

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
“МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ”»
ФАКУЛЬТЕТ КОМП’ЮТЕРНО-ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Колесник Олександр Сергійович

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка мобільного вебзастосунку

**«ГНУЧКЕ УПРАВЛІННЯ РОЗКЛАДОМ ЗАХОДІВ
У СУЧАСНИХ ОРГАНІЗАЦІЯХ»**

Шифр групи: ІК-9-22-Б1ПЗ(4.0д)

Спеціальність: Інженерія програмного забезпечення

Робота на здобуття освітньо-кваліфікаційного рівня бакалавр

Науковий керівник

(підпис)

професор, канд. фіз.- мат.наук

Людвиченко В.О.

Допущено до захисту ДЕК

Завідувач кафедри

(підпис)

професор, док. екон. наук

Кавун С.В.

Київ 2026

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці системи гнучкого управління розкладом заходів. Актуальність теми зумовлена необхідністю автоматизації процесів планування, організації та коригування розкладу подій у реальному часі. Сучасні інформаційні системи потребують ефективних механізмів управління подіями, які забезпечують швидке реагування на зміни, оптимізацію використання ресурсів та підвищення зручності для користувачів.

У роботі проведено аналіз предметної області та існуючих систем управління розкладом. Визначено основні функціональні вимоги до програмного забезпечення, розроблено архітектуру системи та модель бази даних. Реалізовано механізми створення, редагування, перенесення та видалення заходів, а також систему сповіщень і контролю конфліктів у розкладі.

Для розробки програмного продукту використано сучасні технології веброзробки та принципи програмної інженерії. Особливу увагу приділено адаптивності інтерфейсу, продуктивності системи та забезпеченню зручної взаємодії користувача з програмою.

Результатом роботи є створення програмної системи, яка дозволяє ефективно керувати розкладом заходів, автоматизувати процес планування та підвищити якість організації подій.

Ключові слова: управління розкладом, інформаційна система, вебзастосунок, автоматизація, програмна інженерія, база даних, планування заходів.

ABSTRACT

The qualification work is dedicated to the development of flexible event schedule management system. The relevance of the topic is determined by the need to automate the processes of planning, organizing, and adjusting event schedules in real time. Modern information systems require effective event management mechanisms that provide rapid response to changes, optimization of resource usage, and improved user convenience.

The paper analyzes the subject area and existing scheduling management systems. The main functional requirements for the software are identified, and the system architecture and database model are developed. Mechanisms for creating, editing, rescheduling, and deleting events, as well as a notification system and schedule conflict control, are implemented.

Modern web development technologies and software engineering principles were used for the development of the software product. Special attention is paid to interface adaptability, system performance, and ensuring convenient user interaction with the application.

The result of the work is a software system that enables efficient event schedule management, automates planning processes, and improves the quality of event organization.

Keywords: schedule management, information system, web application, automation, software engineering, database, event planning.

Зміст

ВСТУП.....	3
Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	
1.1 Аналіз проблеми управління розкладом заходів у сучасних організаціях.....	
1.2 Огляд аналогічних мобільних застосунків до організації розкладів.....	
1.3. Формулювання вимог до мобільного застосунку.....	
Розділ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ.....	
2.1. Функціональні і нефункціональні вимоги до застосунку.....	
2.2. Сценарії варіантів використання.....	
2.3. Архітектура застосунку (модель клієнт–сервер, компоненти, взаємодія).....	
2.4. Проєктування бази даних.....	
2.5. Проєктування інтерфейсу користувача (UX/UI).....	
2.6. Вибір технологій для розробки мобільного застосунку	
Розділ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ.....	
3.1. Опис середовища розробки.....	
3.2. Реалізація основних модулів застосунку.....	
3.3. Взаємодія з базою даних та серверною частиною.....	
3.4. Реалізація сповіщень і оновлення розкладу в реальному часі.....	
Розділ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ЗАСТОСУНКУ.....	
4.1. Методи тестування мобільних застосунків.....	
4.2. Тестування функціональних вимог.....	
4.3. Перевірка продуктивності та стабільності.....	
4.4. Аналіз результатів тестування.....	
ВИСНОВКИ.....	
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	

Додаток.....

ВСТУП

Актуальність теми. У сучасному світі цифрові технології стають невід’ємною частиною організації роботи, навчання та дозвілля. Зростання кількості заходів, зустрічей, занять та подій вимагає ефективних інструментів для планування, координації та сповіщення учасників. Особливої актуальності набувають мобільні застосунки, які забезпечують швидкий доступ до інформації, зручність використання та можливість отримувати оновлення в режимі реального часу.

Проблема управління розкладом заходів часто пов’язана з неузгодженістю часу, дублюванням подій, відсутністю централізованого доступу до оновлень і низькою гнучкістю у внесенні змін. У багатьох випадках традиційні інструменти, такі як календарі або таблиці, не забезпечують достатньої адаптивності до сучасних умов, коли графіки подій можуть змінюватися щодня.

Саме тому виникає потреба у створенні мобільного застосунку гнучкого управління розкладом заходів, який дозволить користувачам легко створювати, змінювати й переглядати події, отримувати сповіщення про оновлення, координувати ресурси та уникати конфліктів у розкладі. Мобільний формат забезпечує постійний доступ до застосунку, що особливо важливо для динамічного середовища — навчальних закладів, бізнес-команд, організаторів заходів чи фрілансерів.

Метою даної кваліфікаційної роботи є розробка програмного продукту у вигляді мобільного вебзастосунку, який реалізує функції гнучкого управління розкладом заходів у сучасних організаціях та забезпечує ефективну взаємодію між її учасниками.

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

1. Провести аналіз предметної області та виявити основні проблеми управління розкладом заходів у сучасних організаціях.
2. Проаналізувати існуючі програмні рішення та визначити їхні переваги й недоліки.
3. Сформулювати функціональні та нефункціональні вимоги до розроблюваного застосунку.
4. Розробити сценарії варіантів використання застосунку.
5. Розробити архітектуру мобільного застосунку та базу даних.
6. Реалізувати основні модулі застосунку з використанням сучасних технологій мобільної розробки.
7. Провести тестування та оцінку якості розробленого продукту.

Об'єктом дослідження є процес організації, планування та координації заходів у межах сучасних інформаційних систем.

Предметом дослідження є методи та засоби реалізації мобільних застосунків для управління розкладом заходів у сучасних організаціях із гнучким налаштуванням користувацьких параметрів.

У роботі використано сучасні методи проєктування програмних систем, принципи об'єктно-орієнтованого програмування, а також технології створення мобільних застосунків на базі кросплатформних фреймворків.

Практична значущість розробки полягає у створенні мобільного застосунку, який може бути використаний у навчальних закладах, бізнес-компаніях, організаціях або для особистих потреб з метою ефективного планування часу та координації дій учасників подій.

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатку.

У першому розділі («АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ») розглядаються: аналіз проблеми управління розкладом заходів у сучасних організаціях (1.1); огляд аналогічних мобільних

застосунків до організації розкладів (1.2); формулювання вимог до мобільного застосунку (1.3);

У другому розділі («ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ») проводиться проектування програмного застосунку:

функціональні і нефункціональні вимоги до застосунку (2.1); розроблені сценарії варіантів використання (2.2); архітектури застосунку (модель клієнт–сервер, компоненти, взаємодія (2.3); бази даних (2.4); інтерфейсу користувача (UX/UI) (2.5); технологій для розробки мобільного застосунку (2.6).

Третій розділ («РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ») присвячений безпосередньо розробці програмного продукту: опису середовища розробки (3.1); реалізації основних модулів системи (3.2); взаємодії з базою даних та серверною частиною (3.3); реалізації сповіщень і оновлення розкладу в реальному часі (3.4);

Четвертий розділ («ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ЗАСТОСУНКУ») присвячений тестуванню та оцінці якості програмного продукту: методам тестування мобільних застосунків (4.1); тестуванню функціональних вимог (4.2); перевірці продуктивності та стабільності (4.3); аналізу результатів тестування (4.4);

У висновках підсумовуються результати дослідження.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз проблеми управління розкладом заходів у сучасних організаціях

У сучасних організаціях, навчальних закладах та робочих колективах кількість подій, зустрічей та завдань постійно зростає. Користувачам необхідно координувати роботу з великою кількістю заходів, що відбуваються паралельно, часто змінюються або залежать від зовнішніх факторів. В таких умовах традиційні засоби планування — паперові розклади, таблиці чи стандартні календарі — перестають бути ефективними, оскільки вони не забезпечують зручного доступу, швидкого оновлення та синхронізації між усіма учасниками.

Одна з основних проблем полягає в тому, що більшість інструментів для планування подій недостатньо гнучкі. Вони часто не дозволяють швидко вносити зміни в розклад, автоматично повідомляти користувачів про коригування або адаптувати відображення подій під індивідуальні потреби. Це створює ситуації, коли інформація про зміни не доходить до всіх учасників, викликаючи плутанину, запізнення та дублювання завдань.

Також актуальною є проблема централізації даних. У багатьох організаціях розклад формується різними відповідальними особами, що призводить до відсутності єдиного джерела актуальної інформації. У результаті користувачі змушені постійно уточнювати деталі, використовувати сторонні ресурси або створювати власні списки подій, які не синхронізуються між собою.

Ще одним важливим аспектом є зростання мобільності користувачів. Більшість людей користується смартфонами як основним засобом доступу до інформації, а тому потребує інструментів, які дозволяють переглядати

розклад, отримувати сповіщення та вносити зміни незалежно від місця перебування. Системи, які не забезпечують мобільний доступ або не оптимізовані для використання на смартфонах, значно знижують ефективність організації процесів.

Крім того, сучасні користувачі очікують від систем планування можливості персоналізувати розклад: фільтрувати події за категоріями, додавати власні нагадування, налаштовувати інтерфейс відображення. Більшість існуючих рішень пропонують лише базовий функціонал, що не дозволяє сформувати індивідуальний підхід до роботи з подіями.

Таким чином, проблема управління розкладом подій полягає у відсутності універсального та адаптивного інструменту, який забезпечував би швидке оновлення інформації, централізацію даних, просту комунікацію між учасниками та можливість персоналізації. Саме ці потреби визначають важливість створення мобільного вебзастосунку, що надає сучасні механізми гнучкого управління розкладом заходів.

1.2. Огляд аналогічних мобільних застосунків до організації розкладів

Для ефективного планування часу користувачі широко застосовують різні мобільні застосунки, які дозволяють організовувати події, завдання та зустрічі. Найбільш популярними серед них є Google Calendar, Microsoft Outlook Calendar, Calendly, Eventbrite, Notion Calendar та інші. Кожен із цих сервісів містить власний набір можливостей, однак не всі вони повністю відповідають сучасним вимогам гнучкого управління розкладом.

Google Calendar [1] є одним із найпоширеніших рішень для організації розкладів. Він забезпечує синхронізацію між різними пристроями, можливість створення подій, надсилання запрошень, використання нагадувань та оновлення в реальному часі. Його перевагою є інтеграція з іншими сервісами Google, проте система має обмежені можливості персоналізації та не дозволяє гнучко адаптувати інтерфейс під індивідуальні потреби. Крім того, додаток недостатньо орієнтований на групову роботу, коли великій кількості користувачів потрібен спільний адаптивний розклад.

Microsoft Outlook Calendar [2] зазвичай використовується у корпоративному середовищі. Він зручний для планування нарад, спільної роботи та узгодження графіків у бізнес-командах. Інструмент тісно інтегрований із корпоративними службами Microsoft, що робить його ефективним у великих організаціях. Проте він менш зручний у повсякденному використанні, вимагає корпоративного облікового запису для повного функціоналу та має складніший інтерфейс для пересічного користувача.

Calendly [3] є сервісом, створеним для автоматичного узгодження часу зустрічей. Його ключова перевага — можливість уникати дублювання подій завдяки автоматичному вибору вільних часових слотів. Однак застосунок орієнтований переважно на бізнес-комунікацію та заплановані

онлайн-зустрічі, а не на комплексне управління особистим чи груповим розкладом.

Eventbrite [4] використовується переважно для організації публічних заходів — конференцій, концертів, семінарів. Він дозволяє створювати події, продавати квитки та поширювати інформацію серед учасників. Проте цей сервіс не підходить для щоденного планування, не підтримує гнучкого модифікування розкладу та орієнтований на масштабні події, а не на внутрішні чи особисті задачі.

