Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка програмного забезпечення системи обліку та управління навчальними заняттями на базі Flutter»

Виконав(ла): студент(ка) 4 курсу, групи СПс-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Козлівський В.В. (прізвище та ініціали)

Керівник (підпис) Бачинський М.В. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Козлівському Володимиру Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення системи обліку та управління навчальними заняттями на базі Flutter»

Керівник роботи Бачинський Михайло Володимирович, к.т.н., доцент кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина. Розділ 1. Аналіз предметної області та аналіз існуючих рішень. Розділ 2. Проєктування та реалізація програмного рішення. Розділ 3. Тестування те верифікація програмного забезпечення. Розділ 4. Безпека життєдіяльності та основи охорони праці. Висновки. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми UML, ER-діаграма бази даних, ілюстрації роботи бізнес логіки окремих компонентів системи, знімки екрану віртуального смартфону з інтерфейсом додатку, зображення інтерфейсу автогенерованої документації серверної частини, слайди презентації на захист кваліфікаційної роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці			
Нормоконторь			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Узгодження теми кваліфікаційної роботи з керівником		
2	Ознайомлення із завданням до кваліфікаційної роботи		
3	Пошук інформації по темі кваліфікаційної роботи		
4	Аналіз предметної області та існуючих рішень		
5	Вибір мов та фреймворків реалізації програмної системи		
6	Проєктування архітектури програмної системи		
7	Реалізація клієнтської та серверної частин		
8	Тестування розробленої системи		
9	Розробка звіту виконаної роботи		
10	Виконання завдань безпеки життєдіяльності		
11	Виконання завдання основи охорони праці		
12	Оформлення кваліфікаційної роботи		
13	Нормоконтроль		
14	Перевірка на плагіат		
15	Попередній захист кваліфікаційної роботи		
16	Захист кваліфікаційної роботи		

Студент _____ Козлівський В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бачинський М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконав Козлівський Володимир Віталойович, студент групи СПс-41 Тернопільського національного технічного університету імені Івана Пулюя. Тема роботи присвячена проєктуванню та розробці програмного забезпечення системи обліку та управління навчальними заняттями на базі Flutter. Робота має обсяг 68 сторінок, містить 32 рисунків, 6 таблиць, 3 додатки та бібліографію з 27 джерел.

Ключові слова: мобільний застосунок, розклад занять, Flutter, Django, Firebase, чиста архітектура, push-сповіщення, REST API, тестування.

Основною метою кваліфікаційної роботи є проєктування архітектури та розробка програмного забезпечення, зокрема, мобільного додатку, для обліку та управління навчальними заняттями.

Цілі кваліфікаційної роботи включають дослідження обраного домену, аналіз складних випадків предметної області, аналіз популярних рішень реалізації, а також побудова архітектури застосунку.

У першому розділі досліджено предметну область управління навчальними заняттями. Виявлено ключові процеси, обґрунтовано вибір технологічного стеку Flutter, Django та Firebase.

У другому розділі спроєктовано та реалізовано повну систему TutorGlide на базі клієнт-серверної архітектури. Реалізовано підсистему розкладу занять з підтримкою повторюваних та одноразових занять.

У третьому розділі описано комплексну стратегію тестування системи, реалізовано модульні, інтеграційні, а також автоматизовані тести.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці при розробці програмного забезпечення.

Результатом роботи є кросплатформний мобільний додаток, який вирішує проблеми управління навчальними заняттями як на рівні освітніх організацій та закладів, так і на рівні приватного репетиторства.

ABSTRACT

Bachelor's qualification thesis completed by Kozlivskyi Volodymyr, a student of group SPs-41 at Ternopil Ivan Puluj National Technical University. The thesis is dedicated to the design and development of software for a system of accounting and management of educational sessions based on Flutter. The work comprises 68 pages, contains 32 figures, 6 tables, 3 appendices, and a bibliography of 27 sources.

Keywords: mobile application, class schedule, Flutter, Django, Firebase, clean architecture, push notifications, REST API, testing.

The primary objective of the qualification thesis is to design the architecture and develop software, specifically a mobile application, for accounting and management of educational sessions.

The goals of the qualification thesis include researching the selected domain, analysing complex cases within the subject area, reviewing popular implementation solutions, and constructing the application architecture.

The first chapter examines the subject area of educational session management. Key processes are identified, a comparative analysis of mobile development approaches and server-side frameworks is conducted, and the choice of the technology stack Flutter, Django, and Firebase is justified.

The second chapter covers the design and implementation of the complete TutorGlide system based on a client-server architecture. A scheduling subsystem with support for both recurring and one-time sessions has been implemented.

The third chapter describes the comprehensive testing strategy for the system, including unit, integration, and automated tests.

The fourth chapter addresses occupational health, safety, and working conditions in the context of software development.

The outcome of the thesis is a cross-platform mobile application that addresses the challenges of educational session management both at the level of educational organisations and institutions, and in the context of private tutoring.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface (інтерфейс програмування додатків)

BLoC – Business Logic Component (компонент бізнес-логіки)

CI – Continuous Integration (безперервна інтеграція)

DI – Dependency Injection (ін'єкція залежностей)

DRF – Django REST Framework (фреймворк побудови REST API на Django)

E2E – End-to-End (наскрізне тестування)

FCM – Firebase Cloud Messaging (хмарний сервіс сповіщень від Firebase)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

IANA – Internet Assigned Numbers Authority (організація з розподілу адрес)

JWT – JSON Web Token (стандарт передачі даних у вигляді JSON-об'єкта)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

REST – Representational State Transfer (передача репрезентативного стану)

TDD – Test-Driven Development (розробка через тестування)

UI – User Interface (інтерфейс користувача)

UUID – Universally Unique Identifier (універсальний ідентифікатор)

UX – User Experience (досвід користувача)

ЗМІСТ

ВСТУП 9
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ УПРАВЛІННЯ НАВЧАЛЬНИМИ ЗАНЯТТЯМИ 10
1.1 Характеристика предметної області 10
1.2 Огляд та аналіз існуючих рішень 12
1.3 Аналіз специфіки та складних випадків предметної області 14
1.4 Формування вимог до системи 17
1.4.1 Функціональні вимоги 17
1.4.2 Нефункціональні вимоги 18
1.5 Актори системи та опис сценаріїв використання 19
1.6 Порівняльний аналіз підходів до розробки мобільних систем 20
2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ОБЛІКУ НАВЧАЛЬНИХ ЗАНЯТЬ 23
2.1 Загальна архітектура системи 23
2.2 Архітектура та реалізація серверної частини 26
2.2.1 Модель даних та ER-діаграма 26
2.2.2 Сервісний шар та шар вибірки даних 27
2.2.3 Проєктування REST API ендпоінтів та OpenAPI-специфікації 29
2.3 Архітектура та реалізація мобільного клієнта на Flutter 31
2.3.1 Опис шарів та їх відповідальність 32
2.3.2 Управління станом та обробка потоків подій 34
2.3.3 Ін'єкція залежностей (GetIt) та конфігурація DI-контейнера 36
2.3.4 Маршрутизація та навігація 36
2.3.5 Створення HTTP-клієнта та стратегія обробки помилок 37
2.4 Реалізація підсистеми розкладу 37
2.4.1 Модель повторюваних та одноразових занять 38
2.4.2 Алгоритм генерації розкладу для заданого діапазону дат 40
2.4.3 Управління часовими зонами на клієнті 41
3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 42
3.1 Стратегія тестування 42
3.2 Наскрізне та ручне тестування 43
3.2.1 Автоматизоване E2E-тестування 44

3.2.2 Ручне тестування інтерфейсу 47
3.3 Безперервна інтеграція 51
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 54
4.1 Моделювання та прогнозування небезпечних ситуацій 54
4.2 Вимоги ергономіки до організації робочого місця оператора ПК 56
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60

ВСТУП

В умовах стрімкого розвитку ринку репетиторських послуг та зростання попиту на навчання в закладах освіти, зростає й потреба у спеціалізованих цифрових інструментах для управління навчальними заняттями. Вчителі, що працюють з кількома студентами одночасно, стикаються з проблемами планування розкладу, відстеження змін у заняттях, та своєчасного сповіщення учасників. Існуючі загальні рішення не враховують специфіку навчального процесу, що обумовлює актуальність розробки спеціалізованої мобільної системи.

Мета роботи полягає в розробці мобільної системи обліку та управління навчальними заняттями на базі Flutter, що забезпечує планування розкладу, управління заняттями та своєчасне сповіщення вчителів та студентів.

Для досягнення поставленої мети необхідно дослідити предметну область управління навчальними заняттями, виявити предметно-специфічні складнощі та проаналізувати існуючі рішення, сформувати функціональні та нефункціональні вимоги до системи, обрати та обґрунтувати технологічний стек, спроєктувати та реалізувати серверну частину системи на базі Django та REST API з використанням Firebase автентифікації, а також розробити мобільний клієнт на Flutter.

Об'єктом дослідження є процеси обліку та управління навчальними заняттями в системі репетиторства, а предметом дослідження є методи та засоби розробки мобільних систем управління навчальними заняттями.

Розроблена система може бути використана репетиторами, лекторами, вчителями та студентами для ефективного управління навчальним процесом. Реалізовані архітектурні рішення є практичним прикладом побудови мобільних застосунків та можуть бути використані у подібних проєктах.

Основні результати аналізу та дослідження предметної області обговорювались на IX Міжнародній студентській науково-технічній конференції «Природничі та гуманітарні науки. Актуальні питання» Тернопільського національного технічного університету імені Івана Пулюя (м. Тернопіль, 2026 р.).

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ УПРАВЛІННЯ НАВЧАЛЬНИМИ ЗАНЯТТЯМИ

Управління навчальними заняттями є частиною освітнього процесу як у закладах формальної освіти, так і у сфері приватного репетиторства. Предметна область охоплює сукупність процесів, пов'язаних із плануванням, проведенням та обліком навчальних занять, які було винесено в тезах наукової конференції, наведені у додатку А. Таким чином, обраний домен характеризується чіткою рольовою структурою та потребою у координації дій між різними категоріями користувачів.

1.1 Характеристика предметної області

Основним бізнес-процесом у предметній області є організація навчального заняття – від першого контакту між викладачем і студентом до завершення або продовження курсу. Цей процес охоплює кілька логічно пов'язаних етапів, що разом утворюють повний життєвий цикл навчальної взаємодії.

Усе починається з пошуку та домовленості. Студент шукає викладача за предметом і рівнем, вони обговорюють формат, час та умови навчання. Після досягнення згоди обидві сторони переходять до організаційного етапу.

Наступний крок – формування розкладу. Викладач та студент узгоджують зручні дні й час, визначають тривалість курсу та склад групи. Розклад стає спільною основою, навколо якої будується весь подальший процес.

Коли домовленості зафіксовано, починається регулярне проведення занять. Сторони дотримуються встановленого ритму, відстежують прогрес, користуються спільними матеріалами та готуються до кожної наступної зустрічі. З огляду на непередбачувані обставини, що є природною частиною будь-якого тривалого процесу, не виключається хвороба однієї зі сторін, зміна часового поясу, технічні труднощі, тощо. Саме тут важливо мати зрозумілий механізм реагування, щоб ситуація не погіршила процес навчання.

Це реагування відбувається на етапі коригування розкладу, заняття скасовується або переноситься, всі учасники отримують повідомлення, узгоджується новий час. Процес адаптується, але не переривається.

Зрештою, настає завершення або продовження курсу. Повний цикл цієї взаємодії відображено на рисунку 1.1.

Рисунок 1.1 – Життєвий цикл навчального процесу

Наочним прикладом застосування описаного процесу є тижневий розклад репетитора, що наведено на рисунку 1.2.

ЧАС	ПОНЕДІЛОК	ВІВТОРОК	СЕРЕДА	ЧЕТВЕР	П'ЯТНИЦЯ	СУБОТА
09:00 – 10:00	Математика	Англійська мова	Математика	Англійська мова	Математика	Українська мова
10:00 – 11:00	Англійська мова	Математика	Українська мова	Математика	Українська мова	Математика
11:00 – 12:00	Українська мова	Фізика	Англійська мова	Фізика	Англійська мова	Англійська мова
12:00 – 13:00	Фізика	—	Фізика	—	Фізика	Хімія
14:00 – 15:00	Хімія	Українська мова	—	Українська мова	Хімія	Фізика
15:00 – 16:00	—	Хімія	Хімія	Хімія	—	Історія України
16:00 – 17:00	Математика	Англійська мова	Математика	Англійська мова	Математика	—
17:00 – 18:00	Англійська мова	Математика	Історія України	Математика	Англійська мова	—
18:00 – 19:00	Історія України	—	Англійська мова	Історія України	—	—
19:00 – 20:00	—	Історія України	—	—	Історія України	—

УМОВНІ ПОЗНАЧЕННЯ:

Математика · Українська мова · Хімія · Англійська мова · Фізика · Історія України

ПРИМІТКИ:

- Тривалість одного заняття – 60 хвилин.
- Між заняттями передбачено 10-хвилинну перерву.
- Розклад може коригуватися за домовленістю.

Рисунок 1.2 – Приклад розкладу навчальних занять репетитора

1.2 Огляд та аналіз існуючих рішень

Ринок програмного забезпечення пропонує ряд рішень, що частково вирішують задачі управління навчальними заняттями. Серед найпоширеніших, Google Calendar, Calendly та Preply. Кожне з цих рішень орієнтоване на певний сегмент завдань, проте жодне з них не покриває повний спектр потреб приватного репетитора та його студентів.

Google Calendar є універсальним календарним застосунком, що підтримує повторювані події та часові зони. Однак система не передбачає рольової моделі «викладач/студент», управління навчальними групами чи системи запрошень із контролем ємності. Інтерфейс Google Calendar зображено на рисунку 1.3.

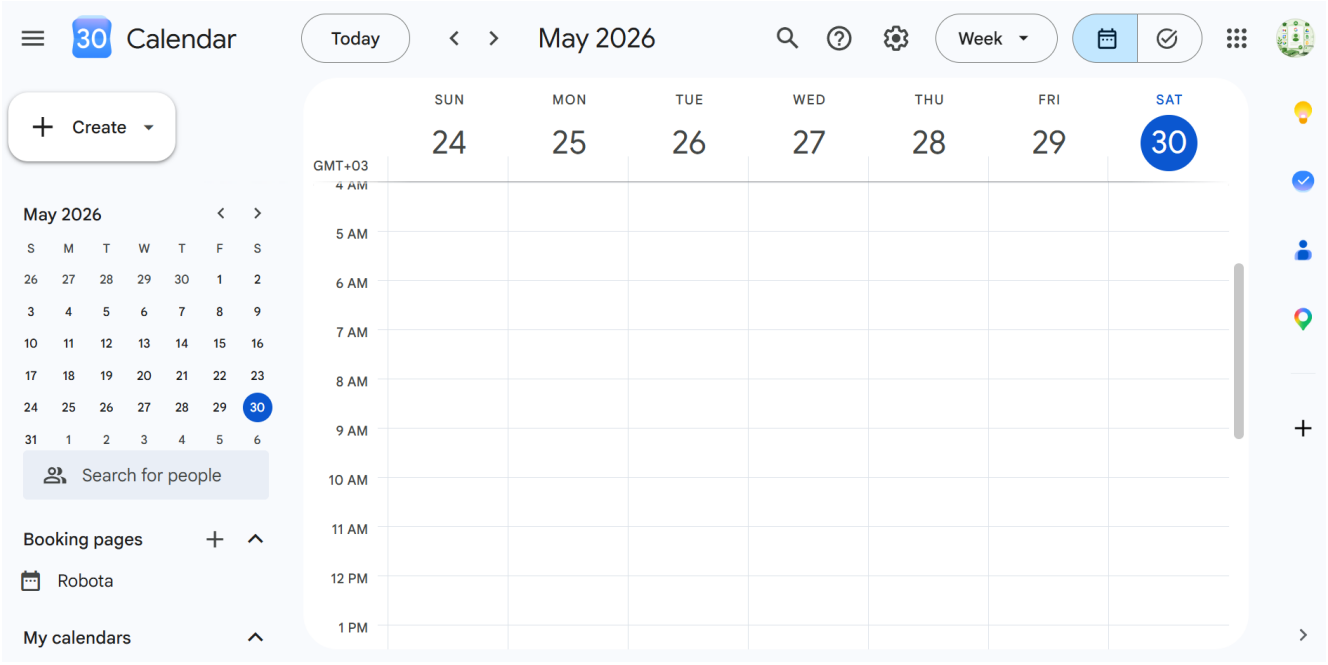

Рисунок 1.3 – Інтерфейс Google Calendar

Calendly – сервіс планування зустрічей, орієнтований переважно на B2B-сегмент. Система дозволяє організатору визначати доступні часові слоти, а запрошеним самостійно обирати зручний час. Проте Calendly не підтримує концепцію постійних навчальних груп та не дає студентам доступу до спільного розкладу. Інтерфейс Calendly зображено на рисунку 1.4.

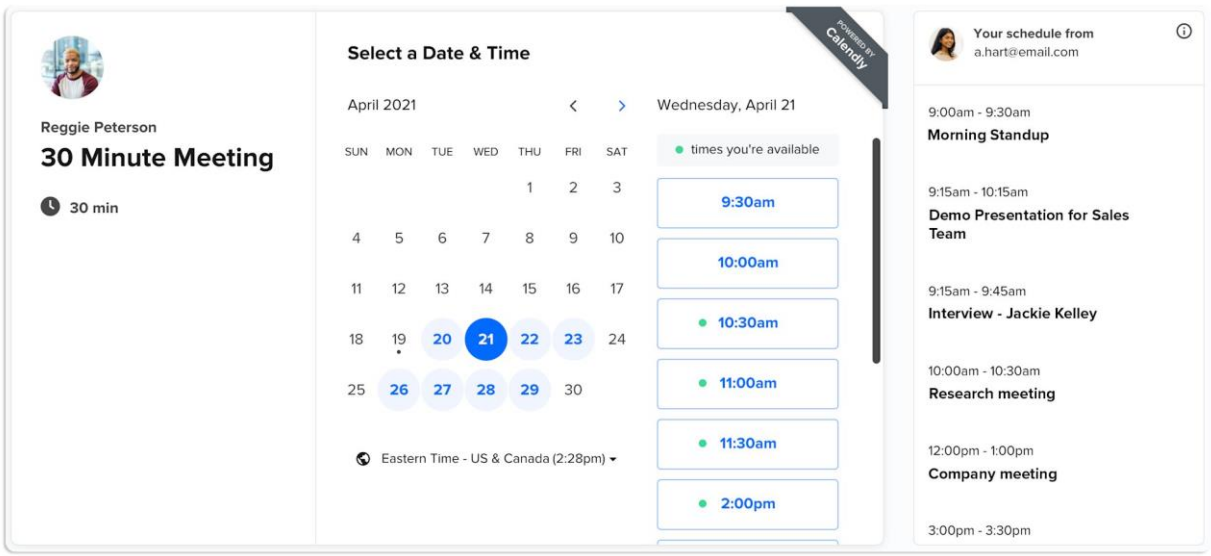

Рисунок 1.4 – Інтерфейс сервісу Calendly

Preply є спеціалізованою платформою для репетиторства, яка підтримує рольову модель «викладач/студент» та управління заняттями. Водночас платформа є закритою екосистемою зі значними комісійними відрахуваннями та обмеженими можливостями кастомізації для стороннього розробника.

Отже, кожна з розглянутих систем має певні обмеження щодо гнучкості, функціональності або умов використання, що не дозволяє повною мірою задовольнити потреби цільової аудиторії. Порівняльний аналіз функціональних можливостей зазначених систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Функціональна вимога	**Google Calendar**	**Calendly**	**Preply**	**TutorGlide**
Повторювані заняття з підтримкою виключень	Частково	Частково	Відсутнє	Підтримується
Управління навчальною групою студентів	Відсутнє	Відсутнє	Підтримується	Підтримується
Рольова модель Викладач / Студент	Відсутнє	Відсутнє	Підтримується	Підтримується
Підтримка різних часових зон	Підтримується	Підтримується	Підтримується	Підтримується
Система запрошень із контролем ємності	Відсутнє	Відсутнє	Відсутнє	Підтримується
Push-сповіщення про зміни розкладу	Частково	Частково	Підтримується	Підтримується

Таким чином, проведений аналіз свідчить про наявність незайнятої ніші для спеціалізованого мобільного застосунку, що поєднує рольову модель доступу, управління навчальними групами, гнучке планування повторюваних занять та систему push-сповіщень у єдиному рішенні.

