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

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ «ПЕРСОНАЛЬНИЙ МЕНЕДЖЕР ЗАВДАНЬ» З ВИКОРИСТАННЯМ REACT NATIVE ТА FIREBASE

DEVELOPMENT OF A MOBILE APPLICATION «PERSONAL TASK MANAGER» USING REACT NATIVE AND FIREBASE

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Виримчук Богдан Ігорович

Керівник:
Суринович Олена Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«10» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Виримчуку Богдану Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка мобільного додатку «Персональний менеджер завдань» з використанням React Native та Firebase»*
Керівник роботи: *к.т.н., доцент Суринович О. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *інструменти та середовища розробки (React Native, Expo, Firebase, Visual Studio Code), методичні вказівки до виконання кваліфікаційної роботи бакалавра, офіційна документація до використаних технологій (React Native Documentation, Firebase Documentation), наукові публікації з теми дослідження.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *У роботі представлено аналіз сучасних рішень для планування завдань та сформовано вимоги до розроблюваного додатку. Обґрунтовано вибір технологічного стеку (React Native, Firebase), описано проектування архітектури системи з використанням UML-діаграм. Детально висвітлено етапи практичної реалізації, тестування функціоналу та забезпечення захисту даних. Окрему увагу приділено механізму синхронізації інформації між пристроями через хмарні сервіси Firebase.*
5. Перелік графічного матеріалу: *16 рисунків, 4 таблиці, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Суринович О. М.*		
Специфікація вимог до розробленої системи	*Суринович О. М.*		
Розробка об'єкта проектування	*Суринович О. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,01 %		
Академічна доброчесність	*Суринович О. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Богдан ВИРИМЧУК

Керівник кваліфікаційної роботи _____ Олена СУРИНОВИЧ

АНОТАЦІЯ

Виримчук Б. І. Розробка мобільного додатку «Персональний менеджер завдань» з використанням React Native та Firebase. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз сучасного стану проблем управління часом, вивчено наявні програмні рішення для планування справ та визначено цілі для створення власної мобільної програми.

У другому розділі здійснено специфікацію вимог до системи, вибрано технологічний стек (React Native, Firebase, Expo) та спроектовано архітектурні рішення з використанням UML-діаграм.

У третьому розділі представлено практичну реалізацію додатку, включаючи розробку інтерфейсу, інтеграцію з хмарними сервісами, тестування та забезпечення безпеки даних. У висновках підведено підсумки роботи, підтверджено ефективність вибраних технологій та визначено шляхи подальшого покращення проекту.

Ключові слова: мобільний додаток, планувальник завдань, React Native, Firebase, Expo, база даних, хмарні технології, захист інформації.

ABSTRACT

Vyrymchuk B. Development of a Mobile Application "Personal Task Manager" using React Native and Firebase. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter analyzes the current state of time management problems, examines existing software solutions for case planning, and defines the goals for creating its own mobile application.

The second section specifies the requirements for the system, selects the technology stack (React Native, Firebase, Expo) and designs architectural solutions using UML diagrams.

The third section presents the practical implementation of the application, including interface development, integration with cloud services, testing, and data security. The conclusion summarizes the results of the work, confirms the effectiveness of the selected technologies and identifies ways to further improve the project.

Keywords: mobile application, task scheduler, React Native, Firebase, Expo, database, cloud technologies, information security.

ЗМІСТ

ВСТУП

Актуальність теми. Робота зумовлена нагальною потребою у дієвих інструментах тайм-менеджменту, що особливо важливо в умовах віддаленої роботи, інформаційного перевантаження та загальної непередбачуваності сьогодення.

Мета роботи – створення мобільного додатку «Персональний менеджер завдань», використовуючи передові технології React Native та Firebase, задля спрощення організації щоденних справ та збільшення продуктивності користувачів.

Об’єкт роботи – процес створення мобільного додатку, який призначений для планування різноманітних завдань.

Предмет роботи – методи та засоби його реалізації з інтеграцією хмарних сервісів.

Для досягнення поставленої мети були поставлені наступні завдання:

- провести аналіз актуального стану проблеми та наявних рішень;
- визначити необхідні критерії для програмного забезпечення;
- обрати оптимальні інструменти для досягнення поставленої мети;
- реалізувати практичну складову об’єкта проєктування;
- виконати тестування, налаштування системи;
- створити структуру і підключити до бази даних;
- реалізувати синхронізацію даних між пристроями;
- забезпечити безпеку інформаційно-обчислювальної системи.

Апробація результатів дослідження (додаток А): Виримчук Б. І. Суринович О. М. Інструмент Expo – оптимізація процесу створення мобільних додатків засобами React Native. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 170-174.

РОЗДІЛ 1
АНАЛІЗ СФЕРИ ДОДАТКІВ ПЛАНЕРІВ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Через швидкий ритм повсякдення, завантаженість та нервову напругу, втрата контролю над особистим графіком стала звичною буденністю, тому мобільні додатки-планувальники особистого часу стали незамінним помічником для структурування дня та власних справ. Вони знаходять своє застосування як для індивідуальних користувачів, так і для команд, надаючи функціонал, такий як: перелік завдань, сповіщення та спільну роботу над проєктами. Розвиток віддаленої зайнятості значно збільшив їх популярність, адже люди прагнуть знаходити ефективні шляхи досягнення результативності у власному житті та цифровому середовищі.

Застосування мобільних додатків-планерів суттєво збільшилося впродовж останніх років, це пов’язано з попитом у інструментах самоорганізації, необхідних в умовах стрімких змін у житті та роботі. Ця тенденція певною мірою пояснюється довготривалими наслідками пандемії COVID-19, яка вразила кожен куточок планети, змусивши компанії та працівників адаптуватися до віддаленої праці. Перехід на дистанційний формат роботи акцентував важливість ефективного керування часом та завданнями, оскільки кордони між роботою та особистим життям часто розмивалися, це створювало потребу у цифрових рішеннях для планування. В Україні цей процес ускладнився внаслідок повномасштабного вторгнення, яке розв’язала країна-агресор у 2022 році. Постійні обстріли, локдауни та перебої з електропостачанням змусили українців не тільки адаптуватися до винятково віддаленої роботи, а й оптимізувати щоденні справи в умовах графіків екстрених і погодинних відключень електроенергії.

Ці виклики підкреслили потребу у гнучких інструментах, що допомагають планувати робочий і особистий час з урахуванням непередбачуваних обставин.

Додатки для планування подарували користувачам гнучкість та можливість швидкого реагування на зміни. Такі програми значно полегшили процес перенесення зустрічей, зміни пріоритетів та координації командної роботи, зменшуючи навантаження на користувачів і дозволяючи зберігати продуктивність навіть у нестабільних умовах. Інтеграція з локальними сервісами, наприклад, з платформами сповіщення про аварійні відключення чи повітряні тривоги, перетворила ці додатки на справжніх асистентів у кризових ситуаціях, надаючи можливість не лише планувати графік роботи, але й оперативно ухвалювати рішення з огляду на зовнішні обставини.

Завдяки розширеним функціям, планувальники містять календарі з підтримкою повторюваних подій, трекери звичок, інтеграцію з поштовими клієнтами, системами відеозв'язку, а також можливість командної роботи з підтримкою коментарів, тегів, дедлайнів та статусів. Поодинокі планувальники вже включають елементи штучного інтелекту, що автоматично рекомендують зміни в графіку на основі поведінки користувача або навколишніх обставин, дають змогу аналізувати продуктивність користувача, будуючи графіки виконання задач, рівня завантаження або дотримання цілей.

Особливої популярності набули вузькоспеціалізовані платформи, до-прикладу Notion (рис. 1.1). Він є гібридом планера, бази знань, таск-менеджера і текстового редактора. Дає змогу формувати структуровані робочі середовища, які можна підлаштувати під індивідуальні потреби чи потреби команди. Функціональність сервісу дозволяє створювати бази даних, переліки справ, нотатки, вікі-документацію, інтерактивні шаблони та багато іншого. Це робить платформу надзвичайно універсальною для різних завдань – від особистої організації до менеджменту великими колективними проєктами. Але, попри велику кількість можливостей, саме ця багатофункціональність ускладнює розуміння інтерфейсу. Notion не є очевидним навіть для користувачів з досвідом, і щоб повністю опанувати платформу, потрібен час і практика. Водночас, цей інструмент є чудовим для командної роботи, адже підтримує створення спільних робочих просторів, де гнучко налаштовуються структури сторінок, інформація

організовується за темами, розділами або цілими проєктами. Система дає змогу призначати користувачам різний рівень доступу: від можливості редагування до перегляду з обмеженнями «тільки читання», що забезпечує ефективний розподіл ролей та контролює зміни. Слід звернути увагу на функцію гостьового доступу: користувачі, які не мають доступу до простору або обмежені правами, все одно можуть переглядати сторінки за посиланнями для публічного перегляду. Це полегшує обмін інформацією з тими, хто не входить до команди: з партнерами, клієнтами чи сторонніми учасниками проєкту. У сукупності це робить Notion потужним середовищем для організації персональних даних і спільної командної роботи на всіх етапах реалізації проєктів.

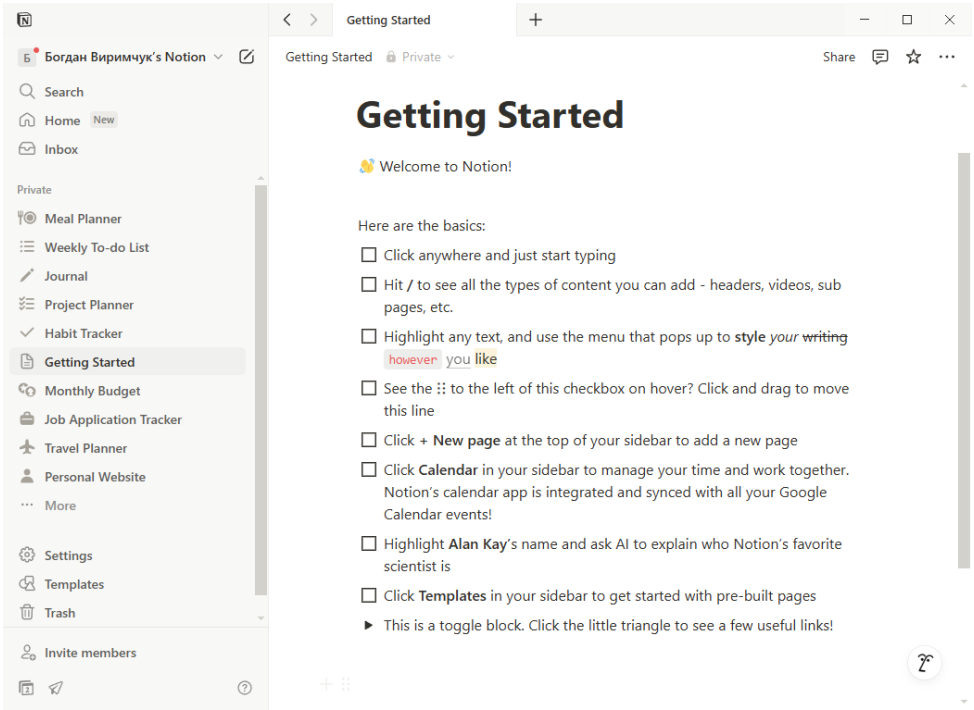

Рисунок 1.1 – Декстопна домашня сторінка планеру Notion [1]

TickTick (рис. 1.2), багатогранний додаток для планування, об'єднує в собі функції списку справ, календаря та системи управління проєктами. Основною перевагою є його кросплатформність. Синхронізація між мобільними та десктопними версіями. Водночас, безкоштовна версія має помітні обмеження, як-от ліміт на кількість списків та відсутність нагадувань.

Рисунок 1.2 – Домашня сторінка мобільного додатку TickTick [2]

Ключовою особливістю TickTick є його універсальність, що робить його привабливим варіантом для користувачів, які потребують комплексного підходу до організації часу у професійній та особистій сферах.

Trello (рис. 1.3), застосовує систему канбан-дошок, щоб візуалізувати завдання. Найголовніша його перевага – зрозумілий і інтуїтивно зрозумілий інтерфейс, що дає змогу миттєво зорієнтуватися в системі навіть новачкам у сфері проєктного менеджменту. Завдяки цьому інструмент ефективно підходить для управління проєктами будь-якого розміру – від особистих планів до масштабних командних починань. Фундаментом системи є інтерактивні дошки, списки та картки, які відображають завдання, етапи їх виконання та інші ключові компоненти робочого процесу. Кожен з цих елементів можна гнучко адаптувати до індивідуальних вимог користувача чи команди: додавати кінцеві терміни, прикріплювати файли, змінювати статуси та визначати відповідальних. Такі можливості дають змогу створити комфортне цифрове середовище для контролю прогресу та узгодження дій між усіма учасниками.

Рисунок 1.3 – Домашня сторінка мобільного додатку Trello [3]

Слід, проте, враховувати, що безкоштовна версія сервісу містить певні обмеження, які можуть позначитися на ефективності використання у великих або паралельних проєктах. Зокрема, доступна лише одна командна дошка, що суттєво ускладнює організацію роботи над кількома проєктами одночасно, змушуючи користувача або поступатися структурованістю, або переходити на платний тариф. Крім того, у безкоштовній версії відсутній модуль аналітики, що міг би надавати цінну статистичну інформацію про виконання завдань: наприклад, відсоток завершення проєкту, тривалість затримок, поради щодо перерозподілу обов'язків або реорганізації структури задач. Відсутність таких інструментів знижує ефективність управління ресурсами та часовими рамками, особливо в умовах складних або термінових проєктів.

