КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

СТВОРЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

Виконав студент 4-го курсу
Ростислав МАХАНЬКО

(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Євген ІВАНОВ

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент (підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«___»_____2021р.,
протокол №
Завідувач кафедри
Олександр ПРОВОТАР (підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 40 сторінок, 15 джерел посилань, 2 ілюстрації. Ключові слова: МОБІЛЬНА РОЗРОБКА, IOS, ANDROID, API, REST API, WEBVIEW, REACT NATIVE, NODE.JS, GOOGLE PLAY, APP STORE, OPEN BANKING.

Об’єктом роботи є створення мобільного застосунку для управління персональними фінансами.

Метою роботи є огляд існуючих засобів мобільної розробки під операційні системи Android та iOS, аналіз фінансових застосунків та можливостей оптимізації управління рахунками, створення прототипу застосунку для ведення особистих фінансів для українського ринку.

Результат роботи: розглянуто можливості розробки мобільних додатків використовуючи фреймворк React Native. Проаналізовано можливості застосунків для управління персональними фінансами: YNAB, Mint, PocketGuard. Розглянуто можливості інтеграції з банківськими, страховими та державними сервісами. Обрано інфраструктуру для розробки серверної та клієнтської частини застосунку призначеного для користувачів України. Реалізовано прототип системи, а саме: користувацький інтерфейс з екранами реєстрації та входу, головного екрану, де можна переглянути загальний баланс рахунків та список нещодавніх транзакцій, екран зі списком рахунків, екран для додавання та керування рахунками, реалізовано відповідну серверну частину.

ЗМІСТ

ВСТУП

Останнє десятиріччя ми спостерігаємо стрімкий розвиток мобільних телефонів: їхньої потужності, можливостей, функцій, які вони виконують, кількість таких, що перебувають у використанні. Зараз чи не у кожної людини, від дитини, що ходить в дитсадок, до бабусь та дідусів можна побачити мобільний телефон. Цей пристрій все частіше стає першим, який з’являється у людини, тобто персональний комп’ютер чи ноутбук вже не є базовою необхідністю, адже функціонал потрібний у повсякденному житті та на роботі вже є і на мобільних пристроях, які завжди під рукою та мають менші габарити. Основними досягненнями, які переосмислили використання мобільних пристроїв, стали:

- розробка сенсорних екранів, які зробили екран телефона зручним для ока, таким що може використовуватися для перегляду відео, а не лише аудіодзвінків та надсилання СМС, що спричинило введення у широкий обіг смартфонів (з англ. smart — розумний та phone — телефон) — мобільних телефонів, які поєднують функції надання стільникового зв'язку з виконанням широкого спектру додаткових функцій і можливостей, забезпечуваних відкритими операційними системами й додатками до них;
- розвитку Інтернету, а саме появи мережі 3G та розростання її покриття, разом з чим з’явилася можливість для будь-кого підключитися до швидкого Інтернету зі значно більшої кількості місць, де наприклад немає Wi-Fi;
- поява популярних комплексних мобільних операційних систем Android та iOS, які надали можливість розробникам створювати програми та публікувати їх в маркетах Google Play та AppStore відповідно, щоб потім їх могли завантажити користувачі з усього світу.

Тож зараз на телефоні будь-якої людини вже більше 10-20 застосунків для різних цілей – починаючи від калькулятор та нотатника, закінчуючи 3D картами та сканерами шкірних захворювань.

Серед десятків застосунків чільне місце займають і ті, що допомагають здійснювати чи відслідковувати фінансові операції. iOS має гаманець для карток Apple Pay, Android – Google Pay, вони дозволяються проводити безконтактні платежі навіть без потреби носити з собою банківську картку. Сайти у браузері та застосунки, які продають певні послуги та товари, використовують форми для оплати краткою та іншими платіжними методами. Більшість банків побачили попит на можливість відкривати рахунки та керувати ними онлайн, і створили свої застосунки для мобільних пристроїв.

Людині стає простіше витрачати кошти, а розмаїття фінансових сервісів якими вона користується ускладнює можливість відслідковувати стан рахунків, витрат та грошових збережень. Тому виникає й інша потреба - розумно користуватися усіма доступними інструментами для досягнення фінансових цілей, накопичення та інвестування коштів.

1 ОПЕРАЦІЙНІ СИСТЕМИ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Android

Android - це операційна система з відкритим кодом, заснована на модифікованому ядрі Linux, належить компанії Google. За її розробку відповідає Google разом із Open Handset Alliance (OHA). Через три роки після придбання Android Google був проданий перший смартфон під управлінням ОС Android. Близько 80% мобільного ринку використовує Android, а більше мільярда застосунків завантажують її користувачі щомісяця.

Шар додатків - це верхній шар в архітектурі ОС Android, він є шаром, де розташовані програми встановлені на пристрої. Програми можуть бути попередньо встановлені та забезпечувати основні функціональні можливості телефону, такі як телефонні дзвінки чи перегляд веб-сторінок в Інтернеті, але також цей рівень обробляє ті програми, які завантажені або знаходяться в стадії розробки. Ці програми написані переважно на мовах Java або Kotlin, і згодом компілюються в машинний код для встановлення на смартфон.

Створення нового проєкту для Android в інтегрованому середовищі розробки (IDE), такому як Android Studio, Intelij чи Eclipse, забезпечує базовий додаток, де Android SDK допомагає структурувати роботу і є обов'язковим інструментом для розробників. Android SDK використовує перевагу мови програмування Java під назвою Java Android Library, яка містить усі пакети, структуру додатків та бібліотеки класів, необхідні розробнику для створення програми. Синтаксис такий самий, як у оригінальної Java, але є деякі специфічні класи та пакети Android, такі як Activity і View Class. Коли створено проєкт Android, розробник може скомпілювати код Java, використовуючи Android SDK та середовище розробки. Це призведе до створення програми, що складається з пакета Android (APK), стиснутої колекції коду, що складає програму, щоб застосунок можна було встановити та запустити на реальному або віртуальному пристрої. Кожна версія Android має власний віртуальний пристрій Android (AVD), який можна налаштувати та запустити в IDE. AVD -

це емулятор, який містить конкретну ОС смартфона і зручний, якщо у розробника немає пристрою на Android. У розробці та реалізації користувальницького інтерфейсу використовуються подібні компоненти та концепції інтерфейсу, як власний інтерфейс Java, однак є деякі відмінності. Усі компоненти користувальницького інтерфейсу в Android побудовані на представленнях Views, а взаємодія користувача із елементами інтерфейсу складається з Activity. Імплементація взаємодії із застосунком пишеться переважно мовою Java, а стилістика виконується XML-подібним чином, як при розробці веб-сторінки за допомогою HTML та CSS [1].

1.2 iOS

iOS (раніше iPhone OS) - це мобільна операційна система, створена та розроблена Apple Inc. виключно для свого обладнання. Саме операційна система керує багатьма мобільними пристроями компанії, включаючи iPhone та iPod Touch; цей термін також включав версії, що працюють на iPad, доки назва iPadOS не була введена з версією 13 у 2019 році. Це друга за частотою встановлена мобільна операційна система у світі після Android. Вона є основою для трьох інших операційних систем Apple: iPadOS, tvOS і watchOS. Це запатентоване програмне забезпечення, хоча деякі його частини є відкритими за ліцензією Apple Public Source License та іншими ліцензіями.

Розробка застосунків для iOS - це процес створення прорам для обладнання Apple, включаючи iPhone, iPad та iPod Touch. Програмне забезпечення написано мовою програмування Swift або Objective-C потім розгортається в App Store для завантаження користувачами.

Кожному розробнику потрібен комп'ютер Mac - і Mac, як правило, дорожчий за аналоги на базі Windows. Крім того, після закінчення розробки застосунку, він має пройти через жорсткий процес перевірки якості App Store.

Вимоги для розробки програм під iOS:

- Комп’ютер, на якому встановлена підтримувана версія macOS.

