МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

ПЛОДНІК КАТЕРИНА ЮРІЇВНА

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ МЕНЕДЖМЕНТУ ФІНАНСОВИХ ОПЕРАЦІЙ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
Мамчич Тетяна Іванівна,
кандидат фіз.-мат наук, доцент
кафедри комп'ютерних наук та
кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол № _____
засідання кафедри _____
від_____2024 р.
Завідувач кафедри

(_____) _____
(підпис) ПІБ

ЛУЦЬК – 2024

ЗМІСТ

ВСТУП

У сучасному світі технології розвиваються з приголомшливою швидкістю. Ще нещодавно щоб здійснити фінансову операцію чи просто отримати інформацію про стан рахунків обов'язково потрібно було відвідувати відділення банку та стояти в довжелезних чергах. А лист, відправлений родичем з іншої країни, або навіть міста, потрібно було чекати днями, а інколи й тижнями.

Зараз технології повність перевернули наше уявлення про всі ці процеси. За допомогою мобільних додатків та інтернет-банкінгу ми можемо миттєво здійснювати фінансові операції, сплачувати комунальні послуги, податки, штрафи та навіть отримувати кредити без жодної необхідності відвідування банківських установ. Електронна пошта та різні месенджери дозволяють залишатися на зв'язку з рідними та друзями, незалежно від відстані. А сервіси по типу Zoom та Google Meet надають можливість проводити відеоконференції, онлайн-зустрічі та навіть навчання, забезпечуючи якісний зв'язок та взаємодію в режимі реального часу. Але дещо таки залишилося незмінним – бажання людей спростити облік їхніх фінансових операцій, особливо коли йде мова про групові витрати.

Актуальності набувають додатки для ефективного управління коштами у групах, оскільки вони спрощують і оптимізують розподіл коштів та боргів у групах, а також дозволяють зручно відстежувати витрати. Завдяки таким сервісам можна легко розподіляти витрати на подорожі, спільні покупки чи оренду житла, зменшуючи фінансові непорозуміння.

Метою роботи є розробити додаток на основі фреймворку React Native мови програмування JavaScript , що буде надавати можливість менеджменту фінансових операцій. У додатку буде можливість аутентифікації та авторизації, та набір інструментів для створення та управління групами людей, а також їх витратами.

Для реалізації поставленої задачі нам потрібно виконати наступне:

- визначити функціональних можливостей проекту;

- вивчити технології JavaScript, необхідні для реалізації проекту;
- дослідити принципи роботи фреймворку, для створення додатку;
- провести аналіз схожих додатків на ринку;
- визначити архітектурні принципи для розробки додатка;
- налаштувати можливості аутентифікації та авторизації за допомогою Firebase;
- спроектувати базу даних використовуючи Firestore;
- локалізувати додаток за допомогою react-i18next;
- розробити навігацію додатку;
- зверстати всі необхідні екрани для забезпечення візуалізації функціональних можливостей додатку;
- провести тестування розробленого застосунку;

Об’єкт дослідження – мобільний додаток для менеджменту фінансових операцій.

Предмет дослідження – налаштування процесів керування групами та фінансовими операціями у цих групах.

РОЗДІЛ 1 БАЗОВІ КОНЦЕПЦІЇ У МЕНЕДЖМЕНТІ ФІНАНСІВ. АЛГОРИТМ МІНІМІЗАЦІЇ ПЕРЕДАЧІ БОРГІВ. НАБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ДОДАТКУ ДЛЯ УПРАВЛІННЯ КОШТАМИ.

1.1 Сутність фінансового менеджменту

Фінансовий менеджмент слід розглядати як комплексне явище з різноманітними формами прояву. З функціональної точки зору, це система економічного управління та складова фінансового механізму. З інституційної перспективи, фінансовий менеджмент виступає як орган управління. З організаційно-правової точки зору, він є видом підприємницької діяльності.

Фінансовий менеджмент являє собою специфічну систему управління грошовими потоками, рухом фінансових ресурсів і організацією фінансових відносин.

Фінансовий менеджмент, як багатогранне поняття, охоплює різноманітні аспекти і може бути розглянутий з різних точок зору. Згідно з наведеною схемою (рис 1.1), це поняття включає в себе кілька ключових компонентів, кожен з яких має своє особливе значення та роль у загальній системі:

- наука: фінансовий менеджмент є науковою дисципліною, яка вивчає принципи та методи управління фінансами. Вона базується на теоретичних дослідженнях і використовує математичні моделі для аналізу фінансових процесів та прийняття рішень.
- філософія, мистецтво і творча діяльність: важливим аспектом фінансового менеджменту є його філософська та мистецька складова. Це мистецтво ухвалення рішень у складних фінансових ситуаціях, яке вимагає творчого підходу та інтуїції.
- система управління фінансами: фінансовий менеджмент можна розглядати як систему управління фінансами організації. Вона включає в себе структуру, процедури та інструменти, необхідні для ефективного контролю та координації фінансових ресурсів.

- процес управління: це динамічний процес, що включає планування, організацію, мотивацію та контроль фінансової діяльності. Процес управління фінансами охоплює всі етапи фінансового циклу від залучення ресурсів до їх розподілу та використання.
- орган управління фінансами: фінансовий менеджмент включає в себе органи управління, які відповідають за прийняття рішень та реалізацію фінансової політики. Це можуть бути фінансові відділи, департаменти чи спеціальні комісії в межах організації.
- вид підприємницької діяльності: фінансовий менеджмент є також видом підприємницької діяльності, яка спрямована на ефективне використання фінансових ресурсів з метою досягнення максимального прибутку. Це включає інвестиційну діяльність, управління ризиками та фінансове планування.

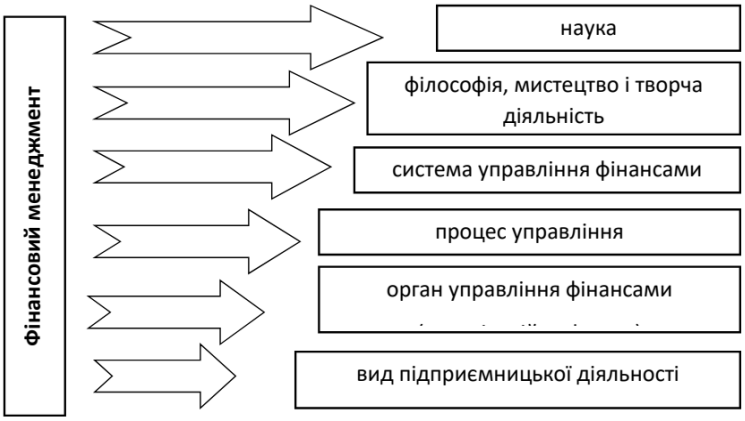


Рис. 1.1 Фінансовий менеджмент як багатоаспектне поняття

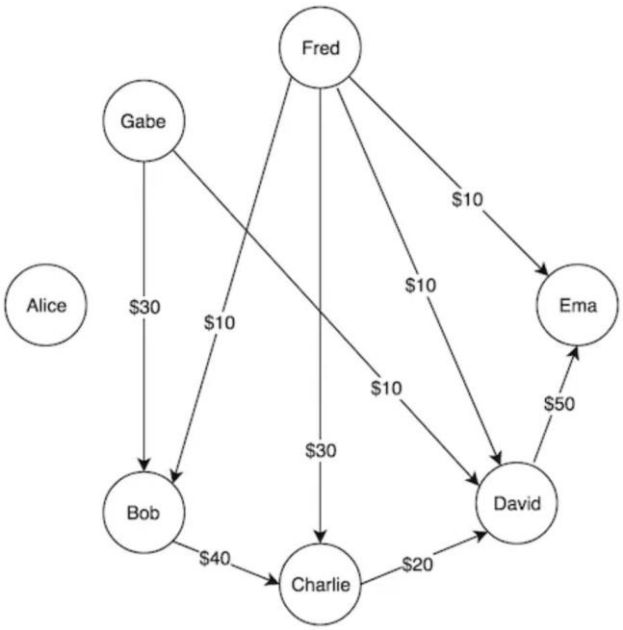
1.2 Алгоритм мінімізації передачі боргів

Спрощення боргів — це функція, яка реструктуризує борги в групах людей. Це не змінює загальну суму боргів, але полегшує повернення людям, мінімізуючи загальну кількість платежів.

Наприклад: Анна, Яна і Оля ділять квартиру. Анна винна Олі 200 гривень, а Оля винна Яні 200 гривень. Замість здійснення двох окремих платежів алгоритм «сказав би» Анні сплатити 200 гривень безпосередньо Яні, таким чином мінімізуючи загальну кількість зроблених платежів. Це гарантує, що люди отримають гроші швидше та ефективніше.

Виведемо алгоритм, який може працює під капотом цієї функції, за допомогою наведеного нижче прикладу.

Розглянемо групу з семи людей, а саме: Аліса, Боб, Чарлі, Девід, Ема, Фред і Гейб. Вони разом поїхали в тур і в кінці туру зрозуміли, що мають наступні борги: Гейб винен Бобу 30 доларів, Гейб винен Девіду 10 доларів, Фред винен Бобу 10



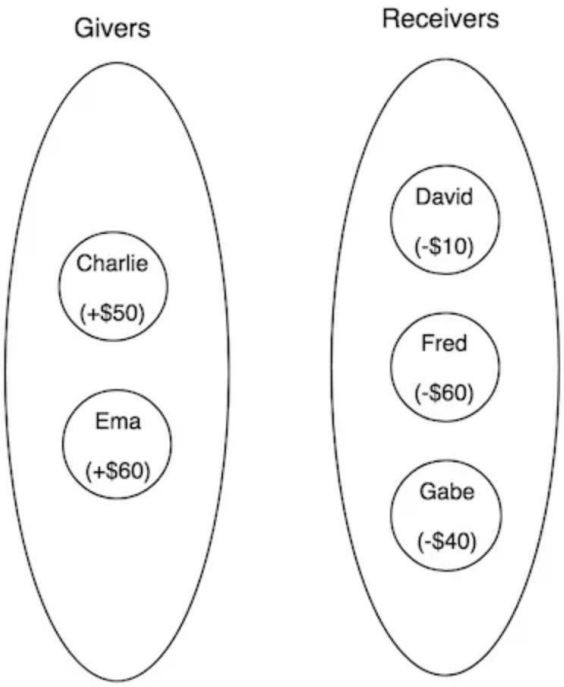
доларів, Фред винен Чарлі 30 доларів, Фред винен Девіду 10 доларів, Фред винен Емі 10 доларів, Боб винен Чарлі 40 доларів, Чарлі винен Девіду 20 доларів, Девід винен Емі 50 доларів. Для кращого розуміння інформацію наведено у вигляді орієнтованого графіка (рис 1.2).

Рис. 1.3 Представлення боргів у формі орієнтованого графа

Тепер, оскільки метою функції спрощення боргів є мінімізація загальної кількості платежів, здійснених у групі, розробимо алгоритм, що стоїть за нею.

Першим кроком є визначення чистого балансу кожної особи в групі, яку можна визначити за такою формулою: чистий баланс = (сума отриманих грошей - сума позичених грошей).

Наприклад, чистий баланс Чарлі становить 50 доларів, яка розраховується, наступним чином: чистий баланс (Чарлі) = (сума отриманих грошей (Чарлі) - сума позичених грошей (Чарлі)) = ((40+30) - 20) = 50. Таким чином, загальна чиста зміна готівки становить 0 доларів для Аліси, 0 доларів для Боба, 50 доларів для Чарлі, -10 доларів для Девіда, 60 доларів для Еми, -60 доларів для Фреда та -40 доларів для Гейба.



Тепер ми розділяємо їх на два типи людей (рис. 1.3), а саме:

- кредитори (ті, хто має додаткові гроші, що позначається позитивним значенням в чистому балансі);
- боржники (ті хто отримує додаткові гроші, що позначається від’ємним значенням в чистому балансі).

Рис. 1.4 Кластеризація кредиторів і боржників у дві різні групи

У наведеному прикладі Чарлі та Ема є кредиторами, тоді як Девід, Фред і Гейб є боржниками. Аліса та Боб уже розрахувалися з боргами, тому вони не є ані кредиторами, ані боржниками.

Щоб мінімізувати загальні платежі, які необхідно здійснити для погашення боргів, боржники повинні переказувати гроші лише кредиторам, а кредитори мають отримувати гроші лише від боржників. Крім того, можна зазначити, що загальна сума грошей, яку заборгували боржники, завжди дорівнює загальній сумі грошей, які мають отримати кредитори.