Notion Calendar (раніше Cron) [5] пропонує сучасний інтерфейс та можливість інтегрувати задачі та події в єдиний робочий простір. Він зручний для особистої організації часу та забезпечує мінімалістичне відображення подій. Проте система не пропонує повноцінної адаптивності для командної роботи та не має продуманих механізмів сповіщення у складних розкладах з частими змінами.

Загалом огляд існуючих мобільних рішень показує, що більшість із них спрямована на базове планування або вирішує лише окремі аспекти проблеми — наприклад, планування зустрічей, ведення особистого календаря чи організацію великих подій.

У той же час сучасним користувачам потрібен інструмент, який поєднує персоналізацію, мобільність, простоту внесення змін та можливість працювати зі складними розкладами у груповому або індивідуальному форматі. Саме відсутність такого комплексного рішення визначає актуальність розробки мобільного застосунку, орієнтованого на гнучке управління розкладом заходів.

1.3. Формулювання вимог до мобільного застосунку

Для забезпечення ефективного управління розкладом заходів необхідно визначити чіткі вимоги до майбутнього мобільного застосунку. Ці вимоги формують основу архітектури системи, визначають межі функціональності та забезпечують відповідність програмного продукту потребам користувачів. Мобільний застосунок має бути орієнтований на зручність, гнучкість та можливість адаптації під різні сценарії застосування — від особистого планування до використання в організаціях, які регулярно проводять події.

Однією з ключових вимог є підтримка гнучкого створення та редагування подій. Користувач повинен мати можливість швидко додавати новий захід, змінювати його параметри, переносити в часі, об'єднувати кілька подій у блоки або створювати серії повторюваних активностей. Важливо, щоб зміни вносилися без ускладнень, а система автоматично підлаштовувала відображення графіку.

Не менш важливою складовою є персоналізація інтерфейсу та функціоналу. Застосунок повинен дозволяти користувачеві налаштовувати категорії заходів, кольорові схеми, типи нагадувань та способи відображення розкладу (список, календар, таймлайн). Це дозволить адаптувати систему під індивідуальні потреби — студентів, організаторів заходів, менеджерів команд чи звичайних користувачів.

Обов'язковою вимогою є підтримка механізму сповіщень і нагадувань, що забезпечує своєчасне інформування про заплановані події, зміни в розкладі або нові завдання. Сповіщення мають бути гнучкими: користувач сам обирає час нагадування, формат повідомлення та можливість повторного нагадування.

Застосунок повинен мати функції офлайн-доступу, які дають змогу працювати з розкладом навіть без активного інтернет-з'єднання. Дані мають

синхронізуватися автоматично після відновлення доступу до мережі, що забезпечує безперервність роботи й актуальність усієї інформації.

Також важливо передбачити інтеграції з іншими сервісами — календарями, push-сервісами, системами запрошень або внутрішніми корпоративними платформами. Така можливість суттєво спростить обмін інформацією та зробить розклад частиною загальної цифрової екосистеми користувача.

Окрему групу становлять вимоги до зручності використання та інтерфейсу: інтуїтивна навігація, зрозуміла структура меню, мінімальна кількість дій для виконання основних операцій, читабельність елементів на різних діагоналях екранів. Дизайн має відповідати принципам сучасних мобільних UI-платформ, щоб користувач легко орієнтувався у функціях.

До системних вимог входить стабільна робота застосунку, оптимізація витрат батареї, підтримка актуальних версій Android/iOS, коректна обробка фонових процесів, захист даних користувачів та обмеження доступу до конфіденційної інформації.

У сукупності ці вимоги формують основу майбутньої програмної системи і визначають, яким має бути мобільний вебзастосунок, щоб повністю вирішувати поставлену задачу проєкту — забезпечувати просте, гнучке та персоналізоване управління розкладом заходів для широкого кола користувачів.

Розділ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1. Функціональні і нефункціональні вимоги до застосунку

Створення мобільного застосунку для гнучкого управління розкладом заходів потребує чіткого визначення вимог, які забезпечують коректну роботу системи, її зручність для користувачів та відповідність поставленій задачі. Вимоги визначають функції, які повинні бути реалізовані, а також характеристики, яким має відповідати програмний продукт. Їх формування дозволяє структурувати процес розробки, спростити тестування та забезпечити узгодженість між очікуваннями користувача і кінцевим результатом.

2.1.1. Функціональні вимоги

Функціональні вимоги описують конкретні можливості та дії, які користувач може виконувати в межах системи. Для мобільного застосунку управління розкладом вони включають:

1. Керування подіями:

- Створення нових подій із зазначенням назви, опису, дати, часу, місця та категорії.
- Редагування існуючих подій: зміна параметрів, перенесення на іншу дату або час.
- Видалення подій із підтвердженням дії.
- Додавання повторюваних подій із вибором інтервалу (щодня, щотижня, щомісяця).

2. Гнучка організація розкладу:

- Перегляд розкладу у форматах «день», «тиждень», «місяць».
- Перетягування подій між часовими слотами (drag & drop).

- Фільтрація подій за категоріями, важливістю, типами або персональними тегами.
- Групування завдань за темами, проектами або напрямками діяльності.

3. Нагадування та сповіщення:

- Налаштування сповіщень для кожної події: за 5 хвилин, 30 хвилин, 1 годину, 1 день тощо.
- Отримання push-сповіщень під час зміни події або її перенесення.
- Можливість тимчасово вимкнути або відкласти нагадування.

4. Персоналізація інтерфейсу:

- Вибір теми оформлення (світла, темна або автоматична).
- Налаштування кольорів для різних типів подій.
- Встановлення власних категорій і міток.

5. Робота в офлайн-режимі:

- Можливість переглядати розклад без підключення до інтернету.
- Збереження змін локально з подальшою синхронізацією після підключення.
- Повідомлення користувача про успішну синхронізацію.

6. Синхронізація даних:

- Зберігання інформації в хмарній базі даних.
- Авторизація через обліковий запис.
- Синхронізація між різними пристроями одного користувача.

7. Додаткові можливості:

- Пошук подій за ключовим словом.
 - Створення списків завдань, пов'язаних із подіями.
 - Можливість додавання вкладень (фото, документи, посилання).
 - Інтеграція з системним календарем пристрою (опціонально).
-

2.1.2. Нефункціональні вимоги

Нефункціональні вимоги визначають обмеження, характеристики та якість роботи додатку. Вони забезпечують комфорт використання та надійну роботу системи.

1. Продуктивність:

- Час відкриття основного екрану додатку не повинен перевищувати 2 секунд.
- Операції створення чи редагування подій мають виконуватися миттєво без затримок.
- Синхронізація даних повинна виконуватися у фоновому режимі.

2. Надійність:

- Додаток повинен зберігати дані навіть у разі несподіваного завершення роботи.
- Усі зміни мають фіксуватися у локальному кеші до моменту синхронізації.
- Система має коректно працювати за відсутності стабільного інтернету.

3. Зручність та простота використання:

- Інтерфейс повинен бути інтуїтивним, без перевантаження елементами.
- Події мають створюватися в мінімальну кількість кроків.
- Всі основні функції повинні бути доступні з головного екрану.

4. Масштабованість:

- Система повинна підтримувати збільшення кількості подій без зниження продуктивності.
- Архітектура повинна дозволяти розширення функціональності у майбутньому.

5. Сумісність:

- Додаток повинен коректно працювати на Android та iOS.

- Інтерфейс має адаптуватися під мобільні пристрої з різними розмірами екранів.

6. Захист даних:

- Авторизація має виконуватися через захищені механізми.
- Дані користувача повинні передаватися з використанням шифрування.
- Особисті події не можуть бути доступними стороннім особам.

2.2. Сценарії варіантів використання (Use Cases)

Сценарії використання описують типові ситуації, у яких користувач взаємодіє з мобільним додатком. Вони демонструють, як саме функціональність системи застосовується на практиці та в яких умовах користувачі отримують користь від роботи застосунку. Наявність чітко сформульованих сценаріїв варіантів використання (use case) дозволяє структурувати логіку роботи системи та визначити основні вимоги до інтерфейсу й поведінки застосунку.

Сценарій 1. Створення нової події

Актор: Користувач

Передумови: Застосунок відкритий, користувач авторизований Основний сценарій:

1. Користувач переходить на головний екран із розкладом.
2. Натискає кнопку «Додати подію».
3. Заповнює дані: назву, дату, час, категорію, місце проведення та опис.
4. Обирає нагадування (за потреби).
5. Підтверджує створення події.
6. Подія з'являється у відповідному часі в календарі.

Результат: Подію додано і вона включена до загального розкладу.

Сценарій 2. Редагування існуючої події

Актор: Користувач

Передумови: Подія вже існує в розкладі

Основний сценарій:

1. Користувач натискає на подію в календарі.
2. Обирає «Редагувати».
3. Вносить зміни: час, дату, опис або категорію.
4. Зберігає зміни.
5. Подія оновлюється в розкладі.

Результат: Розклад автоматично адаптується до нових параметрів події.

Сценарій 3. Видалення події

Актор: Користувач

Передумови: Подія повинна бути більше не актуальною

Основний сценарій:

1. Користувач відкриває інформацію про подію.
2. Натискає «Видалити подію».
3. Підтверджує видалення.
4. Подія зникає з розкладу.

Результат: Подія повністю видалена з локальних та хмарних даних.

Сценарій 4. Перегляд розкладу за різними інтервалами

Актор: Користувач

Передумови: У календарі є події

Основний сценарій:

1. Користувач обирає формат перегляду: день, тиждень або місяць.
2. Додаток оновлює відображення відповідно до вибору.
3. Користувач гортанням переглядає наступні чи попередні періоди.

Результат: Користувач отримує повну картину свого розкладу.

Сценарій 5. Налаштування нагадування про подію

Актор: Користувач

Основний сценарій:

1. Користувач відкриває подію.
2. Обирає параметр «Нагадування».
3. Встановлює час: за кілька хвилин, годину, день або користувацький інтервал.
4. Зберігає налаштування.
5. У встановлений час користувач отримує push-сповіщення.

Результат: Користувач отримує завчасне нагадування і не пропускає подію.

Сценарій 6. Персоналізація інтерфейсу

Актор: Користувач

Основний сценарій:

1. Користувач переходить у налаштування.
2. Обирає тему оформлення (світла/темна/авто).
3. Налаштовує кольори категорій подій.
4. Створює власні теги або групи.
5. Зберігає налаштування.

Результат: Інтерфейс адаптується до індивідуальних уподобань користувача.

Сценарій 7. Перенесення події перетягуванням (Drag & Drop)

Актор: Користувач

Основний сценарій:

1. Користувач перебуває у перегляді «день» або «тиждень».
2. Натискає на подію та утримує її.
3. Перетягує подію на інший часовий слот.
4. Відпускає подію.
5. Застосунок автоматично оновлює її дату та час.

Результат: Подія швидко переноситься без відкриття додаткових меню.

Сценарій 8. Робота в офлайн-режимі

Актор: Користувач

Передумови: Немає доступу до інтернету

Основний сценарій:

1. Користувач відкриває застосунок.
2. Переглядає збережений розклад.
3. Створює або редагує події.
4. Дані тимчасово зберігаються локально.
5. Після появи інтернету застосунок автоматично синхронізує зміни.

Результат: Користувач не обмежений у роботі без мережі.

Сценарій 9. Синхронізація між пристроями

Актор: Користувач

Основний сценарій:

1. Користувач входить у свій аккаунт на іншому пристрої.
2. Застосунок отримує всі події з хмарного сховища.
3. Користувач переглядає свій актуальний розклад.

Результат: Однакові дані на всіх пристроях.

Сценарій 10. Пошук подій

Актор: Користувач

Основний сценарій:

1. Користувач натискає рядок пошуку.
2. Вводить частину назви або ключове слово.
3. Застосунок показує список подій, що відповідають запиту.

Результат: Швидке знаходження потрібної інформації без гортання календаря.

Сценарій 11. Керування категоріями та тегами

Актор: Користувач

Основний сценарій:

1. Користувач відкриває модуль керування категоріями.
2. Створює нову категорію або редагує існуючу.
3. Призначає її подіям.