- Xcode, який є інтегрованим середовищем розробки (IDE) для macOS, можна безкоштовно завантажити з магазину Mac App Store.
- Активний обліковий запис розробника Apple, який вимагає щорічної плати в розмірі 99 доларів США.
- Ці три вимоги працюють разом: Лише активні учасники Програми розробника Apple можуть розміщувати програми в Apple App Store. Тільки програми, підписані та опубліковані Xcode, можуть надсилатись до App Store. Xcode працює лише на macOS, а macOS працює лише на комп'ютерах Apple.

Однією з головних переваг розробки додатків для iOS є широка колекція доступних для розробника ресурсів. Завдяки стандартизації, функціональності та послідовності розробки додатків для iOS, Apple може випускати власні API та бібліотеки як набори, які є стабільними, багатофункціональними та простими у використанні. Розробник можете використовувати ці SDK для iOS, щоб легко інтегрувати свій додаток до існуючої інфраструктури Apple.

Наприклад, якщо ви працюєте над контролером програми для розумної печі тостера, то можете використовувати HomeKit для стандартизації зв'язку між тостером і телефоном. Користувачі зможуть координувати спілкування між розумною тостером та розумною кавоваркою. Існують набори для розробки ігор (такі як SpriteKit, GameplayKit і ReplayKit), програми для здоров'я, карти, камери, а також Siri, віртуальний помічник Apple.

Ці великі набори дозволяють скористатися перевагами функцій, вбудованих в iOS, та легко інтегрувати сторонні програми, створюючи програми, які підключаються до соціальних мереж, використовують камеру або власний додаток календаря або автоматично записують відео відтворення особливо захоплюючого ігрового моменту [2].

Наразі розробляти застосунки під iOS можна на двох мовах програмування:

- Objective-C - розроблена на початку 1980-х, Objective-C була основною мовою програмування для всіх продуктів Apple протягом

десятиліть. Похідна від мови C, Objective-C - це об'єктно-орієнтована мова програмування, орієнтована на передачу повідомлень різним процесам (на відміну від виклику процесу в традиційному програмуванні на C). Багато розробників вирішують підтримувати свої застарілі програми, написані в Objective-C, замість того, щоб інтегрувати їх у структуру Swift, яка була представлена в 2014 році.

- Swift - нова “офіційна” мова iOS. Незважаючи на те, що вона має багато подібностей до Objective-C, Swift розроблена для використання простішого синтаксису і більше зосереджений на безпеці, ніж його попередник. Оскільки він поділяє час роботи з Objective-C, ви можете легко включити застарілий код в оновлені програми. Swift легко вивчити, навіть для людей, які тільки починають програмувати. Оскільки Swift швидший, безпечніший і простіший у використанні, ніж Objective-C, вам слід планувати використовувати його для розробки програми iOS, якщо у вас немає вагомих причин дотримуватися Objective-C.

1.3 Крос-платформна розробка

Протягом останніх років різні технології та фреймворки доросли до того, щоб запропонувати рішення для створення крос-платформного застосунку. Мета полягає в тому, щоб розробити лише одну програму, до якої можна отримати доступ або розгорнути її в різних операційних системах і забезпечити однакове відчуття від користування. Одним із популярних методів є створення адаптивного веб-сайту, програми в Інтернеті, доступної для мобільних пристроїв через їх браузер. Використовуючи функції HTML5 та CSS3 разом із популярною інтерфейсною бібліотекою Bootstrap, веб-програма матиме інтерфейс користувача, який легко використовувати як із комп’ютером, так і з мобільним пристроєм. Користувацький інтерфейс та стилізація компонентів

підлаштовується відповідно до висоти та ширини екрана пристрою. Це призводить до того, що веб-програма з користувальницьким інтерфейсом стає звичайним веб-сайтом при доступі через комп'ютер, але більше схожа на програму при доступі через мобільний пристрій. Альтернативою є використання гібридного фреймворку для створення програми, яка може використовуватися будь-якою операційною системою. Цей підхід являє собою поєднання використання веб та платформної розробки, оскільки додаток створюється за допомогою веб-технік, але виконується, відображається як застосунок за допомогою WebView. Можливості пристрою розкриваються шаром абстракції як інтерфейс програмування застосунків JavaScript, що дозволяє гібридному застосунку отримувати доступ до функціональних можливостей та функцій пристрою. Однак, навіть незважаючи на те, що вартість розробки гібридних додатків значно нижча, ніж вихідних, гібриди не можуть забезпечити однаковий досвід роботи з власними користувачами, а тому вони не мали успіху в заміні програм розроблених під специфічну платформу.

На конференції React.js у 2015 році Facebook представив свій новий фреймворк React Native - фреймворк, який, на їх думку, зробив би революцію у створенні мобільних застосунків. Коли React Native був випущений, була підтримка лише iOS, але з тих пір підтримка Android була додана і продовжує розширюватися. Facebook став більш відкритою компанією, і це підхід, який вони обрали для React Native. Незважаючи на те, що вихідний код ще не повністю відкритий, Facebook намагається досягти цього і передбачає, що спільнота буде сприяти вдосконаленню системи. Основна мета React Native проста, розробник не повинен вимагати знань або витрачати зайвий час для створення мобільного застосунку, оскільки для підтримки як iOS, так і Android потрібно розробити щонайменше два додатки. Оскільки різні платформи мають різний вигляд, різні дизайн парадигми, які по різному відчувають користувачі, не може бути програми, яка є однорідною для всіх операційних систем. Однак, оскільки графічний інтерфейс відрізняється, розробка може базуватися на одній

і тій же мові, але графіка повинна відображатися по-різному залежно від цільової платформи та використовувати рідні компонентаи. Facebook називає такий підхід "вчи один раз, пиши будь-де", що описує, про що йдеться у React Native. Технологія React Native базується на React, а переваги React передаються в основу, яка застосовує її до власних застосунків. Замість того, щоб запускати React у браузері та відтворювати div-и та тексти у WebView, React Native запускається у вбудованому екземплярі JavaScriptCore (iOS) або V8 (Android) всередині застосунків та відображає на платформі вищого рівня конкретні компоненти. Компоненти JavaScript оголошуються за допомогою набору вбудованих примітивів, що підтримуються компонентами iOS або Android.

Як уже зазначалося, React Native може рендерити компоненти нативним чином для Android або iOS. Це можливо завдяки шару абстракції, відомому як "міст", який дозволяє React Native викликати API візуалізації в Java для Android або Objective-C для iOS. Крім того, фреймворк розкриває інтерфейс JavaScript, що дозволяє додатку отримувати доступ до певних функцій платформи, таких як акумулятор або геолокація. Є три основні потоки, на яких базується React Native: shadow queue (черга тіней), де обробляється макет; основний потік, де виконується візуалізація інтерфейсу користувача; потік JavaScript, де запущені всі скрипти. Ці потоки відповідають за обробку різних подій у нативній програмі [1].

Отож поточні можливості розробки на React Native дозволяють використовувати її для задач, які не потребують дуже високої швидкості роботи застосунку та не використовують специфічні для Android чи iOS API. Оскільки розробка застосунку для управління персональними фінансами – це перш за все робота з текстовими та числовими даними, здійснення запитів на сервер та відображення інформації у веб-подібному стилі, то React Native підходить у цьому варіанті.

2 ФІНАНСОВІ ІНСТРУМЕНТИ ОНЛАЙН

2. 1 Онлайн банкінг, страхування та державні послуги

Мобільний банкінг - це послуга, що надається банком або іншою фінансовою установою, що дозволяє своїм клієнтам здійснювати фінансові операції віддалено за допомогою мобільних пристроїв, таких як смартфон або планшет. На відміну від відповідного Інтернет-банкінгу, він використовує програмне забезпечення, яке зазвичай називається додатком, надане фінансовою установою для цієї мети. Мобільний банкінг зазвичай доступний цілодобово. Деякі фінансові установи мають обмеження щодо доступу до рахунків через мобільний банкінг, а також обмеження суми, яку можна використати за одну операцію. Можливості мобільного банкінгу залежать від наявності Інтернету або мобільного пристрою [3].

Операції за допомогою мобільного банкінгу залежать від особливостей конкретного застосунку для мобільного банкінгу і, як правило, включають отримання балансів на рахунках і списків останніх транзакцій, електронні платежі за рахунками, депозити за допомогою дистанційного чеку, платежі P2P та перекази коштів між рахунками клієнта, інше. Деякі програми також дозволяють завантажувати копії виписок, а іноді й друкувати їх. Використання програми для мобільного банкінгу збільшує простоту управління рахунками, швидкість, гнучкість проведення фінансових операцій, а також покращує безпеку, оскільки вона працює із вбудованими механізмами захисту мобільних пристроїв.