Оскільки наша мета полягає в тому, щоб мінімізувати загальну кількість платежів, які необхідно здійснити, ми повинні переконатися, що кожен кредитор переказує гроші найменшій можливій кількості боржників. Інакше, кредитор зрештою здійснить більше транзакцій, що збільшить загальну кількість операцій. Це означає, що для будь-яких певних боргів ми завжди маємо перевіряти сценарій, коли кожен кредитор здійснює лише одну транзакцію до певного боржника, щоб погасити його борги. У цьому випадку загальна кількість транзакцій буде дорівнювати кількості кредиторів і, отже, буде оптимальним рішенням. Таким чином, в оптимальному сценарії кожен кредитор переказуватиме гроші рівно одному боржнику. Це означає, що кожен боржник повинен або отримати всі гроші від одного кредитора, або взагалі не приймати від нього жодної суми.

Наприклад, нехай G1, G2, G3 і G4 представляють суми, які винні чотири кредитори відповідно. Також нехай R1 і R2 представляють суми, які мають отримати два боржники. Тепер будь-який боржник може або прийняти всю суму G1 від першого кредитора, або не приймати жодної суми взагалі. Аналогічно, він може або прийняти всю суму G2 від другого кредитора, або не приймати жодної суми, і так далі. Це, у свою чергу, означає, що ми шукаємо підмножину кредиторів, яка точно дорівнює сумі боржника, і ми повинні зробити це для всіх боржників. Це не що інше, як задача про суму підмножин, за винятком того, що тут нам може бути надано більше однієї суми (залежно від кількості боржників).

Те, про що ми говорили, стосується оптимального випадку. Також може бути, що найкраще можливе рішення вимагає, щоб деякі кредитори здійснили більше однієї транзакції. У такому разі нам доведеться робити більше, ніж ми планували для оптимального випадку, тобто перевіряти всі можливі способи розподілу суми від кредиторів. Це свідчить про те, що проблема спрощення боргів є щонайменше такою ж складною, як задача про суму підмножин, і тому вона є NP-повною. Це також означає, що не існує поліноміального часу вирішення цієї проблеми, і що для мінімізації загальної кількості платежів буде потрібно експоненціальна кількість кроків.

Це підводить нас до важливого питання: чи використовує процес спрощення боргів можливий експоненціальний алгоритм для вирішення NP-повної проблеми в реальному часі? Під час аналізу цієї проблеми, було виявлено, що існують три основні правила, яких дотримується процес спрощення боргів:

- кожен має однакову чисту суму наприкінці;
- ніхто не винен людині, якій він не був винен раніше;
- ніхто не винен більше грошей у сумі, ніж він був винен до спрощення.

Перше і третє правила є такими, що за замовчуванням повинні дотримуватися. Але що для нас більш цікаво, це друге правило, яке говорить: "Ніхто не винен людині, якій він/вона не був винен раніше".

Отже, проблема зводиться до варіювання суми, що передається по існуючих транзакціях, без введення нових. Це алгоритмічно переводиться до наступного: дано орієнтований граф, що представляє борги, змінити (якщо потрібно) ваги на існуючих ребрах без введення нових.

Тепер питання в тому, як це зробити. Це фактично можна вирішити, якщо розділити проблему на дві частини:

1. Чи буде існуюче ребро (тобто транзакція) частиною графа після спрощення боргів?
2. Якщо воно присутнє в графі після спрощення боргів, то якою буде його вага (тобто сума)?

Відповідь на друге питання полягає в максимізації боргу (тобто ваги) на ребрі, щоб позбутися боргів, що проходять через інші шляхи між тією ж парою вершин. Наприклад, якщо Фред переказує 20$ Емі, то Фред не повинен буде платити нічого Девіду, а Девід буде повинен заплатити Емі лише 40$, таким чином зменшуючи загальну кількість транзакцій з 3 до 2.

Щоб відповісти на перше питання, тобто дізнатися, чи буде ребро частиною графа після спрощення боргів, ми перевіримо всі ребра по черзі. Як тільки ми виберемо ребро, ми можемо максимізувати його вагу (тобто борг) за допомогою алгоритму максимального потоку, який допомагає визначити максимальний потік між джерелом і стоком в заданому орієнтованому графі.

Отже, алгоритм для вирішення проблеми виглядає наступним чином:

1. Подати борги у вигляді орієнтованого графа (позначимо його як G) в алгоритм.
2. Вибрати одне з невідвіданих ребер, скажімо (u, v), з орієнтованого графа G.
3. Тепер, з u як джерелом і v як стоком, запустити алгоритм максимального потоку (можливо, алгоритм Дініца, оскільки це одна з оптимальних реалізацій) для визначення максимального потоку грошей від u до v.
4. Також обчислити залишковий граф (позначимо його як G'), який вказує на додатковий можливий потік боргів у вхідному графі після видалення потоку боргів між u і v.
5. Якщо максимальний потік між u і v (позначимо його як max-flow) більше нуля, то додати ребро (u, v) з вагою як max-flow до залишкового графа.
6. Повернутися до кроку 1 і подати залишковий граф G'.

Після відвідання всіх ребер залишковий граф G', отриманий на фінальній ітерації, буде тим, який має мінімальну кількість ребер (тобто транзакцій).

Застосування зазначеного алгоритму показало, що він зменшує загальну кількість транзакцій, необхідних для вирішення всіх боргів, з 9 до 6 в нашому прикладі. Алгоритм має складність $O(V^2E^2)$, де V – це кількість вершин у графі, а E – кількість ребер у графі. Тут $O(V^2E)$ – це складність реалізації алгоритму Дініца для задачі максимального потоку, а додатковий O(E) виникає через алгоритм, що

ітерується по всіх ребрах у графі. Також існують інші реалізації задачі максимального потоку зі складністю O(EV). Якщо застосувати їх, то складність зазначеного алгоритму знизиться до $O(E^2V)$, що добре масштабується для реальних умов використання процесу спрощення боргів.

1.3 Етапи розробки фінтех-додатків

Фінтех-додаток (фінансово-технологічний додаток) — це програмне забезпечення або платформа, яка використовує сучасні технології для надання фінансових послуг або виконання фінансових операцій. Ці додатки об'єднують інноваційні рішення в області фінансів і технологій для спрощення, автоматизації та покращення фінансових процесів. Основна мета фінтех-додатків полягає в тому, щоб зробити фінансові послуги більш доступними, зручними і ефективними для користувачів.

Розробка додатків фінтех передбачає використання передового досвіду розробки програмного забезпечення, методологій і сучасних технологій для створення фінансових додатків, адаптованих як для мобільних, так і для веб-платформ. Цей процес також включає інтеграцію дизайну користувацького інтерфейсу та фінансової експертизи для створення інтуїтивно зрозумілих, функціональних і орієнтованих на користувача програм, що задовольняють фінансові потреби користувачів. Розробка фінансової програми зазвичай проходить через такі етапи:

1. Визначення завдання додатку: встановлення цілей застосунку або визначення конкретних питань, які вона має вирішити.
2. Дизайн інтерфейсу користувача: проектування інтерфейсу користувача застосунку, формування його функцій і наповнення його відповідним функціоналом.
3. Розробка бекенд-інфраструктури: побудова бекенд-інфраструктури програми або формулювання та впровадження бізнес-логіки, яка відрізняє її від існуючих програм в даній сфері.

4. Тестування та перевірка: ретельна оцінка технічних аспектів програми та її тестуванням реальним користувачами для перевірки функціональності, безпеки та рівня задовільнення потреб користувачів.

Розробка додатків фінтех часто охоплює передові та інноваційні технології, такі як блокчейн, машинне навчання, штучний інтелект, хмарні обчислення тощо. Поєднання цих технологій забезпечує високий рівень конфіденційності, безпеки, продуктивності та безперебійне надання фінансових послуг і виконання транзакцій.

1.4 Дослідження переваг обраних інструментів та технологій

Проаналізуємо стек інструментів та технології, які використовуватимуться для створення мобільного додатку для менеджменту фінансових за допомогою мови програмування JavaScript і фреймворку React Native.

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. JS є однією з найбільш використовуваних мов програмування у світі. Без неї веб-сторінки обмежувалися б лише відображенням зображень і тексту. HTML створює структуру веб-сторінки, а CSS додає стиль, але саме програмування на JavaScript надає їй життя.

Без цієї дивовижної мови бізнес зазнав би значних втрат. Компанії залежні від JavaScript для взаємодії зі своїми клієнтами в сучасному онлайн-світі. Застосування JavaScript включають в себе:

- створення інтерактивних веб-сторінок: JavaScript відповідає за майже всі взаємодії на веб-сайті, які змінюють його вміст. Без нього веб був би надзвичайно обмеженим. Типові інтерактивні дії, які забезпечує JavaScript, включають: розгортання або згортання блоків контенту при натисканні миші; відтворення відео або аудіо файлів на сайті; відображення анімацій; використання випадаючих меню.
- фронтенд-розробка: існує багато популярних JavaScript-фреймворків для фронтенд-розробки, які допомагають створювати чудові додатки для користувачів. До них належать Angular, React та Vue.

Наприклад відомі додатки, створені за допомогою JavaScript, це Netflix, PayPal, YouTube і Facebook;

- бекенд-розробка: хоча JavaScript відомий як мова фронтенд-розробки, що застосовується разом з HTML і CSS, він також має вражаючі можливості для бекенд-розробки. Завдяки фреймворкам, таким як Node.js, JavaScript може використовуватися для створення серверного коду;
- ігри: JavaScript корисний не лише для стандартних веб-сайтів. Він також відповідає за більшість ігор у браузері, які підходять як для розваг, так і для початківців у розробці;
- штучний інтелект: відносно новим напрямом використання JavaScript є штучний інтелект. Бібліотеки JavaScript, такі як TensorFlow, дозволяють розробникам використовувати JavaScript для машинного навчання, створюючи моделі, які можуть передбачати майбутні події на основі класифікації минулих даних;

1.4.1. Технічні особливості React Native

React Native — це популярна платформа для розробки мобільних додатків, яка дозволяє створювати нативні додатки для iOS і Android, використовуючи JavaScript та React. Вона була створена компанією Facebook і з моменту свого випуску здобула значну популярність серед розробників завдяки своїм унікальним технічним можливостям та зручності використання. У цьому документі ми розглянемо основні технічні особливості React Native, які роблять цю платформу потужним інструментом для розробки мобільних додатків.

Однією з головних особливостей React Native є можливість створення кросплатформенних додатків. Це означає, що один і той же код можна використовувати для розробки додатків як для iOS, так і для Android. Це значно зменшує витрати часу та ресурсів на розробку, оскільки немає потреби писати окремий код для кожної платформи. Крім того, React Native дозволяє використовувати спільні компоненти для обох платформ, що забезпечує консистентність і зручність у підтримці коду.

React Native базується на JavaScript та бібліотеці React, що робить його доступним для великої кількості веб-розробників, які вже знайомі з цими технологіями. Завдяки цьому, веб-розробники можуть легко перейти до розробки мобільних додатків, не вивчаючи нові мови програмування. React Native використовує JSX (JavaScript XML), що дозволяє писати компоненти з використанням синтаксису, схожого на HTML, що робить код зрозумілим і легким для читання.

React Native використовує компонентний підхід до розробки інтерфейсу користувача. Кожен елемент інтерфейсу визначається як окремий компонент, що забезпечує модульність і повторне використання коду. Компоненти можна легко комбінувати і вкладати один в одного, що сприяє створенню складних інтерфейсів з простим і зрозумілим кодом. Це також полегшує тестування і підтримку додатків.

React Native надає можливість використовувати нативні модулі та API, що дозволяє отримати доступ до функціональності, специфічної для кожної платформи. Наприклад, можна використовувати нативні модулі для роботи з камерою, геолокацією, датчиками пристрою та іншими можливостями, які недоступні через JavaScript. Це забезпечує високу продуктивність і гнучкість у розробці додатків.

Однією з визначних технічних особливостей React Native є можливість гарячого перезавантаження (Hot Reloading). Ця функція дозволяє розробникам миттєво бачити зміни в коді без необхідності перезавантажувати додаток. Це значно прискорює процес розробки, оскільки дозволяє швидко тестувати ітерації і бачити результати в реальному часі. Гаряче перезавантаження також зберігає стан додатка, що дозволяє розробникам не втрачати прогрес під час внесення змін.

React Native забезпечує високу продуктивність мобільних додатків завдяки використанню нативних компонентів. Хоча код пишеться на JavaScript, він компілюється у нативний код, що дозволяє додаткам працювати швидко і ефективно. Крім того, React Native використовує асинхронну модель обробки

даних, що забезпечує плавність роботи інтерфейсу навіть при великій кількості обчислень.

React Native має велику і активну спільноту розробників, що постійно розширює можливості платформи через створення нових бібліотек і модулів. Це означає, що розробники мають доступ до великої кількості готових рішень для різних задач, що значно скорочує час розробки. Крім того, завдяки відкритому коду React Native, розробники можуть вносити свої покращення і доповнення, що робить платформу ще більш гнучкою і потужною.