Todoist (рис. 1.4) пропонує збалансоване рішення для тих, хто шукає ефективний інструмент управління завданнями без надмірної складності.

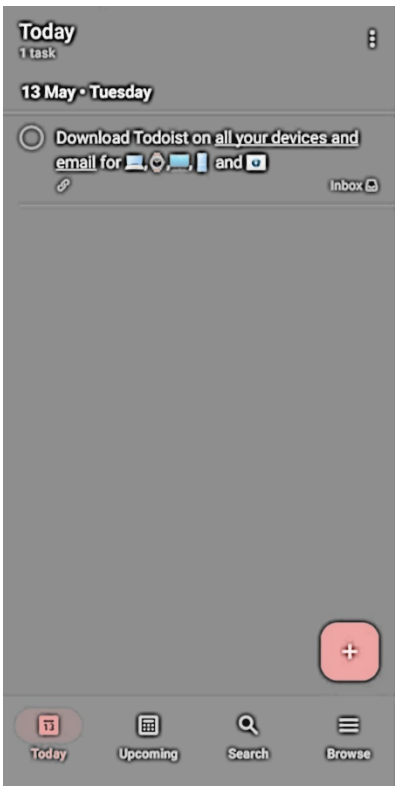

Рисунок 1.4 – Домашня сторінка мобільного додатку Todoist [4]

Але маючи достатньо функцій для серйозного планування. Головна його перевага – лаконічний, мінімалістичний вигляд, що дозволяє швидко організовувати справи, не переобтяжуючи. Однак, безкоштовна версія має кілька обмежень, зокрема: цілковита відсутність сповіщень, а також обмежена кількість фільтрів і міток.

Планувальники ефективно вирішують декілька нагальних проблем сучасного користувача. В умовах багатозадачності, інформаційного перевантаження та нестабільності, люди часто втрачають фокус, відчувають тривожність або не можуть зібратися з думками для досягнення особистих цілей. Саме тут цифрові планувальники надають структурованість, упорядковують потік справ і забезпечують відчуття контролю. Завдяки розбивці великих задач на під цілі, можливості фіксувати прогрес, наявності щоденних оглядів та нагадувань, додатки створюють середовище, у якому легше досягати бажаного результату. У складні періоди, зокрема під час війни, це сприяє зниженню

психологічного навантаження, формуванню стабільних звичок та підтримці емоційної рівноваги.

Загальносвітовий тренд на цифрове планування також посилюється впровадження нових технологій, таких як: інтеграція штучного інтелекту (ШІ) та хмарних обчислень. Це суттєво розширює функціонал та привабливість мобільних додатків для планування особистого часу, роблячи їх більш персоналізованими, комфортними та пристосованими до вимог користувачів, що сприяє зростанню їхньої популярності. Штучний інтелект дозволяє програмам аналізувати дії користувачів, їхні звички та пріоритети, формуючи персоналізовані рекомендації, які оптимізують розподіл часу та задач. Скажімо, ШІ може визначати найкращий час для виконання певних справ, ґрунтуючись на попередній діяльності, наприклад, призначати робочі зустрічі у періоди найбільшої продуктивності або нагадувати про відпочинок, якщо користувач перевантажений. Хмарні обчислення, у свою чергу, забезпечують миттєву синхронізацію даних між різними пристроями, що дає змогу користувачам вільно перемикатися між смартфоном, комп’ютером чи планшетом, зберігаючи актуальність розкладів та списків справ. Хмарні обчислення полегшують командну роботу, дозволяючи ділитись задачами та синхронізувати плани між колегами чи членами сім’ї, що особливо ціно для координації в умовах віддаленої роботи або сімейних обов’язків.

Сфера мобільних додатків для персонального планування демонструє значний потенціал для розвитку, це підтверджується дослідженнями ринку. За даними статті Verified Market Research [5], 2024 року обсяг ринку цифрових планерів становив 1,2 мільярда доларів США, та прогнозується збільшення до 3,5 мільярда доларів до 2033 року. Інше дослідження, проведене у статті Market Research Intellect [6], демонструє оцінку в 1,5 мільярда доларів у 2024 році, з прогнозом зростання до 3,2 мільярда доларів до 2033 року. На рисунках 1.5 та 1.6 графічно зображені стовбчасті діаграми досліджень із статей.

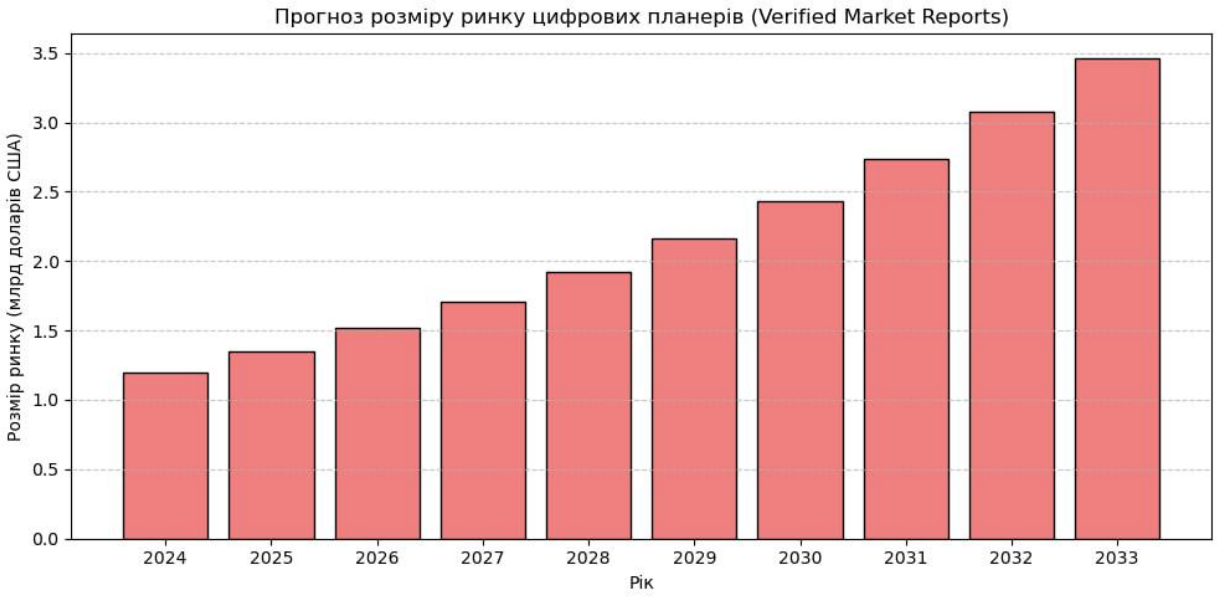

Рисунок 1.5 – Діаграма прогнозу зростання ринку за версією статті Verified Market Research [5]

Варто відмітити, що у першій діаграмі (рис .1.5) демонструється стійке зростання, також помітно прискорення в період 2032–2033 років, саме тоді розмір ринку переважає 3 мільярди.

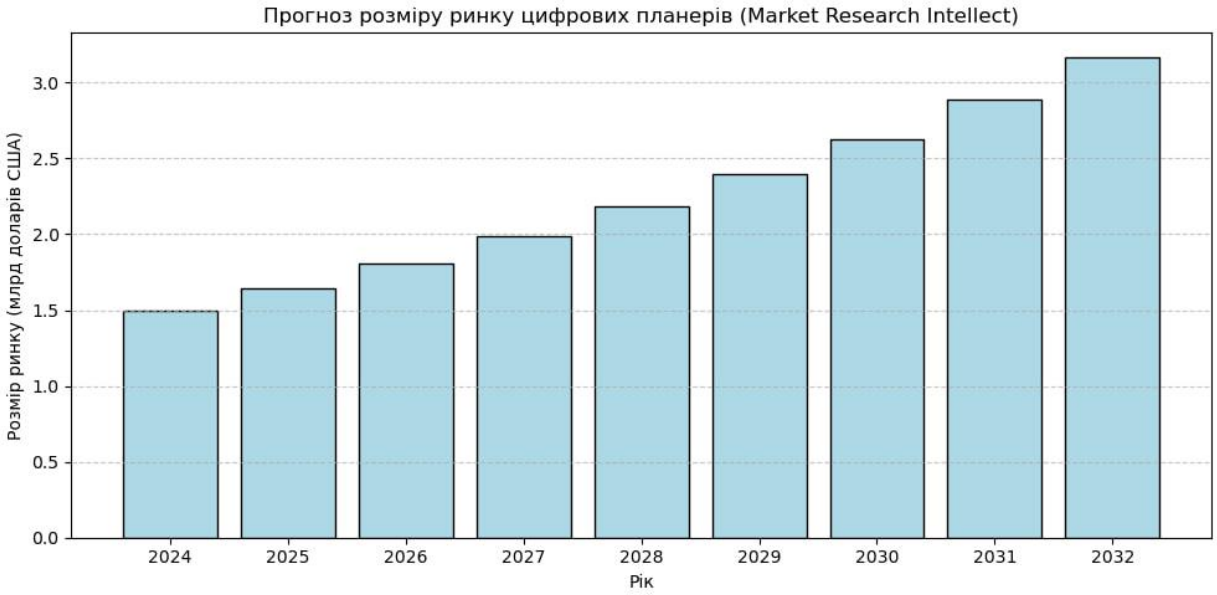

Рисунок 1.6 – Діаграма прогнозу зростання ринку за версією статті Market Research Intellect [6]

На відміну від першої діаграми (рис. 1.5), друга (рис. 1.6) не демонструє прискорення, але все ще є стійко зростаючою.

Таке зростання пояснюється кількома ключовими факторами. Перш за все, постійно зростаюча потреба в ефективному управлінні часом та завданнями серед широкого кола користувачів, зокрема студентів, професіоналів та бізнес-структур. Таке сприяння збільшує популярність таких додатків. По-друге, активне впровадження цифрових технологій та збільшення використання смартфонів і хмарних сервісів полегшують доступ до таких інструментів. Крім того, інноваційні рішення, такі як інтеграція штучного інтелекту для персоналізованих рекомендацій, підвищують привабливість цих додатків для користувачів. Ринок подібних програм активно розширюється та розвивається. Хоча методології оцінки можуть відрізнятися, подібні прогнози підкреслюють значний потенціал. Це підтримується вклиненням смартфонів у повсякденне життя людей, а також потребую у самоконтролі та реалізації за для підвищення продуктивності, зокрема серед тих, хто працює дистанційно.

1.2 Постановка завдань на кваліфікаційну роботу бакалавра

У процесі виконання кваліфікаційної роботи бакалавра слід вирішити декілька важливих задач, що ведуть до створення інформаційно-комп’ютерної системи із застосуванням новітніх технологій. Головні завдання охоплюють:

- провести аналіз актуального стану проблеми та наявних рішень;
- визначити необхідні критерії для програмного забезпечення;
- обрати оптимальні інструменти для досягнення поставленої мети;
- реалізувати практичну складову об’єкта проєктування;
- виконати тестування, налаштування системи;
- створити структуру і підключити до бази даних;
- реалізувати синхронізацію даних між пристроями;
- забезпечити безпеку інформаційно-обчислювальної системи.

Висновки до розділу 1

Програми для особистого планування стали важливими у сучасному світі, допомагаючи упорядковувати справи в умовах стрімкого темпу життя, інформаційного перевантаження та нестабільності – зокрема під час віддаленої праці, пандемії чи війни. Вони дають змогу структурувати день, координувати завдання та пристосовуватися до перебоїв з електропостачанням чи повітряних тривог. Частина можливостей вимагає підписки, але базовий функціонал уже є корисним. Завдяки хмарним сервісам і штучному інтелекту такі додатки стають персоналізованими, зручними й популярними. Ринок активно збільшується –це стимулюється поширенням смартфонів і попитом на продуктивні рішення.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Моделювання програмних систем – це побудова моделей, що демонструють функціонування системи, її складові частини та способи взаємодії між ними. Це допомагає розробникам планувати та керувати розробкою проєктів, мінімізуючи ймовірність виникнення помилок. Моделювання є невід'ємною частиною розробки програмного забезпечення, що дає можливість формувати абстрактні версії систем задля глибшого розуміння їх будови, функціонування та взаємодії. Цей етап сприяє візуалізації компонентів програм, роблячи простішим планування, створення та управління цими системами. Завдяки моделям можна завчасно виявити можливі проблеми, знизити витрати на виправлення помилок та підвищити результативність розробки. Як приклад, вони можуть допомогти у представленні того, як дані опрацьовуються системою або яким чином користувач взаємодіє з інтерфейсом.

