Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Мобільний застосунок для керування контактами та взаємодії на основі React Native»

08-54.БКР.003.00.000 ПЗ

Виконав: студент 4 курсу, групи 1СП-21б
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма 123 — «Системне програмування»
Адамчук І. В.

Керівник phd., ст.викл. каф. ОТ
Обертюх М. Р.

«____» _____2025 р.

Рецензент к.т.н., доц., доцент каф. ПЗ
Ракитянська Г. Б.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«19» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф._____Азаров О. Д.
«21» березня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Адамчуку Ігору Віталійовичу

1 Тема роботи: «Мобільний застосунок для керування контактами та взаємодії на основі React Native», керівник роботи Обертюх М.Р. phd., ст.викл. каф. ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року № 97.

2 Строк подання студентом роботи 13.05.2025 р.

3 Вихідні дані до роботи: актуальність, аналіз аналогів, розробка додатка, структурна схема, алгоритм роботи, тестування.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз сучасного стану методів і засобів управління соціальними і професійними контактами, огляд технологічного стеку та інструментів розробки, програмна реалізація мобільного застосунку, тестування мобільного застосунку, висновки, список використаних джерел, додатки.

5 Перелік графічного матеріалу: екран нагадувань та планування подій

6 Консультанти розділів роботи представлені у таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 4	phd., ст.викл. каф. ОТ Обертюх М.Р	21.03.2025	13.05.20
Нормоконтроль	ас. каф. ОТ Швець С.І.	14.05.2025	30.05.20

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Підпис
1.	Постановка задачі роботи	21.03.2025	21.03.2025	виконано
2.	Аналіз літературних джерел. Попередня розробка основних розділів	21.03.2025	23.03.2025	виконано
3.	Розробка технічного завдання (ТЗ)	24.03.2025	26.03.2025	виконано
4.	Розробка алгоритму роботи програмного засобу	27.03.2025	01.04.2025	виконано
5.	Програмна реалізація мобільного застосунку	02.04.2025	20.04.2025	виконано
6.	Тестування розробленого програмного засобу	21.04.2025	25.04.2025	виконано
7.	Оформлення пояснювальної записки	26.04.2025	12.05.2025	виконано
8.	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.2025	13.05.2025	виконано

Студент Ігор АДАМЧУК

Керівник роботи phd., ст.викл. Максим ОБЕРТЮХ

АНОТАЦІЯ

УДК 004.94

Адамчук І.В. Мобільний застосунок для керування контактами та взаємодії на основі React Native. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2025, 73 с.

На укр. мові. Бібліогр.: 20 назв; рис.: 4; табл.: 2.

У бакалаврській кваліфікаційній роботі був розроблений мобільний додаток для управління соціальними та професійними контактами на платформі React Native. Досліджено наявні методи та інструменти для зберігання та організації контактної інформації, а також вибрані оптимальні технології для розробки додатку. Для реалізації кросплатформного додатку було обрано React Native, а для зберігання даних використано Firebase Firestore, що забезпечує синхронізацію даних у реальному часі.

Ключові слова: мобільний додаток, кросплатформна розробка, React Native, Firebase, Redux, управління контактами, NoSQL бази даних, синхронізація даних в реальному часі.

ABSTRACT

Adamchuk I.V**.** Mobile Application for Managing Contacts and Interactions Based on React Native. Bachelor's Bachelor qualification work on specialty 123 "Computer Engineering", educational Program "System Programming". Vinnytsia: VNTU, 2025, 73 p.

In Ukrainian language. Bibliography: 20 titles; figures: 4; tables: 2.

In this bachelor's qualification work, a mobile application was developed for managing social and professional contacts on the React Native platform. Existing methods and tools for storing and organizing contact information were explored, and optimal technologies for the application's development were selected. React Native was chosen for cross-platform development, and Firebase Firestore was used for data storage, providing real-time data synchronization.

Keywords: mobile application, cross-platform development, React Native, Firebase, Redux, contact management, NoSQL databases, real-time data synchronization.

ЗМІСТ

ВСТУП

Актуальність теми обумовлена тим, що у сучасному світі, де цифрові технології займають все більш значущу роль, соціальні та професійні зв’язки стали важливою складовою повсякденного життя. Для багатьох людей ефективне управління такими контактами є основою для досягнення успіху в кар’єрі, особистих зв’язках та бізнесі. З розвитком цифрових технологій, зокрема мобільних додатків, постала нагальна потреба у створенні зручних та ефективних інструментів для управління контактами [1]. У цьому контексті особливо важливими є платформи, що об’єднують як соціальні, так і професійні зв’язки в одному місці.

В Україні, де процеси цифровізації активно розвиваються як у повсякденному житті, так і в професійній діяльності [2], виникає потреба у створенні зручних мобільних додатків для організації та управління контактами. Однак на сьогоднішній день більшість існуючих рішень орієнтовані або на соціальні мережі, або на професійні платформи, що не дозволяє користувачам ефективно управляти своїми зв’язками в обох аспектах одночасно. Тому питання створення багатофункціонального мобільного додатку для управління контактами, який би поєднував соціальні та професійні зв’язки, є особливо актуальним.

Метою роботи є розробка мобільного додатку на платформі React Native [3], який дозволить користувачам зручно управляти як соціальними, так і професійними контактами.

Завданнями цієї роботи є:

— аналіз існуючих рішень для управління контактами;

— проектування структури та функціоналу мобільного додатку;

— реалізація основних функцій для зберігання контактної інформації, налаштування сповіщень, управління взаємодіями та інтеграції з іншими сервісами;

— проведення тестування додатку на відповідність потребам користувачів та оцінка ефективності.

Об’єктом дослідження є мобільний додаток для управління соціальними та професійними контактами.

Предметом дослідження є алгоритми і програми мобільного додатку для зберігання контактної інформації, управління нагадуваннями, організації взаємодії з іншими користувачами.

Практичне значення цієї роботи полягає в підвищенні ефективності управління контактами через створення мобільного додатку, що дозволить користувачам організовувати як соціальні, так і професійні зв'язки в одному зручному інтерфейсі. Завдяки цьому додатку користувачі зможуть зберігати, категоризувати та взаємодіяти з контактами, а також отримувати нагадування про важливі події та взаємодії. Цей інструмент стане корисним для студентів, молодих спеціалістів [4], бізнесменів і всіх тих, хто активно використовує цифрові технології для розвитку особистих і професійних зв’язків.

Розробка мобільного застосунку для управління контактами є актуальною в умовах активної цифровізації суспільства, зокрема для користувачів в Україні, де важливість мобільних рішень для організації особистих та професійних справ зростає. Впровадження такого додатку дасть змогу значно підвищити ефективність комунікації та управління контактами, оптимізувати робочі процеси та зекономити час на взаємодії з важливими зв'язками. Це рішення стане важливим інструментом для тих, хто прагне удосконалити свою організацію взаємодій у цифровому середовищі.

1 АНАЛІЗ СУЧАСНОГО СТАНУ МЕТОДІВ І ЗАСОБІВ УПРАВЛІННЯ СОЦІАЛЬНИМИ І ПРОФЕСІЙНИМИ КОНТАКТАМИ

1.1 Поняття соціальних і професійних контактів

Соціальні та професійні контакти є важливими елементами в сучасному житті, адже вони визначають рівень комунікації та взаємодії в будь-якому суспільстві [5]. Соціальні контакти відносяться до взаємин між людьми, які сприяють обміну інформацією, ідеями та емоціями, а також створенню взаємодопомоги в різних сферах діяльності. Професійні контакти, у свою чергу, відображають зв'язки, які необхідні для виконання робочих завдань, обміну досвідом і розвитку кар'єри [6]. Управління такими контактами є важливим аспектом для ефективної діяльності, як на особистому, так і на професійному рівні.

Сучасне суспільство, завдяки розвитку технологій, дає можливість для створення більш гнучких і зручних методів взаємодії, зокрема за допомогою мобільних додатків [7]. Зокрема, застосунки для управління соціальними і професійними контактами дозволяють ефективно зберігати інформацію, планувати зустрічі, координувати співпрацю і контролювати процеси взаємодії з іншими людьми в реальному часі. Це забезпечує більшу зручність для користувачів і підвищує ефективність комунікаційних процесів.

1.2 Специфіка роботи з соціальними та професійними контактами

Соціальні і професійні контакти мають свою специфіку в залежності від того, для яких цілей вони використовуються. Соціальні контакти, як правило, мають неформальний характер і здебільшого зберігаються в рамках приватних комунікацій. Це можуть бути знайомства, друзі, родина, а також партнери по хобі чи інші особи, з якими людина взаємодіє в повсякденному житті.

Професійні контакти мають більш формальну структуру і спрямовані на досягнення певних робочих цілей, таких як виконання проектів, досягнення кар'єрних висот, налагодження співпраці між компаніями або окремими професіоналами. Управління такими контактами вимагає ретельного підходу до

зберігання даних, планування взаємодії, а також відстеження ефективності комунікації для подальшого розвитку стосунків.

Серед важливих завдань, які можуть бути вирішені за допомогою мобільних додатків для управління контактами, виділяються наступні:

— збереження та систематизація контактної інформації — зберігання даних про осіб та організації, з якими здійснюється взаємодія;

— аналіз ефективності взаємодії — відстеження, як часто і яким чином взаємодіяти з конкретними контактами для досягнення кращих результатів;

— налаштування сповіщень та нагадувань — отримання нагадувань про важливі події, зустрічі або комунікації;

— інтеграція з іншими сервісами — можливість синхронізувати контакти з іншими додатками та платформами, наприклад, соціальними мережами або поштовими сервісами [8].

1.3 Технології управління контактами: теоретичний огляд

Управління контактами вимагає використання спеціальних інструментів, які дозволяють не лише зберігати контактні дані, але й здійснювати з ними різноманітні операції, такі як відстеження комунікацій, створення заміток, планування подій та багато іншого. З розвитком технологій, зокрема мобільних платформ і додатків [9], з'явилася можливість більш ефективно організувати управління контактами.

Одним з основних напрямків є використання мобільних додатків на базі React Native, що дозволяють створювати кросплатформенні рішення для Android і iOS [10]. React Native дає змогу розробникам будувати мобільні додатки з однаковим кодом для двох платформ, що робить розробку значно дешевшою і швидшою [11]. Оскільки цей фреймворк має доступ до широкого спектру компонентів і можливостей, таких як бази даних, інтеграція з іншими додатками та сервісами, це дозволяє створювати мобільні додатки, які задовольняють всі потреби користувачів для управління контактами.

Однією з ключових особливостей такого підходу є мобільність додатків. Користувачі можуть здійснювати операції, пов'язані з управлінням контактами, без необхідності перебувати за комп'ютером або в офісі. Це дозволяє значно підвищити продуктивність і зручність роботи з контактами, зокрема для бізнесменів, фахівців або студентів, яким важливо ефективно управляти своїми соціальними та професійними взаємодіями.

1.4 Існуючі рішення для управління контактами

Управління соціальними та професійними контактами в сучасному цифровому середовищі стало необхідною складовою для організації ефективної взаємодії. Існує безліч рішень для зберігання та обробки контактної інформації, і вони варіюються від простих контактних книжок до складних платформ для управління взаєминами в бізнесі та соціальних мережах. Розглянемо деякі з найбільш поширених інструментів для управління контактами, які можуть бути корисними для як особистого, так і професійного використання.

LinkedIn є однією з найбільш відомих і популярних соціальних мереж для професіоналів [12]. Вона дозволяє створювати профілі, зберігати контактну інформацію та налагоджувати зв'язки з іншими фахівцями в різних галузях. Ця платформа орієнтована на професійний розвиток і мережеве взаємодія, що дозволяє користувачам ефективно будувати кар'єру, знайомитися з новими можливостями для працевлаштування та співпраці, а також взаємодіяти через публікації та обговорення в тематичних групах.

Проте, попри велику популярність LinkedIn, ця платформа має деякі обмеження для управління не тільки професійними контактами, а й особистими. Наприклад, хоча вона пропонує можливість збереження контактів та їх категоризації, функціонал організації особистих зустрічей або інтеграції з іншими календарями є обмеженим [13]. Крім того, наявність платних функцій (наприклад, LinkedIn Premium) для доступу до більш детальної інформації може бути бар'єром для деяких користувачів.

Google Contacts є одним з найпоширеніших і доступних рішень для зберігання контактної інформації [14]. Цей інструмент інтегрується з іншими сервісами Google, такими як Gmail, Google Calendar, Google Meet і іншими. Однією з основних переваг є синхронізація даних через обліковий запис Google, що дозволяє користувачам мати доступ до своїх контактів з будь-якого пристрою, на якому є підключення до інтернету.

Google Contacts забезпечує основні функції збереження і систематизації контактів, дозволяючи додавати номери телефонів, адреси електронної пошти, фізичні адреси та інші важливі дані. Водночас, Google Contacts більше орієнтований на базове використання та не надає функцій для інтеграції з іншими платформами або для управління складними професійними взаємодіями, такими як налаштування персональних нагадувань чи аналіз ефективності контактів.

