorization", "key=" + serverKey},
            {"Content-Type", "application/json"}},
        cpr::Body{
            R"({
                "to": ")" + deviceToken + R"(",
                "notification": {
                    "title": ")" + title + R"(",
                    "body": ")" + body + R"("
                },
                "data": {
                    "custom_key": "value"
                }
            })"});
    std::cout << "FCM Response: " << r.status_code << " - " <<
r.text << std::endl;
}
```

Код функції add_notification представлено в додатку Ж.

Таким чином, реалізовані функції на сервері забезпечують повний цикл обробки та взаємодії з даними відвідуваності, розкладу та сповіщень. Завдяки структурованій архітектурі та використанню сучасних підходів серверна частина гарантує надійність, масштабованість і зручну інтеграцію з клієнтським застосунком.

3.2 Реалізація клієнту

Після реалізації серверної частини наступним етапом стало створення користувацького інтерфейсу застосунку на Flutter. Основна мета цього підрозділу – описати, яким чином реалізовано візуальні компоненти, маршрутизацію, взаємодію з API а також забезпечити інтуїтивно зрозумілу навігацію для користувачів різних ролей.

Початковою точкою запуску клієнтської частини застосунку «e-Студент» є функція main(), в якій виконуються ключові налаштування перед стартом інтерфейсу користувача. Код реалізації функції main представлено в лістингу 3.13.

Лістинг 3.13 – Код функції main

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initializeDateFormatting('uk', null);
  final storage = SecureStorage();
  final apiClient = RestAPIClient(baseUrl, storage);
  final dio =
Dio()..interceptors.add(AuthInterceptor(apiClient));
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgrou
ndHandler);
  runApp(
    ChangeNotifierProvider(
      create: (_) => ThemeNotifier(),
      child: MyApp(apiClient: apiClient, dio: dio),
    ),
  );
}
class MyApp extends StatelessWidget {
  final RestAPIClient apiClient;
  final Dio dio;
  const MyApp({super.key, required this.apiClient, required
this.dio});
  @override
  Widget build(BuildContext context) {
    final themeNotifier = Provider.of<ThemeNotifier>(context);
    return MaterialApp(
      title: 'e-Student',
      debugShowCheckedModeBanner: false,
      theme: themeNotifier.currentTheme,
      home: SplashScreen(),);}}
```

На першому етапі ініціалізується Flutter-оточення та встановлюється українська локалізація дати. Далі створюється клієнт для роботи з REST API (RestAPIClient) разом із Dio та перехоплювачем аутентифікації (AuthInterceptor), який автоматично оновлює токени. Також реєструється обробник фонових push-сповіщень через Firebase. Після всіх підготовчих кроків запускається основна частина застосунку – віджет MyApp, який обгортається у ChangeNotifierProvider, щоб забезпечити перемикання теми інтерфейсу. Початково відкривається SplashScreen, який виконує перевірку авторизації перед перенаправленням користувача до відповідного інтерфейсу.

Для взаємодії із серверною частиною реалізовано клас RestAPIClient, у якому визначені методи для авторизації, отримання інформації про користувача, тощо. Кожен з цих методів використовує await [10], щоб дочекатися результату від сервера, не блокуючи при цьому основний потік додатку. Усі методи повертають об'єкти Future, які дозволяють UI «дочекатися» даних без заморожування інтерфейсу. Реалізація структури класу та конструктора представлено в лістингу 3.14.

Лістинг 3.14 – Конструктор класу RestAPIClient

```
class RestAPIClient {
  final String baseUrl;
  final SecureStorage storage;
  late final Dio dio;
  RestAPIClient(this.baseUrl, this.storage) {
    dio = Dio(BaseOptions(
      baseUrl: baseUrl,
      headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
      },
    ));
    dio.interceptors.add(AuthInterceptor(this));
  }
}
```

Також реалізовано AuthInterceptor, який забезпечує централізоване керування токенами автентифікації в Dio клієнті, автоматично додаючи «access_token» до кожного запиту та перевіряючи його термін дії. При

виявленні протермінованого токена інтерцептор автоматично виконує його оновлення через механізми оновлення і повторює початковий запит, а у випадку невдалого оновлення очищує сесію користувача. Такий підхід дозволяє уникнути дублювання логіки авторизації в кожному API-методі, централізуючи всю обробку токенів в одному місці. Повний код реалізації інтерцептора наведено в додатку І.

Для безпечного зберігання чутливих даних у застосунку реалізовано клас SecureStorage, який інкапсулює взаємодію з бібліотекою «flutter_secure_storage» [11]. Цей клас використовує синглтон паттерн [12], щоб гарантувати єдиний екземпляр сховища в межах усього застосунку. Він надає методи для зчитування, запису, видалення окремих ключів, а також очищення всього сховища. Крім того, SecureStorage підтримує роботу з об’єктами у форматі JSON, дозволяючи зберігати та відновлювати цілі структури даних за допомогою методів writeObject і readObject. Усі операції обгорнуті в «try-catch» блоки з детальними логами, що забезпечує контроль над помилками й спрощує налагодження. Такий підхід дозволяє зручно і безпечно зберігати токени авторизації, ролі та інші конфіденційні дані. Повний код реалізації наведено в додатку К.

Клас SplashScreen відповідає за початкову ініціалізацію застосунку. Під час запуску він перевіряє, чи користувач уже авторизований (checkLoginStatus). Залежно від результату, користувача автоматично перенаправляє на головний екран (HomeScreen) або екран входу (LoginScreen). Поки триває перевірка, відображається індикатор завантаження. Реалізація функції checkLoginStatus представлена в лістингу 3.15.

Лістинг 3.15 – Функція checkLoginStatus

```
Future<bool> checkLoginStatus() async {
    final refreshToken = await storage.read('refresh_token');
    if (refreshToken == null) return false;

    if (await isRefreshTokenExpired()) {
      await clearTokens();
      await storage.clearAll();return false;}return true;}
```

Для того щоб забезпечити користувачам можливість персоналізувати застосунок, було реалізовано модуль з темами. Він складається із трьох основних файлів: кольори, оповіщувач змін, та самі теми.

В файлі із кольорами створено класи для темної та світлої тем, де прописано основну кольорову гаму для кожної з тем, з кольорами які було обрано під час проєктування інтерфейсу. Їх використано при створенні ThemeData для основних тем застосунку.

Клас ThemeNotifier відповідає за керування темою інтерфейсу користувача в застосунку. Він реалізує механізм зміни та збереження вибраної теми за допомогою SharedPreferences, а також сповіщення всіх залежних від теми віджетів про зміни.

Після ініціалізації, клас завантажує тему з локального сховища (за замовчуванням світла тема) й оновлює відповідний стиль. Метод setLightTheme та setDarkBlueTheme дозволяють змінити тему на одну з попередньо визначених (lightTheme, darkBlueTheme), з одночасним збереженням вибору у пам’яті пристрою. Зміни автоматично поширюються на всі компоненти інтерфейсу через виклик notifyListeners(), оскільки клас реалізує ChangeNotifier. Повний код реалізації наведено в додатку Л.

Екран LoginScreen реалізує інтерфейс входу до застосунку і відповідає за автентифікацію користувача. У класі передбачені два текстові поля для введення логіну та паролю, а також кнопка для підтвердження дії. Код реалізації представлено в лістингу 3.16.

Лістинг 3.16 – Поле входу

```
body: Center(
   child: Padding(
      padding: const EdgeInsets.all(32.0),
      child: ConstrainedBox(
        constraints: const BoxConstraints(maxWidth: 400),
         child: Column(
           mainAxisAlignment: MainAxisAlignment.center,
            children: [
              TextField(
               controller: _loginController,
               decoration: const InputDecoration(
```

```
                  labelText: 'Логін',
                   border: OutlineInputBorder(),),
                 const SizedBox(height: 20),
                 TextField(controller: _passwordController,
                   decoration: const InputDecoration(labelText:
'Пароль',border: OutlineInputBorder(),),obscureText:
true,),const SizedBox(height: 20),
                 ElevatedButton(
                   onPressed: _login,
                   style: ElevatedButton.styleFrom(
                     minimumSize: const Size(200, 50),
                   ),child: const Text('Увійти'),),],),),),),
```

Після введення облікових даних та натискання кнопки «Увійти», виконується метод _login, який відправляє запит до серверу за допомогою RestAPIClient. У разі успішної авторизації користувач перенаправляється на головний екран застосунку (HomeScreen). У разі помилки на екрані з’являється повідомлення про невдалий вхід. Код реалізації представлено в лістингу 3.17.

Лістинг 3.17 – Метод _login

```
Future<void> _login() async {
    final login = _loginController.text;
    final password = _passwordController.text;
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => const Center(
        child: CircularProgressIndicator(),
      ),
    );
    try {
      final result = await _apiClient.login(login, password);
      if (mounted) Navigator.of(context).pop();
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder:     (context)     =>     const
HomeScreen()),
      );
    } catch (e) {
      print('Login failed: $e');
      if (mounted) Navigator.of(context).pop();
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Помилка входу: $e')),
      );
    }}
```

Дана функція використовує функцію login в класі RestApiClient. Реалізацію функції представлено в лістингу 3.18.

Лістинг 3.18 – Функція login

```
Future<Map<String, dynamic>> login(String login, String
password) async {
    print("[${DateTime.now().toIso8601String()}] Вхід
користувача");
    try {
      final response = await dio.post(
        '/v1/auth/login',
        data: {'login': login, 'password': password},
      );
      await saveTokens({
        'access': response.data['tokens']['access'],
        'refresh': response.data['tokens']['refresh'],
        'access_expiry':
_parseExpiry(response.data['tokens']['access']),
        'refresh_expiry':
_parseExpiry(response.data['tokens']['refresh']),
      });
      print("[${DateTime.now().toIso8601String()}] Успішний
вхід");
      return response.data;
    } on DioException catch (e) {
      print(
          "[${DateTime.now().toIso8601String()}] Помилка логіну:
${e.message}");
      throw Exception(e.response?.data['message'] ?? 'Login
failed');
    }
  }
```

Після успішного входу користувача здійснюється перехід на головну сторінку, де відображається розклад занять. Для зручної навігації між основними розділами застосунку було реалізовано компонент CustomBottomNavigationBar, який доступний на всіх сторінках інтерфейсу.

Цей віджет відповідає за нижню панель навігації, що дозволяє користувачу швидко перемикатися між п’ятьма основними вкладками для студента: «Головна», «Відвідуваність», «Сповіщення», «Оцінки» та «Профіль». Кожен пункт меню має власну SVG-іконку, яка динамічно змінює колір відповідно до активної вкладки. Компонент інтегрується зі світлою або темною темою через Theme.of(context) і адаптується до розміру екрана

пристрою, забезпечуючи зручність користування на різних платформах. Якщо роль користувача не студент, то відображаються лише вкладки головної сторінки, відвідуваності, сповіщень та профілю. Повний код реалізації представлено в додатку М.

Для головної сторінки реалізовано appbar, в якому відображається назва застосунку, поточна дата, та те який зараз типу тижня (верхній чи нижній). Код представлено в лістингу 3.19.

Лістинг 3.19 – Appbar головної сторінки

```
appBar: AppBar(
automaticallyImplyLeading: false,
toolbarHeight: MediaQuery.of(context).size.height * 0.09,title:
Row(children: [SvgPicture.asset('assets/icons/logo.svg',height:
MediaQuery.of(context).size.height * 0.05, color:
theme.iconTheme.color,),
const SizedBox(width: 4),
Text('e-Студент',style: TextStyle(fontSize:
MediaQuery.of(context).size.width * 0.05,),),
const Spacer(),
Column(
crossAxisAlignment: CrossAxisAlignment.end,
children: [
Text(
getFormattedDate(),style: TextStyle(fontSize:
MediaQuery.of(context).size.width * 0.04,),),
if (_weekType != null)
Text(weekType!,style: TextStyle(fontSize:
MediaQuery.of(context).size.width * 0.04,color:
theme.primaryColorDark,),),],),],),),
```

Для отримання розкладу студента з серверу використовується функція fetchSchedule. Код реалізації представлено в лістингу 3.20.

Лістинг 3.20 – Код функції “fetchSchedule”

```
Future<List<FullScheduleInfo>> fetchSchedule(
      {DateTime? from, DateTime? to, DateTime? date}) async {
    final start = DateTime.now().toIso8601String();
    print("[$start] Запит розкладу /v1/users/me/schedule");
    try {
      final queryParams = <String, String>{};
      if (date != null) {
        queryParams['date'] = _formatDate(date);
      } else {
        if (from != null) queryParams['from'] =
_formatDate(from);
```

```
        if (to != null) queryParams['to'] = _formatDate(to);
      }
      final response = await dio.get(
        '//v1/users/me/schedule',
        queryParameters: queryParams,
      );
      final end = DateTime.now().toIso8601String();
      print("[$end] Розклад отримано ${response.data}");
      return FullScheduleInfo.fromApiResponse(response.data);
    } on DioException catch (e) {
      print(
          "[${DateTime.now().toIso8601String()}] Помилка при
отриманні розкладу: ${e.message}");
      throw Exception('Failed to fetch schedule: ${e.message}');
    }
  }
```

Вона повертає список об’єктів класу FullScheduleInfo. Цей клас є моделлю, що описує повну інформацію про заняття, включаючи назву предмета, групу, час проведення, викладачів та інші параметри. Аналогічно реалізовано функцію для розкладу викладача.

У конструкторі класу FullScheduleInfo передбачені всі необхідні поля, зокрема і додаткові «buildingNumber» та «scheduleTypeName», які містять номер корпусу та тип розкладу відповідно. Метод fromApiResponse використовується для обробки та трансформації JSON-відповіді від серверу у список об’єктів FullScheduleInfo. При цьому окремо формується карта викладачів, яка дозволяє об’єднати повні імена за «teacher_ids». У разі, якщо один і той самий розклад зберігає кількох викладачів, їх імена об’єднуються у список. Повний код реалізації представлено в додатку Н.

Для того щоб виводити картки з розкладом, у застосунку створено віджет DayScheduleCard. Цей компонент відповідає за відображення всіх занять певного дня у зручному вигляді. Він приймає на вхід дату, список занять (FullScheduleInfo), тип тижня та роль користувача (студент або викладач).

Картка автоматично форматує дату, наприклад, виводячи «сьогодні» для поточного дня. Заняття сортуються за номером пари, і при цьому

враховуються заміни: якщо на певну пару запланована заміна, то вона відображається замість постійної.

У середині віджета кожна пара відображається з детальною інформацією: час початку і завершення, номер пари, аудиторія та корпус, тип заняття (якщо він відрізняється від стандартного), а також імена викладачів або назва групи – залежно від ролі користувача. Для візуального акценту заміни та особливі типи занять позначаються кольоровими бейджами.

Весь розклад подається у стилізованій прокручуваній картці з заокругленими краями та тінню, що забезпечує сучасний та приємний зовнішній вигляд. Компонент адаптовано до розміру екрану і підтримує зміну тем оформлення. Повний код реалізації представлено в додатку П.

На головному екрані застосунку реалізовано логіку динамічного завантаження та відображення розкладу занять для користувача відповідно до його ролі (студент або викладач). Після ініціалізації екрану виконується асинхронне завантаження інформації про користувача, розклад занять на поточний та найближчі дні, а також визначення типу тижня (верхній чи нижній) для кожного дня окремо.

Для отримання даних про користувача, включно з роллю, використовується функція getUserInfo(). Код представлено в лістингу 3.21.

Лістинг 3.21 – Функція getUserInfo()

```
Future<Map<String, dynamic>?> getUserInfo() async {
    final start = DateTime.now().toIso8601String();
    print("[$start] Запит інформації про користувача
/v1/users/me");
    try {
      final response = await dio.get('/v1/users/me');
      print(
          "[${DateTime.now().toIso8601String()}] Користувач:
${response.data}");
      await storage.write('name', response.data['name']);
      await storage.write('surname', response.data['surname']);
      await storage.write('id', response.data['id']);
      await storage.write('role', response.data['role']);
      return response.data;
    } on DioException catch (e) {
      print(
```

```
          "[${DateTime.now().toIso8601String()}] Помилка при
отриманні користувача: ${e.message}");
      return null;}}
```

Після отримання даних генерується список днів для відображення, а також визначається тип тижня лише для тих днів, де в розкладі є пари з позначкою «верх» або «низ».

Розклад виводиться у вигляді горизонтального перегортання карток (PageView), кожна з яких відповідає окремому навчальному дню. Код представлено в лістингу 3.22.

Лістинг 3.22 – Виведення розкладу

```
PageView.builder(
  controller: _pageController,
  itemCount: _daysList.length,
  itemBuilder: (context, index) {
    final day = _daysList[index];
    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 3),
      child: DayScheduleCard(
        date: day,
        lessons: finalLessons,
        weekType: resolvedIsUpperWeek ? 'Верхній тиждень' :
'Нижній тиждень',
        role: _role,
      ),
    );
  },
)
```

Для побудови вмісту кожної картки використовується компонент DayScheduleCard, який уже описано раніше. У цьому компоненті враховується наявність замін, тип тижня, сортування пар за номером (classNumber), а також логіка приховування основних пар, які замінено (тобто якщо для певного номера пари є заміна – базова пара не показується). Це дозволяє формувати адаптивне і точне відображення розкладу, яке динамічно підлаштовується під конкретну дату та ситуацію в навчальному процесі.

Для методиста на головному екрані реалізовано панель керування розкладом, яка дозволяє йому вносити зміни в пари, змінювати періоди

навчання, прив’язувати їх до груп, тощо. Повний код реалізації представлено в додатку Р.

Сторінку відвідуваності створено таким чином, щоб в залежності від ролі користувача виводилося відповідне тіло для сторінки. Код реалізації наведено в лістингу 3.23.

Лістинг 3.23 – Виведення тіла сторінки в залежності від ролі

```
Widget _buildRoleSpecificBody() {
    if (_role == null) {
     return const Center(child: CircularProgressIndicator());
    }
    switch (_role) {
      case 'Monitor':
        return const MonitorAttendanceBody();
      case 'Student':
        return const StudentAttendanceBody();
      case 'Teacher':
        return TeacherAttendanceBody();
          case 'Admin':
        return AdminAttendanceBody();
      default:
  return const Center(child: Text('Роль не підтримується'));
    }
  }
```

Для студента створено тіло сторінки, де він може переглядати звіти про свою відвідуваність протягом поточного сестру. Для отримання даних з серверу реалізовано функцію fetchStudentAttendance. Код представлено в лістингу 3.24.

Лістинг 3.24 – Отримання відвідуваності студента

```
Future<List<FullAttendanceInfo>> fetchStudentAttendance() async{
    final start = DateTime.now().toIso8601String();
    print("[$start] Запит відвідуваності");
    try {
      final response = await dio.get(
        '/v1/users/me/attendance',
      );
      final end = DateTime.now().toIso8601String();
   print("[$end] Відвідуваність отримано ${response.data}");
     return FullAttendanceInfo.fromApiResponse(response.data);
    } on DioException catch (e) {
      print(
          "[${DateTime.now().toIso8601String()}] Помилка при
отриманні: ${e.message}");
  throw Exception('Failed to fetch: ${e.message}');}}
```

Аналогічно до функцій отримання розкладу, даний метод повертає список об’єктів класу FullAttendanceInfo.

Щоб ефективно виводити звіти про відвідуваність за місяцями, створено клас карток. Вона виводить отримані дані в вигляді табличок. Код представлено в лістингу 3.25.

Лістинг 3.25 – Картка відвідуваності студента

```
...summary.subjects.entries.map((entry) {
final subj = entry.key;
final data = entry.value;
return TableRow(
children: [
Padding(
padding: const EdgeInsets.symmetric(vertical: 4),
child: Text(subj),),
Padding(
padding: const EdgeInsets.symmetric(vertical: 4),
child: Text('${data.excused}H'),),
Padding(
padding: const EdgeInsets.symmetric(vertical: 4),
child: Text('${data.unexcused}H'], );}),
```

Використання функції в StudentAttendanceBody() представлено в лістингу 3.26.

