МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
«____»_____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **«БАКАЛАВР»**

Тема: «Застосунок для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio»

Виконавець: Іван КУРИЛО

Керівник: к.пед.н., доцент Юрій СІНЬКО

Нормоконтролер: к.т.н., доцент Вікторія СИДОРЕНКО

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____Аліна САВЧЕНКО
«____»_____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Курила Івана Юрійовича
(прізвище, ім'я, по батькові здобувача вищої освіти в родовому відмінку)

1. Тема роботи: «Застосунок для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio» затверджена наказом ректора від «05» квітня 2024р. № 517/ст.
2. Термін виконання роботи: з 06 травня 2024 року по 16 червня 2024 року.
3. Вихідні дані до роботи: Застосунок для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio.
4. Зміст пояснювальної записки: 1. Загальна характеристика додатків з контролю та прогнозування фінансових витрат. 2. Обґрунтування вибору засобів проєктування. 3. Практична реалізація проєкту.
5. Перелік обов'язкового ілюстративного матеріалу: 1. Вступ. 2. Аналіз та Огляд. 3. Вибір середовища розробки. 4. Концептуальне проєктування інформаційної системи. 5. Розробка структури бази даних та use case діаграми. 6. Опис файлів даних та інтерфейсу програми. 7. Висновок.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Загальна характеристика додатків з контролю та прогнозування фінансових витрат. Написання 1 розділу, представлення керівнику.	06.05.2024-10.05.2024	
2.	Обґрунтування вибору засобів проєктування. Написання 2 розділу, представлення керівнику.	11.05.2024-16.05.2024	
3.	Практична реалізація проєкту. Написання 3 розділу, представлення керівнику.	17.05.2024-23.05.2024	
5.	Загальне редагування та друк пояснювальної записки.	24.05.2024-26.05.2024	
6.	Проходження нормоконтролю, перепліт пояснювальної записки.	27.05.2024-01.06.2024	
7.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	2.06.2024-07.06.2024	

7. Дата видачі завдання «06» травень 2024р.

Керівник кваліфікаційної роботи _____ Юрій СІНЬКО
(підпис керівника)

Завдання прийняв до виконання _____ Іван КУРИЛО
(підпис здобувача вищої освіти)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної на тему: «Застосунок для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio» містить: 72 сторінки друкованого тексту, 23 рисунка та нараховує 19 джерел використаної інформації.

Об'єкт дослідження: мобільний застосунок.

Предмет дослідження: сукупність необхідних умов, що забезпечують найкращий підхід до створення застосунку для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio.

Мета кваліфікаційної роботи: створення застосунку для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio.

Результати кваліфікаційної роботи: перелічено головні принципи контролю та прогнозування фінансових витрат. Констатовано, що фінансове планування повинно базуватися на реалістичних прогнозах і вирішувати грошові проблеми, встановлюючи чіткі терміни вирішення. Результатом бюджетування мають бути важливі дані, які потрібно регулярно аналізувати для прийняття обґрунтованих рішень.

Проведено аналіз існуючих аналогів з контролю фінансових витрат на ринку. Проведено обґрунтування засобів проєктування та окремо охарактеризувати особливості операційної системи Android 10. Однією з основних переваг платформи Android є її відкритість. Операційна система Android побудована на основі відкритого вихідного коду і знаходиться у вільному поширенні. Це дозволяє розробникам отримати доступ до вихідного коду і зрозуміти, яким чином реалізовані властивості і функції додатків.

Дана характеристика технологій JavaScript, React Native, Redux, Kotlin, Firebase. За допомогою дерева цілей розглянуто функціональність розробки та її перспективність. Зроблено концептуальне проєктування інформаційної системи, розробку структури бази даних та use case діаграми.

Практична частина була виконана у вигляді веб-застосунку для обліку та контролю сімейних витрат, де користувачі можуть відстежувати свої

особисті та спільні витрати і доходи. Зручний та адаптивний інтерфейс полегшує навігацію сайтом для користувачів, які не є досвідченими в Інтернеті.

Тестування додатку охоплювало різноманітні сценарії використання та введення різних типів даних. Перевірялися правильність обробки введених даних, реакція додатку на некоректні введення та загальна стабільність його роботи. У результаті тестування було виявлено кілька незначних помилок і можливостей для вдосконалення, які можуть бути враховані при подальшому розвитку додатку.

Ключові слова: РОЗРОБКА ДОДАТКУ, ANDROID, ДЕРЕВО ЦІЛЕЙ, АРХІТЕКТУРА, ПРОГРАМНИЙ КОД, ФІНАНСОВІ ВИТРАТИ, USE CASE ДІАГРАМА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

БД	–	База даних
ОС	–	Операційна система
СУБД	–	Система управління базами даних
ADT	–	Android Development Tools
PC	–	Персональний комп’ютер

ВСТУП

Станом на 2021 рік майже кожна людина має смартфон, імовірніше за все з операційною системою Android або iOS. Це пояснюється тим, що Google і Apple займають домінуючі позиції на ринку мобільних телефонів. Аналізуючи найчастіше використовувані додатки, можна побачити, що це переважно соціальні мережі або ігри, що призвело до формування окремого сегмента – мобільних додатків [1].

Особливість цього сегмента полягає в тому, що розробка Android-додатків має враховувати специфіку мобільних пристроїв, зокрема інтерфейс, розмір екрану і сенсорне управління. Якщо інтерфейс розрахований на тактильне управління, необхідно збільшувати розміри відображуваних елементів, оскільки занадто малі елементи важко натискати. Через різноманітність версій мобільних пристроїв, кожна з яких потребує певної платформи для роботи з додатками, розробники повинні враховувати багато факторів. Хоча мобільні додатки існували задовго до появи iPhone і Android, нові пристрої та ідеології суттєво змінили ринок.

Актуальність теми. Мобільний додаток призначений для контролю за персональними витратами і надає інформативну візуалізовану аналітику. Ці функції дозволять користувачам оптимізувати свої видатки, раціонально використовувати фінансові ресурси, спростять процес планування бюджету і дозволять заощаджувати. Саме заощадливість, ефективне управління коштами і грамотне планування бюджету і є основними ознаками фінансової грамотності. Тобто додаток надає користувачеві зручний інструмент для роботи з видатками і опосередковано впливає на підвищення фінансової грамотності [10].

Окрім контролю персональних витрат, не менш актуальним є питанням довготривалого збереження документів. Щоб скористатися своїми правами на гарантійне обслуговування, заміну або повернення товару, чи підтвердити факт оплати, споживач має пред’явити розрахунковий документ. Така необхідність змушує зберігати розрахункові документи в продовж тривалого

часу. Спосіб традиційного фізичного збереження документів є трудомістким і не надійним, адже папір на якому друкуються чеки швидко зношується, а пошук необхідного документу займає багато часу. Зважаючи на розповсюдженість мобільних пристрої, зокрема з ОС Android споживачеві можливо запропонувати обробку і структуроване збереження розрахункових документів у цифровому вигляді. Представлене на Ваш розсуд дослідження присвячене розробці застосунку для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio.

Мета і завдання роботи. Створення застосунку для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio. Відповідно до мети було визначено ряд завдань даної роботи:

– перелічити головні принципи контролю та прогнозування фінансових витрат;

– провести аналіз існуючих аналогів на ринку;

– обґрунтувати вибір засобів проєктування, дати характеристику операційної системи Android 10 та середовища розробки;

– реалізувати обґрунтований вибір цільового призначення додатку на основі дерева цілей;

– зробити розробку структури бази даних та use case діаграми.

– провести практичну реалізацію проєкту та провести його тестування.

Об'єкт і предмет дослідження. Об'єкт дослідження цієї роботи є мобільний застосунок. Предметом дослідження є сукупність необхідних умов, що забезпечують найкращий підхід до створення застосунку для контролю та прогнозування фінансових витрат, розроблений за допомогою IDE Android Studio.

Методи дослідження. Теоретичний аналіз наукової літератури за напрямом дослідження включав використання статистичних методів для аналізу літературних даних. Основою дослідження стали методи порівняльного аналізу та класифікації. Завдання роботи вирішувалися за допомогою системного підходу до добору матеріалу, методів індуктивного та

логічного аналізу, спостереження та статистичних методів аналізу літературних даних.

Наукова новизна отриманих результатів. Зроблено широкий літературний пошук з детальним аналізом наукової інформації. Проведено систематизацію та адаптацію отриманих літературних результатів.

Практичне значення отриманих результатів. Практичне значення отриманих результатів полягає у наявності теоретичного матеріалу, ретельно відібраного під час пошуку інформації з теми, а також у систематизації матеріалу за напрямком дослідження. Проведене дослідження має більш глибокий рівень розробки в порівнянні з попередніми роботами вчених, дисертантів та дослідників у цій галузі.

Особистий внесок здобувача. Особистий внесок здобувача включає аналіз літератури, розробку та реалізацію додатку, тестування та оптимізацію додатку.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДОДАТКІВ З КОНТРОЛЮ ТА ПРОГНОЗУВАННЯ ФІНАНСОВИХ ВИТРАТ

1.1. Головні принципи контролю та прогнозування фінансових витрат

Фінансове планування повинно базуватися на реалістичних прогнозах і вирішувати грошові проблеми, встановлюючи чіткі терміни вирішення. Також важливо, що бюджетування це цікаве заняття, яке з часом може стати захоплюючим. Люди, які ведуть облік витрат і доходів декілька місяців, можуть настільки захопитися цим процесом, що забувають про його основну мету. Бюджет не має сам по собі значення, якщо він не вирішує фінансових завдань. Результатом бюджетування мають бути важливі дані, які потрібно регулярно аналізувати для прийняття обґрунтованих рішень [4].

Головними етапами для контролю та прогнозування фінансових витрат є таке:

– Крок 1. Почніть з установлення, де зберігаються ваші гроші. У більшості людей є гаманець, куди регулярно або час від часу потрапляють гроші. Тому перший запис у списку місць зберігання грошей буде «готівка». Деякі можуть на цьому закінчити, а інші продовжать, вказавши, наприклад, «депозит в банку», зарплатну картку тощо. У програмах обліку особистих фінансів це називається «створенням рахунків». Іноді гроші можуть бути на кредитних картах, тому важливо включити їх також у ваш список рахунків, зазначивши, що баланс буде негативним. Також можна врахувати боргові рахунки.

Під час запису рахунків корисно вказувати, в якій валюті вони ведуться.

Наприклад, у одному банку може бути кілька вкладів – гривневий і доларовий, тому ви можете записати кожен окремо та вказати валюту в назві.

– Крок 2. Створіть власний перелік категорій доходів і витрат. Якщо рахунки допомагають розуміти, де знаходяться гроші, то категорії дозволяють зрозуміти, на що саме ви витрачаєте свої гроші і звідки вони надходять. Багато програм пропонують готові категорії, але найкраще створити свій перелік, оскільки кожна ситуація унікальна. Чужий перелік можна використовувати як основу, додавати, видаляти або змінювати категорії. Типові категорії:

Доходи: Зарплата; Банківські вклади; Стипендія; Пенсія.

Витрати: Автомобіль (ремонт, обслуговування, мийка); Благодійність; Банківське обслуговування; Гігієна (перукарні, косметика, засоби особистої гігієни); Квартира (електрика, газ, вода); Медицина; Непередбачені витрати; Громадське харчування (їжа поза домом); Одяг, взуття, аксесуари; Подарунки; Поїздки (відпустка, вихідні); Допомога родичам; Продукти; Розваги (ресторани, кіно); Зв’язок (інтернет, мобільні телефони); Спорт; Транспорт (бензин, громадський транспорт).

– Крок 3. Розпочніть вести облік доходів і витрат. Кожна транзакція (дія з грошима, отримання чи витрати) повинна мати відповідний рахунок, суму, категорію витрат чи доходу, а також можливі примітки для подальшого аналізу.

1.2. Аналіз існуючих аналогів на ринку

Деякі люди почали використовувати програми для особистої бухгалтерії ще до того, як з’явилися сучасні мобільні пристрої. І зараз на ринку є дуже прості додатки, в які можна вводити свої доходи і витрати, а також сортувати їх за категоріями. Але є й більш складні фінансові додатки з багатшим функціоналом. Вони дозволяють користувачам відслідковувати курс акцій, вести облік витрат за складними схемами, стежити за інвестиціями тощо. Ці додатки надають докладні звіти про витрати на різні аспекти [2].

Проте такі великі інструменти мають свої недоліки: не завжди є безкоштовними; часто спрямовані на досить досвідчених користувачів; не завжди зручні у використанні на мобільних платформах.

Для огляду та аналізу таких додатків корисно визначити їх категорію. Наприклад, категорія «Фінанси» може включати додатки для бухгалтерського обліку та управління фінансами.

Наприклад, додаток Andromoney є одним з лідерів за кількістю завантажень у Google Play з високою оцінкою. Він має простий і зручний інтерфейс, який дозволяє вести облік фінансових операцій, переглядати звіти і керувати бюджетом. Такі додатки надають користувачам широкі можливості у веденні особистої бухгалтерії та фінансового планування [13].

AndroMoney є додатком, який відображає баланс грошових коштів і дозволяє здійснювати транзакції між рахунками без зайвих ускладнень. Ви легко можете додавати фінансові операції, а також створювати бюджети на різні періоди часу – рік, місяць, тиждень або навіть день (рис. 1.1. а). Користувач може встановлювати бюджет для окремих категорій та отримувати попередження, якщо ви перевищите місячний бюджет. Налаштувавши основну валюту, ви прив’яжете всі грошові операції до неї.