1.3 Аналіз специфіки та складних випадків предметної області

Поглиблений аналіз предметної області виявляє ряд нетривіальних сценаріїв, коректна обробка яких є критично важливою для надійності та зручності системи. Встановлені сценарії охоплюють управління повторюваними заняттями, часовими зонами, конфліктами у розкладі та динамікою навчальних груп.

Повторювані заняття є базовою концепцією системи. Більшість навчальних занять проводяться у фіксований день тижня та час протягом семестру або навчального року. Система зберігає єдиний шаблон повторення, з якого динамічно генерується розклад для заданого часового діапазону. Порівняно з ним, одноразові заняття представляють собою самостійні події без зв'язку із серією.

Ключовою проблемою є необхідність підтримки виключень – ситуацій, коли конкретний екземпляр повторюваного заняття відрізняється від загального шаблону. Виділяються два основних типи виключень:

– Скасування окремого заняття.

– Перенесення заняття на інший час.

Окрім типу повторення, заняття також розрізняються за своєю тривалістю та складом учасників. Заняття може проводитись індивідуально, між одним викладачем і одним студентом, або у груповому форматі, коли до однієї навчальної кімнати залучено кількох студентів. Обидва формати можуть бути як одноразовими, так і повторюваними, що в кінцевому результаті утворює чотири можливі комбінації на практиці.

Отже, механізм виключень дозволяє зберігати цілісність шаблону повторення, не порушуючи загальної структури розкладу. Типи занять та механізм виключень наведено на рисунку 1.5.

Рисунок 1.5 – Типи навчальних занять та механізм виключень у розкладі

У сучасних умовах репетитор і студенти можуть знаходитись в різних часових зонах. Некоректна обробка часових особливостей призводить до плутанини у розкладі та пропущених занять. Система зберігає всі часові мітки у форматі UTC, а відображення часу здійснюється у локальній часовій зоні кожного користувача відповідно до місця перебування його смартфону.

Особливу складність становлять крос-опівнічні заняття – заняття, що починаються в один календарний день і закінчуються в інший. У UTC поданні такі заняття перетинають межу доби, що вимагає спеціальної обробки під час генерації розкладу та перевірки конфліктів.

Конфлікт у розкладі виникає, коли два або більше занять перетинаються у часі для одного учасника. Система застосовує асиметричну логіку обробки конфліктів залежно від ролі користувача.

Для викладача конфлікт є жорстким обмеженням, тобто система не дозволяє створити нове заняття, що перетинається з існуючим у розкладі викладача. Це

обумовлено тим, що викладач веде заняття особисто і фізично не може бути присутнім у двох місцях одночасно. В такий спосіб, викладач не зможе випадково встановити кілька занять на одну і ту саму годину, в результаті виключається будь-яка можливість перетину занять.

Для студента конфлікт є м'яким попередженням, система повідомляє студента про перетин занять, але не забороняє участь у них. Це рішення відображає реальну практику, бо студент може записатися на заняття в різних групах, що частково перетинаються, та самостійно вирішувати питання відвідування. Логіку виявлення конфліктів наведено на рисунку 1.6.

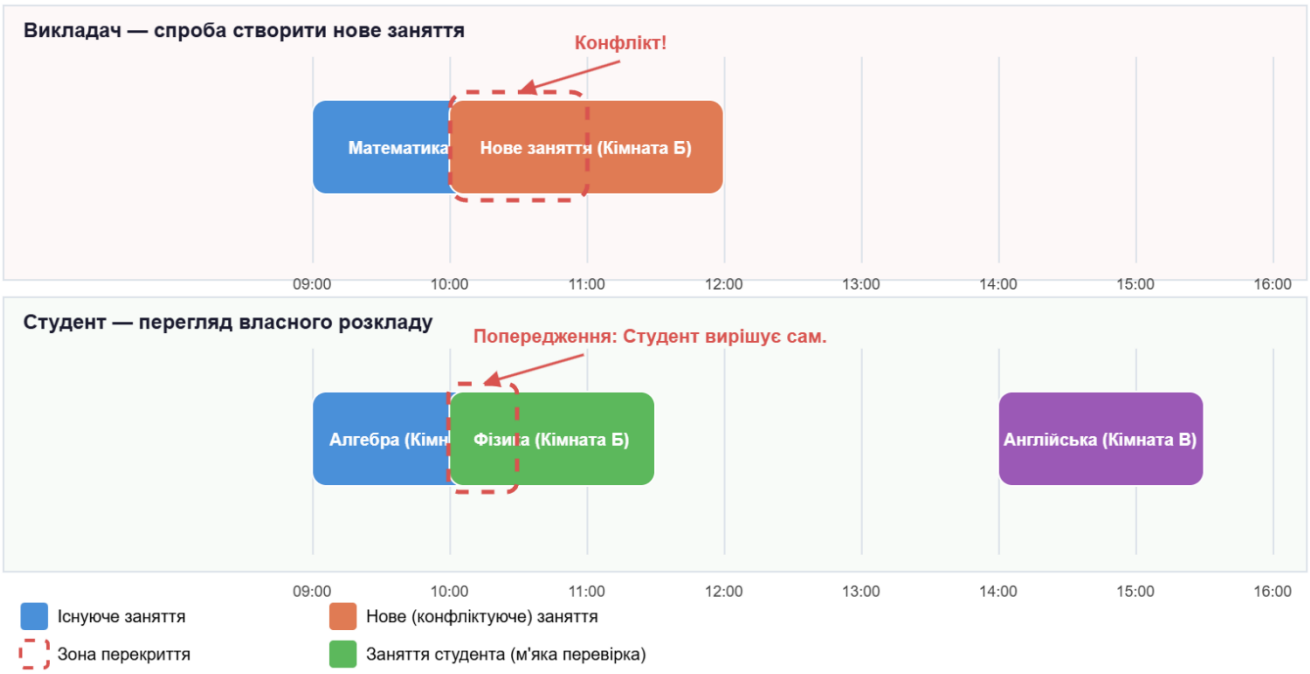

Рисунок 1.6 – Логіка виявлення конфліктів у розкладі

Кожна навчальна кімната має обмежену ємність – максимальну кількість студентів-учасників. Управління ємністю тісно пов'язане зі станом запрошень та поточним складом групи. Подібні сценарії потребують атомарної перевірки ємності при прийнятті запрошення. При виході студента з кімнати або відкликанні запрошення ємність звільняється, що дає змогу надіслати нові запрошення іншим потенційним учасникам. Отже, управління ємністю також є важливим аспектом цілісності даних системи.

1.4 Формування вимог до системи

На основі проведеного аналізу предметної області та виявлених проблемних аспектів, сформовано комплекс вимог до розроблюваної системи. Вимоги охоплюють цільові платформи, функціональні та нефункціональні характеристики, а також визначення акторів і сценаріїв використання [1].

Система розробляється як мобільний застосунок з підтримкою обох домінуючих мобільних платформ – Android та iOS. Вибір крос-платформного підходу зумовлений необхідністю охоплення максимальної аудиторії без подвоєння витрат на розробку та підтримку двох незалежних кодових баз.

Серверна частина системи розгортається у хмарному середовищі та надає REST API для взаємодії з мобільним клієнтом. Хмарна автентифікація забезпечується платформою Firebase Authentication, що підтримує авторизацію через електронну пошту. Push-сповіщення доставляються через Firebase Cloud Messaging, що є перевіреним рішенням для обох мобільних платформ.

1.4.1 Функціональні вимоги

Функціональні вимоги визначають конкретні функції, які система надає користувачам. Вимоги згруповано за функціональними блоками.

Блок автентифікації та управління профілем включає такі вимоги:

– Реєстрація нового користувача з підтвердженням електронної пошти.
– Вхід в систему за обліковими даними.
– Скидання пароля через електронну пошту.

Блок управління кімнатами передбачає створення навчальної, перегляд списку доступних кімнат, а також перегляд деталей кімнати та складу учасників.

Блок управління запрошеннями включає надсилання запрошень студентам за електронною адресою, перегляд вхідних запрошень, прийняття або відхилення запрошення, відкликання надісланого запрошення, а також вихід студента з кімнати.

Блок управління заняттями та розкладом передбачає створення одноразових та повторюваних занять, налаштування параметрів повторення (день тижня, час, дата завершення серії), перегляд розкладу занять кімнати, скасування або перенесення конкретного екземпляра заняття, а також автоматичне виявлення конфліктів у розкладі.

Блок сповіщень охоплює автоматичне надсилання push-сповіщень про нові запрошення, сповіщення про зміни в розкладі (скасування, перенесення) та сповіщення про вступ нового учасника до кімнати.

1.4.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що не пов'язані безпосередньо з функціями, але суттєво впливають на її придатність до використання.

Вимоги до безпеки, зокрема автентифікація користувачів здійснюється на основі Firebase ID Token із перевіркою на сервері, доступ до ресурсів захищено на рівні API відповідно до ролі користувача, передача даних між клієнтом і сервером здійснюється виключно через захищений протокол HTTPS.

Вимоги до продуктивності включають час відповіді API на стандартні запити до 500 мс, генерація розкладу для місячного діапазону виконується на сервері та не блокує інтерфейс користувача.

Вимоги до масштабованості мають враховувати, що архітектура серверної частини дозволяє горизонтальне масштабування без змін у кодовій базі, система підтримує одночасну роботу 1000 активних користувачів без вагомого зниження продуктивності роботи системи.

Вимоги до UX декларують, що інтерфейс відповідає рекомендаціям Material Design для Android та Human Interface Guidelines для iOS, навігація між основними розділами застосунку виконується не більше ніж за два кроки, а застосунок підтримує темну та світлу теми оформлення.

1.5 Актори системи та опис сценаріїв використання

Система передбачає двох основних акторів, викладач та студент. Обидва актори мають спільний набір базових функцій, пов'язаних з автентифікацією та переглядом інформації, а також специфічні функції відповідно до своєї ролі. Діаграму варіантів використання наведено на рисунку 1.7.

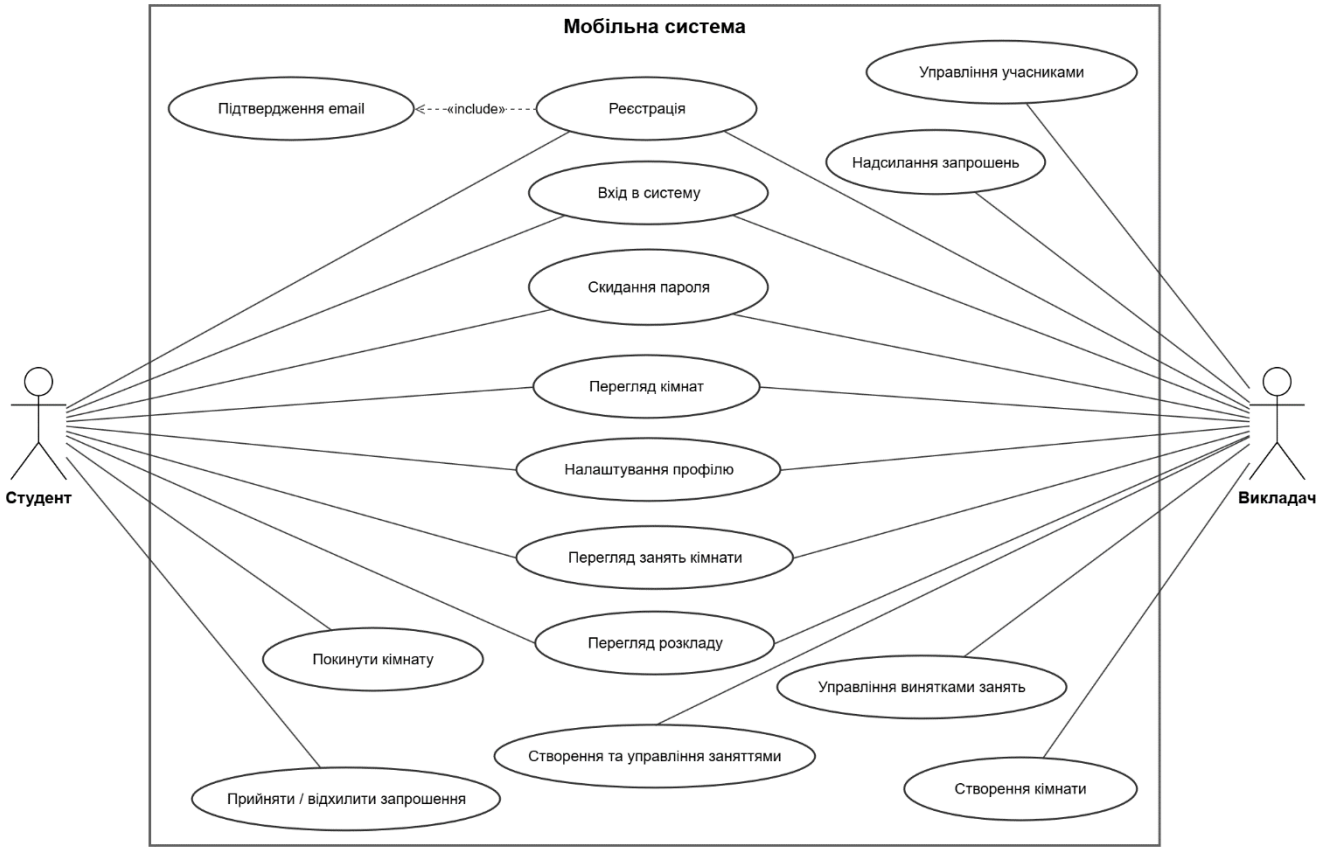

Рисунок 1.7 – Діаграма варіантів використання системи TutorGlide

Користувач створює обліковий запис у системі, при цьому автоматично включається підтвердження email. Функція створення та управління заняттями є центральною для навчального процесу, і забезпечує основну взаємодію між викладачем і студентами. Обидва актори можуть переглядати доступні навчальні кімнати та їх вміст. Цей варіант використання забезпечує навігацію по системі та є відправною точкою для подальших дій, таких як перегляд занять або розкладу. Деталізований опис варіантів використання системи наведено у таблиці 1.2.

Таблиця 1.2 – Варіанти використання системи TutorGlide

№	Актор	Найменування	Опис
1	Обидва	Реєстрація	Створення нового облікового запису
2	Обидва	Вхід в систему	Автентифікація за email та паролем через Firebase
3	Обидва	Скидання пароля	Відновлення доступу через лист на email
4	Обидва	Налаштування профілю	Редагування імені та прізвища
5	Обидва	Перегляд кімнат	Перегляд списку навчальних кімнат
6	Обидва	Перегляд занять кімнати	Перегляд списку занять обраної навчальної кімнати
7	Обидва	Перегляд розкладу	Відображення розкладу занять
8	Студент	Прийняти / відхилити запрошення	Реакція студента на запрошення до кімнати
9	Студент	Покинути кімнату	Вихід студента зі складу учасників кімнати
10	Викладач	Створення кімнати	Формування нової навчальної кімнати
11	Викладач	Управління учасниками	Перегляд та видалення учасників кімнати
12	Викладач	Надсилання запрошень	Запрошення студентів до кімнати
13	Викладач	Створення та управління заняттями	Додавання, редагування та видалення занять
14	Викладач	Управління винятками занять	Скасування або перенесення окремих занять

1.6 Порівняльний аналіз підходів до розробки мобільних систем

Вибір технологічного стеку є одним із ключових архітектурних рішень при розробці мобільної системи. Від обраних технологій залежать продуктивність застосунку, якість користувацького досвіду, складність підтримки та можливості масштабування. У цьому підрозділі проводиться порівняльний аналіз основних підходів та технологій.

Flutter – фреймворк від Google для розробки крос-платформних застосунків із єдиної кодової бази мовою Dart. Ключовою особливістю Flutter є власний рушій рендерингу (Skia/Impeller), що відмальовує кожен піксель інтерфейсу незалежно від нативних компонентів операційної системи. За рахунок цього, зовнішній вигляд

на обох платформах ідентичний, а продуктивність анімацій залишається на рівні 60-120 кадрів на секунду.

React Native – фреймворк від Meta, що використовує JavaScript/TypeScript для опису інтерфейсу, але відмальовує нативні компоненти кожної платформи. Перевагою є можливість повторного використання коду розробниками, знайомими з React. Основним недоліком є міст між JavaScript та нативним кодом, що може спричиняти затримки при інтенсивних операціях з інтерфейсом.

.NET MAUI (Multi-platform App UI) – фреймворк від Microsoft на основі C#, наступник Xamarin.Forms. MAUI надає доступ до нативних API через єдиний API-шар і підтримує Android, iOS, macOS та Windows. Фреймворк орієнтований на розробників екосистеми Microsoft.

Flutter вирізняється найвищою продуктивністю рендерингу, розвиненою екосистемою пакетів та активним співтовариством розробників. Саме тому Flutter обрано як основний фреймворк для розробки мобільного клієнта системи TutorGlide [2].

Серверна частина системи реалізує REST API для взаємодії з мобільним клієнтом. Серед популярних фреймворків для побудови REST API на мові Python виділяються Django REST Framework, FastAPI та Flask.

Django REST Framework (DRF) – розширення фреймворку Django, що надає потужні інструменти для побудови REST API, зокрема серіалізатори, viewsets, permissions та автоматичну генерацію документації. Django в цілому пропонує розвинену ORM, систему міграцій та вбудовану адмін-панель, що суттєво прискорює розробку.

FastAPI – сучасний асинхронний фреймворк з автоматичною валідацією на основі Pydantic та генерацією OpenAPI-специфікації. FastAPI показує вищу продуктивність за рахунок асинхронної обробки запитів, однак має менш зрілу екосистему ORM та адміністративних інструментів порівняно з Django.

Для реалізації TutorGlide обрано Django REST Framework, оскільки складність доменної моделі та потреба у надійній ORM з підтримкою транзакцій роблять Django оптимальним вибором для даного проєкту [3, 4].

Firebase – платформа від Google, що надає набір хмарних сервісів для мобільних та веб-застосунків. У контексті розробки TutorGlide використовуються два ключові сервіси, Firebase Authentication та Firebase Cloud Messaging [5].

Firebase Authentication забезпечує надійну та безпечну автентифікацію користувачів без необхідності реалізації власної системи управління ідентифікаторами. Сервіс підтримує реєстрацію та вхід за email/паролем, підтвердження електронної пошти, скидання пароля та видачу JWT-токенів (Firebase ID Token) для авторизації запитів до REST API.

Firebase Cloud Messaging (FCM) є стандартним сервісом для доставки push-сповіщень на Android та iOS пристрої. FCM гарантує доставку повідомлень навіть при закритому застосунку та підтримує як data-повідомлення (обробляються кодом застосунку), так і notification-повідомлення (відображаються системою автоматично). Таким чином, Firebase надає надійну інфраструктуру автентифікації та сповіщень без необхідності розгортання власних сервісів [6].

За результатами проведеного аналізу для розробки системи TutorGlide обрано наступний технологічний стек:

- Flutter (Dart) для мобільного клієнта.
- Django REST Framework (Python) для серверної частини.
- PostgreSQL як основна реляційна база даних.
- Firebase Authentication для автентифікації користувачів.
- Firebase Cloud Messaging для push-сповіщень.

Flutter забезпечує єдину кодову базу для Android та iOS з нативною продуктивністю та повною бібліотекою готових компонентів. Django REST Framework надає зрілу ORM, потужні засоби валідації та вбудовані механізми безпеки. PostgreSQL забезпечує транзакційну цілісність для операцій із ємністю кімнат та розкладом. Firebase вирішує завдання автентифікації та сповіщень на рівні перевіреної хмарної інфраструктури. Отже, обраний стек оптимально відповідає вимогам системи за критеріями продуктивності, надійності та швидкості розробки.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ОБЛІКУ НАВЧАЛЬНИХ ЗАНЯТЬ

Наведено повний технічний опис проєктування та реалізації мобільної системи для управління навчальними заняттями TutorGlide. Розглянуто загальну архітектуру системи, архітектурні рішення для серверної частини на базі Django REST Framework та мобільного клієнта на Flutter, а також підсистеми розкладу занять і push-сповіщень.