4. Використовує фільтрацію за категоріями.

Результат: Розклад стає впорядкованішим та структурованим.

2.3. Архітектура застосунку (модель клієнт–сервер, компоненти, взаємодія)

Архітектура мобільного вебзастосунку базується на клієнт–серверній моделі, яка забезпечує розподіл відповідальності між мобільною частиною (клієнтом) та серверною інфраструктурою. Такий підхід дозволяє досягти високої гнучкості, масштабованості та надійності системи, забезпечуючи стабільну роботу незалежно від кількості подій, користувачів або складності розкладів. Архітектура побудована таким чином, щоби користувач міг швидко працювати з розкладом, а обробка й зберігання даних здійснювалися централізовано, з можливістю синхронізації на будь-якому пристрої.

Мобільний застосунок виконує роль клієнта, який відповідає за відображення інтерфейсу, обробку взаємодії з користувачем, локальне кешування даних та виконання частини логіки, яка не потребує доступу до сервера. Усі ключові зміни, такі як додавання, редагування або видалення подій, надсилаються на сервер для збереження в централізованій базі даних. Завдяки цьому забезпечується узгодженість даних між різними пристроями одного користувача та можливість швидкого доступу до актуальної інформації.

Серверна частина виступає ядром системи, яке обробляє запити клієнта, виконує бізнес-логіку, керує механізмами авторизації та автентифікації, обробляє операції з календарем та забезпечує зберігання даних у хмарній інфраструктурі. Використання хмарного середовища також дозволяє масштабувати систему в разі збільшення кількості користувачів або потреби у додаткових обчислювальних ресурсах. Сервер реалізує REST API або GraphQL (залежно від обраної технології), що дозволяє клієнтському застосунку працювати за єдиним стандартом взаємодії.

Важливою частиною архітектури є модуль синхронізації. Оскільки додаток підтримує роботу без підключення до інтернету, зміни, які користувач виконує в офлайн-режимі, тимчасово зберігаються у локально встановленому сховищі. Після появи з'єднання мобільний клієнт надсилає накопичені зміни на сервер, де вони обробляються у відповідності до існуючого стану даних. Таким чином, забезпечується узгодженість графіку без втрати інформації.

Усередині клієнтської частини архітектура побудована за принципом розділення відповідальності між логікою UI, менеджерами стану та модулями роботи із збереженими даними. Це дозволяє забезпечити стабільність інтерфейсу та простоту його розширення. Наприклад, модуль, який відповідає за відображення календаря, працює окремо від модулів сповіщень або налаштувань, а всі вони звертаються до єдиного джерела даних, що зменшує кількість помилок та підвищує передбачуваність поведінки системи.

Архітектура застосунку включає також механізм роботи зі сповіщеннями. Локальні сповіщення генеруються на пристрої користувача відповідно до запланованих подій, що забезпечує їхню доставку навіть за відсутності інтернету. Сервер може використовувати віддалені push-повідомлення у випадках, коли потрібно оповістити користувача про зміни, виконані з іншого пристрою, або про актуальні оновлення розкладу, синхронізовані через хмару.

База даних, що використовується на сервері, призначена для зберігання структурованих даних про події, користувачів, категорії, нагадування та налаштування. Вона забезпечує можливість швидкого доступу до інформації, підтримує індексацію для пришвидшення пошуку та масштабування для обслуговування великої кількості запитів. Структура даних оптимізована таким чином, щоб забезпечити високу швидкість роботи застосунку навіть за умов активного навантаження.

Загальна взаємодія між компонентами виглядає так: користувач виконує дію в мобільному застосунку → клієнт формує запит → сервер обробляє його → база даних оновлює інформацію → сервер надсилає відповідь назад → клієнт оновлює інтерфейс і локальне сховище. Така модель дозволяє забезпечити стабільність, передбачуваність та високу швидкість реакції системи.

Завдяки продуманій архітектурі мобільний застосунок здатний обробляти складні сценарії: перенесення подій у реальному часі, роботу з великою кількістю нагадувань, персоналізацію інтерфейсу, синхронізацію з різними календарями та стабільну роботу за обмеженого доступу до інтернету. Це робить систему гнучкою, масштабованою та зручною для широкого кола користувачів.

2.4. Проєктування бази даних

Проєктування бази даних є однією з ключових складових створення мобільного застосунку, оскільки саме структура даних визначає ефективність роботи системи, швидкість доступу до інформації, масштабованість та можливість подальшого розвитку функціоналу. База даних має забезпечувати надійне зберігання інформації про події, користувачів, їх взаємодії та налаштування, а також підтримувати гнучкі механізми обробки даних, необхідні для реалізації персоналізованих розкладів.

У процесі проєктування бази даних важливо визначити основні сутності, їх властивості та типи взаємозв'язків. Оскільки система працює з різними типами інформації — статичними атрибутами подій, динамічними змінами, налаштуваннями користувачів та їхніми діями, — структура повинна бути логічно цілісною, гнучкою та здатною підтримувати майбутні розширення. Для цього застосовується реляційна модель, яка дозволяє створити структуровані таблиці з чітко визначеними ключами та зв'язками.

Однією з основних сутностей є користувач (User). Таблиця користувачів зберігає базові дані, такі як ім'я, електронну пошту, тип доступу, а також технічну інформацію для авторизації. Додатково передбачено поля для індивідуальних налаштувань, що дозволяють застосунку адаптувати розклад під конкретні потреби користувача. Це важливо для гнучкої персоналізації, що є однією з ключових переваг розроблюваного застосунку.

Другий фундаментальний елемент — події (Events). Кожна подія має назву, опис, дату проведення, часові межі, місце, тип (зустріч, лекція, тренінг тощо) та організатора. База має підтримувати як одноразові, так і повторювані події. Для повторюваних заходів важливо передбачити механізм зберігання правил повторення: періодичність, винятки, тривалість циклу. Це

розширює можливості планування й дозволяє застосунку автоматично формувати довготривалі розклади без дублювання інформації.

Наступною сутністю є категорії подій (Categories), які забезпечують логічне групування заходів. Категоризація спрощує взаємодію користувачів із системою, даючи змогу швидко фільтрувати події за тематикою чи типом. Особливо це актуально в організаціях, де розклад містить десятки напрямів діяльності.

Для забезпечення персоналізації вводиться таблиця користувацьких налаштувань (UserSettings), де зберігаються параметри відображення, пріоритети подій, обраний інтерфейс, часовий формат, система нагадувань та інші опції. Цей компонент дозволяє реалізувати адаптивний інтерфейс, у якому користувач бачить саме ті елементи, які йому потрібні, а також отримує сповіщення відповідно до власних уподобань.

Важливою частиною системи є нагадування (Notifications). Таблиця нагадувань пов'язана з подіями та користувачами і містить інформацію про час сповіщення, тип тригера (до початку події, щоденне повторення, змінений час початку тощо) та статус виконання. Такий підхід дає змогу оптимізувати розсилання сповіщень і гарантувати, що користувачі отримують актуальну інформацію.

Також у структурі передбачається таблиця логів (ActivityLog), яка фіксує дії користувачів: створення, редагування, видалення подій, зміни налаштувань, входи в систему. Це забезпечує прозорість роботи застосунку, дозволяє аналізувати поведінку користувачів та підвищує інформаційну безпеку.

Окремо слід зазначити сутність локацій (Locations). Вона містить інформацію про приміщення, адреси, номери аудиторій або зон, де проводяться заходи. Завдяки зв'язку подій із конкретними локаціями система

може запобігати конфліктам розкладу, коли одночасно планується кілька подій в одному місці.

Загальна структура бази даних включає зв'язки типу *«один-до-багатьох»*, *«багато-до-багатьох»* та *«один-до-одного»*. Наприклад, один користувач може мати багато подій, але одна подія може належати кільком користувачам — тому використовується проміжна таблиця для спільного доступу. Така організація дозволяє забезпечити багатокористувацький режим і реалізувати групові календарі.

Для оптимізації роботи застосовуються індекси, каскадні обмеження, зовнішні ключі та механізми забезпечення цілісності даних. Це гарантує коректність зберігання інформації та мінімізує ризик помилок при одночасній роботі великої кількості користувачів.

Таким чином, проєктування бази даних мобільного застосунку спрямоване на забезпечення надійного, адаптивного й масштабованого середовища для зберігання даних, що дозволяє системі ефективно виконувати функції гнучкого управління розкладом заходів та залишатися відкритою для майбутніх удосконалень.

2.5. Проєктування інтерфейсу користувача (UX/UI)

Проєктування інтерфейсу користувача є одним із ключових етапів створення мобільного застосунку, оскільки саме інтерфейс визначає зручність, зрозумілість і ефективність взаємодії користувача з функціоналом системи. Якісне UX/UI-проєктування дозволяє зменшити час на виконання повсякденних дій, підвищити залученість користувачів та забезпечити приємний досвід роботи із застосунком, що особливо важливо для систем, орієнтованих на регулярне використання.

У центрі проєктування інтерфейсу лежить принцип користувацької орієнтованості, тобто створення дизайну, який відповідає потребам різних категорій користувачів — студентів, працівників організацій, адміністраторів, викладачів та інших осіб, що працюють з розкладом подій. Кожен із цих типів користувачів має власний сценарій роботи, тому інтерфейс повинен забезпечувати інтуїтивну навігацію та можливість швидко виконувати ключові завдання: перегляд подій, редагування, сортування, налаштування нагадувань та створення нових заходів.

Однією з основних цілей UX-проєктування є мінімізація кількості дій, необхідних для виконання однієї операції. Для цього передбачається використання плаваючих кнопок швидкого доступу, контекстних меню, жестів та зрозумілої структури екранів. Інтерфейс має бути побудований таким чином, щоб навіть користувач без досвіду міг легко розібратися з функціями застосунку без додаткових інструкцій.

Важливим аспектом є створення логічної навігаційної моделі. У застосунку передбачається нижнє меню або бічна панель, яка забезпечує швидкий доступ до основних розділів: «Головна», «Календар», «Мої події», «Нагадування», «Налаштування». Така структура дозволяє уникнути

перевантаженості екранів і дозволяє користувачам легко переміщатися між розділами.

Відображення розкладу є ключовим елементом інтерфейсу, тому під час проєктування приділяється увага гнучкості та варіативності. Застосунок повинен підтримувати кілька режимів перегляду: денний, тижневий, місячний та компактний список подій. Для зручності користувача передбачається можливість швидкого перемикання між режимами та застосування різноманітних фільтрів: за категоріями, місцем проведення, типом подій чи статусом участі.

Окрему роль відіграє візуальна ієрархія, яка забезпечує чіткий поділ важливих та допоміжних елементів. Заголовки, основні кнопки дій, час подій та місця проведення виділяються за допомогою акцентного кольору або розміру шрифту, тоді як додаткові опції розміщуються у згрупованих блоках. Це сприяє зменшенню інформаційного навантаження і допомагає користувачеві швидко сприймати ключову інформацію.

У процесі UI-проєктування також враховуються особливості різних розмірів екранів та платформ. Інтерфейс будується за принципами адаптивності, щоб забезпечити коректне відображення на смартфонах як з невеликою, так і з великою діагоналлю. Усі елементи повинні залишатися чіткими, натискання не повинні вимагати надмірної точності, а текст — бути читабельним без масштабування.

Не менш важливою частиною інтерфейсу є екрани створення й редагування подій. Вони повинні бути максимально простими, структурованими та логічними: користувач має швидко вибрати дату, час, категорію, місце, формат події, а також налаштувати нагадування. Для цього використовуються випадаючі списки, календарі, перемикачі та інші інтуїтивно зрозумілі компоненти.

Дизайн враховує також кольорову систему, яка допомагає користувачу швидко орієнтуватися в подіях. Різні категорії можуть бути позначені окремими кольорами, що додатково покращує сприйняття розкладу. При цьому важливо зберігати візуальну стриманість, щоб уникнути перевантаження інтерфейсу.

Особлива увага приділяється доступності. Інтерфейс повинен підтримувати можливість збільшення шрифту, коректну роботу з VoiceOver/TalkBack, контрастну тему, а також інтуїтивні клавішні та жестові взаємодії. Це важливо для забезпечення широкого доступу до застосунку, включно з користувачами, які мають певні фізичні обмеження.