З точки зору банку, мобільний банкінг зменшує витрати на обробку транзакцій, зменшуючи потребу клієнтів відвідувати відділення банку для здійснення операцій з безготівкового зняття та відкриття чи закриття депозитів, кредитів. Мобільний банкінг не обробляє операції, пов’язані з готівкою, і клієнту потрібно відвідати банкомат або відділення банку для зняття готівки або депозитів. Багато програм тепер мають можливість віддаленого депозиту;

за допомогою камери пристрою для цифрової передачі чеків у свою фінансову установу [4].

Повільно, але відбувається перехід й інших фінансових інструментів у онлайн. Так зараз в Україні можна оформити ризикове страхування, страхування нерухомості чи авто, накопичувальне страхування життя онлайн. Держава відкриває реєстри у вигляді наборів відкритих даних та розробляє сервіси для отримання довідкової інформації чи сплати податків онлайн, що спрощує життя людей та компаній.

Поява всіх цих сервісів тягне за собою створення нових додатків, які треба встановлювати на телефон, інформація про розміщення коштів, сплату рахунків знаходиться у різних місцях, тому щоб зводити її докупи люди використовують електронні таблиці, записують нагадування у календар чи нотатник. Звідси походить ідея зібрати всі рахунки людини в одному додатку, щоб можна було принаймні бачити поточний стан речей та історію зарахувань і витрат грошей. Наступною метою є збільшення фінансової грамотності, надання підказок, які допоможуть людині заощадити кошти та уникнути ризиків.

2. 2 Порівняння існуючих рішень

Існує багато застосунків, які надають можливість самостійно вносити витрати та зарахування. Один з таких YNAB (You Need A Budget). Користувач може самостійно створити декілька рахунків: наприклад, картка дебетова банку А, картка кредитна банку А, картка кредитна банку Б, готівка у гаманці, готівка у конверті. Потім здійснюючи певну фінансову операцію дані потрібно вносити вручну у вигляді транзакцій, вказуючи її тип, суму, тег. Далі програма може проаналізувати ваші рахунки та дати поради. Такий вид програм і справді допомагає бачити загальну картину того, що відбувається з рахунками людини, вона є простою та зрозумілою, але майже не покращує критерій автоматизації процесу [5].

Найпопулярнішим рішенням на американському ринку є програма Mint. Mint, також відомий як Intuit Mint і раніше відома як Mint.com, - це персональний веб-сайт фінансового управління та мобільний застосунок для США та Канада, вироблені Intuit, Inc.

Спочатку Mint.com був створений Аароном Патцером і забезпечував агрегування рахунків через угоду з Yodlee, але перейшов на використання власної системи Intuit для підключення до облікових записів після того, як був придбаний Intuit в 2009 році. Пізніше його було перейменовано з "Mint.com" на "Mint". Основна послуга дозволяє користувачам відстежувати залишки та транзакції банківських, кредитних карток, інвестицій та позик за допомогою єдиного інтерфейсу користувача, а також створювати бюджети та встановлювати фінансові цілі.

Станом на 2010 рік Mint.com стверджував, що має зв’язок з більш ніж 16 000 фінансовими установами США та Канади та підтримує понад 17 мільйонів індивідуальних фінансових рахунків. У 2016 році Mint.com стверджував, що має понад 20 мільйонів користувачів.

Mint Bills, раніше відомий як Check and Pageonce, був послугою управління фінансовими рахунками та оплатою рахунків, придбаною Intuit в 2014 році та інтегрованою в Mint.com у березні 2017 р. Потім послуга оплати рахунків Mint.com була припинена 30 червня 2018 р.

Mint є відомим в США, оскільки він має доступ до найбільшої кількості фінансових сервісів та займається кредитним скорингом - класифікацією позичальників на групи для оцінки їх кредитоспроможності та рівня кредитного ризику на основі кредитної історії та соціально-демографічних характеристик [6].

3 ВИМОГИ ДО ЗАСТОСУНКУ

3.1 Користувачі

Перша версія застосунку підтримуватиме два типи користувачів. Для більшості фінансових застосунків притаманна бізнес-модель з лімітованим преміум періодом протягом якого користувачу будуть доступні всі функції, далі ж він матиме можливість продовжити користуватися застосунком лише за підпискою. Дана модель є менш ефективною для нових застосунків, про які не знає велика аудиторія. Тому оптимальнішим буде вибір freemium моделі, за якою користувач отримує частину функціоналу безкоштовно, бачить рекламу, а щоб отримати доступ до просунутих інструментів, має оформити підписку. Відповідно до запропонованого варіанту можливим розділенням безкоштовних та платних функцій може бути наступна:

Безкоштовні: додавання до певної кількості рахунків, перегляд виписки та зведеної інформації лише по них.

Підписка: необмежена кількість рахунків, аналітика у вигляді діаграм та графіків, рекомендації.

При першому відкритті людина бачить основну інформацію про можливості застосунку у вигляді каруселі з тезами, які найкраще описують переваги застосунку. Внизу екрану розміщені кнопки «Увійти» та «Зареєструватися».

Під час реєстрації користувач вводить лише найнеобхіднішу інформацію: ПІБ, номер мобільного телефона, електронну пошту. Підтвердження акаунту відбувається шляхом переходу за посиланням надісланим на електронну пошту.

Авторизований користувач на головному екрані бачить загальний баланс всіх своїх карток та електронних гаманців, ця інформація має бути найпомітніше відображеною. Також на головному екрані є список останніх транзакцій агрегованих з усіх рахунків та кнопка, яка відкриє екран з усіма транзакціями.

Біля числа з загальним балансом має розміщуватися кнопка «Всі рахунки», яка переводитиме на екран зі списком всіх рахунків, клік по будь-якому з них переводить на екран для редагування назви чи інтеграційних ключів. Також на екрані з усіма рахунками має бути можливість додати новий рахунок. Користувач вибирає провайдера акаунту рахунку, вводить потрібні для нього дані та після цього натискає на кнопку «Підключити».

Для додавання готівкових транзакцій розробити екран, де можна ввести назву транзакції, її опис та суму, яка може бути як від'ємним, так і додатнім числом.

3.2 Архітектура системи

Користувацький інтерфейс реалізовується мовою TypeScript з допомогою фреймворку React Native. Процес неперервної інтеграції та доставки застосунку може бути налаштований через системи Jenkins, TeamCity, CircleCI. Для запуску end-to-end тестів знадобиться запуск віртуального пристрою у хмарі або декілька bare metal пристроїв Android та iOS. У другому випадку потрібно буде на більш низькому рівні керувати виділенням пристроїв під запуск тестів, тож для прототипної версії краще використати запуск віртуальних пристроїв.

Онвлення версій застосунку можна зробити регулярним: наприклад у певний день тижня о 20:00 обирається останній «зелений» (той що пройшов усі перевірки CI/CD) комміт з головної гілки репозиторію, для нього створюємо RC (release candidate) версію, збираємо .apk та .ipa файли, передаємо їх тестувальникам. Якщо у тестувальників немає зауважень і все працює як слід можна починати дистрибуцію у Google Play та AppStore. Процес показаний на рисунку 1.