Кожна підсистема описана з урахуванням конкретних технічних рішень, що забезпечують надійність, розширюваність та зручність подальшого супроводу коду. Для ілюстрації архітектурних рішень та бізнес-процесів використано набір UML-діаграм, компонентні, класів, послідовності, діяльності та станів.

2.1 Загальна архітектура системи

Система TutorGlide побудована за клієнт-серверною архітектурою з чітко розподіленими відповідальностями між мобільним клієнтом та серверною частиною. Серверна частина реалізована у вигляді REST API, що обробляє бізнес-логіку та керує даними, тоді як мобільний клієнт зосереджений на відображенні інформації та взаємодії з користувачем.

Інтеграція зовнішньої платформи Firebase забезпечує автентифікацію користувачів та доставку push-сповіщень, делегуючи ці функції перевіреній хмарній інфраструктурі. Таким чином, загальна архітектура системи складається з чотирьох основних компонентів, зокрема мобільного клієнта, серверного API, реляційної бази даних та зовнішньої платформи Firebase.

Для побудови системи TutorGlide було обрано клієнт-серверну архітектуру, яка є загальноприйнятим підходом до розробки мобільних застосунків з централізованим управлінням даними [7]. У рамках цієї архітектури мобільний клієнт, реалізований на Flutter, виконує роль тонкого клієнта, що взаємодіє із серверною частиною через мережу, не зберігаючи власної копії бізнес-логіки.

Серверна частина, побудована на базі Django REST Framework, є єдиним джерелом бізнес-логіки та стану системи. Взаємодія між клієнтом та сервером відбувається через REST API. Усі запити та відповіді передаються у форматі JSON. API організовано за принципом версіонування з префіксом /api/v1/ для внесення зворотньо несумісних змін без порушення роботи застарілих клієнтів [4].

Для ефективної роботи зі списками даних у REST API застосовано курсорну пагінацію, яка є більш продуктивною порівняно зі стандартною пагінацією на основі зсуву при роботі з великими обсягами даних. Відповідь сервера на запити зі списками має уніфіковану структуру. Поле cursor містить токен для отримання наступної сторінки результатів, а поле data – масив об'єктів. У разі виникнення помилки сервер повертає об'єкт з полями error.code та error.message для того, щоб однозначно ідентифікувати тип помилки та відображати відповідне повідомлення користувачу. Повний перелік компонентів системи, що взаємодіють у межах клієнт-серверної архітектури TutorGlide, наведено в таблиці 2.1.

Таблиця 2.1 – Компоненти системи TutorGlide та їх відповідальність

Компонент	**Технологія**	**Відповідальність**
Мобільний клієнт	Flutter 3.x / Dart	Інтерфейс користувача, управління станом (BLoC), валідація форм, навігація, відображення розкладу
Серверна частина (REST API)	Django 5.2 / DRF 3.16	Бізнес-логіка, авторизація та автентифікація, генерація розкладу, доступ до даних
База даних	PostgreSQL 16	Зберігання та цілісність даних, транзакційність, реляційні обмеження
Автентифікація	Firebase Authentication	Реєстрація та вхід, верифікація email, генерація та верифікація Firebase ID Token
Push-сповіщення	Firebase Cloud Messaging	Доставка push-сповіщень на мобільні пристрої Android та iOS

Архітектура системи охоплює п'ять ключових компонентів, кожен з яких виконує чітко визначену роль та взаємодіє з іншими через стандартизовані інтерфейси. Обраний тип розподілу відповідальності має на меті встановити незалежність кожного компонента та впровадити можливість їх окремого тестування і масштабування.

Архітектуру системи TutorGlide зображено у вигляді компонентної діаграми, що відображає чотири основні компоненти та їх взаємозв'язки. Мобільний клієнт TutorGlide на Flutter взаємодіє із сервером бізнес-логіки Django REST API шляхом надсилання REST-запитів із заголовком авторизації Bearer, отримуючи у відповідь JSON-дані. Сервер, у свою чергу, звертається до бази даних PostgreSQL.

Окремим компонентом виступає платформа Firebase. Django-сервер звертається до Firebase Admin SDK для верифікації отриманих від клієнта токенів, а також для відправки FCM push-сповіщень на мобільні пристрої користувачів [6]. Візуалізація взаємодії компонентів системи TutorGlide зображено на рисунку 2.1.

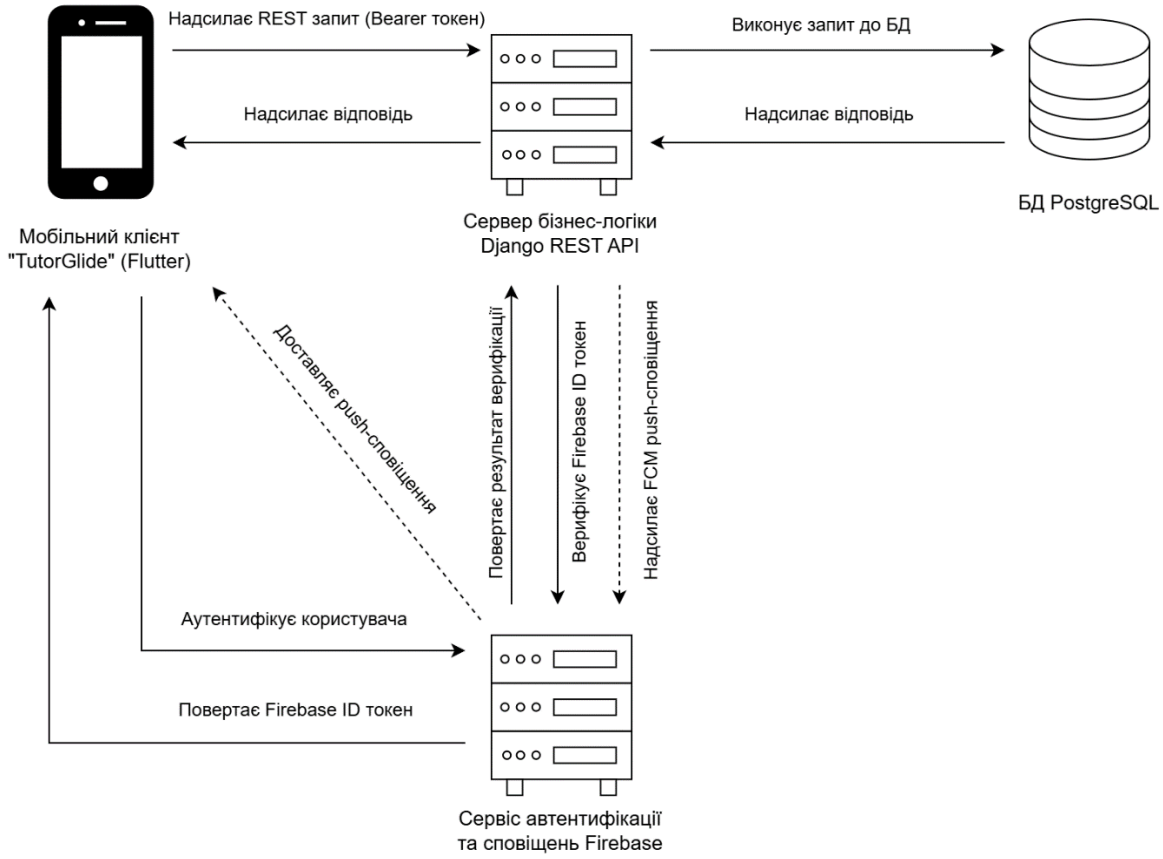

Рисунок 2.1 – Візуалізація взаємодії компонентів мобільної системи

Наведена компонентна діаграма унаочнює чітку розмежованість обов'язків між компонентами. Мобільний клієнт не має прямого доступу до бази даних, а всі операції з даними виконуються виключно через серверний REST API в цілях безпеки та цілісності даних системи.

2.2 Архітектура та реалізація серверної частини

Серверна частина системи TutorGlide реалізована на базі Django 5.2 та Django REST Framework 3.16.1 – надійних та широко використовуваних технологій для розробки REST API на мові Python. Вибір саме такого стеку технологій обґрунтовано в попередньому розділі. У цьому підрозділі розглянуто конкретні архітектурні рішення та механізми реалізації компонентів серверної частини.

2.2.1 Модель даних та ER-діаграма

Модель даних системи TutorGlide спроєктовано на основі сутностей, виявлених у процесі аналізу предметної області. Для зберігання даних використовується реляційна СУБД PostgreSQL, а доступ до неї здійснюється через Django ORM – об'єктно-реляційний маппер, що дозволяє описувати схему бази даних у вигляді Python-класів і виконувати запити без написання SQL вручну.

Базовою абстракцією для більшості моделей є абстрактний клас BaseModel, що успадковує від UUIDModel та TimeStampedModel. Клас UUIDModel забезпечує автоматичну генерацію UUID як первинного ключа замість автоінкрементного цілого числа, що є поширеною практикою для API-орієнтованих систем, де ідентифікатори ресурсів не мають бути передбачуваними. Клас TimeStampedModel автоматично заповнює поля created_at та updated_at відповідними часовими мітками при кожному збереженні об'єкта.

Центральною сутністю системи є User – модель користувача, що розширює стандартний AbstractUser з Django. Поле email є унікальним ідентифікатором користувача, а поле firebase_uid зберігає унікальний ідентифікатор Firebase для зіставлення акаунтів. Поле role визначає роль користувача у системі та може приймати одне з чотирьох значень, зокрема pending, student, teacher або admin.

Навчальна кімната (Room) є агрегуючою сутністю, що об'єднує викладача та студентів. Поле timezone зберігає часовий пояс кімнати у форматі бази даних часових поясів (наприклад, Europe/Kyiv).

Модель Lesson підтримує два типи занять у межах однієї таблиці бази даних, регулярні (regular) та одноразові (one_time). Регулярні заняття задаються полями recurrence_day, recurrence_start_time, recurrence_end_time. Модель LessonException дозволяє вносити виключення до регулярного розкладу, скасувати (exception_type = cancelled) або перенести на інший час (exception_type = time_changed) конкретне заняття. ER-діаграму бази даних системи TutorGlide зображено на рисунку 2.2.

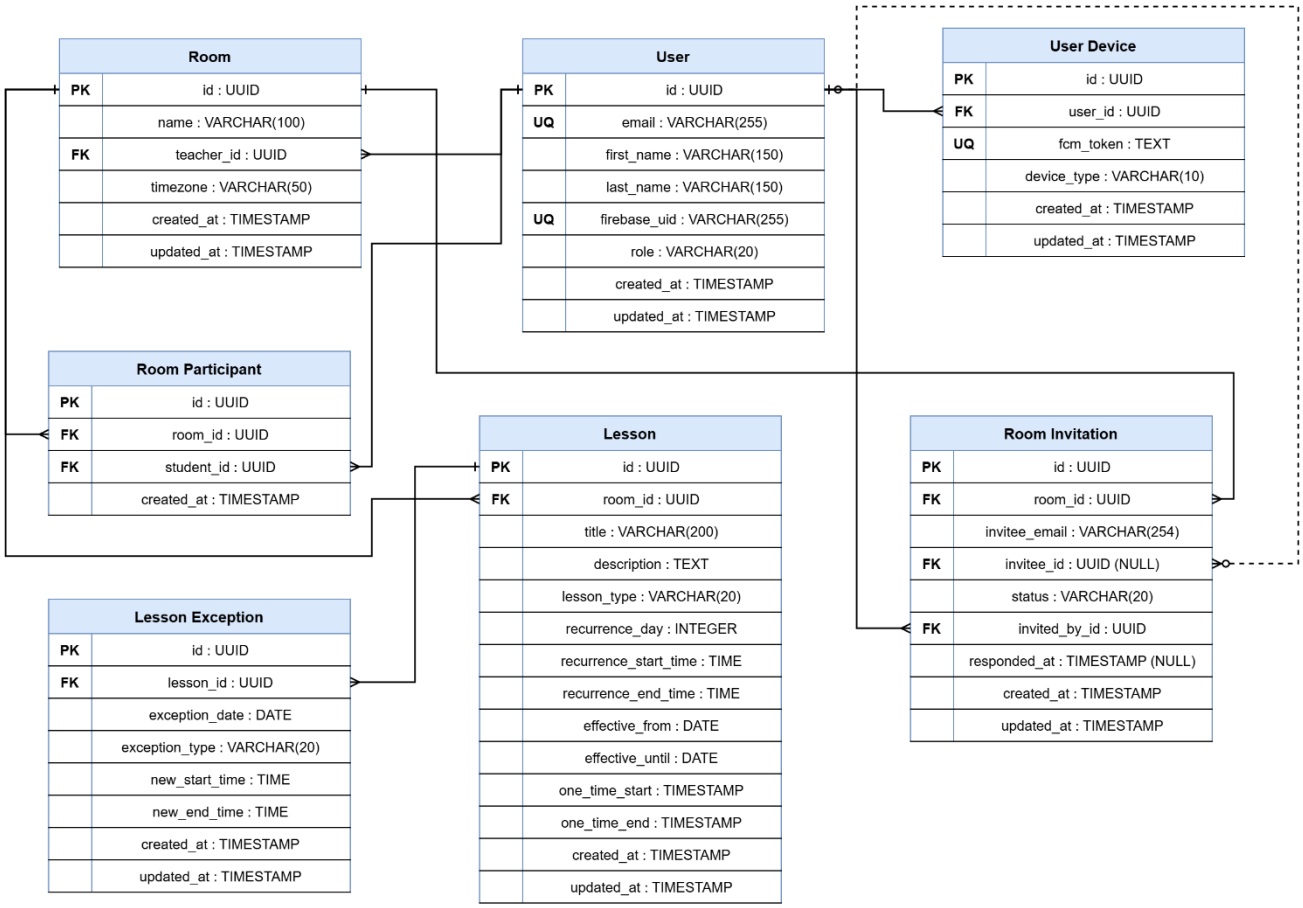

Рисунок 2.2 – ER-діаграма бази даних системи TutorGlide

2.2.2 Сервісний шар та шар вибірки даних

Серверна частина TutorGlide дотримується архітектурного патерну Service / Selector, що розподіляє бізнес-логіку на два чітко відокремлені шари. Такий підхід забезпечує принцип єдиної відповідальності та спрощує модульне тестування компонентів серверної частини.

Сервісний шар (Service Layer) реалізується у файлах services.py кожного застосунку і містить функції, що виконують операції запису, створення, оновлення та видалення об'єктів. Назви сервісних функцій будуються за принципом "дієслово + іменник": create_lesson(), accept_invitation(), eject_student_from_room(). Усі сервісні функції є атомарними операціями та, у разі необхідності, обертаються декоратором @transaction.atomic для гарантування цілісності даних. Сервіси не звертаються до бази даних напряму, вони делегують операції читання селекторам, а записи виконують через Django ORM.

Шар вибірки даних (Selector Layer) реалізується у файлах selectors.py і містить функції для читання та вибірки даних. Назви функцій будуються за принципом "іменник + умова": get_teacher_rooms(), get_room_with_participants(), get_lesson_exception_for_date(). Функції-селектори активно використовують методи select_related() та prefetch_related() для мінімізації кількості SQL-запитів до бази даних. Додатково застосовуються annotate() для обчислення агрегованих значень безпосередньо в запиті. В якості прикладу взаємодії, наведено діаграму класів підсистеми запрошень, що ілюструє взаємодію між компонентами сервісного шару, шару вибірки та представлень, зображено на рисунку 2.3.

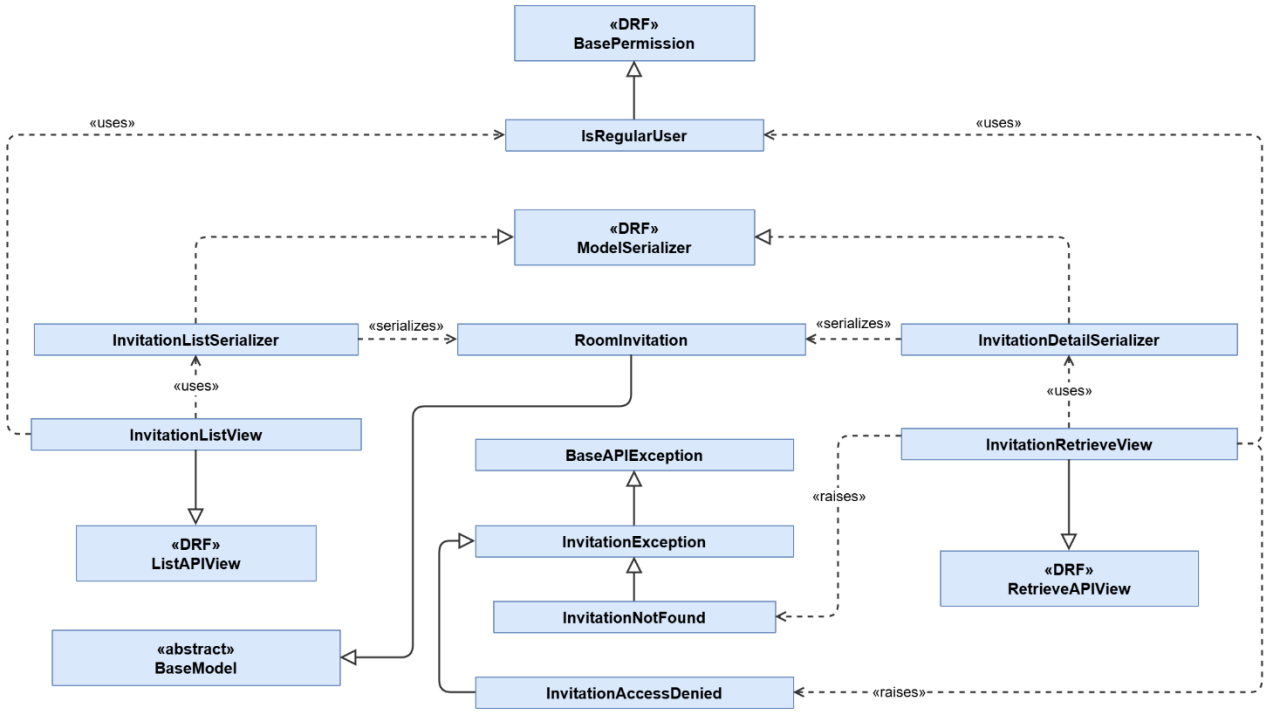

Рисунок 2.3 – Діаграма класів підсистеми запрошень TutorGlide

Наведена діаграма демонструє, що клас InvitationRetrieveView отримує запит і делегує отримання об'єкта функції-селектору, після чого передає дані у серіалізатор для формування відповіді. Запити на прийняття або відхилення запрошення обробляються сервісними функціями. Отже, розподіл логіки між представленням, сервісом та селектором забезпечує тестованість кожного компонента в ізоляції без необхідності звертатись до реальної бази даних.

2.2.3 Проєктування REST API ендпоінтів та OpenAPI-специфікації

REST API системи TutorGlide організовано навколо чотирьох основних ресурсів, доступних за базовим шляхом /api/v1/. Усі ендпоінти захищені автентифікацією на основі Firebase ID Token, що гарантує доступ виключно верифікованим користувачам системи.

Ресурс users надає ендпоінти для роботи з профілем, POST /api/v1/users/profile/ для початкового налаштування профілю та вибору ролі, GET та PATCH /api/v1/users/profile/ для отримання та часткового оновлення профілю.

Ресурс rooms включає повний набір операцій CRUD над навчальними кімнатами, управління учасниками (GET /api/v1/rooms/{id}/participants/, DELETE /api/v1/rooms/{id}/participants/{student_id}/), відправлення масових запрошень (POST /api/v1/rooms/{id}/invitations/) та вихід студента з кімнати (POST /api/v1/rooms/{id}/leave/). Ресурс lessons охоплює управління заняттями (GET/POST /api/v1/rooms/{id}/lessons/, GET/PUT/DELETE /api/v1/lessons/{id}/), виключеннями та отримання персоналізованого розкладу занять (GET /api/v1/lessons/schedule/).