Існує доволі багато типів моделювання, кожен з який фокусується на різни аспектах систем, ось декілька основних з них:

– функціональне моделювання, зосереджується на описі того, що робить система, її функціях і процесах. Воно візуалізує перетворення вхідних даних на вихідні в системі. Цей підхід до моделювання сприяє глибшому розумінню та фіксації функціональних вимог, підтверджуючи відповідність системи всім необхідним задачам;

– архітектурне моделювання, акцентується на структурі системи, що включає в себе складові елементи, зв'язки між ними та спосіб їхньої організації. Воно візуалізує взаємодію складових частин системи, наприклад, модулів чи сервісів. Застосування такої моделі порібна для розуміння загальної архітектури системи, а також для забезпечення її здатності до масштабування та підтримки;

– поведінкове моделювання, ілюструє зміну поведінки системи з плином часу, зокрема реагуючи на певні події або зовнішні дані. Це відображає динамічні характеристики системи, зокрема робочі процеси, зміни станів та способи взаємодії між її елементами;

– моделювання, орієнтоване на дані, зосереджується на шляху руху даних через систему та їхніх перетвореннях. Це критично важливо для систем, де обробка даних відіграє ключову роль, зокрема для систем управління запасами або аналізу даних.

Моделювання програмних систем, являє собою важливим інструментом для розуміння сутності та контролю над складними програмними системами, що еволюціонують в нинішньому світі, де ефективне управління стає нагальною потребою. Цей процес охоплює низку різновидів моделей, на зразок функціональних, архітектурних, поведінкових, орієнтованих на дані, подієво-орієнтованих моделей та моделей процесів розробки, кожна з яких зосереджується на конкретних аспектах системи, сприяючи її результативному проектуванню та впровадженню. Для візуалізації цих аспектів залучаються різні методики та засоби, зокрема UML-діаграми, котрі пропонують стандартизовані способи репрезентації моделей, полегшуючи комунікацію між розробниками та учасниками [7].

Після ретельного вивчення наявних методів моделювання інформаційних систем було прийнято рішення на користь архітектурної моделі. Цей вибір обумовлений її здатністю до наочного відображення структури та взаємозв’язків між ключовими складовими програмного забезпечення. Завдяки цьому підходу полегшується розуміння внутрішньої організації застосунку, що позитивно впливає як на процес розробки, так і на подальше обслуговування системи. В рамках цієї архітектурної моделі велику увагу зосереджено на використанні діаграми класів, яка є важливим інструментом в об’єктно-орієнтованому аналізі та проектуванні. Вона дає змогу представити систему у вигляді набору взаємопов’язаних класів, кожен з яких визначається своїми властивостями, методами та зв’язками з іншими класами. Завдяки чіткому представленню

взаємодій між компонентами, діаграма класів сприяє легкому розумінню логіки роботи застосунку, навіть за наявності складної бізнес-логіки. Її легкість у побудові та одночасно висока інформативність роблять її цінним інструментом як для розробників, так і для аналітиків. До того ж, діаграма класів дозволяє виявити можливі недоліки у структурі ще на стадії проектування, що мінімізує ризики помилок при подальшій реалізації та підвищує загальну якість програмного продукту.

На рисунку 2.1 зображена UML-діаграма класів, у ній виділені основні компоненти, класи мобільного додатку планування.

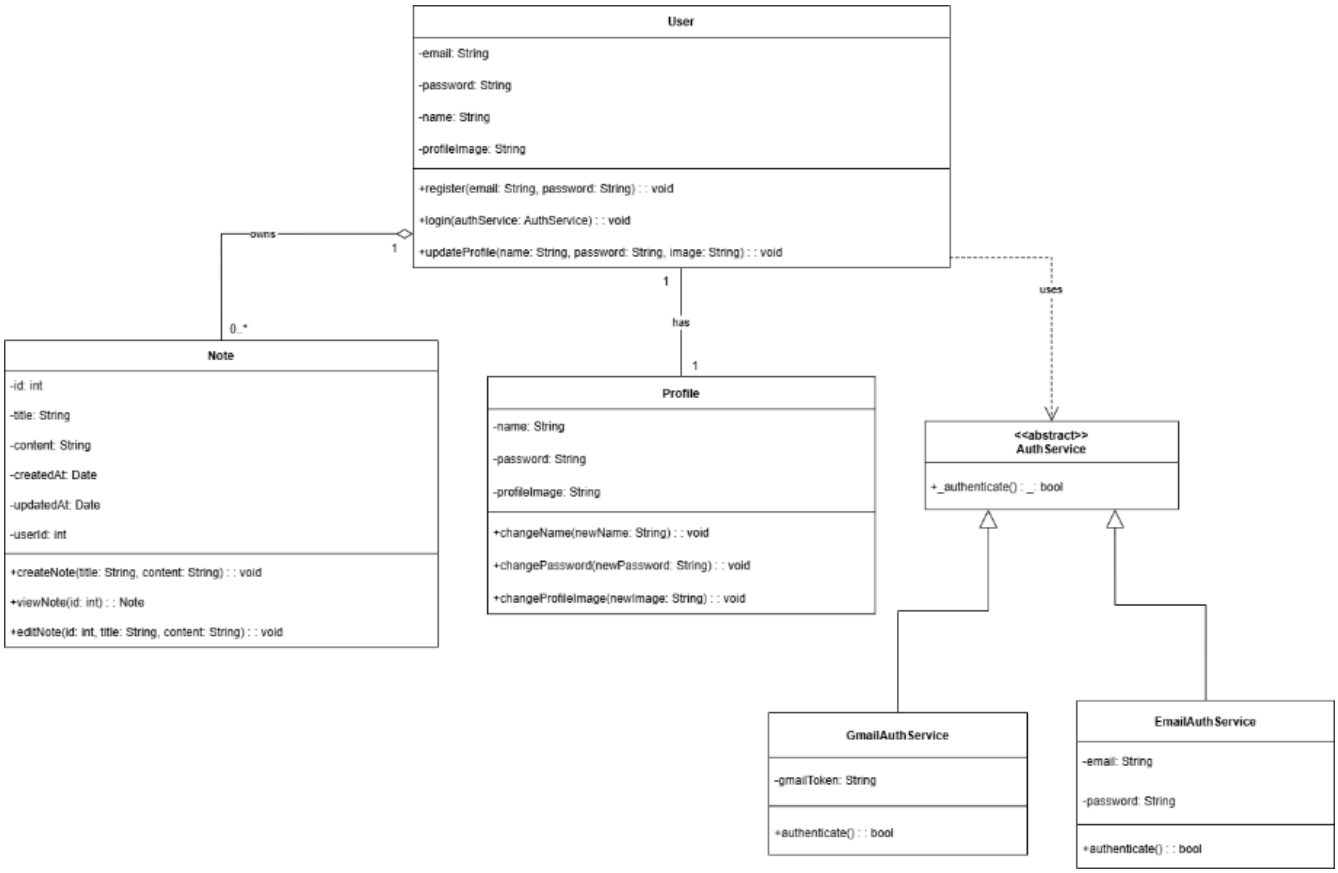

Рисунок 2.1 – UML-діаграма класів мобільного додатку планування

Отож, діаграма класів, описує побудову мобільного додатку-планувальника. Діаграма складається з класів, зокрема User, Note, Profile, AuthService, GmailAuthService та EmailAuthService, віддзеркалюють ключові компоненти системи, задіяні у діях користувача, нотатками та авторизацією.

Основні дані тут, це атрибути, як-от ім’я, пароль, фото профілю та вміст нотаток, а також зв’язки між цими класами.

Безсумнівно, додаток передбачає збереження даних користувачів у хмарному сховищі, які планують ним користуватися, тому клас User детально описується в діаграмі класів, він містить у собі базову інформацію: електронну адресу, пароль, назву профілю та фото профілю. Ці дані дозволяють користувачам реєструватися, входити та змінювати власний профіль. Клас Note є центральним для функціональності планувальника, зберігаючи відомості про нотатки, включно з ідентифікатором, заголовком, змістом та датою створення, також ідентифікатором користувача, якому належить нотатка. Це забезпечує можливість створювати, переглядати, сортувати та коригувати нотатки.

Клас Profile підтягує інформацію про користувача, даючи змогу змінювати ім’я, пароль та зображення, разом з методами для їх зміни, що дозволяє користувачу адаптувати свій особистий профіль. Авторизація в додатку реалізується через абстрактний клас AuthService, котрий містить метод аутентифікації, а його конкретні реалізації такі як, GmailAuthService та EmailAuthService, вони забезпечують авторизацію за допомогою Google або електронної пошти з паролем відповідно. Ключові дані тут – токен для Gmail або комбінація email та паролю.

Зв’язки між класами є надзвичайно важливими в структурі додатку. Наприклад, зв’язок між User та Note реалізується як асоціація «один до багатьох», де один користувач може мати кілька нотаток, що відтворює логіку планувальника. Зв’язок між User та Profile є асоціацією «один до одного», що вказує на те, що кожен користувач має тільки один профіль. Залежність User від AuthService показує, що клас користувача використовує сервіси авторизації для входу. Спадкування від AuthService до GmailAuthService і EmailAuthService ілюструє різні способи реалізації авторизації.

Головним акцентом цієї діаграми є представлення структури додатку у наочній формі. Вона допомагає зрозуміти архітектуру системи та сприяє її практичному втіленню, наприклад, при створені навігації та структури розробки.

Інтуїтивно зрозуміле представлення у вигляді діаграми дає змогу презентувати додаток третій стороні, також подібна діаграма корисна під час тестування на ранніх стадіях розробки, задля усунення можливих недоліків, забезпечуючи ефективну взаємодію у процесі реалізації проєкту.

Також, на мій погляд, варто звернути увагу до функціонального моделювання. Цей тип моделей, переважно, застосовують для опису функціоналу розробки, та застосовують, для цього, діаграму варіантів використання (Use Case Diagram), вона представлена на рисунку 2.2, рисунок ілюструє взаємодію користувача з функціоналом розробки.

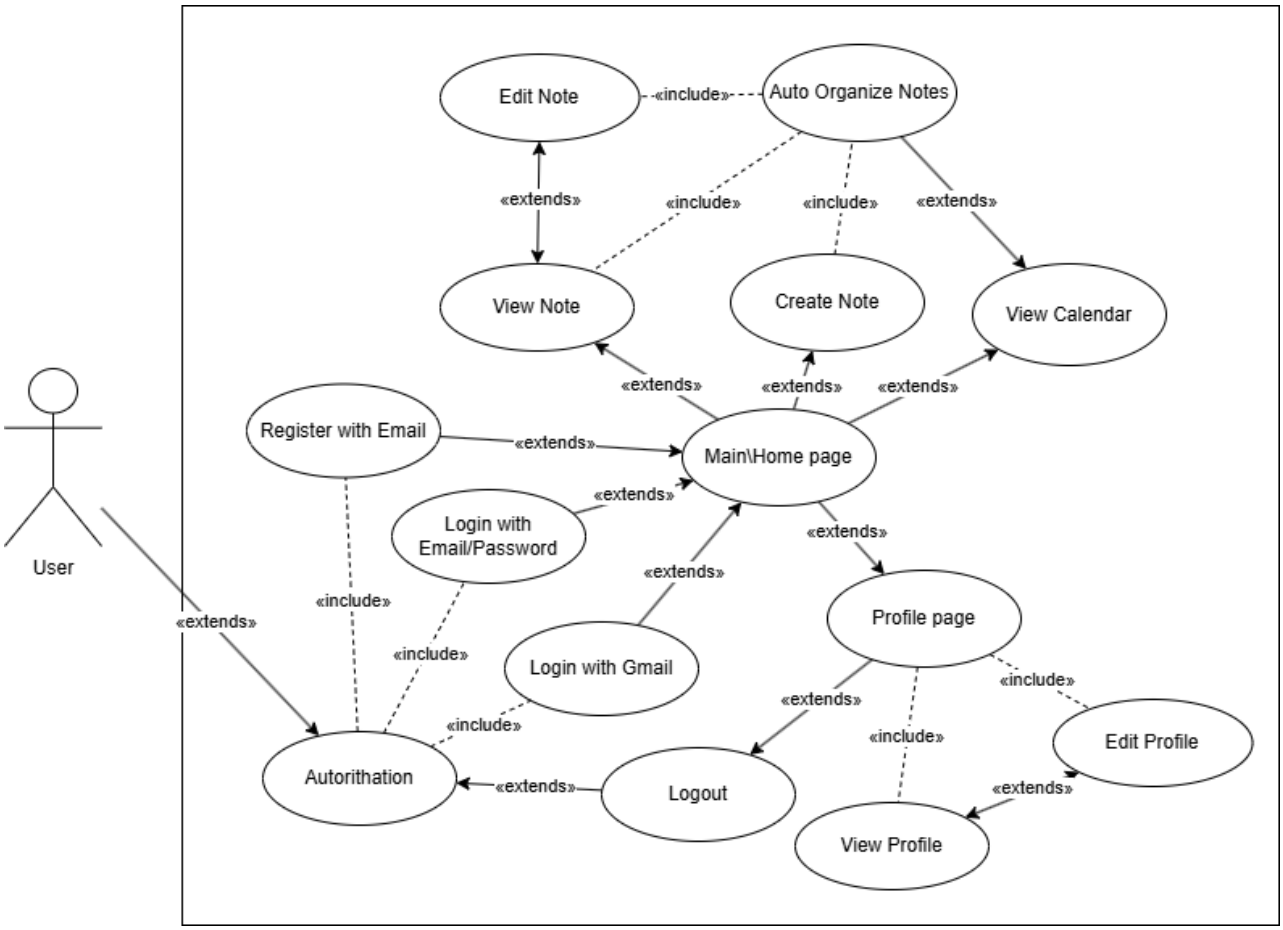

Рисунок 2.2 – UML-діаграма варіантів мобільного додатку планування

Діаграма зображає використання мобільного застосунку-планувальника, демонструючи взаємодію користувача з доступним функціоналом. Система

оперує одним суб’єктом – Користувачем (User), який взаємодіє з різноманітними можливостями додатку для керування завданнями та нотатками.