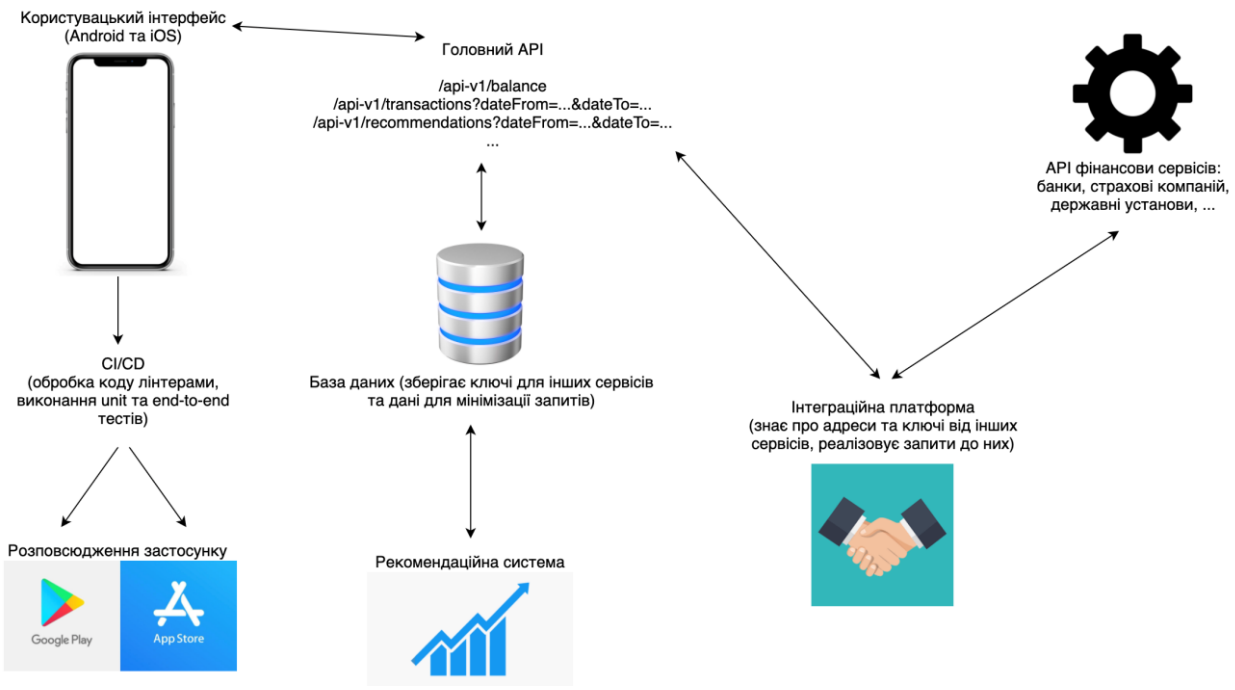

Рисунок 1 – Архітектура системи

3.3 Node.js

Node.js - це середовище виконання з відкритим вихідним кодом, міжплатформене та внутрішнє середовище виконання JavaScript, яке працює на двигуні V8 і виконує код JavaScript поза веб-браузером. Завдяки Node.js розробники можуть створювати утиліти для командного рядка та обробку серверних запитів, комунікацію з базами даних. Отже, Node.js представляє парадигму "JavaScript скрізь", що об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для сценаріїв на стороні сервера та клієнта. Поява Node.js дозволила писати веб сайти повністю однією мовою [7].

Хоча .js - це стандартне розширення імені файлу для коду JavaScript, назва "Node.js" не посилається на певний файл у цьому контексті, а є лише назвою продукту. Node.js має керовану подіями архітектуру, здатну до асинхронного вводу-виводу. Ці варіанти дизайну спрямовані на оптимізацію пропускної здатності та масштабованості у веб-додатках з багатьма операціями введення / виведення, а також для веб-додатків у реальному часі (наприклад, програм спілкування та браузерних ігор).

Раніше розподілений проєкт розробки Node.js керувався Node.js Foundation, і тепер він об’єднався з JS Foundation, щоб сформувати OpenJS Foundation, що сприяє програмі спільних проектів Linux Foundation.

Node.js схожий за дизайном та розроблявся під впливом таких систем, як Ruby's Event Machine та Python's Twisted. Node.js підтримує модель події трохи краще. Він представляє цикл подій як конструкцію середовища виконання, а не як бібліотеку. В інших системах завжди є виклик блокування для запуску циклу подій. Як правило, поведінка визначається через зворотні виклики на початку сценарію, а в кінці сервер запускається через блокуючий виклик, як EventMachine :: run (). У Node.js немає такого виклику циклу запуску події. Node.js просто вводить цикл подій після виконання вхідного сценарію. Node.js виходить із циклу подій, коли більше немає зворотних викликів. Ця поведінка схожа на JavaScript браузера - цикл подій прихований від користувача [8].

3.4 Express.js

Express.js (або просто Express) — бібліотека для розробки серверної частини веб-застосунків та API для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Express довгий час вважається стандартною частиною ядра Node.js. Автор фреймворка, TJ Holowaychuk, описує його як дуже мінімалістичний, але з великою кількістю плагінів. TJ Holowaychuk створив Express на основі написаного на мові Ruby каркаса Sinatra.

Express є бекендом для програмного стека MEAN, де використовується база даних MongoDB і фреймворк AngularJS для фронтенду. Частково MEAN був обраний для реалізації застосунку, окрім фронтенд частини, де було використано стек React/React Native [9].

3.5 MongoDB

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB є однією з найпопулярніших баз даних, що оперують даними у форматі ключ-значення, цінується у швидких та масштабованих системах, є зручною для форматування запитів.

Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має гнучку мову для формування запитів, може створювати індекси для збережених атрибутів, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД. У MongoDB є вбудовані засоби для забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин.

MongoDB масштабується горизонтально. Користувач вибирає ключ-фрагмент, який визначає спосіб розподілу даних у колекції. Дані поділяються на діапазони (на основі ключа) і розподіляються між кількома shard - осколками. Осколок - це майстер-об’єкт з однією або кількома копіями. Крім того, ключ осколка можна хешувати, щоб зіставити його з осколком, що забезпечує рівномірний розподіл даних [10].

MongoDB може працювати на декількох серверах, балансуючи навантаження або дублюючи дані, щоб підтримувати роботу системи в разі несправності обладнання.

MongoDB пропонує три способи виконання агрегації: конвеєр агрегування, функція зменшення map та одноцільові методи агрегування.

Зменшення map можна використовувати для пакетної обробки даних та операцій агрегування. Але згідно з документацією MongoDB, процес агрегації забезпечує кращу продуктивність для більшості операцій агрегування.

Структура агрегування дозволяє користувачам отримувати результати, для яких використовується речення SQL GROUP BY. Оператори агрегації можуть бути об’єднані. Структура агрегування включає оператор пошуку $, який може приєднувати документи з декількох колекцій, а також статистичні оператори, такі як стандартне відхилення.

3.6 React Native Navigation

Як правило, будь-яка мобільна програма складається з різних станів, які відображають певний вміст для користувача. І у переважній більшості випадків використання програми означає перехід між цими пунктами призначення. React n Native Navigation забезпечує способи розміщення вмісту на екрані користувача та переходи між ними в логічній та ефективній формі.

Макет стека React Native Navigation дозволяє пересувати екрани, а також повертатися до попередніх екранів. Екрани, всунуті в стек, приховують попередній екран у стеку, завдяки чому користувач фокусується на одному екрані за раз. Давайте розглянемо макет стека, який забезпечує базову навігацію для програми.

Кореневий екран - це точка входу у макет застосунку. Зазвичай це перший елемент інтерфейсу користувача, з яким він взаємодіє. Корінь можна змінювати кілька разів протягом життя програми. Наприклад, якщо програма вимагає входу користувачів, зазвичай використовується кореневий код на

основі стека та перехід до кореневого режиму на вкладках або бокового меню, якщо вхід успішний. У випадку нашого застосунку кореневим екраном для авторизованого користувача є екран з балансом та списком нещодавніх транзакцій, якщо користувач не увійшов до системи, то у стеку перебуватиме екран з банерами та кнопками входу та реєстрації.

Оскільки мобільні додатки стають більшими та складнішими, вони, як правило, містять кілька основних пунктів призначення високого рівня, які логічно не залежать одне від одного. Макет BottomTabs (нижні вкладеи) часто використовується, коли програма містить від трьох до п’яти пунктів призначення верхнього рівня, до яких має бути доступ з будь-якої точки програми.

Розглянемо приклад того як основну навігацю можна перетворити його на застосунок на основі вкладок. Використовуватимемо макет BottomTabs для відображення вкладок внизу екрана. Подібно до макета стека, макет BottomTabs також містить дочірню властивість, де кожний дочірній елемент відображатиметься на вкладці.