React Native легко інтегрується з іншими популярними технологіями і інструментами, такими як Redux для управління станом додатку, Axios для роботи з HTTP-запитами, і багато інших. Це забезпечує гнучкість у виборі інструментів і технологій для реалізації конкретних задач, а також дозволяє використовувати найкращі практики і підходи у розробці мобільних додатків.

React Native надає потужні інструменти для створення анімацій, що дозволяють розробникам створювати плавні і ефектні переходи між екранами і елементами інтерфейсу. Використання бібліотек, таких як Animated API і Reanimated, дозволяє досягти високого рівня деталізації і реалістичності анімацій, що значно покращує користувацький досвід.

Отже, React Native є потужною і гнучкою платформою для розробки кросплатформенних мобільних додатків. Завдяки своїм технічним особливостям, таким як кросплатформеність, використання JavaScript і React, компонентний підхід, підтримка нативних модулів і API, гаряче перезавантаження, висока продуктивність, розширюваність і інтеграція з іншими технологіями, React Native забезпечує швидкий і ефективний процес розробки додатків, що відповідають високим стандартам якості і продуктивності.

1.4.2 Можливості React Native Navigation для навігації у додатку

React Native Navigation — це потужна бібліотека для керування навігацією у мобільних додатках, створених на платформі React Native. Вона надає зручні інструменти для реалізації різних типів навігації, дозволяючи створювати інтуїтивно зрозумілі та багатофункціональні інтерфейси користувача. У цій статті ми розглянемо основні можливості React Native Navigation та їх застосування у розробці мобільних додатків.

React Native Navigation пропонує кілька основних компонентів для організації навігації у додатку (рис. 1.5):

1. Stack Navigator. Stack Navigator дозволяє організувати екранну навігацію у вигляді стеку. Кожен новий екран додається поверх попереднього, а при поверненні назад екран видаляється зі стеку. Цей тип навігації часто використовується для реалізації переходів між різними екранами додатку, таких як перегляд деталей продукту або заповнення форми.
2. Tab Navigator. Tab Navigator забезпечує навігацію за допомогою вкладок. Користувач може перемикатися між різними екранами, натискаючи на вкладки внизу або вгорі екрану. Цей тип навігації зазвичай використовується для організації основних розділів додатку, таких як головна сторінка, пошук, профіль користувача тощо.
3. Drawer Navigator. Drawer Navigator створює бокове меню, яке можна відкрити, провівши пальцем від краю екрана або натиснувши на спеціальну іконку. Це меню може містити посилання на різні розділи додатку, забезпечуючи швидкий доступ до них. Drawer Navigator часто використовується у додатках з великою кількістю розділів або налаштувань.

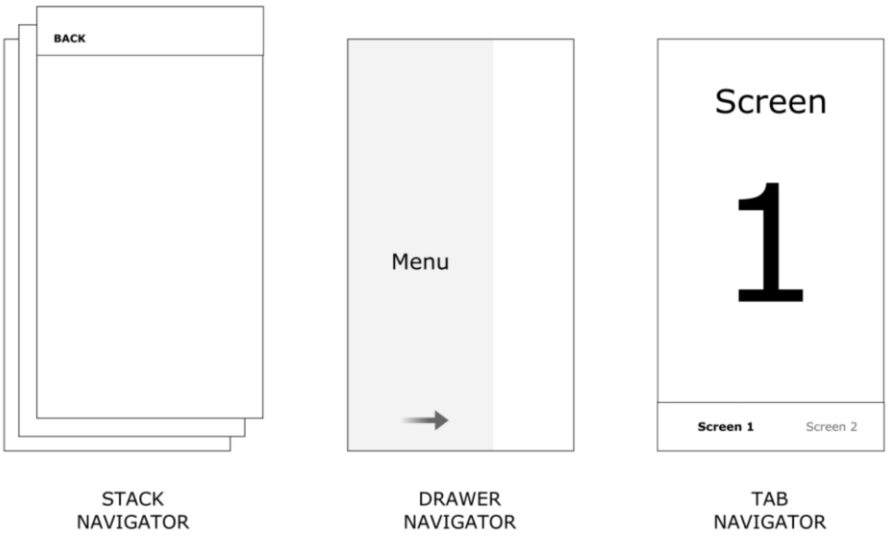


Рис. 1.5 Основні типи навігації у React Native

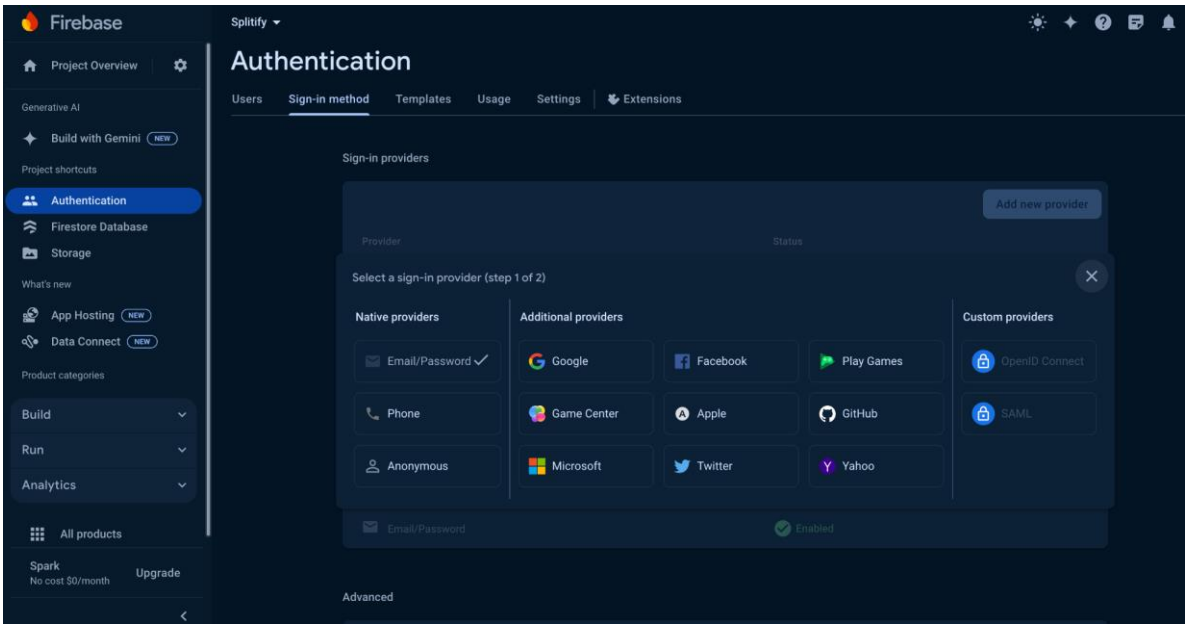
1.4.3 Firebase як інструмент для аутентифікації

Firebase є потужним інструментом для розробників, що надає безліч можливостей для створення мобільних та веб-додатків. Однією з ключових функцій Firebase є його модуль для аутентифікації користувачів. Firebase Authentication дозволяє легко та безпечно управляти процесом входу користувачів у додаток, надаючи різні методи аутентифікації та інтеграцію з популярними провайдерами. У цьому документі ми розглянемо основні можливості Firebase Authentication та їх використання.

Firebase Authentication підтримує кілька методів аутентифікації (рис. 1.6), що дозволяє розробникам вибрати найбільш підходящий для їхнього додатку. Серед підтримуваних методів:

- електронна пошта та пароль: класичний метод, де користувачі реєструються за допомогою електронної пошти та пароля;
- телефонний номер: аутентифікація за допомогою SMS-коду, що надсилається на телефон користувача;

- OAuth провайдери: Інтеграція з популярними платформами, такими як Google, Facebook, Twitter, GitHub та



Microsoft. Це дозволяє користувачам входити в додаток за допомогою своїх існуючих облікових записів;

Рис. 1.6 Налаштування методів аутентифікації у Firebase

Інтеграція Firebase Authentication у додаток є простою та швидкою. Firebase надає SDK для різних платформ, включаючи iOS, Android та веб, що дозволяє легко додати аутентифікацію до вашого додатку.

Firebase забезпечує високий рівень безпеки для зберігання та обробки даних користувачів. Всі дані передаються через захищені канали (HTTPS), а паролі зберігаються у хешованому вигляді з використанням надійних алгоритмів хешування. Крім того, Firebase надає можливості для налаштування правил безпеки доступу до даних, що дозволяє захистити інформацію від несанкціонованого доступу.

Firebase Authentication надає інструменти для управління обліковими записами користувачів. Ви можете створювати, видаляти та оновлювати облікові записи користувачів, а також відновлювати паролі та підтверджувати електронні

адреси. Це забезпечує повний контроль над користувацькими даними і дозволяє забезпечити безперебійну роботу додатку.

Firebase Authentication дозволяє користувачам прив'язувати кілька методів аутентифікації до одного облікового запису. Наприклад, користувач може зареєструватися за допомогою електронної пошти та пароля, а потім додати аутентифікацію через Google або Facebook. Це підвищує зручність для користувачів та надає їм більше можливостей для входу в додаток.

Firebase Authentication є потужним інструментом для аутентифікації користувачів у мобільних та веб-додатках. Він забезпечує простоту інтеграції, підтримку різних методів аутентифікації, високу безпеку та зручність використання. Завдяки Firebase Authentication розробники можуть швидко та легко реалізувати надійну аутентифікацію у своїх додатках.

1.4.4 Гнучка хмарна база даних Firestore

Firebase Firestore — це гнучка, масштабована хмарна база даних, розроблена компанією Google для мобільних і веб-додатків. Вона забезпечує зберігання даних у реальному часі, просту інтеграцію з іншими сервісами Firebase та безліч можливостей для керування даними. У цьому документі ми розглянемо основні особливості Firestore, її архітектуру, переваги та приклади використання.

Ця база даних забезпечує синхронізацію даних у реальному часі, що дозволяє додаткам миттєво оновлюватися при зміні даних. Це особливо корисно для додатків, які потребують постійного оновлення даних, таких як чати, соціальні мережі, багатокористувацькі ігри тощо.

Firestore використовує документо-орієнтовану модель зберігання даних, яка дозволяє зберігати дані у вигляді колекцій та документів. Кожен документ містить пари "ключ-значення" і може мати вкладені субколекції, що забезпечує гнучкість у структуруванні даних. Це дозволяє легко організовувати та масштабувати базу даних відповідно до потреб додатку.

Також Firestore підтримує складні запити, що дозволяє ефективно фільтрувати, сортувати та обробляти дані. Ви можете використовувати різні типи запитів, включаючи фільтри за полями, сортування, обмеження кількості результатів, об'єднання декількох умов тощо. Крім того, Firestore автоматично індексує дані, що забезпечує швидке виконання запитів.

Firestore забезпечує роботу в офлайн-режимі, що дозволяє додаткам продовжувати працювати навіть без підключення до інтернету. Дані зберігаються локально на пристрої користувача і синхронізуються з базою даних, коли з'єднання відновлюється. Це надає безперебійний користувацький досвід і підвищує надійність додатку.

Ця база даних розроблена для масштабованості, що дозволяє обробляти великі обсяги даних та високе навантаження. Вона автоматично масштабується відповідно до потреб додатку, забезпечуючи високу продуктивність і швидкий доступ до даних незалежно від кількості користувачів або обсягу даних.

Firestore забезпечує високий рівень безпеки даних завдяки використанню правил безпеки на основі ролей (Firestore Security Rules). Таким чином можна визначати, хто і які дані може читати або записувати, що забезпечує гнучкий контроль доступу до даних. Крім того, всі дані передаються через захищені канали (HTTPS), що забезпечує конфіденційність і цілісність даних.

А ще Firestore використовує документно-орієнтовану архітектуру, що складається з колекцій і документів. Колекції є контейнерами для документів, а документи містять дані у вигляді пар "ключ-значення". Документи можуть мати вкладені субколекції, що дозволяє створювати складні структури даних.

1.4.5 Локалізація додатку за допомогою react-i18next

Локалізація мобільного додатка – це процес адаптації мобільного додатка до мовних, культурних і технічних вимог різних цільових ринків у всьому світі. Це не лише простий переклад програми, а й адаптація вмісту, дизайну та

функціональності програми відповідно до культурних уподобань, мов і правових вимог різних регіонів.

Інвестуючи в локалізацію мобільних додатків, компанії збільшують кількість завантажень додатків, виходячи на нові ринки. Наприклад, у 2015 році Airbnb збільшив свою базу китайських мандрівників на 700%, створивши локалізовані потоки реєстрації для китайських користувачів. Новий потік пропонував варіанти реєстрації за допомогою облікових записів WeChat або Weibo, які є найпопулярнішими платформами соціальних мереж у Китаї.