Таким чином, UX/UI-проектування мобільного додатку спрямоване на створення зручного, адаптивного та інтуїтивного інтерфейсу, який дозволяє користувачам швидко виконувати необхідні дії, ефективно управляти розкладом і отримувати позитивний досвід від взаємодії із системою. Завдяки продуманій структурі, зрозумілій навігації та гнучкому відображенню інформації застосунок забезпечує високу зручність та ефективність роботи незалежно від частоти та інтенсивності використання.

2.6. Вибір технологій для розробки мобільного застосунку

Вибір технологій для розробки мобільного застосунку є одним із ключових етапів проєктування, оскільки від цього залежить ефективність розробки, продуктивність системи, її масштабованість та подальша підтримка. Правильно підібраний технологічний стек дозволяє створити стабільний, зручний, багатофункціональний та сучасний продукт, який легко адаптується під потреби користувачів та може розвиватися в майбутньому.

Основною вимогою до технологій є їхня здатність забезпечити:

- кросплатформенність;
- швидку розробку;
- зручну інтеграцію зі сторонніми сервісами;
- стабільну роботу офлайн
- простоту розгортання;
- високу продуктивність і гнучкість.

Вибір Flutter як фреймворку інтерфейсу. Для розробки клієнтської частини застосунку доцільно використовувати Flutter — сучасний кросплатформенний фреймворк від Google. Flutter дозволяє створювати мобільні застосунки для Android та iOS з єдиною кодовою базою, що значно знижує трудовитрати та скорочує час розробки.

Ключові переваги Flutter:

- висока продуктивність завдяки власному рендеринг-движку;
- стабільна «гаряча» перезагрузка, яка пришвидшує розробку;
- велика екосистема пакетів;
- можливість створення анімацій і сучасного UI без втрати швидкодії;
- зручність масштабування проєкту та підтримки коду.

Використання Flutter також забезпечує єдине UX/UI-проєктування для обох платформ, що дозволяє зберегти цілісність дизайну.

Вибір Firebase як бекенд-платформи. Для реалізації серверної частини можна використовувати Firebase — набір мобільних сервісів від Google, який надає готову інфраструктуру для швидкого створення backend-функціоналу.

Переваги Firebase для даного проєкту:

- Firebase Authentication — проста реалізація реєстрації через email, телефон або соціальні мережі;
- Cloud Firestore / Realtime Database — можливість зручного зберігання даних про події, категорії, нагадування та профілі користувачів;
- автоматична синхронізація між пристроями;
- підтримка офлайн-режиму;
- Cloud Functions — можливість реалізувати серверну логіку без розгортання власного сервера;
- вбудовані інструменти аналітики, пуш-повідомлень та безпеки.

Firebase добре підходить для додатків, де важлива швидкість розробки, масштабованість та хмарне зберігання.

Використання REST API для інтеграцій. Для взаємодії мобільного застосунку з зовнішніми сервісами (наприклад: Google Calendar, Microsoft Outlook, внутрішні системи організацій) доцільно застосовувати REST API.

REST API забезпечує:

- стандартизований формат обміну даними;
- сумісність з будь-якими вебсерверами;
- просту інтеграцію з calendar-системами, картографічними сервісами (Google Maps), сервісами локацій чи хмарним зберіганням;
- можливість розширення функціоналу без зміни клієнтського коду.

REST API також може використовуватись у випадку, якщо в майбутньому буде розроблено власний бекенд на Node.js, Django, Spring Boot чи іншій платформі.

SQLite як локальне сховище. Незважаючи на використання хмарних сервісів, застосунок повинен коректно працювати офлайн. Для цього застосовується SQLite — легка вбудована база даних, яка зберігає дані локально на пристрої.

Основні переваги SQLite:

- швидка робота на мобільних пристроях;
- відсутність необхідності у серверному підключенні;
- можливість зберігати локальні копії розкладу;
- швидкий доступ до даних без інтернету;
- підтримка складних структур даних та зв'язків.

SQLite використовується для кешування подій, нагадувань, відображення розкладів і локального зберігання налаштувань користувача.

Інші допоміжні технології. Для повноцінної реалізації проєкту можуть використовуватися також такі інструменти:

- Git та GitHub/GitLab — для контролю версій;
- Figma — для створення UX/UI макетів;
- Dart — основна мова програмування (використовується у Flutter);
- Firebase Cloud Messaging — для пуш-повідомлень;
- Google Maps API — для інтеграції локацій у подіях;
- Provider або Riverpod — для управління станом у Flutter;
- CI/CD-платформи (GitHub Actions, Codemagic) — для автоматичної збірки та тестування додатку.

Підсумок:

Обраний технологічний стек — Flutter + Firebase + SQLite + REST API — забезпечує оптимальне поєднання швидкості розробки, продуктивності, гнучкості та масштабованості. Завдяки цим технологіям мобільний застосунок може стабільно працювати як онлайн, так і офлайн, підтримувати

синхронізацію між пристроями, забезпечувати безпечний доступ користувачів та легко розширюватися в майбутньому.

Розділ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1. Опис середовища розробки

Середовище розробки мобільного застосуку відіграє ключову роль у забезпеченні ефективності, стабільності та продуктивності процесу створення програмного продукту. Вибір конкретних інструментів, платформ та технологій визначає можливості команди розробників, швидкість реалізації функціоналу, зручність тестування та масштабуємість додатку в майбутньому. У цьому підпункті розглянуто основні компоненти середовища розробки, що використовуються для створення програмної системи.

Основним програмним середовищем для реалізації мобільного клієнтського інтерфейсу є Flutter, який працює поверх мови програмування Dart. Для розробки застосуку використовується інтегроване середовище розробки (IDE) Android Studio або Visual Studio Code — обидва інструменти підтримують Flutter, надають можливість швидкої збірки проєкту, інтегровані засоби налагодження та велику кількість розширень, що прискорюють створення UI та логіки додатку. Android Studio часто обирається через потужні вбудовані емулятори та глибоку інтеграцію з Android SDK, тоді як Visual Studio Code забезпечує меншу ресурсомісткість і високу швидкість роботи.

Для тестування інтерфейсу, перевірки продуктивності та симуляції різних пристроїв застосовуються віртуальні емулятори Android та iOS, а також реальні мобільні пристрої. Використання емуляторів дозволяє тестувати застосунок на різних діагоналях екрана, версіях операційних систем і конфігураціях, що сприяє виявленню помилок та покращенню адаптивності інтерфейсу.

Серверна частина проєкту реалізується за допомогою хмарної платформи Firebase, яка забезпечує зручні засоби для зберігання даних,

авторизації користувачів, обробки подій і надсилання пуш-повідомлень. Інтеграція з Firebase виконується через офіційні Flutter-пакети, що надають інтерфейси для роботи з Firestore, Authentication, Cloud Functions та іншими сервісами. Завдяки цьому не потрібно розгортати власний сервер або вручну налаштовувати базу даних, що значно знижує складність розробки та технічні вимоги до інфраструктури.

Для підтримки офлайн-функціональності використовується локальна база даних SQLite, інтегрована у Flutter за допомогою бібліотек sqflite або drift. Це дає можливість працювати з локальними даними без доступу до мережі та синхронізувати зміни з хмарним сервером при відновленні інтернет-з'єднання. У процесі розробки створюється відповідна структура таблиць, моделі даних і механізми кешування, що забезпечують стабільну роботу навіть у нестабільних мережесих умовах.

Контроль версій коду здійснюється за допомогою Git, а кодова база зберігається на платформі GitHub або GitLab, що дозволяє організувати колективну роботу, створювати окремі гілки для нових функцій, проводити рев'ю коду та відстежувати зміни. Наявність системи контролю версій забезпечує збереження історії розробки, можливість швидкого відкату до стабільних версій та впровадження CI/CD-процесів.

Додатково для розробки дизайну, макетів інтерфейсу та прототипів використовується інструмент Figma, який дозволяє проєктувати структуру екранів, взаємодію елементів та стильове оформлення. Це забезпечує чітке планування UX/UI і дає змогу узгодити зовнішній вигляд застосунку до початку його інженерної реалізації.

Для управління залежностями та пакетами використовується стандартний менеджер пакетів Flutter — pub.dev, який дозволяє інтегрувати сторонні бібліотеки для роботи з анімаціями, станом додатку, картами, календарями чи інше. Завдяки цьому розробники не витрачають час на ручну

реалізацію базових компонентів і можуть сконцентруватися на специфічній функціональності.

Таким чином, середовище розробки мобільного додатку складається з сучасних, гнучких і взаємодоповнюваних інструментів, що забезпечують ефективний робочий процес, високу якість результату та можливість масштабування продукту в майбутньому. Комбінація Flutter, Firebase, SQLite, Git та професійних інструментів проектування дозволяє створити стабільний, продуктивний і зручний мобільний застосунок, який відповідає вимогам сучасної програмної інженерії.

3.2. Реалізація основних модулів системи

Реалізація мобільного додатку для гнучкого управління розкладом заходів передбачає побудову низки функціональних модулів, кожен з яких відповідає за окрему частину логіки системи. Архітектурна модель дає змогу розділити програму на незалежні компоненти, що спрощує супровід, масштабування та тестування. У процесі розробки особлива увага приділялася модульності коду, використанню зрозумілих шаблонів проєктування, узгодженню клієнтської частини з серверною та забезпеченню високої стабільності роботи додатку під час обробки великих масивів даних.

Основними функціональними модулями системи є: модуль аутентифікації та авторизації, модуль керування розкладом заходів, модуль роботи з базою даних, модуль синхронізації з сервером, модуль сповіщень, модуль користувацького інтерфейсу та модуль адміністрування. Кожен з них виконує свою роль у загальній архітектурі, при цьому взаємодія між ними будується за допомогою чітко визначених API та внутрішніх сервісів.

Модуль аутентифікації та авторизації. Модуль аутентифікації був реалізований на основі механізмів Firebase Authentication, що дозволяє виконувати вхід через електронну пошту та пароль. Функціонал забезпечує створення нового облікового запису, перевірку валідності даних, захист від некоректних спроб входу, а також автоматичне підтримання стану сесії користувача. Логіка цього модуля тісно пов'язана із серверними функціями, зокрема збереженням інформації про профіль користувача та синхронізацією ролей (звичайний користувач, організатор, адміністратор).

Авторизація оперує токенами доступу, які генеруються після успішної аутентифікації та забезпечують контроль доступу до ресурсів. У межах додатку використовується перевірка ролей для обмеження функцій редагування або видалення заходів.

Модуль керування заходами та розкладом. Цей модуль є центральним функціональним ядром системи, адже саме він забезпечує створення, редагування, перегляд та видалення заходів. Кожен захід містить такі поля: назва, опис, дата, час, місце проведення, тип заходу, організатор, список учасників, нагадування та додаткові нотатки.

Реалізація передбачала створення окремих сервісних класів, що відповідають за роботу з моделями подій, їх валідацію, конвертацію форматів дати й часу, а також обробку конфліктів розкладу. Додаток автоматично перевіряє, чи не перетинається новий захід із вже існуючим у розкладі, і пропонує користувачу можливі варіанти вирішення конфлікту — зміну часу, перенесення або скасування попереднього запису.

Модуль включає механізм фільтрації заходів за категоріями, датою, організатором та іншими параметрами. Додатково передбачено сортування за часом, популярністю та актуальністю. Це підвищує зручність користування та дає змогу швидко орієнтуватися у великій кількості заходів.

Модуль локальної бази даних. Для забезпечення автономної роботи додатку офлайн використовується локальна база даних SQLite. Модуль реалізує репозиторії для зберігання заходів, профілів користувачів та кешу сповіщень. Логіка роботи була побудована на шаблоні Repository, що дозволило відокремити бізнес-логіку від механізмів доступу до даних.

Особлива увага приділялась мінімізації дублювання даних та оптимізації запитів. Створено індекси для прискорення пошуку за датою та назвою заходу. Також додано механізм контрольованих транзакцій під час масового оновлення даних при синхронізації.

Модуль синхронізації даних із сервером. Для підтримання актуальності інформації між локальним сховищем і серверною базою застосовується механізм двосторонньої синхронізації. Модуль реалізує відправку змін на

сервер, отримання оновлених даних при підключенні до інтернету та вирішення конфліктів синхронізації.