HubSpot CRM — це система управління взаємодіями з клієнтами, орієнтована в основному на малий та середній бізнес. Вона дозволяє зберігати всі контактні дані та історію взаємодій із клієнтами в єдиній платформі, що допомагає побудувати ефективну стратегію взаємодії та розвитку бізнесу. Однією з головних особливостей HubSpot є інтеграція з іншими інструментами для автоматизації процесів, що включає налаштування автоматичних листів, нагадувань і сповіщень.

Цей інструмент надає більш складний набір функцій порівняно з іншими рішеннями, такими як Google Contacts, і здатний вирішувати завдання з управління не тільки персональними, а й професійними контактами, особливо у бізнес-середовищі. Важливою перевагою є можливість аналізу взаємодій, що дозволяє користувачам оцінювати ефективність комунікацій та адаптувати стратегію взаємодії з клієнтами. Однак для індивідуальних користувачів або тих, хто не займається бізнесом, HubSpot може бути надмірно складним і функціонально перевантаженим, з багатьма додатковими опціями, які можуть бути не потрібні.

Contacts+ — це мобільний додаток, який спеціалізується на управлінні контактами та синхронізації їх з іншими платформами, такими як Facebook, LinkedIn, Twitter та інші [15]. Додаток дозволяє автоматично зберігати інформацію про контакти, збагачувати її даними з соціальних мереж, що робить цей інструмент

корисним для тих, хто активно взаємодіє в цифровому середовищі. Contacts+ також пропонує можливість керувати записами, відстежувати взаємодії та створювати нагадування для важливих подій, таких як зустрічі чи ділові зустрічі.

Contacts+ є відмінним вибором для тих, хто хоче зберігати всі свої контакти в одному місці, а також має потребу в інтеграції з соціальними мережами для більш зручного доступу до інформації про своїх знайомих, колег і партнерів. Однак цей додаток може не підходити для бізнес-користувачів, які потребують більш складних функцій для управління клієнтами або іншими професійними контактами. Контакти, що зберігаються в Contacts+, можуть бути не зовсім структурованими і залежати від джерела їх отримання, що створює додаткові труднощі при управлінні великими базами даних.

1.5 Порівняння існуючих рішень з запропонованим

Існуючі платформи для управління контактами мають певні обмеження, що полягають у відсутності спеціалізованих функцій для інтеграції соціальних і професійних контактів в одному додатку. Вони здебільшого орієнтовані або на професіоналів (LinkedIn, HubSpot), або на зберігання базових контактів (Google Contacts), але не забезпечують інтеграцію з іншими комунікаційними та організаційними інструментами. Крім того, більшість цих рішень орієнтовані на зберігання контактної інформації без можливості аналізувати ефективність взаємодії.

Запропоноване мобільне рішення, засноване на React Native, дозволить поєднати переваги існуючих платформ, одночасно вирішуючи проблеми інтеграції та персоналізації досвіду. Ось основні переваги:

— інтеграція різних джерел даних: додаток може синхронізувати контакти з різних платформ (соціальні мережі, поштові сервіси, календарі);

— гнучкість і доступність: кросплатформне рішення на базі React Native дає можливість використовувати додаток як на Android, так і на iOS пристроях.

1.6 Проблемні питання та перспектива розвитку

Основною проблемою, яка стоїть перед розробниками мобільних додатків для управління контактами, є забезпечення безпеки персональних даних користувачів. Важливо розуміти, що інформація про соціальні і професійні контакти може бути чутливою, і тому додатки повинні відповідати вимогам міжнародних стандартів безпеки, таких як GDPR, для захисту особистих даних.

Ще одним викликом є інтеграція з іншими платформами. Багато користувачів використовують кілька сервісів одночасно, тому додаток має бути здатним без проблем синхронізувати дані з іншими популярними платформами (LinkedIn, Google Contacts, Facebook, Twitter тощо).

Перспективи розвитку таких додатків можуть включати використання штучного інтелекту для прогнозування найбільш ефективних способів взаємодії з контактами, а також автоматизацію певних процесів, таких як планування зустрічей і надання рекомендацій на основі попередніх комунікацій.

2 ОГЛЯД ТЕХНОЛОГІЧНОГО СТЕКУ ТА ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 JavaScript

JavaScript — це мова програмування, яка займає одну з ключових позицій у сучасній веб- та мобільній розробці [16]. Вона була створена для того, щоб додати інтерактивності веб-сторінкам, надаючи можливість обробляти події, маніпулювати елементами DOM (Document Object Model) та взаємодіяти з сервером без необхідності перезавантаження сторінки. Завдяки своїй універсальності, JavaScript не тільки залишається основою для фронтенд-розробки, але й активно використовується на серверній стороні завдяки платформі Node.js.

З моменту свого створення, JavaScript став основою для розробки складних веб-додатків і є незамінним елементом технологічного стека для багатьох програмістів. Він здатен забезпечити швидке реагування додатків на дії користувача, забезпечуючи зручність та інтерактивність інтерфейсів. В останні роки JavaScript вийшов далеко за межі веб-розробки, ставши основною мовою для кросплатформної мобільної розробки через фреймворки, такі як React Native.

JavaScript є однією з найважливіших мов програмування в сучасних технологіях завдяки своїй здатності інтегруватися з іншими веб-технологіями, такими як HTML та CSS [17]. Роль JavaScript у веб-додатках полягає в забезпеченні динамічного контенту та взаємодії з користувачем в реальному часі. Завдяки можливості працювати на клієнтській стороні, JavaScript дозволяє уникнути постійних запитів до сервера, тим самим покращуючи швидкість роботи програми та зменшуючи навантаження на сервер.

У мобільних додатках JavaScript також знаходить своє місце через використання таких фреймворків, як React Native. Цей підхід дозволяє розробляти додатки для Android та iOS з єдиною кодовою базою, що значно скорочує час і витрати на розробку, а також дозволяє досягти високої продуктивності за рахунок рідної компоненти, що використовуються в React Native.

Таким чином, JavaScript забезпечує універсальність, гнучкість і ефективність у створенні додатків, що працюють як на веб-платформі, так і на мобільних пристроях, дозволяючи підтримувати інтерактивність і високу швидкість роботи додатків.

Однією з найважливіших особливостей JavaScript є підтримка асинхронного програмування. Програми, які працюють асинхронно, можуть виконувати кілька завдань одночасно, не блокуючи основний потік виконання програми. Це особливо важливо для веб-додатків, які виконують численні операції з сервером, таких як запити до бази даних або взаємодія з API.

Механізм Promises в JavaScript дозволяє обробляти асинхронні операції, забезпечуючи відкладене виконання функцій до отримання результату. Проте для більшої зручності та зменшення кількості коду, ES6 ввів ключове слово async/await, яке дозволяє писати асинхронний код, що виглядає як синхронний, що полегшує читання і підтримку програми.

Наприклад:

```
async function fetchData() {
   let response = await fetch('https://api.example.com/data');
   let data = await response.json();
   return data;
}
```

Цей підхід робить код значно чистішим та зменшує ймовірність виникнення помилок, пов'язаних з вкладеністю колбеків (callback hell).

JavaScript активно використовує події для взаємодії з користувачем. Коли користувач натискає кнопку, вводить текст або прокручує сторінку, це викликає події, які можуть бути оброблені програмою. Обробка подій є важливою частиною JavaScript, дозволяючи додаткам реагувати на взаємодію користувача в реальному часі.

DOM (Document Object Model) — це програмний інтерфейс для HTML та XML документів. Завдяки JavaScript, розробники можуть змінювати структуру сторінки без необхідності її перезавантаження. Наприклад, можна додавати,

видаляти або змінювати елементи на сторінці на основі подій користувача. Взаємодія з DOM через JavaScript є основою для динамічних веб-сторінок, на яких змінюється контент у відповідь на дії користувача.

Приклад:

```
document.getElementById('myButton').addEventListener('click', function() {
   alert('Button clicked!');
});
```

Цей код дозволяє обробляти подію натискання кнопки, сповіщаючи користувача про його дію.

JavaScript є однією з найпростіших мов для початку програмування, що пояснюється його простотою синтаксису та широкою документацією. Мова має низький поріг входу, що дозволяє новачкам швидко освоїти основи програмування та почати створювати свої перші додатки.

Крім того, JavaScript має велику кількість інструментів та бібліотек, що полегшують розробку. Це дозволяє навіть початківцям створювати складні веб-додатки без необхідності писати великий обсяг коду з нуля.

Однією з основних причин популярності JavaScript є величезна кількість бібліотек та фреймворків, що доступні для розробників. Таких як React, Vue.js, Angular для фронтенд-розробки та Node.js для серверної. Ці фреймворки допомагають значно пришвидшити процес розробки, оскільки вони вже надають багато готових рішень для популярних завдань.

Завдяки такій великій спільноті розробників, JavaScript має величезну підтримку в інтернеті, де можна знайти рішення для будь-яких проблем. Це створює потужну екосистему, де кожен розробник може знайти необхідні інструменти для реалізації своїх ідей.

Таким чином, JavaScript є незамінною мовою програмування в сучасній розробці, яка дозволяє створювати інтерактивні та динамічні веб- і мобільні додатки, зберігаючи простоту використання і доступність для широкого кола розробників.

2.2 React

React — це бібліотека для створення інтерфейсів користувача, яка була розроблена компанією Facebook у 2013 році [18]. Спочатку React використовувався лише для внутрішніх потреб Facebook, але з часом став відкритим проектом, і сьогодні є однією з найбільш популярних бібліотек для фронтенд-розробки. React дозволяє створювати складні інтерфейси користувача завдяки своїй компонентній архітектурі, яка полегшує розробку масштабованих і підтримуваних веб-додатків.

Основною метою React є забезпечення високої продуктивності при розробці інтерактивних користувацьких інтерфейсів. Завдяки використанню віртуального DOM (Document Object Model), React зменшує кількість змін у реальному DOM, що дозволяє оптимізувати рендеринг і підвищити швидкість роботи додатка.

Історія розвитку React тісно пов'язана з потребами створення зручних і швидких інтерфейсів для великих додатків, таких як Facebook. Важливим досягненням стало введення концепції віртуального DOM [19], який значно покращив продуктивність порівняно з традиційним підходом до оновлення реального DOM. Цей підхід зберігає стан інтерфейсу в пам'яті, зменшуючи кількість перерисувань елементів на екрані.

Основні принципи роботи з бібліотекою: компонентний підхід, JSX. Одним з основних принципів роботи з React є компонентний підхід. У рамках цього підходу додаток розбивається на самостійні компоненти, кожен з яких відповідає за свою частину інтерфейсу користувача. Компоненти можуть бути простими, як, наприклад, кнопка, або складними, як форма або вся сторінка. Всі компоненти React є незалежними, їх можна повторно використовувати в різних частинах додатку. Цей підхід дозволяє розробникам зменшити дублювання коду та спрощує тестування і налагодження.

```
const element = <h1>Hello, world!</h1>;
```

У цьому прикладі JSX використовує синтаксис, подібний до HTML, але насправді є розширенням JavaScript.

React дозволяє створювати динамічні інтерфейси користувача завдяки використанню стану (State) та пропсів (Props). Стан зберігає інформацію, яка може змінюватися в ході роботи додатка, наприклад, введення тексту в поле, зміна кольору кнопки або завантаження нових даних з сервера. Зміна стану автоматично викликає повторний рендер компонента, що дозволяє динамічно оновлювати інтерфейс.

Крім того, React дозволяє ефективно керувати рендерингом за допомогою віртуального DOM. Коли відбуваються зміни в стані, React порівнює новий віртуальний DOM зі старим і виконує тільки мінімальні необхідні оновлення реального DOM, що дозволяє значно скоротити час на рендеринг.

React дозволяє створювати компоненти двома основними способами: за допомогою класових компонентів та функціональних компонентів.

Класові компоненти: Це компоненти, які визначаються за допомогою ES6 класів. Вони дозволяють використовувати всі можливості React, такі як стан (state) та життєвий цикл компонентів.

Приклад класового компонента:

```
class MyComponent extends React.Component {
  render() {
    return <h1>Hello, world!</h1>;
  }
}
```

Приклад функціонального компонента:

```
function MyComponent() {
  return <h1>Hello, world!</h1>;
}
```

Основна перевага функціональних компонентів — це їхня простота та зручність у використанні, особливо з новими хуками.

JSX — це синтаксис, що дозволяє писати компоненти React за допомогою HTML-подібного синтаксису. Хоча JSX виглядає як HTML, насправді це синтаксичне розширення для JavaScript, яке дозволяє інтегрувати логіку компонента в структуру інтерфейсу.

JSX дозволяє об'єднувати структуру користувацького інтерфейсу та поведінку додатка в одному місці, що значно полегшує розуміння коду та зменшує кількість помилок.

State — це внутрішній механізм компонента, що дозволяє зберігати дані, які можуть змінюватися під час роботи додатка. Кожен компонент React має свій власний стан, який може змінюватися через методи, такі як setState (для класових компонентів) або за допомогою хука useState (для функціональних компонентів). Зміна стану призводить до повторного рендерингу компонента з оновленими даними.