Користувачеві доступний ряд функцій у системі. Він може зареєструватися через електронну пошту, використовуючи функцію Register with Email, що дозволяє створити профіль для подальшого користування додатком. Для авторизації передбачено два способи: Login with Gmail для входу через обліковий запис Google та Login with Email/Password для входу, використовуючи email та пароль. Обидва способи входу є частиною загального процесу авторизації, який гарантує захищений доступ до персональних даних та можливостей застосунку.

Головною функцією додатку є робота з нотатками. Користувач може створювати нові нотатки через Create Note, переглядати їх за допомогою View Note, змінювати за потреби через Edit Note. На головній сторінці Main\Home page, відображається список усіх нотаток користувача. Під час створення, перегляду чи редагування нотаток автоматично активується функція Auto Organize Notes, що організовує нотатки за певними параметрами, наприклад, за датами чи категоріями.

Користувач має можливість керувати своїм профілем. Функція View Profile дає змогу переглядати особисту інформацію, таку як ім’я, електронна адреса та зображення профілю, в свою чергу Edit Profile дозволяє змінювати ці дані, наприклад, змінювати пароль або оновлювати зображення. Для завершення сеансу роботи з додатком користувач може вийти з облікового запису, використовуючи функцію Logout.

Окремо на діаграмі зображено процес авторизації. Аналіз типових рішень підкреслює важливість наявності облікового запису для персоналізації та безпеки даних. Тому в додатку реалізована авторизація, що включає як реєстрацію, так і два варіанти входу. Це гарантує, що тільки авторизовані користувачі мають доступ до своїх нотаток, профілю та інших функцій.

Отже, діаграма візуалізує логічну організацію функцій у мобільному застосунку-планувальнику. Вона чітко демонструє можливості користувача,

включаючи роботу з нотатками, керування профілем, перегляд календаря та захищену авторизацію. Діаграма представляє основні процеси, які відбуваються в системі, забезпечуючи зрозуміле представлення та опис функціоналу.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

За для ефективної розробки мобільного додатку-планера потрібно зробити обґрунтований вибір інструментів та методів, що забезпечать швидку розробку з мінімальними витратами на обслуговування та розгортання проєкту, а також зручність самої реалізації розробки.

За фундамент розробки було взято фреймворк React Native. Цей фреймворк розроблений компанією Meta (Facebook) для створення мобільних додатків з використанням JavaScript та бібліотеки React. Додаток описується через React-компоненти, які відображаються за допомогою елементів інтерфейсу платформи Native (View, Text, Image). Спочатку, архітектура React Native використовувала асинхронний «міст» між JavaScript-рушієм і нативним кодом. Це дозволяло виконувати виклики у фоновому режимі, не блокуючи головний потік. Хоча забезпечувало неблокуюче оновлення інтерфейсу, але додаткові витрати на серіалізацію викликів збільшувались при частих оновленнях. У новій архітектурі React Native традиційний міст відсутній: натомість застосовується JavaScript Interface (JSI), який дозволяє напряму викликати C++-об’єкти з JavaScript без серіалізації. Завдяки цьому зменшився час на взаємодію з нативним рівнем, а також стала доступною підтримка сучасних функцій React (Concurrent Mode, Suspense) [8]. Отож, React Native поєднує переваги реактивної моделі React з нативною продуктивністю інтерфейсу.

Переваги цього фреймворку кореняться у компонентній архітектурі та розмаїтті модулів, що гарантує швидкий старт та значний вибір сторонніх бібліотек, таких як: Expo, React Navigation, Reanimated. Додатки, побудовані з використанням React Native, візуалізуються нативно, тобто їх інтерфейс та

поведінка повністю відповідають звичкам користувачів на кожній платформі. React Native демонструє рівень продуктивності, зіставний з нативними застосунками у типових сценаріях використання. Звісно, деякі обмеження архітектури можуть проявлятися при роботі зі складними графічними елементами або інтенсивними анімаціями через додатковий «прошарок» між JS та нативними API, через це, можливі незначні затримки. Окрім того, деякі платформо-специфічні можливості, скажімо: API камер або Bluetooth, можуть потребувати власних нативних модулів, що збільшує загальну складність розробки. Іншим аспектом, що потребує уваги, є узгодження стилів інтерфейсу, через унікальні дизайнерські особливості кожної платформи виникає потреба виділяти час на налаштування розмітки та стилів для досягнення єдиного зовнішнього вигляду.

Прямими аналогами React Native можна вважати: Flutter від Google, Xamarin від Microsoft або Kotlin Multiplatform від JetBrains. Flutter використовує інакший підхід. Замість нативних UI, Flutter малює власний інтерфейс, застосовуючи графічний двигун Skia. Він розроблений на Dart і в фінальних версіях компілюється в машинний код на Android/iOS, забезпечуючи високу продуктивність і однаковий вигляд UI на всіх платформах [9]. Якщо порівнювати з React Native, Flutter показує більш однорідний інтерфейс і часто швидшу анімацію, проте за рахунок більшого розміру бінарних файлів і необхідності володіння Dart. Навпаки, Xamarin дає можливість розробляти на C#/.NET і теж демонструє майже нативну продуктивність, завдяки можливості спільного використання ~90 % коду між iOS і Android та використанню нативних компонентів. Але, програми на Xamarin, як правило, більші за розміром, через .NET-движок, і менш поширені серед розробників, ніж рішення на JavaScript або Dart [10]. Підсумовуючи, React Native є добре збалансованим інструментом з великою спільнотою та стрімким розвитком, в той час як Flutter гарантує кращу продуктивність у складних UI, а Xamarin підійде тим, хто надає перевагу екосистемі .NET.

У таблиці 2.1 представлено наглядне порівняння характеристик між цими фреймворками.

Таблиця 2.1 – Порівняння параметрів, вище зазначених фреймворків

Параметр	React Native	Flutter	Xamarin
Мова	JavaScript/TypeScript	Dart	C# (.NET)
Рендеринг UI	Використання нативних View (UIKit/Android View)	Власний рушій Skia (малиї свій Canvas)	Нативні елементи через обгортки (.NET)
Компіліція	JS-інтерпретатор (JIT) в розробці	Ahead-of-Time компіляція в машинний код	AOT/Just-In-Time через Mono/.NET
Продуктивність	Близька до нативної (можливі затримки через міст)	Дуже висока (60+ FPS) завдяки прямій графіці	Інтерпретовані елементи можуть сповільнюватися
Розмір застосунку	Помірний (залежить від js-рушія); зазвичай менший Flutter	Досить великий (великий C++ рушій і Dart VM)	Зазвичай більший (включно з Mono-рушієм)
Спільнота та екосистема	Велика спільнота, багато бібліотек і готових модулів	Стрімко зростає, активна підтримка Google	Менша спільнота; сильна підтримка Microsoft/Visual Studio

Firebase – це хмарна платформа від Google, призначена для розробки мобільних та веб-застосунків, що реалізує модель Backend-as-a-Service, надаючи готові бекенд-сервіси. Архітектура Firebase побудована на основі інфраструктури Google Cloud і пропонує ряд сервісів: NoSQL бази даних (Realtime Database, Firestore), що підтримують синхронізацію в реальному часі, аутентифікацію користувачів через технологію Firebase Auth, хмарне зберігання файлів Cloud Storage, серверні функції Cloud Functions, аналітику, систему обміну повідомленнями Cloud Messaging та інші інструменти. Розробник може безпосередньо використовувати клієнтські SDK для взаємодії з цими сервісами зі свого мобільного застосунку, не потребуючи розробки власного сервера. Скажімо, інтегрувавши бібліотеку Firestore, розробник здатний миттєво зчитувати та записувати дані у хмару, зокрема з можливістю офлайн-кешування [11].

Основні плюси Firebase – блискавичний старт проекту (розробник позбавлений потреби в налаштуванні серверів), широкий спектр інтегрованих сервісів та бездоганна інтеграція, як-от налаштування аутентифікації через Google/Facebook/Apple, що працює «з коробки». Також варто відзначити високу масштабованість: сервіси Google автоматично адаптуються до навантаження. Firebase безкоштовно надає аналітику та звіти про збої в роботі додатку, що високо цінується багатьма командами. Серед недоліків: закритість рішення та прив'язка до екосистеми Google, вихідний код бекенду недоступний, що унеможливлює внесення змін на серверному боці. Окрім цього, певні обмеження Firestore/Realtime DB, як-от відсутність складних JOIN-запитів, ускладнюють реалізацію нетривіальної логіки даних, змушуючи вдаватися до альтернативних підходів (колекції, ключові фільтри). Іншим мінусом є нестабільна модель оплати: після безкоштовного рівня тариф «Blaze» функціонує за принципом «плати за використання», що іноді може призвести до неочікуваних витрат при різкому збільшенні трафіку. Firebase використовує лише Google Cloud як постачальника, обмежуючи варіанти розгортання одним вендором.

Серед прямих аналогів Firebase часто згадують AWS Amplify (Amazon) та Backendless. AWS Amplify – це хмарна платформа від Amazon, що об'єднує служби AWS в аналогічну екосистему: сервіси ідентифікації користувачів Cognito, бази даних DynamoDB, серверні функції Lambda, сховище S3, GraphQL API (AppSync) [11]. Amplify пропонує CLI та консоль для створення бекенду, а також автоматизоване розгортання через CI/CD. Сильними сторонами Amplify є глибока інтеграція з AWS (масштабованість, безпека) та підтримка різноманітних фреймворків, як-от React, Flutter, Vue, проте система може бути трохи складнішою для початку, а сам Amplify прив'язаний до AWS. Backendless – це low-code/no-code платформа з візуальним конструктором, що надає бекенд для мобільних та веб-застосунків. Вона включає в себе реальну базу даних NoSQL/SQL, управління користувачами, генератор REST API, push-сповіщення та інше [12]. Backendless можна розгорнути як у хмарі компанії, так і на власних серверах (on-premise), що знижує ризик залежності від платформи (lock-in).

Однак ця платформа менш поширена, і для її повного використання потрібне вивчення «безкодового» середовища. У таблиці 2.2 представлено порівняння згаданих вище баз даних.

Таблиця 2.2 – Порівняння параметрів, вище зазначених баз даних

Параметр	Firebase (Google)	AWS Amplify (Amazon)	Backendless
Тип бази даних	NoSQL (Realtime DB, Firestore)	NoSQL(DynamoDB, AppSync GraphQL)	NoSQL/SQL (pealtime DB та традиційні БД)
Аутентифікація	Так (Firebase Auth, соцмережі)	Так (Cognito, IAM)	Так (вбудовані user management)
Хостинг / без серверів	Так (Firebase Hosting, Functions)	Так (S3, Lambda, Serverless)	Так (серверлес-логіка, Cloud Code)
Аналітика/пуш	Вбудовані (Analytics, Crashlytics, FCM)	Вбудована (Pinpoint, CloudWatch)	Так (push, email, SMS)
Гнучкість розгортання	Обмежено GCP (хмарний)	Хмарна (AWS)	Хмарна та on-premise (самохост)
Модель оплати	Безкоштовний\pay-as-you-go	pay-as-you-go (через AWS)	Freemium\помісячні плани від $15
Vendor Lock-in	Високий (закритий сервіс)	Високий (AWS-залежність)	Помірний (опції самохостингу)

Visual Studio Code (VS Code) – безкоштовний кросплатформний редактор коду, розроблений Microsoft, що базується на Electron (суміш Chromium та Node.js). За своєю архітектурою VS Code поєднує веб-технології HTML/CSS/JS з функціональністю нативного додатку: він застосовує рушій Monaco, той самий, що використовується у редакторі для браузерів, і поєднує його з сервісною архітектурою, забезпечуючи підтримку різноманітних мов та технологій, таких як Roslyn, TypeScript, Debugging Engine [13]. Редактор доступний для Windows, macOS та Linux, та інтегрує в себе IntelliSense: автоматичне доповнення коду, розуміння структури коду та потужний відладчик. З коробки VS Code пропонує підсвічування синтаксису, навігацію кодом, рефакторинг коду та вбудований термінал. Відкритість та розширюваність – ключові особливості для VS Code, для нього доступний величезний маркетплейс розширень, включно з розширеннями для React Native, Flutter, Node.js, Git та Docker. Розробники можуть застосовувати VS Code як полегшену IDE для мобільних додатків, встановивши необхідні плагіни та налаштувавши конфігурацію.

До сильних сторін VS Code можна віднести його швидку роботу, компактний розмір, зрозумілий інтерфейс і велику спільноту розробників. Він з коробки підтримує величезну кількість мов і технологій, включаючи вбудовані інструменти для пошуку по проєкту, інтеграцію з Git і платформні дебагери. Наприклад, VS Code має надшвидкий редактор коду з функцією IntelliSense для багатьох мов програмування та розумінням семантики коду . Для мобільної розробки доступні офіційні плагіни, як от: React Native Tools для інтеграції з Metro bundler або Expo, Flutter та Dart розширення для налагодження та Hot Reload у Flutter додатках. Водночас, VS Code залишається редактором коду, а не повноцінним IDE: він не має вбудованого емулятора, а налаштування для Android SDK та Gradle потребують встановлення додаткових плагінів. В роботі з дуже великими проєктами можливе збільшення навантаження на пам’ять, але загалом VS Code залишається досить легким. Слід також зазначити, що незважаючи на активний розвиток VS Code від Microsoft, деякі специфічні Android/Java функції можуть бути реалізовані не так глибоко, як у «рідних» IDE.