Лістинг 3.26 – Використання в StudentAttendanceBody()

```
Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(
      body: FutureBuilder<List<MonthlyAttendanceSummary>>(
        future: _summaryFuture,
        builder: (context, snapshot) {
          if (snapshot.connectionState ==
ConnectionState.waiting){
            return const Center(child:
CircularProgressIndicator());}
else if (snapshot.hasError) {
            return Center(child: Text('Помилка:
${snapshot.error}'));}
else if (!snapshot.hasData || snapshot.data!.isEmpty) {
            return const Center(child: Text('Немає даних про
відвідуваність'));}
          final summaryList = snapshot.data!;
          return ListView.separated(
            itemCount: summaryList.length,
            padding: const EdgeInsets.symmetric(vertical: 12,
horizontal: 8),
```

```
            itemBuilder: (context, index) {
              return MonthlyAttendanceCard(summary:
summaryList[index]);
            },
            separatorBuilder: (context, index) => const
SizedBox(height: 16),);},),);}
```

Для старости реалізовано табличку відвідуваності, з можливістю додавати, редагувати та видаляти пропуски студентам своєї групи. За аналогічним принципом до попередніх функцій створено метод для отримання даних про групу, відвідуваність та пари на певний період, який повертає список об’єктів класу FullAttendanceGroupInfo. Код реалізації функції fetchMonitorAttendance представлено в лістингу 3.27.

Лістинг 3.27 – Функція “fetchMonitorAttendance”

```
Future<FullAttendanceGroupInfo> fetchMonitorAttendance(
      {DateTime? from, DateTime? to}) async {
    final start = DateTime.now().toIso8601String();
    print("[$start] Запит відвідуваності
/v1/monitor/me/attendance");
    try {
      final queryParams = <String, String>{};
      if (from != null) queryParams['from'] = _formatDate(from);
      if (to != null) queryParams['to'] = _formatDate(to);
      final response = await dio.get(
        '/v1/monitor/me/attendance',
        queryParameters: queryParams,
      );
      final end = DateTime.now().toIso8601String();
      print("[$end] Відвідуваність отримано ${response.data}");
      return FullAttendanceGroupInfo.fromMap(response.data);
    } on DioException catch (e) {
      print(
          "[${DateTime.now().toIso8601String()}] Помилка при
отриманні: ${e.message}");
      throw Exception('Failed to fetch: ${e.message}');
    }
  }
```

Для виведення та редагування таблиці відвідуваності у застосунку реалізовано компонент AttendanceTableView. Цей віджет відображає таблицю студентів із динамічною сіткою пар на вибрану дату, дозволяючи старості переглядати або змінювати статуси пропусків.

Під час побудови таблиці спочатку визначаються пари, що відбуваються у вибраний день, з урахуванням як замін, так і тижневого типу. Код реалізації представлено в лістингу 3.28.

Лістинг 3.28 – Визначення інформації для побудови таблиці

```
List<ScheduleItem> _getLessonsForDate(DateTime date) {
  final dateString = _formatDate(date);
  final weekday = date.weekday.toString();
  return widget.attendanceInfo.schedule.where((lesson) {
    if (lesson.date.isNotEmpty && lesson.date == dateString) {
      return true;
    }
    if (lesson.date.isEmpty && lesson.weekdayIndex == weekday) {
      if (lesson.weekType.isNotEmpty) {
        final isUpper = widget.weekTypeByDate[date] ?? true;
        if (lesson.weekType == "upper" && isUpper) return true;
        if (lesson.weekType == "lower" && !isUpper) return true;
        return false;
      }
      return true;
    }
    return false;}).toList();}
```

Далі будується двовимірна таблиця: перший стовпець це ПІБ студентів, наступні – пари, що відповідають за певну дату. Кожна клітинка містить випадаючий список, за допомогою якого можна встановити або змінити статус пропуску використовуючи функції віддалено серверу (додавання, видалення, оновлення). Код реалізації представлено в лістингу 3.29.

Лістинг 3.29 – Побудова таблиці

```
 DropdownButton<String>(
  value: currentStatus,
  items: [
    const DropdownMenuItem(value: '', child: Text('')),
    DropdownMenuItem(value: 'Н', child: Text('Н')),
    DropdownMenuItem(value: 'Н(поважна)', child: Text('Н')),
  ],
  onChanged: (value) async {
    final placedId = await storage.read('id');
    final scheduleId = lessons[i].scheduleId;
    final selectedDateStr = _formatDate(widget.selectedDate);
    final bool excusedBool = value == 'Н(поважна)';
    try {
      if (value == '' && record != null) {
        await apiClient.deleteAttendance(record.attendanceId);
      } else if (record != null) {
```

```
        await apiClient.updateAttendance(
          id: record.attendanceId,
          excused: excusedBool,);
      } else if (value != '') {
        await apiClient.insertAttendance(
          schedule_id: scheduleId,
          student_id: studentId,
          date_of_pass: selectedDateStr,
          excused: excusedBool,
          illness: false,);}
      widget.onDateChange?.call(widget.selectedDate);
    } catch (e) {
      print("Помилка при зміні відвідуваності: $e");}},),
```

Також у таблиці реалізовано можливість вибору дати через showDatePicker, яка оновлює дані таблиці після натискання.

Таким чином, AttendanceTableView забезпечує інтерактивну роботу з відвідуваністю студентів у рамках обраної дати, дозволяючи як перегляд, так і зміну пропусків безпосередньо у таблиці.

У MonitorAttendanceBody() реалізовано головний функціонал сторінки старости для перегляду та редагування таблиці відвідуваності студентів. На цій сторінці після завантаження даних автоматично будується таблиця відвідуваності на поточну дату, а також доступна кнопка для формування звіту за обраними періодами.

При ініціалізації викликаються функції _fetchAttendanceData та _fetchWeekType, які завантажують графік відвідуваності (FullAttendanceGroupInfo) та визначають поточний тип тижня. Формується список днів без вихідних, і для тих днів, де є пари з типом тижня, виконується перевірка верхнього/нижнього тижня. Ці дані передаються до AttendanceTableView, який будує інтерактивну таблицю відвідуваності.

Нижче таблиці розміщено кнопку «Сформувати звіт», що відкриває діалогове вікно, в якому можна обрати періоди для створення звіту та місяць для якого він формується, які потім передаються до API. Після натискання «Створити звіт», виконується запит. Код запиту та функції для нього представлено в лістингу 3.30.

Лістинг 3.30 – Код запиту та функції для нього

```
final data = await apiClient.fetchAttendanceReport(
  month: selectedMonthStr,
  periods: validPeriods,
);
await exportToExcel(context, data);
Future<AttendanceReportData> fetchAttendanceReport({
    required String month,
    required List<Map<String, String>> periods,
  }) async {
    try {
      final response = await dio.post(
        '/v1/monitor/me/attendance_report',
        data: {
          "month": month,
          "periods": periods,
        },
      );
      print('[REPORT SUCCESS] ${response.statusCode} →
${response.data}');
      return AttendanceReportData.fromJson(response.data);
    } catch (e) {
      print('[REPORT ERROR] $e');
      rethrow;
    }
  }
```

Для експорту даних отриманих з сервера в Excel, створено метод exportToExcel, який реалізує повний цикл експорту даних про відвідуваність студентів у файл формату .xlsx. Він створює табличну структуру, стилізує заголовки, заповнює значеннями, підраховує загальні та детальні пропуски, і зберігає файл у директорію завантажень пристрою. Повний код реалізації представлено в додатку С.

Таким чином, цей екран забезпечує повний цикл роботи з відвідуваністю, перегляд, редагування та експорт звітів. Він логічно поєднує AttendanceTableView для таблиці та PeriodsAndExportDialog для створення звітів, забезпечуючи інтерактивну та зручну взаємодію зі сторони старости.

Для викладачів було розроблено аналогічну структуру, проте у них в кутку таблиці є можливість обрати для якої групи вони хочуть заповнити відвідуваність, та відсутня можливість створювати звіти.

Для методиста на сторінці відвідуваності створено інтерфейс який дозволяє обирати спеціальність відділення, для якої потрібно зробити звіт про присутність на парах, після чого вибрати необхідні групи цієї спеціальності, місяць, та отримати відповідні дані з серверу для запису в файл Excel з відповідним форматуванням.

Для цього в класі RestAPIClient створено функцію fetchAdminAttendanceReport, яка приймає в себе список ідентифікаторів обраних груп та місяць для якого необхідно створити звіт, та передає їх в JSON-структурі як аргументи. Код реалізації представлено в лістингу 3.31.

Лістинг 3.31 – Функція fetchAdminAttendanceReport

```
Future<AdminAttendanceReportData> fetchAdminAttendanceReport({
    required String month,
    required List<Map<Int, Int>> groups,
  }) async {
    try {
      final response = await dio.post(
        '/v1/admin/attendance/report',
        data: {
          "groups": groups,
          "month": month,
        },
      );
      print('[REPORT SUCCESS] ${response.statusCode} →
${response.data}');
      return AdminAttendanceReportData.fromJson(response.data);
    } catch (e) {
      print('[REPORT ERROR] $e');
      rethrow;
    }
  }
```

За аналогією до інших методів цього класу, ця функція повертає масив об’єктів AdminAttendanceReportData, які формуються всередині процедури за допомогою вбудованого метода fromJson.

Таким чином було створено універсальну сторінку для роботи з відвідуваністю, яка підлаштовується під роль користувача, змінюючи доступний функціонал відповідно до його прав: староста має змогу переглядати, редагувати та експортувати дані; викладач – вносити пропуски

для обраної групи; методист – формувати звіти по кількох групах певної спеціальності за вибраний місяць, а студент переглядати статистику про свою відвідуваність.

Для студента розроблено сторінку яка дозволяє переглядати власні оцінки в зручному форматі електронних залікових книжок. Для цього використовується уже готовий метод API коледжу fetch_user_grades. В клієнті створено функцію для отримання та обробки цих даних. Код реалізації функції fetchGrades наведено в лістингу 3.32.

Лістинг 3.32 – Код функції fetchGrades

```
Future<List<FullGradeInfo>> fetchGrades() async {
    final start = DateTime.now().toIso8601String();
    print("[$start] Запит оцінок /v1/users/me/grades");
    try {
      final response = await dio.get('/v1/users/me/grades');
      if (response.data == null || response.data is! Map<String,
dynamic>) {
        final end = DateTime.now().toIso8601String();
        print("[$end] Отримано пусту відповідь або некоректний
формат даних");
        return [];
      }
      final end = DateTime.now().toIso8601String();
      print("[$end] Оцінки отримано");
      return FullGradeInfo.fromApiResponse(response.data);
    } on DioException catch (e) {
      print(
          "[${DateTime.now().toIso8601String()}] Помилка при
отриманні оцінок: ${e.message}");
      throw Exception('Failed to fetch grades: ${e.message}');
    }
  }
```

Отримані дані записують в список об'єктів класу FullGradeInfo за допомогою методу fromApiResponse.

Для ефективного виводу на сторінку створено клас GradesTable. Він приймає список об'єктів FullGradeInfo та виводить їх у вигляді таблиці з трьома стовпцями: назва предмета, викладач, та оцінка. Таблиця стилізована з округленими краями, рамками між комірками та автоматичним заповненням на основі переданого списку. У разі відсутності оцінок виводиться

повідомлення «Немає оцінок для цієї категорії». Це забезпечує компактне та зручне відображення результатів навчання в інтерфейсі. Повний код реалізації представлено в додатку Т.

Для відображення оцінок студента на використовується клас GradesScreen, який реалізує повноцінний інтерфейс для фільтрації та перегляду результатів навчання. Ключовою частиною цієї сторінки є функція _fetchUserInfo(), яка виконується при ініціалізації та відповідає за отримання даних з сервера. У межах цієї функції спочатку з SecureStorage зчитуються персональні дані користувача (ім’я та прізвище). Якщо вони ще не збережені, викликається метод getUserInfo() клієнта API для їх завантаження. Далі через метод fetchGrades() завантажуються всі доступні оцінки студента. Код реалізації функції _fetchUserInfo() наведено в додатку У.

Після цього з повного списку оцінок визначаються унікальні типи (gradeType), які використовуються для фільтрації. Також, якщо серед типів є «Залік», зберігається список семестрів, у яких студент має такі оцінки. З отриманих даних одразу генерується перший вибраний тип оцінок для стартового відображення на екрані. Зібрані дані записуються у відповідні поля стану, що дозволяє формувати фільтроване представлення у таблиці GradesTable.

Таким чином, екран GradesScreen забезпечує інтерактивний перегляд оцінок за категоріями (заліки, курсові, екзамени тощо) та, у випадку заліків, за семестрами. Зміна типу або семестру динамічно оновлює вміст таблиці, яка реалізована у вигляді окремого віджета GradesTable. Це дозволяє зручно аналізувати успішність студента в залежності від обраних параметрів.

Сторінка сповіщень реалізована з урахуванням розмежування доступу та відображення інформації відповідно до ролі користувача. Якщо користувач студент він має доступ лише до перегляду своїх сповіщень. Для цього реалізовано функцію в класі RestAPIClient fetchNotifications. Код представлено в лістингу 3.33.

Лістинг 3.33 – Функція fetchNotifications

```
Future<List<NotificationInfo>> fetchNotifications() async {
  final storage = SecureStorage();
  final apiClient = RestAPIClient(baseUrl, storage);
  final userId = await storage.read('user_id');
  final response = await
apiClient.dio.get('/v1/notifications/$userId');
  final List data = response.data;
  return data.map((e) =>
NotificationInfo.fromJson(e)).toList();}
```

Після отримання даних, вони перетворюються в список об’єктів NotificationInfo, який містить поля для ідентифікатора, тексту повідомлення, імені відправника, дати створення, статусу прочитаності та типу. На клієнті ці повідомлення виводяться у вигляді інтерактивного списку на окремій сторінці, де непрочитані повідомлення підсвічуються, а натискання на них дозволяє позначити як прочитані або відкрити деталі. Повний код реалізації сторінки сповіщень для студента представлено в додатку Ф.

Користувачі із роллю викладача мають можливість переглядати свої сповіщення аналогічно до студентів, та додатково можуть надсилати їх.

У клієнтській частині реалізовано функцію sendNotificationToServer, яка забезпечує передачу даних для сповіщення на сервер. Вона приймає вхідні параметри: deviceToken (токен отримувача) та body (текст повідомлення), формує відповідний JSON-запит та відправляє його через HTTP POST на адресу «/v1/notifications/send». Після надсилання функція обробляє відповідь сервера, повідомляючи про успішне надсилання або про виниклу помилку. Сервер у свою чергу здійснює надсилання push-сповіщення через FMC. Таким чином, клієнт ініціює сповіщення, а сам процес його доставки виконується на сервері. Повний код реалізації функції sendNotificationToServer представлено в додатку Х.

У клієнтському інтерфейсі передбачено окрему панель для надсилання сповіщень, яка дозволяє методисту або викладачу зручно керувати процесом інформування користувачів. На цій панелі користувач має змогу обрати, кому

саме буде адресовано повідомлення: одній конкретній особі (наприклад, студенту або викладачу) або одразу цілій групі. Інтерфейс містить випадаючий список з доступними групами, а також поле пошуку або вибору користувача за ПІБ. Після вибору отримувачів, заповнюється поле «Текст повідомлення», після чого натисканням кнопки «Надіслати» сповіщення передається на сервер. Такий підхід дозволяє реалізувати швидке та адресне інформування, що особливо важливо для адміністративних оголошень, термінових повідомлень або індивідуальних зауважень. Повний код реалізації представлено в додатку Ц.

Для всіх користувачів створено однакову сторінку профілю, яка виводить особисті дані аккаунту користувача, та надає можливість змінювати тему застосунку. Повний код реалізації сторінки профілю представлено в додатку Ш.

У процесі розробки клієнтської частини системи «е-Студент» було реалізовано повноцінний функціонал роботи зі сповіщеннями, оцінками, розкладом та відвідуваністю, що адаптується під різні ролі користувачів (студент, викладач, староста, методист). Сторінки динамічно змінюють свій вигляд і доступні дії залежно від прав доступу, що забезпечує персоналізований та зручний інтерфейс.

3.3 Тестування системи «е-Студент»

Мета тестування даної системи полягає у всебічній перевірці коректності реалізації основних сценаріїв використання для всіх типів користувачів, передбачених у застосунку, зокрема студентів, викладачів, старост та методистів. У процесі тестування перевіряється не лише функціональність кожного модуля окремо, але й їх взаємодія в межах єдиного середовища. Особлива увага приділяється стабільності роботи інтерфейсу, швидкості реакції системи, правильному відображенню даних залежно від ролі користувача, а також точності передачі інформації між клієнтською та

серверною частинами. Це дозволяє переконатися в надійності застосунку, його здатності обробляти типові та нестандартні сценарії поведінки користувачів без збоїв чи втрати даних.

Для того щоб провести повне тестування всіх можливостей застосунку, першим кроком стала перевірка функціоналу студента. Було здійснено вхід у систему під заздалегідь зареєстрованим акаунтом з роллю “студент”, що дозволило відтворити реальний сценарій використання (рис. 3.1).

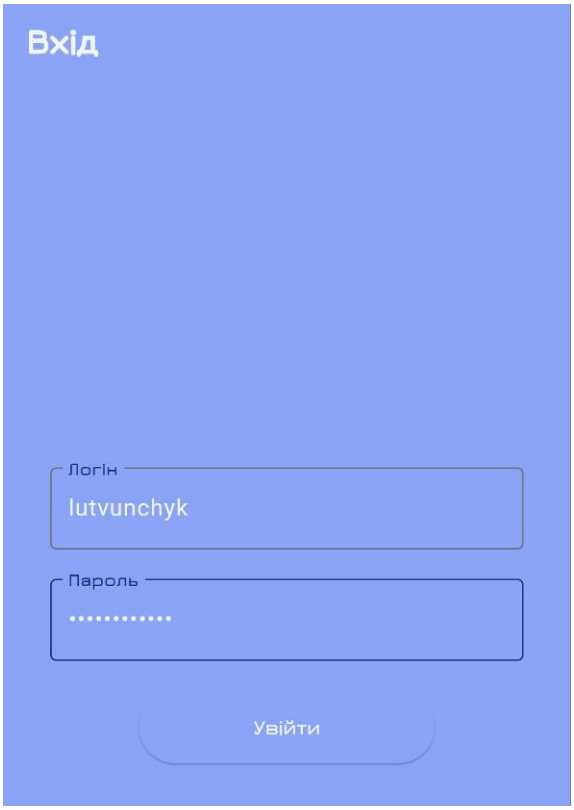

Рисунок 3.1 – Вхід в застосунок

Після входу в застосунок користувача було перенаправлено на головну сторінку, де коректно відображається розклад пар на певний період. Перша картка це завжди заняття на поточну дату. Було перевірено правильність відображення не постійних пар на заміну – бейдж «заміна» (рис. 3.2).

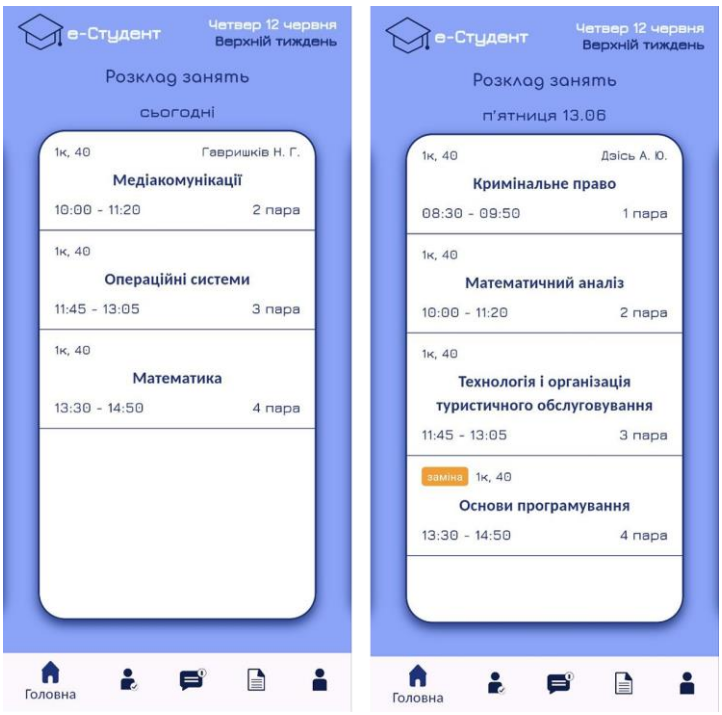

Рисунок 3.2 – Головна сторінка студента

Наступний крок перевірки – правильність відображення сторінки відвідуваності студента (рис. 3.3).

Рисунок 3.3 – Сторінка відвідуваності

Дані в системі правильно сортуються за відповідними предметами та згруповані за місяцями, що дозволяє користувачам зручно аналізувати успішність та відвідуваність у хронологічному порядку. У процесі тестування було перевірено точність підрахунку пропусків –система правильно розділяє їх на ті, що були здійснені з поважних причин (наприклад, хвороба або офіційна відпустка) та ті, що не мають виправдання, що особливо важливо для подальшого аналізу академічної дисципліни студентів та формування звітності.