Приклад:

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Props (скорочення від "properties") — це механізм передачі даних між компонентами в React. Вони дозволяють одному компоненту передавати інформацію іншим компонентам. Props є лише для читання: вони не можуть бути змінені безпосередньо в дочірньому компоненті.

Приклад:

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
function App() {
  return <Greeting name="Alice" />;
}
```

Класові компоненти в React мають визначені етапи свого життєвого циклу. Вони дозволяють виконувати певні дії на різних етапах життя компонента, наприклад, при його створенні, оновленні або видаленні.

Основні методи життєвого циклу:

— componentDidMount: викликається після того, як компонент був змонтований у DOM;

— componentDidUpdate викликається після оновлення компонента;

— componentWillUnmount викликається перед тим, як компонент буде видалений з DOM.

З появою хуків у React 16.8, функціональні компоненти також отримали можливість використовувати логіку життєвого циклу за допомогою хука useEffect. Цей хук дозволяє виконувати побічні ефекти (запити до сервера, налаштування підписок, маніпуляції з DOM) після кожного рендерингу компонента.

Приклад:

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]);
```

Обробка подій у React значно відрізняється від традиційної обробки в HTML, хоча концептуально залишається схожою. Події в React працюють через систему синтетичних подій. Це означає, що React створює свій власний обгортник навколо стандартних подій браузера, щоб забезпечити кращу сумісність з різними браузерами та підтримку всіх сучасних функцій.

У React події передаються через спеціальні атрибути, такі як onClick, onChange, onSubmit, onMouseEnter, та інші. Важливим аспектом є те, що ці атрибути не просто передають функції для обробки подій. У React події є об'єктами, які мають свої власні методи для зупинки події, запобігання її поширенню, тощо.

Наприклад:

```
function handleClick() {
  alert('Button clicked');
}
function App() {
  return (
    <button onClick={handleClick}>Click me</button>
  );
}
```

У цьому прикладі, коли користувач натискає на кнопку, функція handleClick буде викликана, і користувач побачить сповіщення.

В React особливу увагу варто приділяти незмінності даних. Це означає, що будь-яка зміна стану чи властивостей повинна відбуватися за допомогою створення нових об'єктів, а не зміною існуючих. Це дозволяє React ефективно порівнювати попередні та нові стани компонента (за допомогою віртуального DOM) та здійснювати лише необхідні зміни, покращуючи продуктивність.

Незмінність даних також сприяє передбачуваності та зручності при розробці, оскільки легко відслідковувати всі зміни стану. Крім того, незмінність дозволяє уникнути проблем, пов'язаних з «неочікуваними» змінами даних, що можуть призводити до помилок.

Управління станом є однією з найбільш важливих частин розробки додатків на React. Оскільки React використовує компонентний підхід, кожен компонент має свій власний стан (State), але коли додаток стає більш складним, з'являється потреба в управлінні глобальним станом, який буде доступний для кількох компонентів одночасно.

Локальний стан — це стан, який зберігається всередині одного компонента. Це ідеальний вибір для невеликих, ізольованих компонентів, які не потребують взаємодії з іншими частинами програми. Для роботи з локальним станом React використовує хук useState у функціональних компонентах, а в класових компонентах використовувався метод this.setState.

Context API в React дозволяє передавати дані між компонентами без необхідності передавати пропси вручну через кожен рівень компонента. Це особливо корисно для глобальних даних, таких як налаштування теми або мова інтерфейсу. Використовуючи Context.Provider, можна визначити глобальний стан і зробити його доступним для всіх дочірніх компонентів.

Redux — це популярна бібліотека для управління станом у великих додатках, де необхідно централізовано зберігати та обробляти стани, які можуть змінюватися в різних частинах програми [20]. У Redux вся інформація зберігається в одному об'єкті store, і для зміни стану використовуються actions та reducers. Redux

допомагає підтримувати зрозумілу та передбачувану логіку додатку, що дуже важливо для великих проектів з багатьма компонентами.

Простий приклад використання Redux:

```
// action
const increment = () => ({
  type: 'INCREMENT'
});
// reducer
const counter = (state = 0, action) => {
  switch(action.type) {
    case 'INCREMENT':
      return state + 1;
    default:
      return state;
  }
};
```

Redux дозволяє централізовано обробляти всі зміни стану, що зручно для великих додатків, де багато компонентів взаємодіють між собою.

Якщо додаток стає великим і складним, важливо враховувати оптимізацію продуктивності, щоб уникнути неефективного рендерингу компонентів та зберегти хорошу швидкість роботи навіть при великих обсягах даних.

React.memo — це функція вищого порядку, яка дозволяє оптимізувати функціональні компоненти, запобігаючи їх перерендеренню, якщо їхні пропси не змінилися. Вона порівнює попередні та нові пропси і рендерить компонент тільки в разі необхідності.

PureComponent — це спеціальний клас для компонентів, який працює за аналогічним принципом. Власне, PureComponent автоматично порівнює пропси та стан і лише тоді ініціює рендеринг, коли дані змінилися.

Ледача ініціалізація та відкладений рендеринг є техніками, що дозволяють зменшити час завантаження додатка, завантажуючи компоненти або ресурси лише тоді, коли це справді необхідно. Це важливо для покращення користувацького досвіду, особливо при роботі з великими додатками.

Ледача ініціалізація передбачає, що певні елементи або дані ініціалізуються лише коли вони дійсно потрібні. Наприклад, компоненти можуть бути завантажені по мірі того, як користувач взаємодіє з додатком.

Відкладений рендеринг дозволяє відкладати рендеринг певних частин інтерфейсу до того моменту, коли вони фактично з'являться на екрані, тим самим зменшуючи початковий час завантаження додатку.

React також підтримує lazy loading через динамічне імпортування компонентів:

```
const MyComponent = React.lazy(() => import('./MyComponent'));
```

Це дозволяє завантажувати компонент тільки тоді, коли він необхідний, що зменшує розмір початкового пакету і прискорює рендеринг

Ці стратегії дозволяють оптимізувати додатки на React і забезпечити високу продуктивність навіть при великій кількості компонентів або складних взаємодіях з даними.

2.4 Firebase Firestore (База даних)

Firebase Firestore — це хмарна NoSQL база даних, розроблена компанією Google як частина платформи Firebase. Вона зберігає дані у вигляді документів, які є частинами колекцій. Такий підхід дозволяє створювати гнучкі та ефективні схеми для зберігання різноманітної інформації. Однією з ключових переваг Firestore є підтримка масштабованості, що дозволяє додаткам зростати без необхідності переписувати структуру бази даних. Це робить Firestore ідеальним рішенням для розробки додатків, таких як наш — мобільний додаток для керування соціальними та професійними контактами.

Завдяки своїй архітектурі, Firestore дозволяє легко зберігати дані про користувачів, їх контакти, профілі та різні взаємодії між ними. Це забезпечує можливість ефективного управління великими обсягами даних, що необхідно для створення багатофункціонального додатку.

Firestore відрізняється від традиційних реляційних баз даних тим, що використовує документно-орієнтовану модель, де дані зберігаються в документах, а документи об’єднуються в колекції. Цей підхід дозволяє зберігати дані різних структур, що значно спрощує роботу з даними, особливо коли структура документа може змінюватися з часом. Для нашого проекту це дуже зручно, оскільки контактна інформація може змінюватися або доповнюватися новими атрибутами (наприклад, додаванням нових способів зв'язку або профілів у соціальних мережах).

Простіше кажучи, кожен контакт, який користувач додає у додаток, може бути окремим документом, що містить різні поля, як-от ім’я, прізвище, телефон, електронну пошту, а також додаткові дані, такі як категорії контактів чи нотатки.

Масштабованість: Однією з головних переваг Firestore є його здатність автоматично масштабуватися в залежності від навантаження, що є важливим для нашого проекту. Наприклад, якщо додаток стане популярним і кількість користувачів збільшиться, Firestore дозволить без проблем обробляти велику кількість запитів та зберігати величезні обсяги даних про контакти, не вимагаючи від розробника додаткових налаштувань або оптимізацій.

Одна з найбільш корисних функцій Firestore для нашого мобільного додатку є синхронізація в реальному часі. Оскільки додаток орієнтований на управління соціальними та професійними контактами, важливо, щоб усі зміни, які вносяться одним користувачем (наприклад, зміна інформації про контакт), миттєво відображалися на пристроях інших користувачів. З Firestore, кожен раз, коли відбуваються зміни в базі даних, всі підключені клієнти автоматично отримують оновлення, без необхідності вручну оновлювати дані.

Наприклад, якщо один користувач додає новий контакт або оновлює інформацію про свого знайомого, ці зміни будуть миттєво доступні всім, хто має доступ до цієї інформації.

Firestore дозволяє зберігати дані в документах, кожен з яких може містити різноманітні поля, що визначають характеристики об'єкта (наприклад, ім’я, професія, тип взаємодії з контактом). Кожен документ зберігається в колекції, що

дозволяє зручно організовувати дані, наприклад, у колекціях «contacts» або «users», що зберігають відповідно інформацію про контакти та користувачів додатку.

Ця організація даних дає можливість швидко знайти та відновити будь-яку інформацію, що важливо при роботі з великою кількістю контактів.

Firestore надає потужні можливості для створення запитів, що дозволяє гнучко фільтрувати та сортувати дані. Користувачі нашого додатку можуть шукати контакти за різними критеріями: за іменем, категорією, професією чи іншими атрибутами. Наприклад, якщо користувач хоче знайти всіх своїх контактів у певній професійній сфері, він може легко фільтрувати дані за допомогою запиту.

Приклад запиту для фільтрації контактів за категорією:

```
db.collection("contacts")
  .where("category", "==", "business")
  .orderBy("name")
  .get()
  .then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
      console.log(doc.data());
    });
  });
```

Оскільки додаток орієнтований на особисті дані користувачів, таких як їх соціальні та професійні контакти, важливо правильно налаштувати доступ до цих даних. Firebase надає можливість використовувати Security Rules, які дозволяють встановити обмеження для кожного користувача. Це дозволяє забезпечити конфіденційність, гарантуючи, що користувачі можуть бачити та редагувати лише свої власні дані, а не дані інших.

Приклад налаштування правил доступу:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /contacts/{contactId} {
      allow read, update: if request.auth != null && request.auth.uid == contactId;
      allow create: if request.auth != null;
    }
  }
}
```

Ці правила забезпечують, що тільки авторизовані користувачі можуть отримувати доступ до своїх контактів та редагувати їх.

Щоб інтегрувати Firestore в наш додаток на React Native, ми використовуємо Firebase SDK для JavaScript. За допомогою цього SDK ми можемо виконувати всі необхідні операції з базою даних, такі як додавання, оновлення, читання та видалення документів.

Приклад підключення та взаємодії з Firestore:

```
import { getFirestore, collection, getDocs } from 'firebase/firestore';
import { initializeApp } from 'firebase/app';
const firebaseConfig = { /* ваша конфігурація Firebase */ };
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
async function getContacts() {
  const querySnapshot = await getDocs(collection(db, "contacts"));
  querySnapshot.forEach((doc) => {
    console.log(doc.id, " => ", doc.data());
  });}
```

За допомогою методу onSnapshot() можна налаштувати слухачів для відстеження змін у базі даних. Це дозволяє миттєво оновлювати інтерфейс, коли один з користувачів змінює інформацію про контакт, і ці зміни автоматично відображаються на пристроях інших користувачів.