Однією з важливих особливостей AndroMoney є можливість експорту та імпорту баз даних, в тому числі через Dropbox і Google Drive. Проте додаток має деякі недоліки, такі як відсутність української мови, наявність рекламних банерів (для їх видалення потрібно платити), застарілий інтерфейс, який може бути не дуже зручним, та надмірна функціональність, яка може бути зайвою для деяких користувачів.

Додаток «Expense Manager» є досить простим інструментом для планування фінансів, хоча має велику кількість завантажень (приблизно один мільйон) (рис. 1.1. б). В ньому розробники вже включили кілька основних пунктів витрат, але користувачі можуть змінити їх у налаштуваннях. Також в додатку можна налаштувати повторювані платежі та переглядати графік витрат за різними періодами часу.

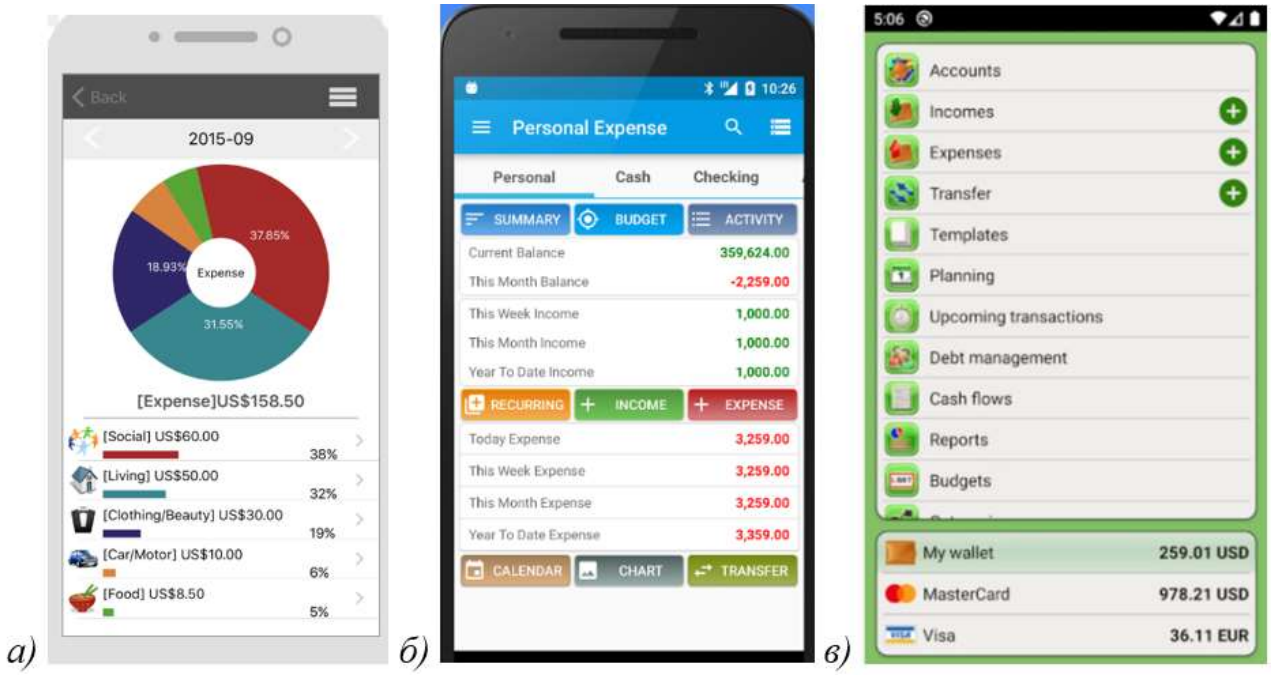


Рис. 1.1. Робоче меню додатків AndroMoney – а), Expense Manager – б) та Finance PM – Expense Manager – в)

На жаль, для отримання повної версії програми доведеться платити. Додаток має деякі недоліки, зокрема, не дуже зручний інтерфейс і відсутність перекладу на українську мову.

«Витрати» – це додаток, який дозволяє вести облік сімейних витрат за різними категоріями і тегами. Користувач може створювати і змінювати категорії витрат, а також переглядати статистику витрат за різними періодами (день, тиждень, місяць, рік). Кожна категорія має повну історію витрат з можливістю внесення змін [18].

Серед мінусів програми «Витрати»: рахує лише витрати; не має українського перекладу; застарілий інтерфейс; не зручний в користуванні; розробники вже більше року не оновлюють додаток; немає пункту «заощадження» дуже дрібний шрифт в пункті «статистика» немає графіків.

Finance PM – Expense Manager (рис. 1.1в). Окрім обліку руху грошових коштів, програма «Finance PM» пропонує управління боргами, бюджетом та майбутніми транзакціями. Одне з переваг цієї програми – відсутність обмежень, що дозволяє користувачеві створювати необмежену кількість

гаманців з різною валютою або задавати ліміти (бюджет) не тільки на певний період часу, але й на конкретні категорії витрат.

«Finance PM» дозволяє планувати операції щодо витрат, доходів або переказів грошей на певну дату з можливістю щоденних, щотижневих, щомісячних повторень, а також в конкретні дні тижня чи місяця. Очікувані транзакції можна переглянути в розділі «Найближчі операції», а всі попередні – у розділі «Грошові переміщення».

Ще однією цікавою особливістю є можливість створення шаблонів для різних типів фінансових операцій, що дозволяє значно економити час при регулярних доходах, витратах або переказах з однаковою сумою. Також у програмі можна налаштовувати зовнішній вигляд і елементи інтерфейсу на головному вікні, що додає ще більше функціональності та зручності користувачеві [9].

Toshl Finance – це додаток, який надає кожному календарному дню власну сторінку з фінансовими операціями, а його помічником виступає миленький монстр, що надає додатку інтерактивності і мотивує користувача.

Для початку користування додатком потрібно зареєструватися, вибрати основну валюту та здійснити синхронізацію з сервером. На відміну від багатьох інших додатків, Toshl Finance пропонує лише базовий набір функцій, таких як витрати, доходи, статистика і бюджет. Замість стандартних категорій витрат і доходів використовуються мітки. Цей додаток є платним, з підпискою $1,99 на місяць або $19,99 на рік, що дозволяє додавати необмежену кількість доходів і бюджетів, експортувати звіти в PDF, Excel і Google Docs. Проте він не має української мови і має застарілий інтерфейс.

Загалом, дизайн Toshl Finance може здаватися дитячим, що може не сподобатися серйозним користувачам, таким як бізнесмени або інвестори (рис. 1.2а). Такі додатки фінансової категорії мають бути більш серйозними і функціональними для задоволення потреб професіоналів.

Monefy: Money Tracker [8] – міжнародний онлайн-додаток, який дозволяє створювати транзакції, акаунти та переглядати історію транзакцій, доданих користувачем у минулому (рис. 1.2б). Система може генерувати

великі обсяги даних, що дає змогу бухгалтерам та податковим службам ефективно працювати з ними. Додаток доступний лише на мобільних пристроях, без веб-версії. На стартовому екрані зверху розташовані кнопка меню та кнопка налаштувань. Нижче відображається інформація про транзакції за поточний місяць у вигляді діаграми, а ще нижче – поточний баланс користувача [6].

У головному блоці є кнопки для переходу до екранів створення транзакцій та можливість переглядати транзакції за категоріями. Додаток також дозволяє отримувати файл з усіма вашими транзакціями. Користувачі можуть створити профіль та налаштувати синхронізацію даних між пристроями.

Основні переваги: дані зберігаються тільки локально і не передаються на сторонні сервіси; можливість експортувати файл з транзакціями. Основні недоліки: додаток доступний лише на мобільних пристроях; відсутня синхронізація з банками.

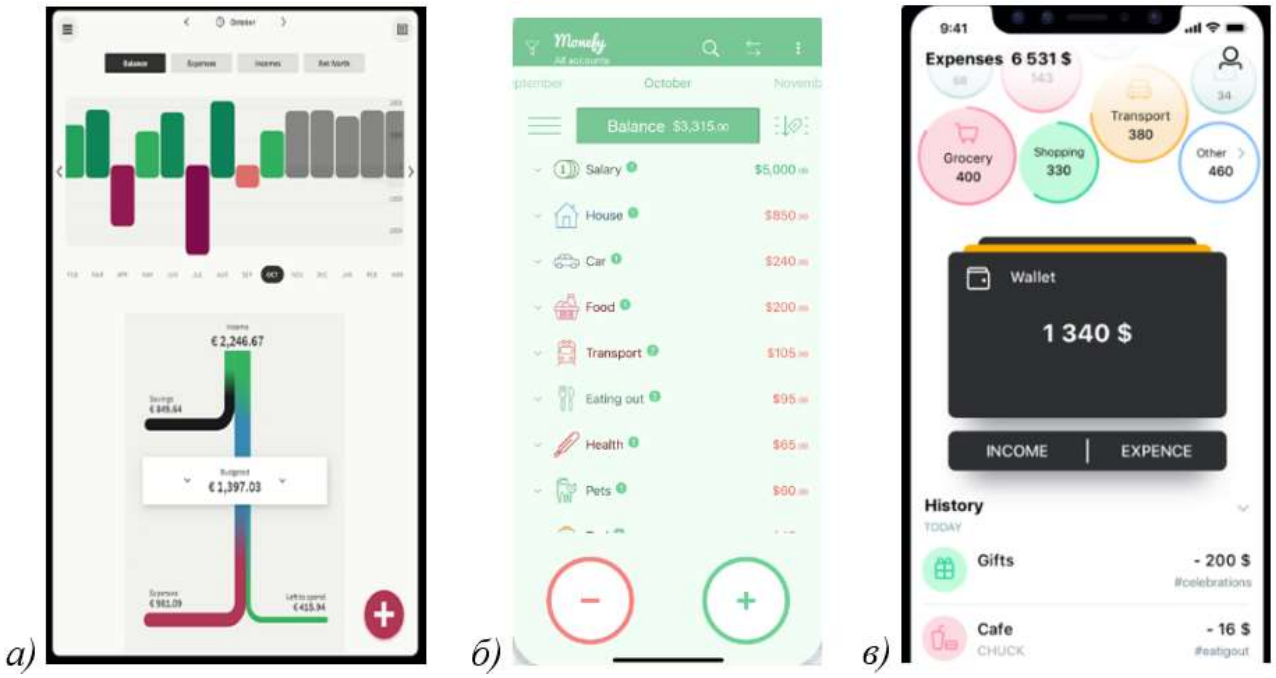


Рис. 1.2. Робоче меню додатків Toshl Finance – а), Monefy: Money Tracker – б) та CoinKeeper: budget planner – в)

CoinKeeper: budget planner – мобільний застосунок, орієнтований на ринок США, який допомагає відстежувати та аналізувати витрати користувачів (рис. 1.2в). Застосунок має привабливий інтерфейс і добре

реалізовану систему відстеження фінансових потоків користувача, а також дозволяє встановлювати фінансові цілі [7].

На головному екрані відображається інформація про доступні акаунти, категорії та звітність. Додаток відзначається гарним дизайном, але не підтримує підключення банків для синхронізації транзакцій. У застосунку є можливість створювати транзакції та відстежувати витрати за окремими акаунтами (див. рис. 1.2в).

Основні переваги:

- гарний дизайн аналітичних елементів;
- підтримка усіх відомих валют.

Основні недоліки:

- доступний лише як мобільний застосунок;
- немає можливості під’єднання банка для синхронізації;
- застосунок спеціалізується на аналітиці витрат користувача.

На основі проведеного огляду існуючих програмних застосунків можна зробити висновок, що розробка мобільного застосунку для відстеження витрат користувачів є актуальним завданням

Splitwise. Найпопулярніший додаток, доступний як у веб-версії, так і в Google Play та App Store. За даними, представленими на офіційному сайті [12], Splitwise дозволяє створювати групи та додавати друзів, розподіляти витрати та борги між членами групи, розподіляти транзакції на відсотки між учасниками, обчислювати загальні залишки, пропонувати можливі витрати, спрощувати борги, додавати повторювані витрати та працювати в офлайн-режимі.

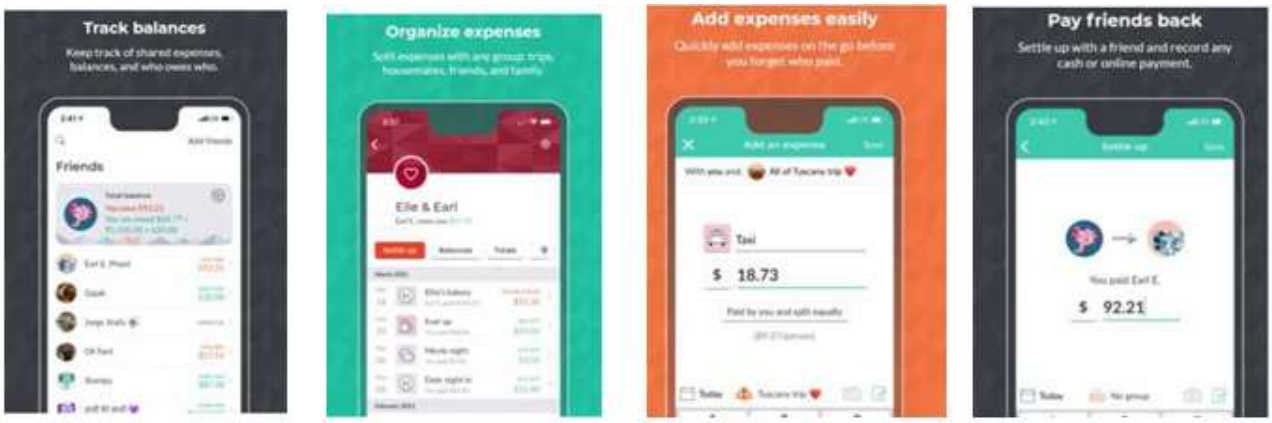


Рис. 1.3. Робоче меню досліджуваного додатку та його функціонал

Splid. Це досить популярний сервіс, який можна використовувати через браузер або завантажити з Google Play або App Store. За даними, представленими на офіційному веб-сайті додатку [4], його головним гаслом є "An end to the chaos" - "Кінець хаосу". На жаль, єдину мову, яку підтримує додаток - англійська, але є цікава можливість експорту транзакцій у форматах PDF або Excel.