Окремо протестовано функціонування підсистеми сповіщень. Було перевірено правильність їхнього відображення на інтерфейсі користувача. На рисунку 3.4 представлено інтерфейс сторінки зі списком усіх сповіщень, що отримав користувач. Також продемонстровано сторінку з детальною інформацією про конкретне повідомлення та приклад вхідного push-сповіщення, яке автоматично з'являється при надходженні нової інформації. Це дозволяє користувачам оперативно реагувати на важливі події або зміни в навчальному процесі.

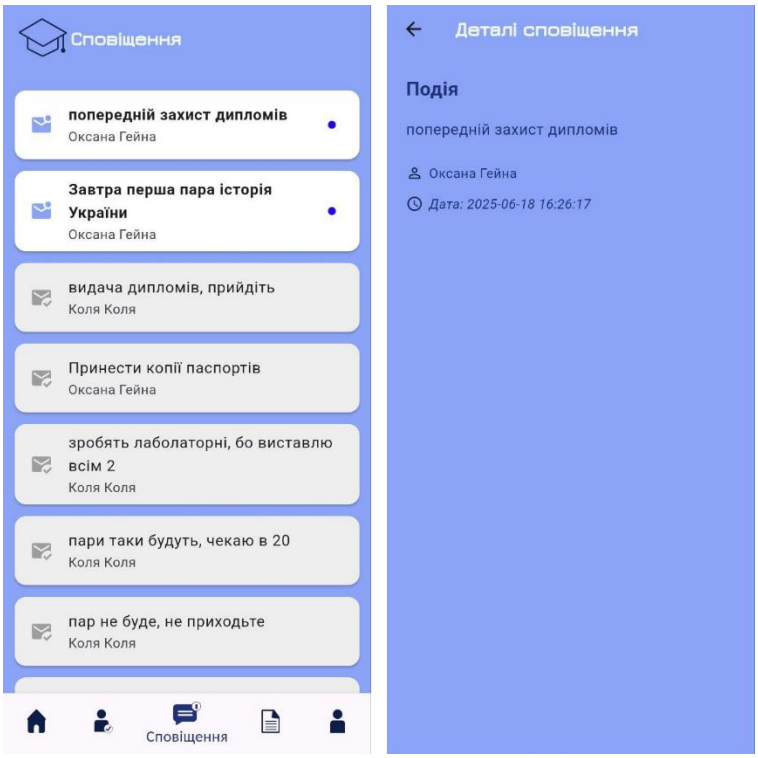

Рисунок 3.4 – Тестування сторінки сповіщень

Після цього було перейдено на сторінку “Оцінки”, на якій перевірено відображення усіх заліків, їх фільтрацію за семестрами та оцінки екзаменів (рис. 3.5).

Рисунок 3.5 – Тестування відображення оцінок

Також було протестовано відображення темної теми (рис. 3.6).

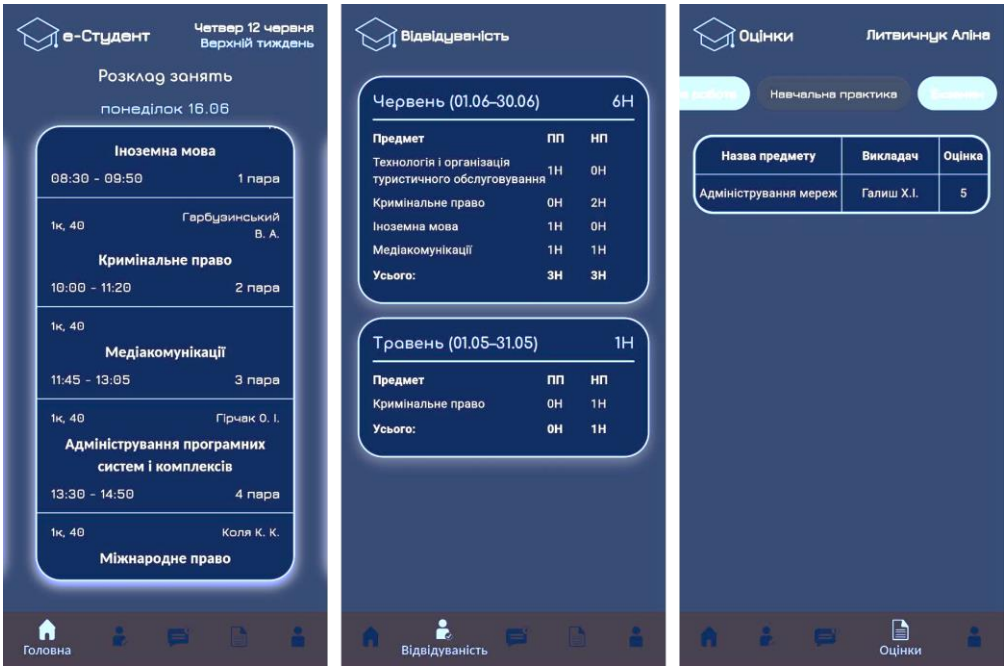

Рисунок 3.6 – Відображення темної теми

Також було протестовано оновлення токенів при закінчені терміну дії токену доступу (рис. 3.7).

```
flutter: [2025-06-12T06:59:56.253049] Додавання access токена до запиту
flutter: [2025-06-12T06:59:56.298843] Спроба оновлення токенів
flutter: [2025-06-12T06:59:56.572120] Збереження токенів
flutter: [2025-06-12T06:59:56.578062] Спроба оновлення токенів
[log] Value saved for key: access_token
[log] Value saved for key: refresh_token
[log] Value saved for key: access_token_expiry
flutter: [2025-06-12T06:59:56.688457] Спроба оновлення токенів
flutter: [2025-06-12T06:59:56.690353] Токени оновлено
[log] Value saved for key: refresh_token_expiry
```

Рисунок 3.7 – Оновлення токенів

Логи застосунку під час тестування показують, що оновлення токенів працює справно.

Після тестування функцій студента, було здійснено вхід в акаунт користувача з роллю старости. Перейдено на сторінку “Відвідуваність”. Спочатку відображається таблиця біля поточної дати, де можна обрати інший день для заповнення пропусків. Додано нові пропуски (рис. 3.8).

Рисунок 3.8 – Тестування відвідуваності

Введені в застосунку дані були успішно передані на сервер та записані в базу даних.

Також було протестовано створення щомісячного звіту відвідуваності. На рисунку 3.9 зображено процес вибору періодів для формування звіту, та повідомлення про успішне його створення і шлях до файлу .xlsx.

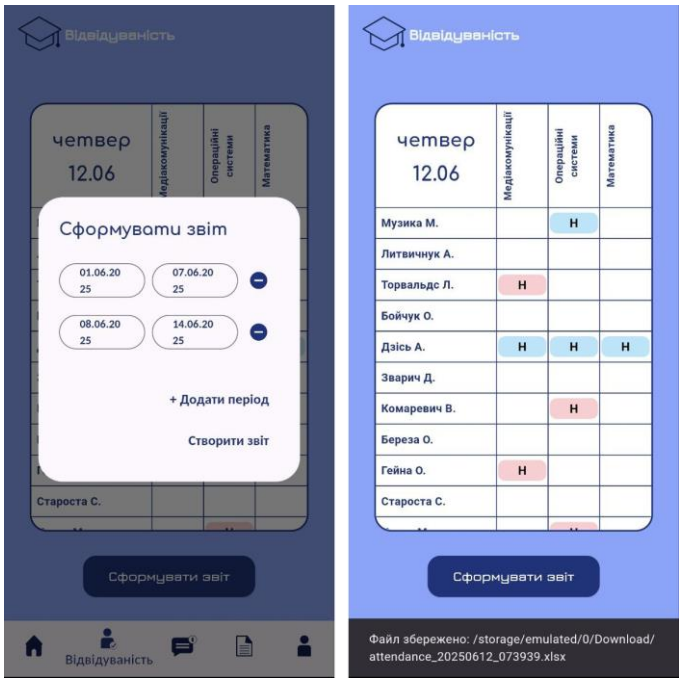

Рисунок 3.9 – Формування звіту відвідуваності

Звіт було правильно сформовано та завантажено на пристрій (рис. 3.10).

ВІДОМІСТЬ
обліку відвідувань студентами групи **КН-41**
спеціальності "Комп'ютерні науки"
за червень місяць 2025 р.

№ пп	Прізвище, ім'я, по батькові студента	Дати тижнів		Пропущено занять			
		01.06-07.06	08.06-14.06	Всього	з них		
					по хворобі	з поважних причин	без поважних причин
1	Гейна Оксана Іванівна	2	8	**10**		4	6
2	Дзісь Артем Юрійович	0	6	**6**		6	0
3	Комаревич Владислав Ярославович	0	2	**2**		0	2
4	Литвичнук Аліна Петрівна	4	8	**12**	2	4	6
5	Музика Михайло Андрійович	0	2	**2**		2	0
6	Пелех Мар'ян	0	2	**2**		0	2
7	Староста Староста Староста	0	10	**10**		4	6
8	Торвальдс Лінус Лінукс	0	2	**2**		0	2
Разом:				**46**	**2**	**20**	**24**

Староста групи _____Аліна ЛИТВИНЧУК

Куратор групи_____Надія ГАВРИШКІВ

Рисунок 3.10 – Створений звіт

Після тестування основних функцій старости, було здійснено вхід в акаунт викладача. На першій сторінці відображається персональний розклад, бейджи з типом пар відображаються правильно (рис. 3.11).

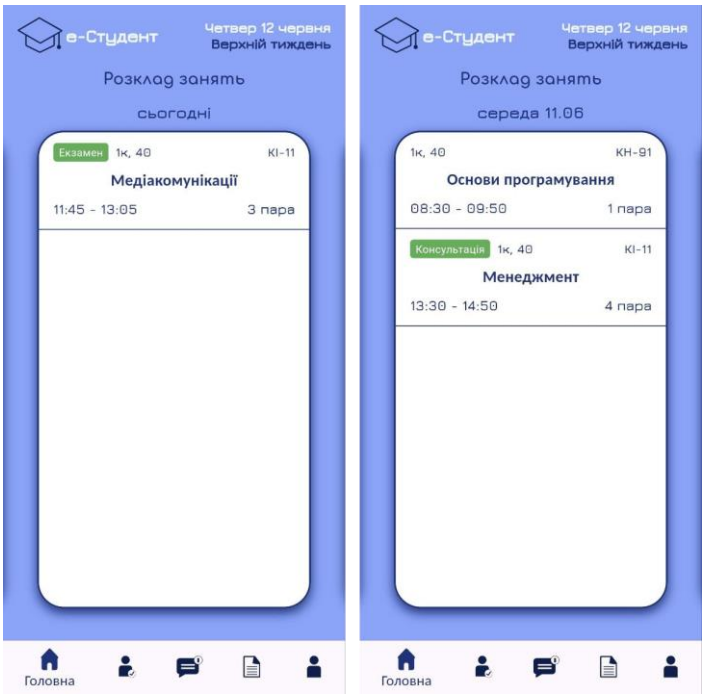

Рисунок 3.11 – Розклад викладача

Аналогічно як для старости було перевірено заповнення відвідуваності, додано пропуски для груп «КН-41» та «КІ-11» (рис. 3.12).

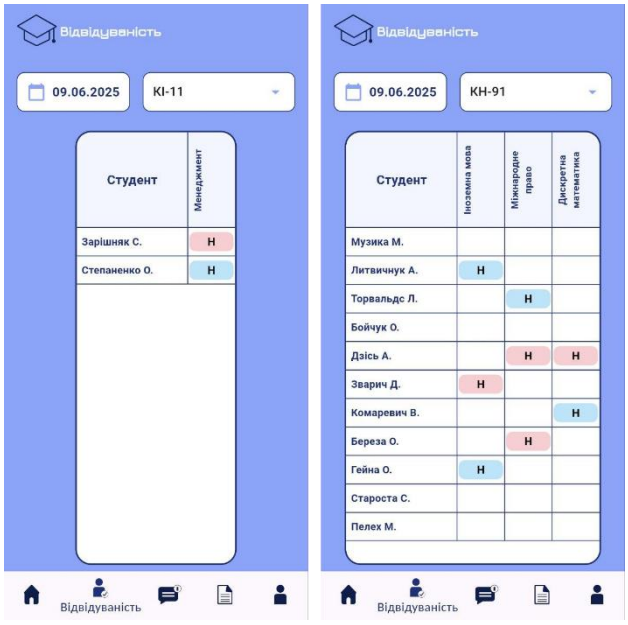

Рисунок 3.12 – Заповнення відвідуваності викладачем

Наступним етапом перевірено надсилання сповіщень студентам: створено сповіщення про проведення тесту на парі програмування та надіслано його групі «КН-41» (рис. 3.13).

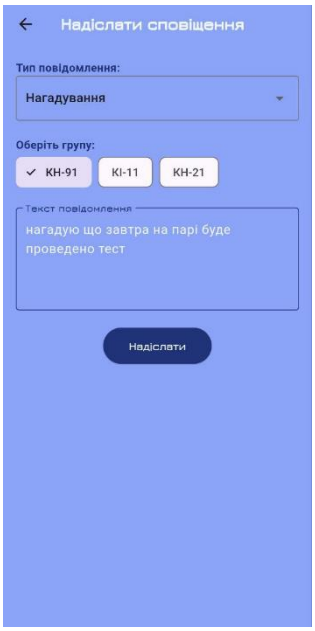

Рисунок 3.13 – Надсилання сповіщення студентам

Перевірено функції методиста. Після входу в акаунт з відповідною роллю, на головній сторінці внесено зміни в розклад для групи “КІ-11”: додано пару на заміну (рис. 3.14).

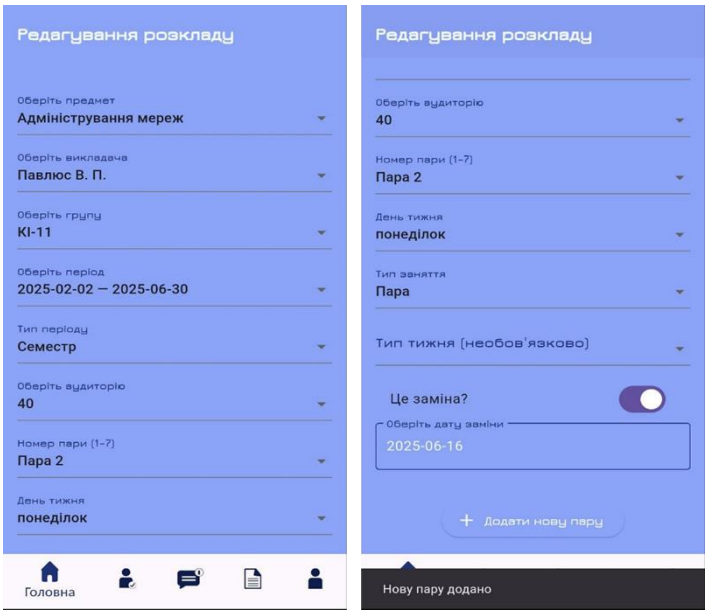

Рисунок 3.14 – Внесення змін в розклад

Дані успішно змінилися в базі даних.

Також було створено звіт відвідуваності для всіх груп спеціальності “Комп’ютерні науки” за червень. На рисунку 3.15 зображено процес створення звіту.

Рисунок 3.15 – Формування звіту відвідуваності

Звіт створено правильно та завантажено локально на пристрій користувача (рис. 3.16).

1	КН-91 — червень		
2			
3	ПІБ студента	всього	без п/п
4	Береза Олександр Андрійович	2	2
5	Бойчук Олександр Олександрович	5	4
6	Гейна Оксана Іванівна	8	4
7	Дзісь Артем Юрійович	7	6
8	Зварич Денис Віталійович	3	1
9	Комаревич Владислав Ярославович	4	2
10	Литвичнук Аліна Петрівна	7	4
11	Музика Михайло Андрійович	2	0
12	Пелех Мар'ян	0	0
13	Староста Староста Староста	4	2
14	Торвальдс Лінус Лінукс	3	1
15	РАЗОМ:	45	26
16			
17			
18			

Sheet1 КН-91 КН-21

Рисунок 3.16 – Документ звіту відвідуваності

Перевірено роботу сповіщень: надіслано групі “КІ-11” повідомлення про зміни в розкладі (рис. 3.17).

Рисунок 3.17 – Надсилання сповіщення студентам

Проведене тестування підтвердило стабільність, точність та функціональну відповідність системи поставленим вимогам. Таким чином, розроблений застосунок є ефективним засобом цифровізації навчального процесу.

У результаті реалізації було створено сучасний клієнт-серверний застосунок, призначений для підтримки та оптимізації освітнього процесу. Система охоплює широкий спектр функцій, адаптованих до різних типів користувачів, забезпечуючи зручний доступ до ключових елементів навчальної діяльності, таких як розклад, облік відвідуваності, оцінювання та сповіщення. Клієнтська частина надає користувачам зручний інтерфейс і можливість взаємодії згідно з їхньою роллю, тоді як серверна частина відповідає за обробку запитів, збереження інформації та забезпечення безперебійної взаємодії між компонентами системи. Реалізовані рішення спрямовані на підвищення ефективності управління навчальним процесом та покращення загального користувацького досвіду.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Кросплатформений застосунок «е-Студент» є економічно доцільним інструментом цифрової трансформації освітнього процесу в Галицькому фаховому коледжі. Завдяки використанню фреймворків Flutter та Pistache у клієнт-серверній архітектурі, система забезпечує зниження витрат на обслуговування та розробку для кількох платформ [13], оптимізує використання людських ресурсів та часу, а також сприяє підвищенню ефективності управління навчальним процесом за рахунок автоматизації рутинних операцій та оперативного обміну інформацією.

Продукт має ряд техніко-економічних та експлуатаційних характеристик, що роблять його гідним конкурентом на ринку схожих продуктів. Застосунок забезпечує повну автоматизацію розкладу занять, системи обліку відвідуваності, та можливість легко сповіщати учасників освітнього процесу про різноманітні події та зміни. Однією з ключових переваг системи є можливість швидко формувати звітність, що економить час працівникам, та є надійнішим за ручні методи. Важливим аспектом також є потенційна масштабованість та адаптивність розробленого продукту, що дозволяє в майбутньому розширювати наявний функціонал, додаючи більше можливостей для студентів та викладачів. Кросплатформеність застосунку забезпечує зручний доступ до системи всім учасникам навчального процесу незалежно від використовуваних ними пристроїв.

Застосунок «е-Студент» не є абсолютно новою концепцією, проте він є унікальним, тому що розроблений цілеспрямовано для потреб Галицького фахового коледжу. На відміну від уже існуючих комерційних рішень як наприклад «Мрія», які підлаштовані під школи, система враховує специфічні внутрішні процеси закладу, особливості його структури, організацію навчального процесу. Такий індивідуальний підхід забезпечує максимальну

відповідність характеристик застосунку реальним потребам його користувачів та підвищує ефективність його впровадження.

На даному етапі єдиним покупцем користувачем продукту є Галицький фаховий коледж, який був замовником цієї системи з метою підвищення ефективності навчального процесу. Проте, існує перспектива адаптації цього застосунку для впровадження інших навчальних закладах України.

В сучасному світі, де існує попит на автоматизовані системи управління освітнім процесом, багато навчальних закладів вищої та передвищої освіти виявляють зацікавленість у локальних рішеннях, які можуть бути адаптовані до їх специфічних потреб, на відміну від комерційних та універсальних систем, що часто потребують додаткової модифікації бізнес-процесів перед впровадженням.

Подальша підтримка застосунку після його впровадження повинна забезпечити стабільність роботи та досягнення максимальної ефективності використання. Для цього перед запуском передбачено проведення початкового навчання персоналу та здобувачів освіти закладу, а також подальші консультації при виникненні питань. Важливим аспектом є періодичні оновлення системи для виправлення виявлених помилок, підвищення безпеки та впровадження нового функціоналу відповідно до потреб користувачів.

Підсумовуючи результати аналізу ринку збуту, можна зробити висновок, що застосунок «е-Студент» має значний потенціал впровадження як локально в Галицькому фаховому коледжі, так і на регіональному ринку освітніх закладів. Розроблений продукт відповідає сучасним технологічним стандартам, має можливість легкого масштабування та розширення функціоналу, та може бути адаптований під потреби замовника. Хоча початковим фокусом є задоволення потреб конкретного навчального закладу, подальший розвиток системи можуть забезпечити його успішне впровадження в інших навчальних закладах.