```
import { onSnapshot, collection } from "firebase/firestore";
const unsubscribe = onSnapshot(collection(db, "contacts"), (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    console.log(change.type, change.doc.id, change.doc.data());
  });
});
```

Це забезпечує синхронізацію даних в реальному часі, що є важливим для функціональності додатку, орієнтованого на динамічне управління соціальними та професійними контактами.

Інтеграція Firebase Firestore в мобільний додаток для управління контактами дозволяє створити гнучку, масштабовану та зручну систему для зберігання та обробки даних, забезпечуючи високий рівень взаємодії з користувачами завдяки синхронізації в реальному часі та контролю доступу через безпекові правила.

2.5 React Native

React Native — це потужний фреймворк для розробки мобільних додатків на JavaScript, який був розроблений компанією Facebook. Його основна мета — дозволити створювати кросплатформні мобільні додатки для iOS та Android з єдиною кодовою базою. React Native дозволяє використовувати один і той самий код для розробки додатків як для Android, так і для iOS, що значно знижує витрати часу та ресурсів на створення та підтримку додатків для двох платформ. За допомогою цього фреймворку можна створювати високопродуктивні мобільні додатки, які мають рідний вигляд та відчуття (native look and feel).

Однією з головних переваг React Native є те, що замість того, щоб писати окремий код для кожної платформи, розробники можуть створювати додатки, які працюють на обох платформах, використовуючи один і той самий код, завдяки можливості компіляції JavaScript коду в рідний код для кожної з платформ. Це забезпечує високу продуктивність і дозволяє значно заощаджувати час на розробку і тестування.

React для веб-розробки і React Native для мобільних додатків мають спільну основу — це одна бібліотека для створення інтерфейсів користувача на JavaScript. Однак, між ними є суттєві відмінності:

— компоненти;

— рендеринг;

— стилі;

— API доступу до пристроїв.

У React для вебу ми працюємо з HTML елементами (наприклад, , <button>, <input>), тоді як у React Native замість стандартних веб-компонентів використовуються компоненти, специфічні для мобільних платформ, такі як <View>, <Text>, <Button>, <Image>. Ці компоненти є абстракцією для рідних елементів Android або iOS, і вони працюють так, як їхні рідні аналоги.

React для вебу рендерить HTML і CSS, використовуючи браузер, в той час як React Native рендерить елементи на нативних компонентах платформ. Це дає

мобільним додаткам на React Native можливість працювати з рідними функціями пристрою та отримувати більш високу продуктивність, ніж у традиційних веб-додатках.

У React для вебу стилі пишуться за допомогою CSS або CSS-in-JS, тоді як у React Native стилі визначаються через StyleSheet, що є специфічним для мобільної платформи.

React Native надає доступ до рідних API пристрою (GPS, камера, сенсори тощо), що недоступно в стандартному React для вебу.

React Native вирішує одну з найбільших проблем розробників мобільних додатків — необхідність створювати окремі додатки для кожної платформи. Завдяки React Native розробники можуть використовувати одну кодову базу для iOS та Android, що значно знижує витрати на розробку та обслуговування додатків.

Ключовим аспектом тут є мостовий підхід: React Native дозволяє використовувати JavaScript для бізнес-логіки та рендеринг інтерфейсу, а сам додаток компілюється в нативний код для кожної з платформ. Це дозволяє використовувати рідні компоненти та API, що дає додаткам високу продуктивність, порівняно з гібридними додатками або веб-додатками.

Основний принцип React Native, як і в React для вебу, — це компоненти. Кожен компонент в React Native відповідає за певну частину інтерфейсу користувача. Наприклад:

— <View> — контейнер для організації елементів на екрані;

— <Text> — використовується для відображення тексту;

— <Button> — стандартна кнопка, яку можна використовувати для взаємодії з користувачем;

— <Image> — для відображення зображень.

Ці компоненти дозволяють будувати складні інтерфейси, які виглядають і поводяться як нативні елементи на Android та iOS.

У React Native стиль компонента задається через StyleSheet, що є об'єктом JavaScript, який містить стилі для елементів додатку. На відміну від традиційного

CSS, StyleSheet не підтримує всі властивості CSS, але надає достатньо інструментів для створення стильних мобільних інтерфейсів.

Приклад:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#fff',
  },
  text: {
    fontSize: 20,
    color: '#333',
  },
});
```

React Native надає вбудовану можливість визначати, на якій платформі працює додаток, використовуючи Platform. Це дозволяє розробникам включати специфічний код для кожної платформи, наприклад, змінювати вигляд компонентів або використовувати різні функціональності залежно від того, чи працює додаток на Android чи iOS.

Приклад:

```
import { Platform } from 'react-native';
if (Platform.OS === 'ios') {
  // код для iOS
} else {
  // код для Android
}
```

Для кожної платформи React Native має набір компонентів, які оптимізовані для певної платформи. Наприклад, компонент DatePickerIOS доступний тільки для iOS, а для Android доступний DatePickerAndroid. Це дозволяє створювати інтерфейси, які виглядають рідно на обох платформах, а також використовувати платформенезалежні компоненти, що адаптуються під кожну платформу.

Однією з основних задач мобільних додатків є ефективне управління навігацією між різними екранами або розділами додатку. React Navigation — це

одна з найбільш популярних бібліотек для організації навігації в React Native додатках. Вона дозволяє створювати стеки навігації, перехід між екранами, вкладену навігацію, таби, бічні меню та інші навігаційні елементи.

Основними видами навігації в React Native є:

— Stack Navigation (Стек навігації): дає змогу переходити від одного екрану до іншого, при цьому попередні екрани залишаються в стеку, і до них можна повернутися;

— Tab Navigation (Навігація через таби): дозволяє користувачам перемикатися між екранами через вкладки внизу екрана;

— Drawer Navigation (Бічне меню): забезпечує доступ до різних частин додатку через бокове меню, яке можна відкрити свайпом.

Приклад створення навігації через стек екранів:

```
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import HomeScreen from './HomeScreen';
import ProfileScreen from './ProfileScreen';
const Stack = createStackNavigator();
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Використання цієї бібліотеки в нашому мобільному додатку дозволяє створити зрозумілу і зручну навігацію між екранами, такими як "Список контактів", "Деталі контакту", "Налаштування" тощо.

Однією з важливих переваг React Native є доступ до рідних можливостей мобільного пристрою через API. Завдяки цьому можна інтегрувати такі функціональні можливості, як GPS, камеру, сенсори руху тощо, без необхідності написання рідного коду для кожної платформи.

У нашому додатку, орієнтованому на управління соціальними та професійними контактами, може виникнути потреба використовувати GPS для відстеження місцеположення контактів або створення нагадувань про важливі зустрічі в певних локаціях. Також можна інтегрувати камеру для сканування візиток або створення профілів користувачів за допомогою фото.

Приклад використання GPS:

```
import Geolocation from '@react-native-community/geolocation';
Geolocation.getCurrentPosition(
  (position) => {
    console.log("Latitude: ", position.coords.latitude);
    console.log("Longitude: ", position.coords.longitude);
  },
  (error) => console.log(error)
);
```

Для інтеграції з камерою можна використати популярну бібліотеку React Native Camera. Це дасть змогу користувачам сканувати візитки, зберігати фото контактів або здійснювати інші дії, що покращують взаємодію з додатком.

React Native також підтримує інтеграцію з різноманітними сенсорами, такими як акселерометр, гіроскоп або датчики руху. Це дозволяє додавати додаткові можливості для вашого додатку. Наприклад, для певних інтерактивних функцій або ігор можна використовувати ці сенсори, або навіть для створення "динамічних" нагадувань або взаємодій з користувачем.

2.6 Операційна система Android та емулятор Android Studio

Android — це операційна система для мобільних пристроїв, створена компанією Google. Вона є найбільш популярною в світі, і підтримує широкий спектр пристроїв, від смартфонів і планшетів до телевізорів і носимих гаджетів. Оскільки Android є відкритою платформою, розробники мають можливість створювати різноманітні додатки для цієї системи, використовуючи доступ до численних функцій пристроїв — від камери та GPS до датчиків руху та мікрофонів.

У нашому випадку, для мобільного додатку для управління соціальними та професійними контактами Android надає нам зручний доступ до функцій, таких як

GPS для відстеження місцезнаходження контактів або камера для сканування візиток. Це дозволяє зробити додаток більш інтерактивним і корисним для користувачів.

Особливості Android для мобільної розробки включають можливість гнучко налаштовувати додатки під різні моделі пристроїв, а також використовувати різні апаратні функції, такі як сенсорні екрани, GPS, Bluetooth, камери та багато іншого. Ці функції дозволяють створювати багатофункціональні додатки, які відповідають вимогам користувачів і забезпечують зручну взаємодію з пристроєм.

Android SDK (Software Development Kit) — це набір інструментів для розробки додатків на платформі Android. Для роботи з SDK використовується Android Studio — офіційне середовище розробки для Android. Це середовище включає все необхідне для створення, тестування та налагодження Android-додатків, що робить процес розробки значно зручнішим.

Налаштування Android Studio для розробки включає в себе налаштування Android SDK, а також емуляторів, що дозволяє тестувати додатки на різних пристроях без необхідності мати фізичний телефон. Для нас, як розробників додатку для управління контактами, це дає можливість перевіряти, як додаток виглядатиме і працюватиме на різних пристроях без необхідності перевіряти кожен окремий телефон.

Однією з найбільших переваг Android є доступ до сенсорів, камери, мікрофона, GPS та інших апаратних можливостей. Ці можливості є важливими для додатків, які взаємодіють з навколишнім середовищем і пристроєм. Для нашого проекту, де необхідно зберігати та обробляти інформацію про контакти, Android дозволяє нам використовувати камеру для сканування візиток або геолокацію для відстеження місцезнаходження контактів.

Завдяки Android ми можемо реалізувати в додатку корисні функції, які дозволяють користувачам швидко додавати нові контакти, переглядати їх на карті або отримувати автоматичні нагадування про важливі події, з урахуванням геолокації.

Переваги Android для кросплатформної розробки полягають у тому, що за допомогою таких технологій, як React Native, ми можемо створити один додаток, який буде працювати як на Android, так і на iOS. Це дозволяє значно зекономити час і ресурси на розробку, одночасно забезпечуючи високу продуктивність і доступ до всіх можливостей пристроїв.

2.7 Емулятор Android Studio

Емулятор Android Studio — це інструмент, який дозволяє тестувати додатки без необхідності використовувати фізичні пристрої. Це дуже корисно, оскільки дає змогу симулювати роботу додатка на різних моделях смартфонів і планшетів, не маючи їх під рукою. Він дає можливість перевірити, як додаток виглядатиме на різних екранах, з різними характеристиками і версіями Android.

Переваги використання емулятора очевидні: це дозволяє заощадити час на тестуванні, не вимагаючи підключення кожного пристрою вручну. Крім того, емулятор підтримує перевірку роботи додатка в різних умовах, таких як низька швидкість мережі, обмежений рівень заряду батареї або погане з’єднання Wi-Fi.

Щоб протестувати додаток на різних пристроях, можна налаштувати емулятор Android Studio. Для цього потрібно вибрати модель пристрою, версію Android, розмір екрана та інші характеристики, щоб симулювати реальні умови використання додатка. Це дозволяє тестувати додаток на різних версіях Android і перевіряти його роботу на різних типах пристроїв.

Огляд інтерфейсу емулятора дозволяє легко змінювати параметри пристрою, такі як роздільна здатність екрану, обсяг пам'яті або потужність процесора. Це дозволяє адаптувати тестування під різні умови та переконатися, що додаток працює стабільно на будь-якому пристрої.

Хоча емулятор і є потужним інструментом для тестування, тестування на реальних пристроях є важливим етапом в процесі розробки мобільного додатку. Це дає можливість перевірити реальну продуктивність, а також виявити можливі проблеми, які можуть виникнути через специфіку роботи з апаратними

компонентами. Тестування на реальних пристроях також дозволяє перевірити, як додаток взаємодіє з такими функціями, як GPS, камера, сенсори.

Підключення Android пристрою до комп'ютера для тестування — це простий процес, який включає в себе увімкнення режиму розробника на пристрої, підключення через USB-кабель і запуск додатка на фізичному пристрої. Це дозволяє побачити, як додаток працює в реальних умовах, і виявити будь-які проблеми, які можуть бути не помічені при тестуванні на емуляторах.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

У цьому розділі розглядається програмна реалізація мобільного застосунку для управління соціальними та професійними контактами, створеного на основі React Native. Він логічно продовжує попередні розділи, в яких було проаналізовано існуючі рішення та технологічний стек, і демонструє, як теоретичні задуми було втілено у практичний інструмент.

Метою розділу є докладний опис структури додатку, ключових модулів і механізмів, що забезпечують його функціонування. Особлива увага приділяється організації даних, реалізації основних можливостей, таких як керування контактами, пошук і фільтрація, а також взаємодії з базою даних у реальному часі.

Крім того, розглядається користувацький інтерфейс, який забезпечує інтуїтивність і зручність роботи, а також інтеграція з апаратними можливостями мобільного пристрою, що розширює функціонал і покращує користувацький досвід.

Цей розділ допоможе зрозуміти, яким чином технічні рішення та архітектурні підходи поєдналися, щоб створити сучасний, ефективний і надійний мобільний додаток, здатний задовольнити потреби користувачів у цілісному управлінні контактами.

3.1 Архітектура застосунку

Архітектура мобільного додатку, розробленого на React Native, орієнтована на використання компонентного підходу, що є основою сучасної фронтенд-розробки. Така архітектура дозволяє створювати модульні компоненти, кожен з яких відповідає за свою частину логіки або інтерфейсу. Це сприяє зручності розробки, масштабованості, а також полегшує тестування та подальшу підтримку додатку.

Компоненти інтерфейсу користувача відповідають за візуальне відображення даних і взаємодію з користувачем. Кожен екран, будь то список контактів, деталі контакту, форми для редагування або налаштування, реалізований як окремий

React-компонент. Цей підхід дозволяє забезпечити модульність і гнучкість при розробці, де кожен екран або компонент може бути змінений або розширений без впливу на інші частини додатку.

Всі компоненти мають чітко визначені функції:

— екрани списку контактів відображають список усіх контактів, надаючи можливість фільтрації та сортування;

— екрани деталей контактів показують повну інформацію про конкретний контакт з можливістю редагування або видалення;

— форми для додавання або редагування контактів дозволяють користувачам оновлювати дані або додавати нові записи;

— налаштування додатку дають можливість змінювати персональні параметри користувача, синхронізувати дані та налаштовувати сповіщення.

Цей компонентний підхід значно спрощує тестування, оскільки кожен компонент можна тестувати незалежно від інших.

Для забезпечення ефективної взаємодії з базою даних Firebase Firestore створено спеціалізовані сервіси. Вони інкапсулюють усю бізнес-логіку роботи з базою даних та виконують CRUD-операції (створення, читання, оновлення та видалення). Завдяки цьому компоненти інтерфейсу не мають безпосереднього доступу до бази даних, що дозволяє розділити логіку доступу до даних та їх відображення.

Цей підхід підвищує безпеку та зручність підтримки, оскільки зміни в логіці роботи з даними можна впровадити без необхідності змінювати компоненти інтерфейсу. Це також дає змогу централізовано змінювати структуру даних у разі масштабування або зміни вимог до додатку.

У додатку використано Redux для централізованого управління станом, що дозволяє зберігати всі глобальні дані в одному місці. Це дає змогу компонентам отримувати та змінювати дані без необхідності передавати пропси через кілька рівнів. Використання Redux дозволяє ефективно синхронізувати інтерфейс та дані в реальному часі, що підвищує продуктивність і зручність розробки, особливо при роботі з великими обсягами динамічних даних, таких як контакти та нагадування.

Завдяки централізованому управлінню станом, додаток може забезпечити більш стабільну та передбачувану поведінку, що робить його більш надійним і простим у тестуванні.

Навігація між екранами додатку реалізована через бібліотеку React Navigation, яка підтримує стекову, табову та drawer-навігацію. Це дозволяє користувачеві безперешкодно переміщатися між різними розділами додатку, такими як список контактів, деталі контакту, налаштування та нагадування.

— стекова навігація забезпечує перехід між детальними екранами (наприклад, між екраном списку контактів і екраном деталей контакту);

— табова навігація дозволяє швидко перемикатися між основними функціями додатку, такими як список контактів, нагадування та налаштування;

— Drawer-навігація дає доступ до додаткових налаштувань і функцій через бічне меню.

Цей підхід дозволяє створити логічно послідовну та зручну навігацію для користувача, що покращує його досвід роботи з додатком.

Для зберігання контактної інформації та інших даних використано Firebase Firestore — документно-орієнтовану базу даних, яка ідеально підходить для мобільних додатків завдяки своїй гнучкості та здатності синхронізувати дані в реальному часі. Використання Firestore дозволяє додатку миттєво оновлювати інформацію на екрані при будь-яких змінах у базі даних.

Firestore забезпечує високу продуктивність навіть при роботі з великими обсягами даних завдяки можливості виконувати запити безпосередньо на сервері, зменшуючи навантаження на клієнтську частину. Всі дані зберігаються в колекціях, що дозволяє гнучко організовувати структуру додатку, а також оптимізувати доступ до даних через запити з фільтрацією та сортуванням.

Нижче, у таблиці 1, наведено основні колекції і структура документів, що використовуються у додатку.

Таблиця 1 — Інформація про колекції в базі даних Firebase Firestore

Колекція	Призначення	Основні поля документа
users	Дані користувачів додатку	userId, name, email, createdAt
contacts	Контактна інформація користувачів	contactId, userId, name, phone, email, category, notes, createdAt
reminders	Нагадування, пов’язані з контактами або подіями	reminderId, contactId, userId, date, description, isActive

Ця модель дає змогу ефективно структурувати інформацію, підтримувати розділення контактів за категоріями (соціальні чи професійні), а також організовувати систему нагадувань.

Управління станом у додатку реалізоване через React Context API, що дає змогу централізовано зберігати та оновлювати ключові дані — список контактів, інформацію про користувача, налаштування та нагадування. Такий підхід дозволяє кожному компоненту отримувати необхідні дані без складних зв’язків і перепередачі пропсів.

Навігація організована таким чином, щоб користувач міг легко переходити між основними функціональними блоками. Для цього застосовується стекова навігація при переході до деталей і форм, а також табова — для швидкого перемикання між різними розділами.

Ця архітектура забезпечує не тільки логічне і структуроване розподілення відповідальностей, а й гнучкість у розвитку та масштабуванні додатку, що є важливим фактором для підтримки сучасних мобільних продуктів.

3.2 Реалізація основних можливостей

У цьому підрозділі детальніше розглянемо, як у додатку реалізовані основні функції, що забезпечують зручне, ефективне та надійне управління соціальними та професійними контактами. Мобільний додаток був розроблений із врахуванням потреб користувачів у легкому доступі до контактної інформації, її організації та ефективному використанні. Ключовими можливостями є створення, редагування,

видалення контактів, їх категоризація, фільтрація, пошук, а також планування нагадувань.

Основою додатку є можливість працювати з контактами, що дозволяє створювати нові записи, редагувати існуючі та видаляти непотрібні. Кожен контакт має набір полів, які містять базову інформацію, таку як ім’я, номер телефону, електронну пошту, а також додаткові атрибути. До таких атрибутів належать категорія контакту (соціальний або професійний), нотатки щодо конкретної особи, дата створення запису, що дозволяє відслідковувати зміни і додавати додаткову інформацію.

Для зручності користувачів при створенні або редагуванні контакту відкривається спеціальна форма, де є всі необхідні поля для введення або оновлення інформації. Користувач може заповнити ім’я, прізвище, телефонний номер, електронну пошту, вибрати категорію контакту (соціальний чи професійний), додати нотатки та інші відомості. Після заповнення форми та натискання кнопки збереження дані відправляються в базу даних Firebase Firestore.

Процес збереження даних у Firestore є безпечним та надійним. Всі дані зберігаються у вигляді окремих документів у колекції contacts. Це дозволяє додавати, редагувати та оновлювати дані в реальному часі, забезпечуючи оперативне відображення змін у користувацькому інтерфейсі.

Видалення контактів у додатку також реалізовано з додатковим підтвердженням дії, що дозволяє уникнути випадкових операцій. Після натискання кнопки видалення користувач отримує попередження з запитом на підтвердження. Це дає можливість уникнути ненавмисного видалення важливих даних, що підвищує безпеку та контроль користувача над його інформацією.

Одна з важливих функцій додатку — це система категорій для контактів. Для того, щоб краще організувати інформацію і полегшити пошук, користувач може розподіляти свої контакти на дві основні групи: соціальні та професійні. Це дає змогу швидко фільтрувати список контактів за категорією, що робить доступ до потрібної інформації набагато зручнішим.

Категорії представлені у вигляді вкладок або фільтрів у верхній частині екрана, що дозволяє користувачеві миттєво змінювати відображення списку контактів. Для кожного контакту в базі даних є поле category, яке визначає приналежність запису до певної групи. Це дозволяє користувачеві бачити лише ті контакти, які йому потрібні на даний момент, що підвищує ефективність роботи з додатком.

Пошук і фільтрація контактів є невід'ємною частиною функціоналу додатку. Користувач може вводити ім'я, номер телефону або інші атрибути контакту в пошуковий рядок. Завдяки цьому додаток динамічно фільтрує результати в реальному часі, оновлюючи список контактів. Такий підхід значно пришвидшує процес пошуку необхідної інформації.

Крім того, фільтрація дозволяє вибирати категорію контактів (наприклад, «соціальний» або «професійний»), що робить цей процес ще зручнішим. Сортування контактів за алфавітом або за датою додавання допомагає швидко знаходити необхідні записи, навіть якщо база даних значно зростає.

Для оптимізації роботи з великими обсягами даних запити виконуються на стороні Firestore з використанням методів .where() та .orderBy(). Це забезпечує швидке отримання потрібної інформації навіть у разі великої кількості контактів.

Однією з важливих можливостей є реалізація модуля нагадувань. Користувач може створювати нагадування, пов'язуючи їх із конкретними контактами або подіями. Це дозволяє не забути про важливі дати, такі як дні народження, зустрічі чи інші події, пов'язані з контактами.

Нагадування зберігаються у колекції reminders в Firestore і синхронізуються з додатком у реальному часі. Для кожного нагадування користувач може вибрати дату і час, коли потрібно отримати сповіщення, а також додати короткий опис події.

Система локальних push-повідомлень використовується для оповіщення користувача про майбутні події. Нагадування активуються у вказаний час, що допомагає користувачеві своєчасно реагувати на важливі події.

Таким чином, основний функціонал додатку побудований на принципах зручності та гнучкості, що дозволяє користувачам ефективно організовувати свої

контакти, планувати нагадування та взаємодіяти з контактною інформацією у різних сферах їхнього життя. Інтерфейс додатку спрощує виконання всіх операцій, а централізоване зберігання даних в Firestore дозволяє забезпечити надійність і масштабованість системи.

3.3 Особливості користувацького інтерфейсу

Інтерфейс користувача — це ключовий елемент мобільного додатку, що безпосередньо впливає на досвід користування та ефективність роботи з додатком. У процесі розробки додатку для управління соціальними та професійними контактами було поставлено за мету створити інтуїтивний, привабливий і функціональний інтерфейс, який допомагає швидко знаходити потрібну інформацію і легко виконувати операції з контактами.

Головний екран відображає контакти користувача, розподілені на дві категорії — соціальні та професійні (рис.1). Зверху розташована панель із вкладками або фільтрами для швидкого перемикання між категоріями. Нижче — список контактів із аватарами (за наявності), іменами та ключовою інформацією.

Пошуковий рядок дозволяє миттєво відфільтрувати контакти за іменем, номером телефону або іншими атрибутами. Результати оновлюються динамічно під час введення.

Додатково поруч з кожним контактом розташовані кнопки швидких дій (наприклад, дзвінок або повідомлення), що пришвидшує комунікацію.

Цей екран, наведений на рисунку 2, відображає повну інформацію про контакт: ім’я, телефони, електронну пошту, адресу, а також додаткові поля, наприклад нотатки чи посилання на соціальні мережі.

Також тут можна побачити історію взаємодій з цим контактом (дати останніх дзвінків, зустрічей тощо), а також пов’язані нагадування.

Внизу розташовані кнопки для редагування чи видалення контакту.

Для створення нового контакту або редагування існуючого відкривається окремий екран (рис.3) із формою, що містить поля: ім’я, прізвище, номер телефону, email, категорія, нотатки тощо.

Валідація забезпечує коректність введених даних (наприклад, перевірка формату email та номеру телефону).

Після збереження інформація миттєво оновлюється у базі даних і відображається у списку контактів.

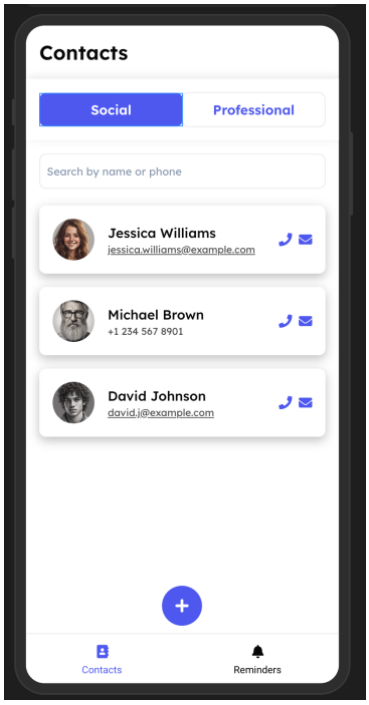

Рисунок 1 — Екран списку контактів із розподілом за категоріями

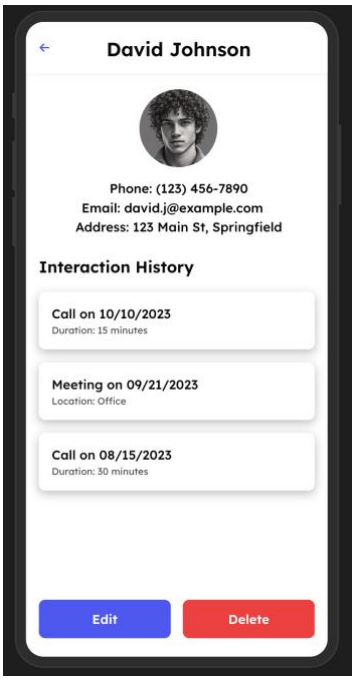

Рисунок 2 — Екран деталей контакту з інформацією та історією взаємодій

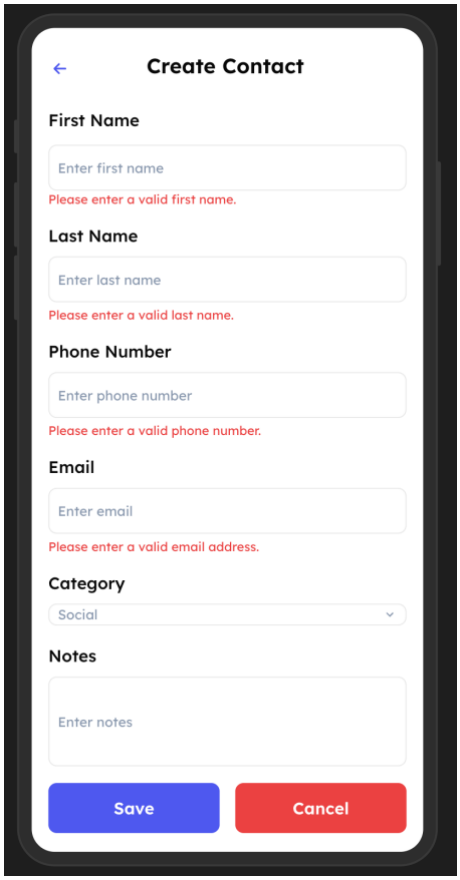

Рисунок 3 — Форма додавання нового контакту

Цей екран, наведений у додатку В.1, відображає список нагадувань з назвою події, датою та часом. Користувач може переглядати активні події, а також редагувати або видаляти їх. Кожне нагадування має кнопку для взаємодії, а список організовано за датою для зручного перегляду.

Важливими рішеннями для кращого користувацького досвіду є:

— проста та логічна навігація;

— швидкий доступ до дій;

— реакція на дії користувача;

— адаптивність.

Кнопки дій розташовані в зручних для досяжності місцях, що економить час користувача і підвищує продуктивність.

Візуальні індикатори та анімації підтверджують виконання дій, наприклад, збереження контактів або оновлення списку, що підвищує довіру і комфорт у користуванні.

Навігація побудована таким чином, щоб користувач завжди розумів, де він знаходиться і як швидко повернутися до потрібного розділу. Мінімалістичний дизайн.

Використано чіткі шрифти, помірну кольорову гаму та достатній відступ між елементами. Це допомагає зменшити візуальне навантаження і сприяє кращій сприйнятності інформації.

Додаток коректно відображається на різних розмірах екранів, що забезпечує однаково комфортний досвід на смартфонах різної конфігурації.

Завдяки цим рішенням додаток стає не просто інструментом для зберігання контактів, а зручним помічником, який полегшує організацію і підтримку важливих зв’язків.

3.4 Взаємодія з Firebase Firestore

Ефективна робота мобільного додатку значною мірою залежить від правильної організації зберігання, оновлення та доступу до даних. Для цього у нашому додатку використовується Firebase Firestore — хмарна NoSQL база даних, яка ідеально підходить для динамічних мобільних рішень завдяки підтримці масштабованості, синхронізації в реальному часі та простоті інтеграції з React Native.

Початковий етап роботи з Firestore полягає у підключенні SDK Firebase у додаток і налаштуванні конфігурації, що містить ключі доступу та параметри проєкту. Це забезпечує авторизацію і з’єднання з сервером бази даних.