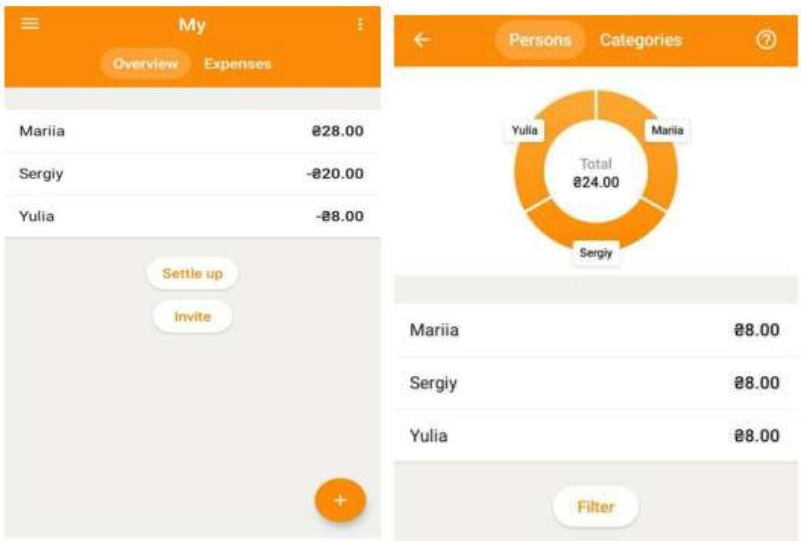


Рис. 1.4. Головна сторінка додатку Splid та графічне зображення витрат користувача

Splid надзвичайно простий у використанні та зрозумілий для користувачів. Представлені можливості включають створення груп та додавання учасників, реєстрацію доходів/витрат, які можна поділити між членами групи, а також перегляд часток витрат у вигляді діаграми.

CoinKeeper - ще один інструмент для контролю за витратами. У відміну від попередніх, CoinKeeper [5] забезпечує зв'язок із вашою банківською карткою. Серед цікавих можливостей можна виділити такі: можливість входу з різних пристроїв, наявність розумного алгоритму, що дозволяє прогнозувати витрати, можливість встановлення ліміту на витрати та зручний аналіз динаміки витрат за допомогою статистики.

Рис. 1.5. Головна сторінка додатку CoinKeeper

Money manager - це практичний інструмент, який сприяє ефективному управлінню особистими фінансами. Він дозволяє вам контролювати власні доходи та витрати, надаючи можливість перегляду статистики за різними періодами: день, тиждень, місяць, рік або певний проміжок часу.

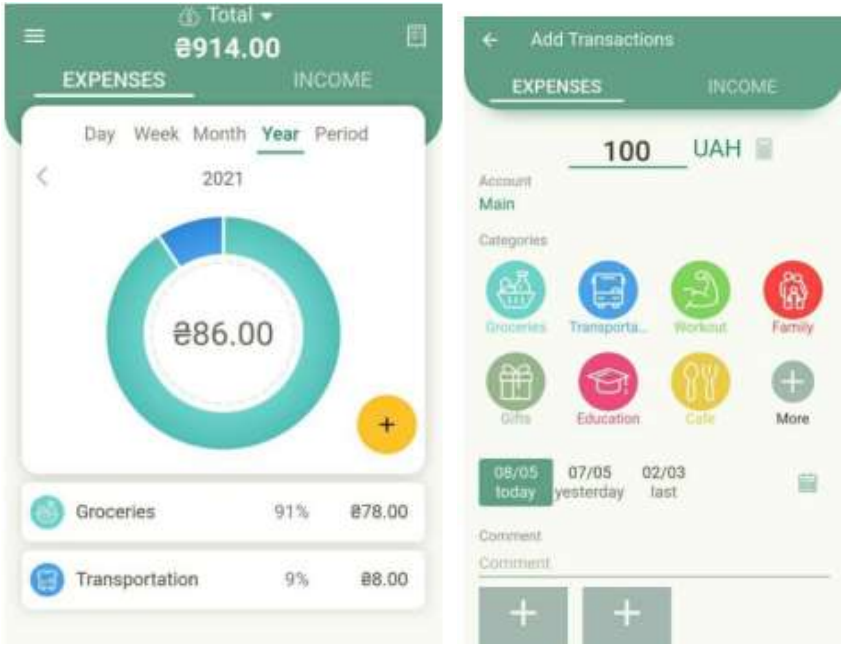


Рис. 1.6. Графічне зображення витрат користувача в додатку Money manager

ВИСНОВКИ ДО РОЗДІЛУ 1

В розділі проведена загальна характеристика додатків з контролю та прогнозування фінансових витрат та проаналізовано головні принципи контролю та прогнозування фінансових витрат. Перелічено та охарактеризовано головні етапи для контролю та прогнозування фінансових витрат. Наголошено на тому, що деякі витрати можуть бути складно класифікувати, а деякі відбуваються рідко чи в невеликих розмірах, тому їх можна внести в категорію «непередбачені витрати». Також сюди можуть включатися помилки у веденні обліку, якщо такі виникають.

Виконано аналіз існуючих розробок в цій області та названо їх поширення, обмеження використання.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ ПРОЄКТУВАННЯ

2.1. Характеристика операційної системи Android 10 та її ключові особливості

Перші мобільні телефони на базі операційної системи Android з'явилися на ринку у жовтні 2008 року. За даними звіту Netmarketshare станом на кінець першого кварталу 2019 року, Android володів 65% глобального ринку смартфонів, у порівнянні з 15,3% у Apple, 2,5% у Microsoft. Конкуренція серед розробників мобільних платформ та сервісів призвела до швидкого впровадження інновацій та стрімкого зниження цін. Різна апаратна і програмна база пристроїв Android стимулює інновації в співтоваристві Android.

Однією з основних переваг платформи Android є її відкритість. Операційна система Android побудована на відкритому вихідному коді, що дозволяє розробникам мати доступ до вихідного коду, розуміти його функціональність та внести свої внески в розвиток. Будь-хто може прийняти участь у вдосконаленні системи, подаючи звіти про помилки або беручи участь у спільноті Open Source Project. Крім того, інтернет заповнений різноманітними програмами Android з відкритим вихідним кодом, розробленими Google та іншими компаніями [3].

Початково у нашому дослідженні ми розглянемо, чому було обрано платформу Android для розробки мобільних додатків. ОС Android є простою для вивчення та надає достатні можливості для створення додатків, що дозволяє навіть новачкам конкурувати з досвідченими програмістами.

Причин для цього кілька [14]:

– як було вказано раніше, згідно з даними глобального сервісу моніторингу Netmarketshare на травень 2019 року, пристрої на базі цієї операційної системи займають 65% ринку;

– більш дружня політика стосовно розробників робить ринок мобільних додатків для Android менш статичним і, отже, більш схильним до інновацій;

– існує широкий спектр напрямків, у яких можна розробляти мобільні додатки для Android;

– одна й та ж програма Java може працювати без будь-яких змін на різних комп'ютерах, таких як ПК, Apple, чи інші платформи. Аналогічно, мобільний додаток може бути розроблений як для смартфону, так і для планшета, Android Wear, Android TV, Android Auto, а навіть Google Glass. Фактично, програми, написані на Java, не мають уявлення, на якому пристрої вони працюють, оскільки вони виконуються всередині віртуальної машини JVM (Java Virtual Machine);

– Java дозволяє створювати програмні елементи (класи), які відображають об'єкти з реального світу. JavaScript - це мова програмування, яка дозволяє створити динамічно оновлюваний контент, управляти мультимедіа та анімацією зображень, взагалі, реалізувати будь-які завдання [17].

Отже, Android - це операційна система, яка працює на різноманітних пристроях, таких як смартфони, планшетні комп'ютери, електронні книги, цифрові програвачі, "розумні" наручні годинники, ігрові приставки, нетбуки, смартбуки, окуляри Google, телевізори, системи автоматичного керування автомобілем та інші пристрої.

ОС базується на ядрі Linux і має власну реалізацію віртуальної машини Java від Google. Спочатку розроблялася компанією Android Inc., яку в 2005 році купила Google. Пізніше Google створила альянс Open Handset Alliance (OHA), який зараз займається підтримкою та подальшим розвитком платформи.

Android дозволяє створювати Java-додатки, що керують пристроєм через розроблені Google бібліотеки. Android Native Development Kit дозволяє портувати (але не налагоджувати) бібліотеки та компоненти додатків, написані на C та інших мовах.

Широка гнучкість Android зумовлена тим, що ця система побудована на ядрі Linux з відкритим програмним кодом, що надає безліч можливостей для розробників. Android може працювати на пристроях з обсягом оперативної пам'яті менше 256 Мб, а найновіші версії системи вимагають 512 Мб, що є невеликим для сучасних пристроїв. Система не потребує наявності потужного процесора і може працювати на пристроях з ядром частотою 600 МГц.

Операційна система надає можливість установки додатків з офіційного репозиторію Google, який містить найбільшу в світі базу програм. Кожен розробник може написати програму для пристрою і розмістити її в магазині. Також можливо встановлювати додатки безпосередньо з пристрою або через комп'ютер, завантажуючи файл .apk.

Android відрізняється своєю інтеграцією з послугами Google, такими як Gmail, Hangouts, Voice Search і багатьма іншими. На Android також офіційно підтримується браузер Chrome, що дозволяє синхронізувати вкладки, які відкриті на смартфоні, з комп'ютерним браузером.

Наприклад, ви можете розпочати перегляд сторінок зі свого телефону і, за бажанням, продовжити вивчати інформацію, відкривши ті самі вкладки на комп'ютері, без потреби у повторному пошуку.

Всі необхідні програми цієї операційної системи розташовані одночасно на головному екрані та в меню пристрою, яке можна викликати натисканням на центральну сенсорну клавішу або відповідну кнопку на екрані. Усі налаштування знаходяться в розділі "Налаштування", а кожен крок користувача пояснюється коментарями і підказками під час першого запуску пристрою. Операційна система дуже швидко реагує на дії користувача, здійснюючи встановлення та завантаження необхідних програм і файлів з такою самою швидкістю, як і інші сучасні мобільні ОС.

Також можна розробляти додатки у середовищі Eclipse або IntelliJ IDEA, використовуючи плагіни, такі як Android Development Tools (ADT). Для цього потрібна версія JDK 5.0 або вище. З точки зору програміста, Android-платформа абстрагує розробника від ядра і дозволяє створювати код на Java. Android має кілька корисних можливостей [15]:

– По-перше, це фреймворк, який пропонує великий набір API для створення різних типів додатків і забезпечує можливості повторного використання і заміни компонентів, які пропонуються платформою і сторонніми додатками;

– По-друге, наявність віртуальної машини Dalvik, що відповідає за запуск додатків на Android. Для роботи над додатками доступні різноманітні бібліотеки, серед яких:

– Bionic (бібліотека стандартних функцій, несумісна з glibc);

– мультимедійні бібліотеки на базі PacketVideo OpenCORE (підтримують такі формати, як MPEG–4, H. 264, MP3, AAC, AMR, JPEG і PNG);

– SGL (двигун двомірної графіки);

– OpenGL ES 1.0 ES 2.0 (двигун тривимірної графіки);

– Surface Manager (забезпечує для додатків доступ до 2D / 3D);

– WebKit (готовий двигун для веб-браузера;

– обробляє HTML, JavaScript);

– FreeType (двигун обробки шрифтів);

– SQLite (легка СУБД, доступна для всіх додатків);

– SSL (протокол, що забезпечує безпечну передачу даних по мережі).

У порівнянні з традиційними додатками Linux, додатки для Android підпорядковуються додатковим правилам: Content Providers забезпечують обмін даними між додатками; Resource Manager дає доступ до ресурсів, таких як файли XML, PNG, JPEG; Notification Manager надає доступ до рядка стану; Activity Manager відповідає за управління активними додатками.

Google надає безкоштовний інструментарій для розробки (Software Development Kit), який працює на x64-машин під операційними системами Linux, Mac OS X (10.4.8 або новіше), Windows 7, Windows 8, а також Windows 8.1 і Windows X. Для розробки потрібен JDK 5 або більш пізні версії.

Для створення додатків для Android можна використовувати мову програмування Java (не нижче версії 1.5). Існує плагін для Eclipse - Android Development Tools (ADT), призначений для версій Eclipse з 3.3 до 3.7. Також доступний плагін для IntelliJ IDEA, який спрощує розробку Android-додатків, і плагін для середовища розробки NetBeans IDE, який, починаючи з версії NetBeans 7.0, не є вже експериментальним, хоча ще не є офіційно підтримуваним. Крім цього, існує Motodev Studio for Android - інтегроване середовище розробки на базі Eclipse, яке дозволяє працювати безпосередньо з Google SDK [16].

2.2. Вибір середовища розробки

Наступним етапом розробки мобільного додатку є вибір середовища розробки. Найбільш популярними платформами для розробки є офіційне середовище Android Studio і Xamarin.