4.2 Розрахункова частина

Для розрахунку загальних витрат на розробку застосунку використано формулу «Срозр = Спраця + Сел + Сінш», де «Спраця» це витрати на заробітну плату працівникам та витрати податки, «Сел» – витрати на електроенергію, що була використана в процесі розробки та «Сінш» – інші витрати, наприклад, канцелярія, витрати на зв’язок та комунальні послуги.

Нижче наведено детальні розрахунки усіх складових формули.

Найбільшу частку витрат на розробку проєкту є оплата праці спеціалістів, що над ним працюють. Заробітна плата обчислюється на кількості фактично відпрацьованих годин кожним учасником проєкту та погодинних ставок, визначених відповідно до середньоринкових показників для фахівців відповідних кваліфікацій в ІТ сфері в Україні станом на 2025 рік.

Для розрахунку витрат на заробітну плату було складено таблицю 4.1 із працівниками, їх погодинними ставками та годинами роботи.

Таблиця 4.1 – Заробітна плата учасників проєкту

Посада	Погодинна ставка	Кількість пропрацьованих годин	Отримана заробітна плата
UI/UX дизайнер	461 грн/год	25 год.	11525 грн
Розробник	653 грн/год	480 год.	313440 грн
Тестувальник	220 грн/год	30 год	6600 грн
Всього			313565 грн

Таким чином, загальні витрати на заробітну плату складають 313565 грн. Однак, окрім безпосередньої оплати праці, необхідно враховувати обов'язкові податкові відрахування.

В Україні роботодавець обов’язково має сплачувати єдиний соціальний внесок (ЄСВ) від нарахованої заробітної плати працівникам [14]. Станом на 2025 рік він становить 22% від нарахованої заробітної плати. Також обов’язково має бути сплачено ПДФО (18%) та військовий збір, що становить

5%. Для зручності складено таблицю 4.2 для підрахунку заробітної плати разом з податками для 3 працівників проєкту

Таблиця 4.2 – Витрати на податки

Посада	Отримана ЗП	ЄСВ	ПДФО	Військовий збір	Всього податків
UI/UX дизайнер	11525 грн	2535.5 грн	2074.5 грн	576.25 грн	5186.25 грн
Розробник	313440 грн	68956.80 грн	56419.20 грн	15672 грн	141048 грн
Тестувальник	6600 грн	1452 грн	1188 грн	330 грн	2970 грн
Всього					149204.25 грн

Відповідно до розрахунків, всього на податки витрачено 149204.25 грн, отже Спраця = 313565 + 149204.25 = 462769.25 грн.

Для розрахунку витрат на електроенергію було обчислено час роботи основних пристроїв, задіяних у процесі розробки, та їх енергоспоживання.

В процесі розробки проєкту використовувався комп’ютер, 2 ноутбука та монітор, для розрахунку витраченої електроенергії було створено таблицю 4.3.

Таблиця 4.3 – Витрати на електроенергію

Пристрій	Кількість	Потужність	Час роботи	Вартість
Комп’ютер	1	170 вт/год	305 год	224 грн
Монітор	1	50 вт/год	305 год	66 грн
Ноутбук	2	75 вт/год	205 год	198 грн
Всього				488 грн

Під час розробки проєкту крім основних витрат розрахованих вище також можуть бути інші незначні затрати, які впливають на кінцеву ціну продукту. Витрачені кошти на цю категорію можна розрахувати за формулою

«Сішн = Сканц + Сзв + Ском», де «Сканц» це витрати на канцелярію, «Сзв» – витрати на зв’язок та інтернет, та «Ском» – витрати на комунальні послуги. Відповідно до розрахунків – Сінш = 300 + 900 + 2000 = 3100 грн.

Отже, враховуючи всі розраховані витрати на розробку, Срозр = 462769.25 + 488 + 3100 = 466357.25 грн.

Впровадження системи орієнтоване на спрощення навчального процесу в більшості для студентів, що не несе економічного ефекту, проте застосунок передбачає переведення багатьох паперових методів обліку в електронний.

Розроблена система «е-Студент» має подвійне призначення: по-перше, вона створена для підвищення якості навчального процесу та зручності студентів, що не має прямого економічного виміру, але є важливим фактором конкурентоспроможності навчального закладу; по-друге, система передбачає автоматизацію численних адміністративних процесів, які раніше виконувались вручну або з використанням паперових носіїв інформації. Саме цей аспект дозволяє досягти вимірюваного економічного ефекту.

Можливість вести облік відвідуваності студентів в електронному форматі усуває потребу закупки журналів обліку роботи академічних груп. В цьому випадку економічний ефект «Е» можна обрахувати за формулою «Е = Варт * Кгруп”, де “Варт» – це вартість одного журналу, а «Кгруп» – кількість груп в освітньому закладі. Тому Е = 80 * 65 = 5200 грн/рік.

Також цей модуль скорочує час створення звітів відвідуваності для методистів відділень. Економічний ефект в цьому випадку можна розрахувати за формулою «Е= (Тручн − Тавт) · Ргод · N», де «Тручн» це час який використовувався на виконання завдання вручну, відповідно, «Тавт» – на виконання після автоматизації, «Ргод» – середня вартість години часу та «N» – кількість повторень в рік.

До впровадження автоматизованої системи створення звітів займало в середньому 60 хвилин. Після впровадження застосунку «е-Студент» цей процес скоротився до одної хвилини. У закладі освіти працює шість

методистів, які отримують 40 грн за годину. Формування таких звітів здійснюється щомісяця протягом навчального року, тобто дев’ять разів на рік. Відповідно, Е = ((60 – 1) · 40 · 9)*6 = 127440 грн.

Враховуючи два вищеперераховані економічних ефекти, Езагал = 5200 + 127440 = 132640 грн/ рік.

Окупністю проєкту називають час, за який витрати на розробку системи та її впровадження будуть компенсовані економічним ефектом. Окупність показує наскільки швидко розроблений продукт зможе виправдати вкладені в нього інвестиції.

Термін окупності «Ток» можна визначити за формулою «Ток = Срозр / Е», де «Е» це економічний ефект, а «Срозр» – витрачені кошти на розробку.

Відповідно до попередніх розрахунків, Ток = 466357.25 / 132640 = 3,5 роки.

Термін окупності в три з половиною роки демонструє, що розробка та впровадження застосунку «е-Студент» є економічно доцільним рішенням для Галицького фахового коледжу імені В'ячеслава Чорновола. Після проходження точки окупності система продовжуватиме генерувати економічні вигоди, забезпечуючи позитивний фінансовий результат для навчального закладу.

4.3 Обґрунтування необхідності розробки

Сучасний освітній процес вимагає впровадження інноваційних технологічних рішень, здатних трансформувати та оптимізувати взаємодію між усіма учасниками навчального середовища. Кросплатформений застосунок «е-Студент» був розроблений як комплексна відповідь на актуальні потреби Галицького фахового коледжу імені В'ячеслава Чорновола в контексті діджиталізації ключових елементів освітнього процесу.

Головною метою створення даного застосунку є системне підвищення якості освітнього середовища та формування єдиного інформаційного простору для студентів, викладачів та адміністрації навчального закладу. Для студентів система забезпечує оперативний доступ до розкладу занять, поточних оцінок, важливих сповіщень та даних щодо власної відвідуваності. Викладацький склад отримує ефективний інструментарій для оцінювання, моніторингу присутності студентів та комунікації через систему оголошень. Адміністративний персонал коледжу має можливість гнучко керувати розкладом, здійснювати комплексний контроль освітніх процесів та автоматизувати формування різноманітної звітності.

Впровадження застосунку «е-Студент» значно зменшує кількість непорозумінь між учасниками навчального процесу через чітку фіксацію та своєчасне оновлення інформації в єдиній системі. Комунікація між студентами та викладачами набуває нової якості завдяки додатковим цифровим каналам взаємодії, що функціонують паралельно з традиційними формами спілкування.

Система сповіщень у реальному часі забезпечує оперативне реагування на різноманітні зміни в навчальному процесі, будь то коригування розкладу, призначення консультацій чи важливі організаційні повідомлення.

Ці переваги формують важливий соціальний ефект, що безпосередньо впливає як на якість освітніх послуг, так і на мотивацію здобувачів освіти до навчання через створення сучасного, технологічно прогресивного освітнього середовища.

Незважаючи на необхідність певних одноразових капітальних вкладень у розробку, впровадження та початковий супровід системи, ефективність проєкту проявляється у декількох взаємопов'язаних напрямках. Освітній ефект полягає у суттєвому покращенні організаційних аспектів навчального процесу, мінімізації бюрократичних процедур та скороченні обсягів ручної праці.

Соціальний ефект реалізується через зниження рівня стресу у студентів завдяки своєчасній поінформованості щодо всіх аспектів навчального процесу та покращення якості комунікації між усіма зацікавленими сторонами. Організаційний ефект досягається шляхом централізації та уніфікації даних, що забезпечує цілісність інформаційного середовища закладу та автоматизує численні внутрішні процеси коледжу.

Додатковим, але не менш важливим є економічний ефект від впровадження системи, що виражається у значному зменшенні витрат на паперову документацію, суттєвому скороченні часових витрат на формування різноманітної звітності та оптимізації робочого навантаження адміністративного персоналу коледжу за рахунок автоматизації рутинних операцій.

Таким чином, застосунок «е-Студент» являє собою не просто технічну розробку, а комплексний інструмент системного вдосконалення якості освітнього процесу, що повністю відповідає сучасним вимогам до фахової передвищої освіти. Впровадження даної системи створює міцний фундамент для подальшої цифрової трансформації освітнього середовища коледжу та підвищення його конкурентоспроможності на ринку освітніх послуг.

ВИСНОВКИ

У даній кваліфікаційній роботі реалізовано поставлену мету – створення мобільного застосунку «е-Студент», призначеного для забезпечення ефективної цифрової взаємодії між усіма учасниками освітнього процесу в межах одного інформаційного середовища. В межах проєкту виконано повний цикл розробки: від аналізу аналогів і потреб користувачів до реалізації клієнт-серверної архітектури, тестування системи та підготовки її до впровадження. Усі визначені завдання були виконані в повному обсязі.

Застосунок забезпечує доступ до персоналізованої інформації про розклад, оцінки, відвідуваність і сповіщення, адаптуючи інтерфейс і функціонал відповідно до ролі користувача: студент, викладач, староста або методист. Студенти отримують доступ до індивідуального навчального плану та статистики; старости можуть редагувати пропуски та формувати звіти; викладачі – реєструвати відвідуваність; методисти – отримувати узагальнені звіти по групах. В системі передбачено авторизацію, розмежування прав доступу та захист переданих даних, що підвищує рівень безпеки.

Наступні етапи розвитку проєкту включають впровадження додаткових функцій, таких як: впровадження автоматизованого генератора розкладу, адаптацію клієнтської частини під веб та ПК, інтеграцію особистого щоденника завдань для студентів з нагадуваннями та дедлайнами, а також створення вбудованої бібліотеки навчальних матеріалів. Ці доповнення дозволять перетворити «е-Студент» у ще більш повноцінну цифрову платформу, здатну покривати широкий спектр потреб учасників навчального процесу.

Таким чином, застосунок «е-Студент» може бути використаний для подальшого вдосконалення цифрової інфраструктури закладів фахової передвищої освіти та створення сучасного навчального середовища, яке відповідає сучасним вимогам часу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мрія – державна освітня екосистема для учнів, батьків і вчителів. Мрія – державна освітня екосистема : вебсайт. URL: https://mriia.gov.ua/app (дата звернення: 10.02.2025).

2. SMART університет. SMART Університет : вебсайт. URL: https://smart.wunu.edu.ua (дата звернення: 10.02.2025).

3. Flutter - Build apps for any screen. Flutter - Build apps for any screen : вебсайт. URL: https://flutter.dev (дата звернення: 18.02.2025).

4. Повний огляд REST: нюанси, поради, приклади. DOU : вебсайт. URL: https://dou.ua/forums/topic/50364/ (дата звернення: 18.02.2025).

5. Morast A. Building an API in C++ with pistache. Medium : вебсайт. URL: https://levelup.gitconnected.com/building-an-api-in-c-with-pistache-413247535fd3 (дата звернення: 28.02.2025).

6. Firebase Cloud Messaging | Send notifications across platforms. Firebase : вебсайт. URL: https://firebase.google.com/products/cloud-messaging (дата звернення: 28.02.2025).

7. Реляційні бази даних: структура та застосування у практиці. FoxmindEd : вебсайт. URL: https://foxminded.ua/reliatsiini-bazy-danykh/ (дата звернення: 28.02.2025).

8. UUID | MariaDB Documentation. MariaDB Enterprise Open Source Database | MariaDB : вебсайт. URL: https://mariadb.com/kb/en/uuid/ (дата звернення: 28.02.2025).

9. JSON web token introduction jwt.io. JSON Web Tokens - jwt.io : вебсайт.
URL: https://jwtio.translate.goog/introduction?_x_tr_sl=en&_x_tr_tl=uk&_x_tr_hl=uk&_x_tr_pto=sc (дата звернення: 18.03.2025).

10. Asynchronous programming: futures, async, await. Dart programming language | Dart : вебсайт. URL: https://dart.dev/libraries/async/async-await (дата звернення: 18.03.2025).

11. Kumar R. Working with flutter secure storage in a project. structure to follow with bloc. Medium : вебсайт. URL: https://medium.com/@rk0936626/working-with-flutter-secure-storage-in-a-project-structure-to-follow-with-bloc-9b65e8e55138 (дата звернення: 18.03.2025).

12. Singleton Method Design Pattern. GeeksforGeeks: вебсайті. URL: https://www.geeksforgeeks.org/system-design/singleton-design-pattern/ (дата звернення: 28.03.2025).

13. Flutter app development cost: features & pricing. Product Engineering Company | IoT App Development Company | HashStudioz Technologies : вебсайт. URL: https://www.hashstudioz.com/blog/flutter-app-development-cost-features-pricing/ (дата звернення: 30.03.2025).

14. Розрахунок заробітної плати у 2025 році: формули, нарахування, податки, приклади, "на руки" працівникам. Онлайн-бухгалтерія SMARTFIN.UA: зарплата, кадри, звіти, облік ФОП : вебсайт. URL: https://smartfin.ua/page/rozrakhunok-zarobitnoyi-platy-u-2025-rotsi (дата звернення: 28.04.2025).

15. Литвинчук А. Реалізація асинхронної взаємодії клієнта з Restful API сервером у кросплатформеному середовищі Flutter. МАТЕРІАЛИ СТУДЕНТ НАУКОВО-ПРАКТ. КОНФ., м. Тернопіль, 15 травня 2025 р. Тернопіль, 2025.

ДОДАТКИ

Додаток А

Лістинг функції fetch_user_shedule

```
void RestAPI::fetch_user_shedule(const Rest::Request &request,
Http::ResponseWriter response)
{
     std::string user_id =
request.param(":id").as<std::string>();
    std::string from = request.query().get("from").value_or("");
    std::string to = request.query().get("to").value_or("");
    if (user_id == "me") {
        const auto auth_header =
request.headers().tryGet<Http::Header::Authorization>();
        if (!auth_header) {
            response.send(Http::Code::Unauthorized,
"Unauthorized: Invalid token.\n");
            return;
        }
        const std::string session_id = jwt::decode(auth_header-
>value().substr(7)).get_payload_claim("session_id").as_string();
        const auto session_info = Database::getSession(conn,
session_id);
        if (!session_info.has_value()) {
            response.send(Http::Code::Not_Found, "NotFound:
Invalid session");
            return;
        }
        user_id = session_info->user_id;
    }
    std::unique_ptr<sql::PreparedStatement> group_stmt(conn-
>prepareStatement(R"(
        SELECT group_id FROM `users_groups` WHERE user_id = ?
    )"));
    group_stmt->setString(1, user_id);
    std::unique_ptr<sql::ResultSet> group_res(group_stmt-
>executeQuery());
    if (!group_res->next()) {
        response.send(Http::Code::Not_Found, "User group not
found");
        return;
    }
    int group_id = group_res->getInt("group_id");
    std::unique_ptr<sql::PreparedStatement> subgroup_stmt(conn-
>prepareStatement(R"(SELECT subgroup_id FROM `user_subgroups`
WHERE user_id = ? )"));
 subgroup_stmt->setString(1, user_id);
```

```
std::unique_ptr<sql::ResultSet> subgroup_res(subgroup_stmt
>executeQuery());

    if (!subgroup_res->next()) {
        response.send(Http::Code::Not_Found, "User subgroup not
found");
        return;}
    int subgroup_id = subgroup_res->getInt("subgroup_id");
    std::string sql = R"(
        SELECT
s.id,s.group_id,s.subject_id,s.subgroup_id,s.auditorium_id,s.cla
ss_number_id,s.date,s.weekday_index,s.week_type,s.period_id,s.is
_replacement,s.group_id,sub.subject_name,cn.class_number,cd.star
ttime AS class_starttime,cd.endtime AS class_endtime, a.name AS
auditorium_name,a.building_number,u.id AS teacher_id,u.name AS
teacher_name,u.surname AS teacher_surname,
u.middle_name AS teacher_midname,gp.prefix AS
group_prefix,g.group_course,st.name AS schedule_type_name
FROM schedule sJOIN subjects sub ON s.subject_id = sub.id
JOIN class_numbers cn ON s.class_number_id = cn.id
LEFT JOIN classes_duration cd ON cn.id = cd.class_number_idJOIN
auditoriums a ON s.auditorium_id = a.id JOIN groups g ON
s.group_id = g.id JOIN group_prefixes gp ON g.group_prefix =
gp.id JOIN schedule_type st ON s.schedule_type_id = st.id LEFT
JOIN teacher_schedule ts ON ts.schedule_id = s.id LEFT JOIN users
u ON ts.teacher_id = u.id WHERE s.group_id = ? AND (s.subgroup_id
= ? OR s.subgroup_id IS NULL) AND s.period_id = ?)";
    std::string period_id;
    std::vector<int> weekdays_in_range;
    std::string formatted_from, formatted_to;
    if (!from.empty() && !to.empty()) {
        formatted_from = convertDateToSqlFormat(from);
        formatted_to = convertDateToSqlFormat(to);
        std::unique_ptr<sql::PreparedStatement> periodStmt(conn-
>prepareStatement(R"(SELECT p.idFROM periods pJOIN group_periods
gp ON gp.period_id = p.id WHERE gp.group_id = ? AND p.start_date
<= ? AND p.end_date >= ?)"));
       periodStmt->setInt(1, group_id);
        periodStmt->setString(2, formatted_from);
        periodStmt->setString(3, formatted_to);
        std::unique_ptr<sql::ResultSet> periodRs(periodStmt-
>executeQuery());
      if (!periodRs->next()) {
        std::string todayStr = getTodayDateString();
        std::unique_ptr<sql::PreparedStatement>
fallbackStmt(conn->prepareStatement(R"(SELECT p.id FROM periods p
 JOIN group_periods gp ON gp.period_id = p.id WHERE gp.group_id =
? AND p.start_date <= ? AND p.end_date >= ?)"));
            fallbackStmt->setInt(1, group_id);
            fallbackStmt->setString(2, todayStr);
            fallbackStmt->setString(3, todayStr);
```

```
            std::unique_ptr<sql::ResultSet>
fallbackRs(fallbackStmt->executeQuery());
            if (!fallbackRs->next()) {
response.send(Http::Code::Not_Found, "No period found for the
given range or today");
                return;}
  period_id = fallbackRs->getString("id");
 } else {period_id = periodRs->getString("id");}
        auto weekdays_set = getWeekdaysInRange(from, to);
        weekdays_in_range.assign(weekdays_set.begin(),
weekdays_set.end());
        if (weekdays_in_range.empty()) {
            response.send(Http::Code::Ok, R"({"periods": [],
"teachers": [], "subjects": []})", MIME(Application, Json));
            return;}
        std::string placeholders =
generatePlaceholders(weekdays_in_range.size());
        sql += " AND ((s.date BETWEEN ? AND ? AND
s.is_replacement = true) OR (s.weekday_index IN (" +
placeholders +") AND s.is_replacement = false)) ORDER BY
s.weekday_index";} else {
        std::string todayStr = getTodayDateString();
        std::unique_ptr<sql::PreparedStatement> periodStmt(conn-
>prepareStatement(R"(SELECT p.id FROM periods p JOIN group_periods
gp ON gp.period_id = p.id WHERE gp.group_id = ? AND p.start_date
<= ? AND p.end_date >= ?)"));
        periodStmt->setInt(1, group_id);
        periodStmt->setString(2, todayStr);
        periodStmt->setString(3, todayStr);
        std::unique_ptr<sql::ResultSet> periodRs(periodStmt-
>executeQuery());
        if (!periodRs->next()) {
            response.send(Http::Code::Not_Found, "No current
period found");
            return;}
        period_id = periodRs->getString("id");
        sql += "AND s.is_replacement = false ORDER BY
s.weekday_index";}
    std::unique_ptr<sql::PreparedStatement> main_stmt(conn-
>prepareStatement(sql));
    int param_idx = 1;
    main_stmt->setInt(param_idx++, group_id);
    main_stmt->setInt(param_idx++, subgroup_id);
    main_stmt->setString(param_idx++, period_id);
    if (!from.empty() && !to.empty()) {
        main_stmt->setString(param_idx++, formatted_from);
        main_stmt->setString(param_idx++, formatted_to);
        for (int w : weekdays_in_range) {
            main_stmt->setInt(param_idx++, w);}}
    std::unique_ptr<sql::ResultSet> rs(main_stmt-
>executeQuery());
    nlohmann::json response_json;
```

```
    nlohmann::json periods = nlohmann::json::array();
    nlohmann::json teachers = nlohmann::json::array();
    nlohmann::json subjects = nlohmann::json::array();
    std::map<std::string, nlohmann::json> teacher_map;
    std::map<int, nlohmann::json> subject_map;
    while (rs->next()) {
        std::string teacher_id = rs->isNull("teacher_id") ? "" :
std::string(rs->getString("teacher_id"));
        if (!teacher_id.empty() &&
!teacher_map.count(teacher_id)) {
            teacher_map[teacher_id] =
{{        "id",                                          teacher_id},
                {"first_name",    std::string(rs-
>getString("teacher_name"))},
                { "last_name", std::string(rs-
>getString("teacher_surname"))},
                {   "midname", std::string(rs-
>getString("teacher_midname"))} };}
        int subj_id = rs->getInt("subject_id");
        if (!subject_map.count(subj_id)) {
            subject_map[subj_id] = {
                {"subject_id",
  subj_id},
                {      "name", std::string(rs-
>getString("subject_name"))},
                {  "schedule",  nlohmann::json::array()}};}
        std::string group = std::string(rs-
>getString("group_prefix")) + "-" + std::to_string(rs-
>getInt("group_course"));
        nlohmann::json schedule_entry = {
            {        "shedule_id",
                                    rs->getInt("id")},
            {            "group",
                                              group},
            {          "group_id",
                                rs->getInt("group_id")},
            {        "subject_id",
                                            subj_id},
            {        "auditorium",
        std::string(rs->getString("auditorium_name"))},
            {   "building_number",
        std::string(rs->getString("building_number"))},
            {      "class_number",
          std::string(rs->getString("class_number"))},
            {       "teacher_ids", teacher_id.empty() ?
nlohmann::json::array() : nlohmann::json::array({teacher_id})},
            {         "starttime",
        std::string(rs->getString("class_starttime"))},
            {           "endtime",
          std::string(rs->getString("class_endtime"))},
            {   "subgroup_number",
                            rs->getInt("subgroup_id")},
```

```
            {"schedule_type_name",
    std::string(rs->getString("schedule_type_name"))},
            {         "period_id",
                              rs->getInt("period_id")},
            {    "is_replacement",
                    rs->getBoolean("is_replacement")},
            {         "week_type",
              std::string(rs->getString("week_type"))},
            {     "weekday_index",
           std::string(rs->getString("weekday_index"))},
              {              "date",
                    std::string(rs->getString("date"))}};
        subject_map[subj_id]["schedule"].push_back(schedule_entr
y);}
    for (const auto &[key, t] : teacher_map) {
        teachers.push_back(t);}
    for (const auto &[key, s] : subject_map) {
        subjects.push_back(s);}
    if (!period_id.empty()) {
        std::unique_ptr<sql::PreparedStatement> pStmt(conn-
>prepareStatement(R"(SELECT p.id, pt.name, p.start_date,
p.end_date FROM periods p JOIN period_types pt ON p.type_id =
pt.id WHERE p.id = ?)"));
        pStmt->setString(1, period_id);
        std::unique_ptr<sql::ResultSet> pr(pStmt-
>executeQuery());
        if (pr->next()) {
            periods.push_back({{        "id",
      pr->getInt("id")},
                {      "type",       std::string(pr-
>getString("name"))},
                {"start_date", std::string(pr-
>getString("start_date"))},
                {  "end_date",   std::string(pr-
>getString("end_date"))}
            });
        }
    }
    response_json["periods"] = periods;
    response_json["teachers"] = teachers;
    response_json["subjects"] = subjects;
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));
}
```