Стек - це макет контейнера, що сприяє ієрархічній навігації. Застосовується для навігації між екранами на послідовних рівнях ієрархії, кроків у потоці або між програмами.

Першим дочірнім елементом у стеку (представленим дочірнім масивом) є корінь і відображається внизу стека. Останньою дочірньою структурою дочірнього масиву є дочірня дитина, яка відображається.

У цьому макеті в будь-який момент часу видно лише один дочірній екран, і послідовний екран можна додати у верх стека за допомогою команди Navigation.push. Натискання кнопки "Назад" відкриє стек і видалить верхній екран.

Стек управляє TopBar у верхній частині екрана. На верхній панелі відображаються назва та кнопки поточного екрана. Його можна приховати за допомогою опції topBar: {visible: false}. За замовчуванням екрани

відображаються під TopBar. Цю поведінку можна змінити, встановивши topBar: {drawBehind: true} у параметрах поточного екрана.

У дизайні, орієнтованому на людину, коли ми говоримо про модальний спосіб, ми часто маємо на увазі набір методів, спрямованих на привернення уваги користувача до певної події / екрану / дії / тощо. Модали часто вимагають введення з боку користувача. Спливаюче вікно на веб-сайті, діалогове вікно підтвердження видалення файлу на вашому комп'ютері або попередження з проханням увімкнути службу визначення місцезнаходження на телефоні - все це можна вважати модальним способом. Такі вікна-модали використовуватимемо для виведення найважливіших повідомлень.

Модал (modal) - це термін, який використовується в світі iOS, тоді як на Android часто можна почути про діалоги (dialogs) для імплементації подібних поведінок.

3.7 Remx

Управління станом відноситься до управління станом одного або декількох елементів керування інтерфейсом користувача, таких як текстові поля, кнопки, перемикачі тощо в графічному інтерфейсі користувача. У цій техніці програмування користувальницького інтерфейсу стан одного елемента керування інтерфейсом залежить від стану інших елементів керування інтерфейсом. Наприклад, елемент керування інтерфейсом, такий як кнопка, буде у включеному стані, коли поля введення мають провалідовані вхідні значення, а кнопка буде у вимкненому стані, коли поля введення порожні або мають незадовільні значення. У міру зростання застосунків це може в кінцевому підсумку стати однією з найскладніших проблем у розробці користувальницького інтерфейсу [11].

Особливо це стосується випадків, коли стан будь-якого конкретного повідомлення чи форми на сторінці залежить від факторів, що перебувають поза поточною сторінкою, або доступні на кількох сторінках. Наприклад,

розглянемо користувача, який увійшов до системи та бачить повідомлення про привітання під час першого відвідування будь-якої сторінки, але не під час наступних відвідувань. Чи кожна сторінка керує станом користувача, до якого входить? Це створило б занадто багато вставлення копій та дублювання коду. Натомість можна використовувати шаблон управління станом для обробки повідомлень (це може також включати обробку повідомлень про помилки та інформативних повідомлень, разом з описаним привітальним повідомленням), щоб потім викликати його, щоб отримати повідомлення, коли воно стане доступним.

3.8 Detox

Detox - це end-to-end система для тестування React Native застосунків. Це означає, що вона запускає застосунок на реальному пристрої або симуляторі та взаємодіє з ним так само, як справжній користувач. Цей тип тестування може надати впевненості у вашому додатку та допомогти автоматизувати процес контролю якості.

Коли виконується тест Detox, ви насправді маєте дві різні частини, що працюють поруч:

Сам мобільний застосунок, як правило, працює на симуляторі / емуляторі. На пристрої встановлюється та виконується звичайна власна збірка застосунку. Програма зазвичай створюється один раз перед початком роботи тестів.

Тестовий пакет, що працює на Node.js, використовує Jest. Зазвичай тести написані на JavaScript або TypeScript. Оскільки тести мають асинхронний характер (кожен тестовий рядок вимагає доступу до програми та очікування відповіді), тести значною мірою покладаються на async-await.

Ці дві частини, як правило, працюють окремо на вашій машині. Також можна запустити дві деталі на різних машинах. Зв'язок між двома частинами відбувається через мережу за допомогою веб-сокета.

На практиці, щоб зробити зв'язок більш стійким, обидві частини реалізовані як клієнти та взаємодіють із сервером Detox, який діє як проксі. Це дозволяє, наприклад, одній стороні відключатися (під час завантаження симулятора, наприклад, або перезапуску програми), не відключаючи іншу сторону та втрачаючи свій стан [12].

Однією з ключових особливостей Detox є його здатність автоматично синхронізувати виконання тесту з застосунком. Найбільш непевний аспект end-to-end тестів - це нестійкість - тести іноді дають збої. Нестабільність буває тому, що тести недетерміновані. Щоразу, коли запускається тест, все відбувається у дещо іншому порядку.

Розглянемо сценарій, коли програма одночасно робить кілька мережевих запитів. Який порядок виконання? Це залежить від того, який запит буде виконаний першим. Це залежить від завантаженості мережі та наскільки зайнятий сервер.

Традиційний метод боротьби з нестабільністю - це додавання різних команд sleep(), waitFor() протягом тесту, намагання створити певний порядок виконання. Це погана практика, повна великої кількості неточних значень часу, які часто змінюються, якщо машина, що запускає тести, стає швидшою або повільнішою.

Detox намагається усунути нерівність, автоматично синхронізуючи тести із застосунком. Тест не може переходити до наступної команди. Detox уважно стежить за застосунком, щоб знати, коли він не працює, відстежує кілька асинхронних операцій одночасно і чекає, поки вони завершаться. Це включає у себе:

- Відстеження всіх мережевих запитів, які зараз перебувають у виконанні, та очікування завершення;
- відстеження очікуваних анімацій та очікування їх завершення;
- відстеження таймерів та очікування їх закінчення чи скасування;
- відстеження операцій React Native.

У нашому випадку Detox використаний для написання тестів із fake сервером, який містить тестові дані (mock data), щоб не виконувати запити до реального сервера. Втім для точної перевірки чи працюють правильно основні функції, найпопулярніші випадки взаємодії із застосунком покриваються тестами із запитами до реального сервера. У такому випадку якщо у структурі відповідей сервера відбудуться зміни, про які не знатиме клієнтська частина, можна буде швидше ідентифікувати проблему та почати її вирішення.

4 РОЗРОБКА СЕРВЕРА ТА МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Розробка back end частини