Серед багатьох різних інструментів для локалізації мобільних додатків, таких як FormatJS, LinguiJS, react-native-localize, react-i18nify, і react-persian, ми обрали react-i18next. Розглянемо можливості цього інструменту:

- забезпечує легку інтеграцію з додатками, створеними на основі React та React Native;
- підтримує багатомовність, що дозволяє додавати необмежену кількість мов до вашого додатку;
- цей інструмент підтримує асинхронне завантаження мовних файлів, що дозволяє динамічно завантажувати необхідні переклади на основі потреб користувача, що підвищує продуктивність додатку та зменшує час завантаження;
- підтримує формат International Components for Unicode (ICU), який дозволяє використовувати складні мовні конструкції, такі як числові та часові формати, плюралізація та інші мовні особливості;
- надає гнучкі можливості для обробки перекладів, включаючи підтримку вкладених об'єктів, функцій для форматування та умовних перекладів;
- легко інтегрується з іншими бібліотеками та інструментами, такими як redux, що дозволяє ефективно керувати станом локалізації в додатку;

Цей інструмент був обраний, оскільки він відповідає всім нашим вимогам щодо локалізації, забезпечуючи необхідні функції для адаптації додатку до різних мов і культурних особливостей. Завдяки простій інтеграції та зручній документації, він дозволяє швидко додавати підтримку нових мов, що значно скорочує час і зусилля на впровадження локалізації. Крім того, активна спільнота та постійна підтримка, а також гнучкість та масштабованість інструменту роблять його ідеальним вибором для нашого проекту.

1.5 Дослідження аналогів додатку

Розглянемо детальніше деякі найпопулярніші програмні продукти в сфері менеджменту фінансів серед групи людей: Splitwise, Tricount, Settle Up та BillPin.

Splitwise (рис. 1.7) — це популярний додаток для управління фінансами, який спрощує процес розподілу витрат між друзями, сім'єю чи колегами. Додаток дозволяє легко відстежувати, хто кому скільки винен, автоматично обчислюючи борги та полегшуючи процес їх погашення. Splitwise особливо корисний під час подорожей, спільного проживання чи будь-яких інших ситуацій, де виникає необхідність розподілу витрат між кількома людьми.

Основні переваги:

- додаток автоматично розраховує, хто кому скільки винен, зменшуючи необхідність у ручних обчисленнях та зводячи до мінімуму ймовірність помилок;
- підтримує синхронізацію даних між різними пристроями, що дозволяє користувачам мати доступ до своїх даних з будь-якого пристрою.

Можна виділити серед недоліків:

- безкоштовна версія має обмежені можливості, такі як відсутність доступу до графіків, можливість пошуку витрат, підтримку різних валют та інші;
- вимагає активного інтернет-з'єднання для синхронізації даних між пристроями;

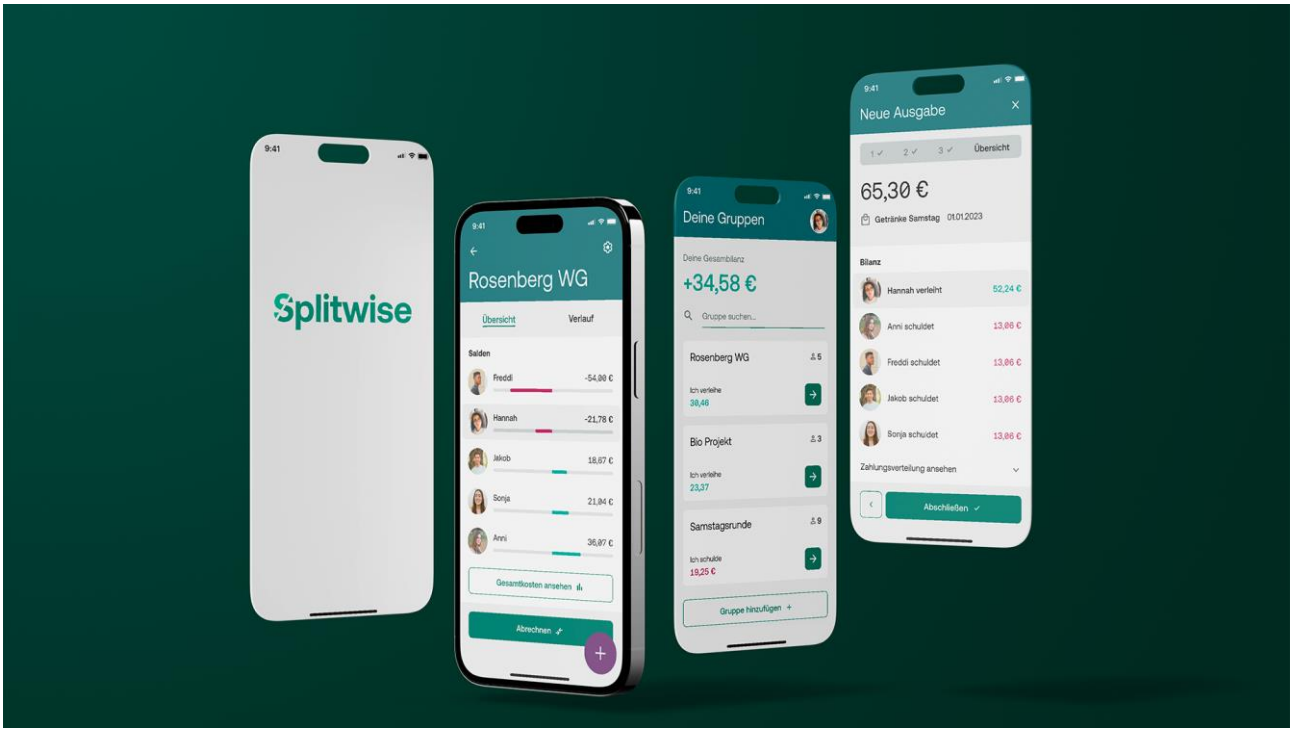


Рис. 1.7 Приклад того як виглядає інтерфейс Splitwise

Розглянемо далі інструмент для управління фінансами у групі людей **Tricount** (рис. 1.8), який є доступним для iOS та Android.

Tricount має простий та інтуїтивно зрозумілий інтерфейс, що робить його легким для початківців. Можна створити групу та поділитися посиланням з друзями, щоб вони могли додати свої витрати. Додаток показує баланс кожного учасника групи, щоб ви могли бачити, хто кому скільки винен. Також він працює навіть без підключення до інтернету, тому можна відстежувати витрати навіть у віддалених місцях.

Проте додаток має і ряд недоліків. Безкоштовна версія Tricount обмежує кількість учасників групи до 10 та кількість груп до 3. Користувачам потрібно вручну синхронізувати свої витрати з іншими учасниками групи. А також деякі функції, такі як розподіл боргів на основі відсотків та створення звітів, доступні лише в платній версії.

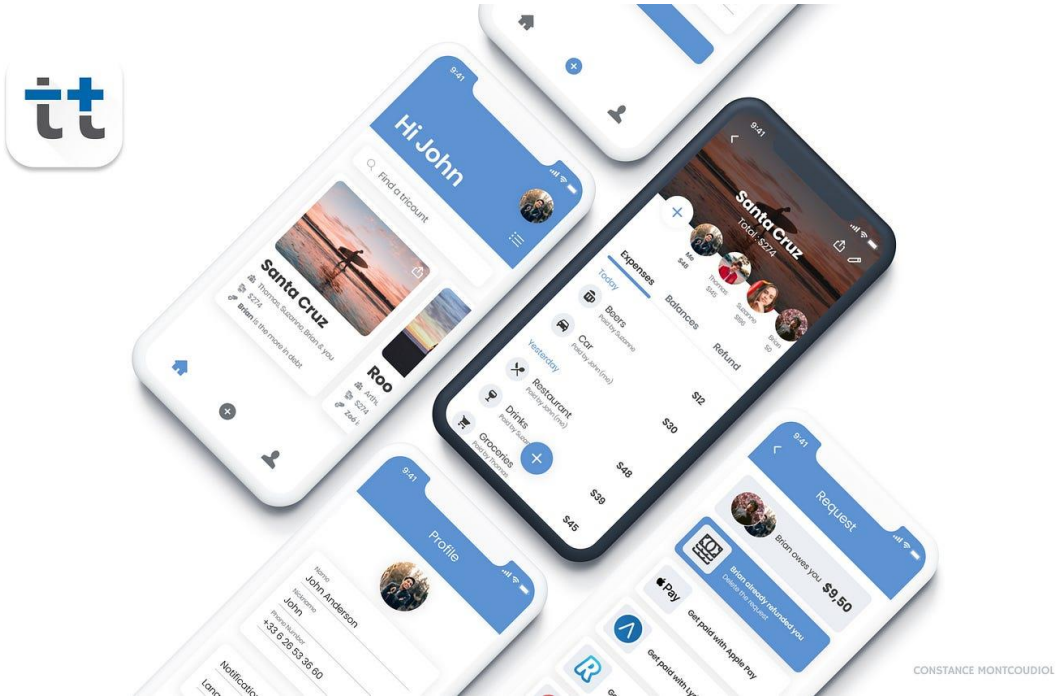


Рис. 1.8 Приклад того як виглядає інтерфейс Tricount

Останнім проаналізуємо програмний продукт **Settle Up** (рис. 1.9)**,** який є додатком для керування спільними витратами.

Основні переваги:

- додаток показує баланс кожного учасника групи, щоб ви могли бачити, хто кому скільки винен;
- доступні розширення для браузера, яке дозволяє додавати витрати з веб сайтів;
- є безкоштовним для завантаження та використання;

Рис. 1.9 Приклад того як виглядає інтерфейс Splitwise

РОЗДІЛ 2. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ МЕНЕДЖМЕНТУ ФІНАНСОВИХ ОПЕРАЦІЙ

2.1 Постановка задачі та визначення вимог до додатку

Метою даної розробки є додаток на основі фреймворку React Native та мови програмування JavaScript , що буде надавати можливість управління фінансами, який спрощуватиме процес розподілу витрат між людьми. Оглянувши та проаналізувавши різні застосунки визначимо основні вимоги до проекту:

- інтерфейс повинен бути простим і зрозумілим, з логічною навігацією та легко доступними основними функціями. Дизайн повинен бути привабливим, але не перевантаженим деталями, з чіткими та читабельними шрифтами;
- користувачі повинні мати можливість створити обліковий запис, надавши електронну пошту та пароль. Вхід до системи здійснюється за допомогою логіна (електронної пошти) та пароля;
- користувачі повинні мати можливість створювати групи для різних подій або проектів, надаючи групі назву та опис. Це дозволяє організувати фінансові витрати для конкретних заходів або спільних проектів;
- користувачі повинні мати можливість запрошувати інших до своїх груп, надаючи їм унікальний код групи. Це спрощує процес приєднання нових учасників до групи;
- користувачі повинні мати можливість додавання витрат до групи з вказанням суми, опису та дати. Вибір учасників, які брали участь у витратах, та визначення їхньої частки;
- додаток автоматично розраховує фінансові зобов’язання між учасниками групи, показуючи, хто кому і скільки винен. Це спрощує процес розподілу витрат і уникнення конфліктів;
- інтерфейс додатку повинен швидко реагувати на дії користувача, забезпечуючи плавну та безперервну взаємодію. Це підвищує задоволеність користувачів та ефективність роботи з додатком.

- дизайн додатку повинен бути адаптивним, щоб коректно відображатися на різних пристроях з різними розмірами екранів. Це забезпечує комфортне використання додатку на смартфонах і планшетах.

- додаток повинен підтримувати кілька мов, щоб бути доступним для користувачів з різних країн. Користувачі повинні мати можливість змінювати мову в налаштуваннях.

Ці вимоги до користувачів допоможуть забезпечити коректне та безпечне використання додатку для управління фінансами, а також сприятимуть створенню сприятливого середовища для співпраці між користувачами.

2.2 Загальний опис проекту

Додаток для управління фінансами у групі допомагає користувачам спрощувати процес розподілу витрат між друзями, сім'єю чи колегами. Він дозволяє створювати групи для різних подій або проектів, додавати витрати, визначати, хто скільки винен, та автоматично розраховувати фінансові зобов’язання між учасниками групи.

Після запуску додатку користувач реєструється або входить до свого облікового запису за допомогою електронної пошти та пароля. Після цього він може створити нову групу для конкретної події або проекту (наприклад, подорожі, вечірки, спільні покупки). Користувач запрошує інших до групи за допомогою коду групи.