Особливості та переваги Android Studio: Android Studio – це офіційне середовище розробки (IDE) від Google для створення додатків на ОС Android. Основні перевагами Android Studio:

- Створене розробниками ОС Android;
- Гнучка система збирання;
- Потужний редактора коду, який максимально спрощує процес розробки;
- Багатофункціональний і швидкий емулятор;
- Єдине середовище для розробки для різних пристроїв на ОС Android;
- Застосування зміни для push-коду та змін ресурсів у працюючій

програмі відбуваються без перезапуску програми;

- Можливість використання шаблонів та інтеграція з GitHub;
- Широкий вибір інструментів та фреймворків для тестування;
- Інструменти Lint для діагностики проблем з сумісністю версій, продуктивністю, зручністю використання, тощо;
- Об’єктно-орієнтовані мови програмування Java, Kotlin і C++;
- Підтримка Google Cloud Platform, що надає можливості зручної інтеграції Google Cloud Messaging та App Engine;
- Можливості роботи з UI компонентами за допомогою Drag-and-Drop;
- Генерація декількох. apk файлів та різноманітні види збірок;
- Утиліта ProGuard для підписки додатків;
- Структурована і вичерпна документація;
- Підтримка розробки додатків для Android TV і Android Wear [19].

Для реалізації функціональних вимог додатка були залучені і сторонні бібліотеки, які знаходяться у відкритому доступі на сайті GitHub. Перелік цих бібліотек і їх короткий опис:

- Parceler – це бібліотека, що використовується для генерації коду Android Parcelable. Parcelable – це інтерфейс, який дозволяє передавати складні об’єкти між Activity. Посилання на репозиторій: https://github.com/johncarl81/parceler;
- google-gson() – це бібліотека, що використовується для перетворення об’єктів Java в JSON- представлення. Надає можливості використовувати для перетворення строки JSON в об’єкт Java. Посилання на репозиторій: https://github.com/google/gson;
- Stetho – це бібліотека від Фейсбук, що надає можливості швидкого налагодження додатку. Основними перевагами даної бібліотеки є можливість

відстеження мережевої активності і простого та зручного перегляду ієрархії додатку, що реалізується за допомогою інструменту Chrome DevTools. Посилання на бібліотеку: http://facebook.github.io/stetho ;

- Butter Knife – це бібліотека, що використовується для ініціалізації візуальних об’єктів, таких як ImageView, TextView, EditText, тощо. Основними перевагами даної бібліотеки над вбудованою є значне спрощення роботи з елементами, що значною мірою пришвидшує темп розробки. Посилання на бібліотеку: https://github.com/JakeWharton/butterknife;

Робота даних бібліотек не розповсюджується на модуль графічного редактора і використовується лише для Дані бібліотеки були підключені в модулі додатку і не розповсюджуються. Модуль графічного редактора використовує тільки вмонтовані бібліотеки від компанії Google котрі постачаються разом з SDK

Перечисленні бібліотеки були підключені лише в модулі додатку і не впливають на роботу модуля графічного редактора. Для цього модуля використовуються лише вбудовані бібліотеки від компанії Google, що постачаються разом з SDK.

Наступним етапом дослідження методів розробки є вибір версії Android SDK на якій буде проводитись розробка додатка. Android SDK – це пакет розширень Android, який надає додаткові, зручні інструменти розробки. За допомогою цього пакету можна відслідковувати стан операційної системи, читати лог-файли, створювати віртуальні пристрої для тестування роботи додатка на різних версіях ОС і багато іншого. Від версії SDK залежить і версія операційної системи мобільного додатка. Для розробки було обрано версію SDK – 29. Операційна система пристроїв для роботи з додатком має бути версії 10 і вище. За даними API version distribution, що надає Android Studio, кількість пристроїв, які працюють на операційній системі версії 10 і вище – 62,8%.

Середовище Android Studio підтримує такі мови програмування: Java, Kotlin, C++.

2.3. Характеристика технологій JavaScript, React Native, Redux, Kotlin, Firebase

JavaScript мова програмування, яка дозволяє створити динамічно оновлюваний контент, управляє мультимедіа, анімує зображення, втім, робить все, що завгодно. Ядро мови JavaScript складається з певної кількості звичайних можливостей, які дозволяють робити наступне:

- Зберігати дані всередині змінних. Записаний код потрібно зберігали під іменем в змінній name;
- Операції над фрагментами текстів (відомі в програмуванні як «рядки»);
- Запускати код відповідно до певних подій відбуваються на web сторінці. Більш важливим є її функціональність, яка може бути створена поверх основної мови JavaScript. Так звані інтерфейси прикладного програмування (API) надають додаткові можливості для використання у коді JavaScript.

Розглянемо ключові переваги JavaScript:

– Ефективність для кінцевого користувача: JavaScript, який використовується на клієнтській стороні, не потребує підтримки веб-сервера. Це означає, що він не вимагає компіляції на стороні клієнта, що дає йому перевагу у швидкості. Оскільки JavaScript виконується на комп'ютері користувача, результати можуть бути отримані майже миттєво, знижуючи навантаження на сервер. Наприклад, можна перевірити користувацький ввід перед відправленням запиту на сервер.

– Простота: JavaScript є відносно простою мовою для вивчення та реалізації. Вона використовує модель DOM, яка має безліч вбудованих функцій для різних об'єктів на сторінках, що робить її легкою для написання сценаріїв для досягнення мети користувача.

– Універсальність: JavaScript працює чудово з іншими мовами і може використовуватися в різних додатках. В даний час існує безліч способів використання JavaScript, включаючи використання на сервері з допомогою

Node.js. Наприклад, якщо ви використовуєте Node.js з Express, ви можете використовувати базу даних документів, таку як MongoDB.

1. React Native дозволяє створювати мобільні додатки, використовуючи лише JavaScript і маючи таку ж структуру, як у React. Це дозволяє складати багатофункціональний мобільний інтерфейс за допомогою декларативних компонентів. Додатки, розроблені з використанням React Native, не є мобільними веб-додатками, оскільки вони використовують ті ж компоненти, що й звичайні програми для iOS і Android. Замість того, щоб використовувати мови Swift, Kotlin або Java, можна створювати ці компоненти за допомогою JavaScript і React.

Переваги React Native включають такі можливості:

– Платформова незалежність. React Native дозволяє створювати додатки для різних платформ, таких як Android і iOS, використовуючи тільки JavaScript. Це дозволяє розробникам створювати мобільні додатки з єдиною кодовою базою.

– Легкість освоєння. Оскільки React Native базується на React і JavaScript, вивчення цього фреймворку відкриває можливості для розробки не лише мобільних додатків, але й веб-додатків.

– Час збірки з React Native значно скорочено порівняно з Android Studio. Якщо раніше збірка займала 2-3 хвилини, то зараз, завдяки функції "гарячого перезавантаження" (Hot Reloading), тестування користувацького інтерфейсу стає набагато простішим. Ця функція дозволяє автоматично перезавантажувати додаток кожного разу, коли JS-файл зберігається.

– JavaScript в React Native дуже зручний для передачі даних по мережі. Здійснення виклику API, рендеринг зображень за URL та інші процеси стають простими. Ви більше не маєте потреби використовувати такі інструменти як Retrofit, OkHttp, Picasso і т. д., що дозволяє значно скоротити час, який витрачається на налаштування. Наприклад, коли дані надходять з API на платформі Android, вони спочатку перетворюються в POJO-модель, а потім використовуються в елементах інтерфейсу користувача (UI). Проте дані JSON, отримані в React Native, зручні для JavaScript і можуть безпосередньо

використовуватися для попереднього перегляду інтерфейсу користувача (UI). Це значно спрощує розробку веб-інтерфейсу для GET або POST-запитів від REST API.

– Розробка UI. Використання модуля flexbox для розмітки UI в React Native дозволяє створювати гнучкі та адаптивні дизайни, але це також може вимагати більшого обсягу коду порівняно з нативною розробкою.

Недоліки React Native включають:

– Непопулярність JavaScript. Деякі розробники не приймають JavaScript через його відмінності від традиційних мов програмування, таких як Java чи C++.

– Обмеженість сторонніх бібліотек. Спільнота React Native ще не настільки розвинена, щоб підтримувати так багато сторонніх бібліотек, як у нативній розробці для Android.

2. Redux - це інструмент управління станом для JavaScript-додатків. В основному він використовується разом з React, але може бути застосований і в інших контекстах. Хоча React має свої вбудовані можливості управління станом, вони не завжди ефективні для складних додатків. У таких випадках краще використовувати Redux, який дозволяє зручно керувати станом за допомогою зовнішнього сховища даних, що полегшує розробку і підтримку програмного забезпечення.

Redux - це метод управління станом програми, що ґрунтується на декількох ключових концепціях. Засвоївши ці концепції, можна ефективно вирішувати проблеми, пов'язані зі станом додатка. Одним з викликів у розумінні праці з Redux є розуміння різноманітних термінів, таких як редюсери, селектори та ефекти. Для кращого осмислення цих понять можна розглянути розширений цикл Flux.

Redux ідеально підходить для середніх і великих додатків, особливо у тих випадках, коли необхідно ефективно управляти станом програми, і стандартні менеджери стану в React або інших бібліотеках не вистачають. У Redux загальний стан програми представлений одним об’єктом JavaScript-state

або state tree. Це незмінне дерево станів доступне тільки для читання, і його можна змінювати лише через відправку action (дій).

Основна перевага Redux полягає в управлінні як станом даних, так і станом інтерфейсу. Redux є основним джерелом істини під час розробки, оскільки спрощує пошук актуальної інформації. При роботі з Redux є безліч способів та підходів.

Серед недоліків Redux можна виділити таке:

– Повна залежність від порядку виклику. Наприклад, якщо поміняти місцями withUsers і withPresents, то HOC не зможе впоратися із завданням, оскільки withPresents не знайде списку користувачів, що може бути обов'язковим параметром. Крім того, HOC може сам змінювати дані. І коли ми зіткнемося з такою проблемою, нам потрібно буде спочатку зрозуміти, що у нас з цим є проблема, а в більшості випадків це може бути складно. Для її вирішення нам потрібно або писати новий HOC (такий же як вихідний з деякими змінами), або правити вже існуючий, що може зламати логіку в справно працюючих місцях.

– Actions можна розглядати як складні переходи між станами, проте вони в основному лише встановлюють одне значення. У додатках, побудованих на Redux, часто накопичується велика кількість таких простих дій, що схоже на роботу з функціями сетерами вручну у Java.

Перенесення фрагментів стану з компонентів до сховища Redux часто видається простою додатковою дією без належного рівня абстракції, навіть якщо той самий фрагмент стану можна було б використовувати по всьому додатку, але насправді він відноситься до конкретної частини інтерфейсу. Функції-редюсери можуть бути вражаючо ефективними вирішувати складні завдання метапрограмування, але в більшості випадків вони обмежуються примітивною обробкою action відповідно до його типу. Це не проблема для мов програмування, таких як Elm або Erlang, які володіють лаконічним і виразним синтаксисом pattern matching, однак у Javascript вам доведеться мати справу з громіздкими конструкціями switch. Крім того, стан компонентів у React здебільшого неповороткий для роботи з наскрізною функціональністю,

яка зачіпає багато модулів програми, наприклад, інформація про користувача або оповіщення. Саме для цього в Redux існує дерево станів, яке не залежить від конкретного інтерфейсу користувача. Крім того, при обробці стану поза інтерфейсом легше підтримувати сталість, оскільки серіалізація в localStorage або URL відбувається в єдиному місці.

3. Firebase сприяє швидкому розвитку високоякісних додатків, збільшенню кількості користувачів і збільшенню доходів. Платформа містить різноманітні корисні функції для вашого додатка, такі як серверний код для мобільних сервісів, статистика, а також інструменти для монетизації і розширення аудиторії.

Основними перевагами Firebase є:

– Швидкість розробки. Firebase має інтуїтивно зрозумілі API, які допомагають швидко створювати високоякісні додатки. У нього також є інструменти для розширення користувацької бази і збільшення доходів, що дозволяє вам вибрати відповідні рішення для ваших потреб;

– Готова інфраструктура. Firebase забезпечує готову інфраструктуру, що усуває необхідність у складних налаштуваннях та панелях управління, дозволяючи зосередитися на потребах користувачів;

– Статистика. Firebase використовує безкоштовний аналітичний інструмент Google Analytics, спеціально розроблений для мобільних пристроїв. Це дозволяє отримувати дані про дії користувачів і швидко реагувати за допомогою додаткових функцій;

– Кросплатформенність. Firebase працює на різних платформах завдяки пакетам розробника для Android, iOS, JavaScript і C++. Ви також можете використовувати Firebase з серверними бібліотеками або REST API;

– Масштабованість. Якщо ваш додаток стає популярним і збільшується навантаження, Firebase автоматично масштабується без необхідності зміни коду сервера або залучення додаткових ресурсів.

2.4. Вибір цільового призначення додатку на основі дерева цілей

Згідно основної мети даного дослідження є робота з створення андроїд додатку для контролю та прогнозування фінансових витрат. Досягти поставлену мету можна за допомогою реалізації таких цілей: аналіз предметної області, проєктування системи, реалізація програмної частини, удосконалення системи, підтримка продукту. Описане дерево цілей для даної роботи можна переглянути на рис. 2.1 у такому вигляді.