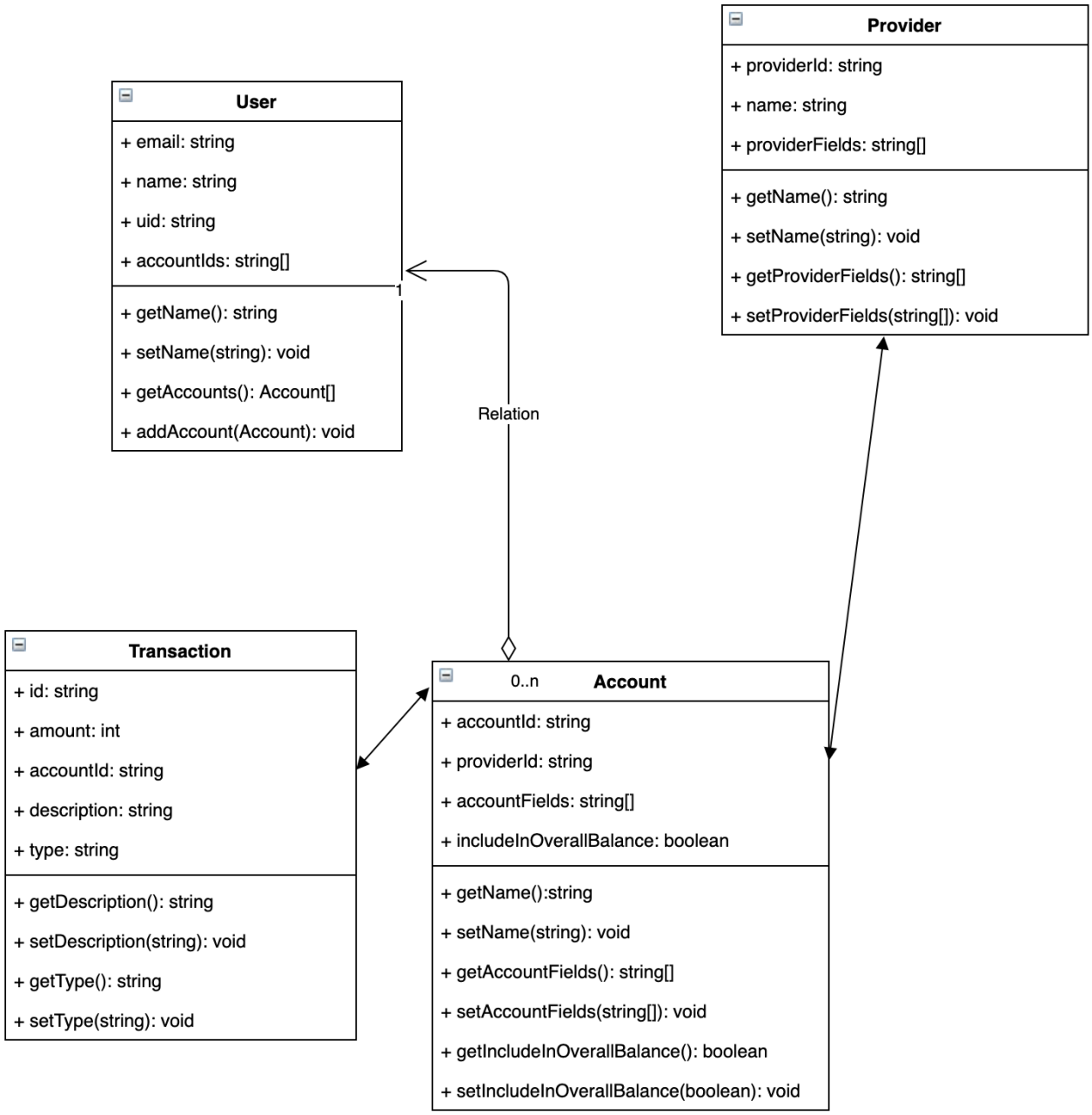

Рисунок 2 – Діаграма класів

4.1.1 Автентифікація

Першим кроком у реалізація сервера було створення можливості реєструвати та автентифікувати користувачів. Оскільки режим доступу до застосунку матиме єдину роль і у системі відсутній адміністратор, то для вирішення задачі було використано Firebase Auth REST API. Для першої версії було реалізовано можливість реєстрації за допомогою електронної пошти та

пароля, але сервіс надає зручні API для розширення реєстрації та підтвердження користувача за допомогою мобільного телефона чи наприклад oAuth провайдерів Google, Facebook, Apple та інших.

Для реєстрації нового користувача надсилається запит на ендпоінт основного API системи /api-v1/signUp. У body запиту повинні бути наступні дані: name, email, password. Обробник запиту спершу здійснить пошук за базою даних чи не зареєстрований користувач із такою електронною поштою, якщо так, то поверне відповідну помилку. Якщо ж ні, то він здійснить запит на https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=[API_KEY] з параметами email, password та returnSecureToken=true, де API_KEY – це ключ зареєстрованого проєкту у Firebase, який є однаковим для всіх користувачів і зберігається серед системних змінних сервера. Firebase самостійно створить uid для нового користувача, який разом з даними idToken, email, refreshToken та expiresIn прийдуть у відповідь. Дані про uid, електронну пошту та ім’я зберігаються до нашої бази даних у документі users. На запит користувача з клієнта повертаємо всі дані, які отримали з identitytoolkit.googleapis.com.

Надалі для авторизації зареєстрованого користувача використовується схожий підхід, де наш Express сервер буде виконувати функцію Proxy-сервера, який перенаправляє запит на Firebase Auth REST API.

Для отримання даних пов’язаних з акаунтом конкретного користувача до запитів треба у body додавати параметр uid.

4.1.2 Рахунки

Після реєстрації у застосунку користувач поки не має жодного підв’язаного рахунку, тому щоб його додати він має здійснити запит на /api-v1/connectAccount?provider=[PROVIDER_ID]. Список провайдерів з їх id та назвами дістаються за запитом на /api-v1/connectAccount?provider=[PROVIDER_ID].

Провайдером може бути будь-яка фінансова установа, де користувач зберігає чи інвестує свої кошти, але для першої версії застосунку до уваги беруться лише банківські системи, адже це найпопулярніший інструмент, і майже у кожного є банківська картка, а зазвичай їх декілька, тому й виникає потреба зібрати дані в одному місці.

З п’ятірки найбільших Українських банків за кількістю платіжних карток в обігу (ПриватБанк, Ощадбанк, Райфайзенбанк Аваль, Пумб, Альфа-Банк) лише ПриватБанк надає публічні API для роботи з фізичними клієнтами. На сайті з документацією ПриватБанк констатує, що став першим банком у світі, що відкрив свій API у вересні 2009 року, але з огляду на стан документації, яка доступна лише російською мовою та використання технологій SOAP та формату XML для передачі даних, оновлювалися ці API досить давно [13]. Також з огляду на появу та розвиток необанків – банків, які не мають фізичних відділень, тобто клієнт може оформити рахунок лише онлайн через сайт чи мобільний застосунок, візьмемо найбільший за кількістю користувачів український необанк Monobank, який також надає публічний API для отримання даних для фізичних осіб [14].

Для початку робити з API Приватбанку потрібно:

1. Увійти в особистай акаунт Приват24 для фізичних осіб за допомогою посилання http://privat24.ua;
2. Ввести логін і статичний пароль;
3. Введіть динамічний пароль OTP, отриманий в СМС чи мобільному застосунку;
4. Перейти в розділ меню «Всі послуги» -> «Бізнес» -> «Мерчант»;
5. Прив’язати карту для роботи з мерчантом;
6. Вказати IP-адресу інтернет-ресурсу, з якого здійснюються запити;
7. Відмітити необхідні сервіси, до яких надаватиметься доступ (для початку наприклад знадобляться лише сервіси, що повідомляють дані про баланс та виписки картки);
8. Підтвердити пароль OTP;

9. Після успішної реєстрації мерчанту присвоюється ID та пароль. Мерчант знаходиться у статусі "тестовий" (можна здійснювати запити за дозволеними сервісами).

Усі ці кроки буде змушений здійснювати користувач нашого застосунку, щоб надати ключі доступу до вказаних сервісів своєї картки. За оновленими стандартами, які використовують платіжні системи та інші відомі застосунки процес надання таких дозволів відбувається за допомогою oAuth: щоб надати третім сторонам доступ до своїх даних користувач може перейти за посиланням з самого додатку в браузер, пройти авторизацію у сервісі, з якого братимуться дані, і у разі успішної авторизації сервіс виконає переправлення (redirect) на інший url, а в тілі відповіді до запиту будуть всі потрібні ключі. Але поки ні ПриватБанк, ні Монобанк не надають можливості здійснювати авторизацію за допомогою oAuth, тому всі ключі та ідентифікатори користувач має передати нам самостійно, і вони потім будуть використані для отримання даних з відповідного сервісу, як це відбувається можна побачити на прикладі інтеграції з API ПриватБанку у додатку А.

Також варто наголосити, що ПриватБанк дозволяє робити запити лише з одного IP, тобто користувачі коли будуть реєструвати мерчанта у Приват24 матимуть вказувати саме IP адресу нашого сервера. Для зручності користування, щоб не треба було постійно змінювати адресу у налаштуваннях Приват24, треба підтримувати статичну адресу сервера, що можна зробити скориставшись наприклад ElasticIP у випадку використання AWS або QuotaGuard Static IP - у випадку Heroku для виконання серверної частини застосунку.

Отож після того як користувач отримав відповідні дозволи від банку-провайдера він має передати id, ключ та номер картки. Взагалі окрім номера картки нічого більше не потрібно зберігати, ні дату до якої вона дійсна, ні CVV-код, тому застосунок буде відповідати PCI DSS. Payment Card Industry Data Security Standard (PCI DSS) — стандарт безпеки даних індустрії платіжних карток, розроблений Радою зі стандартів безпеки індустрії платіжних карток

(Payment Card Industry Security Standards Council, PCI SSC), заснованою міжнародними платіжними системами Visa, MasterCard, American Express, JCB і Discover. Стандарт являє собою сукупність 12 деталізованих вимог щодо забезпечення безпеки даних про власників платіжних карток, які передаються, зберігаються і обробляються в інформаційних інфраструктурах організацій. Прийняття відповідних заходів щодо забезпечення відповідності вимогам стандарту представляє комплексний підхід до забезпечення інформаційної безпеки даних платіжних карток [15].