Для введення витрат використовується інтуїтивно зрозумілий інтерфейс. Користувач вводить суму, опис та дату витрати, а також вибирає учасників, які брали участь у витраті. Додаток автоматично розраховує фінансові зобов’язання між учасниками групи та відображає, хто кому і скільки винен. Якщо витрати введені неправильно або дані некоректні, додаток повідомляє про помилку та пропонує внести правильні дані.

У додатку передбачено інтерфейс для перегляду історії витрат та розрахунків, що дозволяє користувачам легко відстежувати свої фінансові операції. Гравець має можливість вийти з додатку у будь-який момент, будучи впевненим що всі дані збережені.

2.2 Вибір моделі розробки програмного засобу

Компонентна архітектура React є ключовою для розробки цього додатку. Вона дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти, що полегшує розробку, тестування та підтримку коду. Кожен компонент відповідає за свою частину інтерфейсу, що робить код більш структурованим та зрозумілим.

Компоненти використовують стан (state) та пропси (props) для передачі даних та подій. Це забезпечує динамічне оновлення інтерфейсу відповідно до змін у даних.

Завдяки використанню компонентної архітектури React, наш додаток є легко масштабованим, зручним у підтримці та розширенні. Це дозволяє швидко впроваджувати нові функції та покращувати існуючі.

Також додаток використовує архітектуру клієнт-сервер, де мобільний додаток виступає як клієнт, взаємодіючи з сервером для обробки запитів користувачів та управління фінансовими даними. Основні компоненти моделі включають клієнтську частину, серверну частину, логіку взаємодії та базу даних.

Клієнтська частина:

- користувачі, які взаємодіють з додатком через мобільні пристрої;
- мобільний додаток, який дозволяє користувачам створювати групи, додавати витрати, переглядати розрахунки та отримувати сповіщення;

Серверна частина:

- сервер, що розміщений на комп’ютері з постійним підключенням до Інтернету, відповідає за обробку запитів від мобільного додатку;
- сервер бази даних, який зберігає інформацію про користувачів, групи, витрати та розрахунки;

Логіка взаємодії:

- користувач взаємодіє з додатком, створює групи та додає витрати;
- мобільний додаток відправляє запити на сервер для зберігання та обробки даних;
- сервер приймає запити від мобільного додатку;
- сервер обробляє запити, виконує необхідні дії для управління даними витрат та розрахунків;
- мобільний додаток отримує оброблені дані від сервера;
- додаток відображає результати розрахунків та оновлену інформацію про витрати;

База даних:

- сервер бази даних зберігає інформацію про користувачів (облікові записи, профілі), групи (назви, учасники) та витрати (суми, опис, дати);
- використовує сучасні методи захисту даних, шифрування та хешування паролів, база даних доступна лише певному колу осіб у приватному вигляді;

2.4 Обґрунтування вибору інструментальних засобів розробки

2.4.1 Вибір середовища розробки WebStorm для розробки

WebStorm – це потужне середовище розробки (IDE) для JavaScript та пов’язаних з ним технологій, створене компанією JetBrains. Вибір WebStorm для розробки додатку для управління фінансами у групі надає безліч переваг, завдяки своїм потужним інструментам та інтеграціям, що значно полегшують процес

розробки, тестування та підтримки коду. Хоч цей продукт і є платним, JetBrains надає користувачу можливість користуватися ним безкоштовно, якщо цей користувач є студентом та може підтвердити цей факт.

WebStorm підтримує всі сучасні фреймворки та бібліотеки, такі як React, Angular, Vue.js, Node.js та інші. Це робить його ідеальним вибором для розробки додатку на базі React.

WebStorm надає інтелектуальне автозавершення коду, що допомагає розробникам писати код швидше та з меншою кількістю помилок. Автозавершення враховує структуру проекту, бібліотеки та модулі, що використовуються, що забезпечує точні підказки.

Потужні інструменти для рефакторингу дозволяють легко змінювати структуру коду без ризику внесення помилок. Це включає перейменування змінних, функцій, класів, модулів та інших елементів. Автоматичне виправлення помилок та рекомендації щодо покращення коду.

WebStorm підтримує інтеграцію з Git, Mercurial, SVN та іншими системами контролю версій. Це дозволяє ефективно керувати версіями коду, відстежувати зміни та співпрацювати з іншими розробниками. Візуальний інтерфейс для управління репозиторіями, порівняння змін та вирішення конфліктів.

Доступна велика кількість доступних плагінів для розширення функціональності IDE. Це дозволяє налаштовувати середовище розробки під конкретні потреби проекту.

Отже, потужні інструменти для автозавершення коду, рефакторингу, тестування, налагодження та інтеграції з системами контролю версій роблять WebStorm ідеальним середовищем для створення якісного програмного забезпечення. Інтеграція з базами даних, підтримка плагінів, таких як Prettier, та доступність для студентів додатково спрощують процес розробки.

2.4.2 Налаштування навігації у додатку за допомогою React Native Navigation

У додатку використовується бібліотека @react-navigation для налаштування навігації на базі React Native. Навігація включає як Drawer Navigator, так і Stack Navigator для управління переходами між різними екранами додатку. Нижче описано два основних стеки навігації: MainStackNavigator та AuthStackNavigator.

Основні компоненти навігації:

1. Drawer Navigator. Відповідає за навігацію в боковому меню (drawer).

В нашому додатку використовується для відображення основного екрану додатку

```
const DrawerStack = () => (
  <Drawer.Navigator
    screenOptions={{
      headerShadowVisible: false,
      headerBackTitleVisible: false,
      headerTintColor: COLORS.white,
      headerStyle: {
        backgroundColor: COLORS.primary,
      },
      contentStyle: {backgroundColor: 'green'},
    }}
    drawerContent={props : DrawerContentComponentProps  => <CustomDrawer {...props} />}>
    <Drawer.Screen
      name="First"
      component={FirstScreen}
      options={{
        headerTitle: '',
      }}
    />
  </Drawer.Navigator>
);
```

(FirstScreen) в боковому меню (рис 2.1)

Рис 2.1 Налаштування Drawer Navigator з основними параметрами і стилями

2. Stack Navigator. Відповідає за навігацію між різними екранами додатку, дозволяючи переходити назад та вперед. Використовується для організації основного стеку навігації додатку.

```
const MainStackNavigator = () => (
  <Main.Navigator
    screenOptions={{
      headerShadowVisible: false,
      headerBackTitleVisible: false,
      headerTintColor: COLORS.white,
      headerStyle: {
        backgroundColor: COLORS.primary,
      },
    }}>
    <Main.Screen
      name="Drawer"
      component={DrawerStack}
      options={{
        headerShown: false,
      }}
    />
    <Main.Screen
      name="MyGroups"
      component={MyGroupsScreen}
      options={{
        headerTitle: 'My groups',
      }}
    />
    <Main.Screen
      name="JoinGroupScreen"
      component={JoinGroupScreen}
      options={{
        headerTitle: 'Join group',
      }}
    />
  </Main.Navigator>
);
```

Рис 2.2 Налаштування MainStackNavigator

У додатку використовується createStackNavigator для створення основного стеку навігації, який забезпечує плавні переходи між різними екранами додатку. Основний стек навігації (MainStackNavigator) включає в себе головний екран Drawer Navigator (DrawerStack), який додається як перший екран з вимкненим заголовком (headerShown: false). Крім того, в цей стек додаються екрани MyGroupsScreen та JoinGroupScreen з відповідними заголовками (headerTitle).

Для налаштування навігації для авторизації та реєстрації користувачів використовується createStackNavigator, що дозволяє створити стек навігації для авторизації (AuthStackNavigator). У цей стек додаються екрани WelcomeScreen, RegisterScreen та LoginScreen з відповідними параметрами навігації. Також налаштовуються стилі хедера (headerStyle, headerTintColor тощо) для кожного екрану, щоб забезпечити консистентний вигляд інтерфейсу.

Компонент DrawerStack містить налаштування Drawer Navigator та визначає вміст бокового меню, забезпечуючи зручний доступ до основних функцій додатку. Основний стек навігації (MainStackNavigator) включає Drawer Navigator та інші екрани додатку, забезпечуючи гнучке управління переходами. На основі умов emailVerified та isAuthorized додаток перемикається між головним стеком навігації (MainStack) та стеком авторизації (AuthStack)(рис. 2.3). Стек навігації для авторизації та реєстрації користувачів (AuthStackNavigator) забезпечує окремий шлях для користувачів, які ще не ввійшли в систему, або тих, хто

```
return (
  <NavigationContainer>
    {emailVerified && isAuthorized ? <MainStack /> : <AuthStack />}
  </NavigationContainer>
);
```

реєструється вперше.

Рис 2.3 Перемикання між MainStack та AuthStack

Таким чином, наша структура навігації забезпечує ефективне управління переходами між різними екранами у додатку, забезпечуючи зручний та інтуїтивно зрозумілий користувацький досвід.

2.4.3 Застосування Firebase для аутентифікації

В нашому додатку для управління фінансами у групі використовується Firebase для забезпечення функціоналу аутентифікації користувачів. Нижче наведено приклади коду, що показують, як відбувається реєстрація та вхід користувачів за допомогою Firebase Authentication.

Код для обробки реєстрації користувача міститься у функції onSubmit, яка виконується при натисканні кнопки реєстрації. Після введення електронної пошти, пароля та імені користувача виконується створення нового користувача (рис. 2.4) за допомогою методу createUserWithEmailAndPassword з бібліотеки Firebase Authentication.

```
const onSubmit = async ({email, password, username}) : Promise<void>  => {
  try {
    const data : UserCredential  = await auth().createUserWithEmailAndPassword(email, password);
    const user : User  = data.user;

    await user.updateProfile( updates: {
      displayName: username,
    });
    await user.sendEmailVerification();
    console.log(user);
    await firestore().collection( collectionPath: 'users').doc(user.uid).set({
      name: username,
    });

    const userProfile : {...}  = {
      login: user.displayName,
      email: user.email,
      userId: user.uid,
      emailVerified: user.emailVerified,
    };

    dispatch(updateUserProfile(userProfile));
  } catch (error) {
    if (error.code === 'auth/email-already-in-use') {
      formikRef.current?.setFieldError(
        field: 'email',
        message: 'That email address is already in use!',
      );
      return;
    }
    if (error.code === 'auth/invalid-email') {
      formikRef.current?.setFieldError(
        field: 'email',
        message: 'That email address is invalid!',
      );
      return;
    }

    formikRef.current?.setFieldError( field: 'password', error.nativeErrorMessage);
  }
```

Рис. 2.4 Фрагмент коду створення нового користувача

Код для обробки входу користувача міститься у функції onSubmit, яка виконується при натисканні кнопки входу. Після введення електронної пошти та пароля виконується вхід користувача за допомогою методу signInWithEmailAndPassword.

```
const onSubmit = async ({email, password}) : Promise<void>  => {
  await auth() Module
    .signInWithEmailAndPassword(email, password) Promise<UserCredential>
    .then(({user : User }) : void  => {
      if (!user.emailVerified) {
        formikRef.current?.setFieldError( field: 'password',  message: 'Not verified');
      } else {
        const userProfile : {...}  = {
          login: user.displayName,
          email: user.email,
          userId: user.uid,
          emailVerified: user.emailVerified,
        };

        dispatch(updateUserProfile(userProfile));
      }
    }) Promise<void>
    .catch(error => {
      console.error(error);
      formikRef.current?.setFieldError( field: 'password', error.nativeErrorMessage);
    });
};
```

Рис. 2.4 Фрагмент коду для входу в додаток зареєстрованого користувача

Використання Firebase Authentication у додатку забезпечує надійну та зручну аутентифікацію користувачів. Реєстрація, верифікація електронної пошти та вхід користувачів реалізовані з використанням методів Firebase, що дозволяє забезпечити безпеку та зручність для користувачів. Обробка помилок допомагає користувачам отримувати зворотний зв'язок у разі неправильного введення даних або інших проблем.