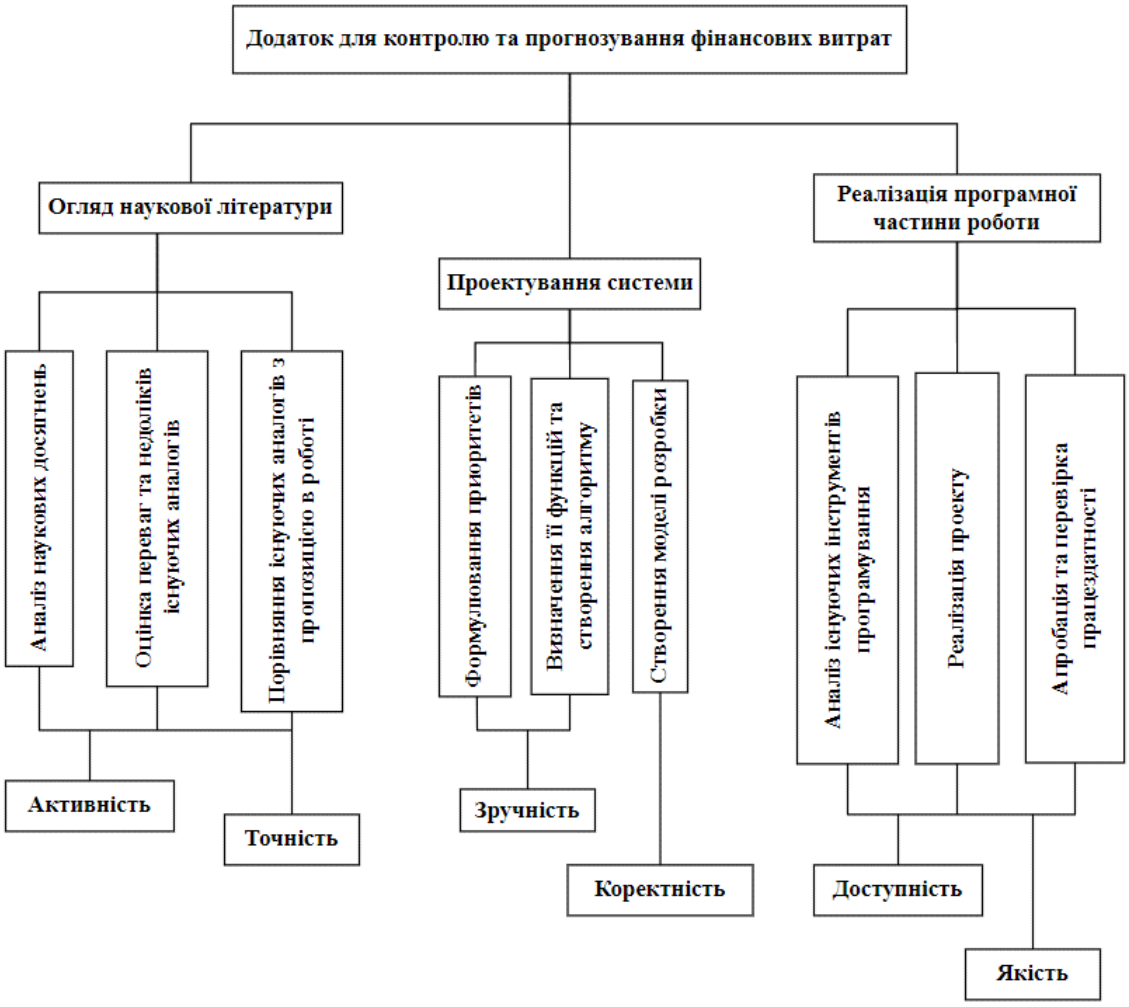


Рис. 2.1. Зовнішній вигляд дерева цілей згідно досліджуваної проблематики в роботі

Наступними кроком при системному аналізі об’єкту дослідження є створення матриці порівняння критеріїв з типами систем (таб. 2.1). Нормалізацією елементів стовпця та знаходженням середнього значення рядка визначено вектор пріоритетів.

Таблиця 2.1

Ранжування критеріїв в дереві цілей при створенні андроїд додатку для контролю та прогнозування фінансових витрат

№	Назва критерію	Порівняння критеріїв						Вектор пріоритету
		1	2	3	4	5	6	
1	Актуальність	1	1 / 4	1 / 3	5	1 / 4	2	0,15
2	Точність	1	3	7	7	1	1	0,26
3	Зручність	1 / 7	4	1	5	1 / 2	1 / 7	0,114
4	Коректність	2	1	4	1 / 6	3	1	0,17
5	Доступність	1 / 3	1 / 7	1 / 5	1 / 5	1 / 8	1	0,04
6	Якість	1 / 2	7	1	6	1	8	0,33

Після чого здійснюється попарне порівняння критерію з типами систем. Таким чином було здійснене попарне порівняння систем для кожного визначеного критерію (таб. 2.2–2.8).

Таблиця 2.2

Матриця попарного порівняння систем за критерієм «Актуальність» в дереві цілей для додатку для контролю та прогнозування фінансових витрат

№	Назва системи	Порівняння систем					Вектор пріоритету
		1	2	3	4	5	
1	Інформаційно- консультаційні	1 / 2	1 / 7	6	1 / 6	1 / 7	0,4
2	Інформаційно-керівні	6	1 / 3	5	1 / 4	1 / 4	0,2
3	Інформаційно-пошукові	1 / 5	1 / 6	1 / 2	1 / 4	1 / 6	0,3
4	Інтелектуальні інформаційні	4	1 / 3	5	1	1 / 2	0,5
5	СППР	5	3	5	1 / 2	1	0,45

Таблиця 2.3

Матриця попарного порівняння систем за критерієм «Точність» в дереві цілей для контролю та прогнозування фінансових витрат

№	Назва системи	Порівняння систем					Вектор пріоритету
		1	2	3	4	5	
1	Інформаційно-консультаційні	1 / 2	6	1 / 4	1 / 5	7	0,74
2	Інформаційно-керівні	1 / 2	3	1 / 4	1 / 4	6	0,28
3	Інформаційно-пошукові	1 / 5	1 / 7	1 / 5	1 / 3	5	0,22
4	Інтелектуальні інформаційні	1 / 4	3	3	1 / 8	1 / 3	0,27
5	СППР	1 / 6	1 / 2	1 / 4	1 / 5	1 / 4	0,21

Таблиця 2.4

Матриця порівняння систем для критерію «Зручність» в дереві цілей при системному аналізі об’єкту дослідження

№	Назва системи	Порівняння систем					Вектор пріоритету
		1	2	3	4	5	
1	Інформаційно- консультаційні	1 / 2	1 / 4	2	1 / 5	5	0,2
2	Інформаційно-керівні	4	1 / 2	5	1 / 4	3	0,5
3	Інформаційно-пошукові	1 / 5	1 / 6	1 / 5	1 / 4	7	0,4
4	Інтелектуальні інформаційні	1 / 7	3	5	2	4	0,38
5	СППР	3	1 / 3	1 / 3	1 / 4	4	0,17

Таблиця 2.5

Матриця порівняння систем для критерію «Коректність» дереві цілей для додатку з контролю та прогнозування фінансових витрат

№	Назва системи	Порівняння систем					Вектор пріоритету
		1	2	3	4	5	
1	Інформаційно- консультаційні	1 / 3	2	3	3	7	0,6
2	Інформаційно-керівні	1 / 3	4	1 / 5	4	5	0,24
3	Інформаційно-пошукові	1 / 2	3	1 / 3	3	1 / 2	0,36
4	Інтелектуальні інформаційні	1 / 2	1	1 / 2	1 / 2	3	0,22
5	СППР	1 / 2	3	4	1 / 5	1 / 4	0,28

Таблиця 2.6

Матриця порівняння систем для критерію «Доступність» в дереві цілей для додатку з контролю та прогнозування фінансових витрат

№	Назва системи	Порівняння систем					Вектор пріоритету
		1 / 2	3	4	3	5	
1	Інформаційно- консультаційні	3	1 / 4	½	5	1 / 3	0,24
2	Інформаційно-керівні	1 / 7	1 / 2	1 / 3	2	3	0,22
3	Інформаційно-пошукові	3	6	4	3	4	0,6
4	Інтелектуальні інформаційні	4	3	¼	3	1 / 3	0,6
5	СППР	3	2	1 / 3	1 / 4	1	0,49

Таблиця 2.7

Матриця порівняння систем для критерію «Якість» в дереві цілей для додатку з контролю та прогнозування фінансових витрат

№	Назва системи	Порівняння систем					Вектор пріоритету
		2	3	3	6	3	
1	Інформаційно-консультаційні	3	7	9	1 / 5	2	0,47
2	Інформаційно-керівні	5	3	1 / 8	3	4	0,33
3	Інформаційно-пошукові	1 / 2	1 / 7	1 / 3	1 / 2	3	0,24
4	Інтелектуальні інформаційні	1 / 9	4	1 / 3	1 / 7	½	0,26
5	СППР	1 / 7	6	4	1 / 4	6	0,44

Далі під час аналізу дерева цілей була розроблена таблиця, яка містить усі коефіцієнти критеріїв для кожного типу системи, і була визначена остаточна пріоритетність інформаційної системи у таб. 2.8.

Таблиця 2.8

Результати методу аналітичної ієрархії в дереві цілей для додатку з контролю та прогнозування фінансових витрат

Критерій Тип систем	Актуальність	Точність	Зручність	Коректність	Доступність	Якість	Пріоритет
Інформаційно-консультаційні	0,015	0,034	0,165	0,018	0,086	0,2566	0,49
Інформаційно-керівні	0,15	0,064	0,044	0,029	0,018	0,085	0,28
Інформаційно-пошукові	0,17	0,029	0,023	0,024	0,023	0,036	0,12
Інтелектуальні інформаційні	0,032	0,035	0,042	0,026	0,022	0,033	0,19
СППР	0,033	0,019	0,011	0,028	0,019	0,045	0,29

Після порівняння найвищим значенням виявилася інформаційно-консультаційна система. Цей варіант було вибрано як найбільш критичний фактор успішності для майбутнього розроблення додатка з контролю та прогнозування фінансових витрат.

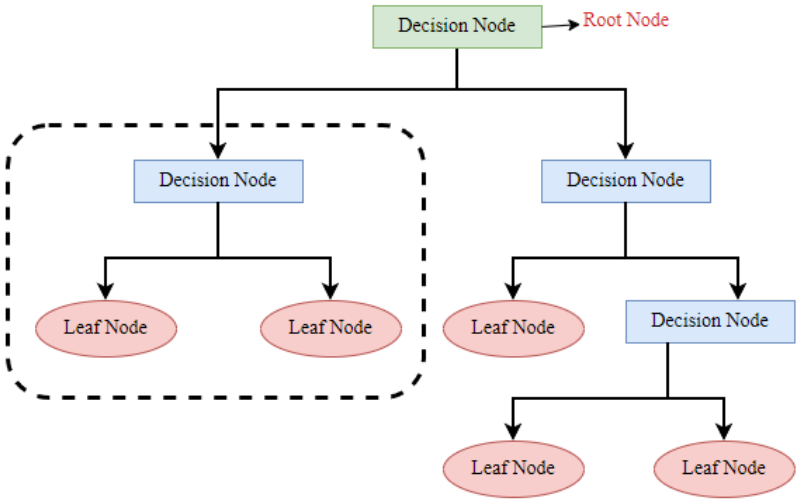


Рис. 2.2. Приклад роботи алгоритму дерева цілей для додатку з контролю та прогнозування фінансових витрат

Метою використання цього типу дерева рішень є побудова навчальної моделі, яка може бути використана для прогнозування класу або значення цільової змінної шляхом вивчення простих правил прийняття рішень на основі історичних даних (навчальних даних). У дереві рішень, щоб передбачити клас/мітку запису, починають з кореня дерева. Значення атрибутів кореня порівнюються з атрибутами запису. На основі порівняння гілка, що відповідає цьому значенню, прямує до наступного вузла. Тип дерева рішень залежить від типу цільової змінної.

2.5. Концептуальне проєктування інформаційної системи

На основі сформованих вимог було проведено моделювання роботи додатка з використанням методології IDEF0. Згідно вимог до розроблюваного додатка, основними його функціями є отримання і збереження інформації про персональні витрати від користувача і формулювання аналітики створеної на основі цих даних, а також оцифрування і збереження розрахункових документів. Таким чином, визначимо головний процес контекстної діаграми, як «Контроль персональних витрат». Діаграма містити вхідні і вихідні дані, а також управління і механізми.

Рис. 2.3. Контекстна діаграма процесу «Контроль персональних витрат»

Розглянемо детально ці елементи:

- Вхідні дані – це інформація, яку надає користувач. На основі обробки цієї інформації системою будуть сформовані вихідні дані. До вхідних даних належать: інформація про витрати, розрахункові документи;
- Вихідні дані – це кінцевий результат, який має отримати

користувач в результаті взаємодії з додатком. До вихідних даних належать: сформована історія витрат, архів збережених документів і аналітичні дані створені на основі інформації про витрати;

- Управління – елементом управління в роботі додатка є база даних. БД очікує від користувача правильно і в повному обсязі заповнених полів при створенні нової витрати, чи нового документа. Реакцією на помилку користувача буде відмова в збереженні інформації.
- Механізмам – це сам користувач, який взаємодіє з додатком.

Далі проведемо декомпозицію процесу «Контроль персональних витрат», описавши більш детально роботу додатка.