Для автоматичної генерації OpenAPI-специфікації API використовується бібліотека drf-spectacular, яка аналізує код представлень та серіалізаторів і формує повний опис API у стандарті OpenAPI 3.0 [8]. На основі цієї специфікації в режимі розробки доступний Swagger UI – інтерактивний інтерфейс для перегляду та ручного тестування усіх ендпоінтів API. Інтерфейс Swagger UI системи TutorGlide зображено на рисунку 2.4.

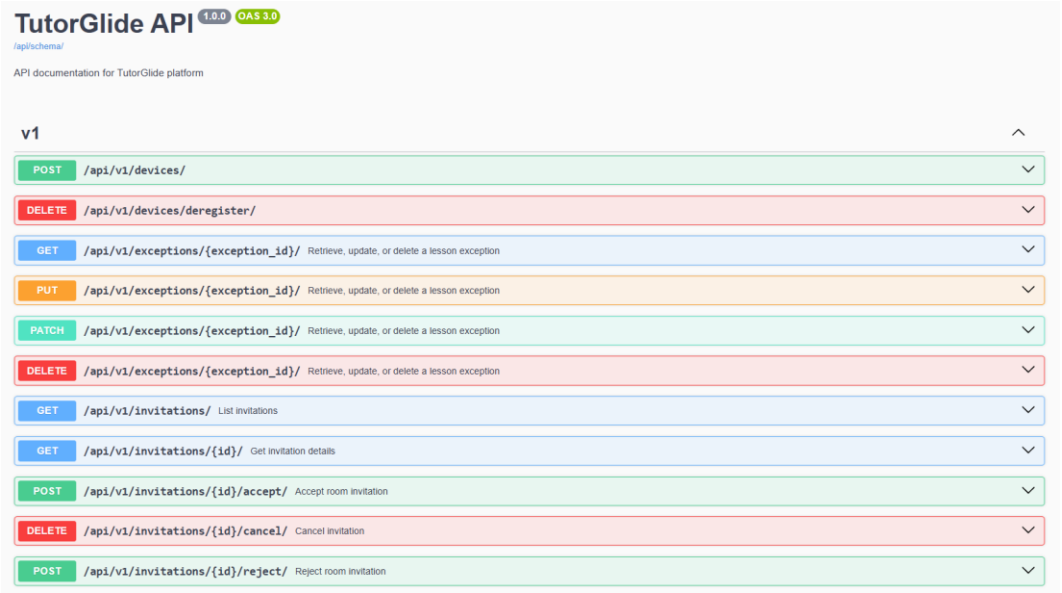

Рисунок 2.4 – Інтерфейс OpenAPI-документації системи

Для аналізу доцільності обраних підходів, було змодельовано діаграму послідовності для процесу прийняття запрошення студентом. Діаграма відображає, перевірку автентифікації, верифікацію статусу запрошення та відправку push-сповіщення викладачу. Описана діаграма послідовності наведена на рисунку 2.5.

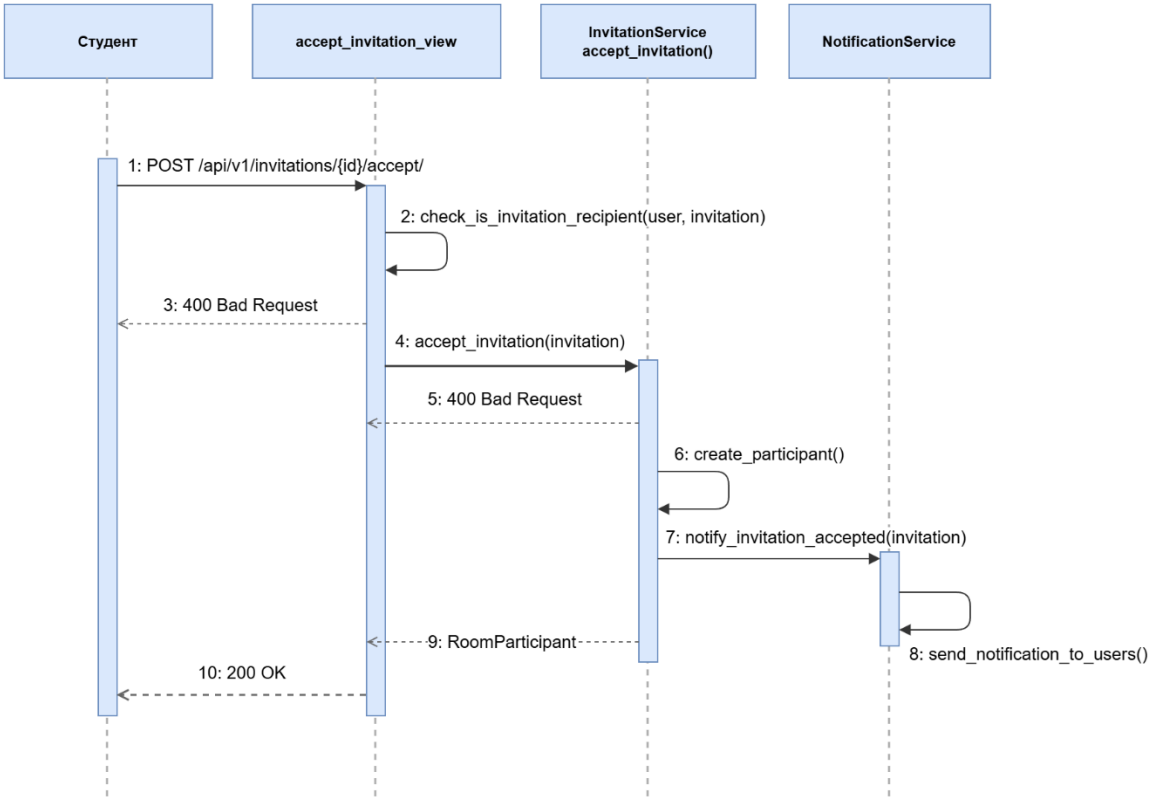

Рисунок 2.5 – Діаграма послідовності прийняття запрошення студентом

Для формального опису можливих станів запрошення та переходів між ними було розроблено діаграму станів. Запрошення створюється у стані Pending (очікує відповіді студента). Зі стану Pending запрошення може перейти у стан Accepted (студент прийняв), Rejected (студент відхилив) або Cancelled (викладач скасував). При прийнятті запрошення відбувається автоматичний перехід. Студент додається до учасників кімнати з перевіркою максимальної ємності. Візуалізація роботи підсистеми запрошень зображено на рисунку 2.6.

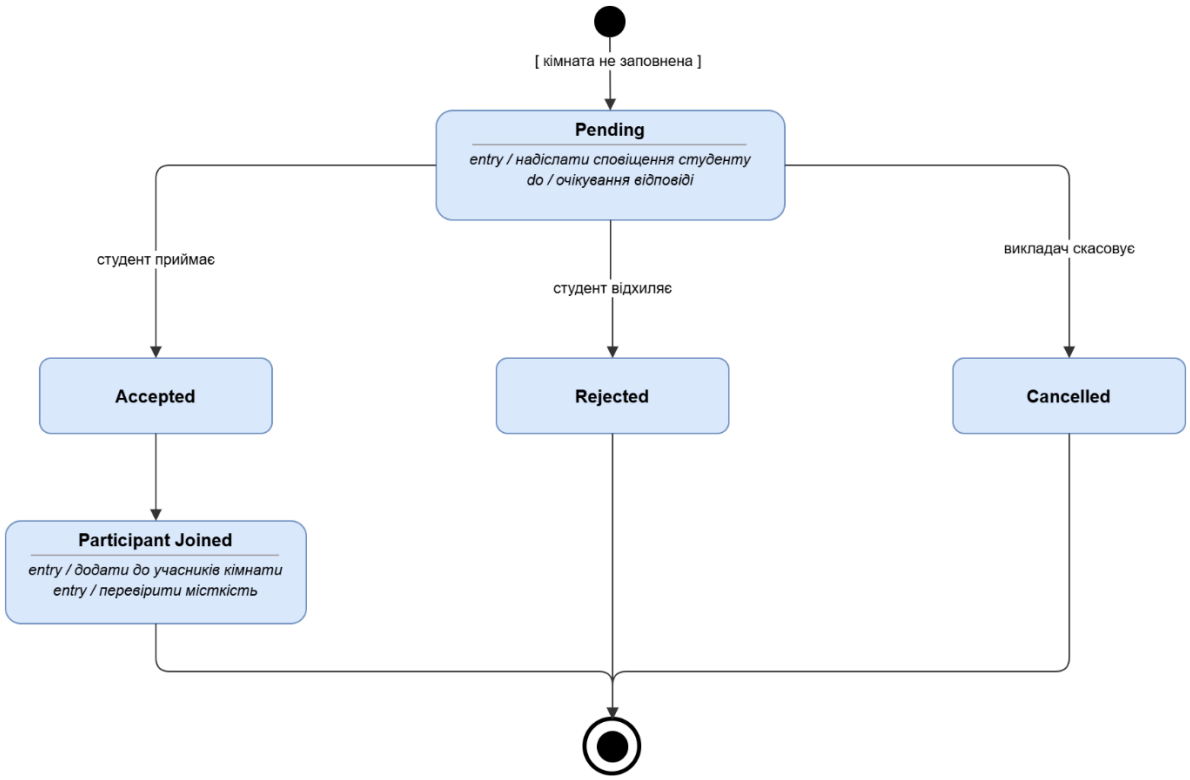

Рисунок 2.6 – Візуалізація робити підсистеми запрошень

2.3 Архітектура та реалізація мобільного клієнта на Flutter

Мобільний клієнт системи TutorGlide розроблено на платформі Flutter з використанням мови програмування Dart. Вибір Flutter обґрунтовано в розділі 1; у цьому підрозділі описано конкретні архітектурні рішення, що були застосовані при побудові мобільного застосунку: Clean Architecture для організації коду, патерн BLoC для управління станом, GetIt для ін'єкції залежностей та GoRouter для організації навігації.

2.3.1 Опис шарів та їх відповідальність

Мобільний клієнт системи TutorGlide побудовано відповідно до принципів Clean Architecture – архітектурного підходу, що забезпечує незалежність бізнес-логіки від деталей реалізації, фреймворків, UI та джерел даних [9, 10]. Ключовим правилом Clean Architecture є правило залежностей. Внутрішні шари не мають залежностей від зовнішніх, тобто бізнес-логіка не знає про конкретні реалізації репозиторіїв, а репозиторії не залежать від деталей HTTP-клієнта. Загальну концепцію Clean Architecture зображено на рисунку 2.7.

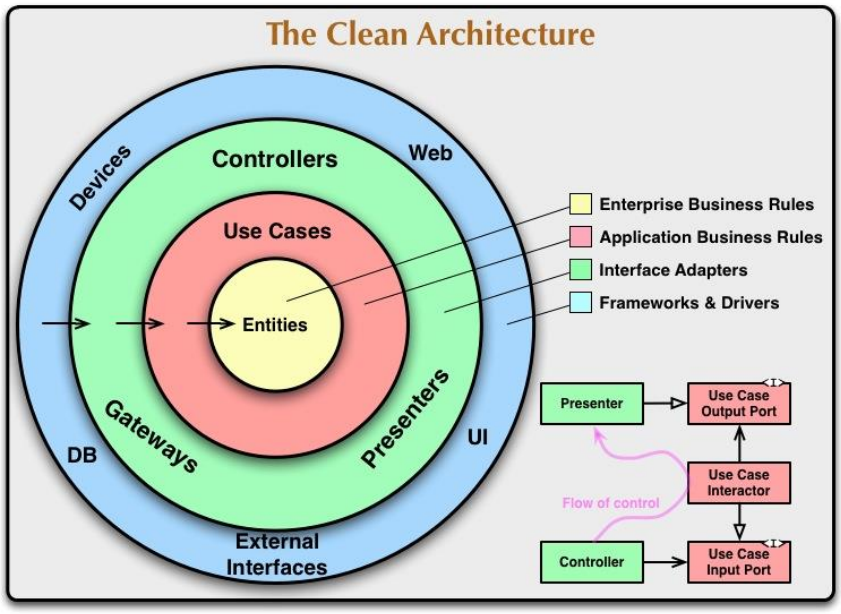

Рисунок 2.7 – Концепція Clean Architecture та її шари [10]

Отже, керуючись принципами чистої архітектури, клієнтську частину системи структуровано у чотири основні шари:

- Domain Layer.
- Data Layer.
- Presentation Layer.
- Core.

Перелік шарів Clean Architecture у мобільному клієнті TutorGlide та їх відповідальність наведено в таблиці 2.2.

Таблиця 2.2 – Шари Clean Architecture у мобільному клієнті TutorGlide

Шар	**Каталог**	**Відповідальність**
Domain	lib/src/domain/	Сутності (entities), інтерфейси репозиторіїв, Use Cases – чисті Dart-класи без зовнішніх залежностей
Data	lib/src/data/	Реалізація репозиторіїв, DTO-моделі (Freezed), Remote Data Sources (HTTP-запити через Dio), маппери toEntity()
Presentation	lib/src/presentation/	BLoC-компоненти (події, стани, логіка), UI-сторінки та форми (Pages, Forms), BlocBuilder/BlocListener
Core	lib/src/core/	DioClient та інтерцептори, спільні UI-компоненти, тема застосунку, локалізація, валідація (Formz)
Config	lib/src/config/	Конфігурація Firebase, GoRouter та маршрути, DI-контейнер (injector.dart), точка входу застосунку

Кожен шар має чітко визначену зону відповідальності та залежить лише від шарів, що знаходяться ближче до центру архітектури. Діаграму класів модуля занять мобільного клієнта, що ілюструє реалізацію Clean Architecture у конкретному функціональному модулі, зображено на рисунку 2.8.

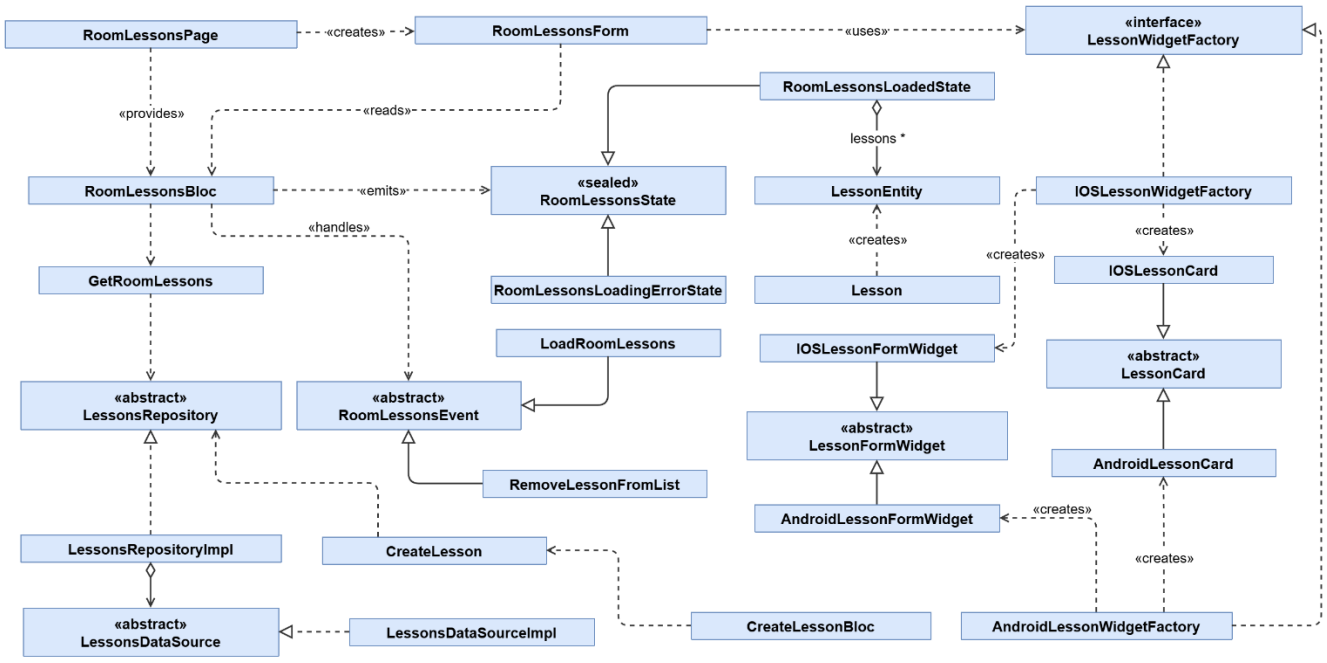

Рисунок 2.8 – Діаграма класів модуля занять мобільного клієнта

На діаграмі класів відображено взаємозв'язки між класами всіх чотирьох шарів на прикладі модуля занять. RoomLessonsPage та RoomLessonsForm – це компоненти шару Presentation, GetRoomLessons – Use Case доменного шару.

Клас з назвою LessonsRepositoryImpl – реалізація репозиторію шару Data, а LessonsDataSourceImpl забезпечує комунікацію з Django REST API через DioClient [11]. Крім наслідування чистою архітектури, також використовуються патерни проєктування там, де це доцільно. Зокрема, на діаграмі представлена абстрактна фабрика LessonWidgetFactory, що забезпечує платформо-залежне відображення компонентів для Android та iOS.

2.3.2 Управління станом та обробка потоків подій

Для управління станом мобільного клієнта застосовано патерн BLoC (Business Logic Component) – архітектурний шаблон, що забезпечує однонаправлений потік даних і відокремлює бізнес-логіку від відображення [12]. BLoC побудований на основі потоків (Streams) мови Dart, UI-шар генерує Events (події), BLoC-компонент обробляє їх та емітує States (стани), на основі яких UI перебудовується. Архітектуру патерну BLoC зображено на рисунку 2.9.

Рисунок 2.9 – Архітектура патерну BloC [12]

Взаємодія з BLoC відбувається за єдиним протоколом. Компонент UI (форма або список) надсилає подію до BLoC через метод add(event). BLoC отримує подію в обробнику (event handler), викликає відповідний Use Case, отримує результат типу Either<Failure, Entity> та емітує новий стан через emit(state). BlocBuilder або BlocListener у дереві віджетів реагує на зміну стану та перебудовує відповідну частину UI. Стани та події є незмінними (immutable) об'єктами, що унеможливлює непередбачені зміни стану з боку UI.

У застосунку TutorGlide реалізовано BLoC-компоненти для кожного функціонального модуля, автентифікації, налаштування профілю, управління кімнатами, списку занять, створення та редагування занять, управління запрошеннями та персонального розкладу. Кожен BLoC реєструється у GetIt як factory-об'єкт, забезпечуючи щоразу новий екземпляр при переході на відповідний екран. Діаграму послідовності, що відображає повний цикл обробки запиту через всі шари Clean Architecture, зображено на рисунку 2.10.