Додаток Б

Лістинг функції fetch_user_notifications

```
void RestAPI::fetch_user_notifications(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string sql = R"(SELECT n.id AS
notification_id,n.user_id,n.sender_id,u.login,u.surname,u.name,u
.middle_name,u.email,n.notification_text,n.is_read,n.created_at,
nt.name AS type_name FROM notification n JOIN notification_type nt
ON n.type_id = nt.id JOIN users u ON n.sender_id = u.id
WHERE n.user_id = ? ORDER BY n.created_at DESC;)";
    std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(sql));
    stmt->setString(1, user_id);
    std::unique_ptr<sql::ResultSet> res(stmt->executeQuery());
    std::map<std::string, nlohmann::json> senders_map;
    while (res->next()) {
        std::string sender_id = std::string(res-
>getString("sender_id"));
        std::string surname = std::string(res-
>getString("surname"));
        std::string name = std::string(res->getString("name"));
        if (senders_map.find(sender_id) == senders_map.end()) {
            senders_map[sender_id] = {
                {      "surname",                surname},
                {         "name",                   name},
                {"notifications", nlohmann::json::array()}
            };
        }
        nlohmann::json notification_json = {
            {        "id",      res->getInt("notification_id")},
            {   "user_id",           res->getString("user_id")},
            {      "text", res->getString("notification_text")},
            {      "type",         res->getString("type_name")},
            {   "is_read",          res->getBoolean("is_read")},
            {"created_at",        res->getString("created_at")}
        };
        senders_map[sender_id]["notifications"].push_back(notifi
cation_json);
    }
    nlohmann::json senders_array = nlohmann::json::array();
    for (auto &[id, sender] : senders_map) {
        senders_array.push_back(sender);
    }
    nlohmann::json response_json;
    response_json["notifications"] = senders_array;
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));
}
```

Додаток В

Лістинг функції fetch_user_notification

```
void RestAPI::fetch_user_notification(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string user_id =
request.param(":id").as<std::string>();
    std::string notification_id =
request.param(":notification_id").as<std::string>();
        const std::string session_id = jwt::decode(auth_header-
>value().substr(7)).get_payload_claim("session_id").as_string();
        const auto session_info = Database::getSession(conn,
session_id);
        if (!session_info.has_value()) {
            response.send(Http::Code::Not_Found, "NotFound:
Invalid session");
            return;}
        user_id = session_info->user_id;}
    std::string sql = R"(SELECTn.id AS
notification_id,n.user_id,n.sender_id,u.login,u.surname,u.name,
u.middle_name,u.email,n.notification_text,n.is_read,n.created_at
,nt.name AS type_name FROM notification n JOIN notification_type nt
ON n.type_id = nt.id JOIN users u ON n.sender_id = u.id
WHERE  n.user_id = ? AND n.id = ?)";
    std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(sql));
    stmt->setString(1, user_id);
    stmt->setString(2, notification_id);
    std::unique_ptr<sql::ResultSet> res(stmt->executeQuery());
    if (!res->next()) {
        response.send(Http::Code::Not_Found, "Notification not
found");
        return;}
    nlohmann::json sender_json = {
        {         "id",   std::string(res-
>getString("sender_id"))},
        {      "login",       std::string(res-
>getString("login"))},
        {    "surname",     std::string(res-
>getString("surname"))},
        {       "name",        std::string(res-
>getString("name"))},
        {"middle_name", std::string(res-
>getString("middle_name"))},
        {      "email",       std::string(res-
>getString("email"))}
    };
    nlohmann::json notification_json = {
```

```
        {        "id",                    res-
>getInt("notification_id")},
        {   "user_id",           std::string(res-
>getString("user_id"))},
        {      "text", std::string(res-
>getString("notification_text"))},
        {      "type",         std::string(res-
>getString("type_name"))},
        {   "is_read",                       res-
>getBoolean("is_read")},
        {"created_at",        std::string(res-
>getString("created_at"))},
        {    "sender",                                          send
er_json}
    };
    response.send(Http::Code::Ok, notification_json.dump(2),
MIME(Application, Json));
}
```

Додаток Г

Лістинг функції fetch_schedule

```
void RestAPI::fetch_schedule(const Pistache::Rest::Request
&request, Pistache::Http::ResponseWriter response){
    std::string date = request.query().get("date").value_or("");
    std::string from = request.query().get("from").value_or("");
    std::string to = request.query().get("to").value_or("");
    std::unique_ptr<sql::PreparedStatement> periodStmt;
    std::string period_id;
    std::string sql = R"(SELECT s.id,s.group_id,
s.subject_id,s.subgroup_id,s.auditorium_id,s.class_number_id,s.d
ate,s.weekday_index,s.week_type,s.period_id,s.is_replacement,s.g
roup_id,sub.subject_name,cn.class_number,cd.starttime AS
class_starttime,cd.endtime AS class_endtime,a.name AS
auditorium_name,a.building_number,u.id AS teacher_id,u.name AS
teacher_name,u.surname AS teacher_surname,u.middle_name AS
teacher_midname,gp.prefix AS group_prefix,g.group_course FROM
schedule s JOIN subjects sub ON s.subject_id = sub.id JOIN
class_numbers cn ON s.class_number_id = cn.id LEFT JOIN
classes_duration cd ON cn.id = cd.class_number_id JOIN
auditoriums a ON s.auditorium_id = a.id JOIN groups g ON
s.group_id = g.id JOIN group_prefixes gp ON g.group_prefix =
gp.id LEFT JOIN teacher_schedule ts ON ts.schedule_id = s.id LEFT
JOIN users u ON ts.teacher_id = u.id)";
    std::vector<std::string> conditions;
    std::vector<std::string> weekday_list;
    std::string formatted_date, formatted_from, formatted_to;
    std::set<int> weekdays_in_range;
    int weekday_index = 10;
    if (!date.empty()) {
        formatted_date = convertDateToSqlFormat(date);
        std::istringstream iss(date);
        std::chrono::sys_days sys_date;
        iss >> std::chrono::parse("%d.%m.%Y", sys_date);
        std::chrono::weekday wd(sys_date);
        periodStmt.reset(conn->prepareStatement(R"(
            SELECT id FROM periods WHERE start_date <= ? AND
end_date >= ?;)"));
        periodStmt->setString(1, formatted_date);
        periodStmt->setString(2, formatted_date);
        std::unique_ptr<sql::ResultSet> periodRs(periodStmt-
>executeQuery());
        if (!periodRs->next()) {
            response.send(Http::Code::Not_Found, "No period
found for the given date or range");
            return;}
        period_id = periodRs->getString("id");
        weekday_index = wd.c_encoding();
```

```
        sql +="WHERE (s.date = ? OR (s.weekday_index = ? AND
s.is_replacement = false)) AND s.period_id = ? ORDER BY
s.weekday_index";} else if (!from.empty() && !to.empty()) {
        formatted_from = convertDateToSqlFormat(from);
        formatted_to = convertDateToSqlFormat(to);
        periodStmt.reset(conn->prepareStatement(R"(SELECT id
FROM periods WHERE start_date <= ? AND end_date >= ?;)"));
        periodStmt->setString(1, formatted_from);
        periodStmt->setString(2, formatted_to);
        std::unique_ptr<sql::ResultSet> periodRs(periodStmt-
>executeQuery());
        if (!periodRs->next()) {
            response.send(Http::Code::Not_Found, "No period
found for the given date or range");
            return;}
        period_id = periodRs->getString("id");
        weekdays_in_range = getWeekdaysInRange(from, to);
        if (weekdays_in_range.empty()) {
            response.send(Http::Code::Ok, R"({"periods": [],
"teachers": [], "subjects": []})", MIME(Application, Json));
            return;}
        std::string placeholders =
generatePlaceholders(weekdays_in_range.size());
        sql += "WHERE (s.date BETWEEN ? AND ? AND
s.is_replacement = true) OR (s.weekday_index IN (" +
placeholders +") AND s.is_replacement = false) AND s.period_id =
? ORDER BY s.weekday_index";}
    std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(sql));
    int param_idx = 1;
    if (!date.empty()) {
        stmt->setString(param_idx++, formatted_date);
        stmt->setInt(param_idx++, weekday_index);
    } else if (!from.empty() && !to.empty()) {
        stmt->setString(param_idx++, formatted_from);
        stmt->setString(param_idx++, formatted_to);
        for (const auto &w : weekdays_in_range)
            stmt->setInt(param_idx++, w);}
    if (!period_id.empty())
        stmt->setString(param_idx++, period_id);
    std::unique_ptr<sql::ResultSet> rs(stmt->executeQuery());
    nlohmann::json response_json;
    nlohmann::json periods = nlohmann::json::array();
    nlohmann::json teachers = nlohmann::json::array();
    nlohmann::json subjects = nlohmann::json::array();
    std::map<std::string, nlohmann::json> teacher_map;
    std::map<int, nlohmann::json> subject_map;
    while (rs->next()) {
        std::string teacher_id = rs->isNull("teacher_id") ? "" :
std::string(rs->getString("teacher_id"));
        if (!teacher_map.count(teacher_id)) {
```

```
            teacher_map[teacher_id] =
{{        "id",                          teacher_id},
                {"first_name",    rs-
>getString("teacher_name")},
                { "last_name", rs-
>getString("teacher_surname")},
                {   "midname", rs->getString("teacher_midname")}
            };
        }
        int subj_id = rs->getInt("subject_id");
        if (!subject_map.count(subj_id)) {
            subject_map[subj_id] = {
                {"subject_id",                          subj_id},
                {      "name", rs->getString("subject_name")},
                {  "schedule",       nlohmann::json::array()}
            };
        }
        std::string group = std::string(rs-
>getString("group_prefix")) + "-" + std::to_string(rs-
>getInt("group_course"));
        subject_map[subj_id]["schedule"].push_back({
            {     "shedule_id",                    rs-
>getInt("id")},
            {          "group",                                gr
oup},
            {       "group_id",             rs-
>getInt("group_id")},
            {     "subject_id",                           subj
_id},
            {     "auditorium",    rs-
>getString("auditorium_name")},
            {   "class_number",       rs-
>getString("class_number")},
            {    "teacher_ids",
nlohmann::json::array({teacher_id})},
            {      "starttime",    rs-
>getString("class_starttime")},
            {        "endtime",      rs-
>getString("class_endtime")},
            {"subgroup_number",           rs-
>getInt("subgroup_id")},
            {      "period_id",             rs-
>getInt("period_id")},
            { "is_replacement",    rs-
>getBoolean("is_replacement")},
            {      "week_type",          rs-
>getString("week_type")},
            {  "weekday_index",      rs-
>getString("weekday_index")},
            {           "date",               rs-
>getString("date")}
        });}
```

```
    for (const auto &[_, t] : teacher_map) {
        teachers.push_back(t);}
    for (const auto &[_, s] : subject_map) {
        subjects.push_back(s);}
    if (!period_id.empty()) {
        std::unique_ptr<sql::PreparedStatement> pStmt(conn-
>prepareStatement(R"(SELECT p.id, pt.name, p.start_date,
p.end_date FROM periods p JOIN period_types pt ON p.type_id =
pt.id WHERE p.id = ?)"));
        pStmt->setString(1, period_id);
        std::unique_ptr<sql::ResultSet> pr(pStmt-
>executeQuery());
        if (pr->next()) {
            periods.push_back({
                {        "id",            pr->getInt("id")},
                {      "type",       pr->getString("name")},
                {"start_date", pr->getString("start_date")},
                {  "end_date",   pr->getString("end_date")}});}
    response_json["periods"] = periods;
    response_json["teachers"] = teachers;
    response_json["subjects"] = subjects;
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));
}
```

Додаток Ґ

Лістинг функції fetch_teachers_shedule

```
void RestAPI::fetch_teachers_shedule(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string user_id =request.param(":id").as<std::string>();
    std::string from = request.query().get("from").value_or("");
    std::string to = request.query().get("to").value_or("");
    std::string formatted_from = from.empty() ?
getTodayDateString() : convertDateToSqlFormat(from);
    std::string formatted_to = to.empty() ? getTodayDateString()
: convertDateToSqlFormat(to);
    std::vector<std::string> period_ids;{
        std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(R"(SELECT DISTINCT p.id FROM periods p JOIN
group_periods gp ON gp.period_id = p.id WHERE p.end_date >= ? AND
p.start_date <= ?)"));
        stmt->setString(1, formatted_from);
        stmt->setString(2, formatted_to);
        std::unique_ptr<sql::ResultSet> rs(stmt-
>executeQuery());
        while (rs->next()) {
            period_ids.push_back(static_cast<std::string>(rs-
>getString("id")));}}
    if (period_ids.empty()) {
        response.send(Http::Code::Not_Found, "No periods found
for the given date range or today.");
        return;}
    std::vector<int> weekdays_in_range;
    if (!from.empty() && !to.empty()) {
        auto weekdays_set = getWeekdaysInRange(from, to);
        weekdays_in_range.assign(weekdays_set.begin(),
weekdays_set.end());
        if (weekdays_in_range.empty()) {
            response.send(Http::Code::Ok, R"({"periods": [],
"teachers": [], "subjects": []})", MIME(Application, Json));
            return;}}
    std::string sql =
R"(SELECT  s.id,s.group_id,s.subject_id,s.subgroup_id,s.auditoriu
m_id,s.class_number_id,s.date,s.weekday_index,s.week_type,s.peri
od_id,s.is_replacement,s.group_id,sub.subject_name,cn.class_numb
er,cd.starttime AS class_starttime            cd.endtime AS
class_endtimea.name AS auditorium_name,a.building_number,u.id AS
teacher_id, u.name AS teacher_name,u.surname AS
teacher_surname,u.middle_name AS teacher_midname,
gp.prefix AS group_prefix,g.group_course,
st.name AS schedule_type_name FROM schedule s JOIN subjects sub ON
s.subject_id = sub.i JOIN class_numbers cn ON s.class_number_id
= cn.id LEFT JOIN classes_duration cd ON cn.id =
```

```
cd.class_number_id JOIN auditoriums a ON s.auditorium_id = a.id
JOIN groups g ON s.group_id = g.id JOIN group_prefixes gp ON
g.group_prefix = gp.id JOIN schedule_type st ON
s.schedule_type_id = st.id LEFT JOIN teacher_schedule ts ON
ts.schedule_id = s.id LEFT JOIN users u ON ts.teacher_id = u.id
WHERE ts.teacher_id = ?)";
    std::string period_placeholders =
generatePlaceholders(period_ids.size());
    sql += " AND s.period_id IN (" + period_placeholders + ")";
    if (!from.empty() && !to.empty()) {
        std::string wd_placeholders =
generatePlaceholders(weekdays_in_range.size());
        sql += " AND ((s.date BETWEEN ? AND ? AND
s.is_replacement = true) OR (s.weekday_index IN (" +
wd_placeholders +") AND s.is_replacement = false)) ORDER BY
s.weekday_index";} else { sql += " AND s.is_replacement = false
ORDER BY s.weekday_index";}
    std::unique_ptr<sql::PreparedStatement> main_stmt(conn-
>prepareStatement(sql));
    int param_idx = 1;
    main_stmt->setString(param_idx++, user_id);
    for (const auto &pid : period_ids) {
        main_stmt->setString(param_idx++, pid);}
    if (!from.empty() && !to.empty()) {
        main_stmt->setString(param_idx++, formatted_from);
        main_stmt->setString(param_idx++, formatted_to);
        for (int w : weekdays_in_range) {
            main_stmt->setInt(param_idx++, w);}}
    std::unique_ptr<sql::ResultSet> rs(main_stmt-
>executeQuery());
    nlohmann::json response_json;
    nlohmann::json periods = nlohmann::json::array();
    nlohmann::json teachers = nlohmann::json::array();
    nlohmann::json subjects = nlohmann::json::array();
    std::map<std::string, nlohmann::json> teacher_map;
    std::map<int, nlohmann::json> subject_map;
    while (rs->next()) {
        std::string teacher_id = rs->isNull("teacher_id") ? "" :
std::string(rs->getString("teacher_id"));
        if (!teacher_id.empty() &&
!teacher_map.count(teacher_id)) {
            teacher_map[teacher_id] =
{       "id",                                       teacher_id},
                {"first_name",    std::string(rs-
>getString("teacher_name"))},
                { "last_name", std::string(rs-
>getString("teacher_surname"))},
                {   "midname", std::string(rs-
>getString("teacher_midname"))}
            };}
        int subj_id = rs->getInt("subject_id");
        if (!subject_map.count(subj_id)) {
```

```
            subject_map[subj_id] = {
                {"subject_id",
  subj_id},
                {      "name", std::string(rs-
>getString("subject_name"))},
                {  "schedule",                      nlohmann::json
::array()}
            };
        }
        std::string group = std::string(rs-
>getString("group_prefix")) + "-" + std::to_string(rs-
>getInt("group_course"));
        nlohmann::json schedule_entry = {
            {        "shedule_id",
                                    rs->getInt("id")},
            {            "group",
                                              group},
            {          "group_id",
                                rs->getInt("group_id")},
            {        "subject_id",
                                           subj_id},
            {        "auditorium",
        std::string(rs->getString("auditorium_name"))},
            {   "building_number",
        std::string(rs->getString("building_number"))},
            {      "class_number",
          std::string(rs->getString("class_number"))},
            {       "teacher_ids", teacher_id.empty() ?
nlohmann::json::array() : nlohmann::json::array({teacher_id})},
            {         "starttime",
        std::string(rs->getString("class_starttime"))},
            {           "endtime",
          std::string(rs->getString("class_endtime"))},
            {   "subgroup_number",
                          rs->getInt("subgroup_id")},
            {"schedule_type_name",
    std::string(rs->getString("schedule_type_name"))},
            {         "period_id",
                                rs->getInt("period_id")},
            {    "is_replacement",
                   rs->getBoolean("is_replacement")},
            {          "week_type",
              std::string(rs->getString("week_type"))},
            {     "weekday_index",
          std::string(rs->getString("weekday_index"))},
            {               "date",
                 std::string(rs->getString("date"))}};
        subject_map[subj_id]["schedule"].push_back(schedule_entr
y);}
    for (const auto &[_, t] : teacher_map) {
        teachers.push_back(t);}
```

```
    for (const auto &[_, s] : subject_map)
{subjects.push_back(s);}
    for (const auto &pid : period_ids) {
        std::unique_ptr<sql::PreparedStatement> pStmt(conn-
>prepareStatement(R"(SELECT p.id, pt.name, p.start_date,
p.end_date FROM periods p JOIN period_types pt ON p.type_id =
pt.id WHERE p.id = ?)"));
        pStmt->setString(1, pid);
        std::unique_ptr<sql::ResultSet> pr(pStmt-
>executeQuery());
        if (pr->next()) {
            periods.push_back({
                {        "id",                          pr-
>getInt("id")},
                {      "type",       std::string(pr-
>getString("name"))},
                {"start_date", std::string(pr-
>getString("start_date"))},
                {  "end_date",   std::string(pr-
>getString("end_date"))}});}}
    response_json["periods"] = periods;
    response_json["teachers"] = teachers;
    response_json["subjects"] = subjects;
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));}
```