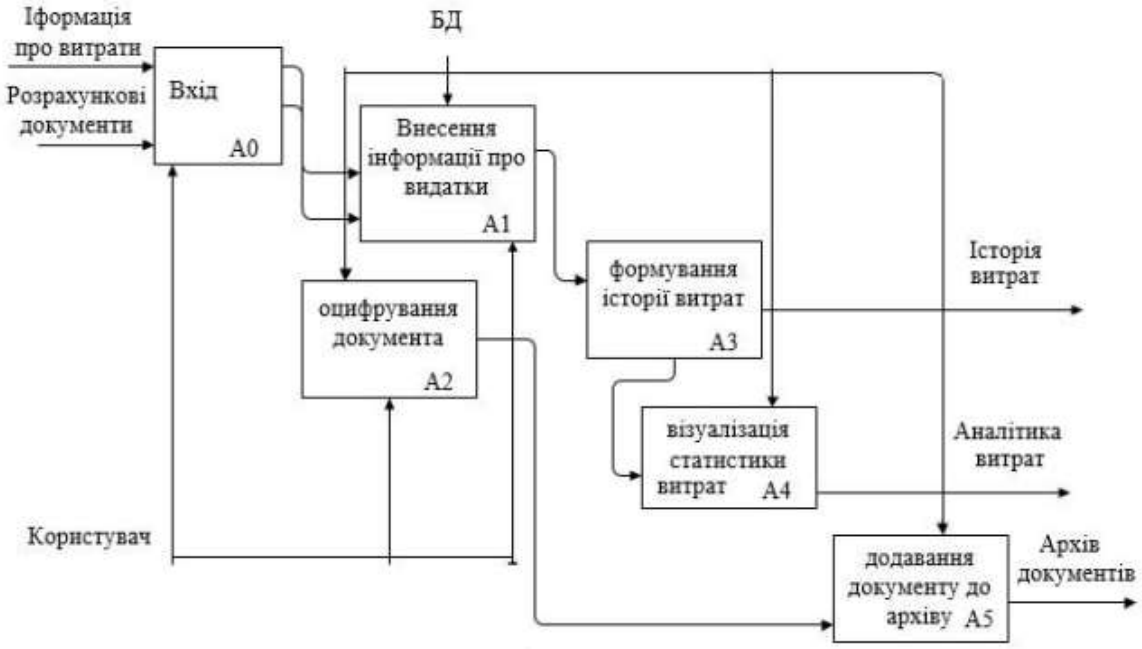


Рис. 2.4. Декомпозиція процесу «Контроль персональних витрат»

Розглянемо детально послідовність дій:

- після відкриття додатка користувач може ввести дані про видатки;
- користувач може оцифрувати документ;
- правильність введення даних в цих випадках контролює БД;
- на основі внесених даних формується історія витрат і архід документів;
- на основі історії витрат користувач отримує аналітику даних.

Далі проведемо декомпозицію блока 1 «Внесення інформації про

видатки» і блока 2 «Оцифрування документа».

Декомпозиція процесу «Внесення інформації про видатки»

Розглянемо послідовність дій при внесенні інформації про витрати:

- Першим, що робить користувач, це ініціює процес створення нової витрати, натиснувши на відповідну кнопку [19];
- Відкривається форма для заповнення інформації про витрату;
- Проходить перевірка правильності введення даних, на даному етапі користувач може отримати відмову в збережені даних в разі некоректного заповнення полів;
- На основі вірно внесених даних створюється новий запис в БД;

Вихідними даними даного блоку є формування історії витрат.

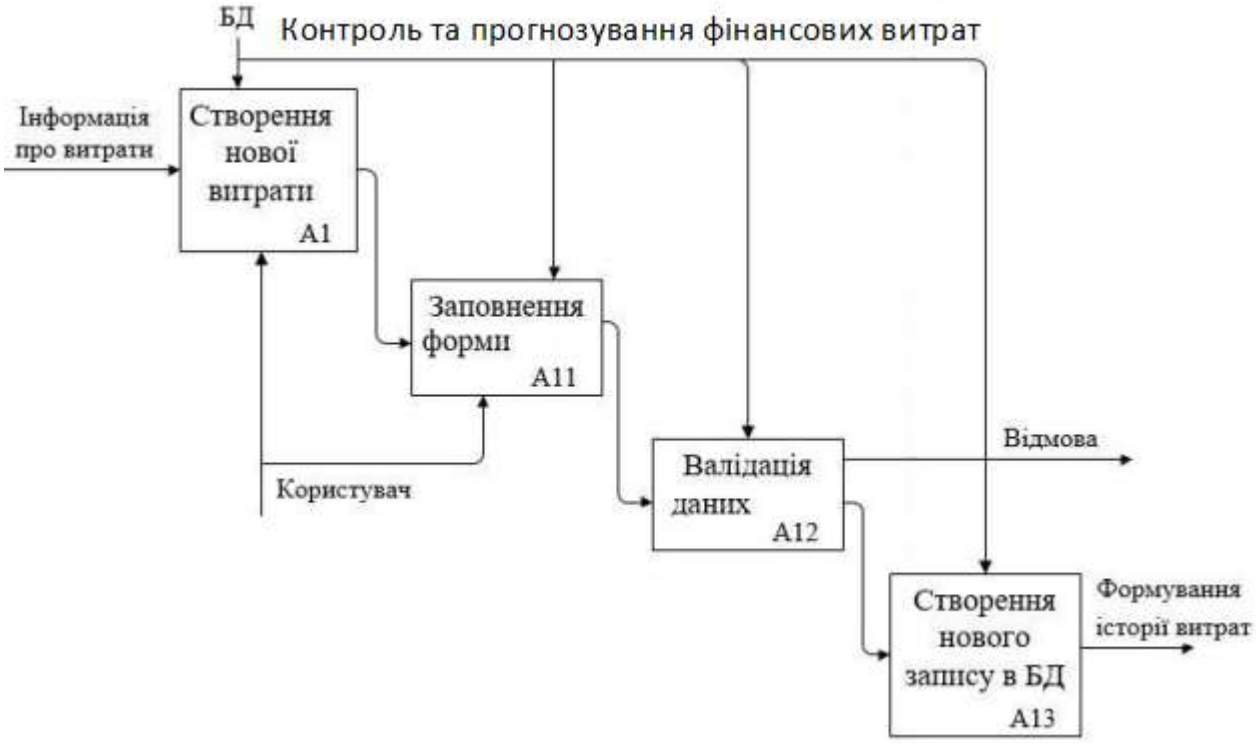


Рис. 2.5. Декомпозиція процесу «Оцифрування документа»

Розглянемо послідовність дій при оцифруванні документів:

- Як і при створенні витрат, першим, що робить користувач, це ініціює процес натисненням кнопки;
- Заповнює форму;
- Робить знімок документу;

- Збереження запису:
- Фото зберігається до файлової системи;
- Дані з форми зберігаються в БД;
- Процес контролює БД;
- Вихідними даними буде додавання нового документа до архіву.

На даному етапі, коли змодельовані процес додатка, створюємо діаграму варіантів використання. Діаграма варіантів використання складається з множини акторів і варіантів використання обмежених межею системи, представленою у вигляді прямокутника. Діаграма демонструє асоціації між акторами та варіантами використання, відношення між варіантами використання і узагальнені відношення між акторами. Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей або акторів, що взаємодіють із системою за допомогою варіантів використання. Варіант використання застосовується для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система під час діалогу з актором. Діаграма використання, розроблюваного мобільного додатка, наведена на рис. 2.5.

З рис. 2.5 видно, що при вході в додаток користувач має доступ до таких функцій:

- Створення нового документа;
- Створення нової витрати;
- Перегляд історії витрат;
- Перегляд деталей витрати;
- Головне меню;
- Меню налаштувань;
- Калькулятор.

Також з діаграми видно, як здійснюється перехід від одної одного процесу до іншого і взаємозв’язок та залежність процесів розроблюваного додатка.

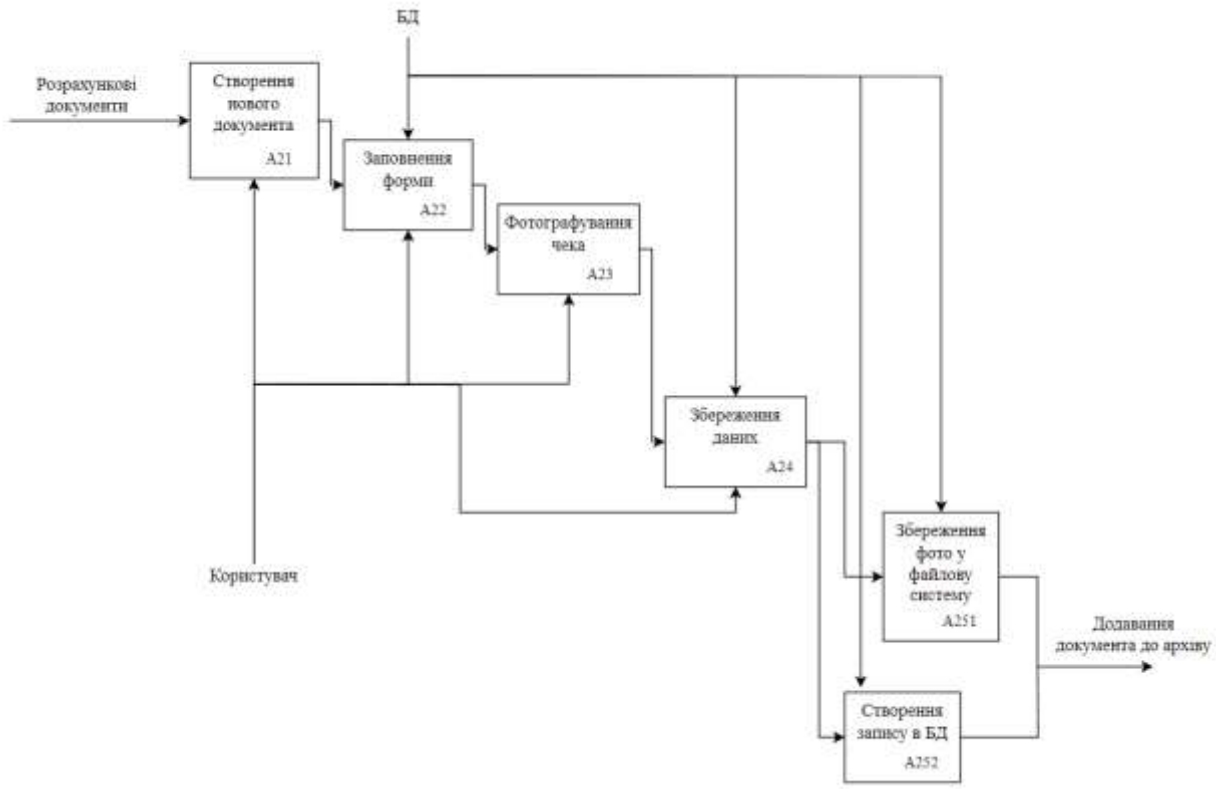


Рис. 2.6. Декомпозиція процесу «Оцифрування документа»

2.6. Розробка структури бази даних та use case діаграми

Firebase Realtime Database – сучасна хмарна NoSQL база даних від компанії Google, оптимізована для створення мультикористувацьких додатків з можливістю синхронізації в режимі реального часу. Вона надає безпечний доступ до даних безпосередньо з клієнтського коду як для читання, так і для запису.

Зміни кешуються локально та автоматично синхронізуються з хмарним сервером при наявності підключення. Це дозволяє користувачам отримувати актуальні дані та реагувати на оновлення в реальному часі незалежно від стану мережі. При синхронізації локальних та віддалених змін конфлікти вирішуються автоматично за вбудованими правилами.

Контроль доступу до ресурсів бази даних реалізується за допомогою гнучких правил на основі виразів. Інтеграція з сервісом аутентифікації Firebase Auth дозволяє на рівні правил визначати, які саме користувачі і які дані можуть читати чи змінювати.

Будучи базою даних типу NoSQL, Firebase Realtime Database оптимізована саме під швидкі транзакції з даними в реальному часі, а не під складну обробку та узгодженість, як традиційні реляційні СУБД. Тому при проєктуванні бази важливо правильно структурувати дані та враховувати сценарії доступу до них з боку користувачів. Це вимагає обережного підходу та оптимізації запитів.

На відміну від класичних реляційних СУБД, NoSQL бази даних, до яких належить і Firebase, жертвують деякими характеристиками ACID (атомарність, узгодженість, ізольованість, довговічність) на користь масштабованості, продуктивності та доступності. Більшість сучасних NoSQL баз даних реалізують альтернативну модель BASE (Basically Available – базова доступність, Soft state – «м'який» стан, Ultimately consistent – потенційна узгодженість). Вона гарантує узгодженість даних з часом, але не миттєву узгодженість, на відміну від ACID.

Основні переваги NoSQL баз даних:

1. Простота та швидкість розробки додатків завдяки більш інтуїтивному відображенню даних у пам'яті програм.

2. Масштабованість та можливість обробки великих обсягів даних в розподіленому кластері серверів.

3. Висока доступність та відмовостійкість за рахунок реплікації та автоматичного відновлення.

При проєктуванні архітектури додатків з використанням Redux як централізованого контейнеру стану починають з аналізу необхідної структури даних стану системи. Вона описує, які саме дані повинні зберігатися в додатку в будь-який момент часу для коректної роботи усіх функцій.

На відміну від типових підходів, де стан зазвичай розподілений по різних моделях, сервісах та компонентах, в архітектурі Redux весь стан додатку зберігається в єдиному централізованому сховищі (Store) у вигляді зв'язаних JavaScript об'єктів та масивів. Така організація даних дозволяє абстрагуватися від конкретної поведінки окремих елементів системи і зосередитись саме на чистих даних, що описують її поточний стан. Вся бізнес-

логіка та поведінка реалізується відокремлено від стану за допомогою спеціальних функцій (ред'юсерів).