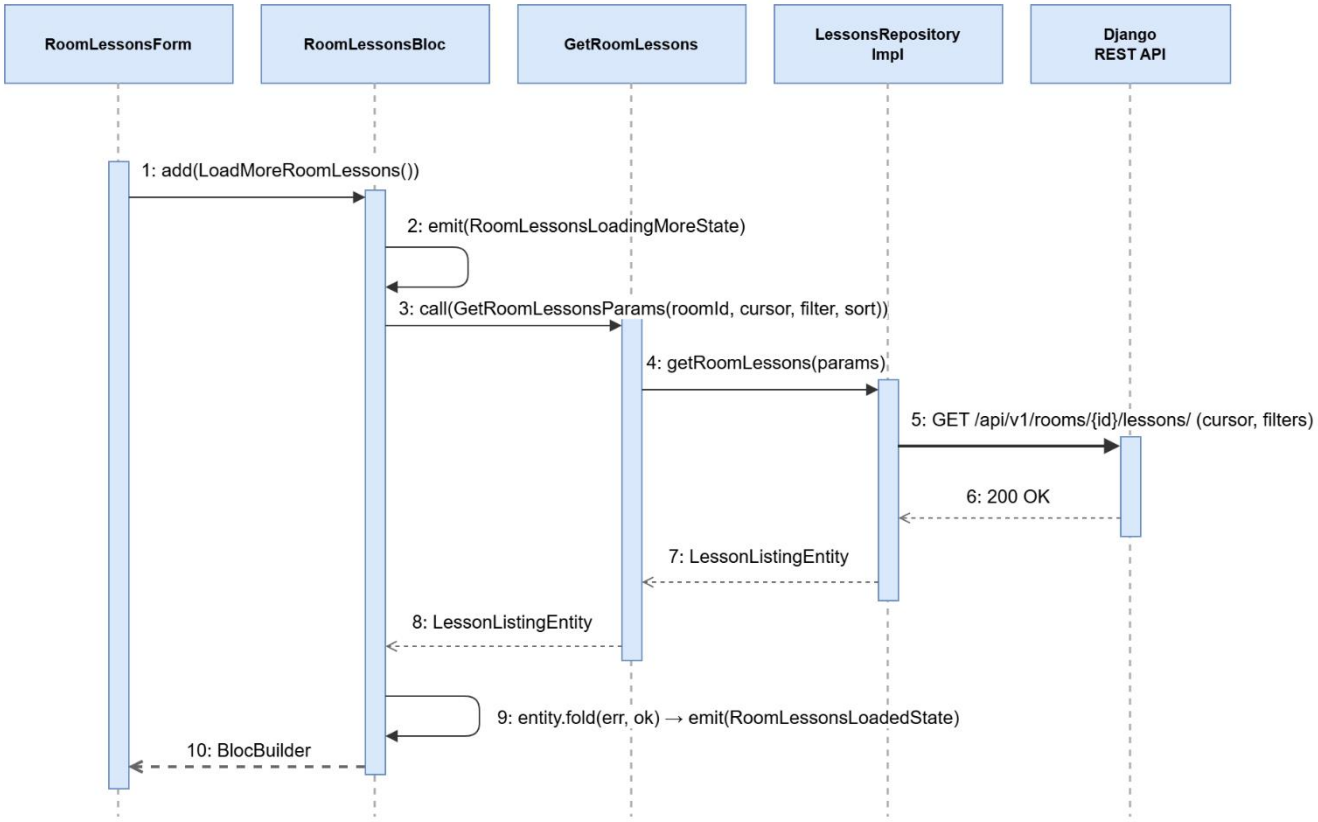

Рисунок 2.10 – Діаграма послідовності обробки запиту через шари Clean Architecture та BLoC

Діаграма послідовності наочно демонструє, як подія LoadMoreRoomLessons, згенерована UI при прокручуванні списку, послідовно проходить через RoomLessonsBloc, UseCase GetRoomLessons, LessonsRepositoryImpl та LessonsDataSourceImpl до Django REST API, а результат у вигляді JSON-відповіді перетворюється на Entity та повертається у зворотньому напрямку, після чого BLoC емітує новий стан і UI перебудовується.

2.3.3 Ін'єкція залежностей (GetIt) та конфігурація DI-контейнера

Для управління залежностями між компонентами мобільного застосунку TutorGlide використовується пакет GetIt – сервіс-локатор, що забезпечує централізовану реєстрацію та надання залежностей [13]. GetIt дозволяє отримувати залежності в будь-якому місці коду без необхідності передавати їх вручну через конструктори або параметри викликів, що суттєво спрощує структуру коду.

Усі залежності реєструються у єдиному файлі injector.dart при старті застосунку з двома стратегіями існування. registerLazySingleton застосовується для об'єктів, що створюються один раз та використовуються повторно протягом всього часу роботи застосунку, зокрема клас DioClient, Remote Data Source-класи, Repository-реалізації та Use Case-класи. registerFactory застосовується для BLoC-компонентів, оскільки кожен екран при відкритті має отримувати власний, свіжий екземпляр BLoC з чистим початковим станом, що відповідає рекомендованій практиці роботи з пакетом flutter_bloc [14].

2.3.4 Маршрутизація та навігація

Навігація у мобільному застосунку TutorGlide організована на основі бібліотеки GoRouter, яка реалізує декларативну маршрутизацію для Flutter [15]. GoRouter описує повний граф маршрутів у вигляді дерева GoRoute-об'єктів, де кожен маршрут визначає шлях URL, відповідний widget-клас сторінки та можливі дочірні маршрути. Особливістю архітектури маршрутизації TutorGlide є використання механізму redirect для захисту маршрутів відповідно до стану автентифікації та ролі користувача. Клас AppRouter підписується на зміни стану автентифікації через AsyncListenable-обгортку навколо AuthBloc. При зміні стану GoRouter автоматично викликає функцію redirect, що перенаправляє користувача на відповідний екран, для неавтентифікованого на екран входу, для ролі pending на екран вибору ролі, для ролей teacher або student на головний екран.

2.3.5 Створення HTTP-клієнта та стратегія обробки помилок

Для виконання HTTP-запитів до серверного API в мобільному клієнті TutorGlide використовується бібліотека Dio, потужний HTTP-клієнт для Dart з підтримкою перехоплювачів, тайм-аутів та конвертації відповідей [11]. Клас DioClient – це singleton-обгортка над Dio – налаштовує базовий URL, заголовки та набір перехоплювачів при ініціалізації.

В застосунку налаштовано три послідовних перехоплювачі. AuthInterceptor автоматично додає заголовок Authorization Bearer <token> до кожного запиту, отримуючи актуальний Firebase ID Token через Firebase Authentication SDK. При закінченні терміну дії токена перехоплювач прозоро оновлює його перед повторним надсиланням запиту. ErrorInterceptor перехоплює HTTP-відповіді з кодами помилок (4xx, 5xx) та перетворює їх на типізовані Dart-виключення відповідно до поля error.code у відповіді сервера.

Таким чином, це дозволяє BLoC-компонентам обробляти конкретні типи помилок та відображати точні повідомлення. LoggingInterceptor в режимі розробки виводить деталі кожного запиту та відповіді в термінал розробника для зручності відлагодження. Отже, рівень мережевої взаємодії є повністю інкапсульованим, що звільняє вищі шари від логіки управління токенами та помилками.

2.4 Реалізація підсистеми розкладу

Підсистема розкладу є одним з ключових та найбільш складних компонентів системи TutorGlide з точки зору доменної логіки. Вона охоплює модель заняття, алгоритм генерації розкладу на довільний діапазон дат, механізм виключень для окремих занять та управління часовими зонами при взаємодії клієнта і сервера, а також забезпечує узгодженість даних між різними пристроями, підтримує гнучке налаштування повторюваних подій, обробку конфліктів у розкладі та масштабованість при роботі з великою кількістю користувачів і занять.

2.4.1 Модель повторюваних та одноразових занять

Регулярне заняття задається набором полів recurrence_day – ціле число від 0 до 6, де 0 відповідає понеділку, recurrence_start_time та recurrence_end_time визначають час початку та завершення заняття в часовому поясі кімнати, поля effective_from та effective_until визначають діапазон дат, протягом якого заняття є активним. Одноразове заняття задається у форматі UTC, що дозволяє коректно порівнювати абсолютні часові відмітки незалежно від часового поясу.

При створенні заняття сервіс виконує перевірку на накладання часу нового заняття із існуючими заняттями того самого викладача в усіх його кімнатах. При виявленні конфлікту сервіс генерує виключення LessonOverlapError з посиланням на конфліктуюче заняття, що повертається клієнту як HTTP 409 Conflict. Повний бізнес-процес управління заняттями зображено на рисунку 2.11.

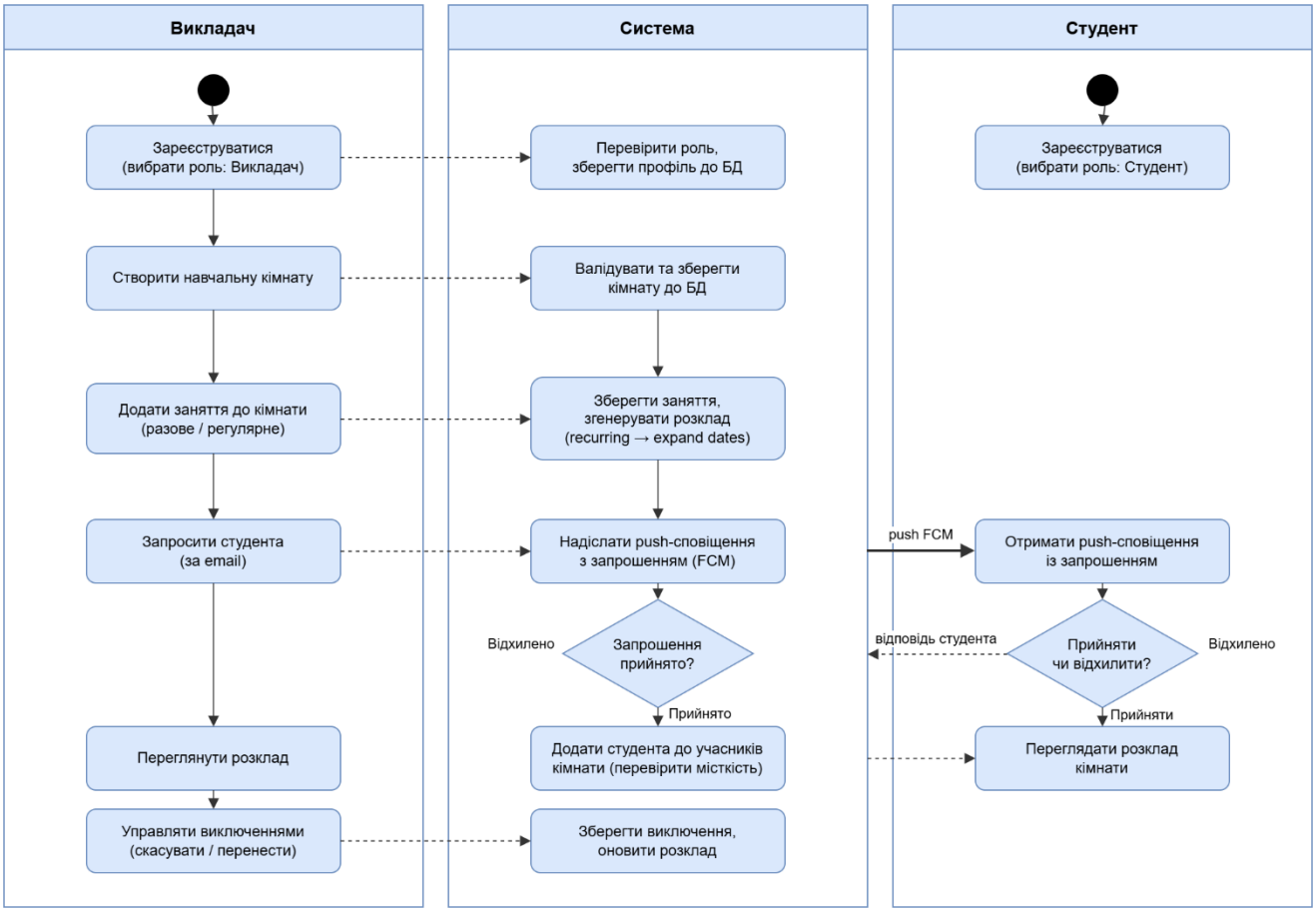

Рисунок 2.11 – Діаграма діяльності бізнес-процесу управління заняттями

Діаграма діяльності ілюструє взаємодію між трьома учасниками процесу, викладачем, системою та студентом. Викладач ініціює процес, створюючи навчальну кімнату та додаючи заняття. Система виконує збереження даних, генерацію розкладу та доставку сповіщень. Детальну послідовність взаємодії між шарами серверної архітектури при створенні заняття зображено на рисунку 2.12.

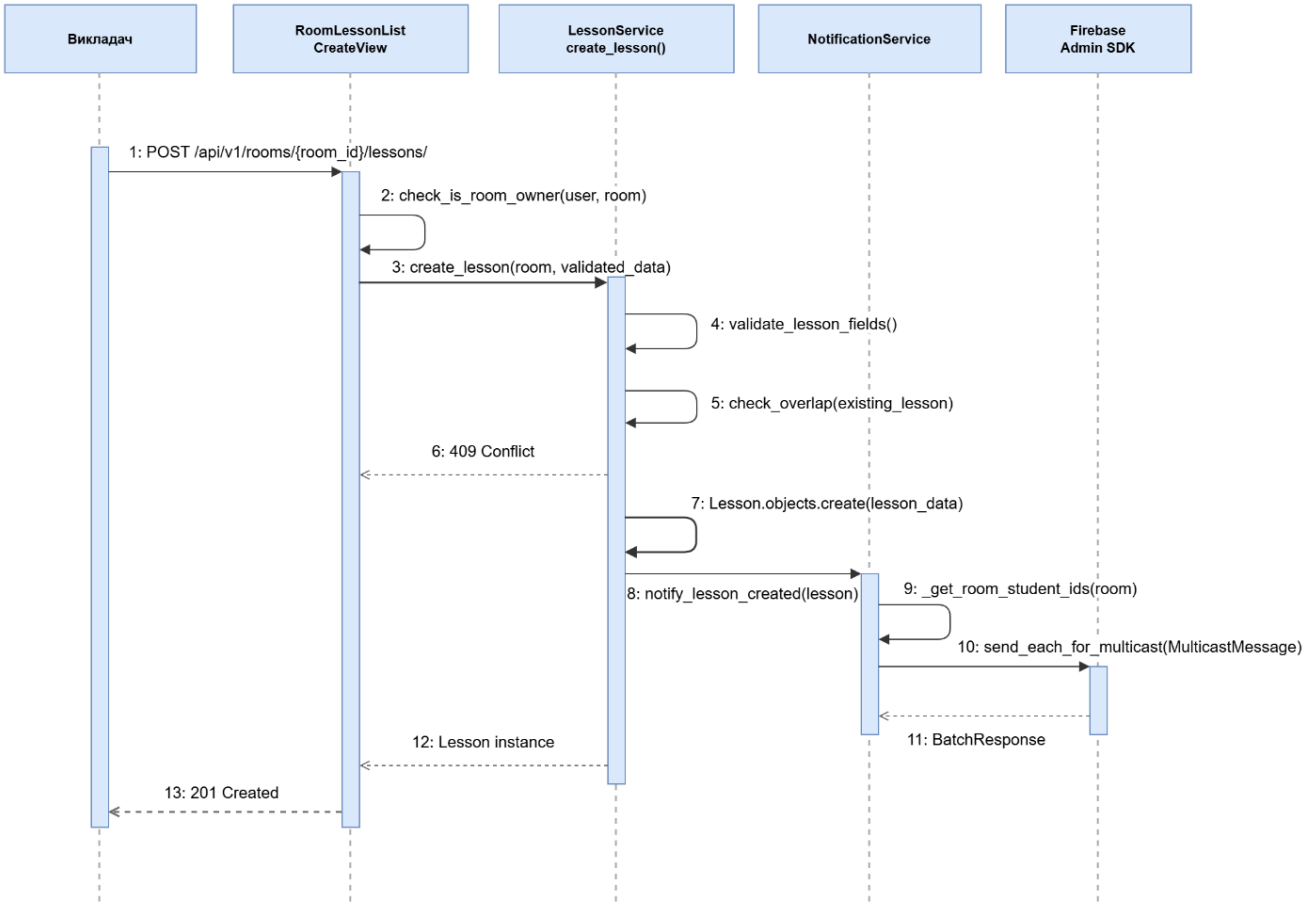

Рисунок 2.12 – Діаграма послідовності процесу створення заняття

Діаграма послідовності відображає, як запит викладача на створення заняття проходить через серіалізатор LessonCreateSerializer для первинної валідації, передається до сервісу create_lesson(), де виконується перевірка конфліктів через LessonsSelector, після чого заняття зберігається засобами Django ORM.

Механізм Django Signals автоматично ініціює відправку push-сповіщень студентам кімнати через Firebase Admin SDK. Варто зазначити, що серіалізатор відповідає виключно за валідацію вхідних даних.

2.4.2 Алгоритм генерації розкладу для заданого діапазону дат

Алгоритм генерації розкладу реалізовано функцією, яка приймає ідентифікатор користувача та діапазон дат, а потім повертає впорядкований список об'єктів LessonOccurrence – дата-класу, що представляє одне конкретне заняття.

Алгоритм виконується у такій послідовності. Спочатку система отримує перелік занять відповідного користувача. Для викладача всі заняття з кімнат, де він є власником, а для студента заняття з кімнат, де він є учасником. Для кожного регулярного заняття генерується перелік дат у заданому діапазоні, що збігаються з полем recurrence_day заняття та потрапляють в активний діапазон effective_from/effective_until. Для кожного одноразового заняття перевіряється, чи дата one_time_start потрапляє в заданий діапазон.

Для кожної згенерованої дати виконується перевірка наявності виключення у моделі LessonException. Якщо exception_type = cancelled, дата виключається зі списку результатів, а якщо exception_type = time_changed, то час заняття для даної дати замінюється на new_start_time та new_end_time з об'єкта виключення.

Усі отримані об'єкти LessonOccurrence конвертуються до часового поясу кімнати через бібліотеку pytz та сортуються за часом початку. Для студентів додатково виконується функція _mark_overlapping_lessons(), що встановлює прапор has_overlap для занять, які перетинаються за часом, але з яких студент може відвідати лише одне. Для наочності, було розроблено діаграма діяльності алгоритму генерації розкладу, що наведено у додатку Б.

Окремою особливістю алгоритму є обробка занять, що перетинають північ. Якщо час завершення заняття є меншим за час початку, алгоритм коректно обчислює тривалість з урахуванням переходу на наступну добу. Це особливо актуально для вечірніх занять, які можуть розпочинатися, наприклад, о 23:30 і тривати до 01:00 наступного дня.

Після сортування та конвертації список LessonOccurrence повертається клієнту у відповідь на запит GET /api/v1/lessons/schedule/ з параметрами start_date

та end_date. Фінальне сортування гарантує, що заняття у відповіді завжди впорядковані хронологічно, незалежно від порядку їх зберігання в базі даних.

2.4.3 Управління часовими зонами на клієнті

Часові зони є одним з найскладніших аспектів розробки систем управління розкладом, особливо при взаємодії користувачів з різних регіонів [16]. У системі TutorGlide прийнято стратегію, що базується на часовому поясі навчальної кімнати як єдиному авторитетному джерелі для відображення розкладу.

Часовий пояс кімнати зберігається у форматі бази даних часових поясів IANA (наприклад, Europe/Kyiv) та визначає базовий часовий пояс, в якому задаються поля recurrence_start_time та recurrence_end_time регулярних занять. Одноразові заняття зберігаються у UTC (DateTimeField з use_tz=True) та конвертуються до часового поясу кімнати при генерації розкладу. Для конвертації між часовими зонами на сервері використовується бібліотека pytz, що забезпечує коректне врахування переходу на літній/зимовий час.

На клієнті мобільний застосунок відображає часи занять відповідно до часового поясу кімнати, незалежно від локального часового поясу пристрою, що забезпечує узгодженість відображення розкладу для всіх учасників кімнати незалежно від їх географічного розташування.

Отже, централізоване зберігання часового поясу на рівні кімнати та конвертація виключно при генерації розкладу забезпечують коректне відображення часу та уникнення неоднозначностей.

3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування є невід'ємною складовою процесу розробки програмного забезпечення, яка гарантує відповідність реалізованої системи визначеним вимогам та забезпечує коректність її функціонування. Описано комплексний підхід до тестування розробленої мобільної системи, що охоплює автоматизовані модульні та інтеграційні тести як серверної, так і клієнтської частини, наскрізне тестування користувацьких сценаріїв за допомогою інструменту Maestro, ручне тестування інтерфейсу, а також налаштування процесу безперервної інтеграції на платформі GitHub Actions [17].

3.1 Стратегія тестування

Вибір стратегії тестування для системи TutorGlide ґрунтується на концепції піраміди тестування, яка передбачає ієрархічну організацію тестів за рівнем абстракції та вартістю їх виконання:

– Нижній рівень піраміди складають модульні тести, що перевіряють окремі функції та класи в ізоляції від інших компонентів.

– Середній рівень охоплює інтеграційні тести, які верифікують взаємодію між компонентами системи.

– Верхній рівень представлений наскрізними тестами, що імітують реальні сценарії використання застосунку.