```
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID"
};
```

```
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
```

Цей код створює ініціалізоване підключення до Firestore, яке використовується у всьому додатку для роботи з даними.

Для забезпечення функціоналу створення нового контакту у додатку використовується метод addDoc, який додає об’єкт із інформацією про контакт до колекції contacts у Firestore. Цей метод повертає унікальний ідентифікатор, що дозволяє надалі оперувати цим записом. Наприклад, функція для створення контакту виглядає так:

```
async function addContact(contact) {
  try {
    const docRef = await addDoc(collection(db, "contacts"), contact);
    console.log("Contact added with ID: ", docRef.id);
  } catch (e) {
    console.error("Error adding contact: ", e);
  }
}
```

Для отримання списку всіх контактів застосовується метод getDocs, який запитує всі документи з колекції. Результатом є перелік контактів, що далі відображається у відповідному компоненті інтерфейсу:

```
async function getContacts() {
  const querySnapshot = await getDocs(collection(db, "contacts"));
  querySnapshot.forEach((doc) => {
    console.log(`${doc.id} =>`, doc.data());
  });
}
```

Якщо виникає необхідність змінити інформацію про контакт, використовується метод updateDoc. Він дозволяє звернутися до конкретного документа за його унікальним ідентифікатором і оновити вибрані поля без створення нового запису:

```
async function updateContact(contactId, updatedData) {
  const contactRef = doc(db, "contacts", contactId);
```

```
  await updateDoc(contactRef, updatedData);
}
```

Для видалення контакту застосовується функція deleteDoc, що повністю вилучає відповідний документ із бази:

```
async function deleteContact(contactId) {
  await deleteDoc(doc(db, "contacts", contactId));
}
```

Ці операції інкапсульовані у сервісному шарі додатку, що дозволяє відокремити логіку роботи з базою від логіки інтерфейсу, спрощуючи подальший розвиток і підтримку проєкту.

Однією з найбільш потужних можливостей Firestore є підтримка оновлення даних у реальному часі за допомогою підписки на зміни у колекціях чи документах.

У додатку реалізовано підписку на колекцію контактів, що дозволяє автоматично оновлювати інтерфейс при будь-яких змінах у базі, наприклад, при додаванні або редагуванні контакту іншим пристроєм.

```
import { collection, onSnapshot } from 'firebase/firestore';
const unsubscribe = onSnapshot(collection(db, "contacts"), (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    if (change.type === "added") {
      console.log("New contact: ", change.doc.data());
    }
    if (change.type === "modified") {
      console.log("Modified contact: ", change.doc.data());
    }
    if (change.type === "removed") {
      console.log("Removed contact: ", change.doc.data());
    }
  });
});
```

Такий підхід значно підвищує користувацький досвід, оскільки додаток миттєво реагує на зміни, не вимагаючи ручного оновлення.

Оскільки додаток працює з персональними даними користувачів, важливо забезпечити належний рівень захисту інформації. Для цього у Firestore

використовуються правила безпеки (Security Rules), які контролюють, хто і які операції може виконувати.

Приклад простих правил, що дозволяють користувачу читати і змінювати лише свої дані:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /contacts/{contactId} {
      allow read, write: if request.auth != null && request.auth.uid == resource.data.userId;
    }
  }
}
```

Ці правила гарантують, що користувачі не зможуть отримати доступ до чужих контактів, що є важливою вимогою для конфіденційності.

Firestore надає потужні можливості для формування запитів, що дозволяє фільтрувати і сортувати дані вже на рівні бази, зменшуючи обсяг переданих даних і підвищуючи продуктивність.

У додатку пошук реалізований через комбіновані запити .where() і .orderBy(), що дає можливість, наприклад, шукати контакти певної категорії та сортувати їх за ім’ям.

```
import { query, where, orderBy, getDocs, collection } from "firebase/firestore";
const q = query(collection(db, "contacts"), where("category", "==", "business"), orderBy("name"));
const querySnapshot = await getDocs(q);
querySnapshot.forEach((doc) => {
  console.log(doc.id, " => ", doc.data());
});
```

Ця логіка застосовується у компонентах інтерфейсу для відображення відфільтрованого списку контактів відповідно до вибору користувача.

Таким чином, інтеграція Firebase Firestore у додаток забезпечує гнучку, надійну і масштабовану систему зберігання даних із підтримкою синхронізації в реальному часі і високим рівнем безпеки, що є запорукою якісного користувацького досвіду.

3.5 Додаткові функції та інтеграції

Окрім базового функціоналу управління контактами, мобільний додаток включає низку додаткових можливостей, які покращують користувацький досвід та розширюють сферу застосування. У цьому підрозділі розглянемо інтеграцію з апаратними API мобільного пристрою, а також організацію локальних push-повідомлень.

Для розширення функціональності додатку використано можливості мобільних пристроїв, зокрема геолокація (Geolocation API).

Додаток має можливість отримувати поточне місцезнаходження користувача або інших контактів, якщо ця інформація доступна. Це дозволяє, наприклад, відображати контакти на карті або створювати геозалежні нагадування — коли користувач заходить у певну зону, він отримує сповіщення.

Використання геолокації реалізовано за допомогою бібліотеки @react-native-community/geolocation, що забезпечує коректну роботу як на Android, так і на iOS.

```
import Geolocation from '@react-native-community/geolocation';
Geolocation.getCurrentPosition(
  (position) => {
    console.log("Latitude: ", position.coords.latitude);
    console.log("Longitude: ", position.coords.longitude);
  },
  (error) => {
    console.error(error);
  }
);
```

Таке використання розширює можливості організації нагадувань і дозволяє планувати зустрічі з урахуванням місця проведення.

Для своєчасного інформування користувача про важливі події і нагадування у додатку реалізована система локальних push-повідомлень. Це дозволяє повідомляти про заплановані зустрічі, дні народження, або інші важливі події, пов’язані з контактами.

Локальні повідомлення налаштовані таким чином, щоб активуватися у заданий час без потреби постійного підключення до інтернету, що підвищує надійність оповіщень.

Використовується пакет react-native-push-notification, який дозволяє налаштовувати повідомлення для Android та iOS.

Приклад налаштування простого локального повідомлення:

```
import PushNotification from 'react-native-push-notification';
PushNotification.localNotificationSchedule({
  message: "Нагадування про зустріч з Олексієм",
  date: new Date(Date.now() + 60 * 1000), // через 1 хвилину
});
```

Користувач може налаштовувати нагадування для кожного контакту окремо, що робить систему гнучкою і персоналізованою.

Хоча на поточному етапі додаток використовує базові можливості апаратних API, у майбутньому можна розширити функціонал, додавши, наприклад:

— сканування візиток за допомогою камери для автоматичного створення контактів;

— інтеграцію з календарями користувача для автоматичного планування подій.

Такі можливості підвищать корисність додатку і зроблять його ще більш універсальним інструментом для організації соціальних і професійних зв’язків.

Таким чином, додаткові інтеграції забезпечують більшу гнучкість і комфорт користування, підвищують ефективність організації контактів і дозволяють підлаштовувати додаток під індивідуальні потреби користувачів.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

Тестування є важливою частиною процесу розробки мобільного додатку, оскільки дозволяє забезпечити якість і стабільність продукту. Важливо, щоб додаток працював без помилок і відповідав вимогам користувачів, що дозволяє уникнути проблем у майбутньому і підвищити ефективність роботи користувачів. Тестування допомагає виявити дефекти на ранніх етапах розробки, що дозволяє їх оперативно виправити до релізу додатку.

Основною метою тестування є забезпечення надійної роботи додатку, перевірка всіх його функціональних можливостей, а також оцінка продуктивності і взаємодії з користувачем. Всі модулі та компоненти повинні бути протестовані для гарантування їх безпомилкової роботи.

4. 1 Методи тестування

У розробці мобільного додатку важливо застосовувати різноманітні методи тестування для перевірки всіх його функцій та взаємодій. Основні методи тестування, які використовуються в цьому процесі, включають тестування функціональності та тестування інтерфейсу користувача.

Тестування функціональності передбачає перевірку роботи основних функцій додатку, таких як створення, редагування, видалення контактів, налаштування нагадувань і взаємодія з базою даних. Основна мета цього тесту — перевірити, чи працюють ці функції відповідно до вимог. Для цього використовуються юніт-тести та інтеграційні тести, які забезпечують перевірку функціональності окремих компонентів і їх взаємодії.

Тестування інтерфейсу користувача зосереджено на перевірці взаємодії користувача з додатком через графічний інтерфейс. Це включає перевірку коректності відображення елементів на екрані, таких як кнопки, текстові поля, списки, а також правильність переходів між екранами. Важливим є тестування на різних типах пристроїв і різних екранах для забезпечення зручності використання додатку на різних розмірах екранів.

4.2 Типи тестів

Для забезпечення високої якості розробленого додатку застосовуються різні типи тестів, кожен з яких виконує свою роль у перевірці конкретних аспектів роботи додатку. Основними типами тестів є юніт-тести та інтеграційні тести.

Юніт-тести — це найдрібніші тести, що перевіряють окремі функції або модулі додатку. Кожна функція або метод додатку тестується незалежно від інших компонентів, щоб переконатися, що він виконує свою задачу правильно. Юніт-тести дозволяють виявити помилки на рівні окремих компонентів додатку та виправити їх ще до того, як вони стануть проблемою для більш складних інтеграцій.

Для мобільного додатку використовуються Jest або інші подібні бібліотеки для тестування JavaScript-коду. Тестуються не лише базові функції, а й логіка взаємодії з базою даних, наприклад, перевірка CRUD-операцій (створення, читання, оновлення, видалення) для контактів.

Інтеграційні тести перевіряють взаємодію між різними модулями та компонентами додатку. Вони тестують, чи правильно працюють функції разом і чи взаємодіють між собою компоненти додатку. Наприклад, після того як користувач додає новий контакт, інтеграційний тест перевіряє, чи цей контакт з’являється у списку контактів, чи синхронізується з базою даних і чи відображається на всіх екранах правильно.

Ці тести дозволяють виявити проблеми в роботі додатку, які можуть виникнути через неправильну взаємодію між різними частинами системи. Інтеграційні тести забезпечують більш високий рівень перевірки та гарантують, що додаток працює як єдина система.

4.3 Інструменти тестування

Для ефективного тестування мобільного додатку використовуються різні інструменти, що допомагають автоматизувати процеси тестування, спрощують

виявлення помилок і підвищують ефективність роботи розробників. Ось кілька інструментів, які застосовуються для тестування:

Jest — це популярний фреймворк для тестування JavaScript, який широко використовується для юніт-тестів у додатках на React Native. Jest забезпечує швидке та ефективне тестування, дозволяючи створювати тести для перевірки логіки функцій додатку.

Основні переваги використання Jest:

— підтримка мокінгу (mocking), що дозволяє замінювати реальні функції та об'єкти на імітації для тестування;

— інтеграція з іншими інструментами для тестування та покриття коду;

— можливість використання snapshot-тестів для перевірки візуальних компонентів;

— проста конфігурація та налаштування для роботи з React Native.