2.4.4 Використання Firestore для збереження даних

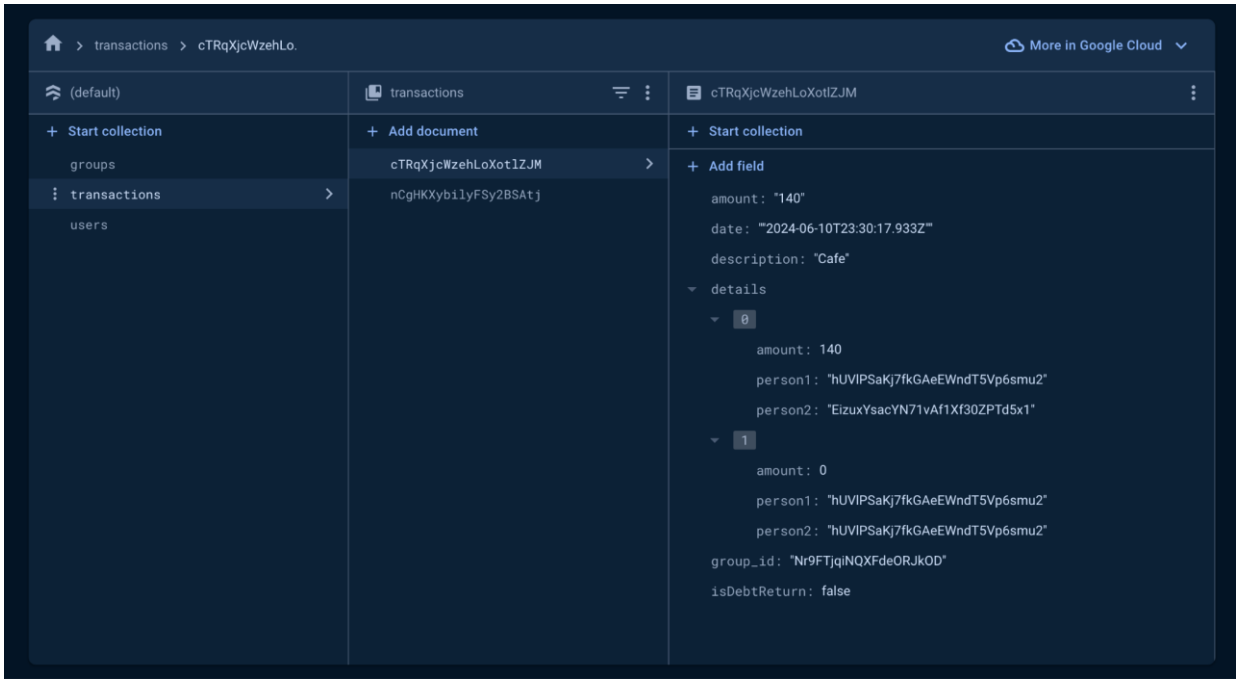
Firestore є хмарною базою даних, яка забезпечує зберігання даних у структурованому вигляді та дозволяє швидко отримувати доступ до них.

Структура даних у Firestore:

Колекція users: кожен документ представляє окремого користувача та містить його дані, такі як ім'я (name) та інші особисті дані.

Колекція groups: кожен документ представляє групу та містить дані про групу, такі як назва (name) та список користувачів (users_id).

Колекція transactions(рис. 2.5): кожен документ представляє транзакцію та містить дані про транзакцію, такі як сума (amount), дата (date), опис (description)



та деталі учасників (details).

Рис. 2.5. Приклад зберігання даних в колекції transactions

Після реєстрації нового користувача, його дані зберігаються у колекції users. Це дозволяє зберігати інформацію про кожного користувача окремо та легко отримувати доступ до цих даних у майбутньому. При створенні нової групи, дані про групу зберігаються у колекції groups. Це включає назву групи та список користувачів, які входять до цієї групи. Це дозволяє легко керувати групами та їх учасниками. Коли користувач додає нову транзакцію, дані про транзакцію зберігаються у колекції transactions. Це включає суму, дату, опис та деталі учасників. Це дозволяє легко відстежувати всі фінансові операції в межах групи.

2.4.5 Локалізація додатку за допомогою react-i18next

В додатку використовується бібліотека react-i18next для забезпечення підтримки багатомовності та локалізації. Це дозволяє додатку відображати інтерфейс різними мовами залежно від налаштувань користувача. Нижче описано, як налаштована локалізація у нашому додатку.

Для визначення мови користувача використовується languageDetectorPlugin, який зберігає та отримує інформацію про мову з локального сховища (AsyncStorage).

```
const languageDetectorPlugin :{...}  = {
  type: 'languageDetector',
  async: true,
  init: () => {},
  detect: async function (callback) :Promise<void>  {
    try {
      const language :string | null  = await AsyncStorage.getItem(STORE_LANGUAGE_KEY);

      callback(language || 'en');
    } catch (error) {
      console.log('Error reading language:', error);
    }
  },
  cacheUserLanguage: async function (language) :Promise<void>  {
    try {
      await AsyncStorage.setItem(STORE_LANGUAGE_KEY, language);
    } catch (error) {
      console.log('Error saving language:', error);
    }
  },
};
```

Рис. 2.5. Плагін визначення мови

Функція detect намагається отримати збережену мову з AsyncStorage і повертає її через callback. Якщо мова не знайдена, за замовчуванням використовується англійська ('en').

Ресурси для локалізації визначаються в об'єкті resources (рис. 2.6), який містить переклади для різних мов.

```
const resources : {en: {...}, uk: {...}}  = {
  uk: {
    translation: uk,
  },
  en: {
    translation: en,
  },
};
```

Рис. 2.6. Об'єкті resources

Налаштування i18n виконується за допомогою методу use для підключення react-i18next та languageDetectorPlugin. Метод use підключає initReactI18next для інтеграції з React та languageDetectorPlugin для визначення мови користувача. (рис 2.7)

```
i18n
  .use(initReactI18next)
  .use(languageDetectorPlugin)
  .init( options: {
    resources,
    compatibilityJSON: 'v3',
    fallbackLng: 'en',
    interpolation: {
      escapeValue: false,
    },
  });
```

Рис. 2.7. Підключення initReactI18next

Використання react-i18next у нашому додатку дозволяє забезпечити підтримку багатомовності та легке управління перекладами. Завдяки languageDetectorPlugin додаток може визначати та зберігати мову користувача, забезпечуючи персоналізований користувацький досвід. Налаштування ресурсів для різних мов та їх інтеграція з i18n забезпечують гнучкість та зручність у розробці багатомовних інтерфейсів.

1.5 Особливості програмної реалізації

При реалізації програми було використано такі наступний алгоритм мінімізації передачі боргів:

```
// Функція для підняття елемента у купі вгору
1+ usages  new *
function upheapify(heap, idx) : void  {
  if (idx === 0) return; // Якщо індекс 0, то елемент на вершині купи
  let pi : number  = Math.floor( x: (idx - 1) / 2); // Індекс батьківського елемента
  if (heap[pi].first < heap[idx].first) {
    // Якщо батьківський елемент менший за поточний
    let temp = heap[pi];
    heap[pi] = heap[idx];
    heap[idx] = temp;
    upheapify(heap, pi); // Рекурсивно піднімаємо батьківський елемент
  } else {
    return;
  }
}
```

Рис. 2.8. Функція upheapify

```
// Функція для опускання елемента у купі вниз
1+ usages  new *
function downheapify(heap, idx) : void  {
  let lc : number  = 2 * idx + 1; // Індекс лівого дитини
  let rc : number  = 2 * idx + 2; // Індекс правого дитини
  if (lc >= heap.length && rc >= heap.length) return; // Якщо немає дітей
  let largest = idx;
  if (lc < heap.length && heap[lc].first > heap[largest].first) {
    largest = lc; // Знайти найбільшого серед батьківського та дітей
  }
  if (rc < heap.length && heap[rc].first > heap[largest].first) {
    largest = rc;
  }
  if (largest === idx) return;
  let temp = heap[largest];
  heap[largest] = heap[idx];
  heap[idx] = temp;
  downheapify(heap, largest); // Рекурсивно опускаємо найбільший елемент
}
```

Рис. 2.9. Функція downheapify

```
// Функція для додавання нового елемента у купу
1+ usages  new *
function push_heap(heap, key, val) : void  {
  let ob : {first: any, second: any}  = {first: key, second: val}; // Створюємо об'єкт з ключем і значенням
  heap.push(ob); // Додаємо об'єкт у купу
  upheapify(heap, heap.length - 1); // Відновлюємо купу
}
function heap_top(heap) : undefined | any  {
  if (heap.length === 0) {
    return;
  }
  return heap[0];
}
```

Рис. 2.10. Функція push_heap

```
function pop_heap(heap) : number | any  {
  if (heap.length === 0) return 0; // Якщо купа порожня, повертаємо 0
  let i : number  = heap.length - 1;
  let temp = heap[0];
  heap[0] = heap[i];
  heap[i] = temp;
  let popped = heap.pop(); // Видаляємо найбільший елемент (останній)
  downheapify(heap, idx: 0); // Опускаємо новий верхній елемент вниз
  return popped;
}
```

Рис. 2.11. Функція pop_heap

```
function heap_top(heap) : undefined | any  {
  if (heap.length === 0) {
    return;
  }
  return heap[0];
}
```

Рис. 2.12. Функція heap_top

```
export function splitwise(transactions) : any[]  {
  let net_balance : {}  = {};
  for (let i : number  = 0; i < transactions.length; i++) {
    let e = transactions[i];

    if (e.person1 in net_balance) {
      net_balance[e.person1] += e.amount; // Додаємо суму до балансу person1
    } else {
      net_balance[e.person1] = e.amount;
    }

    if (e.person2 in net_balance) {
      net_balance[e.person2] -= e.amount; // Віднімаємо суму від балансу person2
    } else {
      net_balance[e.person2] = -e.amount;
    }
  }

  let positive : any[]  = [];
  let negative : any[]  = [];

  for (const p in net_balance) {
    if (net_balance[p] > 0) {
      push_heap(positive, net_balance[p], p); // Додаємо кредиторів у купу
    } else {
      push_heap(negative, key: -1 * net_balance[p], p); // Додаємо боржників у купу
    }
  }

  let result : any[]  = [];

  while (positive.length > 0 && negative.length > 0) {
    let p1 = heap_top(positive); // Отримуємо найбільшого кредитора
    let p2 = heap_top(negative); // Отримуємо найбільшого боржника

    pop_heap(positive); // Видаляємо найбільшого кредитора з купи
    pop_heap(negative); // Видаляємо найбільшого боржника з купи

    let settledAmount : number  = Math.min(p1.first, p2.first); // Визначаємо суму для врегулювання

    result.push({
      person1: p2.second, // Боржник
      person2: p1.second, // Кредитор
      amount: settledAmount, // Сума врегулювання
    });

    if (p1.first > p2.first) {
      push_heap(positive, key: p1.first - settledAmount, p1.second); // Оновлюємо залишок кредитора
    } else if (p1.first < p2.first) {
      push_heap(negative, key: p2.first - settledAmount, p2.second); // Оновлюємо залишок боржника
    }
  }

  return result; // Повертаємо результати врегулювання
}
```

Рис. 2.13 Функція splitwise

Пояснення роботи коду:

1. Функція upheapify (рис 2.8) піднімає елемент вгору по купі, щоб зберегти властивість купи. Якщо значення елемента більше значення батьківського елемента, вони міняються місцями, і процес повторюється рекурсивно до кореня купи або до моменту, коли батьківський елемент стане більшим.

2. Функція downheapify (рис 2.9) опускає елемент вниз по купі, щоб зберегти властивість купи. Якщо значення елемента менше значення одного з його дітей, вони міняються місцями, і процес повторюється рекурсивно до листків купи або до моменту, коли всі діти стануть меншими.
3. Функція push_heap (рис 2.10) додає новий елемент у купу та викликає upheapify, щоб зберегти властивість купи.
4. Функція pop_heap (рис 2.11) видаляє верхній елемент купи, переміщуючи останній елемент на його місце, і викликає downheapify, щоб зберегти властивість купи.
5. Функція heap_top (рис 2.12) повертає верхній елемент купи, не видаляючи його.
6. Функція splitwise (рис 2.13) врегульовує борги між людьми. Вона спочатку обчислює чистий баланс кожної особи, додаючи кредиторам та віднімаючи боржникам суми з кожної транзакції. Потім вона розділяє людей на дві купи: кредиторів та боржників, використовуючи функції push_heap, pop_heap, та heap_top, щоб мінімізувати кількість транзакцій, необхідних для вирівнювання всіх боргів. У кінці вона повертає результати

```
useEffect( effect: () => {
  const subscriber = firestore() Module
    .collection( collectionPath: 'groups') CollectionReference<DocumentData>
    .where( fieldPath: 'users_id', opStr: 'array-contains', userId) Query<DocumentData>
    .onSnapshot( onNext: snapshot : QuerySnapshot<DocumentData> => {
      const updatedGroups : ({...})[] = snapshot.docs.map(doc : QueryDocumentSnapshot<DocumentData> => ({
        id: doc.id,
        name: doc.data()?.name,
      }));
      setGroups(updatedGroups);
    });

  return () : void => subscriber();
}, deps: []);
```

врегулювання.