Додаток Д

Лістинг функції fetch_lesson_attendance

```
void RestAPI::fetch_lesson_attendance(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string user_id =
request.param(":id").as<std::string>();
    std::string date = request.query().get("date").value_or("");
    std::string formatted_date = date.empty() ?
getTodayDateString() : convertDateToSqlFormat(date);
    std::vector<std::string> period_ids;{
        std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(R"(SELECT DISTINCT p.id FROM periods p WHERE
p.end_date >= ? AND p.start_date <= ?)"));
        stmt->setString(1, formatted_date);
        stmt->setString(2, formatted_date);
        std::unique_ptr<sql::ResultSet> rs(stmt-
>executeQuery());
        while (rs->next()) {
            period_ids.push_back(static_cast<std::string>(rs-
>getString("id")));}}
    if (period_ids.empty()) {
        response.send(Http::Code::Not_Found, "No periods found
for the given date.");
        return;}
    int weekday_index = getWeekdayIndex(formatted_date);
    std::string sql = R"( SELECT s.id AS schedule_id,
s.group_id,s.subject_id, s.subgroup_id,s.class_number_id,s.weekd
ay_index,s.week_type,s.period_id,s.is_replacement,s.date,
sub.subject_name,cn.class_number,gp.prefix
group_prefix,g.group_course,a.id AS
attendance_id,a.placed_id,a.student_id,a.date_of_pass,a.excused
FROM schedule s JOIN subjects sub ON s.subject_id = sub.id JOIN
class_numbers cn ON s.class_number_id = cn.id JOIN groups g ON
s.group_id = g.id JOIN group_prefixes gp ON g.group_prefix =
gp.id JOIN teacher_schedule ts ON ts.schedule_id = s.id JOIN
periods p ON s.period_id = p.id LEFT JOIN attendance a ON
a.schedule_id = s.id AND a.date_of_pass = ? WHERE ts.teacher_id =
? AND s.period_id IN ()" +
generatePlaceholders(period_ids.size()) + R"() AND (  (s.date = ?
AND s.is_replacement = true)  OR   (s.weekday_index = ? AND
s.is_replacement = false)))";
    std::unique_ptr<sql::PreparedStatement> main_stmt(conn-
>prepareStatement(sql));
    int param_idx = 1;
    main_stmt->setString(param_idx++, formatted_date);
    main_stmt->setString(param_idx++, user_id);
    for (const auto &pid : period_ids) {
        main_stmt->setString(param_idx++, pid);}
```

```
    main_stmt->setString(param_idx++, formatted_date);
    main_stmt->setInt(param_idx++, weekday_index);
    std::unique_ptr<sql::ResultSet> rs(main_stmt-
>executeQuery());
    std::map<int, nlohmann::json> schedule_map;
    std::set<int> unique_group_ids;
    while (rs->next()) {
        int schedule_id = rs->getInt("schedule_id");
        int group_id = rs->getInt("group_id");
        unique_group_ids.insert(group_id);
        if (schedule_map.find(schedule_id) ==
schedule_map.end()) {std::string group_name = std::string(rs-
>getString("group_prefix")) + "-" + std::to_string(rs-
>getInt("group_course"));
            schedule_map[schedule_id] = {
                {    "schedule_id",                    schedule_
id},
                {       "group_id",         rs-
>getInt("group_id")},
                {     "subject_id",       rs-
>getInt("subject_id")},
                {   "subject_name",  rs-
>getString("subject_name")},
                {      "week_type",     rs-
>getString("week_type")},
                {  "weekday_index", rs-
>getString("weekday_index")},
                {      "period_id",        rs-
>getInt("period_id")},}
        if (!rs->isNull("attendance_id")) {
            nlohmann::json att = {
                {"attendance_id",   rs-
>getInt("attendance_id")},
                {    "placed_id",    rs-
>getString("placed_id")},
                {   "student_id",   rs-
>getString("student_id")},
                {         "date", rs-
>getString("date_of_pass")},
                {      "excused",      rs-
>getString("excused")}};
            schedule_map[schedule_id]["attendance"].push_back(at
t);}
    nlohmann::json group_students_map =
nlohmann::json::object();
    for (int group_id : unique_group_ids) {
        std::unique_ptr<sql::PreparedStatement>
students_stmt(conn->prepareStatement(R"(SELECT u.id, u.name,
u.surname FROM users u JOIN users_groups ug ON ug.user_id =
u.id WHERE ug.group_id = ?
        )"));
        students_stmt->setInt(1, group_id);
```

```
        nlohmann::json students_array = nlohmann::json::array();
        std::unique_ptr<sql::ResultSet>
students_rs(students_stmt->executeQuery());
        while (students_rs->next()) {
            students_array.push_back({
                {        "id",      students_rs-
>getString("id")},
                {"first_name",    students_rs-
>getString("name")},
                { "last_name", students_rs-
>getString("surname")}
            });
        }
        group_students_map[std::to_string(group_id)] =
students_array;
    }
    nlohmann::json schedule_array = nlohmann::json::array();
    for (auto &entry : schedule_map) {
        schedule_array.push_back(entry.second);}
    nlohmann::json period_details = nlohmann::json::array();
    for (const auto &pid : period_ids) {
        std::unique_ptr<sql::PreparedStatement> pStmt(conn-
>prepareStatement(R"(SELECT p.id, pt.name, p.start_date,
p.end_date FROM periods p JOIN period_types pt ON p.type_id =
pt.id WHERE p.id = ?
        )"));
        pStmt->setString(1, pid);
        std::unique_ptr<sql::ResultSet> pr(pStmt-
>executeQuery());
        if (pr->next()) {
            period_details.push_back({
                {        "id",            pr->getInt("id")},
                {      "type",       pr->getString("name")},
                {"start_date", pr->getString("start_date")},
                {  "end_date",   pr->getString("end_date")}
            });}}
    nlohmann::json response_json = {
        { "periods",     period_details},
        {"students", group_students_map},
        {"schedule",     schedule_array}
    };
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));}
```

Додаток Е

Лістинг функції form_group_attendance_report

```
void RestAPI::form_group_attendance_report(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string user_id =
request.param(":id").as<std::string>();
    std::unique_ptr<sql::PreparedStatement> group_stmt(conn-
>prepareStatement(R"(
        SELECT group_id FROM `users_groups` WHERE user_id =
?)"));
    group_stmt->setString(1, user_id);
    std::unique_ptr<sql::ResultSet> group_res(group_stmt-
>executeQuery());
    if (!group_res->next()) {
        response.send(Http::Code::Not_Found, "User group not
found");
        return;}
    int group_id = group_res->getInt("group_id");
    try {
        auto json = nlohmann::json::parse(request.body());
        std::string month = json["month"].get<std::string>();
        std::vector<nlohmann::json> periods = json["periods"];
        std::string period_columns;
        for (size_t i = 0; i < periods.size(); i++) {
            period_columns += ", COUNT(CASE WHEN a.date_of_pass
BETWEEN ? AND ? THEN 1 END) AS total_period_" +
std::to_string(i) + ", COUNT(CASE WHEN a.date_of_pass BETWEEN ?
AND ? AND a.excused = 1 THEN 1 END) AS excused_period_" +
std::to_string(i) +", COUNT(CASE WHEN a.date_of_pass BETWEEN ?
AND ? AND a.illness = 1 THEN 1 END) AS illness_period_"
+std::to_string(i);}
std::string sql = R"(SELECT u.surname,u.name,u.middle_name, g.id
AS group_id,gp.prefix AS group_prefix,g.group_course, COUNT(CASE
WHEN a.id IS NOT NULL THEN 1 END) AS total_absences,COUNT(CASE
WHEN a.excused = 1 THEN 1 END) AS total_excused,COUNT(CASE WHEN
a.illness = 1 THEN 1 END) AS total_illness)" + period_columns
+R"(FROM users u JOIN users_groups ug ON u.id = ug.user_id
JOIN groups g ON ug.group_id = g.id JOIN group_prefixes gp ON
g.group_prefix = gp.id LEFT JOIN attendance a ON u.id =
a.student_id WHERE g.id = ? AND DATE_FORMAT(a.date_of_pass, '%Y-
%m') = ? GROUP BY u.id, u.surname, u.name, u.middle_name, g.id,
gp.prefix, g.group_course ORDER BY gp.prefix, g.group_course,
u.surname, u.name; )";
        std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(sql));
        int param_idx = 1;
        for (const auto &period : periods) {
```

```
            std::string start =
period["start_date"].get<std::string>();
            std::string end =
period["end_date"].get<std::string>();
            stmt->setString(param_idx++, start);
            stmt->setString(param_idx++, end);
            stmt->setString(param_idx++, start);
            stmt->setString(param_idx++, end);
            stmt->setString(param_idx++, start);
            stmt->setString(param_idx++, end);}
        stmt->setInt(param_idx++, group_id);
        stmt->setString(param_idx++, month);
        std::unique_ptr<sql::ResultSet> res(stmt-
>executeQuery());
        nlohmann::json report;
        report["periods"] = periods;
        std::map<int, nlohmann::json> group_map;
        while (res->next()) {
            std::string group_name = std::string(res-
>getString("group_prefix")) + "-" + std::to_string(res-
>getInt("group_course"));
            if (!group_map.count(group_id)) {
                group_map[group_id] = {
                    {      "id",                group_id},
                    {   "group",              group_name},
                    {"students", nlohmann::json::array()}
                };
            }
            std::string surname = std::string(res-
>getString("surname"));
            std::string name = std::string(res-
>getString("name"));
            std::string middle_name = std::string(res-
>getString("middle_name"));
            int total_absences = res->getInt("total_absences");
            int total_excused = res->getInt("total_excused");
            int total_illness = res->getInt("total_illness");
            int excused_without_illness = total_excused -
total_illness;
            nlohmann::json student_json = {
                {       "full_name", surname + " " + name + " "
+ middle_name},
                {  "total_absences",                          t
otal_absences},
                {"excused_absences",                 excused_wi
thout_illness},
                {"illness_absences",
total_illness},
                {
"period_absences",                 nlohmann::json::array()}};
            for (size_t i = 0; i < periods.size(); i++) {
```

```
                int total = res->getInt("total_period_" +
std::to_string(i));
                int excused = res->getInt("excused_period_" +
std::to_string(i));
                int illness = res->getInt("illness_period_" +
std::to_string(i));
                int excused_without_illness_period = excused -
illness;
           student_json["period_absences"].push_back({
                {  "total",                          total},
                {"excused", excused_without_illness_period},
                {"illness",                        illness}});}
       group_map[group_id]["students"].push_back(student_json);}
        nlohmann::json groups_array = nlohmann::json::array();
        for (auto &[_, group_json] : group_map) {
            groups_array.push_back(group_json);}
        report["groups"] = groups_array;
        response.send(Http::Code::Ok, report.dump(2),
MIME(Application, Json));} catch (const
nlohmann::json::exception &e) {
        response.send(Http::Code::Bad_Request, "JSON error: " +
std::string(e.what()));
    } catch (const sql::SQLException &e) {
        response.send(Http::Code::Internal_Server_Error,
"Database error: " + std::string(e.what()));
    } catch (const std::exception &e) {
        response.send(Http::Code::Internal_Server_Error, "Error:
" + std::string(e.what()));
    }
}
```

Додаток Є

Лістинг функції fetch_department_info

```
void RestAPI::fetch_department_info(const
Pistache::Rest::Request &request, Pistache::Http::ResponseWriter
response){
    std::string user_id =
request.param(":id").as<std::string>();
    std::string sql = R"(SELECT ad.department_id, d.name AS
department_name, s.id AS specialty_id,s.specialty_name,g.id AS
group_id,g.group_course,gp.prefix AS group_prefix FROM
admin_departments ad JOIN departments d ON ad.department_id =
d.id JOIN specialties s ON s.department_id = d.id JOIN groups g ON
g.specialty_id = s.id JOIN group_prefixes gp ON g.group_prefix =
gp.id WHERE ad.admin_id = ? ORDER BY d.id, s.id, g.id)";
    std::unique_ptr<sql::PreparedStatement> stmt(conn-
>prepareStatement(sql));
    stmt->setString(1, user_id);
    std::unique_ptr<sql::ResultSet> res(stmt->executeQuery());
    nlohmann::json result = nlohmann::json::object();
    std::map<int, nlohmann::json> departments_map;
    while (res->next()) {
        int department_id = res->getInt("department_id");
        std::string department_name = std::string(res-
>getString("department_name"));
        int specialty_id = res->getInt("specialty_id");
        std::string specialty_name = std::string(res-
>getString("specialty_name"));
        int group_id = res->getInt("group_id");
        std::string group_prefix = std::string(res-
>getString("group_prefix"));
        int group_course = res->getInt("group_course");
        std::string group_name = group_prefix + "-" +
std::to_string(group_course);
        if (departments_map.find(department_id) ==
departments_map.end()) {
            departments_map[department_id] = {
                {         "id",           department_id},
                {       "name",         department_name},
                {"specialties", nlohmann::json::array()}};}
        nlohmann::json *specialty_ptr = nullptr;
        auto &specialties =
departments_map[department_id]["specialties"];
        for (auto &spec : specialties) {
            if (spec["id"] == specialty_id) {
                specialty_ptr = &spec;
                break;}}
        if (specialty_ptr == nullptr) {
            nlohmann::json new_specialty = {
                {    "id",           specialty_id},
```

```
                {  "name",          specialty_name},
                {"groups", nlohmann::json::array()}
            };
            specialties.push_back(new_specialty);
            specialty_ptr = &specialties.back();}
        nlohmann::json group_json = {
            {        "id",   group_id},
            {"group_name", group_name}
        };
        (*specialty_ptr)["groups"].push_back(group_json);}
    nlohmann::json departments_array = nlohmann::json::array();
    for (auto &[id, dept] : departments_map) {
        departments_array.push_back(dept);}
    nlohmann::json response_json;
    response_json["departments"] = departments_array;
    response.send(Http::Code::Ok, response_json.dump(2),
MIME(Application, Json));}
```

Додаток Ж

Лістинг функції add_notification

```
void RestAPI::add_notification(const Pistache::Rest::Request
&request, Pistache::Http::ResponseWriter response)
{
    if (!checkAuthorize(request)) {
        response.send(Http::Code::Unauthorized, "Unauthorized");
        return;
    }

    auto json = nlohmann::json::parse(request.body(), nullptr,
false);
    if (json.is_discarded() || !json.contains("user_id") ||
!json.contains("sender_id") || !json.contains("type_id") ||
        !json.contains("text")) {
        response.send(Http::Code::Bad_Request, "Missing required
fields: user_id, sender_id, type_id, text");
        return;
    }

    std::string user_id = json["user_id"];
    std::string sender_id = json["sender_id"];
    int type_id = json["type_id"];
    std::string text = json["text"];
    bool is_read = json.value("is_read", false);

    Database::ConnectionGuard conn(*m_connPool);
    auto inserted_id = Database::insertNotification(conn,
user_id, sender_id, type_id, text, is_read);

    if (!inserted_id.has_value()) {
        response.send(Http::Code::Internal_Server_Error, "Failed
to insert notification");
        return;
    }

    response.send(Http::Code::Created,
std::to_string(*inserted_id));
}
```

Додаток І

Лістинг програмного коду AuthInterceptor

```
class AuthInterceptor extends Interceptor {
  final RestAPIClient apiClient;
  bool _isRefreshing = false;
  final List<({RequestOptions options, RequestInterceptorHandler
handler})>
      _pendingRequests = [];
  AuthInterceptor(this.apiClient);
  @override
  Future<void> onRequest(
    RequestOptions options,
    RequestInterceptorHandler handler,
  ) async {
    if (options.path.contains('/auth/login') ||
        options.path.contains('/auth/refresh')) {
      return handler.next(options);}
    if (_isRefreshing) {
      _pendingRequests.add((options: options, handler:
handler));
    }
    final token = await apiClient.getAccessToken();

    if (token == null || await apiClient.isAccessTokenExpired{
      _isRefreshing = true;
      try {
        final refreshed = await apiClient.refreshTokens();
        if (!refreshed) {
          return handler.reject(DioException(
            requestOptions: options,
            error: 'Token refresh failed',
            type: DioExceptionType.cancel,));}} catch (e) {
        return handler.reject(DioException(
          requestOptions: options,
          error: 'Token refresh failed due to exception: $e',
          type: DioExceptionType.cancel,
        ));
      } finally {
        _isRefreshing = false;
        _processPendingRequests();}}
    final newToken = await apiClient.getAccessToken();
    if (newToken != null) {
      options.headers['Authorization'] = 'Bearer $newToken';
    } else {
      return handler.reject(DioException(
        requestOptions: options,
        error: 'Access token is null after refresh attempt',
        type: DioExceptionType.cancel,));
    handler.next(options);
```

```
  void _processPendingRequests() async {
    final pending = List.of(_pendingRequests);
    _pendingRequests.clear();
    for (final entry in pending) {
      final options = entry.options;
      final handler = entry.handler;
      try {
        final newToken = await apiClient.getAccessToken();
        if (newToken != null) {
          options.headers['Authorization'] = 'Bearer $newToken';
        } else {
          handler.reject(DioException(
            requestOptions: options,
            error: 'Access token is null for pending request
after refresh',
            type: DioExceptionType.cancel,));
          continue;}
        final response = await apiClient.dio.fetch(options);
        handler.resolve(response);
      } on DioException catch (e) {
        handler.reject(e);}}}
  @override
  void onError(DioException err, ErrorInterceptorHandler
handler) async {
    final request = err.requestOptions;
    final statusCode = err.response?.statusCode;
    if (statusCode == 401 &&
!request.extra.containsKey('retried')) {
      request.extra['retried'] = true;
      try {
        final refreshed = await apiClient.refreshTokens();
        if (refreshed) {
          final newToken = await apiClient.getAccessToken();
          if (newToken != null) {
            request.headers['Authorization'] = 'Bearer
$newToken';} else {
            return handler.reject(DioException(
              requestOptions: request,
              error: 'New token is null after refresh',
              type: DioExceptionType.badResponse,
              response: Response(
                requestOptions: request,
                statusCode: 401,
                statusMessage: 'Authentication failed',),,));}
          final response = await apiClient.dio.fetch(request);
          return handler.resolve(response);}} catch (e) {
        print("Refresh failed, attempting to clear tokens and
log out: $e");}
      return handler.reject(DioException(
        requestOptions: request,
        error: 'Authentication failed after refresh attempt',
        type: DioExceptionType.badResponse,
```

```
        response: Response(
          requestOptions: request,
          statusCode: 401,
          statusMessage: 'Authentication failed',),))}
    if (statusCode == 404 &&
        request.path.contains('/v1/users/me') &&
        !request.extra.containsKey('retried')) {
      request.extra['retried'] = true;
      try {
        final refreshed = await apiClient.refreshTokens();
        if (refreshed) {
          final newToken = await apiClient.getAccessToken();
          if (newToken != null) {
            request.headers['Authorization'] = 'Bearer
$newToken';} else {
            return handler.reject(DioException(
              requestOptions: request,
              error: 'New token is null after refresh for 404',
              type: DioExceptionType.badResponse,
              response: Response(
                requestOptions: request,
                statusCode: 401,
                statusMessage: 'Authentication failed (404 was
auth related)',),));}
          final response = await apiClient.dio.fetch(request);
          return handler.resolve(response); catch (e) {
        print("Refresh failed for 404 (auth related): $e");
        await apiClient.storage.clearAll();}
      return handler.reject(DioException(
        requestOptions: request,
        error: 'Authentication failed for /users/me (404)',
        type: DioExceptionType.badResponse,
        response: Response(
          requestOptions: request,
          statusCode: 401,
          statusMessage: 'Authentication failed',),));}
    handler.next(err);}
```