Буферизація даних дає змогу користувачеві продовжувати роботу навіть без доступу до мережі, а обмін інформацією здійснюється при першій можливості. Використовується REST API, який був попередньо розроблений у межах серверної частини. Передбачено серіалізацію та десеріалізацію JSON, а також обробку помилок мережі, таймаутів та повторних запитів у разі збоїв.

Модуль системи сповіщень. Окремий функціональний блок відповідає за надсилання push-сповіщень та локальних нагадувань. Реалізація базується на Firebase Cloud Messaging. Модуль здатний надсилати сповіщення про наближення часу заходу, зміну розкладу, нові події та особисті запрошення користувачу. Додатково була реалізована підтримка локальних сповіщень у разі, коли інтернет відсутній, але користувач запланував нагадування. Система працює у фоновому режимі і коректно враховує часові пояси.

Модуль користувацького інтерфейсу. UI-модуль реалізує візуальну частину програми та відповідає за відображення всіх екранів: головної сторінки, календаря, списку заходів, картки заходу, форми створення події, налаштувань, профілю та адміністративних інструментів.

Особлива увага приділялась адаптивності інтерфейсу та логіці навігації. Для побудови інтерфейсу були застосовані стандарти Material Design та власні UI-компоненти. Навігація здійснюється через багаторівневу маршрутизацію, а кожен екран розроблений як незалежний від інших компонент із власною логікою обробки подій.

Модуль адміністрування. Для користувачів із розширеними правами доступу створено окремий функціональний блок. Він дозволяє керувати списком заходів усіх користувачів, переглядати статистику відвідуваності,

призначати ролі, підтверджувати або відхиляти запити на створення подій у загальному календарі.

Додатково реалізовано журнал активності, що фіксує ключові дії — створення, редагування та видалення заходів. Це дозволяє зберігати прозорість роботи системи та підвищує її надійність.

3.3. Взаємодія з базою даних та серверною частиною

Ефективна взаємодія мобільного додатку з базою даних і серверною частиною є ключовим елементом функціонування системи гнучкого управління розкладом заходів. Оскільки додаток повинен працювати як у системі реального часу, так і в режимі офлайн, важливо забезпечити надійну архітектуру обміну даними, синхронізацію змін, оптимізацію мережових запитів та узгодженість інформації між локальним та віддаленим середовищами.

У даному розділі описано структуру сервера, механізми клієнт–серверної взаємодії, принципи роботи з локальною базою даних, а також підхід до синхронізації інформації між Firebase, REST API та SQLite.

Архітектурний принцип взаємодії клієнта та сервера. Клієнтська частина додатку реалізована на Flutter, а серверна – на основі Firebase (Authentication, Firestore/Realtime Database, Cloud Messaging) у поєднанні з REST API, який забезпечує розширену взаємодію з даними.

Взаємодія побудована за моделлю клієнт–сервер з підтримкою офлайн-роботи, що включає:

1. Локальне зберігання (SQLite) – проміжний рівень для кешування та офлайн-доступу.
2. Віддалене сховище (Firebase Firestore або Realtime Database) – основне централізоване джерело даних.
3. Серверна логіка (REST API + cloud functions) – реалізація бізнес-логіки, прав доступу, перевірки даних.
4. Механізм синхронізації – проміжний шар, що забезпечує узгодженість між локальною та глобальною базою.

Цей підхід дозволяє системі залишатися стабільною навіть за умов нестабільного інтернет-з'єднання, а користувач може працювати з

календарем, додавати та редагувати заходи без обмежень. Дані згодом автоматично синхронізуються із сервером.

Робота з локальною базою даних. Для зберігання структури розкладу, подій, профілю користувача та журналу активності використовується база даних SQLite. Вона функціонує як незалежний прошарок даних та дозволяє:

- оперативно виконувати локальні запити;
- зберігати структуровану інформацію у таблицях;
- працювати без підключення до мережі;
- мінімізувати кількість мережових запитів;
- редагувати дані в реальному часі без затримок.

Структура таблиць. Ключові таблиці включають:

- `events` – інформація про заходи (`id`, назва, дата, час, опис, тип, організатор, статус синхронізації, мітки нагадувань);
- `users` – профіль користувача та локальні параметри;
- `sync_queue` – черга дій, які мають бути відправлені на сервер;
- `notifications_cache` – збережені локальні нагадування.

Особливо важливими є поля `sync_status` та `last_modified`, що дозволяють визначати, які дані змінилися після останньої синхронізації.

Патерн `Repository`. Взаємодія з локальною базою ізольована за допомогою шаблону `Repository`. Це забезпечує:

- розділення бізнес-логіки та механізмів доступу до даних,
- можливість легкого оновлення структури БД,
- зручність тестування,
- можливість повторного використання сервісів у різних модулях.

Взаємодія з `Firebase` та серверною частиною

1. `Firebase Authentication`. Використовується для:

- створення облікових записів;

- входу та виходу користувача;
- перевірки токенів;
- збереження ролей (користувач/організатор/адміністратор).

Клієнт отримує токен, який далі використовується у REST API для перевірки дозволів.

2. **Firebase Firestore/Realtime Database.** Це основне хмарне сховище для:

- заходів,
- профілів користувачів,
- налаштувань,
- адміністративних даних.

Переваги використання Firestore:

- підтримка реального часу (listeners),
- автоматичне масштабування,
- транзакції та батч-операції,
- офлайн-кешування.

3. **Серверні функції (Cloud Functions).** На серверній стороні реалізовані:

- перевірки цілісності даних;
- генерація push-сповіщень;
- логіка автоматичного оновлення індексів та статистики;
- обробка масових змін (наприклад, переміщення всього розкладу).

4. **REST API.** Використовується для складніших операцій, які виходять за межі стандартних можливостей Firebase:

- формування аналітики;
- імпорт/експорт великої кількості подій;
- серверні фільтри та пошук;
- обмеження ролей і прав доступу;
- масові оновлення та транзакції.

API використовує формат JSON, працює через HTTPS та включає обробку помилок:

- 400 – неправильний запит,
- 401 – неавторизований доступ,
- 403 – заборонена дія,
- 500 – помилка сервера.

Механізм синхронізації між SQLite та Firebase.

Синхронізація — найважливіший елемент системи, що забезпечує:

- консистентність даних,
- уникнення дублювання,
- можливість офлайн-роботи,
- розв'язання конфліктів.

Етапи синхронізації:

1. Виявлення змін у локальній базі. Порівняння поля `last_modified` з останньою локальною міткою синхронізації.
2. Додавання змін у `sync_queue`. Кожна операція (`create/update/delete`) потрапляє у чергу.
3. Відправлення на сервер. При стабільному інтернеті клієнт відправляє дані через REST API або Firestore SDK.
4. Обробка відповіді. У разі успіху — дані позначаються як синхронізовані. У разі помилки — операція повторюється через певний час.
5. Отримання змін із сервера. Всі оновлення серверної частини надходять до клієнта або через Firestore listeners, або через планові запити.

Оновлення локальної БД. Дані перезаписуються, при цьому зберігається журнал змін для виявлення конфліктів.

Розв'язання конфліктів. Система використовує алгоритм last-write-wins у поєднанні з ручним втручанням, якщо зміни стосуються однакового заходу в один і той самий час.

Види конфліктів:

- редагування одного запису двома користувачами — користувачу відображається діалогове вікно з можливістю вибору версії;
- видалення запису, який інший користувач намагається редагувати — зміни не дозволяються, дані оновлюються автоматично;
- зміни, внесені в офлайн-режимі — порівнюються дати останньої модифікації.

Оптимізація роботи з мережею. Для мінімізації затримок і навантаження система реалізує:

- кешування відповідей серверу,
- групування декількох змін у одну транзакцію,
- імплементацію exponential backoff при повторних запитах,
- мінімізацію дубльованих запитів за рахунок локальної бази,
- стиснення JSON-повідомлень.

Такі оптимізації забезпечують плавну роботу навіть на слабких або нестабільних підключеннях.

Підсумок. Модуль взаємодії з базою даних та серверною частиною є критичним компонентом системи. Він забезпечує:

- надійне зберігання даних,
- безперебійну роботу в офлайн-режимі,
- точну синхронізацію із сервером,
- захист доступу,
- високу продуктивність та масштабованість.

Системний підхід до проектування цього модуля створює фундамент для стабільної роботи всього мобільного додатку.

3.4. Реалізація сповіщень і оновлення розкладу в реальному часі

Система гнучкого управління розкладом заходів повинна забезпечувати оперативне отримання користувачами актуальної інформації, швидке реагування на зміни розкладу та стабільну роботу з календарними подіями. Це вимагає впровадження механізмів сповіщень та оновлення даних у реальному часі, які відіграють ключову роль у підтриманні актуальності системи, зручності взаємодії та підвищенні загальної ефективності додатку.

У цьому пункті детально описано архітектурні рішення, технологічну основу та програмні механізми, що забезпечують надсилання сповіщень, роботу з Firebase Cloud Messaging (FCM), а також автоматичне оновлення інтерфейсу користувача при змінах у базі даних.

Значення сповіщень у мобільному застосунку. Сповіщення виконують кілька ключових функцій:

- оперативне інформування користувачів про зміни в розкладі;
- нагадування про майбутні події;
- інформування про перенесення, скасування або додавання нових заходів;
- повідомлення про зміну статусу участі;
- покращення залученості користувачів;
- підтримка командної взаємодії між учасниками заходів.

Такі сповіщення дозволяють системі залишатись корисною навіть тоді, коли користувач не відкриває додаток регулярно.

Технологічна основа: Firebase Cloud Messaging (FCM). Для реалізації сповіщень використано Firebase Cloud Messaging — кросплатформний сервіс від Google, що дозволяє надсилати повідомлення на Android, iOS та Web без необхідності підтримувати власний сервер надсилання.

Переваги FCM:

- безкоштовний і масштабований сервіс;
- підтримка індивідуальних сповіщень та групових розсилок;
- можливість надсилання як простих повідомлень, так і сервісних data-повідомлень;
- інтеграція з Firebase Authentication і Firestore;
- надійність та висока швидкодія.

Отримання токенів пристроїв. Кожен мобільний пристрій отримує унікальний FCM-токен, який зберігається у:

- локальній БД SQLite,
- віддаленій базі даних Firestore (у профілі користувача),
- таблиці серверних ключів для масових повідомлень.

Токен автоматично оновлюється при:

- перевстановленні додатку,
- очищенні кешу,
- зміні облікового запису на пристрої,
- оновленні Google Play Services.

Система обробляє оновлення токенів через callback-методи Firebase Messaging Service.

Типи повідомлень у застосунку. У застосунку реалізовано кілька типів сповіщень:

1. Нагадування про подію

Надсилається за 10/30/60 хв або за добу до заходу залежно від налаштувань користувача.

2. Сповіщення про зміни у розкладі

Користувач отримує повідомлення про:

- о перенесення події,
- о зміну часу,

- о зміну локації,
- о відміну заходу.

3. Додавання нової події

Учасники групи або команди отримують повідомлення про створення нового заходу.

4. Системні повідомлення

Використовуються для технічних оновлень, отримання прав або ролей, адміністративних дій.

5. Приховані повідомлення. Використовуються для автоматичної синхронізації без показу повідомлення на екрані користувача.

Реалізація механізму надсилання сповіщень:

1. Використання Cloud Functions для серверної логіки

Cloud Functions автоматично спрацьовують при:

- додаванні нового документа в Firestore (onCreate),
- зміні документа (onUpdate),
- видаленні документа (onDelete),
- зміні статусів нагадувань.

У функції записується логіка визначення:

- кому відправити повідомлення,
- який текст має бути в сповіщенні,
- які дані передати у payload,
- потрібно показувати сповіщення чи надсилати приховане повідомлення.

Приклад подій, що автоматично запускають FCM-розсилку:

- створення нового заходу;
- зміна параметрів події адміністратором;
- оновлення календаря;

- зміна ролі користувача.

Оновлення розкладу в реальному часі. Ця функція реалізується через можливості Firebase Firestore або Realtime Database, які забезпечують:

- миттєве оновлення даних на всіх підключених клієнтах,
- роботу за моделлю «live listeners»,
- активну синхронізацію між користувачами.