Принципи розробки через тестування (Test-Driven Development, TDD) застосовувались при реалізації бізнес-логіки серверного рівня, зокрема сервісів і селекторів. Обрана стратегія дозволить забезпечити чітку специфікацію поведінки кожного компонента до його реалізації та мінімізувати кількість регресійних дефектів у процесі подальшої розробки. Відповідно до архітектурних рішень, ухвалених у попередньому розділі, кожен рівень системи тестується окремо, серверний рівень через pytest, мобільний клієнт через flutter_test, а наскрізна взаємодія – через Maestro [18].

Окремо від автоматизованих тестів виконується ручне тестування інтерфейсу користувача, яке дозволяє перевірити візуальну коректність відображення компонентів, коректність анімацій, поведінку форм та загальну якість взаємодії. Поєднання автоматизованих та ручних методів тестування гарантує повноцінне покриття функціональності системи на всіх рівнях. Обрані види тестів та їх характеристики окреслені у таблиці 3.1.

Таблиця 3.1 – Типи тестів та їх характеристики

Тип тестування	**Інструмент**	**Рівень**	**Покриття**	**Кількість**
Модульні (бекенд)	pytest + factory-boy	Unit	Сервіси, селектори, алгоритм розкладу	11 файлів
Інтеграційні (бекенд)	pytest-django + APIClient	Integration	REST API ендпоінти, права доступу	6 файлів
Модульні (фронтенд)	flutter_test + mocktail	Unit	Use cases, валідація, сутності	18 файлів
Наскрізні (E2E)	Maestro	End-to-End	Сценарії реальних користувачів	3 сценарії
Ручне UI-тестування	–	Manual	Візуальна коректність, UX	6 екранів

Для серверної частини, зокрема, використовується бібліотека pytest та factory-boy [19, 20]. Для клієнтської частини представлений підхід, який замінює реальні залежності тестовими. Кожен тип тестів вирішує конкретну задачу верифікації, зокрема модульні тести забезпечують правильність бізнес-логіки в ізоляції, інтеграційні – коректність взаємодії компонентів через API, наскрізні – відповідність реальним сценаріям використання [21].

3.2 Наскрізне та ручне тестування

Автоматизовані модульні та інтеграційні тести верифікують коректність бізнес-логіки та інтерфейсів, однак не охоплюють реальний досвід взаємодії користувача з мобільним застосунком, тому застосовуються два підходи, автоматизоване наскрізне тестування через Maestro та ручне тестування інтерфейсу з документуванням результатів у вигляді скріншотів.

Поєднання обох підходів формує багаторівневу систему контролю якості, яка охоплює технічну коректність та зручність використання застосунку.

3.2.1 Автоматизоване E2E-тестування

Maestro – це спеціалізований інструмент для наскрізного тестування мобільних застосунків, що описує сценарії тестування у декларативних YAML-файлах. На відміну від низькорівневих інструментів, Maestro взаємодіє з реальним емулятором Android через ідентифікатори доступності (accessibility IDs) та текстові мітки елементів, що наближає умови тестування до реальної роботи кінцевого користувача. Після кожного виконання тестового сценарію Maestro автоматично генерує відеозапис у форматі .mp4, що фіксує весь процес тестування і може бути використаний для аналізу виявлених дефектів.

Середовище Maestro Studio надає інтерактивний графічний інтерфейс. Це значно спрощує процес написання YAML-сценаріїв, дозволяючи безпосередньо виявляти accessibility ID елементів без необхідності аналізу вихідного коду. Інтерфейс Maestro Studio наведено на рисунку 3.1.

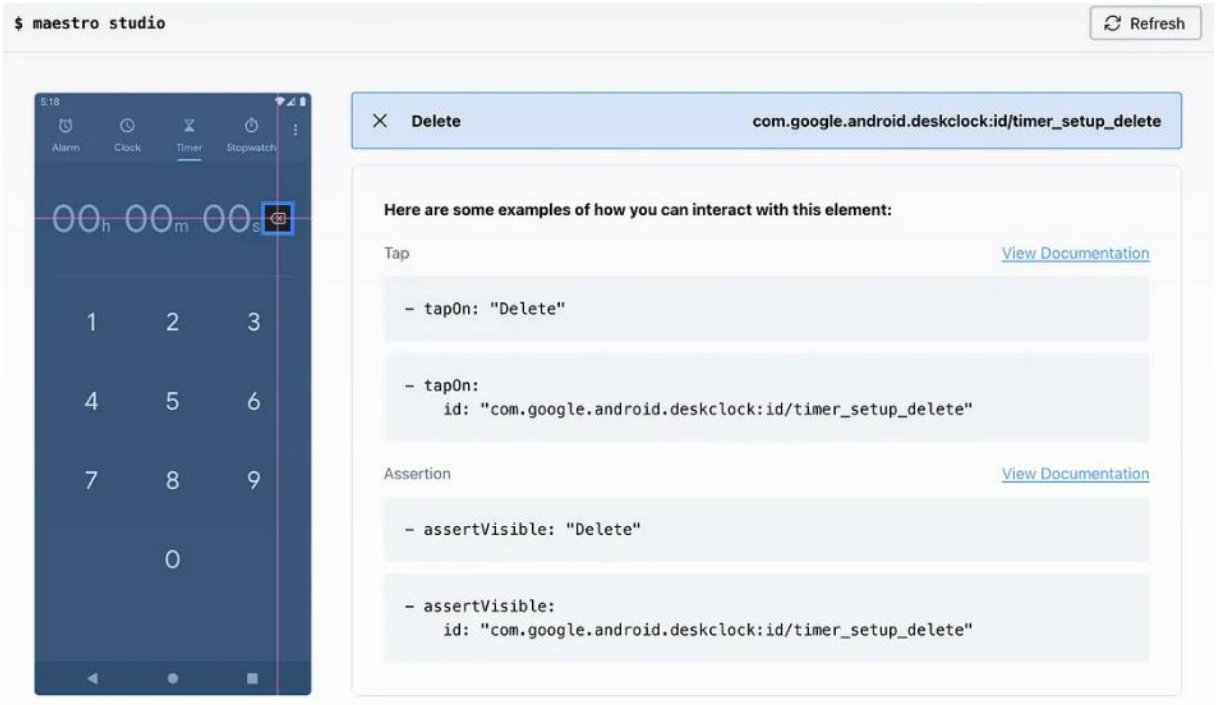

Рисунок 3.1 – Інтерфейс Maestro Studio з відображенням емулятора

В межах кваліфікаційної роботи було описано 3 тестових сценарії, лістинги коду яких наведені у додатку В. Перший тестовий сценарій, описаний у файлі 1-user-authentication.yaml, перевіряє процес реєстрації та входу в систему. Сценарій запускає застосунок із очищеним станом (clearState: true), переходить на екран реєстрації, заповнює форму тестовими обліковими даними (email та пароль), підтверджує реєстрацію, після чого здійснює вхід із тими самими даними та верифікує успішне відображення головного екрану. На рисунку 3.2 зображено результати виконання цього сценарію.

```
Running on emulator-5554

> Flow: 1-user-authentication

  +   Launch app "com.tutorglide.app" with clear state
  +   Start recording recording-auth
  +   Run flow when id: navigate_to_sign_up_button is visible
  +   Tap on id: email_field
  +   Input text test.user@example.com
  +   Tap on id: password_field
  +   Input text TestPassword123!
  +   Tap on id: confirm_password_field
  +   Input text TestPassword123!
  +   Tap on id: sign_up_submit_button
  +   Run flow when id: navigate_to_sign_in_button is visible
  +   Tap on id: email_field
  +   Input text test.user@example.com
  +   Tap on id: password_field
  +   Input text TestPassword123!
  +   Tap on id: sign_in_submit_button
  +   Stop recording
```

Рисунок 3.2 – Результати виконання сценарію автентифікації користувача

Другий сценарій 2-create-room-flow.yaml тестує створення навчальної кімнати. Сценарій виконується з передумовою, тобто станом, або характеристиками, які повині передувати виконанню сценарію.

Передумовою, в даному випадку, є авторизований стан застосунку (clearState: false). Сценарій відкриває бічне меню, переходить на сторінку кімнат, натискає кнопку «New Room» в AppBar, заповнює форму назвою «Mathematics Study Group» та підтверджує створення. Команди waitForAnimationToEnd та assertVisible гарантують стабільність сценарію незалежно від швидкості роботи емулятора. Результати виконання наведено на рисунку 3.3.

```
Running on emulator-5554

> Flow: 2-create-room-flow

  +   Launch app "com.tutorglide.app"
  +   Start recording recording-create-room
  +   Tap on id: drawer_menu_button
  +   Wait for animation to end
  +   Tap on id: drawer_rooms_item
  +   Assert that "Rooms" is visible
  +   Tap on id: create_room_appbar_button
  +   Assert that id: room_name_field is visible
  +   Tap on id: room_name_field
  +   Input text Mathematics Study Group
  +   Tap on id: create_room_submit_button
  +   Assert that id: create_room_submit_button is not visible
  +   Stop recording
```

Рисунок 3.3 – Результати виконання сценарію створення навчальної кімнати

Третій сценарій 3-create-lesson-flow.yaml здійснює наскрізну перевірку навігації між усіма основними розділами застосунку, зокрема Calendar, Rooms, Invitations, Profile та Settings. Кожен перехід верифікується командами assertVisible і assertNotVisible, що підтверджують коректне відображення відповідного екрану та зникнення попереднього. Описаний сценарій гарантує цілісність маршрутизації застосунку та відсутність критичних збоїв при навігації між основними розділами. Результати виконання наведено на рисунку 3.4.

```
> Flow: 3-create-lesson-flow

  +   Launch app "com.tutorglide.app"
  +   Start recording recording-navigation
  +   Assert that "Calendar" is visible
  +   Tap on id: drawer_menu_button
  +   Wait for animation to end
  +   Tap on id: drawer_rooms_item
  +   Assert that "Rooms" is visible
  +   Tap on id: drawer_menu_button
  +   Wait for animation to end
  +   Tap on id: drawer_menu_button
  +   Wait for animation to end
  +   Tap on id: drawer_calendar_item
  +   Assert that "Calendar" is visible
  +   Stop recording
```

Рисунок 3.4 – Результати виконання сценарію навігації по застосунку

Усі три сценарії успішно завершуються на реальному емуляторі Android. Відеозаписи .mp4, що автоматично генеруються після кожного запуску, слугують доказом коректної роботи відповідних функцій застосунку та можуть бути використані для демонстрації функціональності системи. Всі три реалізовані Maestro-сценарії та їх результати наведені у таблиці 3.2.

Таблиця 3.2 – Тестові сценарії Maestro

Сценарій	**Файл**	**Кількість кроків**	**Результат**
Автентифікація	1-user-authentication.yaml	12	Пройдено
Створення кімнати	2-create-room-flow.yaml	10	Пройдено
Навігація застосунку	3-create-lesson-flow.yaml	18	Пройдено

3.2.2 Ручне тестування інтерфейсу

Ручне тестування доповнює автоматизоване верифікацією аспектів, що важко формалізувати у вигляді сценаріїв, зокрема відповідність візуального відображення дизайн-макетам, коректність анімацій переходів між екранами, читабельність та доступність елементів інтерфейсу.

Тестування виконується безпосередньо на емуляторі Android з перевіркою ключових користувацьких сценаріїв у реальному режимі роботи застосунку. Особлива увага надається граничним станам інтерфейсу – відображенню порожніх списків, поведінці при відсутності мережевого з'єднання та коректності обробки помилок на рівні UI. Кожен перевірений сценарій супроводжувався скріншотами відповідних екранів, що дає змогу сформувати наочну документацію фактичного стану застосунку на момент тестування.

Сторінка календаря – основний екран застосунку TutorGlide, що відображає розклад занять у вигляді горизонтальної стрічки дат та вертикального списку занять поточного дня. Перевірялась коректність відображення регулярних та разових занять, індикація конфліктів розкладу та навігація між датами. Результат перевірки сторінки календаря наведено на рисунку 3.5.

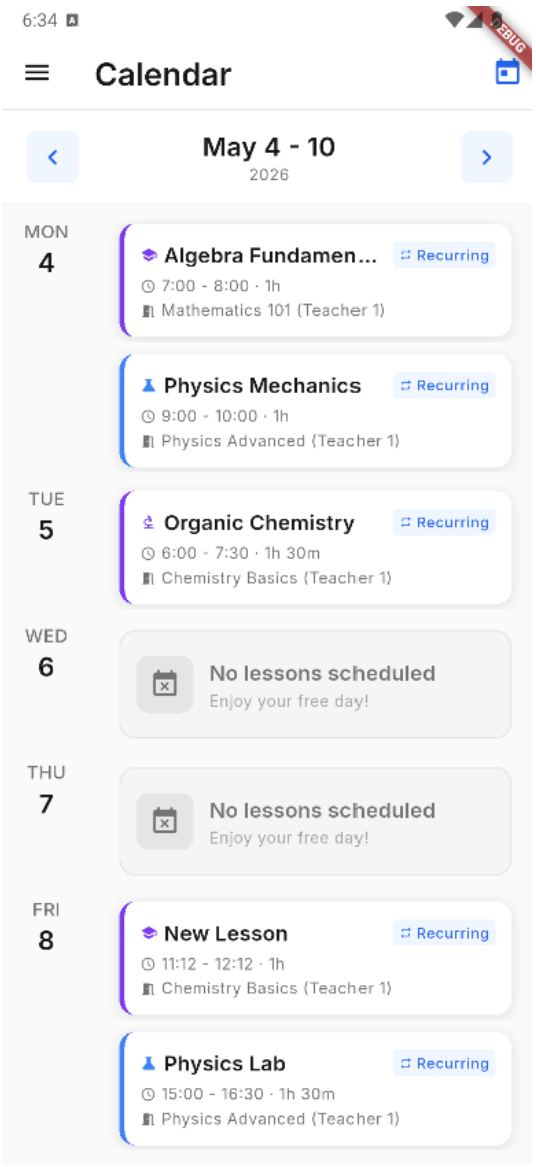

Рисунок 3.5 – Ручне тестування сторінки календаря

Наступний елемент ручного тестування – це діалог створення нової навчальної кімнати. Діалог містить текстове поле для введення назви кімнати з лічильником символів та кнопку підтвердження, що забезпечує інтуїтивно зрозумілий процес створення навчального простору.

Перевірялась коректність роботи текстового поля, зокрема обрізання пробілів та обмеження максимальної довжини назви, а також стани кнопок підтвердження та скасування. Окремо тестувалась поведінка інтерфейсу при введенні граничних значень – порожнього рядка, назви з одного символу та рядка максимально дозволеної довжини.

Візуальне оформлення діалогу відповідає загальному стилю застосунку, використовується характерна синя кольорова схема, заокруглені кути елементів та чітка типографіка, що узгоджується з рештою екранів TutorGlide. Протестований екран наведено на рисунку 3.6.

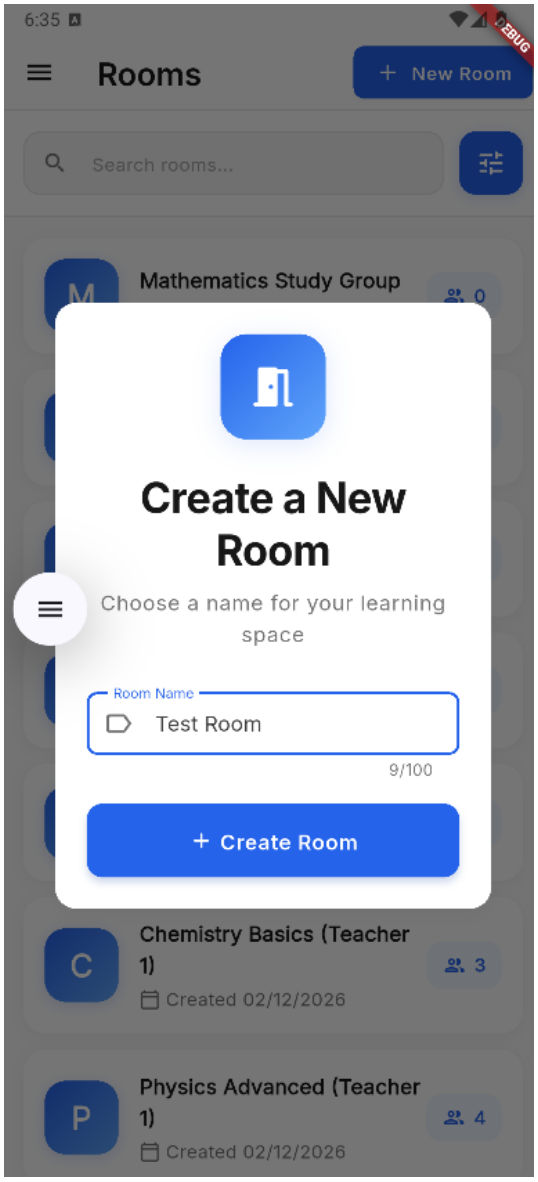

Рисунок 3.6 – Ручне тестування діалогу створення кімнати

Наступний протестований екран – редагування заняття, який відображає форму з повним набором параметрів, зокрема назва, тип заняття, часовий діапазон та дата. Тестувалась коректна підстановка поточних значень при відкритті форми редагування, валідація часових обмежень та збереження змін.

Окремо протестовано правильність, обраних користувачем, часових проміжків. Спочатку було обрано час початку заняття, потім перевірено, чи блокує мобільний додаток можливість обрати такий час закінчення заняття, який передує часу початку. Встановлено, що додаток коректно опрацьовує часові проміжки, тестування описаного екрану наведено на рисунку 3.7.

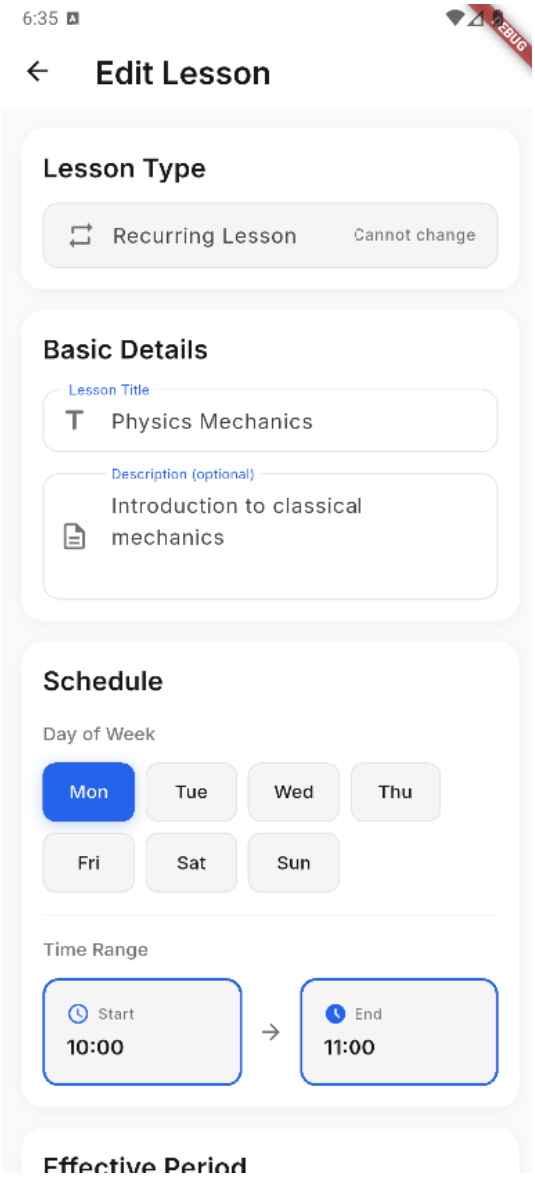

Рисунок 3.7 – Ручне тестування екрана редагування заняття

Рисунок 3.8 демонструє екран виключень заняття, що дозволяє вчителю переглядати та керувати скасуваннями і перенесеннями окремих дат регулярного заняття. Тестувалась коректність відображення типів виключень та дат, на які вони поширюються.