Типова структура use case діаграми наведена на рис. 2.7 на прикладі гіпотетичного додатку для кур'єрської доставки.

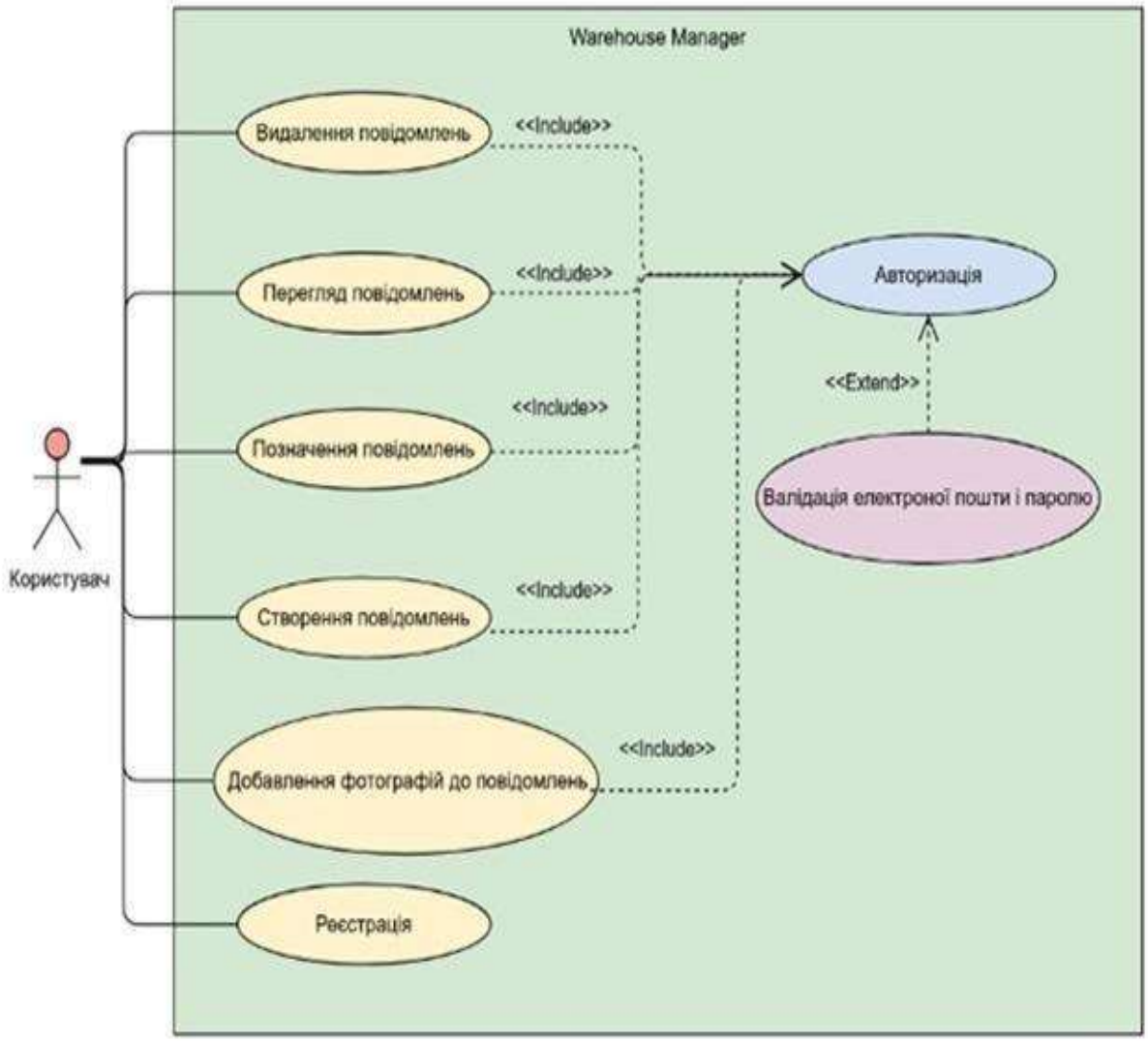


Рис. 2.7. Зовнішній вигляд use case діаграми

На прикладі можливої структури даних для веб-додатку з контролю фінансових витрат показано використання словників (асоціативних масивів) для зберігання колекцій об'єктів та звичайних масивів – для списків даних.

Поведінкову модель програмної системи при її проєктуванні формалізують за допомогою різних UML-діаграм (діаграм класів, діяльності, послідовності тощо). Одним з ключових типів є діаграми варіантів використання (use case diagrams), які описують функціональні вимоги у вигляді сценаріїв або варіантів використання системи.

Вони моделюють взаємодію зовнішніх сутностей (акторів, тобто користувачів, інших систем) з розроблюваною системою для досягнення певних цілей. На діаграмах зображуються актори та їх зв’язки з конкретними бізнес-процесами чи функціями системи, які необхідні для виконання поставлених цілей.

Основні кроки розробки діаграм варіантів використання:

1. Визначення ролей користувачів системи

2. Встановлення цілей і завдань для кожної ролі

3. Деталізація сценаріїв взаємодії з системою для досягнення цілей

4. Зв’язування ролей із конкретними бізнес-процесами чи функціями

Переваги використання діаграм варіантів використання:

- Структурування функціональних вимог до системи
- Формалізація очікуваної поведінки та сценаріїв роботи системи
- Встановлення зв’язків між користувачами та функціями системи

Отже, UML моделювання дозволяє впорядкувати та задокументувати вимоги до функціонування розроблюваної системи, спростивши подальший процес її проєктування та імплементації. В поєднанні з сучасними підходами та інструментами на кшталт Firebase та Redux, з’являється можливість будувати масштабовані, відмовостійкі та водночас зручні у використанні веб та мобільні додатки.

ВИСНОВКИ ДО РОЗДІЛУ 2

Надано огляд операційної системи Android 10 разом із вказівкою на її ключові особливості. Відзначено, що Android 10 ґрунтується на відкритому вихідному коді та доступний для вільного використання. Це дає можливість розробникам отримувати доступ до вихідного коду та розуміти, як працюють функції та особливості додатків. Кожен користувач може приєднатися до процесу вдосконалення операційної системи, подавши звіт про виявлені помилки або беручи участь у спільноті Open Source Project. Гнучкість Android зумовлена його базовим ядром Linux, яке також є відкритим для розробників, надаючи їм безліч можливостей. Окрім цього, Android може працювати на пристроях з об'ємом оперативної пам'яті менше 256 Мб.

Проведено обгрунтування використаного середовища та дано характеристики таких технологій, як JavaScript, React Native, Redux, Kotlin, Firebase. Виконано вибір цільового призначення додатку на основі дерева цілей та сказано, що головним призначенням нашого проєкту є інформаційно-консультаційний тип. Для створюваного проєкту виконано концептуальне проєктування та дано його розшифровку.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1. Програмне виконання мети дослідження

Для відображення елементів Категорій, Витрат та Документів у додатку використовується компонент інтерфейсу RecyclerView. Цей компонент призначений для швидкого та ефективного відображення великих обсягів даних. Для коректної роботи RecyclerView необхідно створити:

- клас, відповідальний за сам компонент;
- клас, відповідальний за поведінку та зовнішній вигляд окремого елемента списку;
- адаптер – клас, що об’єднує дані з відповідними представленнями;
- клас layout manager, який відповідає за розташування окремих елементів у RecyclerView.

Presenter класи у вигляді адаптерів наведено у таб. 3.1

Таблиця 3.1

Перелік класів-адаптерів

Назва класу	Опис
BaseExpenseAdapter	Використовується для відображення елементів історії на екрані «історія витрат».
CategoriesAdapter	Відповідальний за побудову списку категорій на екрані «Категорії»
Categories AdapterFolder	використовуючи окремий layout manager, відображує категорії у вигляді директорій на екрані «Документи».
CategoriesSpinner Adapter	Наповнює даними віджети Spinner на екранах створення нової витрати та на екрані створення нового документа.
DocumentsAdapter	Після вибору категорії на екрані Документи даний Адаптер наповнює даними RecyclerView який відповідає за відображення списку документів обраної категорії.
MainExpenseAdapter	Використовується для наповнення даними переліку витрат на екрані «Витрати»

Класи View перелічені у таб. 3.2

Таблиця 3.2

Перелік класів View

Назва	Відображення
Categories: - CategoriesFragment	Екран категорій
Documents: DocumentFragment NewDocumentActivity NewDocumentFragment	Екран документів із екраном додавання / оновлення документа
ExpenseCalculator: CalculateFactorial ExCalculatorActivity ExtendedDoubleEvaluator	Екран калькулятора
Expenses: ExpenseDetailActivity ExpenseDetailFragment ExpensesContainerFragment ExpensesFragment ExpensesViewPagerAdapter NewExpenseFragment	Класи витрати, оновлення / додавання витрати
Folders: - FolderFragment	Категорії у вигляді директорій
Help: - HelpActivity	Екран допомоги
History: - HistoryFragment	Екран історії
Settings: - SettingsActivity	Екран налаштувань
Statistic: - StatisticsFragment	Статистика
MainActivity	Головний екран додатка

Model – папка, в якій розміщені Java-класи, що відповідають сутностям у базі даних. Анотація @Entity позначає клас як сутність і вимагає наявності пустого конструктора та принаймні одного поля з анотацією @Id, що вказує на первинний ключ. Анотація @GeneratedValue вказує, що це поле буде автоматично генеруватися відповідно до заданого типу. Анотація @Temporal використовується для полів типу Date чи Calendar, які у базі даних мають тип Date. Анотація @JoinColumn позначає зовнішній ключ і вказує, з яким полем з іншої таблиці він зв'язаний. Анотація @ManyToOne відповідає за зв'язок «багато до одного».

Repository – папка, що містить класи для створення запитів до сутностей. На рис. 3.1 представлений код класу UsersRepository, який успадковується від CrudRepository. Цей інтерфейс належить до можливостей, які надають Spring Boot та Hibernate, щоб уникнути необхідності самостійного написання запитів до бази даних.

CrudRepository надає набір базових методів для роботи з сутностями:

```
<S extends T> S save (S entity);
<S extends T> Iterable<S> saveAll (Iterable<S> entities);
Optional<T> findById (ID id);
boolean existsById (ID id);
Iterable<T> findAll();
Iterable<T> findAllById (Iterable<ID> ids);
long count();
void deleteById (ID id);
void delete (T entity);
void deleteAll (Iterable<? extends T> entities);
void deleteAll();
```

Ці методи викликаються на сервері з відповідними параметрами. Проте, для створення власних запитів, які не надає інтерфейс, потрібно використовувати анотацію @Query та вказувати необхідний запит у значенні анотації.

Рис. 3.1. Інтерфейс UsersRepository

Service – папка, де зберігаються файли з бізнес-логікою програми, позначається анотацією @Service. На рис. 3.2 представлено частину коду класу GroupsService. Тут анотація @Autowired забезпечує створення екземпляра GroupsRepository groupsRepository. Метод getAllGroups() повертає список груп, які зберігаються в базі даних, використовуючи метод findAll() з інтерфейсу CrudRepository. Цей метод не потребує написання запиту, що демонструє зручність використання Hibernate.

```
@Service
public class GroupsService {

    @Autowired
    private GroupsRepository groupsRepository;


    public List<Groups> getAllGroups() {

        List<Groups> groupsList = new ArrayList<>();
        groupsRepository.findAll().forEach(groupsList::add);
        return groupsList;
    }
```

Рис. 3.2. Частина коду класу GroupsService

Під час створення об’єктів та функцій на Java дотримувалися принципів ООП, а також організації файлів відповідно до клієнт-серверної архітектури. Згідно з функціональними вимогами, зазначеними у попередньому розділі даної роботи можна розглянути їх програмну реалізацію:

Реєстрація

Для реєстрації користувачу необхідно заповнити форму, яка містить такі поля: ім’я, прізвище, логін, пароль та група. Якщо користувач не знає назви групи, до якої він хоче приєднатися, він може створити нову групу. Для цього потрібно натиснути кнопку «Додати групу» і перейти на сторінку створення групи. Ця форма містить лише одне поле – ім’я групи. Після натискання кнопки «Зберегти» виконується ajax-запит методом POST до сервера на URL «/groups». Дані про групу зберігаються в об’єкти веб-сховища sessionStorage і зберігаються у сесії. Після повернення на сторінку реєстрації ім’я групи з сесії записується у відповідне поле.

Під час введення даних відбувається їх перевірка відповідно до вказаних типів (наприклад, date для дати, password для паролю) та відповідності регулярним виразам. Якщо користувач не заповнив якесь поле та натиснув кнопку «Зберегти», система виведе повідомлення про те, що якесь поле залишилося пустим, використовуючи атрибути форми. Коли всі поля заповнені правильно, виконується ajax-запит методом POST до сервера на URL «/users / add». Rest controller хешує пароль за допомогою створеного біну bCryptPasswordEncoder Spring Security, встановлює користувачу роль і передає його до сервісу, який записує дані в базу даних.

Авторизація на сайті

Для авторизації користувачу потрібно ввести логін та пароль у форму. Поля перевіряються регулярними виразами під час введення у поля input. Засобами Thymeleaf (з використанням th: action=«@{/login}» method=«post») перевірка введених даних виконується автоматично. Якщо введені дані не відповідають записам у базі даних, використовується умова th: if=«${param. error}», щоб повернути користувачу повідомлення про помилку.