Додаток К

Лістинг класу SecureStorage

```
class SecureStorage {
  static final SecureStorage _instance =
SecureStorage._internal();
  factory SecureStorage() {
    return _instance;}
  SecureStorage._internal();
  final _storage = const FlutterSecureStorage();
  Future<void> write(String key, String value) async {
    try {
      await _storage.write(key: key, value: value);
      log("Value saved for key: $key");
    } catch (e) {
      log("Error writing key ($key): $e");}}
  Future<String?> read(String key) async {
    try {
      return await _storage.read(key: key);
    } catch (e) {
      log("Error reading key ($key): $e");
      return null;}}
  Future<void> delete(String key) async {
    try {
      await _storage.delete(key: key);
      log("Value deleted for key: $key");
    } catch (e) {
      log("Error deleting key ($key): $e");}}
  Future<void> clearAll() async {
    try {
      await _storage.deleteAll();
      log("Secure storage cleared successfully!");
    } catch (e) {
      log("Error clearing storage: $e");}}
  Future<void> writeObject(String key, Map<String, dynamic>
value) async {
    try {
      final jsonString = jsonEncode(value);
      await _storage.write(key: key, value: jsonString);
      log("Object saved as JSON for key: $key");
    } catch (e) {
      log("Error saving object for key ($key): $e");}}
  Future<Map<String, dynamic>?> readObject(String key) async {
    try {
      final jsonString = await _storage.read(key: key);
      if (jsonString != null) {
        return jsonDecode(jsonString) as Map<String, dynamic>;}}
catch (e) {
      log("Error reading object for key ($key): $e");}
    return null;}
```

Додаток Л

Лістинг ThemeNotifier

```
class ThemeNotifier with ChangeNotifier {
  ThemeData _currentTheme = lightTheme;

  ThemeNotifier() {
    _loadThemeFromPreferences();
  }

  ThemeData get currentTheme => _currentTheme;

  void _loadThemeFromPreferences() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    String? themeName = prefs.getString('theme') ?? 'light';
    switch (themeName) {
      case 'light':
        _currentTheme = lightTheme;
        break;
      case 'darkBlue':
        _currentTheme = darkBlueTheme;
        break;
      case 'deepDark':
        _currentTheme = deepDarkTheme;
        break;
    }
    notifyListeners();
  }

  void _saveThemeToPreferences(String themeName) async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    prefs.setString('theme', themeName);
  }

  void setLightTheme() {
    _currentTheme = lightTheme;
    _saveThemeToPreferences('light');
    notifyListeners();
  }

  void setDarkBlueTheme() {
    _currentTheme = darkBlueTheme;
    _saveThemeToPreferences('darkBlue');
    notifyListeners();
  }
}
```

Додаток М

Лістинг CustomBottomNavigationBar

```
class CustomBottomNavigationBar extends StatelessWidget {
  final int currentIndex;
  final Function(int) onItemTapped;
  const CustomBottomNavigationBar({
    super.key,
    required this.currentIndex,
    required this.onItemTapped,});
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return SizedBox(
      height: MediaQuery.of(context).size.height * 0.08,
      child: BottomNavigationBar(
        selectedLabelStyle: TextStyle(
          fontSize: MediaQuery.of(context).size.height * 0.02,
          fontFamily: 'Carlito',),
        currentIndex: currentIndex,
        onTap: onItemTapped,
        items: [
          BottomNavigationBarItem(
            icon: SvgPicture.asset(
              'assets/icons/home.svg',
              height: MediaQuery.of(context).size.height * 0.03,
              color: currentIndex == 0
                  ?
theme.bottomNavigationBarTheme.selectedItemColor
                  :
theme.bottomNavigationBarTheme.unselectedItemColor,),
            label: 'Головна',),
          BottomNavigationBarItem(
            icon: SvgPicture.asset(
              'assets/icons/attendance.svg',
              height: MediaQuery.of(context).size.height * 0.03,
              color: currentIndex == 1
                  ?
theme.bottomNavigationBarTheme.selectedItemColor
                  :
theme.bottomNavigationBarTheme.unselectedItemColor,
            ),
            label: 'Відвідуваність',
          ),
          BottomNavigationBarItem(
            icon: SvgPicture.asset(
              'assets/icons/notifications.svg',
              height: MediaQuery.of(context).size.height * 0.03,
              color: currentIndex == 2
```

```
                    ?
theme.bottomNavigationBarTheme.selectedItemColor
                    :
theme.bottomNavigationBarTheme.unselectedItemColor,
              ),
              label: 'Сповіщення',
            ),
            BottomNavigationBarItem(
              icon: SvgPicture.asset(
                'assets/icons/grades.svg',
                height: MediaQuery.of(context).size.height * 0.03,
                color: currentIndex == 3
                    ?
theme.bottomNavigationBarTheme.selectedItemColor
                    :
theme.bottomNavigationBarTheme.unselectedItemColor,
              ),
              label: 'Оцінки',
            ),
            BottomNavigationBarItem(
              icon: SvgPicture.asset(
                'assets/icons/profile.svg',
                height: MediaQuery.of(context).size.height * 0.03,
                color: currentIndex == 4
                    ?
theme.bottomNavigationBarTheme.selectedItemColor
                    :
theme.bottomNavigationBarTheme.unselectedItemColor,
              ),
              label: 'Профіль',
            ),
          ],
        ),
      );
  }
}
```

Додаток Н

Лістинг класу FullScheduleInfo

```
class FullScheduleInfo {
  final int scheduleId;
  final String subjectName;
  final String group;
  final int groupId;
  final int periodId;
  final int subjectId;
  final int classNumber;
  final String auditorium;
  final String buildingNumber;
  final String scheduleTypeName;
  final String startTime;
  final String endTime;
  final int subgroupNumber;
  final bool isReplacement;
  final String weekType;
  final int weekdayIndex;
  final DateTime? date;
  final List<String> teacherFullNames;
  FullScheduleInfo({
    required this.scheduleId,
    required this.subjectName,
    required this.group,
    required this.groupId,
    required this.periodId,
    required this.subjectId,
    required this.classNumber,
    required this.auditorium,
    required this.buildingNumber, // Added
    required this.scheduleTypeName, // Added
    required this.startTime,
    required this.endTime,
    required this.subgroupNumber,
    required this.isReplacement,
    required this.weekType,
    required this.weekdayIndex,
    required this.date,
    required this.teacherFullNames,
  });
  static List<FullScheduleInfo> fromApiResponse(Map<String,
dynamic> data) {
    final teacherMap = {
      for (var t in data['teachers'])
        t['id']: {
          'first_name': t['first_name'],
          'last_name': t['last_name'],
          'midname': t['midname'],}};
    final subjectsList = data['subjects'] as List<dynamic>;
```

```
    final Map<int, FullScheduleInfo> scheduleMap = {};
    for (var subject in subjectsList) {
      final subjectName = subject['name'] as String;
      final scheduleList = subject['schedule'] as List<dynamic>;
      for (var schedule in scheduleList) {
        final scheduleId = schedule['shedule_id'] as int;
        final teacherIds =
            (schedule['teacher_ids'] as
List<dynamic>?)?.cast<String>() ?? [];
        final teacherFullNames = teacherIds.map((id) {
          final teacher = teacherMap[id];
          if (teacher != null) {
         final firstName = teacher['first_name']?.trim() ?? '';
            final lastName = teacher['last_name']?.trim() ?? '';
            final midName = teacher['midname']?.trim() ?? '';
            final initials = [
              if (firstName.isNotEmpty) '${firstName[0]}.',
              if (midName.isNotEmpty) '${midName[0]}.'
            ].join(' ');
            return [lastName, initials].join(' ').trim();}
          return 'Невідомо';}).toList();
        if (scheduleMap.containsKey(scheduleId)) {
          final existing = scheduleMap[scheduleId]!;
          final combinedTeachers =
              {...existing.teacherFullNames,
...teacherFullNames}.toList();
          scheduleMap[scheduleId] = existing.copyWith(
            teacherFullNames: combinedTeachers,
          );} else {
          scheduleMap[scheduleId] = FullScheduleInfo(
            scheduleId: scheduleId,
            subjectName: subjectName,
            group: schedule['group'] as String,
            groupId: schedule['group_id'] as int,
            periodId: schedule['period_id'] as int,
            subjectId: schedule['subject_id'] as int,
            classNumber:
                int.tryParse(schedule['class_number']?.toString(
) ?? '') ?? 0,
            auditorium: schedule['auditorium']?.toString() ??
'',
            buildingNumber:
                schedule['building_number']?.toString() ?? '',
            scheduleTypeName:
                schedule['schedule_type_name']?.toString() ?? ''
            startTime:
_formatTime(schedule['starttime']?.toString()),
            endTime:
_formatTime(schedule['endtime']?.toString()),
            subgroupNumber: schedule['subgroup_number'] as int?
?? 0,
```

```
            isReplacement: schedule['is_replacement'] as bool?
?? false,
            weekType: schedule['week_type']?.toString() ?? '',
            weekdayIndex:
                int.tryParse(schedule['weekday_index']?.toString
() ?? '') ?? 0,
            date: (schedule['date'] != null &&
                    schedule['date'].toString().isNotEmpty)
                ? DateTime.tryParse(schedule['date'].toString())
                : null,
            teacherFullNames: teacherFullNames,
          );
        }
      }
    }
    return scheduleMap.values.toList();
  }
  FullScheduleInfo copyWith({List<String>? teacherFullNames}) {
    return FullScheduleInfo(
      scheduleId: scheduleId,
      subjectName: subjectName,
      group: group,
      groupId: groupId,
      periodId: periodId,
      subjectId: subjectId,
      classNumber: classNumber,
      auditorium: auditorium,
      buildingNumber: buildingNumber,
      scheduleTypeName: scheduleTypeName,
      startTime: startTime,
      endTime: endTime,
      subgroupNumber: subgroupNumber,
      isReplacement: isReplacement,
      weekType: weekType,
      weekdayIndex: weekdayIndex,
      date: date,
      teacherFullNames: teacherFullNames ??
this.teacherFullNames,
    );
  }
}
String _formatTime(String? timeString) {
  if (timeString == null || timeString.isEmpty) return '';
  try {
    final parsedTime = DateTime.tryParse('1970-01-
01T$timeString');
    if (parsedTime != null) {
      return '${parsedTime.hour.toString().padLeft(2,
'0')}:${parsedTime.minute.toString().padLeft(2, '0')}';
    }
  } catch (_) {}
  return timeString;}
```

Додаток П

Лістинг віджета DayScheduleCard

```
class DayScheduleCard extends StatelessWidget {
  final DateTime date;
  final List<FullScheduleInfo> lessons;
  final String weekType;
  final String role;
  const DayScheduleCard({
    super.key,
    required this.date,
    required this.lessons,
    required this.weekType,
    required this.role,
  });
  String _formatDayLabel(DateTime targetDate) {
    final now = DateTime.now();
    final weekday = DateFormat.EEEE('uk_UA').format(targetDate);
    if (targetDate.year == now.year &&
        targetDate.month == now.month &&
        targetDate.day == now.day &&
        weekday != 'неділя') {
      return "сьогодні";}
    final dateFormatted =
DateFormat('dd.MM').format(targetDate);
    return '$weekday $dateFormatted';}
  @override
  Widget build(BuildContext context) {
    final ScrollController scrollController =
ScrollController();
    final theme = Theme.of(context);
    final sortedLessons = [...lessons]
      ..sort((a, b) => a.classNumber.compareTo(b.classNumber));
    final replacedLessonNumbers = sortedLessons
        .where((lesson) => lesson.isReplacement)
        .map((lesson) => lesson.classNumber)
        .toSet();
    final lessonsToDisplay = sortedLessons.where((lesson) {
      if (lesson.isReplacement) return true;
      return
!replacedLessonNumbers.contains(lesson.classNumber);
    }).toList();
    return Column(
      children: [
        Padding(
          padding: const EdgeInsets.symmetric(vertical: 8.0),
          child: AutoSizeText(
            _formatDayLabel(date),
            style: TextStyle(
                fontSize: 18,
```

```
                fontWeight: FontWeight.bold,
                color: theme.primaryColorDark,
                fontFamily: 'Jura'),
        Container(
          clipBehavior: Clip.hardEdge,
          height: MediaQuery.of(context).size.height * 0.66,
          width: MediaQuery.of(context).size.width * 0.80,
          decoration: BoxDecoration(
            color: theme.cardColor,
            borderRadius: BorderRadius.circular(30),
            border: Border.all(color: theme.primaryColorDark,
width: 2),
            boxShadow: [
              BoxShadow(
                color: theme.primaryColorDark,
                blurRadius: 10,
                offset: const Offset(0, 5),),],),
          child: Scrollbar(
            thumbVisibility: false,
            controller: scrollController,
            radius: Radius.circular(10),
            child: SingleChildScrollView(
              controller: scrollController,
              child: Column(
                children: lessonsToDisplay.mapIndexed((i,
lesson) { return Column(
                    children: [
                      Padding(
                        padding: const EdgeInsets.symmetric(
                          vertical: 12.0,
                          horizontal: 16.0,
                        ),
                        child: Column(
                          crossAxisAlignment:
CrossAxisAlignment.start,
                          children: [
                            Row(
                              children: [
                                if (lesson.isReplacement)
                                  Container(
                                    margin: const
EdgeInsets.only(right: 8),
                                    padding: const
EdgeInsets.symmetric(
                                        horizontal: 6, vertical:
2),
                                    decoration: BoxDecoration(
                                      color: Colors.orange,
                                      borderRadius:
BorderRadius.circular(4),
                                    ),
                                    child: const AutoSizeText(
```

```
                                        'заміна',
                                        style: TextStyle(
                                            fontSize: 10, color:
Colors.white),),),
                                if
(lesson.scheduleTypeName.isNotEmpty &&
                                    lesson.scheduleTypeName !=
'Пара')
                                  Container(
                                    margin: const
EdgeInsets.only(right: 8),
                                    padding: const
EdgeInsets.symmetric(
                                        horizontal: 6, vertical:
2),
                                    decoration: BoxDecoration(
                                      color: Colors.green,
                                      borderRadius:
BorderRadius.circular(4),
                                    ),
                                    child: AutoSizeText(
                                      lesson.scheduleTypeName,
                                      style: const TextStyle(
                                          fontSize: 10, color:
Colors.white),
                                    ),
                                  ),

                              const SizedBox(height: 8),
                              Center(
                                child: AutoSizeText(
                                  lesson.subjectName,
                                  maxLines: 3,
                                  minFontSize: 10.0,
                                  overflow: TextOverflow.ellipsis,
                                  textAlign: TextAlign.center,
                                  style: TextStyle(
                                    fontSize:
MediaQuery.of(context).size.height *
                                        0.022,
                                    fontFamily: 'Carlito',
                                    fontWeight:
FontWeight.bold,),),),),
                              const SizedBox(height: 8),
                              Row(
                                children: [
                                  AutoSizeText(
                                    '${lesson.startTime} -
${lesson.endTime}',
                                    style: TextStyle(
                                      fontSize:
```

```
                                        MediaQuery.of(context).s
ize.height *
                                            0.019,
                                      fontFamily: 'Jura',
                                      fontWeight:
FontWeight.w600,),),
                                  const Spacer(),
                                  AutoSizeText(
                                    '${lesson.classNumber} пара',
                                    style: TextStyle(
                                      fontSize:
                                          MediaQuery.of(context).s
ize.height *0.019                                   fontFamily:
'Jura',
                                      fontWeight: FontWeight.w600,
                                    ),
                            останньої, якщо менше 5
                            if (i != lessonsToDisplay.length - 1 ||
                                lessonsToDisplay.length < 5)
                              Divider(
                                thickness: 1,
                                height: 1,
                                color: theme.primaryColorDark,
                                indent: 0,
                                endIndent: 0,),],);}).toList],);}
```

Додаток Р

Лістинг коду панелі керування розкладом

```
class EditScheduleAdminPage extends StatelessWidget {
  const EditScheduleAdminPage({super.key});
  void _showAddDialog(BuildContext context) {
    final _formKey = GlobalKey<FormState>();
    final _subjectController = TextEditingController();
    final _groupController = TextEditingController();
    final _timeController = TextEditingController();
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Додати нову пару'),
        content: Form(
          key: _formKey,
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              TextFormField(
                controller: _subjectController,
                decoration: const InputDecoration(labelText:
'Предмет'),
                validator: (val) =>
                    val == null || val.isEmpty ? 'Введіть
предмет' : null,
              ),
              TextFormField(
                controller: _groupController,
                decoration: const InputDecoration(labelText:
'Група'),
                validator: (val) =>
                    val == null || val.isEmpty ? 'Введіть групу'
: null,),
              TextFormField(
                controller: _timeController,
                decoration: const InputDecoration(labelText:
'Час'),
                validator: (val) =>
                    val == null || val.isEmpty ? 'Введіть час' :
null,),],),),),
actions: [
          TextButton(onPressed: () => Navigator.pop(context),
child: const Text('Скасувати')),
          ElevatedButton(
            onPressed: () {
              if (_formKey.currentState!.validate()) {
                print('Додано: ${_subjectController.text},
${_groupController.text}, ${_timeController.text}');
                Navigator.pop(context); }},
```

```
            child: const Text('Додати'),)        ],)    );}
  void _showEditDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Редагувати розклад'),
        content: const Text('Тут має бути вибір пари та форма
редагування.'),
        actions: [
          TextButton(onPressed: () => Navigator.pop(context),
child: const Text('OK')),]      ),    );}
  void _showDeleteDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Видалити розклад'),
        content: const Text('Ви впевнені, що хочете видалити всі
пари?'),
        actions: [
          TextButton(onPressed: () => Navigator.pop(context),
child: const Text('Скасувати')),
          ElevatedButton(
            onPressed: () {
              print('Розклад видалено');
              Navigator.pop(context);
            },
            style: ElevatedButton.styleFrom(backgroundColor:
Colors.red),
            child: const Text('Видалити'),)        ],), );}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Редагування розкладу'),
        automaticallyImplyLeading: false,),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const Text(
              'Панель адміністратора',
              style: TextStyle(fontSize: 20, fontWeight:
FontWeight.bold),),
            const SizedBox(height: 20),
            ElevatedButton.icon(
              onPressed: () => _showAddDialog(context),
              icon: const Icon(Icons.add),
              label: const Text('Додати нову пару'),),
            const SizedBox(height: 12),
            ElevatedButton.icon(
              onPressed: () => _showEditDialog(context),
```

```
              icon: const Icon(Icons.edit),
              label: const Text('Редагувати розклад'),),
            const SizedBox(height: 12),
            ElevatedButton.icon(
              onPressed: () => _showDeleteDialog(context),
              icon: const Icon(Icons.delete),
              label: const Text('Видалити розклад'),
              style: ElevatedButton.styleFrom(backgroundColor:
Colors.red),),
          ],
        ),
      ),
    );
  }}
```