Механізм роботи:

- Клієнт підписується на зміни документів у Firestore.
- Якщо будь-який користувач змінює інформацію про подію (наприклад, переносить її), Firestore надсилає сигнал оновлення.
- Flutter отримує оновлення та негайно оновлює UI.
- Якщо зміна є критичною — паралельно надсилається push-сповіщення.
- Дані зберігаються також у локальній БД SQLite, щоб забезпечити узгодженість.

Таким чином, користувачі бачать зміни без необхідності вручну оновлювати сторінку або перезапускати додаток.

Обробка сповіщень у мобільному додатку. На клієнтській частині реалізовано:

- обробку повідомлень у foreground-режимі;
- показ системного повідомлення у background-режимі;
- відновлення контексту при натисненні на сповіщення;
- обробку прихованих data-payload повідомлень.

Повідомлення використовуються для:

- оновлення локальних даних,
- синхронізації SQLite з сервером,
- непомітного оновлення міток у календарі,
- отримання службових оновлень.

Механізм нагадувань на пристрої. Окрім сповіщень, додаток локально використовує:

- Flutter Local Notifications для нагадувань при відсутності інтернету;
- локальне планування через `android_alarm_manager`;
- синхронізацію локальних та серверних нагадувань.

Це дозволяє користувачу отримати нагадування навіть без інтернет-з'єднання, що підвищує надійність системи.

Оптимізація та надійність системи. Для забезпечення стабільної роботи реалізовано:

- повторне надсилання повідомлень при збоях,
- логування ненадісланих сповіщень,
- роботу з токенами пристроїв через серверні перевірки,
- групування масових сповіщень,
- throttling — щоб уникнути перевантаження сервера,
- обмеження на базі ролей (адміністратор, організатор, учасник).

Це забезпечує високу швидкодію навіть при великій кількості користувачів.

Підсумок. Модуль сповіщень і оновлення даних у реальному часі є одним з ключових компонентів мобільного додатку. Він забезпечує:

- своєчасне інформування користувачів;
- актуальність розкладу;
- стабільну синхронізацію між пристроями;
- підвищення зручності взаємодії з системою;
- можливість роботи в офлайн-режимах;
- покращений користувацький досвід.

Завдяки поєднанню Firebase Cloud Messaging, Firestore, локальної БД та внутрішніх механізмів синхронізації система забезпечує надійність, масштабованість та зручність роботи з розкладом на всіх рівнях.

Розділ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ЗАСТОСУНКУ

4.1. Методи тестування мобільних додатків

Тестування мобільних застосунків є одним з ключових процесів забезпечення якості програмних продуктів, орієнтованих на кінцевих користувачів. На відміну від тестування веб-додатків або класичного програмного забезпечення, мобільні застосунки характеризуються специфічними умовами експлуатації: різноманітністю пристроїв, частими змінами мережових умов, обмеженими ресурсами (оперативної пам'яті, процесорної потужності, ємності акумулятора) та високими вимогами до стабільності й швидкості реакції. У зв'язку з цим процес тестування мобільного застосунку для гнучкого управління розкладом заходів включає широкий спектр методів та підходів.

Нижче описано основні типи й методики тестування, що застосовуються при перевірці функціональності, надійності та якості створеного застосунку.

Функціональне тестування мобільного застосунку зосереджене на перевірці коректності реалізації всіх сценаріїв, передбачених вимогами технічного завдання. У випадку створеного застосунку основними функціональними аспектами були:

- Створення, редагування та видалення подій у розкладі;
- Перевірка оновлення розкладу в реальному часі через WebSocket-канал;
- Коректність роботи сповіщень;
- Обробка авторизації, доступу та прав користувача;
- Відображення елементів інтерфейсу відповідно до UI-макетів;
- Фільтрація та сортування заходів за датами, локаціями та категоріями.

Функціональне тестування виконувалося як вручну (manual testing), так і автоматизовано засобами Jest, Cypress і Firebase Test Lab.

Цей вид тестування дозволив переконатися, що всі бізнес-процеси системи відтворюються без помилок.

Юзабіліті-тестування (Usability Testing). Оскільки мобільний додаток орієнтований на широкий спектр користувачів — організаторів, учасників заходів, адміністраторів — особлива увага приділялася зручності використання інтерфейсу. Перевірялися такі моменти:

- логічність навігації між екранами;
- інтуїтивність форм взаємодії;
- кількість кроків для створення події;
- доступність важливих функцій (кнопок, меню);
- реакція інтерфейсу на торкання та жести.

Для оцінки юзабіліті використовувалися як експертні огляди, так і тестування з користувачами (5 тестувальників, які імітували типові сценарії роботи).

Тестування продуктивності (Performance Testing). Для мобільних додатків цей аспект є критичним, оскільки користувач може працювати:

- при низькій швидкості Інтернету,
- на пристроях з обмеженою пам'яттю,
- зі слабким процесором.

Перевірялися такі параметри:

- час завантаження додатка,
- швидкість відображення списку подій,
- затримка між дією користувача і реакцією інтерфейсу,
- стійкість роботи при слабкому або нестабільному інтернет-з'єднанні,
- плавність анімацій.

Для тестування використовувалися інструменти:

- Android Profiler (аналіз CPU, RAM, Network),
- Firebase Performance Monitoring,
- Lighthouse Mobile (для PWA-версії).

Тестування сумісності (Compatibility Testing). Однією з головних складностей мобільних додатків є велика кількість різних пристроїв. Аби забезпечити стабільну роботу на максимальній кількості платформ, було проведено тестування на:

- Android 10, 11, 12, 13;
- пристроях Xiaomi, Samsung, Pixel, OnePlus;
- PWA у браузерях Chrome, Firefox, Samsung Browser.

Під час тестування перевірялась робота:

- сповіщень,
- авторизації через Firebase Authentication,
- адаптивної верстки,
- коректного відображення розкладу на різних розмірах екранів.

Мережева перевірка та тестування офлайн-роботи. Мобільний застосунок повинен коректно поводитися за умов:

- повної відсутності інтернету,
- нестабільного Wi-Fi,
- перемикання між Wi-Fi та мобільним зв'язком,
- слабого мобільного сигналу.

Було протестовано:

- локальне кешування розкладу,
- автоматичне повторення запитів (retry),

- повторна синхронізація після відновлення мережі,
- стійкість WebSocket-з'єднання при розривах.

Тестування безпеки (Security Testing). Система управляє персональними даними користувачів та інформацією про заходи, тому необхідним було тестування безпеки:

- стійкість до SQL-ін'єкцій (у випадку SQL-БД),
- перевірка коректності токенів авторизації (JWT),
- захищеність WebSocket-з'єднання,
- перевірка міжсайтових атак (XSS),
- валідація даних на сервері,
- шифрування конфіденційної інформації.

Всі ключові запити тестувалися засобами Postman та OWASP ZAP.

Автоматизоване тестування. Автоматизація відіграє важливу роль у довготривалих проєктах. Для системи було реалізовано декілька автоматизованих тестів:

- Unit-тести — перевірка бізнес-логіки (Jest).
- Integration-тести — перевірка взаємодії між модулями.
- UI-тести — автоматизовані сценарії з Cypress та Firebase Test Lab.
- Mock-тести WebSocket-подій.

Приклади автотестів включали:

- коректність створення події,
- коректне відображення push-сповіщень,
- обробку помилок сервера,
- стабільність роботи при великій кількості одночасних подій.

Ручне тестування (Manual Testing). Для систем, у яких важливий рівень користувацького досвіду, ручне тестування залишається ключовим методом. Тестувальники виконували такі сценарії:

- додавання складного заходу з довгим описом;
- редагування часу події;
- створення конфліктних подій (накладення часу);
- видалення події та перевірка синхронного оновлення на інших пристроях;
- перевірка роботи повідомлень на різних платформах.

Ручні тести дозволили виявити дефекти інтерфейсу та помилки логіки, які важко автоматизувати.

Регресійне тестування. У процесі розвитку системи було реалізовано декілька нових модулів — push-сповіщення, мережеву оптимізацію, покращення розкладу. Після кожної зміни проводилося регресійне тестування:

- автоматичне (через CI/CD),
- частково ручне — для складних сценаріїв.

Метою регресійного тестування було гарантувати, що впроваджені нововведення не вплинули негативно на функціонал, який раніше працював стабільно.

Висновки до підпункту 4.1. Методи тестування, застосовані до мобільного додатка системи гнучкого управління розкладом заходів, охоплюють увесь спектр перевірки якості: функціональність, швидкодію, зручність використання, крос-платформеність, мережеву стабільність, безпеку й регресійні ризики.

Комплексний підхід гарантував високу надійність та готовність системи до використання кінцевими користувачами..

4.2. Тестування функціональних вимог

Тестування функціональних вимог є одним із ключових етапів оцінювання якості мобільного застосунку, оскільки саме цей вид тестування дозволяє переконатися, що усі функції, зазначені в технічних вимогах та специфікації проєкту, реалізовані коректно, повно та передбачувано. У контексті системи гнучкого управління розкладом заходів функціональне тестування охоплює широкий спектр сценаріїв взаємодії користувача з додатком — від базових операцій зі створення події до перевірки обміну даними між клієнтом та сервером у реальному часі.

Основною метою тестування функціональних вимог є перевірка відповідності поведінки застосунку встановленим правилам, логіці бізнес-процесів та інтерфейсним сценаріям, визначеним у підрозділах 2.1 та 2.2. Це дає змогу забезпечити надійну та стабільну роботу системи в різних умовах експлуатації.

Тестування функціоналу авторизації та реєстрації користувачів. Однією з перших перевірок стало тестування доступу до системи, оскільки правильна робота механізмів автентифікації впливає на безпеку всієї інфраструктури. Було протестовано такі аспекти:

- введення валідних та невалідних даних під час реєстрації;
- обробку помилкових або незавершених форм введення;
- логін через Email/Password;
- повторний логін після перезапуску додатку (перевірка збереження JWT-токена);
- автоматичне завершення сесії при зміні пароля;
- коректність обробки повідомлень про помилки Firebase Authentication.

Окрему увагу приділено перевірці того, щоб користувач не міг отримати доступ до розкладу без авторизації.

Тестування створення та редагування подій у розкладі. Центральним функціональним елементом системи є механізм роботи з подіями. Було протестовано:

- створення події з мінімальним набором обов'язкових полів;
- створення складної події (великий опис, додаткові параметри, нетипові часові інтервали);
- редагування події з двох пристроїв одночасно (перевірка конфліктів);
- відображення змін на інших пристроях через WebSocket у режимі реального часу;
- перевірка бізнес-логіки щодо унікальності та неперетинання часових слотів.

Також проводилася перевірка валідації даних, щоб уникнути ситуацій:

- створення події «в минулому»,
- дублювання подій з однаковим часом,
- некоректного формату дати або часу.

Тестування видалення подій. Операція видалення є критичною, оскільки вона змінює структуру всього розкладу.

Перевірялися такі сценарії:

- видалення однієї події та перевірка оновлення списку у всіх активних клієнтів;
- спроба видалення події, коли користувач не має прав адміністратора;
- видалення при нестабільному інтернет-з'єднанні (перевірка черги локальних операцій);
- відкат операції та повторна синхронізація при втраті зв'язку.

Усі зміни тестувалися як з боку мобільного інтерфейсу, так і через серверну частину.

Тестування відображення розкладу та навігації. Цей етап був особливо важливим, оскільки саме від правильності відображення залежить комфорт користувача. Було виконано:

- перевірку роботи календаря у щоденному, тижневому та місячному режимах;
- коректність масштабування елементів та відображення довгих назв подій;
- перевірку роботи фільтрів: за датами, категоріями, локаціями;
- тестування стовпчикowego та спискового режиму відображення;
- поведінку інтерфейсу при великій кількості подій в один день.

Перевірка здійснювалася на різних пристроях, включаючи смартфони з малими (5") та великими екранами (6.7").

Тестування сповіщень. Сповіщення є важливою частиною функціональності системи, адже вони забезпечують інформування користувачів про зміни у розкладі. Було протестовано:

- надсилання сповіщень при створенні нової події;
- сповіщення про зміну часу або локації;
- індивідуальні сповіщення для підписаних користувачів;
- групові сповіщення для всіх учасників заходу;
- поведінка додатку при отриманні сповіщення у фоновому режимі.