Серед інших варіантів VS Code для розробки мобільних додатків виокремлюються Android Studio та IntelliJ IDEA, обидва від JetBrains. Android Studio – це офіційне середовище розробки для Android, базується на IntelliJ, яке має все необхідне: систему збирання Gradle, емулятор Android, редактор макетів, шаблони коду, вбудовані засоби тестування та інтеграцію з Google Cloud [14]. Воно пропонує широкий набір інструментів для розробки UI Android, Live Edit Compose та інше. Проте, Android Studio вимагає значних ресурсів, запускається довше, потребує більше оперативної пам’яті та процесорного часу, та менш універсальне для інших технологій, хоча й підтримує Flutter завдяки плагіну. IntelliJ IDEA – це універсальне середовище розробки для JVM (Java/Kotlin), з потужним рефакторингом, аналізом коду, налагодженням та підтримкою Flutter і React Native через відповідні плагіни. IntelliJ забезпечує глибоку інтеграцію з компілятором Kotlin та Android, проте повний функціонал доступний у платній Ultimate-версії. Порівнюючи з цими IDE, VS Code відзначається своєю легкістю,

швидкістю та гнучкістю налаштування. У таблиці 2.3 наведено ключові відмінності між цими середовищами розробки.

Таблиця 2.3 – Порівняння параметрів, вище зазначених середовищ розробки

Параметр	VS Code	Android Studio	IntelliJ IDEA
Тип середовища	Редактор коду (Electron, JS)	Повноцінне IDE (IntelliJ-базове)	Повноцінне IDE (JetBrains)
Мови/технології	Будь-які (через плагіни)	ava/Kotlin, XML, Flutter (плагін)	Java/Kotlin, Flutter (плагін)
Продуктивність	Дуже швидкий запуск, малий ресурс	Тяжкий (велике навантаження на пам’ять)	Тяжкий (вирішення великих проєктів)
UI-інструменти	Плагіни для UI (Flutter, React UI)	Visual Layout Editor, UI Builder	Плагіни, але менше інтеграції для Android UI
Налагодження	Вбудовано (JS/TS, Flutter, React)	Глибока підтримка Android (ADB, Logcat)	Потужний Java/Kotlin дебаг, Android плагін
Розширення/плагіни	Мільйони розширень у Marketplace	Плагіни з акцентом на Android	Багато плагінів для JVM/Flutter
Інтеграція з платформою	Git, Docker, Azure, Firebase плагіни	Глибока з Android SDK та GCP	Інтеграція з Kotlin/Gradle
Вартість	Безкоштовний (OSS)	Безкоштовний (із обмеженнями)	Безкоштовний та Підписка(Community, Ultimate)

Expo – це відкрита платформа-інструментарій, створені поверх React Native, що полегшує розробку, тестування та публікацію мобільних додатків. З архітектурної точки зору, Expo є обгорткою навколо React Native з вже інтегрованими модулями та бібліотеками, включаючи доступ до камери, сповіщень, контактів, файлової системи та Bluetooth. Замість кодування нативним кодом для кожної платформи, розробники використовують JavaScript API, які Expo перетворює у відповідні нативні виклики через вбудовані модулі у своєму «runtime». Expo пропонує два режими розробки: Managed workflow, найбільш зручний варіант, де вся розробка відбувається виключно на JavaScript/TypeScript, а Expo відповідає за нативну частину. Це звільняє від використання Android Studio/Xcode на початкових етапах та Bare workflow, надає більше контролю: дозволяє змінювати нативний код, зберігаючи при цьому

інтеграцію з Expo SDK. Важливою особливістю є Expo Go – додаток, що дозволяє запускати проєкт на реальному пристрої без необхідності збірки APK/IPA, просто скануючи QR-код. Для фінального розгортання використовується EAS (Expo Application Services) – сервіс Expo для збирання, підписання та публікації додатків у Google Play та App Store. Expo особливо приваблює новачків та невеликі команди завдяки: швидкому старту розробки, відсутності потреби встановлювати додаткове ПЗ, широкому вибору готових API, простоті тестування на реальних пристроях за допомогою Expo Go, автоматичному збиранню додатків у хмарі. Завдяки автоматизації та спрощенню процесів [15]. Expo дозволяє сконцентруватися на логіці та інтерфейсі користувача, замість витрачати час на налаштування оточення розробки. Крім того, платформа добре інтегрована з Firebase та популярними бібліотеками екосистеми React Native. Expo – це унікальний за своєю суттю інструмент, який працює на основі React Native.

Серед альтернатив Expo для мобільної розробки варто виділити Capacitor від Ionic і класичний React Native без Expo. Capacitor – сучасний інструмент від Ionic, який дає змогу веброзробникам створювати нативні мобільні додатки на основі HTML, CSS та JavaScript, інтегруючи їх із нативними функціями платформи за допомогою плагінної системи. На відміну від Expo, що тісно інтегрований із React Native, Capacitor дозволяє використовувати будь-який сучасний фронтенд-фреймворк, наприклад, React, Angular чи Vue, та забезпечує прямий доступ до Android та iOS проектів, що відкриває більше простору для нативних налаштувань. Однак, це також вимагає глибших знань структури нативних проєктів, системи збірки та інших аспектів мобільної розробки, які Expo значною мірою приховує. Крім того, екосистема Ionic більше спрямована на гібридні додатки, де інтерфейс рендериться у WebView, що потенційно може впливати на продуктивність у складних UI-сценаріях [16]. React Native без використання Expo – це класичний підхід, коли розробник самостійно ініціалізує проєкт за допомогою CLI «react-native init» та має повний контроль над структурою та залежностями нативного коду. Такий підхід відкриває всю

функціональність React Native, включаючи можливість підключати будь-які нативні модулі або бібліотеки без обмежень, як це є в межах Expo SDK. Але цей шлях вимагає ручного налаштування середовища, як-от Android SDK, CocoaPods, тісної взаємодії з Gradle або Xcode, а також більшої відповідальності за збірку та деплой додатку [17]. У порівнянні з Expo, де можна швидко створити MVP без потреби глибокого знання специфіки Android/iOS, «чистий» React Native орієнтований на розробників з більшим технічним досвідом і потребою у гнучкості. У таблиці 2.4 зображені ключові характеристики даних інструментаріїв.

Таблиця 2.4 – Порівняння параметрів, вище зазначених інструментаріїв

Параметр	Expo	React Native (без Expo)	Capacitor (Ionic)
Мова розробки	JavaScript / TypeScript	JavaScript / TypeScript	JavaScript + HTML/CSS
Наявність нативного коду	Немає (або приховано)	Так, доступний для редагування	Є, але взаємодія через WebView
Підтримка публікації	Через EAS Build або Expo CLI	Ручна або CI/CD	Через CLI, часто потребує Xcode/Gradle
Швидкість старту	Дуже швидкий (1-2 хв)	Середній (встановлення Android SDK)	Швидкий, але з WebView обмеженнями
Потреба в Xcode / Android Studio	Ні (на ранніх етапах)	Так	Так
Гнучкість конфігурації	Середня	Висока	Середня

Отже, після проведеного аналізу, стає очевидно, що зв’язка React Native + Firebase + VS Code є надзвичайно привабливою для створення мобільного застосунку-планувальника. React Native надає перевагу швидкої кросплатформної розробки з єдиною кодовою базою та великою кількістю підтримуваних бібліотек, що зменшує витрати часу та ресурсів на візуальний інтерфейс. Firebase слугує повноцінним бекендом: база даних, аутентифікація в реальному часі, хостинг та аналітика, майже не потребуючи окремого серверного забезпечення. VS Code, зі свого боку, представляє собою легкий, але функціональний редактор: він швидко адаптується до розробки з React Native, пропонує зручне налагодження та інтеграцію з Git, не споживає багато ресурсів,

зосереджуючи увагу на коді, а не на інструментах. Окремої уваги заслуговує Expo – інструментарій, який доповнює React Native та значно спрощує весь процес розробки. Завдяки вбудованим модулям, таким як доступ до готових API, їх можна застосовувати без необхідності налаштування нативного середовища. Expo дозволяє швидко запустити проект за допомогою додатку Expo Go, навіть без компіляції, що робить тестування майже миттєвим. Expo усуває технічні перешкоди початкового налаштування, делегує збирання застосунків у хмару (EAS Build) і дає змогу повністю сконцентруватися на логіці та інтерфейсі продукту.

Висновок до розділу 2

У цьому розділі проведено дослідження, в результаті якого були виявлені функціональні здібності мобільного застосунку. Створено моделі для проєкту, визначені ключові складові системи, розроблені UML-діаграми варіантів застосування та класів, котрі наочно показують структуру даних, головні об’єкти та сценарії взаємодії користувача з системою.

Після моделювання було аргументовано вибір технологічного стеку. Головним фреймворком обрано React Native, завдяки повторному використанню коду, потужній спільноті та значній кількості готових бібліотек. Для втілення хмарної логіки та збереження даних застосовано Firebase, котрий надає сервіси для аутентифікації, баз даних реального часу та хмарного сховища, що зводить до мінімуму необхідність у власному сервері.

Додатково застосовано платформу Expo, що полегшує запуск проєкту, роботу з нативними модулями та автоматизує збирання застосунку. Середовищем розробки обрано Visual Studio Code, яке має підтримку контролю версій, інтегрований термінал та автодоповнення, сприяючи ефективній розробці.

Таким чином, вибраний технологічний стек гарантує оптимальний баланс між легкістю розробки, функціональністю, продуктивністю та можливістю масштабування.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Практична реалізація об’єкта проектування

Початком розробки мобільного додатку, стало розгортання інструментарію Expo в середовищі Visual Code, за для цього потрібно встановити файлову частину з офіційного сайту [15] у папку з майбутнім проєктом, щоб отримати відповідну структру (рис. 3.1).

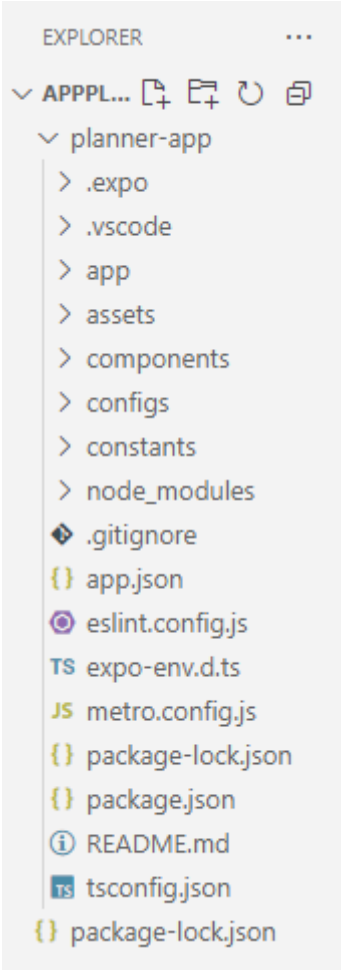

Рисунок 3.1 – Структура проєкту з інструментарієм Expo

Проєкт реалізовано за стандартною структурою, властивою React Native-застосункам, створеним з використанням Expo. Головна логіка програми міститься у папці app, де є окремі підпапки для маршрутизації: екранів, форм автентифікації, створення нотаток та вкладок нижнього меню. У папці tabs

зберігаються файли основних екранів: домашня сторінка, календар, а також додаткові вкладки. Підпапка entr містить компоненти для входу та реєстрації користувача. Для створення нових нотаток використовується директорія create-notes, яка містить відповідний редактор. Компоненти, які використовуються повторно, розміщені в окремій папці components, а медіафайли та шрифти – в assets. Конфігураційні файли, зокрема налаштування Firebase (FirebaseConfig.js), перенесено до папки configs. У каталозі constants зберігаються сталі значення, такі як палітра кольорів (Colors.ts). Основні конфігураційні файли Expo (app.json, expo-env.d.ts), TypeScript (tsconfig.json), ESLint (eslint.config.js) та інші, розташовані в кореневій директорії. Такий підхід до структуризації полегшує навігацію в проєкті, його масштабування та підтримання чистоти кодової бази.

Навігацію між екранами реалізовано через expo-router, що автоматично керує маршрутизацією, орієнтуючись на структуру файлів у проєкті. Для роботи з вкладками та перемикання між екранами застосовано компоненти Tabs, useNavigation та useRoute. Це дозволяє просто створити стек або табуляторну навігацію, уникаючи потреби вручну конфігурувати навігаційні стеки. Функціонал авторизації розроблено з використанням Firebase Authentication. Користувачі можуть входити, використовуючи електронну пошту та пароль, або через Google-акаунт. При запуску програми відбувається перевірка стану аутентифікації, і користувач автоматично переадресовується на головний екран чи на форму входу, враховуючи їх поточний статус. На рисунку 3.2, зображена домашня сторінка мобільного додатку.

Для зберігання подій, записів, розділів та сповіщень використовується Firebase Realtime Database. Структура сховища організована за ключами, де кожен запис асоціюється з унікальним ідентифікатором користувача. Це дає змогу реалізувати персональні дані користувача, доступні лише після автентифікації. Додаток дозволяє створювати нові записи з вказівкою дати, часу та категорії, редагувати наявні, а також видаляти їх.