Рис. 2.14 Приклад використання функції onSnapshot

Функція onSnapshot з Firestore дозволяє отримувати дані у реальному часі з бази даних Firestore, автоматично підписуючись на зміни у колекції або

документі. Це означає, що кожного разу, коли дані у підписаній колекції або документі змінюються, ви автоматично отримуєте оновлення без необхідності вручну оновлювати або повторно запитувати дані. У прикладі (рис. 2.14) код у useEffect підписується на зміни у колекції groups, де масив users_id містить userId, і виконується при завантаженні компонента. Підписка дозволяє автоматично отримувати оновлення кожного разу, коли змінюється список груп, до яких належить користувач.

Це забезпечує актуальні дані без затримок, що особливо корисно для додатків, де важливо мати завжди свіжу інформацію, наприклад, для чатів, спільних робочих просторів або трекерів завдань. Функція onSnapshot спрощує роботу з даними у Firestore, автоматизуючи процес отримання оновлень і оновлення інтерфейсу. Не потрібно вручну стежити за змінами даних або перезавантажувати дані з сервера. При кожному оновленні даних можна автоматично оновлювати стан додатку, що забезпечує відображення актуальної інформації для користувачів.

Завдяки цьому підходу, кожного разу, коли змінюються дані у Firestore, наприклад, додається нова група або змінюється існуюча, ви одразу отримуєте актуалізовані дані. Це дозволяє обробляти дані відразу при отриманні оновлень, виконуючи будь-які необхідні операції з даними перед оновленням стану або відображенням їх у користувацькому інтерфейсі. Таким чином, використання onSnapshot ефективно масштабується для великої кількості користувачів і оновлень, забезпечуючи високу продуктивність і низьку затримку при роботі з великими обсягами даних. Це робить додаток інтерактивним і динамічним, забезпечуючи доступ до актуальної інформації у реальному часі.

2.6 Організація тестування та налагодження програмного засобу

Наш додаток призначений для управління фінансами у групах, надаючи користувачам можливість для створення та управління групами, додавання витрат

та розрахунків боргів між учасниками. Основною метою є спрощення процесу розподілу витрат між людьми.

Фронтенд частина додатку була побудована використовуючи React Native, що дозволило створити кросплатформений додаток для iOS та Android. Для стилів ми використовували власні CSS стилі, що дозволило налаштувати вигляд інтерфейсу відповідно до наших потреб.

Застосунок було протестовано на коректність виконання основного функціоналу, зокрема:

- реєстрація, можливість залогінитися;
- створення груп та приєднання до них;
- створення завдань та відображення їх статусу на сторінці;
- додавання витрат;
- розрахунок боргів.

Робота застосунку відображена на рис 2.15 – 2.

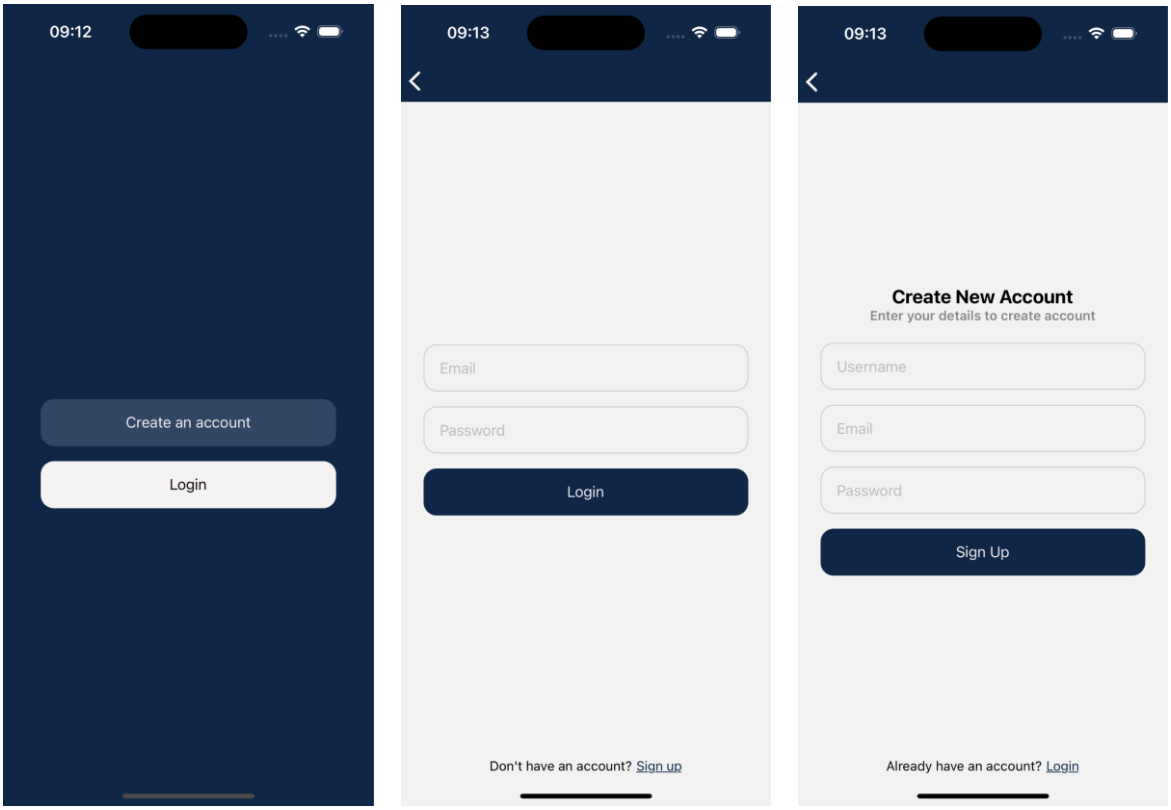


Рис. 2.15 Вигляд сторінок для авторизації

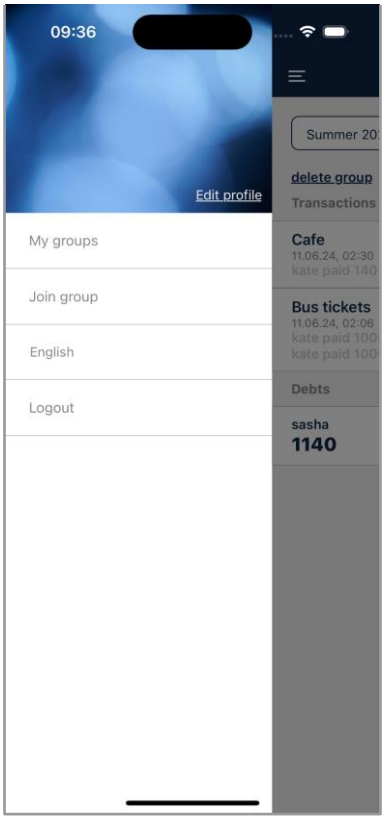


Рис. 2.16 Вигляд бокового меню додатку.

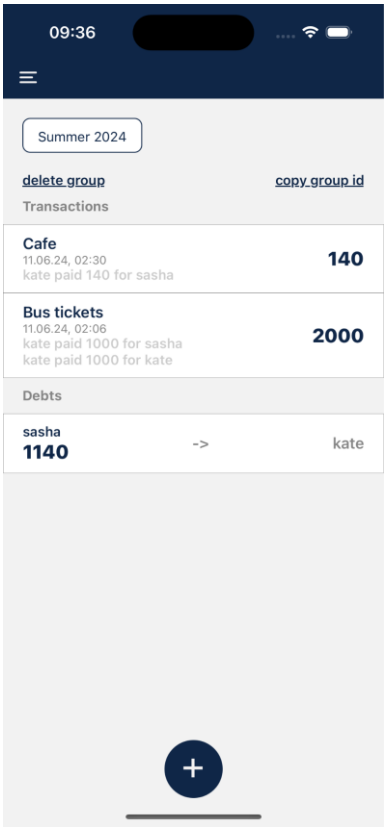


Рис. 2.17 Основна сторінка додатку.

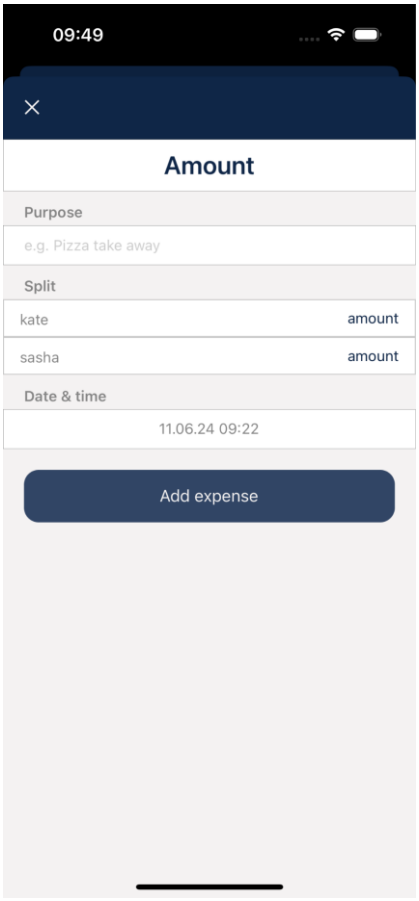


Рис. 2.18 Вигляд модального вікна для створення нової витрати

2.7 Рекомендації по використанню та впровадженню програмного засобу

Програмний засіб надає можливість розподіляти витрати та керувати фінансовими розрахунками у межах конкретної групи. Для коректної роботи функціоналу додатку потрібно:

1. Забезпечити стабільне підключення до Інтернету.
2. Створити акаунт та увійти до нього.
3. Приєднатися до вже існуючої групи або створити власний.
4. Додавати витрати та відслідковувати процес їх розподілу.

Для запуску програми на новому пристрої потрібно:

- Node.js;

- Xcode якщо запуск виконуватиметься на macOS;
- Android Studio якщо запуск виконуватиметься на Windows;
- Cocoapods;
- Запуск можна виконати на Windows чи macOS.

ВИСНОВКИ

У ході роботи було розроблено програмний продукт, який надає засоби управління фінансами у групах, розподілу витрат та ведення фінансових розрахунків. Застосунок дозволяє створювати облікові записи, відстежувати витрати, редагувати та видаляти інформацію, а також спілкуватися в рамках групи через чат. Програмний продукт значно спрощує процес управління фінансами, надаючи користувачам інтуїтивно зрозумілий інтерфейс і ефективні інструменти для спільної роботи.

У процесі розробки ми також вивчили нові функції та особливості фреймворку React Native. Було оглянуто можливості бази даних Firebase для авторизації та збереження даних, що дозволило створити надійну та безпечну систему управління обліковими записами та даними користувачів. Крім того, було написано алгоритм мінімізації передачі боргів, який дозволяє автоматично розраховувати та зводити до мінімуму кількість транзакцій між учасниками групи. Для забезпечення багатомовності у додатку було використано i18next, що дозволило легко додати підтримку кількох мов та забезпечити зручне використання користувачами з різних країн.

Під час тестування були виявлені певні недоліки технології React Native. Незважаючи на те, що вона забезпечує досить високу швидкість роботи і кросплатформність, деякі компоненти можуть працювати не так плавно на різних платформах. Крім того, інтеграція з деякими нативними модулями може бути складною і вимагати додаткових налаштувань. Однак, незважаючи на ці недоліки, технологія React Native має великий потенціал для подальшого розвитку продукту, оскільки дозволяє швидко та ефективно розробляти мобільні застосунки з високою продуктивністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тема 10. Фінансовий менеджмент як підсистема управління організацією. *Навчально-інформаційний портал ВП НУБіП України "Ніжинський агротехнічний інститут"*. URL: http://moodle.nati.org.ua/mod/book/tool/print/index.php?id=5276 (date of access: 10.06.2024).
2. Сутність фінансового менеджменту - Бібліотека BukLib.net. *Головна - Бібліотека BukLib.net*. URL: https://buklib.net/books/22394/ (date of access: 10.06.2024).
3. Розробка фінтех-додатків: тенденції, особливості та перспективи. *Custom Software Development Company | Stfalcon.com*. URL: https://stfalcon.com/uk/blog/post/fintech-app-development-trends-features-perspective (date of access: 10.06.2024).
4. URL: https://codeinstitute.net/global/blog/what-is-javascript-and-why-should-i-learn-it/ (date of access: 10.06.2024).
5. What is JavaScript? - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (date of access: 10.06.2024).
6. What is React Native? | Application Development | CIAT. *California Institute of Applied Technology*. URL: https://www.ciat.edu/blog/what-is-react-native/ (date of access: 10.06.2024).
7. Hello React Navigation | React Navigation. *React Navigation | React Navigation*. URL: https://reactnavigation.org/docs/hello-react-navigation(date of access: 10.06.2024).
8. React Native Firebase | React Native Firebase. *React Native Firebase | React Native Firebase*. URL: https://rnfirebase.io/ (date of access: 10.06.2024).
9. Split expenses with friends. *Splitwise*. URL: https://www.splitwise.com/ (date of access: 10.06.2024).
10. Tricount - Organize group expenses. *Tricount.com*. URL: https://www.tricount.com/en/ (date of access: 10.06.2024).
11. 25 things to remember when localizing mobile apps | Lokalise. Lokalise Blog. URL: https://lokalise.com/blog/best-practices-to-remember-when-localizing-mobile-apps/
12. Jakub Pomyka. List of i18n libraries for React and React Native. SimpleLocalize. URL: https://simplelocalize.io/blog/posts/the-most-popular-react-localization-libraries/