Додаток С

Лістинг методу exportToExcel

```
Future<void> exportToExcel(
  BuildContext context,
  AttendanceReportData attendanceData,
) async {
  var excel = Excel.createExcel();
  Sheet sheet = excel['Sheet1'];
  int currentHeaderCol = 0;
  sheet
      .cell(CellIndex.indexByColumnRow(
          columnIndex: currentHeaderCol, rowIndex: 0))
      .value = '№ пп';
  sheet.merge(
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 0),
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 1));
  currentHeaderCol++;
  sheet
      .cell(CellIndex.indexByColumnRow(
          columnIndex: currentHeaderCol, rowIndex: 0))
      .value = 'Прізвище, ім\'я, по батькові студента';
  sheet.merge(
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 0),
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 1));
  currentHeaderCol++; // col 1
  final initialPeriodColIndex = currentHeaderCol; // Store where
periods start
  for (int i = 0; i < attendanceData.periods.length; i++) {
    final period = attendanceData.periods[i];
    final periodHeader =
        '${DateFormat('dd.MM').format(period.startDate)}-
${DateFormat('dd.MM').format(period.endDate)}';
    sheet
        .cell(CellIndex.indexByColumnRow(
            columnIndex: currentHeaderCol, rowIndex: 0))
        .value = periodHeader;
    sheet.merge(
        CellIndex.indexByColumnRow(columnIndex:
currentHeaderCol, rowIndex: 0),
        CellIndex.indexByColumnRow(columnIndex:
currentHeaderCol, rowIndex: 1));
    currentHeaderCol++;
  }
  sheet
      .cell(CellIndex.indexByColumnRow(
```

```
          columnIndex: currentHeaderCol, rowIndex: 0))
      .value = 'Всього';
  sheet.merge(
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 0),
      CellIndex.indexByColumnRow(columnIndex: currentHeaderCol,
rowIndex: 1));
  currentHeaderCol++;
under it)
  final zNyhStartCol = currentHeaderCol;
  sheet
      .cell(CellIndex.indexByColumnRow(columnIndex:
zNyhStartCol, rowIndex: 0))
      .value = 'з них';
  sheet.merge(
      CellIndex.indexByColumnRow(columnIndex: zNyhStartCol,
rowIndex: 0),
      CellIndex.indexByColumnRow(columnIndex: zNyhStartCol + 2,
rowIndex: 0));
      .cell(CellIndex.indexByColumnRow(columnIndex:
zNyhStartCol, rowIndex: 1))
      .value = 'по хворобі';
  sheet
      .cell(CellIndex.indexByColumnRow(
          columnIndex: zNyhStartCol + 1, rowIndex: 1))
      .value = 'з поважних причин';
  sheet
      .cell(CellIndex.indexByColumnRow(
          columnIndex: zNyhStartCol + 2, rowIndex: 1))
      .value = 'без поважних причин';
  int currentRowIndex = 2; // Data starts from the 3rd row
(index 2)
  for (final group in attendanceData.groups) {
    for (int studentIndex = 0;
        studentIndex < group.students.length;
        studentIndex++) {
      final student = group.students[studentIndex];
      int currentDataCol = 0;
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = studentIndex + 1; // Serial number
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = student.fullName;
      for (final periodAbsence in student.periodAbsences) {
        sheet
            .cell(CellIndex.indexByColumnRow(
```

```
                columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
            .value = periodAbsence.total;
      }
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = student.totalAbsences;
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = student.illnessAbsences;
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = student.excusedAbsences;
      final unexcusedAbsences = student.totalAbsences -
          student.excusedAbsences -
          student.illnessAbsences;
      sheet
          .cell(CellIndex.indexByColumnRow(
              columnIndex: currentDataCol++, rowIndex:
currentRowIndex))
          .value = unexcusedAbsences;
      currentRowIndex++;}}
  final now = DateTime.now();
  final fileName =
      'attendance_${DateFormat('yyyyMMdd_HHmmss').format(now)}.x
lsx';
  var status = await Permission.manageExternalStorage.status;
  if (!status.isGranted) {
    status = await Permission.manageExternalStorage.request();}
  if (status.isGranted) {
    String? downloadDirectoryPath;
    try {
      final externalDir = await getExternalStorageDirectory();
      if (externalDir != null) {
        List<String> pathSegments = externalDir.path.split('/');
        int androidIndex = pathSegments.indexOf('Android');
        if (androidIndex != -1) {
          downloadDirectoryPath =
              pathSegments.sublist(0, androidIndex).join('/');
          downloadDirectoryPath =
'$downloadDirectoryPath/Download';
        } else {
          downloadDirectoryPath =
'/storage/emulated/0/Download';
        }} else {
        downloadDirectoryPath = '/storage/emulated/0/Download';}
```

```
      final filePath = '$downloadDirectoryPath/$fileName';
      print('Attempting to save to: $filePath');
      final bytes = excel.save()!;
      File file = File(filePath);
      if (!await file.parent.exists()) {
        await file.parent.create(recursive: true);
        print('Created directory: ${file.parent.path}');
      }
      await file.writeAsBytes(bytes);

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Файл збережено: $filePath')),
      );
      print('File saved successfully to: $filePath');
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Помилка при збереженні файлу:
$e')),
      );
      print('Error saving file: $e');
    }
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
          content: Text(
              'Доступ до всіх файлів відхилено. Будь ласка,
надайте дозвіл у налаштуваннях програми.')),
    );
    await openAppSettings();
  }
}
```

Додаток Т

Лістинг віджету GradesTable

```
class GradesTable extends StatelessWidget {
  final List<FullGradeInfo> grades;
  const GradesTable({Key? key, required this.grades}) :
super(key: key);
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    if (grades.isEmpty) {
      return Center(child: Text('Немає оцінок для цієї
категорії'));
    }
    return Container(
      margin: const EdgeInsets.all(16),
      padding: const EdgeInsets.all(0),
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(24),
        border: Border.all(
          color: theme.primaryColorDark,
          width: 3,
        ),
      ),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(21),
        child: Table(
          columnWidths: const {
            0: FlexColumnWidth(3),
            1: FlexColumnWidth(2),
            2: FlexColumnWidth(1),
          },
          border: TableBorder(
            horizontalInside: BorderSide(
              color: theme.primaryColorDark,
              width: 1,
            ),
            verticalInside: BorderSide(
              color: theme.primaryColorDark,
              width: 1,
            ),
          ),
          children: [
            const TableRow(
              children: [
                _TableHeaderCell('Назва предмету'),
                _TableHeaderCell('Викладач'),
                _TableHeaderCell('Оцінка'),
              ],
```

```
            ),
            ...grades.map((grade) {
              return TableRow(
                children: [
                  _TableCell(grade.subjectName),
                  _TableCell(grade.teacherFullNames.join(', ')),
                  _TableCell(grade.grade.toString()),
                ],
              );
            }),
          ],
        ),
      ),
    );
  }
}
class _TableHeaderCell extends StatelessWidget {
  final String text;

  const _TableHeaderCell(this.text);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.fromLTRB(1, 12, 1, 12),
      child: Text(
        text,
        style: const TextStyle(fontWeight: FontWeight.bold),
        textAlign: TextAlign.center,
      ),
    );
  }
}

class _TableCell extends StatelessWidget {
  final String text;

  const _TableCell(this.text);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.fromLTRB(3, 12, 3, 12),
      child: Text(
        text,
        textAlign: TextAlign.center,
      ),
    );
  }
}
```

Додаток У

Лістинг функції _fetchUserInfo

```
Future<void> _fetchUserInfo() async {
    final storage = SecureStorage();
    final apiClient = RestAPIClient(baseUrl, storage);

    String? fetchedName = await storage.read('name');
    String? fetchedSurname = await storage.read('surname');

    if (fetchedName == null || fetchedSurname == null) {
      await apiClient.getUserInfo();
      fetchedName = await storage.read('name');
      fetchedSurname = await storage.read('surname');
    }

    final fullGrades = await apiClient.fetchGrades();

    final gradeTypes = fullGrades.map((g) =>
g.gradeType).toSet();

    final firstGradeType = gradeTypes.isNotEmpty ?
gradeTypes.first : null;
    final filteredGrades = firstGradeType == null
        ? fullGrades
        : fullGrades.where((g) => g.gradeType ==
firstGradeType).toList();

    final semestersForExam = fullGrades
        .where((g) => g.gradeType == 'Залік')
        .map((g) => g.semester.toString())
        .toSet()
        .toList()
      ..sort((a, b) => int.parse(a).compareTo(int.parse(b)));

    setState(() {
      name = fetchedName ?? '';
      surname = fetchedSurname ?? '';
      allGrades = fullGrades;
      selectedGradeType = firstGradeType;
      selectedSemesters = semestersForExam;
      selectedSemester = null;
      isLoading = false;
    });
  }
```

Додаток Ф

Лістинг сторінки сповіщень студента

```
class NotificationScreen extends StatefulWidget {
  const NotificationScreen({super.key});
  @override
  _NotificationScreenState createState() =>
_NotificationScreenState();}
class _NotificationScreenState extends
State<NotificationScreen>{
  int _currentIndex = 2;
  bool _isLoading = true;
  NotificationResponse? _notificationResponse;
  late final RestAPIClient _apiClient;
  String? _userRole;
  @override
  void initState() {
    super.initState();
    final storage = SecureStorage();
    _apiClient = RestAPIClient(baseUrl, storage);
    _loadNotifications();
    _loadUserRole();}
  Future<void> _loadUserRole() async {
    final storage = SecureStorage();
    final role = await storage.read('role');
    setState(() {
      _userRole = role;});}
  Future<void> _loadNotifications() async {
    try {
      final response = await _apiClient.fetchNotifications();
      setState(() {
        _notificationResponse = response;
        _isLoading = false;});} catch (e) {
);setState(() => _isLoading = false);}}
  void _onItemTapped(int index) {
    if (_currentIndex != index) {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) =>
appPages[index]),);}}
  void _openSendNotificationPage() {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (_) => const
SendNotificationScreen()),);
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    final notificationWidgets =
```

```
        _notificationResponse?.notifications.expand((group)
{              final fullName = '${group.surname}
${group.name}';
              return group.notifications.map((n) =>
NotificationCard(
                    item: n,
                    senderFullName: fullName,
                    apiClient: _apiClient,
                    onRead: _loadNotifications, // <<< HERE
                  ));
            }).toList() ??
            [];
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        toolbarHeight: MediaQuery.of(context).size.height *
0.09,title: Row(
          children: [
            SvgPicture.asset(
              'assets/icons/logo.svg',
              height: MediaQuery.of(context).size.height * 0.05,
              color: theme.iconTheme.color,
            ),
            const SizedBox(width: 4),
        Text(
              'Сповіщення',
              style: TextStyle(
          fontSize: MediaQuery.of(context).size.width *
0.05,),),const Spacer(),],),),body: _isLoading
          ? const Center(child: CircularProgressIndicator())
          : ListView(
              padding: const EdgeInsets.all(12),
              children: notificationWidgets.isEmpty? [Center(
child: Padding(
 padding: const EdgeInsets.symmetric(vertical: 40),
child: Text("Сповіщень ще немає",style: TextStyle(fontSize: 16,
 color: theme.primaryColorDark,
                                fontStyle: FontStyle.italic,),
                            ),
                          ),
                        )
                    ]
                  : notificationWidgets,
            ),
      bottomNavigationBar: CustomBottomNavigationBar(
        currentIndex: _currentIndex,
        onItemTapped: _onItemTapped,
      ),
    );
  }
}
```

Додаток Х

Лістинг функції sendNotificationToServer

```
Future<void> _sendNotification() async {
    if (!_formKey.currentState!.validate() || _selectedTypeId ==
null) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Заповніть всі поля')),);
      return;}
    final sendingToGroup = _selectedGroupId != null;
    final sendingToTeachers = _selectedTeacherIds.isNotEmpty;
    if (!sendingToGroup && (_userRole != 'Teacher' &&
!sendingToTeachers)) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Оберіть групу або
викладачів')),);return;}
    if (_userRole == 'Teacher' && !sendingToGroup) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Викладач може надсилати
лише групам')),);return;}
    setState(() => _sending = true);
    final storage = SecureStorage();
    final senderId = await storage.read('id');
    try {
      if (sendingToGroup) {
        await _apiClient.sendNotificationToGroup(
          groupId: _selectedGroupId!,
          senderId: senderId!,
          typeId: _selectedTypeId!,
          text: _messageController.text,);
      } else if (sendingToTeachers && _userRole != 'Teacher') {
        for (final userId in _selectedTeacherIds) {
          await _apiClient.sendNotificationToUser(
            userId: userId,
            senderId: senderId!,
            typeId: _selectedTypeId!,
            text: _messageController.text,);}}
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Сповіщення надіслано')),);
      Navigator.pop(context);
    } catch (e) {
      print('[SEND ERROR] $e');
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Помилка: $e')),);
    } finally {
      setState(() => _sending = false);}
  }
```

Додаток Ц

Лістинг сторінки надсилання сповіщень

```
class SendNotificationScreen extends StatefulWidget {
  const SendNotificationScreen({super.key});
  @override
  State<SendNotificationScreen> createState() =>
_SendNotificationScreenState();}
class _SendNotificationScreenState extends
State<SendNotificationScreen> {
  final _formKey = GlobalKey<FormState>();
  final _messageController = TextEditingController();
  bool _sending = false;
  int? _selectedTypeId;
  int? _selectedGroupId;
  Set<String> _selectedTeacherIds = {};
  List<Group> _groups = [];
  List<Map<String, dynamic>> _teachers = [];
  String? _userRole;
  final List<Map<String, dynamic>> _notificationTypes = [
    {'id': 1, 'name': 'Зміна в розкладі'},
    {'id': 2, 'name': 'Подія'},
    {'id': 3, 'name': 'Нагадування'},];
  late final RestAPIClient _apiClient;
  @override
  void initState() {
    super.initState();
    final storage = SecureStorage();
    _apiClient = RestAPIClient(baseUrl, storage);
    _loadUserRoleAndData();}
  Future<void> _loadUserRoleAndData() async {
    final storage = SecureStorage();
    final role = await storage.read('role');
    setState(() {
      _userRole = role;});
    try {
      if (role == 'Teacher') {setState(() {_groups = [Group(id:
18, groupName: 'KH-91'),Group(id: 19, groupName: 'KI-
11'),Group(id: 20, groupName: 'KH-21'),];});} else {
        final departmentData = await
_apiClient.fetchDepartmentInfo();
        final teacherJsonList = await
_apiClient.fetchAllTeachers();
        setState(() {
          _groups = departmentData.departments
              .expand((d) => d.specialties)
              .expand((s) => s.groups)
              .toList();
          _teachers = teacherJsonList;});}
    } catch (e) {
```

```
      print('[LOAD ERROR] $e');}}
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(
      appBar: AppBar(title: const Text('Надіслати сповіщення')),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              const Text('Тип повідомлення:',
                  style: TextStyle(fontWeight:
FontWeight.bold)),
              DropdownButtonFormField<int>(
                value: _selectedTypeId,
                decoration: const InputDecoration(border:
OutlineInputBorder()),
                items:
_notificationTypes.map<DropdownMenuItem<int>>((type) {
                  return DropdownMenuItem<int>(
                    value: type['id'] as int,
                    child: Text(type['name'] as String),);
                }).toList(),
                onChanged: (val) {
                  setState(() {
                    _selectedTypeId = val;});},
                validator: (val) =>
            val == null ? 'Оберіть тип повідомлення' : null, ),
              const SizedBox(height: 20),
              const Text('Оберіть групу:',
                  style: TextStyle(fontWeight:
FontWeight.bold)),
              Wrap(
                spacing: 8,
                children: _groups.map((group) {
                  final selected = _selectedGroupId == group.id;
                  return ChoiceChip(
                    label: Text(group.groupName),
                    selected: selected,
                    onSelected: (_) {
                      setState(() {
                  _selectedGroupId = group.id;
                _selectedTeacherIds.clear();});},);}).toList(),),
              const SizedBox(height: 20),
if (_userRole != 'Teacher') ...[
const Text('Оберіть викладачів:',
style: TextStyle(fontWeight: FontWeight.bold)),
```

```
Wrap(spacing: 8,children: _teachers.map((teacher) { final id =
teacher['id'] as String;final name = '${teacher['surname']}
${teacher['name']}';
final selected = _selectedTeacherIds.contains(id);
return FilterChip(
label: Text(name),
selected: selected,
onSelected: (_) {
setState(() {
if (selected) {
_selectedTeacherIds.remove(id);
} else {
_selectedTeacherIds.add(id);
_selectedGroupId = null; }});},);}).toList(),),
const SizedBox(height: 20),],
TextFormField(
  controller: _messageController,
decoration: const InputDecoration(
labelText: 'Текст повідомлення',
border: OutlineInputBorder(),),
maxLines: 4,
validator: (val) => val == null || val.isEmpty
? 'Введіть текст повідомлення': null,),
const SizedBox(height: 20 Center(
child: ElevatedButton(
onPressed: _sending ? null : _sendNotification,
style: ElevatedButton.styleFrom(
backgroundColor: theme.primaryColorDark,
padding: const EdgeInsets.symmetric(
horizontal: 32, vertical: 14),),
child: _sending
    ? const CircularProgressIndicator(): const
Text('Надіслати'),),)
            ],
          ),
        ),
      ),
    );
  }
}
```

Лістинг Ш

Лістинг сторінки профілю

```
class ProfileScreen extends StatefulWidget {
  const ProfileScreen({super.key});
  @override
  _ProfileScreenState createState() => _ProfileScreenState();}
class _ProfileScreenState extends State<ProfileScreen> {
  String? _name;
  String? _surname;
  int _currentIndex = 4;
  void _onItemTapped(int index) {
    setState(() {
      _currentIndex = index;});
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) =>
appPages[index]),);}
  @override
  void initState() {
    super.initState();
    _loadUserInfo();}
  Future<void> _loadUserInfo() async {
    final storage = SecureStorage();
    final name = await storage.read('name');
    final surname = await storage.read('surname');
    setState(() {
      _name = name;
      _surname = surname;})}
  void _logout() async {
    final storage = SecureStorage();
    await storage.clearAll();
    if (!mounted) return;
    Navigator.pushAndRemoveUntil(
      context,
      MaterialPageRoute(builder: (_) => const LoginScreen()),
      (route) => false,);}
  void _showThemeMenu(ThemeNotifier themeNotifier) {
    showModalBottomSheet(
      context: context,
      builder: (_) {
        return Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            ListTile(
              leading: const Icon(Icons.light_mode),
              title: const Text('Світла тема'),
              onTap: () {
                themeNotifier.setLightTheme();
                Navigator.pop(context);},),
```

```
            ListTile(
              leading: const Icon(Icons.dark_mode),
              title: const Text('Темна тема'),
              onTap: () {
                themeNotifier.setDarkBlueTheme();
                Navigator.pop(context);},),],);},);}
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    final themeNotifier = Provider.of<ThemeNotifier>(context)
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: const Text('Профіль'),
        backgroundColor: theme.primaryColor,),
      body: Center(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisSize: MainAxisSize.min,
          children: [
            const SizedBox(height: 20),
            CircleAvatar(
              radius: 40,
              backgroundColor: theme.primaryColorLight,
              child: Text(
                (_name != null && _surname != null)
                    ? _name![0].toUpperCase() +
_surname![0].toUpperCase()
                    : '',
                style: const TextStyle(fontSize: 24),),),
            const SizedBox(height: 12),
            Text(
              (_name != null && _surname != null)
                  ? '$_name $_surname'
                  : 'Завантаження...',
              style: const TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),),
            const SizedBox(height: 24),
            ElevatedButton.icon(
              onPressed: () => _showThemeMenu(themeNotifier),
              icon: const Icon(Icons.color_lens),
              label: const Text('Змінити тему'),),
            const SizedBox(height: 24),
            ElevatedButton.icon(
              onPressed: _logout,
              icon: const Icon(Icons.logout),
              label: const Text('Вийти з акаунту'),
              style: ElevatedButton.styleFrom(backgroundColor:
Colors.red),),
const SizedBox(height: 24),],),),
bottomNavigationBar: CustomBottomNavigationBar(
currentIndex: _currentIndex,onItemTapped: _onItemTapped,),);}}
```