Рисунок 3.2 – Головна сторінка мобільного застосунку

Для покращення взаємодії з користувачами впроваджено систему push-сповіщень, реалізовану з використанням Expo Notifications API. Користувачі тепер мають можливість налаштовувати нагадування для окремих записів, які активуються у визначений час, навіть якщо застосунок не запущений.

Також було реалізовано функцію перегляду записів у форматі календаря та щоденного переліку справ.

Розглянуто можливість створення віджета для головного екрана мобільного телефону, проте на час написання роботи розроблено лише базову структуру та дизайн інтерфейсу цього компонента (рис. 3.3). Втілення такого функціоналу вимагає додаткових зусиль з розробки, враховуючи специфіку операційних систем Android та iOS.

Додаток було випробувано на справжньому пристрої з операційною системою Android, використовуючи інструмент Expo Go.

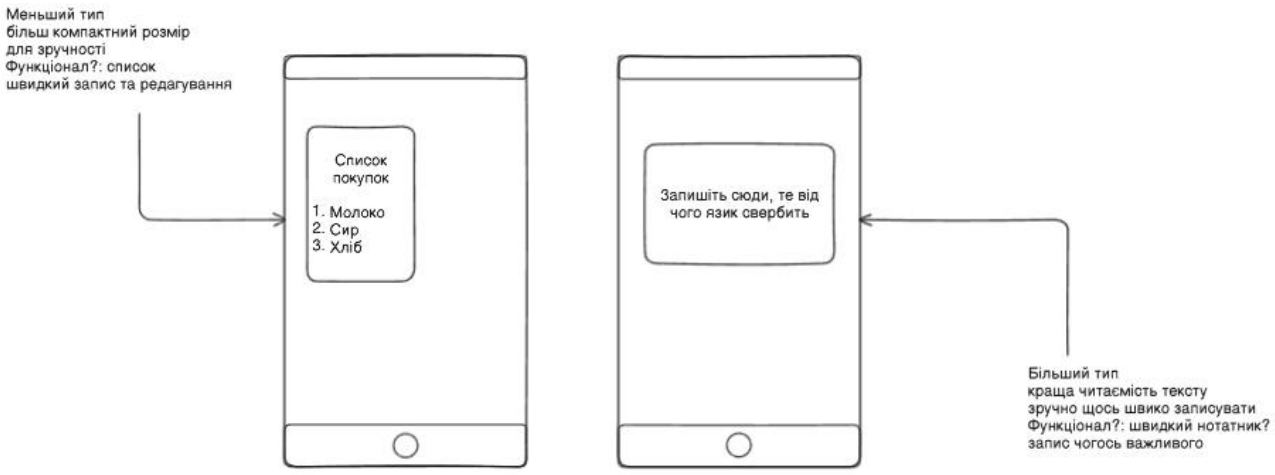

Рисунок 3.3 – Макет майбутнього віджету мобільного застосунку

Під час тестування було перевірено функціонування аутентифікації, можливості створення та редагування записів, налаштування нагадувань та їх активацію. Виявлено декілька невеликих помилок, які було швидко виправлено.

Для розгортання проекту використовувався GitHub, як система керування версіями коду. Якщо виникає необхідність, робочу версію додатку можливо розгорнути на Expo Hosting, або ж опублікувати у Google Play Store та App Store після збирання та цифрового підпису застосунку.

Отже, під часвиконання кваліфікаційної роботи було створено мобільний додаток, що слугує веденням особистих заміток та організації щоденного розкладу. Цей додаток поєднує в собі функції цифрового щоденника з нагадуваннями про важливі події та можливістю сортування справ за категоріями. Завдяки вдало підібраним технологіям, зокрема фреймворку React Native, інструменту Expo для оптимізації процесу розробки, та хмарному сервісу Firebase, що забезпечує зберігання даних і авторизацію користувачів, додаток демонструє гнучкість, адаптивність та перспективу для подальшого розширення функціональності.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Під час створення мобільного додатку, тестування та налаштування відбувалося здебільшого за допомогою технології Expo Go – офіційного додатку від Expo, який дає змогу швидко оцінювати результати розробки безпосередньо на фізичних пристроях, уникаючи створення окремих білдів. Після активації середовища розробки з Metro Bundler, розробник отримує QR-код, його можна відсканувати за допомогою Expo Go (рис. 3.4) для миттєвого запуску додатку. Такий метод суттєво зменшує час між внесенням правок до коду та спогляданням результату.

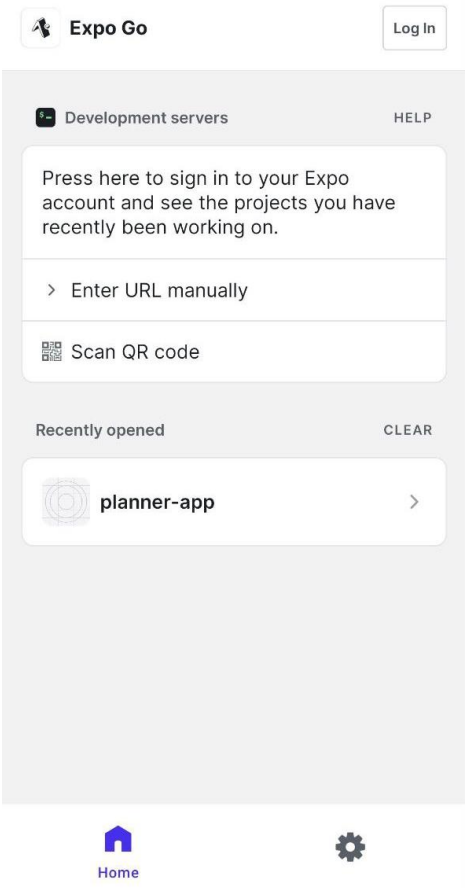

Рисунок 3.4 – Вигляд інтерфейсу Expo Go [17]

Однією з ключових можливостей Expo Go є Fast Refresh – функція швидкого оновлення інтерфейсу, яка спрацьовує щойно ви вносите зміни до коду, позбавляючи необхідності перезавантажувати всю програму. До того ж,

розробники можуть увімкнути режим налагодження, що відкриває доступ до DevTools – інструментів, які дозволяють переглядати журнали, інспектувати елементи інтерфейсу, аналізувати мережеві запити, а також контролювати продуктивність [18]. На рисунку 3.5 зображений вигляд компіляції коду Expo Go у терміналі середовища Visual Code.

Рисунок 3.5 – Вигляд запуску Expo Go у Visual Code

Також доступна віддалена відладка JavaScript-коду, яка дозволяє встановлювати точки зупину, аналізувати помилки, переглядати змінні та відстежувати логіку виконання функцій прямо з інтерфейсу додатку. Для візуального аналізу продуктивності можна використовувати вбудовані індикатори FPS, а також розвантаження JavaScript та UI-потоків. Це дозволяє оптимізувати застосунок у реальному часі, не переходячи до нативної збірки. На рисунку 3.6 зображений інтерфейс мобільного додатку Expo Go з наявними інструментами відладки.

Коли виникає потреба у доступі до нативного коду (скажімо, для тонкої інтеграції з системними API), передбачене наступне рішення: потрібно перейти

на EAS Dev Client, або ж повністю вилучити з Expo. У такому разі доведеться залучити Android Studio чи Xcode [19], але на етапі кваліфікаційної розробки потенціалу Expo Go було більш ніж достатньо.

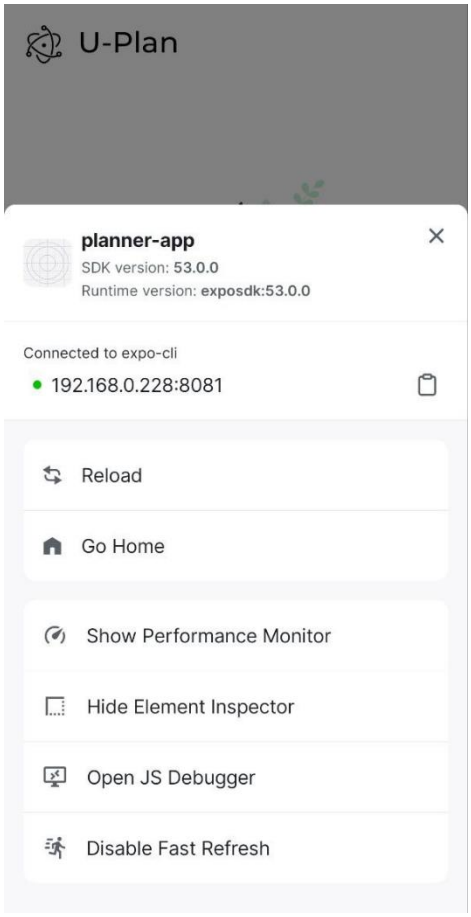

Рисунок 3.6 – Вигляд інтерфейсу Expo Go та панелі розробника DevTools [17]

Завдяки застосуванню Expo Go вдалося суттєво покращити ефективність налагодження та пришвидшити процес тестування програми, не втрачаючи якості кінцевого продукту. Це дало можливість виявити критичні помилки ще на початкових стадіях розробки та негайно вносити необхідні правки.

3.3 Розробка бази даних

Основою для зберігання інформації у створеному мобільному застосунку виступає Firebase Realtime Database. На відміну від традиційних реляційних баз даних, ця система має структуру у вигляді дерева у форматі JSON, де всі дані

представлені як вкладені об’єкти (рис. 3.7). Такий підхід дає змогу швидко реагувати на зміни – будь-яке оновлення даних миттєво поширюється на всі підключені пристрої, що особливо важливо для мобільних застосунків, де синхронізація й оперативність – критичні чинники.

Рисунок 3.7 – Структура Firebase Realtime Database [20]

У процесі реалізації проєкту для зберігання нотаток та інформації про користувача було спроєктовано структуру, в якій кожен запис містить поля: назва нотатки (name), основний текст (text), а також службові параметри, що допомагають відстежувати історію редагування. Структура бази є достатньо гнучкою, що дозволяє в майбутньому розширити її додатковими категоріями, тегами або датами нагадувань. Firebase також забезпечує автоматичне керування авторизованим доступом: кожен користувач має свій унікальний простір для даних, ізольований від інших. На рисунку 3.8 зображений вигляд авторизаційної бази даних.

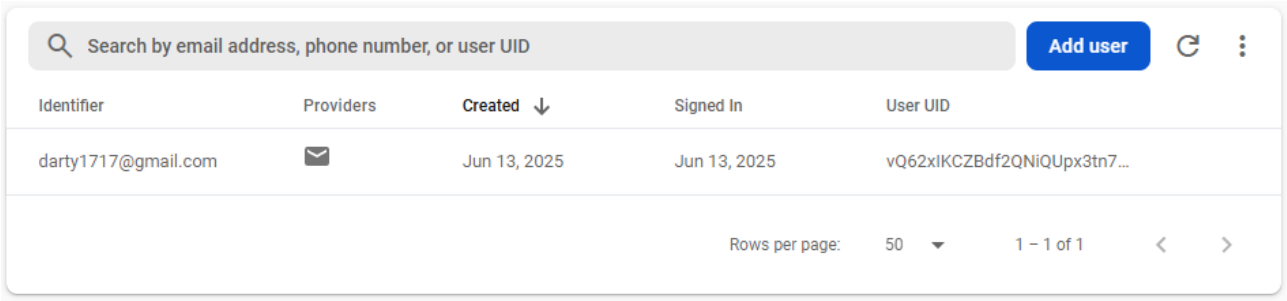

Рисунок 3.8 – Вигляд бази даних з авторизаційними даними користувачів

Функціонал запису та зчитування реалізовано через API бібліотеки firebase/database. Запис даних у базу відбувається після обробки змін, внесених користувачем у поля вводу. Для цього використовується станова логіка React – змінні name і text зберігаються за допомогою хука useState (ліст. 3.1).

Лістинг 3.1 – Хук useState

```
const nameInputRef = useRef(null);
const textInputRef = useRef(null);
```

кінець лістингу 3.1

А самі зміни, обробляються подіями onChangeText (ліст. 3.2). У момент оновлення одного з полів застосунок формує новий об’єкт стану, вносить зміни локально та одночасно передає ці зміни до бази. Наприклад, введення назви нотатки автоматично оновлює поле name та створює відповідний запис у Firebase. У результаті кожна дія користувача викликає синхронізацію, забезпечуючи збереження даних навіть при зміні мережевого стану.

Лістинг 3.2 – Елемент TextInput з подією onChangeText

```
<TextInput
    ref={textInputRef}
    style={styles.input}
    onChangeText={(value) => {
        const newState = {
            ...currentState,
            text: value
        };
        setText(value);
```

```
        updateHistory(newState);
        setLastEditedField('text');
    }}
    value={text}
    placeholder="Текст нотатки"
    placeholderTextColor={Colors.Gray}
    multiline
    textAlignVertical="top"
    onFocus={() => {
        setIsFocused(true);
        setLastEditedField('text');
    }}
    onBlur={() => setIsFocused(false)}
/>
```

кінець лістингу 3.2

Для виведення даних із бази використовується механізм спостерігачів, який дозволяє підписатися на зміни конкретної гілки бази. Якщо користувач додає або редагує запис на іншому пристрої, ці зміни миттєво оновлюються на поточному інтерфейсі без необхідності ручного оновлення сторінки. У випадку втрати підключення Firebase автоматично зберігає дані в кеші пристрою, а після відновлення з'єднання синхронізує їх із сервером.