У тілі POST запиту на /api-v1/connectAccount?providerId=[PROVIDER_ID] треба передати потрібні для підключення рахунку даного провайдера. Щоб отримати список потрібних полів виконується запит на /api-v1/providerFields?providerId=[PROVIDER_ID]. Оскільки для кожного з провайдерів свої поля та типи, тож цей набір буде формуватися окремо у коді серверної частини. Але з часом, коли кількість провайдерів буде збільшуватися, варто створити інтеграційну платформу – окремий сервер з відповідним фронт енд клієнтом, щоб можна було швидше та без написання подібного коду виконати інтеграцію. Ця платформа може бути відкритою для партнерів, на ній будуть розміщені відомості про всі можливості API фінансової установи, які мають бути реалізовані для інтеграції, документування того як саме відбувається інтеграція. Це зробить наш сервіс більш прозорим та дасть можливість будь-кому зрозуміти, які дані будуть використані, та з якою метою.

Для оновлення вже підключеного акаунта треба користатися тим самими /api-v1/providerFields?provider=[PROVIDER_ID] з тими ж самими параметрами, що повертаються /api-v1/providerFields?providerId=[PROVIDER_ID] у body.

У випадку зміни провайдером відповідних полів система потребуватиме оновлення з нашої сторони, але за умови вже реалізованої інтеграційної платформи співробітники провайдера зможуть змінити ці дані значно швидше. Щоб не потрапити у ситуацію, коли багато запитів буде хибними саме з

причини відсутності потрібних полів треба ігнорувати виконання таких запитів та певним чином повідомити користувача застосунку, що треба оновити дані.

Кожному акаунту буде присвоєно унікальний id та назва, яку самостійно зможе змінювати користувач. Щоб отримати дані про акаунти треба надіслати запит на /api-v1/accounts без додаткових параметрів, обов'язковим є лише параметр uid в body.

4.1.3 Баланси

Для агрегації даних про баланси всіх рахунків створимо наступний ендпоінт /api-v1/balance, у тілі відповіді буде лише одне поле balance, яке буде сумою балансів на усіх рахунках, що додав користувач.

На жаль, ні ПриватБанк, ні Monobank не надають доступу до інформації про депозити та кредити користувача, але припустимо що з часом спектр інформації доступної через їхні API збільшиться і акаунтом зможуть бути не тільки картки, а й інші способи вкладень. У такому випадку список рахунків збільшиться, але не всі з них користувач хотітиме бачити у загальній сумі балансу. Тому доцільним рішенням буде надати змогу користувачеві визначати чи враховувати баланс певного рахунку у загальний баланс. Для цього додамо в структуру класу Account (показано на рисунку 2) поле includeInOverallBalance, значення якого можна оновлювати відповідно через /api-v1/connectAccount? provider=[PROVIDER_ID].

4.1.4 Готівкові операції

Для того, щоб контролювати готівкові операції додамо до системи провайдера "Cash", який не потребуватиме жодних полів для заповнення на етапі створення окрім поточної суми у готівці, адже це не сторонній провайдер, а провайдер всередині застосунку. Його імплементацію відділимо від сервісів інтеграційної платформи.

Баланс на рахунку акаунта провайдера “Cash” буде оновлюватися після внесення транзакцій, але слід підтримувати й можливість коритувача імперативно оновити баланс готівкового рахунку, адже не всі транзакції обов’язково вноситимуться у застосунок, що створить різницю між реальною сумою готівки та тим, що записано на сервері.

4.1.5 Транзакції

У першій версії застосунку жодні з транзакцій окрім готівкових не будуть зберігатися на нашому сервері, тому доведеться обробляти, групувати, сортувати їх у режимі реального часу виходячи з тих даних, що нададуть API інтегрованих сервісів. Це збільшує кількість виконання однотипних операцій, які можуть бути складними в обчисленні. Наприклад задля визначення до якого типу відноситься транзакція (продукти, побутова техніка, подорожі, медицина, …) потрібно обробляти текстове поле з описом транзакції, де може вказуватися назва товару, назва та адреса магазина. Це неструктуровані дані, які потребують додаткових алгоритмів, щоб визначити з певною точністю до якої групи товарів відноситься транзакція, тому доцільно на певний час кешувати дані по коду транзакції не зберігаючи ніяких додаткових відомостей про неї. Оскільки кількість транзакцій може бути доволі великою, то ендпоінт для їх підвантаження матиме параметри, які визначатимуть проміжок часу, за який треба розглянути транзакції: /api-v1/transactions?dateFrom=[DATE_FROM_TIMESTAMP]&dateTo=[DATE_TO_TIMESTAMP]. Повернутий список міститиме транзакції з усіх рахунків користувача, згодом можна додати фільтри за рахунками.

4.2 Розробка мобільної частини

4.2.1 Екран привітання

Якщо в AsyncStorage не збережено ніяких даних про авторизованих користувачів, то кореневим екраном встановлюється екран привітання. На ньому відображаються основні тези про можливості застосунку. Знизу знаходяться дві кнопки «Вхід» та «Реєстрація», натискання на які призводить до відкриття відповідних екранів.

4.2.2 Екрани входу та реєстрації

Екрани входу та реєстрації реалізовуються через один компонент LoginScreen, адже їх єдина відмінність у тому, що при реєстрації окрім електронної пошти та паролю треба вводити ще й ім’я, а також кнопка матиме інший текст та іншу реалізацію функції onPress. Екран привітання буде передавати у навігацію серед props значення isRegistrationScreen: true, що ідентифікуватиме яку роль виконує модал LoginScreen.

Після успішного входу у AsyncStorage зберігаються authTokent, таймстемп, до якого часу він дійсний та refreshToken.

4.2.3 Головний екран

При завантаженні головного екрану асинхронно виконуються запити на отримання загального балансу та 5 останніх останніх транзакцій за поточний місяць. Для того, щоб додаток міг працювати в офлайні, отримані дані зберігаються як у Remx state, так і у AsyncStorage.

На головному екрані великим шрифтом відображається загальний баланс рахунків користувача, під цим числом є кнопка «Всі рахунки», натискання на яку відкриває екран зі списком рахунків.

Якщо користувач не здійснив жодної транзакцій за поточний місяць, то він бачитиме лише кнопку «Всі транзакції», натисканням на яку можна перейти на екран зі списком транзакцій.

4.2.4 Екран рахунків

При відкритті екрану відбувається завантаження даних про рахунки для користувача, отримані дані зберігаються лише у state оскільки використовуються рідше та непотрібні в офлайн режимі. Після завантаження відображається список рахунків, де для кожного рахунку видно його назву, баланс та провайдера. Якщо для провайдера є відповідний логотип, то показувати його. Натискання на елемент рахунку переводить на екран управління.

4.2.5 Екран підключення рахунку та управління рахунком

Для підключення та управління використовується однаковий компонент ConnectAccountScreen, адже поля, як і функція onPress однакова для обох випадків. Потрібно лише змінювати текст на кнопці: якщо рахунок вже підключений, то напис «Оновити», якщо ще ні – «Підключити».

Перед кнопкою показується форма у вигляді списку полів, які вимагають провайдер та акаунт для надання доступу нашому додатку, до даних користувача. Всі вони або текстового, або числового формату.

4.2.6 Екран транзакцій

Зважаючи на інформацію згадану у секції 4.1.5 про потребу здійснювати обробку транзакцій у режимі реального часу після запиту користувача, вирішено показувати список транзакцій за конкретний часовий період без можливості завантажувати додаткові транзакції, коли користувач догортав до кінця списку. За замовчуванням таким періодом буде 1 місяць. Тобто на екрані відображається список транзакцій здійснених за всіма рахунками за останній місяць. Дані про транзакції зберігаються лише у state.