Для тестування інтерфейсу користувача можна використовувати інструменти, такі як React Native Testing Library. Цей інструмент дозволяє тестувати компоненти інтерфейсу без необхідності запускати реальний додаток на пристрої, а лише перевіряти, чи працюють компоненти так, як очікується, шляхом взаємодії з віртуальним DOM.

React Native Testing Library дає можливість:

— тестувати компоненти користувацького інтерфейсу;

— перевіряти, чи відображаються правильні елементи на екрані;

— симулювати взаємодію користувача з компонентами (наприклад, натискання кнопок).

Firebase Test Lab — це потужний інструмент від Google для тестування мобільних додатків на реальних пристроях в хмарі. Firebase Test Lab дозволяє протестувати додаток на різних моделях пристроїв і операційних системах, що гарантує кросплатформену сумісність і безпеку.

Основні можливості:

— автоматичне тестування додатку на реальних пристроях і емуляторах;

— перевірка на сумісність з різними пристроями та операційними системами;

— детальні звіти про тестування, які допомагають швидко знаходити проблеми.

Ці інструменти використовуються для тестування функціональності, перевірки взаємодії з іншими компонентами та оцінки продуктивності додатку.

4.4 Проведення тестування

Проведення тестування мобільного додатку є важливим етапом у процесі розробки. Це дозволяє перевірити всі функціональні можливості додатку, переконатися в його працездатності та зручності для користувачів, а також забезпечити стабільність роботи в різних умовах. Основними етапами проведення тестування є підготовка тестових сценаріїв, створення тестових кейсів і тестування в реальному середовищі.

Перед тим, як почати тестування, необхідно створити тестові сценарії, які описують конкретні умови, що перевіряються в додатку. Це можуть бути такі сценарії, як:

— перевірка додавання нового контакту;

— перевірка редагування існуючого контакту;

— перевірка видалення контакту;

— перевірка роботи функції пошуку серед контактів;

— перевірка налаштування нагадувань для контактів.

Тестові сценарії повинні покривати всі основні функціональні можливості додатку, включаючи рідкісні та крайні випадки. Це дозволить забезпечити високий рівень тестування і виявити можливі помилки.

Для кожного тесту необхідно підготувати відповідні тестові дані. Це можуть бути контактні дані, які будуть використовуватись для перевірки функцій додавання, редагування та видалення. Тестові дані мають відображати реальні

умови використання додатку, що дозволить максимально наблизити тести до реальних сценаріїв.

Тестові кейси — це індивідуальні інструкції для виконання конкретних тестів. Кожен тестовий кейс містить інформацію про:

— вхідні дані (наприклад, контакт, який додається або редагується);

— опис того, що має бути перевірено (наприклад, контакт має бути доданий до бази даних);

— очікуваний результат (наприклад, після додавання контакту він з’являється у списку контактів).

Тестові кейси допомагають систематизувати процес тестування і забезпечують точність у перевірці кожної функціональності додатку.

Тестування в реальному середовищі передбачає запуск додатку на реальних пристроях або емуляторах, що дозволяє перевірити його функціональність у реальних умовах. Це дозволяє:

— перевірити сумісність з різними пристроями і версіями операційних систем;

— оцінити продуктивність додатку;

— перевірити взаємодію з іншими додатками та системами (наприклад, синхронізація з контактами, інтеграція з іншими сервісами).

Тестування в реальному середовищі дозволяє виявити проблеми, які можуть бути не помітні під час тестування в контрольованих умовах.

Після проведення тестування важливо зробити оцінку ефективності тестування та результатів. У цьому пункті ми підсумуємо всі етапи тестування і визначимо, наскільки додаток відповідає вимогам та стандартам якості.

Під час тестування було перевірено всі основні функції додатку, включаючи створення, редагування, видалення контактів, а також налаштування нагадувань. Результати тестів показали, що:

— всі основні функції працюють коректно і відповідають вимогам;

— інтерфейс додатку стабільно працює на різних пристроях і платформах;

— продуктивність додатку є на високому рівні, навіть при великій кількості даних.

Під час тестування були виявлені незначні проблеми, такі як:

— деякі елементи інтерфейсу не зовсім коректно відображалися на екранах з малими розмірами;

— виявлено незначні затримки при збереженні контактів при поганому з'єднанні з інтернетом.

Ці проблеми були швидко вирішені шляхом оптимізації коду та покращення адаптивності інтерфейсу для різних розмірів екранів.

Для подальшого покращення якості додатку, варто:

— продовжити тестування на більш широкому колі пристроїв і в умовах низької швидкості інтернету;

— здійснити тестування на різних версіях операційних систем для забезпечення сумісності;

— впровадити тестування нових функцій, таких як сканування візиток і інтеграція з іншими платформами.

Тестування показало, що додаток готовий до використання у реальному середовищі, але також є можливість для подальшого покращення і розширення функціональності.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи була розроблена система для управління соціальними та професійними контактами, яка реалізована у вигляді мобільного додатку на платформі React Native. Під час роботи над проектом було виконано кілька ключових етапів, зокрема аналіз існуючих рішень, вибір технологічного стека, розробка функціоналу додатку та тестування.

У першому розділі було здійснено теоретичний аналіз наявних методів і засобів для управління контактами, розглянуто різні платформи та інструменти, які використовуються для зберігання контактної інформації, а також їх переваги та обмеження. Порівняння існуючих рішень допомогло визначити основні вимоги до розробленого додатку та його функціоналу.

Другий розділ був присвячений огляду технологій для розробки мобільних додатків. Для розробки клієнтської частини було обрано React Native, оскільки ця технологія дозволяє створювати кросплатформні додатки для Android та iOS, що значно знижує витрати часу та ресурсів на розробку. Для роботи з базою даних було вибрано Firebase Firestore, що забезпечує швидку синхронізацію даних у реальному часі та масштабованість додатку.

У третьому розділі була реалізована архітектура додатку, включаючи компоненти інтерфейсу, сервіси для роботи з даними, а також механізми для керування станом додатку. Окрему увагу було приділено інтеграції з Firebase Firestore для зберігання контактів, нагадувань та іншої важливої інформації. Було реалізовано базові операції CRUD, які дозволяють додавати, редагувати, видаляти та переглядати контакти.

Особливості користувацького інтерфейсу були розроблені з урахуванням принципів зручності та інтуїтивності, що дозволяє користувачам легко взаємодіяти з додатком. Навігація між екранами була реалізована через таби та стекову навігацію, що забезпечує простоту і зрозумілість інтерфейсу. Також додаток підтримує функціонал для планування нагадувань, що дозволяє користувачам ефективно організовувати важливі події.

У результаті виконаної роботи був створений мобільний додаток, який дозволяє ефективно управляти соціальними та професійними контактами, налаштовувати нагадування та інтегруватися з іншими сервісами для покращення користувацького досвіду. Подальший розвиток додатку може включати додаткові функції, такі як сканування візиток за допомогою камери або інтеграцію з календарями користувачів.

Загалом, розробка цього додатку є важливим кроком у створенні багатофункціональних рішень для мобільного управління контактами, що допоможе користувачам краще організовувати свою діяльність, підвищуючи ефективність комунікації як в особистому, так і в професійному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Карасьов О.І. Проєктування мобільних застосунків : навч. посіб. / О.І. Карасьов, І.В. Коваленко, С.В. Гарасимов. — Київ : КПІ ім. Ігоря Сікорського, 2021. — 200 с.

2. Цифрові трансформації в Україні : звіт / Ініціатива «Партнерство Відкрите суспільство». — [Б. м.] : [б. в.], 2021. — 78 с.

3. React Native. — [Електронний ресурс]. — URL: https://reactnative.dev/

4. IT-освіта в Україні: виклики та можливості. — IT Ukraine Association. — [Електронний ресурс]. — URL: https://itukraine.org.ua/uk/it-osvita-v-ukrayini-vikliki-ta-mozhlivosti/

5. Король В.В. Соціальні комунікації : навч. посіб. / В.В. Король. – Київ : Знання, 2010. – 320 с.

6. Нетворкінг. — Lemon.School. — [Електронний ресурс]. — URL: https://lemon.school/blog/shho-take-netvorking

7. Яворська Т.М. Інформаційні системи та технології : навч. посіб. / Т.М. Яворська. — Львів : Видавництво Львівської політехніки, 2017. — 352 с.

8. Управління взаємовідносинами з клієнтами (CRM). — HubSpot. — [Електронний ресурс]. — URL: https://www.hubspot.com/products/crm

9. Функціонал мобільного додатку: 10 обов'язкових складових. — Wezom. — [Електронний ресурс]. — URL: https://wezom.com.ua/ua/blog/funkcional-mobile-app

10. Кросплатформна розробка додатків: основні переваги та недоліки. — SoftServe. — [Електронний ресурс]. — URL: https://softserve.ua/uk/blog/cross-platform-development-advantages-and-disadvantages/

11. Порівняння React Native та Flutter: вибір для розробки мобільних додатків. — Stfalcon. — [Електронний ресурс]. — URL: https://stfalcon.com/uk/blog/post/flutter-vs-react-native-which-will-do-better

12. LinkedIn. — [Електронний ресурс]. — URL: https://www.linkedin.com/

13. Нетворкінг та пошук роботи у соцмережах: навіщо, як та невже працює?. — Happy Monday. — [Електронний ресурс]. — URL: https://happymonday.ua/kak-iskat-rabotu-в-socsetjah

14. Google Контакти. — [Електронний ресурс]. — URL: https://contacts.google.com/

15. Contacts + — [Електронний ресурс]. — URL: https://www.contactsplus.com/

16. Сучасний підручник з JavaScript — [Електронний ресурс]. — URL: https://uk.javascript.info/

17. HTML і CSS довідник українською — [Електронний ресурс]. — URL: https://html-css.co.ua/

18. React — [Електронний ресурс]. — URL: https://react.dev/

19. What is Virtual DOM in React — [Електронний ресурс]. — URL: https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/

20. Redux — [Електронний ресурс]. — URL: https://redux.js.org/

ДОДАТОК А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О.Д.

21 березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Мобільний застосунок для керування контактами та взаємодії на основі React Native»
08-54.БКР.003.00.000 ТЗ

Науковий керівник: phd., ст.викл. каф. ОТ
_____ Обертюх М.Р.

Студент групи 1СП-21б
_____ Адамчук І.В.

ВНТУ 2025

1 Підстави для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність теми обумовлена швидким розвитком цифрових технологій та потребою в інструментах для ефективного управління соціальними та професійними контактами. У сучасному світі, де мобільні додатки займають важливе місце в організації робочих і особистих зв'язків, виникає необхідність у створенні додатку, що дозволяє зберігати і взаємодіяти з контактами в одному місці.

1.2 Тема БКР затверджена наказом.

2 Мета БКР і призначення розробки

2.1 Метою роботи є розробка мобільного застосунку для керування контактами та взаємодії, що включає функції збереження контактної інформації, налаштування нагадувань і управління взаємодіями.

2.2 Призначення розробки — створення багатофункціонального додатку для організації та планування взаємодії з контактами, що підвищує ефективність комунікації та організації справ.

3 Вихідні дані для виконання БКР

3.1 Аналіз існуючих рішень для управління контактами та їх функціональність.

3.2 Розробка структури та функціональної схеми застосунку, визначення основних екранів і їх взаємодії.

3.3 Проектування бази даних для зберігання контактної інформації та нагадувань з використанням Firebase Firestore.

3.4 Реалізація CRUD-операцій для зберігання, редагування та видалення контактів, налаштувань сповіщень.

3.5 Виконання розрахунків для забезпечення безпеки персональних даних відповідно до чинних норм.