У лістингу 3.3 зображений елемент виведення інформації з бази даних.

Лістинг 3.3 – Елемент виведення інформації

```
useEffect(() => {
  const notesRef = ref(db, 'notes');
  const unsubscribe = onValue(notesRef, (snapshot) => {
    const data = snapshot.val();
    if (data) {
      // Конвертуємо об'єкт в масив
      const notesArray = Object.keys(data).map(key => ({
        id: key,
        ...data[key]
      }));
      setUserNotes(notesArray);
    } else {
      setUserNotes([]);
    }
  });
  return () => unsubscribe();
}, []);
```

кінець лістингу 3.3

Firebase Realtime Database також підтримує базові правила доступу, які налаштовуються у спеціальному редакторі на консолі. Це дозволяє встановити обмеження, наприклад: дозволити запис лише для авторизованих користувачів або доступ лише до своїх даних.

Таким чином, дані кожного користувача ізольовані, що забезпечує конфіденційність та безпечне зберігання інформації.

Завдяки використанню Firebase вдалося уникнути складного налаштування серверної частини та сфокусуватися на функціональності самого застосунку. Це хмарне рішення дозволяє масштабувати базу без значних змін у логіці застосунку, що є суттєвою перевагою для MVP-проєктів або продуктів, які планується поступово розвивати.

3.4 Синхронізація даних між пристроями у хмарному середовищі Firebase

Одним з найважливіших елементів сучасних мобільних додатків, включаючи особисті щоденники, є необхідність у збереженні даних не тільки на пристрої, але й у хмарі, з подальшою синхронізацією між різними приладами. Це дозволяє зберегти ваші записи навіть у разі втрати пристрою, надає змогу працювати з додатком на кількох пристроях і гарантує безперервний доступ до особистої інформації. Такий підхід покращує зручність використання, забезпечує безперервність роботи та дозволяє уникнути втрати важливих даних. Окрім того, це відкриває можливості для розширення функціоналу – створення резервних копій, спільного доступу до записів або інтеграції з іншими хмарними сервісами. У розробленому мобільному щоденнику, синхронізація реалізована за допомогою Firebase Realtime Database у поєднанні з Firebase Authentication, що забезпечує захищену авторизацію користувачів.

Архітектура Firebase, як така, побудована на принципі роботи в реальному часі, забезпечуючи негайне оновлення вмісту вашої програми щоразу, коли відбуваються зміни. Щойно користувач створює або коригує запис у щоденнику,

ці зміни негайно фіксуються в базі даних Firebase Realtime Database. Заздалегідь налаштовані onValue або onSnapshot реагують на ці оновлення практично миттєво, відтворюючи актуальну інформацію на всіх пристроях користувача, де встановлено активний сеанс. У такий спосіб досягається значний рівень інтерактивності та гарантується безперервність взаємодії з додатком.

Характерною рисою імплементації у рамках цього проекту є те, що синхронізація даних прив'язана до унікального ідентифікатора користувача, який згенеровано Firebase після аутентифікації. Отже, кожен запис у базі даних пов'язаний з UID користувача, що унеможливлює доступ до чужих даних без проходження авторизації. Збереження даних у формі вкладеної JSON-структури дає змогу ефективно організовувати колекції щоденникових записів за датами, тематиками чи категоріями. Такий спосіб також дозволяє уникнути дублювання записів під час одночасного використання застосунку на кількох пристроях [21].

У процесі створення мобільного додатку на React Native, першочерговим завданням постала реалізація взаємодії з даними як на пристрої, так і у хмарному сховищі. Для вирішення цієї задачі було обрано бібліотеку «react-native-firebase/database», що пропонує готовий інтерфейс для роботи з Realtime Database. Код для запису інформації у базу даних розміщено у функції, яка викликається при кліку на кнопку «Зберегти». Внесені зміни моментально надсилаються до Firebase, і у випадку стабільного інтернет-підключення, синхронізація даних відбувається майже миттєво. Крім того, кешування, реалізоване Firebase, дозволяє користувачам тимчасово працювати з даними в офлайн режимі, з подальшим їхнім оновленням при відновленні з'єднання. Це є критично важливим у випадках нестійкого інтернет-зв'язку.

Серед ключових рис реалізації у моєму проєкті – оптимізація трафіку під час синхронізації. Щоб не перевантажувати мережу, синхронізація запускається лише тоді, коли виявлено зміни у структурі даних. До того ж, використання child_added, child_changed та child_removed дає змогу реагувати тільки на конкретні дії користувачів, замість того, щоб перебирати всю базу даних при

кожному оновленні. Цей підхід критичний для продуктивності, особливо з ростом кількості записів у щоденнику.

Слід зауважити, що користувачу під час входу надається можливість вибрати будь-який з доступних методів авторизації, наприклад, електронна пошта з паролем чи Google Sign-In. Це збільшує комфорт при доступі до даних у хмарі та забезпечує плавну зміну пристроїв. Функція автентифікації зберігає статус сесії, що дає змогу автоматично підключатися до бази даних при кожному відкритті застосунку, уникнувши потреби повторно вводити дані. Усі ці нововведення, здійснені з використанням Firebase, помітно покращують зручність використання, безпеку та можливості щоденника.

Отже, функція синхронізації даних у мобільному додатку-щоденнику втілена повністю та продуктивно завдяки гнучкості й масштабованості платформи Firebase. Це дає змогу користувачеві мати безперервний доступ до своїх записів, незалежно від пристрою та місця перебування, що є критичною вимогою сучасних мобільних додатків у сфері особистої продуктивності.

3.5 Захист інформаційно-комп’ютерної системи

В межах мобільного додатку «Особистий щоденник» втілено комплекс захисних механізмів, спрямованих на забезпечення безпечного функціонування з даними користувачів. Автентифікація відбувається за допомогою Firebase Authentication, де паролі зберігаються у вигляді хешу за допомогою алгоритму «scrypt». На відміну від простих хеш-функцій, таких як MD5 чи SHA-1, у яких без проблем можна обчислити мільйони разів за секунду, scrypt розроблено спеціально, щоб бути надзвичайно вимогливим як до обчислювальних ресурсів, так і до пам’яті. Саме це робить його захищеним від атак методом перебору brute-force та «райдужних таблиць», особливо якщо їх проводять за допомогою сучасних графічних процесорів або ж спеціалізованого устаткування ASIC. Алгоритм scrypt потребує значних обчислювальних потужностей та достатньо оперативної пам’яті, що суттєво ускладнює масове хешування паролів при

спробах зламу баз даних [22]. Кожен автентифікаційний токен підписується відповідно до стандарту JSON Web Token (JWT) із шифром SHA-256 HMAC. Відтак, «сирі» паролі не зберігаються, а передаються у захищеному вигляді (хеші).

Передача інформації між клієнтом та сервером реалізується за допомогою TLS/HTTPS (SSL 2048-біт), що гарантує їхнє шифрування в мережі та зменшує ризик перехоплення [23]. Водночас, Firebase також шифрує дані у стані зберігання, що забезпечує захист від несанкціонованого доступу – як активного, так і пасивного. Впровадження Firebase Security Rules надає змогу гнучко керувати доступом користувачів до даних, зокрема, дозволяючи запис до бази лише аутентифікованим користувачам та виключно у їхній особистий простір даних. Це усуває можливість читання або запису чужих заміток, навіть у разі компрометації авторизації.

Для боротьби з DDoS-атаками та зловмисним трафіком застосовуються інтегровані механізми Firebase App Check, обмеження швидкості запитів та моніторинг трафіку, що дозволяє виявляти підозрілу активність і блокувати доступ у разі перевищення встановлених лімітів [24].

У майбутньому передбачено інтеграцію з Google Cloud Functions та Cloud Monitoring, що сприятиме ще більшій стабільності програми. Для ускладнення зворотного інжинірингу клієнтської частини до процесу збирання програми додано мініфікацію та обфускацію коду за допомогою Hermes, а також, за потреби, expo-build-properties з ProGuard/R8 для Android [25]. Це не гарантує абсолютного захисту, але суттєво збільшує час та ресурси, потрібні потенційному зловмиснику.

Було також впроваджено рекомендації, щоб уникнути збереження секретної інформації прямо в коді: наприклад, API-ключів та токенів. Замість цього використовуємо конфігураційні файли app.config.js, які не публікуються у репозиторах.

Отже, ця система включає в себе декілька шарів безпеки. Від аутентифікації та захищеного каналу для передачі даних, через контроль доступу

й захист від DDoS-атак, аж до шифрування, обфускації коду та дотримання правил безпеки. Всі ці заходи у сукупності формують цілісний підхід до захисту персональних даних користувача.

Висновки до розділу 3

У цьому розділі детально розглянуто процес створення мобільної програми «Особистий менеджер завдань», від ідеї до забезпечення безпеки інформаційної системи. Сучасні інструменти, зокрема React Native, Expo та Firebase, дозволили розробити адаптивний, масштабований та безпечний продукт.

Реалізація проєкту виконана за структурою, що полегшує навігацію, розширення функціоналу та підтримку коду. Навігація між екранами здійснена за допомогою expo-router, а авторизація та збереження даних реалізовані через Firebase Authentication та Firebase Realtime Database, що гарантує миттєву синхронізацію та цілісність даних, навіть у режимі офлайн.

Тестування проводилось за допомогою Expo Go, що надало змогу оперативно виявляти та виправляти помилки. Система зберігання даних забезпечила ефективне управління та безпеку завдяки шифруванню, контролю доступу та захисту від DDoS-атак.

Функція перегляду завдань у форматі календаря робить навігацію зручною, а серед перспективних напрямів розвитку – впровадження віджетів для головного екрана. Усі етапи розробки та тестування підтверджують готовність додатку до використання, відповідність актуальним вимогам та наявність потенціалу для подальшого вдосконалення.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра, було створено мобільний застосунок під назвою «Персональний менеджер завдань», що поєднує в собі функціонал планувальника, щоденника та системи сповіщень. Розробка була здійснена з використанням передових технологій, зокрема React Native, Expo та Firebase, що забезпечило створення ефективної, масштабованої та захищеної програми.

Проведено аналіз теперішнього стану в сфері управління особистими справами, базуючись на вивченні відомих мобільних програм, а саме: Notion, TickTick, Trello та Todoist. Кожна з них володіє власними перевагами: Notion славиться гнучкістю, проте важкий для новачків; TickTick є комфортним для щоденного планування; Trello пропонує дієві візуальні інструменти для спільної роботи; Todoist – простий та структурований для особистих завдань. Крім того, було розглянуто доповіді Verified Market Research та Market Research Intellect, які вказують на постійне збільшення зацікавленості у цифрових інструментах продуктивності, з акцентом на хмарну синхронізацію, мультиплатформеність та функції для спільної роботи, що дозволило виділити основні тенденції, сильні та слабкі сторони сучасних рішень.

Сформовано список вимог та критеріїв до програмного забезпечення, яке розроблялось. Зокрема, пріоритет було віддано зручному інтерфейсу, кросплатформності, продуктивності, інтеграції з хмарними сервісами та захисту даних.

Відповідно до поставлених цілей, було обрано оптимальні інструменти, підходи та технології, такі як: React Native для кросплатформенної розробки, Firebase для бекенду, Expo для полегшення тестування та збирання, а також Visual Studio Code як основне середовище розробки.

Виконано практичну частину проекту, створено мобільний застосунок «Персональний менеджер завдань» з функціоналом авторизації, створення,

перегляду та редагування нотаток, перегляду у вигляді календаря, а також керування профілем користувача.

Здійснено тестування та налагодження системи з використанням інструменту Expo Go. Це дало можливість швидко виявляти помилки, перевірити стабільність роботи застосунку та його функціонування в різних умовах, зокрема, при відсутності інтернет-з’єднання.

Створено структуру бази даних на базі Firebase Realtime Database та інтегровано її з застосунком. Організація зберігання включає дані користувачів, нотатки, профілі та інформацію для аутентифікації.

Забезпечено синхронізацію даних між пристроями в реальному часі. Firebase гарантує, що зміни, зроблені на одному пристрої, негайно відображаються на інших, що критично важливо для сучасних кросплатформних додатків.

Особливу увагу приділено забезпеченню інформаційної безпеки: реалізовано аутентифікацію з контролем доступу, використано механізми шифрування Firebase, що дає змогу захистити персональні дані від несанкціонованого доступу чи зовнішніх загроз.

Отже, кінцевим продуктом роботи став, повністю функціональний мобільний додаток, що відповідає поточним стандартам функціональності, швидкодії та захисту. Застосовані технології сприяли прискоренню процесу розробки, а також створили фундамент для майбутнього додавання нових можливостей. Цей додаток може виявитися цінним помічником для планування особистого часу та підвищення продуктивності, особливо в умовах сьогодення з його шаленим темпом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The AI workspace that works for you. Notion. URL: https://www.notion.com/ (date of access: 10.02.2025).

2. TickTick. URL: https://ticktick.com/ (date of access: 11.02.2025).

3. Trello. Capture, organize, and tackle your to-dos from anywhere. URL: https://trello.com/ (date of access: 11.02.2025).

4. Todoist. A to-do list to organize your work & life. URL: https://www.todoist.com/ (date of access: 15.02.2025).

5. Custom research – verified market reports. URL: https://www.verifiedmarketreports.com/product/digital-planner-app-market/ (date of access: 16.02.2025).