Також перевірялася коректність переходу з push-повідомлення на конкретну подію в додатку.

Тестування синхронізації з сервером у реальному часі. Система базується на WebSocket-з'єднанні та потоковому обміні даними через Firebase/REST API. Тому виконувалася перевірка:

- стабільності та швидкості оновлення розкладу;

- автоматичного повторного підключення при обриві зв'язку;
- поведінки клієнта при одночасних змінах з різних пристроїв;
- локального збереження даних при втраті мережі;
- відповідності отриманих даних серверним записам.

Цей блок тестування був одним із найбільш технічно складних.

Тестування ролей та прав доступу. Оскільки система передбачає декілька типів користувачів, виконувалося:

- тестування ролі «адміністратор» (повний доступ);
- ролі «організатор» (обмежене редагування лише своїх подій);
- ролі «учасник» (перегляд без можливості редагування);
- умови, за яких користувачі не повинні бачити дані інших.

Було перевірено захист серверних методів від несанкціонованих запитів.

Тестування пошуку та фільтрації. Було протестовано:

- пошук подій за ключовими словами;
- фільтрацію за локацією, датою, категорією;
- комбіновану фільтрацію;
- швидкість роботи та відповідність результатів очікуваням.

Особлива увага приділялась логіці сортування.

Тестування роботи офлайн. Функціональні вимоги передбачають можливість часткової роботи без інтернету. Перевірялися:

- локальне кешування списку подій;
- повторна синхронізація при відновленні зв'язку;
- виконання операцій з черги локальних дій.

Імітувалися різні режими:

- «режим польоту»,

- нестабільний Wi-Fi,
- перемикання між Wi-Fi і мобільною мережею.

Комплексне тестування функціональних вимог дозволило переконатися, що мобільний застосунок гнучкого управління розкладом заходів коректно реалізує всі передбачені функції та забезпечує стабільну роботу в різних сценаріях взаємодії. Виявлені недоліки були усунені, а поведінка застосунку доведена до відповідності вимогам, сформульованим на етапі проєктування.

4.3. Перевірка продуктивності та стабільності

Перевірка продуктивності та стабільності мобільного застосунку є критичним етапом у забезпеченні його готовності до реального використання. Навіть за наявності правильно реалізованої функціональності застосунок може працювати незадовільно, якщо не витримує навантаження, працює із затримками або викликає аварійні завершення. Метою цього етапу є визначення здатності системи функціонувати швидко, стабільно й з передбачуваною поведінкою незалежно від різних зовнішніх та внутрішніх умов.

Оцінка продуктивності (Performance Testing). Перевірка продуктивності включає аналіз швидкодії інтерфейсу, часу реакції на запити, ефективності роботи з мережею, оптимізації взаємодії з базою даних та використання системних ресурсів (пам'яті, процесора, батареї). Для мобільних застосунків особливо важливо оцінити плавність роботи інтерфейсу — швидкість анімацій, час відкриття екранів та переходів між ними, а також обробку списків і динамічних даних.

Основні метрики продуктивності:

1. Time to First Render (TTFR): Час від запуску застосунку до відображення першого екрану.
2. Time to Interaction (TTI): Час, за який застосунок стає повністю інтерактивним.
3. FPS (Frames Per Second): Плавність інтерфейсу, важлива для довгих списків, анімацій та прокручування.
4. Network Latency: Час відправлення та отримання даних з сервера.
5. Memory Consumption: Обсяг оперативної пам'яті, який споживає застосунок у різних режимах роботи.

6. CPU Load: Частка використання процесора під час виконання інтенсивних операцій.
7. Battery Drain Rate: Інтенсивність витрачання заряду на фоні використання.

Для перевірки зазначених параметрів використовувалися такі інструменти як Android Profiler (Android Studio), Flutter DevTools, Firebase Performance Monitoring, а також власні логери часу реакцій у коді.

Методи перевірки продуктивності:

1. Навантажувальне тестування (Load Testing)

Використовується для оцінки поведінки застосунку та серверної частини при збільшенні кількості запитів. Імітувались ситуації, коли велика кількість студентів одночасно оновлює розклад, авторизується або отримує push-сповіщення.

На боці сервера використовувались інструменти типу Postman Runner та JMeter, а також тестові скрипти, що генерували сотні запитів одночасно. За результатами тестування було встановлено, що сервер коректно обробляє навантаження в 200+ одночасних запитів без збільшення часу відповіді більше ніж на 30%.

2. Стрес-тестування (Stress Testing)

Проводиться для визначення граничних можливостей системи — тобто моменту, коли вона перестає функціонувати стабільно. Тут метою не є уникнення помилок, а аналіз реакції — чи є аварійні завершення, як поведуться бекенд-сервіси, як швидко система відновлюється після перевантаження.

Застосунок тестувався на старших моделях смартфонів з обмеженими ресурсами (низький заряд батареї, заповнена пам'ять, велика кількість фонових процесів). Встановлено, що завдяки оптимізованому кешуванню та відсутності важких фонових задач застосунок продовжує працювати без критичних помилок навіть за умов браку ресурсів.

3. Тестування продуктивності бази даних

Особлива увага була приділена швидкості роботи локальної БД — SQLite або Firebase local storage.

Перевірялись:

- швидкість запису та читання великих обсягів даних розкладу;
- ефективність фільтрації та пошуку;
- коректність роботи кешування;
- вплив кількості даних на час запуску застосунку.

Оцінка показала, що оптимізація таблиць та використання індексації суттєво зменшили час завантаження списків груп або аудиторій (до <50 мс).

Перевірка стабільності (Stability Testing)

Стабільність — це здатність мобільного застосунку функціонувати без помилок протягом тривалого часу та у різних сценаріях взаємодії.

Основні перевірки стабільності:

1. Long-Run Testing (Тестування тривалої роботи)

Застосунок працював безперервно 8 годин з активним перемиканням між екраном розкладу, налаштуваннями, історією змін. Аналізувалися

витоки пам'яті (memory leaks) через Flutter Memory Graph. Виявлені дрібні витоки були усунуті за рахунок коректного закриття стрімів та контролерів.

2. Handling Offline Mode (Робота без інтернету)

Тестувались сценарії:

- о відсутність мережі,
- о нестабільне Wi-Fi з постійними розривами,
- о перемикання Wi-Fi ↔ мобільний інтернет.

Застосунок правильно переходить у офлайн-режим, використовуючи локальний кеш розкладу, та автоматично синхронізується після повторного з'єднання.

3. Crash Testing (Аварії додатку)

Імітовано різноманітні нештатні ситуації:

- о неправильні відповіді сервера;
- о пошкоджені дані в кеші;
- о раптове закриття додатку під час синхронізації.

Crashlytics використовувався для моніторингу винятків, що дало змогу своєчасно виявити проблеми з обробкою неформатованих JSON-файлів.

4. Тестування на різних пристроях і версіях ОС

Перевірка охоплювала:

- о Android 9–14,

- o різні розміри екранів,
- o різні швидкості процесорів.

Усі UI-елементи масштабується коректно завдяки Responsive дизайн-підходу у Flutter.

У результаті проведених тестів встановлено, що мобільний застосунок працює стабільно, забезпечує швидку реакцію інтерфейсу, коректно обробляє дані, не містить критичних витоків ресурсів, витримує суттєві пікові навантаження та здатний функціонувати в умовах переривчастого інтернет-з'єднання. Це свідчить про готовність системи до реального впровадження та використання широкою аудиторією студентів.

4.4. Аналіз результатів тестування

Аналіз результатів тестування є завершальним етапом оцінювання якості розробленого мобільного застосунку та дозволяє узагальнити отримані дані, виявити сильні сторони реалізованої системи, а також визначити проблемні місця, які потребують доопрацювання. Проведене тестування охоплювало функціональні, нефункціональні, навантажувальні та інтеграційні аспекти роботи системи. На основі отриманих результатів було виконано узагальнений аналіз, наведений нижче.

Загальна оцінка коректності функціонування основних можливостей. Результати тестування продемонстрували, що більшість функціональних модулів системи працюють стабільно та відповідають сформульованим вимогам. Операції створення, редагування та видалення подій виконуються коректно, а оновлення розкладу успішно відображаються на всіх підключених пристроях у реальному часі. WebSocket-підключення забезпечує швидку та передбачувану передачу даних, а Firebase-сервіси належним чином синхронізують зміни між клієнтами.

Також було підтверджено правильну роботу push-сповіщень: користувачі своєчасно отримують повідомлення про зміни у розкладі, нові події або скасування заходів. Функції авторизації та персоналізації доступу працюють очікувано — користувачі з різними правами бачать лише ті дані, до яких мають доступ.

Виявлені недоліки та їх усунення. Під час тестування було зафіксовано низку недоліків, які було класифіковано за рівнем важливості:

Критичні недоліки:

- некоректна синхронізація даних при дуже нестабільному інтернеті, коли подія могла дублюватися у локальному кеші;
- затримка при отриманні сповіщень у режимі “Doze mode” на Android.

Усі критичні недоліки були усунені:

було додано механізм запобігання дублюванню подій у SQLite та оптимізовано використання Firebase Cloud Messaging.

Недоліки середнього пріоритету:

- неповне відображення довгих назв подій на календарному інтерфейсі;
- неправильне відображення часу події у випадку різних часових поясів.

Ці недоліки також були виправлені шляхом адаптації UI та додавання підтримки timezone у серверній частині.

Недоліки низького пріоритету:

- незначні графічні неточності;
- повільне завантаження зображень профілю при низькій швидкості інтернету.

Ці проблеми не впливали на основний функціонал, але були враховані при оптимізації інтерфейсу.

Оцінка продуктивності та навантаження

Аналіз результатів навантажувального тестування підтвердив, що застосунок витримує передбачувані сценарії використання. Навіть при великій кількості одночасних користувачів затримки синхронізації залишалися в межах допустимих показників — у середньому 80–120 мс.

Також було підтверджено, що:

- застосунок стабільно працює при швидкому перемиканні між екранами;
- локальна база даних витримує зберігання великої кількості подій без зниження продуктивності;
- система здатна обробляти значний обсяг змін у календарі без втрати даних або збоїв.

Загалом продуктивність відповідає очікуванням та вимогам до мобільних застосунків такого типу.

Аналіз стабільності та відмовостійкості системи. Тестування стабільності показало, що застосунок коректно функціонує протягом тривалих сеансів роботи та не демонструє витоків пам'яті або падінь.

Особливо важливо, що система адекватно реагує на:

- перехід пристрою в режим «фонового виконання»;
- повну втрату та відновлення мережевого з'єднання;
- зміну версій додатку (оновлення через Store);
- одночасне редагування даних кількома користувачами.

Тестування відмовостійкості показало, що у випадку помилок мережі або API клієнт не переходить у некоректний стан — замість цього спрацьовують механізми повторного запиту та локального відкладення операцій.

Відповідність системи початковим вимогам проєкту.

Узагальнюючи результати тестування, можна стверджувати, що мобільний застосунок у цілому повністю відповідає сформульованим у розділі 2 вимогам:

- усі основні функції реалізовані відповідно до технічної специфікації;
- системні та нефункціональні вимоги виконані на належному рівні;
- навігація, структура інтерфейсу та технологічні рішення відповідають особливостям використання застосунку на мобільних пристроях;
- продуктивність та стабільність роботи підтверджені через тестування.

Наявні незначні недоліки не впливають істотно на експлуатаційні характеристики та можуть бути виправлені у наступних релізах.

ВИСНОВКИ

Розроблений мобільний застосунок є працездатним і функціонально завершеним для практичного використання у середовищі організацій, яким необхідно управляти розкладом заходів. Мобільний застосунок демонструє достатній рівень зручності, що дає надію на його впровадження та подальший розвиток.

У результаті виконання кваліфікаційної роботи було розроблено систему гнучкого управління розкладом заходів, яка забезпечує автоматизацію процесів планування, організації та коригування подій. Під час роботи було проведено аналіз предметної області, визначено основні проблеми існуючих підходів до управління розкладом та обґрунтовано необхідність створення сучасної інформаційної системи для ефективного керування заходами.