Компонент Transaction містить суму транзакції, категорію та назву рахунку, з якого вона була здійснена.

ВИСНОВКИ

При виконанні роботи було порівняно найпопулярніші операційні системи для мобільних пристроїв, Android та iOS, нативну та кросплатформенну розробку застосунків для цих ОС. Детально розглянуто сучасні підходи до розробки за допомогою фреймворку React Native.

Також розглянуто застосунки для управліннями фінансами серед яких Mint, YNAB, PocketGuard. Розглянуто ініціативи США, Великої Британії та Європейського Союзу задля створення загальних вимог до діджиталізації фінансових установ, які можуть стати засадами для створення громадської організації в Україні, яка займатиметься подібними питаннями у нашій державі.

Відповідно до поточних можливостей сучасних українських онлайн-банків, страхових компаній та державних установ, було складено план розробки мобільного застосунку, який дозволить зібрати дані про доходи та витрати людини в одному місці, дозволивши користувачам бачити загальну ситуацію зі своїми рахунками, динаміку надходжень та витрат, отримувати поради щодо фінансової грамотності.

Було розроблено вимоги щодо клієнтської та серверної частини мобільного застосунку включно з інтеграційною платформою, яка на даний момент реалізовує інтеграції з двома провідними банками України: ПриватБанк та Монобанк.

У рамках створення прототипної версії застосунку було імплементовано один сервер, який виконує роль головного API та інтеграційної платформи, дозволяє зберігати та отримувати дані про рахунки користувача, загальний баланс та окремо баланс для кожного з рахунків, список транзакцій, готівкові операції користувача. Також було розроблено прототип мобільного застосунку для операційних систем iOS та Android, який реалізує основні екрани та компоненти потрібні для управління персональними фінансами, може частково працювати в офлайн режимі.

Для повноцінної реалізації застосунку, який буде повністю готовим для використання реальними користувачами, потрібно: покращення алгоритмів валідації даних та налаштування моніторингу помилок для серверної та клієнтської частин, імплементація логування для серверної частини, створення дизайн системи та імплементація компонентів для мобільного застосунку, розробка моделі монетизації та дослідження ставлення банків до створення подібного застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Danielsson, W., 2016. React Native application development: A comparison between native Android and React Native.
2. iOS app development [Електронний ресурс] – Режим доступу: https://www.ibm.com/cloud/learn/ios-app-development-explained.
3. Open Banking [Електронний ресурс] – Режим доступу: https://developer.americanexpress.com/open-banking.
4. Understanding GDPR: How will the Financial Services sector be affected? [Електронний ресурс] – Режим доступу: https://www2.deloitte.com/mt/en/pages/risk/articles/mt-gdpr-fsi.html.
5. Party Like It's 2017 (and We Launched a New Mobile App) Introducing YNAB's All-New, Full-Featured Mobile App [Електронний ресурс] – Режим доступу: https://www.youneedabudget.com/fashionably-late-but-worth-the-wait/.
6. What is Mint, and how does it work? [Електронний ресурс] – Режим доступу: https://mint.intuit.com/how-mint-works.
7. About Node.js [Електронний ресурс] – Режим доступу: https://nodejs.org/en/about/.
8. The Complete Node.js Developer Course (3rd Edition) [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/the-complete-nodejs-developer-course-2/.
9. Express API Reference [Електронний ресурс] – Режим доступу: https://expressjs.com/en/4x/api.html.
10. What Is MongoDB? [Електронний ресурс] – Режим доступу: https://www.mongodb.com/what-is-mongodb.
11. Remx Documentation. Motivation [Електронний ресурс] – Режим доступу: https://wix.github.io/remx/docs/introduction/motivation.
12. How Detox Works Motivation [Електронний ресурс] – Режим доступу: https://github.com/wix/Detox/blob/master/docs/Introduction.HowDetoxWorks.md .

13. Доступні API ПриватБанка [Електронний ресурс] – Режим доступу: https://api.privatbank.ua/.
14. Monobank open API [Електронний ресурс] – Режим доступу: https://api.monobank.ua/docs/.
15. PCI Security [Електронний ресурс] – Режим доступу: https://www.pcisecuritystandards.org/pci_security/.

ДОДАТОК А

Код написаний для реалізації інтеграції з API ПриватБанку для отримання виписки користувача за певний термін:

```
import { KEY, cardNumber } from "../keys/pb24";

const crypto = require("crypto");
const convert = require("xml-js");
const sha1 = (s) => crypto.createHash("sha1").update(s).digest("hex");
const md5 = (s) => crypto.createHash("md5").update(s).digest("hex");
const https = require("https");

const options = { compact: false };

const hideCard = (card) => `***${card.slice(-4)}`;
const localizeCurr = (curr) => {
  switch (curr.toLowerCase()) {
    case "uah":
      return "грн";
    default:
      return curr.toLowerCase();
  }
};
const splitAmount = (s) => {
  let amount = "";
  let curr = "";
  for (let i = 0; i < s.length; i += 1) {
    if (/[-.0-9]/.test(s[i])) {
      amount += s[i];
    } else if (s[i] !== " ") {
      curr = s.slice(i);
      break;
    }
  }
  return [amount, localizeCurr(curr)];
};
const formatDate = (d) =>
  `${d.getDate()}.${d.getMonth() + 1}.${d.getFullYear()}`;
const daysAgo = (days) => {
  const d = new Date();
  d.setDate(d.getDate() - days);
  return d;
};

const startDate = daysAgo(3);
const endDate = new Date();

const data = {
  type: "element",
  name: "data",
  elements: [
    {
      type: "element",
      name: "oper",
      elements: [{ type: "text", text: "cmt" }],
    },
    {
      type: "element",
      name: "wait",
      elements: [{ type: "text", text: "0" }],
```

```
    },
    {
      type: "element",
      name: "test",
      elements: [{ type: "text", text: "0" }],
    },
    {
      type: "element",
      name: "payment",
      attributes: { id: "" },
      elements: [
        {
          type: "element",
          name: "prop",
          attributes: { name: "sd", value: formatDate(startDate) },
        },
        {
          type: "element",
          name: "prop",
          attributes: { name: "ed", value: formatDate(endDate) },
        },
        {
          type: "element",
          name: "prop",
          attributes: { name: "card", value: cardNumber },
        },
      ],
    },
  ],
};

const dataxml = convert.json2xml(data, options);

const signature = sha1(md5(dataxml + KEY));

const result = convert.json2xml(
  {
    elements: [
      {
        type: "element",
        name: "request",
        attributes: { version: "1.0" },
        elements: [
          {
            type: "element",
            name: "merchant",
            elements: [
              {
                type: "element",
                name: "id",
                elements: [{ type: "text", text: "184965" }],
              },
              {
                type: "element",
                name: "signature",
                elements: [
                  {
                    type: "text",
                    text: signature,
                  },
                ],
              },
            ],
```

```
          },
          data,
        ],
      },
    ],
  },
  options
);

const req = async () =>
  new Promise((resolve, reject) => {
    const options = {
      hostname: "api.privatbank.ua",
      port: 443,
      path: "/p24api/rest_fiz",
      method: "POST",
      headers: {},
    };
    const req = https.request(options, (res) => {
      let data = "";

      // console.log("Status Code:", res.statusCode);

      res.on("data", (chunk) => {
        data += chunk;
      });

      res.on("end", () => {
        resolve(data);
      });
    });

    req.on("error", reject);

    req.write(result);
    req.end();
  });

const run = async () => {
  const xml = await req();
  const json = JSON.parse(convert.xml2json(xml, { compact: true }));
  console.log(JSON.stringify(json, null, 2));
  json.response.data.info.statements.statement
    .map(({ _attributes }) => _attributes)
    .forEach(
      ({
        card,
        appcode,
        trandate,
        trantime,
        amount,
        cardamount,
        rest,
        terminal,
        description,
      }) => {
        const TAB = "  ";
        const tsv = [
          trandate,
          trantime,
          appcode + terminal,
          hideCard(card),
          description,
```

```
          ...splitAmount(cardamount),
          ...splitAmount(amount),
          ...splitAmount(rest),
        ].join(TAB);

        console.log(tsv);
      }
    );
};

run();
```