6. Daily planner app market size, share & industry trends analysis 2033. Market Research Intellect. URL: https://www.marketresearchintellect.com/product/daily-planner-app-market/ (date of access: 20.02.2025).

7. Power of Modelling in today's complex world. Systems Thinking Alliance. URL: https://systemsthinkingalliance.org/decoding-the-complexity-unveiling-the-power-of-systems-modelling-in-todays-world (date of access: 07.03.2025).

8. React Native Learn once, write anywhere. URL: https://reactnative.dev/ (date of access: 08.03.2025).

9. Flutter documentation. Docs – Flutter. URL: https://docs.flutter.dev/ (date of access: 09.03.2025).

10. Full-Cycle software development company – softjourn. Softjourn Inc. URL: https://softjourn.com/ (date of access: 10.03.2025).

11. Your application's backend, simplified. Back4App Blog. URL: https://blog.back4app.com/ (date of access: 10.03.2025).

12. Zeroqode – building apps & business automations with no-code+ai. URL: https://zeroqode.com/ (date of access: 12.03.2025).

13. Microsoft. Visual studio code – code editing. Redefined. URL: https://code.visualstudio.com/ (date of access: 12.03.2025).

14. Android mobile app developer tools – android developers. URL: https://developer.android.com/ (date of access: 12.03.2025).

15. Expo documentation. URL: https://docs.expo.dev/ (date of access: 13.03.2025).

16. Capacitor – cross-platform native runtime for web apps. Capacitor documentation. Capacitor by Ionic – Cross-platform apps with web technology. URL: https://capacitorjs.com/docs (date of access: 13.03.2025).

17. Expo Go. URL: https://expo.dev/go (date of access: 14.03.2025).

18. Expo CLI. Expo Documentation. URL: https://docs.expo.dev/more/expo-cli (date of access: 18.03.2025).

19. Debugging basics react native. React Native Learn once, write anywhere. URL: https://reactnative.dev/docs/debugging (date of access: 18.03.2025).

20. Correct Firebase Database Structure. URL: https://stackoverflow.com/questions/45116390/correct-firebase-database-structure (date of access: 19.03.2025).

21. Realtime database, react native firebase. URL: https://rnfirebase.io/database/usage (date of access: 19.03.2025).

22. RFC 7914: the scrypt password-based key derivation function. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7914 (date of access: 20.03.2025).

23. Firebase Authentication, register/login, are the passwords already encrypted. URL: https://stackoverflow.com/questions/67231302/in-using-firebase-authentication-are-the-passwords-already-encrypted (date of access: 15.04.2025).

24. Firebase – google's mobile and web app development platform. URL: https://firebase.google.com/ (date of access: 18.04.2025).

25. Google cloud community. Sign in – Google Accounts. URL: https://www.googlecloudcommunity.com/ (date of access: 21.04.2025).

ДОДАТКИ

Додаток А – Підтвердження апробації результатів кваліфікаційної роботи

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ІТОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

сайт здатний перетворитися на ефективний інструмент для бізнесу чи особистих проектів.

Список використаних джерел

1. Inertia. *Inertia.js*. URL: https://inertiajs.com/ (дата звернення: 10.05.2025).

2. React. *React*. URL: https://react.dev/ (дата звернення: 10.05.2025).

3. Laravel. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/ (дата звернення: 10.05.2025).

УДК 004.05

ІНСТРУМЕНТ EXPO – ОПТИМІЗАЦІЯ ПРОЦЕСУ СТВОРЕННЯ МОБІЛЬНИХ ДОДАТКІВ ЗАСОБАМИ REACT NATIVE

Виримчук Богдан Ігорович

Луцький національний технічний університет, студент, бакалавр інженерії програмного забезпечення, bodyavyrymchuk@gmail.com

Суринович Олена Миколаївна

Луцький національний технічний університет, к.т.н., доцент кафедри інженерії програмного забезпечення, sivom@ukr.net

EXPO TOOL – STREAMLINING MOBILE APP DEVELOPMENT WITH REACT NATIVE

Vyrymchuk Bohdan Ihorovych

Lutsk National Technical University, Student, Bachelor of Software Engineering, bodyavyrymchuk@gmail.com

Surynovych Olena Mykolaivna

Lutsk National Technical University, PhD in Technical Sciences, Associate Professor of Software Engineering, sivom@ukr.net

Expo is an ecosystem of tools for building mobile apps based on React Native. It provides a wide range of tools to speed up, standardize, and simplify cross-platform app development. With Expo, developers can avoid the complexity of configuring native environments and focus on the functional part of the project.

Keywords: *React Native, Expo, mobile development, developer tools, multiplatform applications, integrated environments, process improvement, and simplification.*

У теперішніх реаліях мобільна розробка стає дедалі важливішою складовою IT-індустрії, що обумовлює великий попит на швидкі, гнучкі та масштабовані засоби для створення та підтримки мобільних застосунків. Це особливо актуально для стартапів, малого бізнесу та незалежних розробників, які прагнуть зменшити витрати часу та ресурсів. З-поміж інструментів, які найбільше виділяються на масовому ринку та безпосередньо впливають на ефективність процесу розробки, виокремлюється Expo – фреймворк з відкритим кодом, що є доповненням React Native.

Expo надає готове середовище розробки, яке дозволяє розробникам зосередитись на функціональності застосунку, уникаючи складної конфігурації нативних платформ. До його складу входять інструменти для автоматизованого тестування, попереднього перегляду застосунків на реальних пристроях, швидкого деплою, а також доступу до широкого спектру API, таких як push-нотифікації, камера, геолокація, медіа та інші. Крім того, інтеграція з такими сервісами як Firebase або Sentry є майже безшовною, що ще більше прискорює розробку проєктів.

Аналіз процесу розробки мобільного додатку розпочався з дослідження етапів життєвого циклу програмного забезпечення. Було визначено ключові фази, загальні для більшості проєктів: постановка вимог, проєктування архітектури, втілення, тестування, розгортання та супровід. У межах цього дослідження, особливу увагу було приділено тому, як саме ці етапи реалізуються у проєктах з використанням Expo та без нього, зокрема у класичних підходах React Native з нативною збіркою.

Важливим моментом аналізу стало порівняння втілення ключових функціональностей: взаємодії з API пристрою, попереднього перегляду застосунку, публікації в маркетах. З'ясовано, що Expo надає готові модулі для доступу до камери, геолокації, push-нотифікацій, а також дозволяє використовувати Expo Go для миттєвого тестування на реальних пристроях без збирання проєкту. У класичній розробці ці можливості

реалізуються вручну або через підключення сторонніх бібліотек з потребою додаткової конфігурації нативного коду.

Проведене порівняння дозволило виявити не лише переваги використання Expo, але й обмеження. Наприклад, інтеграція нестандартних нативних модулів у рамках Managed Workflow, режим розробки в Expo, який автоматизує більшість процесів: конфігурацію середовища, доступ до API пристрою, тестування й публікацію [1]. Процес розробки ускладнюється або унеможливлюється, що змушує переходити до стороннього програмного забезпечення таке як, Bare, режим розробки в Expo, що надає цілковитий доступ до нативного коду Android (Java/Kotlin) та iOS (Objective-C/Swift). Він підходить для важких проєктів, де необхідно реалізувати нестандартні функції або застосовувати власні нативні модулі [2]. Такі висновки особливо важливі для проєктів, що потребують високої кастомізації або взаємодії з апаратним забезпеченням на нижчому рівні.

На рисунку 1 зображена діаграма класів, вона демонструє ключові етапи розробки мобільного додатку використовуючи інструментарій Expo, на 2 рисунку зображена діаграма класів без використання Expo.

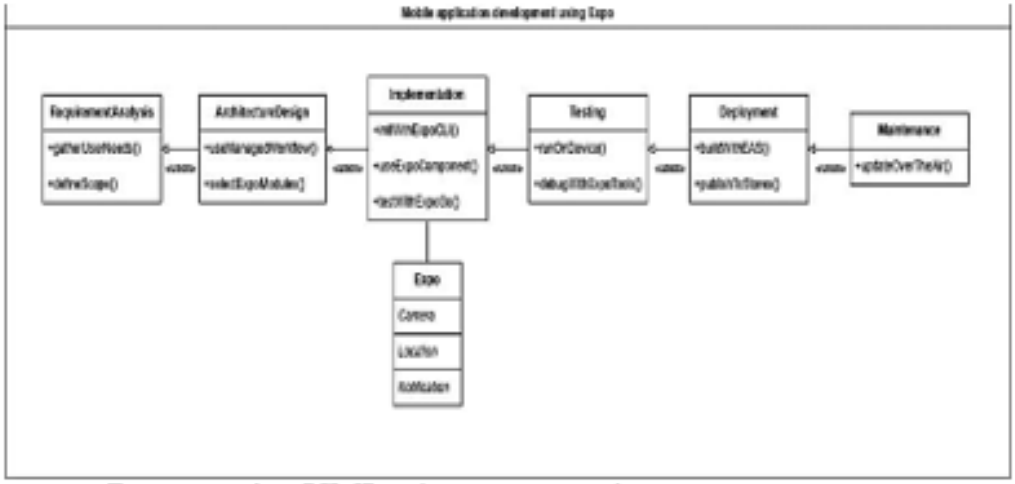

Рисунок 1 – UML-діаграма класів з використанням інструментарію Expo

Рисунок 2 – UML-діаграма класів з використанням нативного середовища

Основні відмінності в етапах розробки:

1. Налаштування середовища, Expo дає змогу налаштувати середовище розробки за лічені хвилини через CLI, тоді як без Expo розробникові доводиться витрачати набагато більше часу на встановлення та конфігурацію додаткового програмного забезпечення таке як, Android Studio або Xcode.

2. Розробка інтерфейсу(ArchitectureDesign), завдяки Expo розробка інтерфейсу починається одразу після ініціалізації проєкту, на відміну від традиційного підходу, де чимало часу витрачається на ручне підключення залежностей та створення базової структури.

3. Робота з API пристрою(Implementation), Експо пропонує готові модулі для доступу до камери, геолокації, push-сповіщень, що усуває потребу у важкій інтеграції з нативними API.

4. Тестування(Testing), у процесі тестування з інструментарій Expo дає змогу негайно спостерігати зміни на справжніх пристроях через додаток Expo Go, в той час як звичний підхід потребує складання проєкту для кожного тесту або запуску на емуляторі.

5. Збірка та публікація(Deployment), Експо автоматизує процес складання та випуску застосунків через EAS Build, тоді як без нього розробник мусить власноруч, знову ж таки, використовувати додатковий програмний супровід, такий як, Gradle або Xcode, а також витрачати час для налаштуванням маркетів.

6. Супровід(Maintenance), оновлення додатку через Expo EAS Update не потребує повторної публікації в App Store або Google Play, на відміну від звичного підходу.

На основі даного аналізу UML-діаграм основних етапів розробки мобільного додатку можна зробити наступний висновок. Інструментарій Expo суттєво зменшує час на розробку мобільного додатку, через своє функціональне та гнучке середовище. Expo зменшує поріг входу у мобільну розробку, роблячи її доступною навіть для новачків, тоді як традиційний підхід передбачає знання нативних інструментів і суттєво більшу технічну підготовку.

Список використаних джерел

1. Managed workflow. Expo Documentation. URL: https://docs.expo.dev/bare/overview/#new-workflows (date of access: 9.05.2025).
2. Bare workflow. Expo Documentation. URL: https://docs.expo.dev/versions/latest/config/metro/#bare-workflow-setup (date of access: 10.05.2025).

УДК 004.4

АКТУАЛЬНІ ПРАКТИКИ СТВОРЕННЯ ВЕБПЛАТФОРМ З ВИКОРИСТАННЯМ REACT ТА СУЧАСНОГО СТЕКУ ТЕХНОЛОГІЙ

Гольонко Максим Анатолійович
Луцький національний технічний університет, здобувач групи ІПЗ-41, golonko.m1309@lntu.edu.ua
Ліщина Наталія Миколаївна
Луцький національний технічний університет, к.т.н., доцент, завідувач кафедри інженерії програмного забезпечення, n.lishchyna@lutsk-ntu.com.ua

CURRENT PRACTICES OF CREATING WEB PLATFORMS USING REACT AND MODERN TECHNOLOGY STACK

Maksym Holonko
Lutsk National Technical University, Student of the Group IPZ-41, golonko.m1309@lntu.edu.ua
Nataliia Lishchyna
Lutsk National Technical University, Ph.D., Associate Professor, Head of the Department of Software Engineering, n.lishchyna@lutsk-ntu.com.ua

СЕРТИФІКАТ

Кафедра цифрових освітніх технологій

Кафедра інженерії програмного забезпечення

Даний сертифікат засвідчує, що

Виримчук Богдан Ігорович

брав(-ла) участь у

X Міжнародній науково-практичній конференції
з проблем вищої освіти і науки
"Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"
загальним обсягом 15 годин (0,5 кредитів ECTS),

яка проводилася 23-24 травня 2025 року у м.Луцьк
на базі Луцького національного технічного університету

Ректор ЛНТУ _____ **Ірина ВАХОВИЧ**

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

№ ІТОНВ-2025-25

Програма конференції: https://itonv.lntu.edu.ua/files/2025/program-itonv-2025.pdf