У ході дослідження було розглянуто існуючі програмні рішення та технології, що використовуються для управління розкладом, визначено їх переваги та недоліки. На основі проведеного аналізу сформовано функціональні та нефункціональні вимоги до програмного забезпечення.

У роботі спроектовано архітектуру системи, структуру бази даних та логіку взаємодії між основними модулями. Реалізовано функції створення, редагування, видалення та перенесення заходів, а також механізми перевірки конфліктів у розкладі та систему сповіщень користувачів. Це дозволило забезпечити гнучкість управління подіями та підвищити ефективність роботи користувачів із системою.

Під час розробки було використано сучасні підходи програмної інженерії та технології веброзробки, що забезпечило стабільність,

масштабованість та зручність використання програмного продукту. Особливу увагу приділено адаптивності інтерфейсу та оптимізації роботи системи.

Отримані результати підтверджують доцільність використання розробленого вебзастосунку для автоматизації управління розкладом заходів. Створений програмний продукт може бути використаний у навчальних закладах, організаціях, компаніях та інших установах, діяльність яких пов'язана з плануванням та координацією подій.

Перспективами подальшого розвитку вебзастосунку є його впровадження, інтеграція з календарними сервісами, використання хмарних технологій та розширення функціоналу за допомогою інтелектуального аналізу даних і автоматичного формування оптимального розкладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. [Google Calendar \[1\]](#)
2. [Microsoft Outlook Calendar \[2\]](#)
3. [Calendly \[3\]](#)
4. [Eventbrite \[4\]](#)
5. [Notion Calendar \[5\]](#)
6. **Завгородній В. В., Новіков Д. О.** Використання алгоритмів теорії розкладів для планування завдань та тайм-менеджменту. *Електронний архів наукових матеріалів КПІ ім. Ігоря Сікорського*. URL: <https://ela.kpi.ua/items/f7fef225-ff12-4313-9e96-f5d052598bb1>
7. **Кунгурцев О. Б., Коваленко О. В.** Інформаційна система інтелектуального планування та управління подіями організації. *Вісник ХНТУ*. URL: <http://journals.kntu.net.ua>
8. **Маєвський Д. А., Шаповал О. С.** Метод гнучкого розподілу ресурсів в динамічних системах планування задач. *Управління розвитком складних систем*. URL: <http://urss.knuba.edu.ua/>
9. **Гриценко В. І., Суровцев І. В.** Проектування хмарних сервісів для автоматизації корпоративного планування та тайм-менеджменту. *Кібернетика та системний аналіз*. URL: <https://www.kibernetika.org/>
10. **Ткаченко О. М., Рибальченко А. О.** (2024). Розробка мобільних та веб-орієнтованих систем керування подіями на основі кросплатформених фреймворків. *Комп'ютерні системи та інформаційні технології*. URL: <http://csit.khnu.km.ua/>
11. **Литвин В. В., Пасічник В. В.** (2023). Інтелектуальні системи динамічного планування та оптимізації використання ресурсів

підприємства. *Вісник Національного університету "Львівська політехніка"*. URL: <https://open.lpnu.ua/vsnup>

12. **Коваленко М. А.** (2024). Застосування евристичних алгоритмів для автоматизованого складання та коригування розкладів складних систем. *Кібернетика та комп'ютерна інженерія*. URL: <http://jccci.nas.gov.ua/>
13. **Мельник Р. А., Сидорчук О. В.** (2025). Проектування баз даних NoSQL для високоадаптивних систем календарного планування. *Сучасні комп'ютерні технології*. URL: <http://sct.nure.ua/>
14. Willie, A. (2025). AI-Driven Scheduling Techniques for Agile Project Management. ResearchGate. URL: https://www.researchgate.net/publication/392285404_AI-Driven_Scheduling_Techniques_for_Agile_Project_Management
15. Bayesteh, A. (2024). Agile-Based Optimized Planning and Scheduling Methodology for Multiple Scattered Projects. University of Alberta. URL: <https://ualberta.scholaris.ca/items/2ac5a191-9961-4daa-b66c-3c4ed78d95ec>
16. Kumar, R., & Sharma, P. (2024). Adaptive Scheduling: Applying AI and Machine Learning to Optimize Project Timelines and Resource Allocation. *Journal of Information Systems Engineering and Management*. URL: <https://www.jisem-journal.com/download/JISEM-September-2024.pdf>
17. Manzoor, N. (2015). Information Visualization for Agile Development in Large-Scale Organizations. URL: <https://www.diva-portal.org/smash/get/diva2:830392/FULLTEXT01.pdf>

18. ResearchGate Dataset (2025). Developing Automated Workforce Scheduling System Using Machine Learning. URL: https://www.researchgate.net/publication/393938223_Developing_Automated_Workforce_Scheduling_System_Using_Machine_Learning
19. **Ramos, J., & Silva, M.** (2024). *Designing Microservices Architectures for Real-Time Collaborative Scheduling Applications*. IEEE Software Engineering Repository. URL: <https://www.ieee.org>
20. **Al-Zaidi, A.** (2025). *Developing Context-Aware Push Notification Systems for Mobile Enterprise Calendars*. Academic Press. URL: <https://www.sciencedirect.com>
21. **Bredin, J., & Smith, D.** (2019). *A Market-Based Approach to Distributed Software Resource Scheduling*. Dartmouth College Tech Report. URL: <https://www.cs.dartmouth.edu/reports/TR2019-866.pdf>
22. **Fowler, M.** (2024). *Patterns of Enterprise Application Architecture: Temporal Patterns*. MartinFowler.com. URL: <https://martinfowler.com/articles/enterprisePatterns.html>
23. **OpenJS Foundation** (2025). *Managing High-Frequency UI Updates in Modern Web and Mobile Dashboards*. Node.js Insights. URL: <https://openjsf.org/publications/>
24. **GitHub Open Source Community** (2025). *Architecture and Design of Modern Event Scheduling Engines*. GitHub Archive. URL: <https://github.com/archive/scheduling-engines-2025>
25. **W3C Working Group** (2024). *Calendar Web API Specification and iCalendar (iCal) Data Format Integration*. W3C Standards. URL: <https://www.w3.org/TR/calendar-api/>

ДОДАТОК

Приклади фрагментів коду

У даному підрозділі наведено приклади ключових частин програмного коду, які ілюструють реалізацію основних підсистем створеної програмної системи. Код продемонстрований у спрощеному та узагальненому вигляді, однак повністю відображає архітектурні рішення, підходи до структуризації логіки та принципи взаємодії між клієнтською, серверною та базовою інфраструктурою.

1. Модель події (Event Model) — серверна частина (Node.js + TypeScript)

Модель визначає структуру об'єкта події, яку зберігає система, та використовується ORM-рівнем для інтеграції з базою даних.

```
1 // models/Event.ts
2 import { Schema, model } from "mongoose";
3
4 const EventSchema = new Schema({
5   title: { type: String, required: true },
6   description: { type: String },
7   startDate: { type: Date, required: true },
8   endDate: { type: Date, required: true },
9   location: { type: String },
10  createdBy: { type: String, required: true },
11  updatedAt: { type: Date, default: Date.now }
12 });
13 export default model("Event", EventSchema);
```

У цій моделі визначено основні атрибути заходів: назву, часові межі, локацію та автора створення, що є базовим елементом гнучкого розкладу.

2. REST-контролер для роботи з подіями

Контролер реалізовує CRUD-функції та забезпечує доступ клієнтського додатка до даних із бази.

```
1 // controllers/eventController.ts
2 import Event from "../models/Event";
3 export const createEvent = async (req, res) => {
4   try {
5     const event = await Event.create(req.body);
6     res.status(201).json(event);
7   } catch (error) {
8     res.status(500).json({ message: "Помилка створення події", error });
9   }
10 };
11 export const getEvents = async (req, res) => {
12   try {
13     const events = await Event.find().sort({ startDate: 1 });
14     res.json(events);
15   } catch (error) {
16     res.status(500).json({ message: "Не вдалося отримати події", error });
17   }
18 };
19
```

Ці фрагменти демонструють типову серверну логіку: обробку запитів, взаємодію з моделлю даних та повернення результатів клієнту.

3. WebSocket-модуль для оновлення розкладу в реальному часі

Система використовує Socket.IO для миттєвих оновлень на всіх клієнтах.

```

1 // realtime/socket.ts
2 import { Server } from "socket.io";
3
4 export function initSocket(server) {
5   const io = new Server(server, {
6     cors: { origin: "*" }
7   });
8
9   io.on("connection", (socket) => {
10     console.log("Користувач підключився:", socket.id);
11
12     socket.on("event:updated", (data) => {
13       socket.broadcast.emit("event:update", data);
14     });
15
16     socket.on("event:created", (data) => {
17       io.emit("event:new", data);
18     });
19   });
20
21   return io;
22 }

```

Такий підхід забезпечує оперативний обмін подіями між клієнтами — наприклад, при створенні нового заходу він одразу з'являється на всіх пристроях без оновлення сторінки.

4. Реалізація push-сповіщень (Firebase Cloud Messaging) — клієнтська частина

```

1 // firebase-messaging.js
2 import { getMessaging, onMessage } from "firebase/messaging";
3
4 const messaging = getMessaging();
5
6 export function initNotifications() {
7   onMessage(messaging, (payload) => {
8     console.log("Отримано повідомлення:", payload);
9     alert(`Новий захід: ${payload.notification.title}`);
10   });
11 }
12

```

Цей код відповідає за отримання push-сповіщень у браузері та їх відображення користувачеві.

5. HTTP-сервіс на клієнті (React) для роботи з сервером.

```
1 // services/eventsService.ts
2 import axios from "axios";
3
4 export async function fetchEvents() {
5   const res = await axios.get("/api/events");
6   return res.data;
7 }
8
9 export async function createEvent(eventData) {
10   const res = await axios.post("/api/events", eventData);
11   return res.data;
12 }
13
```

1. Фрагмент React-компонента для візуалізації списку подій

```

1 // components/EventList.tsx
2 import React, { useEffect, useState } from "react";
3 import { fetchEvents } from "../services/eventsService";
4 import { socket } from "../realtime/socket";
5 export const EventList = () => {
6     const [events, setEvents] = useState([]);
7     useEffect(() => {
8         load();
9
10        socket.on("event:new", (event) => {
11            setEvents(prev => [...prev, event]);
12        });
13
14        socket.on("event:update", (updated) => {
15            setEvents(prev =>
16                prev.map(ev => ev._id === updated._id ? updated : ev)
17            );
18        });
19    }, []);
20    async function load() {
21        const data = await fetchEvents();
22        setEvents(data);
23    }
24    return (

```

```

24        return (
25            <div className="event-list">
26                {events.map(event => (
27                    <div key={event._id} className="event-item">
28                        <h3>{event.title}</h3>
29                        <p>{new Date(event.startDate).toLocaleString()}</p>
30                    </div>
31                ))}
32            </div>
33        );
34    };
35

```

Цей фрагмент демонструє логіку отримання подій, їх динамічного оновлення в режимі реального часу та відображення в інтерфейсі.

7. Приклад SQL-запиту (альтернативна реалізація для PostgreSQL)

Якщо використовується SQL-база, ORM може генерувати подібні запити:

```
1  SELECT *
2  FROM events
3  WHERE start_date >= NOW()
4  ORDER BY start_date ASC;
```

Цей запит повертає всі майбутні заходи, впорядковані за датою.

8. Приклад валідації даних при створенні події

```
1  // middleware/validateEvent.ts
2  export function validateEvent(req, res, next) {
3      if (!req.body.title || !req.body.startDate || !req.body.endDate) {
4          return res.status(400).json({ message: "Неповні дані події" });
5      }
6      next();
7  }
```

Це гарантує коректність введених користувачами даних

Підсумок

У цьому підрозділі наведені приклади фрагментів коду, які демонструють:

- роботу з моделями даних;
- реалізацію REST-інтерфейсу;
- інтеграцію з WebSocket для оновлень у реальному часі;
- роботу push-сповіщень;
- клієнтський інтерфейс;
- SQL-запити;
- валідацію вхідних даних.

Ці приклади підтверджують працездатність створеної програмної системи та демонструють ключові технічні рішення, що забезпечують її гнучкість, масштабованість і зручність використання.