4. Вимоги до виконання БКР

Основні вимоги: використання React Native для кросплатформної розробки, Firebase Firestore для зберігання даних, інтеграція з іншими сервісами та можливість налаштування нагадувань. Забезпечення зручності інтерфейсу та ефективної роботи з великою кількістю контактів.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Огляд сучасного стану методів і засобів управління контактами	20.03.25	25.03.25	Аналіз існуючих рішень, розділ 1 БКР
2	Огляд технологічного стеку та інструментів розробки	26.03.25	02.04.25	Розділ 2
3	Програмна реалізація мобільного застосунку	03.04.25	14.05.25	Розділ 3
4	Тестування мобільного застосунку	15.05.25	20.05-25	Розділ 4
5	Оформлення пояснювальної записки, графічного матеріалу і презентації	21.04.25	08.05.25	ПЗ, графіч. матеріал і презентація
6	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	13.05.25	13.05.25	Оформлені документи

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до

БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123

— «Комп’ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2025; – документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Мобільний застосунок для керування контактами та взаємодії на основі React Native

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібностітекстових запозичень, виявлених у роботі системою StrikePlagiarism 4,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату,фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішенняпро оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявленоознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д., зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Тарновський М.Г., доц. каф. _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Обертюх М.Р., ст. викл. кафедри ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Адамчук І.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ КЕРУВАННЯ КОНТАКТАМИ ТА ВЗАЄМОДІЇ НА ОСНОВІ REACT NATIVE

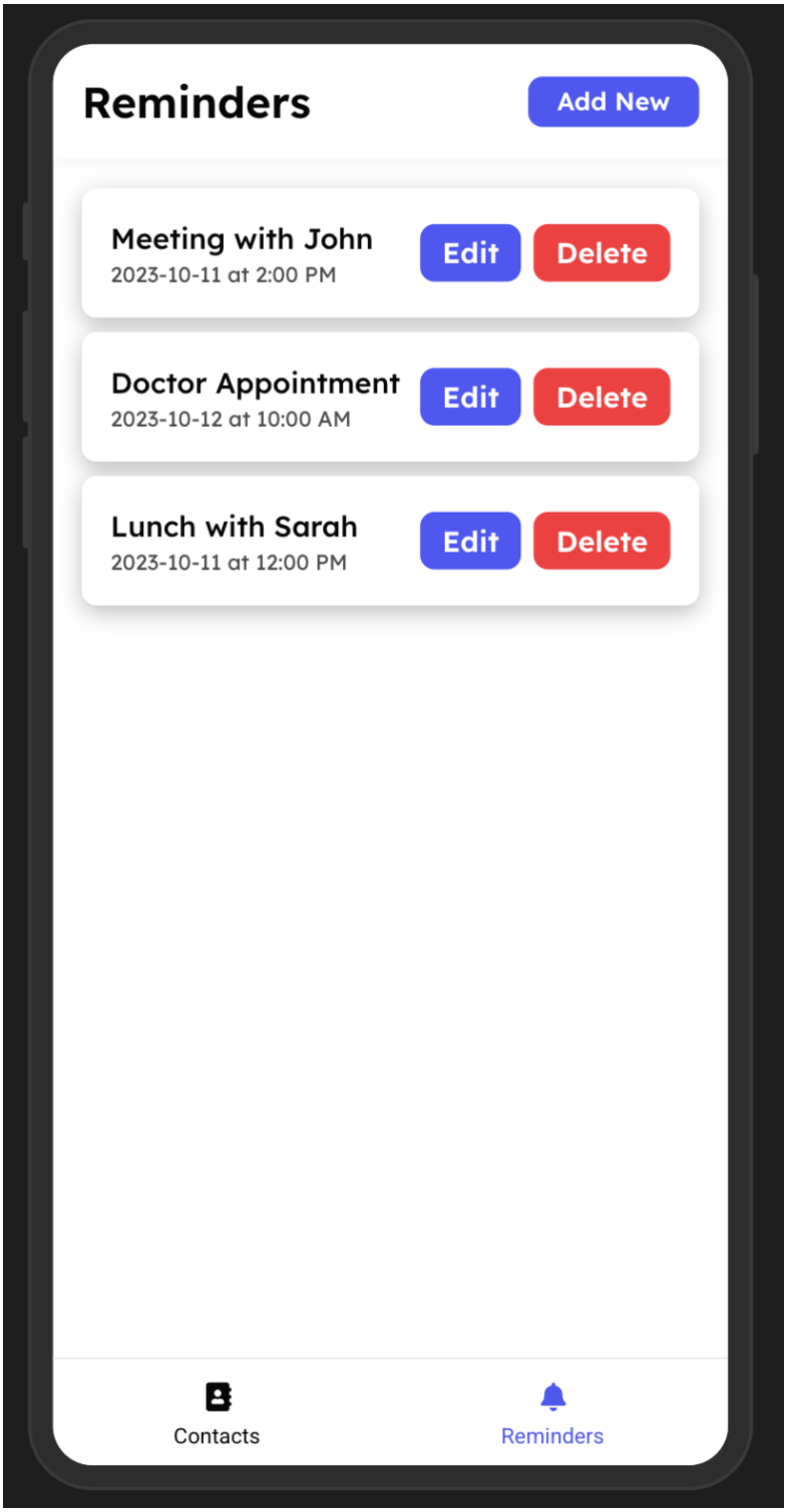

Рисунок В.1 — Екран нагадувань та планування подій

ДОДАТОК Г

Лістинг програмного забезпечення

```
// contactService.ts
import { firestore } from '../firebaseConfig';
export const getContacts = async () => {
  try {
    const snapshot = await firestore.collection('contacts').get();
    const contacts = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    return contacts;
  } catch (error) {
    console.error("Error getting contacts: ", error);
    throw error;
  }
};
export const getContactDetails = async (id: string) => {
  try {
    const doc = await firestore.collection('contacts').doc(id).get();
    if (!doc.exists) {
      throw new Error('Contact not found');
    }
    return { id: doc.id, ...doc.data() };
  } catch (error) {
    console.error("Error getting contact details: ", error);
    throw error;
  }
};
export const updateContact = async (id: string, updatedData: object) => {
  try {
    await firestore.collection('contacts').doc(id).update(updatedData);
  } catch (error) {
    console.error("Error updating contact: ", error);
    throw error;
  }
};
export const deleteContact = async (id: string) => {
  try {
    await firestore.collection('contacts').doc(id).delete();
  } catch (error) {
    console.error("Error deleting contact: ", error);
    throw error;
  }
};
export const addContact = async (contactData: object) => {
```

```
  try {
    await firestore.collection('contacts').add(contactData);
  } catch (error) {
    console.error("Error adding contact: ", error);
    throw error;
  }
};

// ContactListScreen.tsx
import React, { useState, useEffect } from 'react';
import { View, Text, FlatList, TextInput, TouchableOpacity } from 'react-native';
import { getContacts } from '../services/contactService';
import { useDispatch } from 'react-redux';
import { setContacts } from '../redux/actions';
const ContactListScreen = () => {
  const [searchTerm, setSearchTerm] = useState('');
  const [contacts, setContactsState] = useState([]);
  const dispatch = useDispatch();
  useEffect(() => {
    const fetchContacts = async () => {
      const fetchedContacts = await getContacts();
      setContactsState(fetchedContacts);
      dispatch(setContacts(fetchedContacts));
    };
    fetchContacts();
  }, []);
  const handleSearch = (text: string) => {
    setSearchTerm(text);
  };
  const filteredContacts = contacts.filter(contact =>
    contact.name.toLowerCase().includes(searchTerm.toLowerCase())
  );
  return (
    <View style={{ flex: 1, paddingTop: 40 }}>
      <TextInput
        placeholder="Search contacts..."
        value={searchTerm}
        onChangeText={handleSearch}
        style={{ padding: 10, borderWidth: 1, margin: 10 }}
      />
      <FlatList
        data={filteredContacts}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
```

```
        <TouchableOpacity onPress={() => console.log('Navigate to details')}>
          <View style={{ padding: 15, borderBottomWidth: 1 }}>
            <Text>{item.name}</Text>
            <Text>{item.phone}</Text>
          </View>
        </TouchableOpacity>
      )}
    />
  </View>
 );
};
export default ContactListScreen;

// ContactDetailScreen.tsx
import React, { useEffect, useState } from 'react';
import { View, Text, Button, TouchableOpacity } from 'react-native';
import { getContactDetails, deleteContact } from '../services/contactService';
import { useRoute, useNavigation } from '@react-navigation/native';
const ContactDetailScreen = () => {
 const [contact, setContact] = useState(null);
 const route = useRoute();
 const navigation = useNavigation();
 const contactId = route.params?.id;
 useEffect(() => {
  const fetchContactDetails = async () => {
   const contactData = await getContactDetails(contactId);
   setContact(contactData);
  };
  fetchContactDetails();
 }, [contactId]);
 const handleDelete = async () => {
  await deleteContact(contactId);
  navigation.goBack();
 };
 if (!contact) return <Text>Loading...</Text>;
 return (
  <View>
   <Text>Name: {contact.name}</Text>
   <Text>Phone: {contact.phone}</Text>
   <Text>Email: {contact.email}</Text>
   <TouchableOpacity onPress={handleDelete}>
    <Button title="Delete Contact" color="red" />
   </TouchableOpacity>
```

```
            <TouchableOpacity onPress={() => navigation.navigate('EditContact', { id:
contact.id })}>
              <Button title="Edit Contact" />
            </TouchableOpacity>
          </View>
        );
      };
      export default ContactDetailScreen;

      // EditContactScreen.tsx
      import React, { useState, useEffect } from 'react';
      import { View, TextInput, Button, Text } from 'react-native';
      import { getContactDetails, updateContact } from '../services/contactService';
      import { useRoute, useNavigation } from '@react-navigation/native';
      const EditContactScreen = () => {
        const route = useRoute();
        const navigation = useNavigation();
        const contactId = route.params?.id;
        const [contact, setContact] = useState({
          name: '',
          phone: '',
          email: ''
        });
        useEffect(() => {
          const fetchContactDetails = async () => {
            const contactData = await getContactDetails(contactId);
            setContact(contactData);
          };
          fetchContactDetails();
        }, [contactId]);
        const handleSave = async () => {
          await updateContact(contactId, contact);
          navigation.goBack();
        };
        return (
          <View>
            <Text>Name:</Text>
            <TextInput
              value={contact.name}
              onChangeText={(text) => setContact({ ...contact, name: text })}
              style={{ padding: 10, borderWidth: 1, margin: 10 }}
            />
            <Text>Phone:</Text>
            <TextInput
```

```
        value={contact.phone}
        onChangeText={(text) => setContact({ ...contact, phone: text })}
        style={{ padding: 10, borderWidth: 1, margin: 10 }}
      />
      <Text>Email:</Text>
      <TextInput
        value={contact.email}
        onChangeText={(text) => setContact({ ...contact, email: text })}
        style={{ padding: 10, borderWidth: 1, margin: 10 }}
      />
      <Button title="Save" onPress={handleSave} />
    </View>
  );
};
export default EditContactScreen;

// actions.ts
export const setContacts = (contacts: any[]) => ({
  type: 'SET_CONTACTS',
  payload: contacts
});
// reducer.ts
const initialState = {
  contacts: []
};
const contactReducer = (state = initialState, action: any) => {
  switch (action.type) {
    case 'SET_CONTACTS':
      return { ...state, contacts: action.payload };
    default:
      return state;
  }
};
export default contactReducer;

// AppNavigator.tsx
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import ContactListScreen from './screens/ContactListScreen';
import ContactDetailScreen from './screens/ContactDetailScreen';
import EditContactScreen from './screens/EditContactScreen';
const Stack = createStackNavigator();
const AppNavigator = () => {
```

```
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="ContactList">
        <Stack.Screen name="ContactList" component={ContactListScreen} />
        <Stack.Screen name="ContactDetail" component={ContactDetailScreen} />
        <Stack.Screen name="EditContact" component={EditContactScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};
export default AppNavigator;

// firebaseConfig.ts
import firebase from 'firebase/app';
import 'firebase/firestore';
const firebaseConfig = {
  apiKey: 'your-api-key',
  authDomain: 'your-auth-domain',
  projectId: 'your-project-id',
  storageBucket: 'your-storage-bucket',
  messagingSenderId: 'your-sender-id',
  appId: 'your-app-id',
};

if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
} else {
  firebase.app();
}

const firestore = firebase.firestore();

export { firestore };

// App.tsx
import React from 'react';
import { Provider } from 'react-redux';
import AppNavigator from './AppNavigator';
import store from './redux/store';

const App = () => {
  return (
    <Provider store={store}>
      <AppNavigator />
```

```
    </Provider>
  );
};
export default App;

// ContactListScreen.styles.ts
import { StyleSheet } from 'react-native';
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 40,
    paddingHorizontal: 20,
    backgroundColor: '#fff',
  },
  searchInput: {
    padding: 10,
    borderWidth: 1,
    borderColor: '#ccc',
    borderRadius: 5,
    marginBottom: 20,
    fontSize: 16,
  },
  contactItem: {
    padding: 15,
    borderBottomWidth: 1,
    borderColor: '#ddd',
  },
  contactName: {
    fontSize: 18,
    fontWeight: 'bold',
  },
  contactPhone: {
    fontSize: 14,
    color: '#555',
  },
});
export default styles;

// ContactDetailScreen.styles.ts
import { StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
```

```
    padding: 20,
    backgroundColor: '#fff',
  },
  detailText: {
    fontSize: 18,
    marginBottom: 10,
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginTop: 20,
  },
  button: {
    padding: 15,
    backgroundColor: '#007bff',
    borderRadius: 5,
    alignItems: 'center',
    width: '48%',
  },
  buttonText: {
    color: '#fff',
    fontSize: 16,
  },
  deleteButton: {
    backgroundColor: 'red',
  },
});

export default styles;

// EditContactScreen.styles.ts
import { StyleSheet } from 'react-native';
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 40,
    paddingHorizontal: 20,
    backgroundColor: '#fff',
  },
  formContainer: {
    paddingTop: 20,
  },
  input: {
    padding: 10,
```

```
    borderWidth: 1,
    borderColor: '#ccc',
    borderRadius: 5,
    marginBottom: 20,
    fontSize: 16,
  },
  button: {
    padding: 15,
    backgroundColor: '#007bff',
    borderRadius: 5,
    alignItems: 'center',
  },
  buttonText: {
    color: '#fff',
    fontSize: 16,
  },
});

export default styles;

// AppNavigator.styles.ts
import { StyleSheet } from 'react-native';
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  header: {
    backgroundColor: '#007bff',
    padding: 10,
    alignItems: 'center',
  },
  headerTitle: {
    fontSize: 20,
    color: '#fff',
  },
});
export default styles;
```