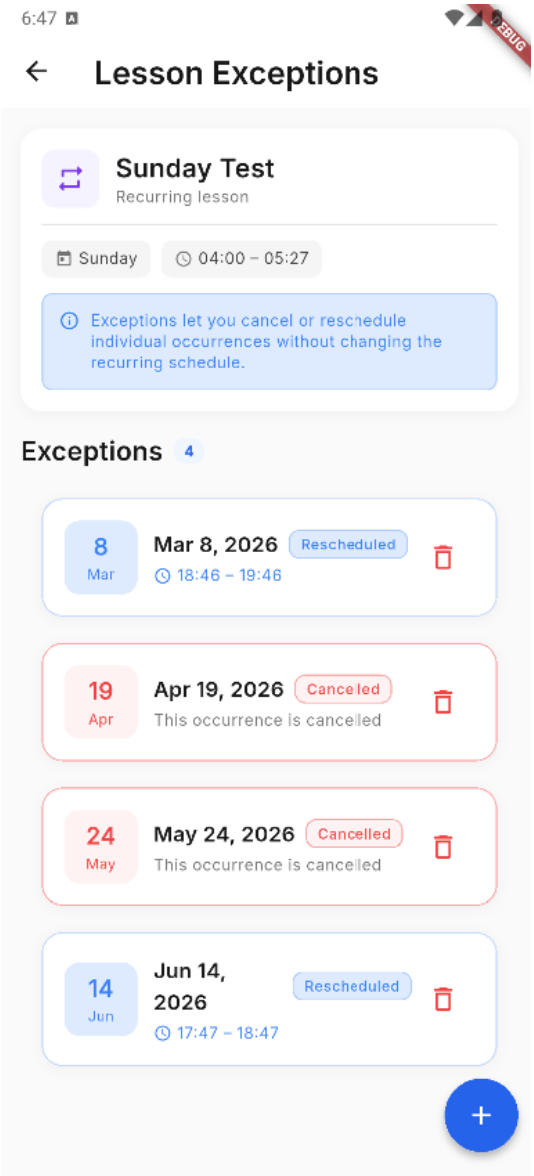

Рисунок 3.8 – Ручне тестування екрана виключень занять

Результати тестування підтвердили відповідність інтерфейсу вимогам у всіх перевірених сценаріях. Виявлені у процесі тестування незначні розбіжності у відступах та розмірах елементів були усунені до фінальної версії застосунку.

3.3 Безперервна інтеграція

Для автоматизації виконання тестів та забезпечення постійного контролю якості коду в обох частинах системи налаштовано процес безперервної інтеграції (CI) на платформі GitHub Actions [22].

Конфігурація CI зберігається у файлі .github/workflows/tests.yml у кожному репозиторії та запускається автоматично при кожному push-коміті до гілки master та при відкритті pull request у цю гілку. Такий підхід унеможливлює злиття коду, який не пройшов усі автоматизовані перевірки.

Конфігурація CI для серверної частини передбачає запуск контейнера PostgreSQL 15 як сервісу з автоматичною перевіркою готовності через pg_isready. Після успішного запуску сервісу виконуються такі кроки:

1. Встановлення Python 3.13.
2. Інсталяція залежностей з requirements.txt.
3. Запуск тестів командою pytest --junitxml=test-results.xml [19].
4. Публікація результатів через дію dorny/test-reporter.

Наявність бази даних у процесі CI гарантує, що інтеграційні тести виконуються в умовах, наближених до релізного середовища.

На рисунку 3.9 відображено типовий вигляд pull request у GitHub під час виконання CI-перевірок. Статус виконання тестів відображається безпосередньо у PR, тому всі учасникам процесу розробки можуть отримувати зворотній зв'язок щодо якості запропонованих змін. Злиття pull request блокується до успішного завершення всіх автоматизованих перевірок.

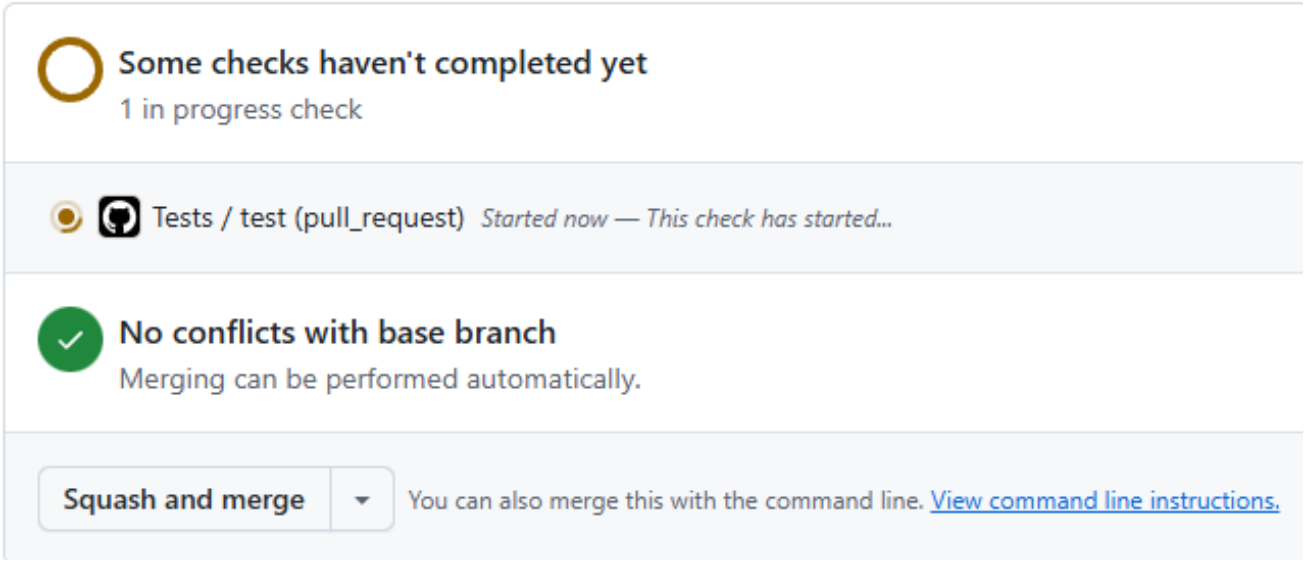

Рисунок 3.9 – Виконання CI-перевірок у pull request на GitHub

Після виконання тестів, генеруються спеціальні звіти. Результати тестування клієнтської та серверної частин наведені на рисунках 3.10 та 3.11 відповідно.

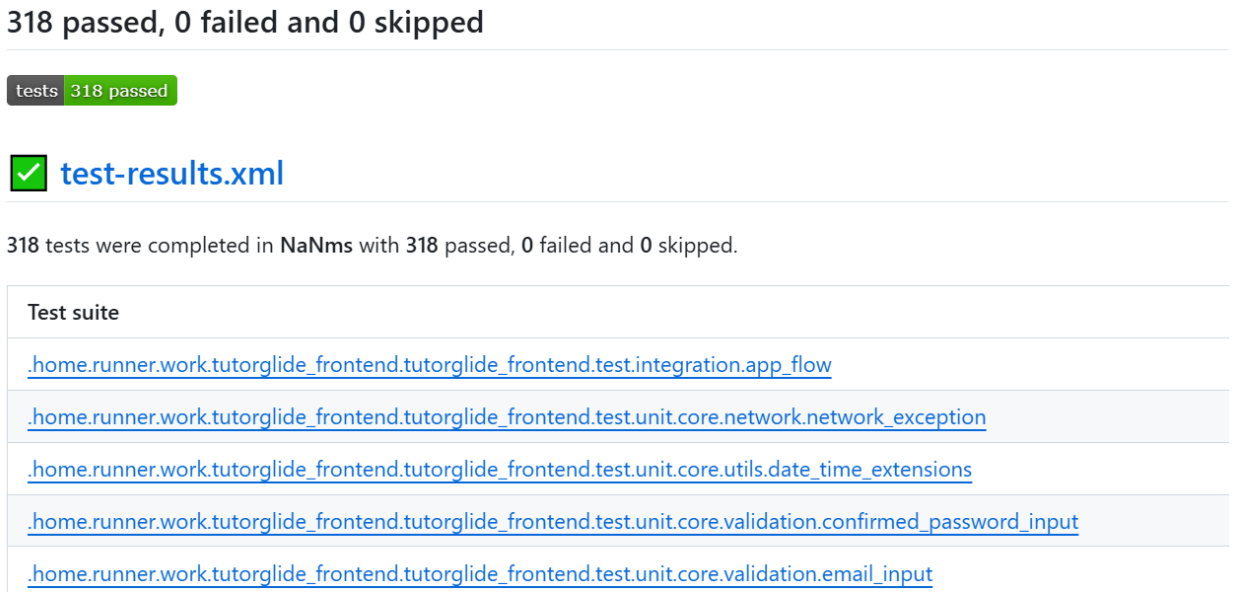

Рисунок 3.10 – Звіт виконання тестів фронтенду в GitHub Actions

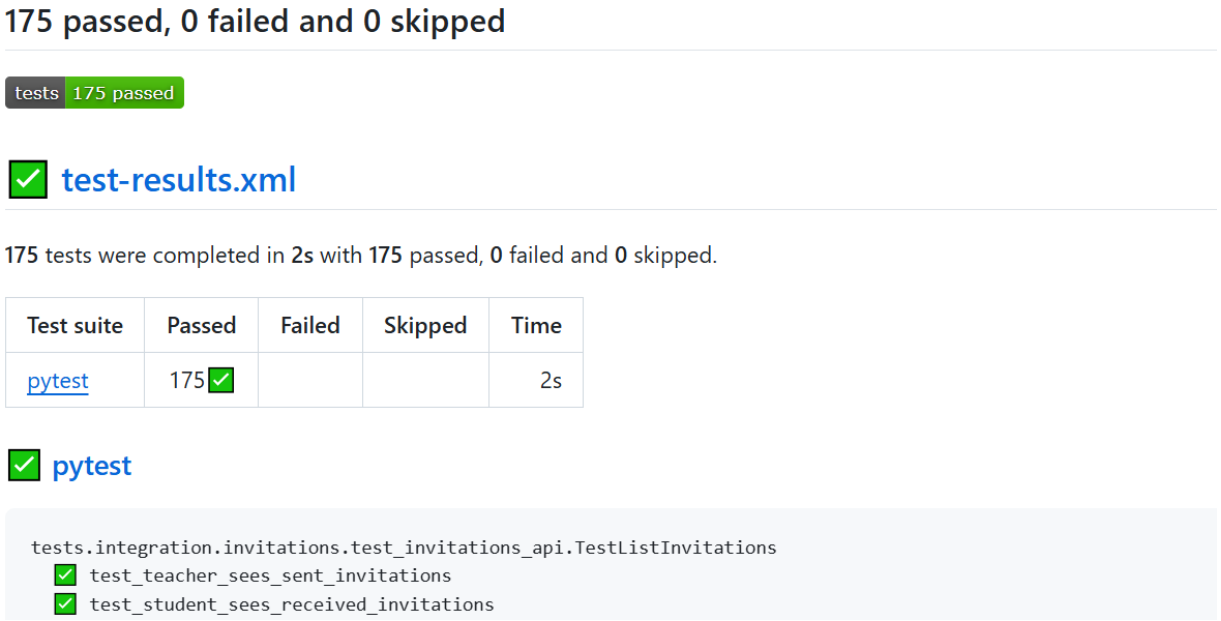

Рисунок 3.11 – Звіт виконання тестів бекенду в GitHub Actions

Налаштована система безперервної інтеграції забезпечує автоматичну верифікацію коректності кожної зміни в обох частинах системи. CI-пайплайни гарантують підтримку якості кодової бази на стабільному рівні протягом усього процесу розробки.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

В даному розділі розглядається два ключових питання, зокрема моделювання та прогнозування небезпечних ситуацій як інструмент запобігання небезпекам, а також ергономічні вимоги до організації робочого місця оператора ПК.

4.1 Моделювання та прогнозування небезпечних ситуацій

Небезпечна ситуація – це сукупність умов і обставин, за яких створюється реальна загроза життю та здоров'ю людини, матеріальним цінностям і навколишньому природному середовищу. Зважаючи на ймовірнісну природу більшості небезпек, ключовими інструментами їх виявлення та оцінювання у безпеці життєдіяльності виступають моделювання і прогнозування [23].

Моделювання небезпечних ситуацій – це метод дослідження процесів виникнення та розвитку небезпек шляхом побудови й аналізу їх моделей, тобто спрощених формалізованих описів, що відтворюють найістотніші властивості та закономірності реального явища. Мета моделювання полягає у вивченні механізмів зародження небезпеки, встановленні причинно-наслідкових зв'язків між чинниками ризику та можливими наслідками, а також у кількісному оцінюванні ймовірності й масштабів подій без проведення натурних експериментів.

За способом опису та характером отримуваних результатів розрізняють кілька типів моделей небезпечних ситуацій. Детерміновані (фізико-математичні) моделі описують перебіг процесу однозначними залежностями та застосовуються для розрахунку зон ураження, полів концентрацій шкідливих речовин, параметрів ударної хвилі тощо. Імовірнісні (стохастичні) моделі враховують випадковий характер виникнення небезпечних подій і дають оцінку ймовірності їх настання. Особливе місце посідають структурно-логічні моделі – «дерево подій» та «дерево відмов», які у вигляді логічних схем відображають послідовності й комбінації відмов та помилок, що призводять до аварії, і дозволяють виявити найвразливіші ланки системи [23, 24].

Прогнозування небезпечних ситуацій – це науково обґрунтоване передбачення ймовірності виникнення, місця, часу, характеру та можливих наслідків небезпечних і надзвичайних ситуацій. Залежно від завчасності розрізняють довгострокове, середньострокове, короткострокове та оперативне прогнозування. За змістом прогноз має дати відповідь на питання, де і коли може виникнути небезпечна подія, які чинники її спричинять, якими будуть масштаби ураження та який обсяг сил і засобів потрібен для ліквідації наслідків.

Для прогнозування застосовують такі основні групи методів як статистичні, експертні, що ґрунтуються на узагальненні думок фахівців за умов браку статистичних даних, аналітично-розрахункові, які використовують математичні моделі фізичних і хімічних процесів, методи аналогій, що переносять закономірності вже вивчених ситуацій на подібні, а також методи математичного та імітаційного моделювання із застосуванням обчислювальної техніки [24].

Сучасною методологічною основою моделювання та прогнозування є ризик-орієнтований підхід, у межах якого небезпека характеризується кількісно, через ризик. Ризик визначається як добуток імовірності виникнення небезпечної події на величину очікуваних наслідків (збитків). Розрізняють індивідуальний та колективний (соціальний) ризик, а також поняття прийнятного (допустимого) ризику – рівня, з яким суспільство погоджується заради отримання певних благ. Оцінювання та порівняння ризиків дозволяють ранжувати небезпеки за ступенем загрози й спрямовувати обмежені ресурси на запобігання насамперед найбільш значущим з них.

Процес визначення та прогнозування небезпечних ситуацій є послідовністю взаємопов'язаних етапів [23]:

1. Виявлення джерел небезпеки та потенційно небезпечних об'єктів, складання їх переліку.

2. Оцінювання ймовірності виникнення небезпечної події на основі статистичних даних та обраних моделей.

3. Розрахунок зон ураження, можливих людських та матеріальних втрат за різними сценаріями.

4. Оцінювання потреби у силах і засобах, необхідних для локалізації та ліквідації наслідків.

5. Розроблення запобіжних та захисних заходів і планів реагування, спрямованих на зниження ризику до прийнятного рівня.

Таким чином, моделювання та прогнозування є нерозривно пов'язаними складовими управління безпекою, що включають перехід від реагування на вже скоєні події до їх завчасного запобігання. Разом вони створюють інформаційну основу для прийняття обґрунтованих управлінських рішень, планування захисних заходів та раціонального розподілу ресурсів, чим суттєво підвищують рівень захищеності людини й навколишнього середовища від небезпечних та надзвичайних ситуацій.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Ергономіка робочого місця оператора персонального комп'ютера є прикладною науково-технічною дисципліною, що вивчає та оптимізує взаємодію людини з технічними засобами й виробничим середовищем з метою досягнення ефективності, безпеки та комфорту праці. Нормативну базу в Україні складають НПАОП 0.00-7.15-18, ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами», ДСТУ 8604:2015 «Ергономіка. Робочі місця з відеодисплейними терміналами», та ДБН В.2.5-28:2018 та ДСН 3.3.6.042-99 [25].

Вимоги до виробничого приміщення. Відповідно до ДСанПіН 3.3.2.007-98 площа на одне робоче місце з ВДТ має становити не менше 6 м², об'єм виробничого приміщення – не менше 20 м³ на одного працівника. Параметри мікроклімату нормуються відповідно до ДСН 3.3.6.042-99 [27]. У холодний період температура повітря 22-24 °С, відносна вологість 40-60 %, швидкість руху повітря не більше 0,1 м/с, а у теплий період 23-25 °С, вологість 40-60 %, швидкість повітря до 0,1 м/с. Повітрообмін забезпечується природною або примусовою вентиляцією, кратність якої визначається проєктом відповідно до ДБН В.2.2-28:2010.

Природне освітлення повинно бути лівобічним відносно робочого місця оператора. Нормований рівень штучної освітленості на горизонтальній робочій поверхні для приміщень з ВДТ має становити не менше 300-500 лк відповідно до ДБН В.2.5-28:2018 [26]. Рівномірність освітленості (відношення мінімальної до максимальної) не повинна бути нижчою за 0,4. Монітор слід орієнтувати перпендикулярно до вікон для уникнення прямих та відбитих відблисків на поверхні екрана, рекомендується застосовувати жалюзі або світлорозсіювальні перегородки, а також монітори з антивідблисковим покриттям.

Геометричні параметри робочого місця є визначальними для профілактики м'язово-скелетних розладів та підтримання стабільної працездатності оператора ПК. На рисунку 4.1 наведено схему раціональної організації робочого місця з позначенням нормованих ергономічних параметрів.

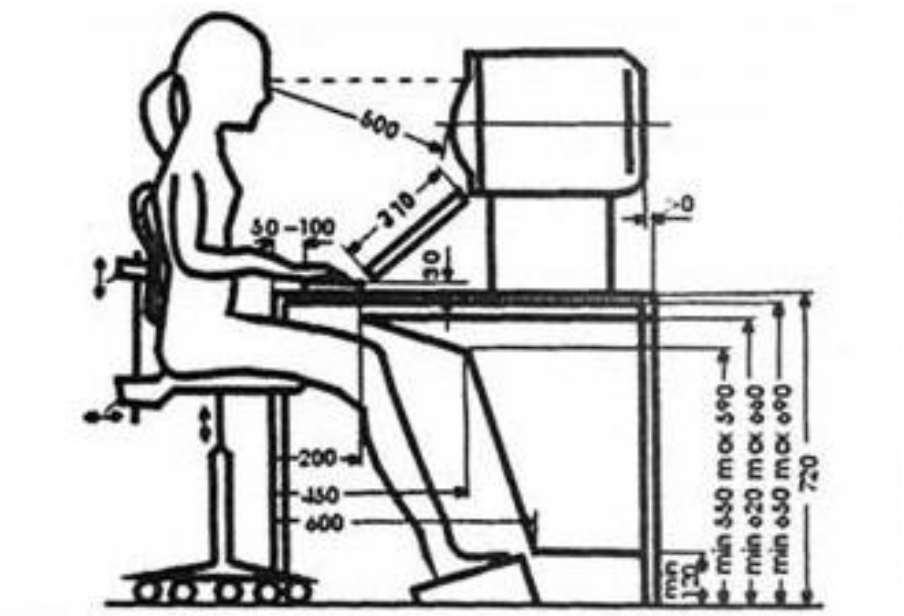

Рисунок 4.1 – Схема організації робочого місця оператора ПК [27]

Відповідно до нормативних вимог та рекомендацій правильна організація робочого місця передбачає дотримання таких параметрів [25]:

– Висота робочої поверхні столу 680-800 мм від рівня підлоги з можливістю регулювання під індивідуальний зріст оператора.

– Відстань від рогівки ока до поверхні екрана монітора 600-700 мм.

– Кут у ліктьовому суглобі при горизонтальному положенні передпліч близько 90°, передпліччя спираються на підлокітники крісла або поверхню столу без статичного напруження.

– Кут між стегном і тулубом (у тазостегновому суглобі) близько 90°, ступні повністю спираються на підлогу або підставку для ніг з регулюванням за висотою та кутом нахилу.

– Клавіатура розміщується на відстані 10-30 мм рахуючи, що він знаходяться від переднього краю столу, для підтримки зап'ясть рекомендується підставка або гелева опора.