13. Marcu F. How to Build a React Native App and Integrate It with Firebase. *freeCodeCamp.org*. URL: https://www.freecodecamp.org/news/react-native-firebase-tutorial/ (date of access: 10.06.2024).
14. React Native Firebase. *Firebase Open Source*. URL: https://firebaseopensource.com/projects/invertase/react-native-firebase/ (date of access: 10.06.2024).
15. How to Authenticate with Google using Firebase in React Native Application ? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/how-to-authenticate-with-google-using-firebase-in-react-native-application/ (date of access: 10.06.2024).
16. Introduction | react-i18next documentation. *Introduction | react-i18next documentation*. URL: https://react.i18next.com/ (date of access: 10.06.2024).
17. Lupachova K. (. React Native Internationalization with i18next. *DEV Community*. URL: https://dev.to/ramonak/react-native-internationalization-with-i18next-568n (date of access: 10.06.2024).
18. Low-code backend to build modern apps. *Back4App Blog*. URL: https://blog.back4app.com/ (date of access: 10.06.2024).
19. Authentication | React Native Firebase. *React Native Firebase | React Native Firebase*. URL: https://rnfirebase.io/auth/usage (date of access: 10.06.2024).
20. Social Authentication | React Native Firebase. *React Native Firebase | React Native Firebase*. URL: https://rnfirebase.io/auth/social-auth (date of access: 10.06.2024).
21. Cloud Firestore | React Native Firebase. *React Native Firebase | React Native Firebase*. URL: https://rnfirebase.io/firestore/usage (date of access: 10.06.2024).
22. Cloud Storage | React Native Firebase. *React Native Firebase | React Native Firebase*. URL: https://rnfirebase.io/storage/usage (date of access: 10.06.2024).
23. Cloud Firestore Library and framework integrations | Firebase. *Firebase*. URL: https://firebase.google.com/docs/firestore/library-integrations (date of access: 10.06.2024).
24. Firebase Realtime Database. *Firebase*. URL: https://firebase.google.com/docs/database (date of access: 10.06.2024).
25. Get realtime updates with Cloud Firestore | Firebase. *Firebase*. URL: https://firebase.google.com/docs/firestore/query-data/listen (date of access: 10.06.2024).

ДОДАТКИ

Додаток А

Технічне завдання

1 Вступ

Технічне завдання описує мобільний додаток для фінансового менеджменту коштів. Додаток дозволяє зручно розподіляти фінансові зобов'язання між учасниками групи, відстежувати стан витрат та боргів.

2 Підстави для розробки

Розробка проводиться на основі індивідуального завдання, поставленого керівником бакалаврської роботи для спеціальності 122 “Комп’ютерні науки”

3 Призначення розробки

Функціональне призначення: програмний продукт призначений щоб надати інструменти для моніторингу коштів та боргів у групах в максимально зрозумілій та зручній формі.

Експлуатаційне призначення: призначений для використання для використання в особистих і робочих групах, де виникає необхідність у спільному управлінні фінансами.

4 Вимоги до програми чи програмного продукту

4.1 Вимоги до функціональних характеристик

1. Система повинна надавати реєстрацію та автентифікацію користувачів, дозволяти новим користувачам створювати облікові записи і забезпечувати зареєстрованим користувачам вхід в систему за допомогою своїх облікових даних.

2. Користувач повинен мати можливість створювати нові групи для управління фінансами, змінювати існуючі групи та видаляти їх за необхідності.

3. Користувач повинен мати можливість додавати нові фінансові витрати до групи, змінювати існуючі витрати та видаляти їх за необхідності. При додаванні витрат користувач повинен мати можливість вказати суму витрат, опис, дату витрат та вибрати учасників, які брали участь у витратах.

4. Система повинна автоматично розраховувати борги між учасниками групи на основі внесених витрат. При додаванні нових витрат або зміні існуючих, система повинна автоматично оновлювати фінансові зобов’язання кожного учасника групи, показуючи, хто кому і скільки винен.

5. Система повинна надавати зручний спосіб введення ідентифікатора групи, після чого користувач буде доданий до групи і зможе брати участь у її фінансових операціях.

6. Система повинна підтримувати кілька мов та надавати користувачам можливість змінювати мову інтерфейсу у налаштуваннях.

4.2 Вимоги до надійності

1. Додаток повинен бути спроектований та налаштований для ефективної роботи під високим навантаженням, здатний обробляти одночасні запити багатьох користувачів без значного зниження продуктивності.

2. Додаток має відповідати на запити користувачів швидко, уникаючи системних збоїв та перебоїв у роботі.

3. Додаток повинен бути стабільним та надійним, забезпечуючи безперебійний доступ до функціоналу для користувачів навіть під великим навантаженням.

4. Перед випуском додаток повинен пройти комплексне тестування для виявлення та виправлення помилок і вразливостей, забезпечуючи високу якість продукту.

5. Додаток повинен бути постійно моніторений для виявлення проблем та своєчасного реагування на них, забезпечуючи стабільну роботу та високу продуктивність.

6. Додаток повинен мати ефективні механізми захисту, такі як надійна аутентифікація та авторизація, для забезпечення безпеки облікових даних користувачів та їх фінансової інформації.

4.3 Умови експлуатації

1. Додаток має бути доступним для користувачів 24/7, за винятком проведення планових технічних робіт.
2. Повинні бути надані інструкції з використання та експлуатації системи.

4.4 Вимоги до складу і параметрів технічних засобів

Для запуску клієнтської додатку Splitify підійде будь-який сучасний мобільний пристрій з операційною системою iOS (версія не нижче 11.0). Пристрій повинен мати доступ до інтернету для забезпечення зв'язку з серверною частиною додатку.

4.5 Вимоги до інформаційної і програмної сумісності

Продукт повинен працювати в режимі онлайн і бути доступним для користувача на пристрої із стабільним підключенням до інтернету.

4.6 Вимоги до транспортування і збереження

Всі дані сайту повинні зберігатися у надійному сервері з відповідними стандартами безпеки.

5 Вимоги до програмної документації

Документація повинна містити:

- опис всіх функцій;
- інструкцію для користувачів;

- опис алгоритмів роботи, а також опис архітектури та дизайну сервісу.

6 Техніко-економічні показники

Техніко-економічні показники мають включати оцінку витрат на розробку, впровадження та на підтримку додатку. Також надавати оцінку економічної ефективності використання додатку та очікуваних результатів і користі для організації.

7 Стадії і етапи розробки

7.1 Передбачені стадії розробки

1. Визначення всіх головних та аналіз можливих функціональних та нефункціональних вимог до системи.
2. Проектування та розробка архітектурної частини проекту та дизайну інтерфейсу для користувача.
3. Розробка та впровадження функціональну:
   - розробка і налаштування бази даних для зберігання і отримання інформації про наявних користувачів, їх групи та транзакції;
   - інтеграція з системою авторизації;
   - створення і опис необхідних екранів компонентів додатку;
4. Тестування, відлагодження та оптимізація додатку.
5. Подальша підтримка для коректної експлуатації системи.
5. Проведення регулярних заходів з оновлення та вдосконалення наявного функціонал.
7. Проведення моніторингу продуктивності додатку, здійснення процесів оптимізації для забезпечення його найкращої роботи.

7.2 Необхідні сторінки та структурні елементи на сайт

Назва елементу	Опис
Welcome Screen	Перший екран додатку де буде міститися: - кнопка для реєстрації; - кнопка для авторизації якщо користувач вже зареєстрований.
Home Screen	Домашній екран міститиме: - список груп до складу яких належить користувач; − всі транзакції обраної групи; - всі борги які є в обраній групі; - кнопка, для початку роботи програми; - кнопка для видалення групи; - кнопка для копіювання коду групи; - кнопка для додавання нової транзакції;
AllGroups	Екран, з списком груп до складу яких належить користувач та кнопкою щоб відкрити модельне вікно щоб додати нову групу, вказавши її назву.
New Transaction Modal	Модальне вікно яке дає можливість додати нову транзакцію, містить: - поле для вказання назви; - поле для вказання суми транзакції; - список усіх учасників для можливості надання інформації як саме ділиться сума цієї транзакції між усіма учасниками; - поле з вибором дати та часу транзакції;

Register Screen	Сторінка реєстрації, серед полів для реєстрації повинні бути username, email, password, також кнопка підтвердження реєстрації. Всі поля повинні мати валідацію вхідних значень з перевіркою як на сервері так і на клієнті.
Login Screen	Опис сторінки входу в особистий акаунт сайту. Щоб увійти, користувач має коректно ввести інформацію email та password і натиснути кнопку підтвердження.
Add Group Modal	Модальне вікно яке дає можливість додати нову групу, містить текстове поле для вказання назви, та кнопку підтвердження дії.
Account Edit	Модальне вікно, де користувач може змінити своє ім’я.

8 Порядок контролю і приймання

Після закінчення кожного етапу роботи необхідно проводити тестувати додаток наступним чином:

- перевірка коректного встановлення та оновлення інформації у таблицях Firestore;
- забезпечення захисту даних користувачів, перевірка коректності аутентифікації та авторизації через Firebase Authentication;
- перевірка працездатності всіх наявних функцій додатку згідно з зазначеними вимогами;
- виконати функціональне тестування для перевірки коректності функціоналу додатку згідно з зазначеними вимогами;
- зробити перевірку роботи додатку на різних моделях пристроїв;
- здійснювати моніторинг у роботі системи та складати відповідні звіти щодо її продуктивності та стабільності та доступності для роботи.

Додаток Б
Інструкція користувача

Для коректної роботи застосунку та початку користування інструментами, що забезпечують управління фінансами у групах, необхідно виконати вхід у акаунт або створити новий обліковий запис, якщо у вас його ще немає.

Перш за все, відкрийте додаток і на екрані входу натисніть кнопку "Реєстрація". Введіть вашу електронну адресу, пароль та інші необхідні дані, після чого підтвердіть реєстрацію. Ви автоматично увійдете у систему. Якщо у вас вже є обліковий запис, просто введіть вашу електронну адресу та пароль на екрані входу та натисніть кнопку "Вхід".

Після входу в систему вам буде надано доступ до всіх можливостей платформи. На головній панелі ви зможете створити нову групу, натиснувши на кнопку "Додати групу". Для приєднання до існуючої групи введіть ідентифікатор групи, отриманий від іншого учасника.

Перейдіть на сторінку групи, до якої ви приєдналися, щоб почати додавати витрати. Натисніть кнопку "Додати витрату" і введіть суму, опис, дату витрат та виберіть учасників, які брали участь у витратах. Усі додані витрати будуть відображатися у списку витрат групи, де ви можете редагувати або видаляти їх за необхідності.

Додаток автоматично розраховує борги між учасниками групи на основі внесених витрат. Ви можете переглянути, хто кому і скільки винен, на сторінці розрахунків. Це допоможе уникнути плутанини та забезпечити прозорість фінансових операцій у групі.

У налаштуваннях групи ви можете редагувати її назву, додавати або видаляти учасників. Якщо бажаєте, учасники можуть покинути групу, натиснувши кнопку "Вийти з групи". Після завершення роботи над проектом або

врегулювання всіх фінансових питань ви можете видалити групу, що призведе до видалення всієї інформації про неї.

АНОТАЦІЯ

Плоднік К. Ю. – Розробка мобільного додатку для менеджменту фінансових операцій – Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2024р.

Робота присвячена розробці програмного продукту який надає засоби управління фінансами у групах, розподілу витрат та ведення фінансових розрахунків. Розглянуто особливості фреймворку React Native, досліджено можливості бази даних Firebase та react-i18next для локалізації додатку.

Метою кваліфікаційної роботи є створення додатку на основі фреймворку React Native мови програмування JavaScript. Створений програмний продукт уможливлює колективний менеджмент фінансових операцій та дозволяє автоматично розраховувати та зводити до мінімуму кількість транзакцій між учасниками групи.

Для обслуговування операцій взаєморозрахунків було розроблено спеціалізований алгоритм з елементами оптимізації.

Робота включає розробку, тестування та рекомендації щодо впровадження програми.

Ключові слова: мобільний додаток, управління фінансами, розподіл витрат, фінансові розрахунки, локалізація, розрахунок боргів, алгоритм мінімізації боргів, Firebase, React Native, JavaScript, react-i18next.