Монітор повинен мати роздільну здатність не нижче 1920×1080 пікселів, частоту вертикальної розгортки не менше 60 Гц, регульовані яскравість і контраст, матову поверхню екрана. Клавіатура має бути відокремленою від монітора, із матовим покриттям клавіш і товщиною, що не перевищує 30 мм. Маніпулятор «миша» підбирається відповідно до характеристик долоні оператора.

Робоче крісло має відповідати вимогам ДСТУ 8604:2015 та бути регульованим за висотою сидіння (420-550 мм від рівня підлоги), висотою та кутом нахилу спинки. Спинка крісла повинна мати підтримку поперекового відділу хребта та нахилятися відносно сидіння під кутом 95-110°. Відповідно до вимог НПАОП 0.00-7.15-18 оператор зобов'язаний регулярно змінювати робочу позу, уникаючи статичного положення тривалістю понад 30-45 хвилин поспіль [25].

Правильна ергономічна організація робочого місця оператора ПК є необхідною передумовою збереження здоров'я, підтримання стабільної працездатності та виконання вимог чинного законодавства про охорону праці. Дотримання описаних санітарних норм і ергономічних рекомендацій безпосередньо впливає на якість та ефективність виконання задач з розробки, тестування та промислової експлуатації програмного забезпечення.

Отже, правильна ергономічна організація робочого місця оператора ПК є комплексним завданням, що охоплює вимоги до виробничого приміщення, освітлення, мікроклімату та геометричних параметрів розміщення обладнання.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи, було вирішено технічну задачу розробки мобільної системи обліку та управління навчальними заняттями на базі Flutter для платформ Android та iOS.

Проведено аналіз предметної області управління навчальними заняттями та огляд існуючих програмних рішень. Встановлено чотири ключові предметно-специфічні задачі, серед яких підтримка повторюваних занять з механізмом виключень, управління часовими зонами та крос-опівнічними заняттями, виявлення конфліктів у розкладі.

Спроєктовано та реалізовано повну систему на базі клієнт-серверної архітектури. Серверну частину побудовано на Django 5.2 з Django REST Framework, організовано у вигляді восьми функціонально відокремлених застосунків з дотриманням патерну Service/Selector. Мобільний клієнт розроблено на Flutter з дотриманням принципів чистої архітектури.

Розроблено та реалізовано комплексну стратегію тестування відповідно до піраміди тестів. Для серверної частини написано модульні тести сервісного шару та інтеграційні тести REST API ендпоінтів. Для мобільного клієнта було реалізовано три наскрізних E2E-сценарії засобами Maestro. Налаштовано систему безперервної інтеграції на GitHub Actions для обох частин системи.

Практичне значення роботи полягає в тому, що розроблена система є функціональним мобільним застосунком, готовим до використання вчителями та студентами для управління навчальним процесом. Реалізовані архітектурні рішення демонструють застосування сучасних підходів до розробки масштабованих мобільних систем та можуть бути використані як еталонна реалізація для подібних проєктів.

Достовірність отриманих результатів підтверджується комплексним тестуванням системи, зокрема модульними, інтеграційними та наскрізними тестами, а також ручним тестуванням інтерфейсу користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Михалик Д.М., Цуприк Г.Б., Бревус В.М. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за освітньо-професійною програмою «Інженерія програмного забезпечення» спеціальності 121 – «Інженерія програмного забезпечення» всіх форм навчання. – Тернопіль : ТНТУ ім. І. Пулюя, 2024. – 45 с.

2. Flutter – Open-Source UI Toolkit for Building Natively Compiled Cross-Platform Applications from a Single Codebase. Official Documentation. URL: https://docs.flutter.dev (дата звернення: 07.02.2026).

3. Django – A High-Level Python Web Framework That Encourages Rapid Development and Clean, Pragmatic Design. Official Documentation. URL: https://docs.djangoproject.com (дата звернення: 14.02.2026).

4. Django REST Framework – A Powerful and Flexible Toolkit for Building Web APIs on Top of Django. Official Documentation. URL: https://www.django-rest-framework.org (дата звернення: 14.02.2026).

5. Firebase – Google's Platform for Mobile and Web Application Development: Authentication, Realtime Database, Cloud Storage, and Hosting. Official Documentation. URL: https://firebase.google.com/docs (дата звернення: 21.02.2026).

6. Firebase Cloud Messaging – A Cross-Platform Cloud Solution for Delivering Push Notifications and Messages to Android, iOS, and Web Clients. Overview. URL: https://firebase.google.com/docs/cloud-messaging (дата звернення: 21.02.2026).

7. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. – 533 p.

8. OpenAPI Specification – A Standard, Language-Agnostic Interface Description for HTTP APIs Enabling Both Humans and Computers to Discover Service Capabilities. Version 3.1. URL: https://swagger.io/specification/ (дата звернення: 03.03.2026).

9. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.

10. Martin R. C. The Clean Architecture – Organizing Application Code Into Concentric Layers Around Business Rules with the Dependency Rule. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 03.03.2026).

11. Flutter Cookbook: Networking – Fetching Data from the Internet via HTTP Requests, JSON Deserialization, and Asynchronous Programming. URL: https://docs.flutter.dev/cookbook/networking/fetch-data (дата звернення: 11.03.2026).

12. Flutter – Introduction to State Management: Ephemeral State, Application State, and Reactive UI Patterns in Flutter Applications. URL: https://docs.flutter.dev/data-and-backend/state-mgmt/intro (дата звернення: 11.03.2026).

13. Fowler M. Inversion of Control Containers and the Dependency Injection pattern. URL: https://martinfowler.com/articles/injection.html (дата звернення: 18.03.2026).

14. Seemann M., van Deursen S. Dependency Injection Principles, Practices, and Patterns. – Manning Publications, 2019. – 584 p.

15. Flutter – Navigation and Routing: Managing Screen Transitions, Named Routes, and the Declarative Router API in Flutter. URL: https://docs.flutter.dev/ui/navigation (дата звернення: 18.03.2026).

16. IANA Time Zone Database – The Internet Assigned Numbers Authority Repository of Canonical Time Zone Identifiers, UTC Offset Rules, and Historical Daylight Saving Time Data. URL: https://www.iana.org/time-zones (дата звернення: 25.03.2026).

17. Vocke H. The Practical Test Pyramid: A Guide to Structuring and Balancing Unit, Integration, and End-to-End Tests in Modern Software Projects. URL: https://martinfowler.com/articles/practical-test-pyramid.html (дата звернення: 23.04.2026).

18. Beck K. Test-Driven Development: By Example. – Addison-Wesley, 2002. – 240 p.

19. pytest – A Mature Full-Featured Python Testing Framework for Writing Simple, Scalable, and Readable Automated Tests. Official Documentation. URL: https://docs.pytest.org (дата звернення: 02.04.2026).

20. factory-boy – A Fixtures Replacement Library for Generating Realistic Test Data Objects in Python with Django and SQLAlchemy Support. Official Documentation. URL: https://factoryboy.readthedocs.io (дата звернення: 09.04.2026).

21. Flutter Cookbook: Testing – Mocking HTTP and External Dependencies Using Mockito for Isolated, Deterministic Unit Tests. URL: https://docs.flutter.dev/cookbook/testing/unit/mocking (дата звернення: 16.04.2026).

22. GitHub Actions – Official Documentation for Automating CI/CD Pipelines, Build, Test, and Deployment Workflows Directly in GitHub Repositories. URL: https://docs.github.com/en/actions (дата звернення: 30.04.2026).

23. Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

24. Запорожець О.І. Безпека життєдіяльності : підручник. 2-ге вид. Київ : Центр учбової літератури, 2020. 448 с.

25. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : затв. наказом Міністерства соціальної політики України від 14.02.2018 № 207.

26. ДБН В.2.5-28 : 2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

27. Гогіташвілі Г. Г., Лапін В. М. Основи охорони праці : навч. посіб. 4-те вид. випр. і доп. Київ : Знання, 2018. 302 с.

ДОДАТКИ

УДК 621.326
Козлівський В. – ст. гр. СПс-41
Тернопільський національний технічний університет імені Івана Пулюя

МОБІЛЬНА СИСТЕМА ОБЛІКУ ТА УПРАВЛІННЯ НАВЧАЛЬНИМИ ЗАНЯТТЯМИ НА БАЗІ FLUTTER

Науковий керівник: к.т.н., доцент Бачинський М. В.

Kozlivskyi V.
Ternopil Ivan Puluj National Technical University

MOBILE LESSON MANAGEMENT SYSTEM BASED ON FLUTTER

Supervisor: Ph.D., Associate Professor Bachynskyi M. V.

Ключові слова: Flutter, мобільний застосунок, управління навчальними заняттями, Django REST Framework, Firebase
Keywords: Flutter, mobile application, lesson management, Django REST Framework, Firebase

Ринок репетиторства та приватного навчання демонструє стале зростання, проте більшість доступних цифрових інструментів є або надто загальними (Google Calendar, Calendly), або надто вузькоспеціалізованими і не враховують специфіку взаємодії між викладачем та групою студентів. Серед ключових незадоволених потреб – підтримка повторюваного розкладу з механізмом виключень, рольова модель «викладач-студент», система email-запрошень до навчальних кімнат та своєчасне push-сповіщення учасників про зміни. Саме тому актуальною є розробка спеціалізованої мобільної крос-платформної системи, орієнтованої на ці задачі [1].

Предметна область управління навчальними заняттями містить ряд нетривіальних технічних викликів, які відсутні у типових планувальниках. По-перше, повторювані заняття потребують механізму виключень, що дозволяє скасовувати або переносити окремі екземпляри без зміни базового шаблону розкладу. По-друге, різниця часових зон між учасниками вимагає централізованого зберігання часу в UTC з конвертацією на клієнті. По-третє, логіка виявлення конфліктів є асиметричною, для викладача паралельне ведення двох занять неможливе, тоді як студенту достатньо попередження. По-четверте, узгодженість стану запрошень, учасників та ємності кімнати вимагає транзакційного контролю на рівні сервера.

Для вирішення зазначених задач доцільно застосовувати стек, що поєднує крос-платформну мобільну розробку з надійним серверним шаром. Flutter – фреймворк від Google, дозволяє генерувати нативний код для Android та iOS з єдиної кодової бази, забезпечуючи продуктивний UI-рендеринг через графічний рушій Impeller без залежності від платформенних компонентів. Django REST Framework надає зрілу ORM, гнучку систему дозволів і ViewSet-архітектуру для швидкого проектування REST API. Firebase забезпечує хмарну автентифікацію через ID-токени та надійну доставку push-сповіщень засобами Firebase Cloud Messaging.

Застосування Clean Architecture на клієнті дозволяє чітко розмежувати відповідальності між шарами presentation, domain та data, що суттєво спрощує тестування та подальшу підтримку коду. Патерн BLoC (Business Logic Component) забезпечує передбачуване управління станом на основі потоків подій, унеможливлюючи

неконтрольовані побічні ефекти. На сервері використання патерну Service/Selector дозволяє відокремити бізнес-логіку від читаючих запитів до бази даних, знижуючи зв'язність компонентів та спрощуючи модульне тестування кожного шару незалежно [2].

Ключовою перевагою запропонованого підходу до управління розкладом є підтримка двох типів занять – одноразових та повторюваних, зі спільним алгоритмом генерації для довільного часового діапазону. Механізм виключень через окрему сутність дозволяє гнучко коригувати окремі екземпляри повторюваного заняття, не порушуючи цілісності базового шаблону. Інтеграція підсистеми сповіщень через реактивні події сервера (Django Signals) гарантує доставку FCM push-повідомлень на всі зареєстровані пристрої учасників одразу після будь-якої зміни у розкладі, незалежно від поточного стану мобільного застосунку.

Модель даних системи організована навколо п'яти ключових сутностей. Room агрегує налаштування розкладу та місткість і виступає межею навчальної групи. Lesson зберігає або конкретну дату й час (одноразове заняття), або шаблон повторення – бітову маску днів тижня і денний час початку – разом із часовим поясом кімнати. LessonException посилається на конкретний екземпляр повторюваного заняття та перевизначає його стан (скасовано або перенесено на інший час). Invitation пов'язує кімнату з email потенційного студента та відстежує стан прийняття. UserDevice реєструє FCM-токен кожного мобільного клієнта, що забезпечує адресну доставку push-повідомлень на конкретні пристрої.

Серверний REST API задокументований засобами OpenAPI (Swagger UI), що дозволяє сторонньому клієнту або майбутній веб-версії застосунку інтегруватися без додаткових узгоджень. Кожен ендпоінт захищений Firebase ID-токеном, що валідується бібліотекою firebase-admin без звернення до стороннього сервера верифікації. Застосування крос-платформного Flutter скорочує час виходу на ринок для Android та iOS одночасно, а модульна організація Django-застосунку та чітке розмежування шарів архітектури спрощують майбутню інтеграцію з платіжними системами, платформами відеоконференцій чи аналітичними інструментами без необхідності переосмислення базової архітектури [2].

Запропонована система охоплює повний цикл взаємодії між викладачем та студентами, від реєстрації та вибору ролі до управління розкладом і отримання сповіщень. Подієво-орієнтована інфраструктура сповіщень масштабується природним чином із зростанням кількості користувачів і кімнат без змін в архітектурі. Поєднання суворого розмежування шарів, транзакційної бізнес-логіки та автоматизованого тестового покриття різних рівнів формує надійне підґрунтя для розвитку продукту, додавання нових функцій – аналітичної панелі викладача, системи оплати занять або інтеграції відео-дзвінків – можливе без ризику регресії вже реалізованої функціональності.

Список використаних джерел:

[1] Flutter Documentation. – [Електронний ресурс]. – URL: https://docs.flutter.dev (дата звернення: 06.04.2026).

[2] Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Boston: Pearson, 2017. – 432 с.

ДОДАТОК Б – Діаграма діяльності алгоритму генерації розкладу

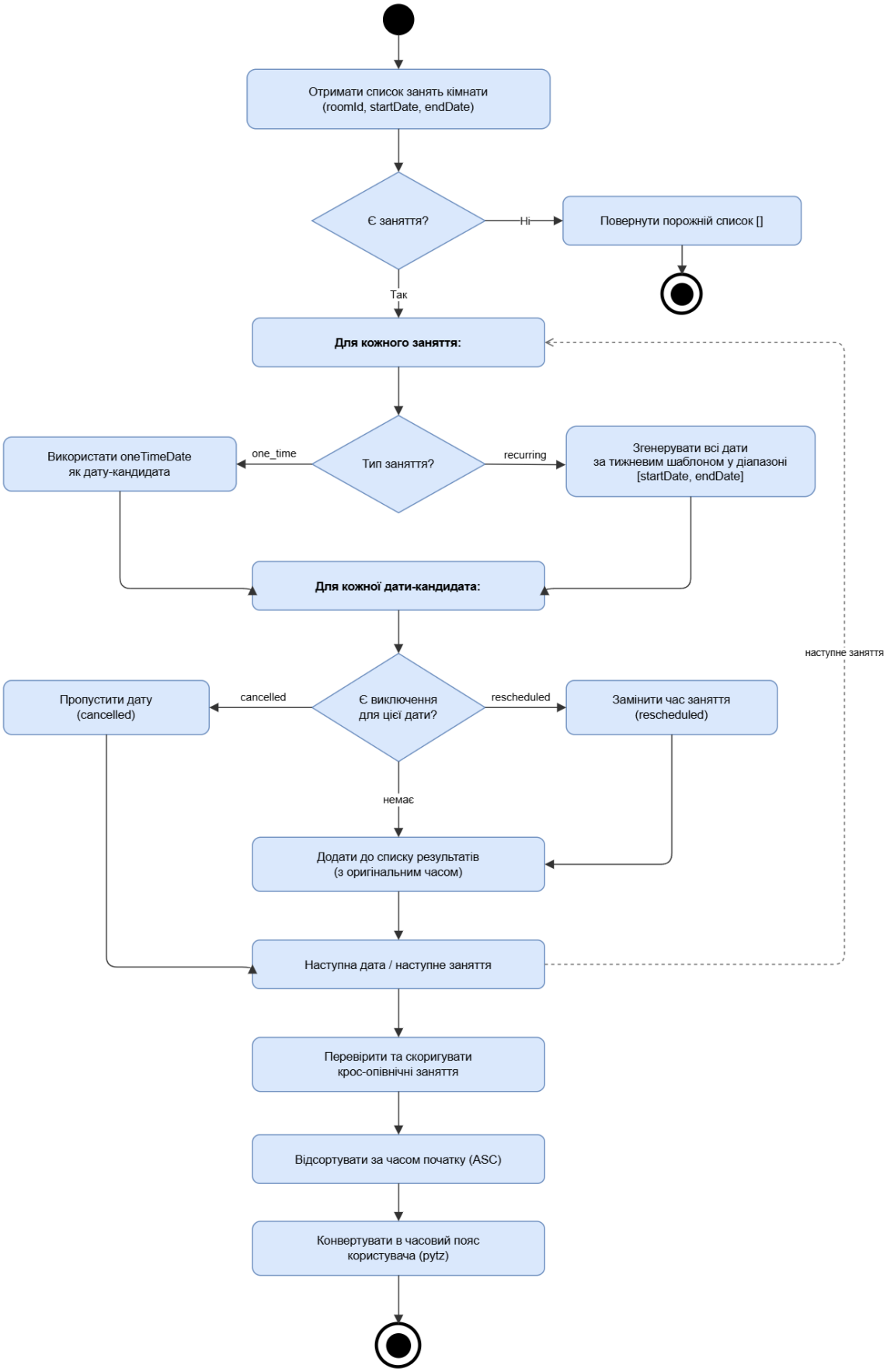

Рисунок Б.1 – Діграма діяльності алгоритму генерації навчального розкладу

Лістинг В.1 – user-authentication.yaml

```
appId: com.tutorglide.app
- launchApp:
    clearState: true
- startRecording: recording-auth
- runFlow:
    when:
      visible:
        id: "navigate_to_sign_up_button"
    commands:
      - tapOn:
          id: "navigate_to_sign_up_button"
- tapOn:
    id: "email_field"
- inputText: "test.user@example.com"

- tapOn:
    id: "password_field"
- inputText: "TestPassword123!"

- tapOn:
    id: "confirm_password_field"
- inputText: "TestPassword123!"
- tapOn:
    id: "sign_up_submit_button"
- runFlow:
    when:
      visible:
        id: "navigate_to_sign_in_button"
    commands:
      - tapOn:
          id: "navigate_to_sign_in_button"
- tapOn:
    id: "email_field"
- inputText: "test.user@example.com"

- tapOn:
    id: "password_field"
- inputText: "TestPassword123!"
- tapOn:
    id: "sign_in_submit_button"
- stopRecording
```

Лістинг В.2 – create-room-flow.yaml

```
appId: com.tutorglide.app
- launchApp:
    clearState: false
```

```
- startRecording: recording-create-room
- tapOn:
    id: "drawer_menu_button"
- waitForAnimationToEnd:
    timeout: 2000
- tapOn:
    id: "drawer_rooms_item"
- assertVisible: "Rooms"

- tapOn:
    id: "create_room_appbar_button"
- assertVisible:
    id: "room_name_field"
- tapOn:
    id: "room_name_field"
- inputText: "Mathematics Study Group"
- tapOn:
    id: "create_room_submit_button"
- assertNotVisible:
    id: "create_room_submit_button"
- stopRecording
```

Лістинг В.3 – create-lesson-flow.yaml

```
appId: com.tutorglide.app
- launchApp:
    clearState: false
- startRecording: recording-navigation
- assertVisible: "Calendar"
- tapOn:
    id: "drawer_menu_button"
- waitForAnimationToEnd:
    timeout: 2000
- tapOn:
    id: "drawer_menu_button"
- waitForAnimationToEnd:
    timeout: 2000
- tapOn:
    id: "drawer_profile_item"
- assertVisible: "Profile"
- tapOn:
    id: "drawer_menu_button"
- waitForAnimationToEnd:
    timeout: 2000
- tapOn:
    id: "drawer_settings_item"
- assertVisible: "Settings"
- tapOn:
    id: "drawer_menu_button"
- waitForAnimationToEnd:
    timeout: 2000
- assertVisible: "Calendar"
- stopRecording
```