Вхід у власний кабінет

Після успішної авторизації користувач потрапляє на персональну сторінку. Тут він може переглянути інформацію про себе, свій дохід, витрати та поточний баланс. Для цього виконується запит на сервер методом GET за відповідним URL, і повертається колекція даних у форматі JSON. Якщо у

користувача є регулярні витрати, про це буде повідомлено. При натисканні на кнопку «У вас є регулярні витрати» з'являється модальне вікно з переліком цих витрат, сумами та частотою платежів. Ці дані отримуються за допомогою функції getJSON() через запит GET на URL «/transactions / regular_transactions/»+userId. На сервері всі транзакції користувача фільтруються за допомогою Stream API (рис. 3.3). Крім того, на сторінці є перелік кнопок, натискання на які викликає метод click(), що переводить користувача на інші сторінки.

```
public List<Transactions> getRegularTransactionsByUserAndMonth(Integer user_id) {
    Users user = new Users();
    user.setId_users(user_id);
    List<Transactions> reg = transactionsRepository.getAllByIdUser(user).stream().
            filter(t -> transactionFilter(t)).collect(Collectors.toList());

    return reg;
}
```

Рис. 3.3. Приклад коду, де відбувається фільтрація даних засобами Stream API

Додавання транзакції

Форма для додавання транзакцій з'являється після натискання кнопки «Додати транзакцію» в особистому кабінеті. На новій сторінці відображається форма, де потрібно ввести дату, опис (можна додати новий або вибрати один із вже доданих користувачем; для цього виконується GET-запит функцією getJSON() на URL «/transactions / allDescription») і суму для нерегулярних транзакцій. Якщо у випадаючому списку вибрати значення поля regular, відкриються додаткові поля, які були раніше приховані за допомогою стилю CSS display=block. Ці поля включають дату початку і кінця, частоту, яку можна вибрати з випадаючого списку. Валідація даних здійснюється за допомогою регулярних виразів та атрибутів тега form. Після успішного заповнення всіх полів і натискання кнопки «Зберегти» виконується ajax-запит методом POST до сервера на URL «/transactions». Якщо результат успішний, користувач повертається на сторінку особистого кабінету [11].

Додавання доходів

Форма для додавання доходів з’являється після натискання кнопки «Додати дохід» в особистому кабінеті. На новій сторінці відображається форма, де потрібно ввести дату та суму. Валідація даних здійснюється за допомогою регулярних виразів та атрибутів тега form. Після успішного заповнення всіх полів і натискання кнопки «Зберегти» виконується ajax-запит методом POST до сервера на URL «/incomes». REST-контролер передає дані у форматі JSON на сервер, де вони методом save() через репозиторій записуються у базу даних. Якщо результат успішний, користувач повертається на сторінку особистого кабінету.

Розподіл боргів

Для заповнення форми розподілу боргів у особистому кабінеті користувача потрібно натиснути кнопку «Розділити борг» і перейти на сторінку за URL «/ui / transactions / debt / user/». Форма містить три поля: два випадаючих списки користувачів (отримані через GET-запит функцією getJSON() на URL «/users / list_users») і поле для введення суми. Після успішної валідації виконуються два ajax-запити методом POST до сервера на URL «/transactions» та «/incomes». Дані додаються у відповідні таблиці в базі даних. Після успішного завершення запитів користувача повертають на сторінку особистого кабінету.

Перегляд інформації про групу

Переглянути інформацію про групу можна, натиснувши кнопку «Про групу» за URL «/ui / groups / read/» + idGroup, яке отримується через GET-запит функцією getJSON() на URL «/users». На сторінці групи відображаються суми доходів, суми транзакцій та баланс усіх учасників групи. Для цього виконуються відповідні GET-запити до сервера. Також повертається список усіх користувачів, які належать до тієї ж групи, що й поточний користувач. У репозиторії створюється кастомний запит для отримання необхідних даних з бази даних.

Крім того, користувач може переглянути два графіки, які показують частку доходів та транзакцій кожного учасника групи. Створення графіків

здійснюється за допомогою методу drawCanvas (container, data, text), де:

container – id тега div, де буде розміщений графік;

data – дані про користувачів у вигляді масиву об’єктів, отримані через GET-запит і відфільтровані на стороні клієнта (наприклад, для підрахунку частки доходу кожного користувача використовується формула:

*сума_користувача/сума_групи*100);*

text – підпис графіка.

Перегляд власних та групових транзакцій

Переглянути особисті транзакції можна з власного кабінету, а транзакції групи – зі сторінки групи, натиснувши кнопку «Всі транзакції». Для цього використовується функція ajaxGet(), яка передає у getJSON() відповідний URL для групи або користувача. Потім виконується GET-запит до сервера, і отримані дані у форматі JSON відображаються на сторінці у вигляді таблиці.

Перегляд власних доходів та доходів групи

Переглянути особисті доходи можна з власного кабінету, а доходи групи – зі сторінки групи, натиснувши кнопку «Всі надходження». Функціонал відображення даних у вигляді таблиці здійснюється аналогічно до відображення транзакцій.

Фільтрація

Фільтрація можлива при перегляді транзакцій або доходів. Для цього є два поля для введення дат, за які користувач бажає переглянути дані. Після успішної валідації виконується асинхронний ajax-запит методом GET на URL / incomes / user / period / ?start= + periodFrom + &end= + periodTo або / incomes / group / period / ?start= + periodFrom + &end= + periodTo. REST-контролер обробляє дати та передає їх на сервер, де вони обробляються методом findAllByDateBetweenByIdUser().

3.2. Опис файлів даних та інтерфейсу програми

Розроблюваний веб-застосунок для розподілу витрат використовує базу даних MySQL для зберігання даних.

«users» – таблиця, яка містить дані про зареєстрованих користувачів. Первинний ключ – id_users, зовнішній ключ – id_group.

id_users – унікальний номер користувача, який генерується в БД.

first_name – ім’я зареєстрованого користувача.

last_name – прізвище зареєстрованого користувача.

login – логін зареєстрованого користувача.

password – пароль зареєстрованого користувача.

id_group – унікальний номер групи, до якої належить користувач.

«user_groups» – таблиця, яка містить дані про групи, до яких належать користувачі. Первинний ключ – id_groups.

id_groups – унікальний номер групи.

name_group – назва групи.

«transactions» – таблиця, яка містить дані про транзакції, здійснені зареєстрованими користувачами. Первинний ключ – id_transaction, зовнішні ключі – id_user, id_type_transaction, id_frequency.

id_transaction – унікальний номер транзакції.

date – дата здійснення транзакції.

destination – ціль транзакції.

sum – сума транзакції.

period_from – дата початку (лише для регулярних транзакцій).

period_to – дата кінця (лише для регулярних транзакцій).

id_user – унікальний номер користувача, який здійснив транзакцію.

id_type_transaction – унікальний номер типу транзакції (може мати лише два значення: «regular» та «irregular»).

id_frequency – унікальний номер частоти для регулярних транзакцій (може мати лише три значення: «day», «week», «month»).

«income» – таблиця, яка містить дані про надходження, здійснені зареєстрованими користувачами. Первинний ключ – id_income, зовнішній ключ – id_user.

id_income – унікальний номер надходження.

date – дата надходження.

id_user – унікальний номер користувача.

sum_income – сума надходження.

«type_transaction» – таблиця, яка містить дані про типи транзакцій. Має лише два поля (див. Додаток А. Додаткові ілюстрації Рисунок 30). Первинний ключ – id_type_transaction.

id_type_transaction – унікальний номер типу транзакції.

name_type_transaction – назва типу транзакції (може мати лише два значення: «regular» та «irregular»).

«frequency» – таблиця, яка містить дані про частоту транзакцій. Має лише три поля (див. Додаток А. Додаткові ілюстрації Рисунок 31). Первинний ключ – id_frequency.

id_frequency – унікальний номер періоду.

value – назва періоду (може мати лише три значення: «day», «week», «month»).

«users_roles» – допоміжна таблиця, що пов’язує користувача з його роллю.

users_id_users – унікальний номер користувача.

roles_id – унікальний номер ролі користувача.

«role» – таблиця, яка містить дані про роль користувача. Має лише два поля. Первинний ключ – id.

id – унікальний номер ролі.

name – назва ролі (має лише два значення; для всіх користувачів встановлюється значення «ROLE_USER»).

Взаємодія з базою даних відбувається на серверному рівні за допомогою методів репозиторію, які надає Hibernate. Для власних запитів використовується анотація @Query. Перед записом даних, що надходять з

клієнтської сторони до бази даних, здійснюється перевірка на валідність у формі, застосовуючи атрибут pattern та регулярні вирази.

Після аналізу блок-схеми взаємодії користувача в додатку виділяються три основні складові: фронтенд (для інтеракції з користувачем), бекенд (для роботи з базою даних) та сама база даних (для зберігання інформації). Цей розподіл відповідає концепції трирівневої моделі клієнт-серверного додатка. Відповідно до цього, програмний код розділений на відповідні компоненти:

папка controller, яка складається з REST та MVC контролерів,

папка entity, яка складається з Java-класів відповідно до сутностей схеми бази даних,

папка service,

папка repository.

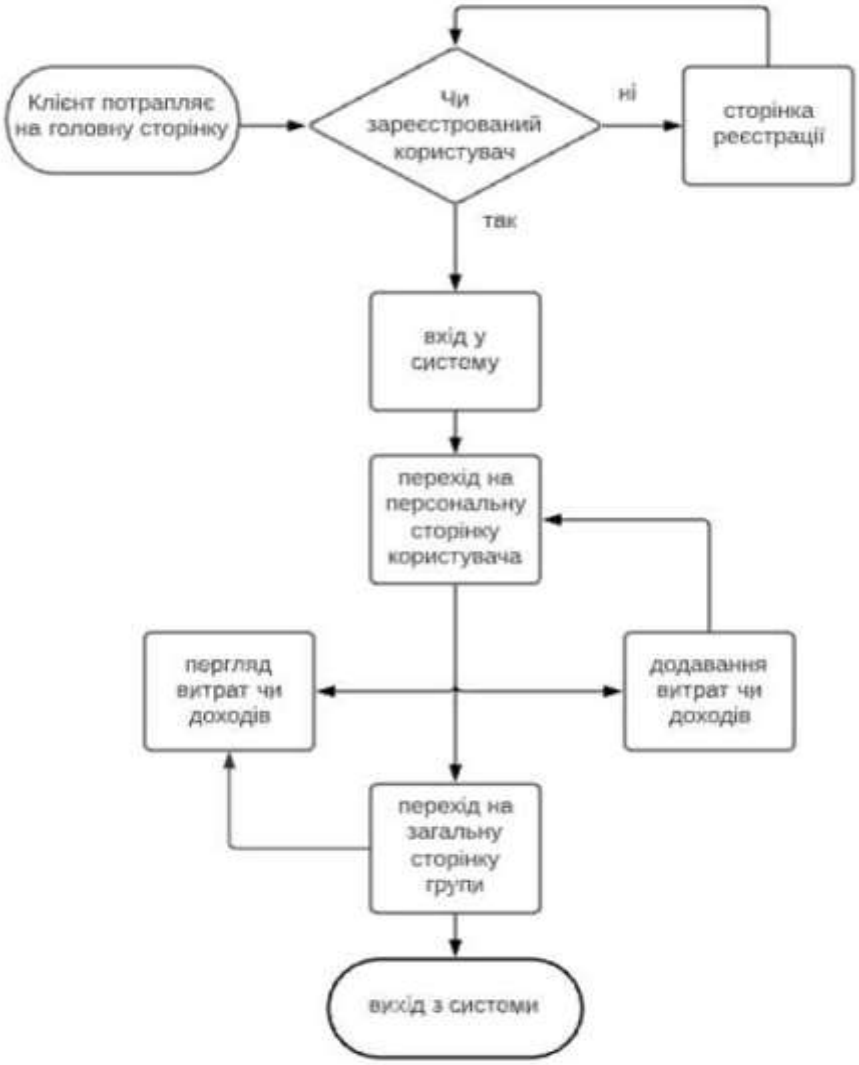


Рис. 3.4. Блок схема роботи застосунку

Rest контролер відповідає за надання даних у форматі JSON, тоді як MVC контролер забезпечує відтворення представлення, тобто HTML-сторінок разом з необхідними даними для передачі на сервіс. Сам сервіс відповідає за обробку запитів, які прийшли з репозиторію, що взаємодіє з базою даних.

Фронтенд взаємодіє з користувачем за допомогою JavaScript та JQuery. Він відображає контент на HTML-сторінках, розміщених у директорії resources проекту.

Бекенд реалізований на Java з використанням фреймворку Spring, який вбудовує ORM фреймворк Hibernate для зв'язку між таблицями у базі даних та Java-класами. Підключення до бази даних відбувається у файлі application. properties, де використана база даних MySQL.

3.3. Тестування програми

Тестування програми проводилося у емуляторі , що вбудований в Android Studio. При вході в додаток, користувач бачить головний екран (рис. 3.6). На цій сторінці неавторизованому користувачу доступні сторінки «Про нас», «Контакти», «Ввійти», «Зареєструватися».

а)

	id_type_transaction	name_type_transaction
▶	1	irregular
	2	regular
*	NULL	NULL

б)

	id_frequency	value
▶	1	day
	2	week
	3	month
*	NULL	NULL

Рис. 3.5. Таблиця type_transaction – а) та таблиця frequency – б)

Якщо користувач не має облікового запису у системі, він може перейти з головної сторінки на сторінку реєстрації та заповнити відповідну форму. Під час введення даних йому будуть надаватися підказки щодо формату, яким слід керуватися. У випадку, якщо користувач вже зареєстрований, він може негайно увійти до системи.

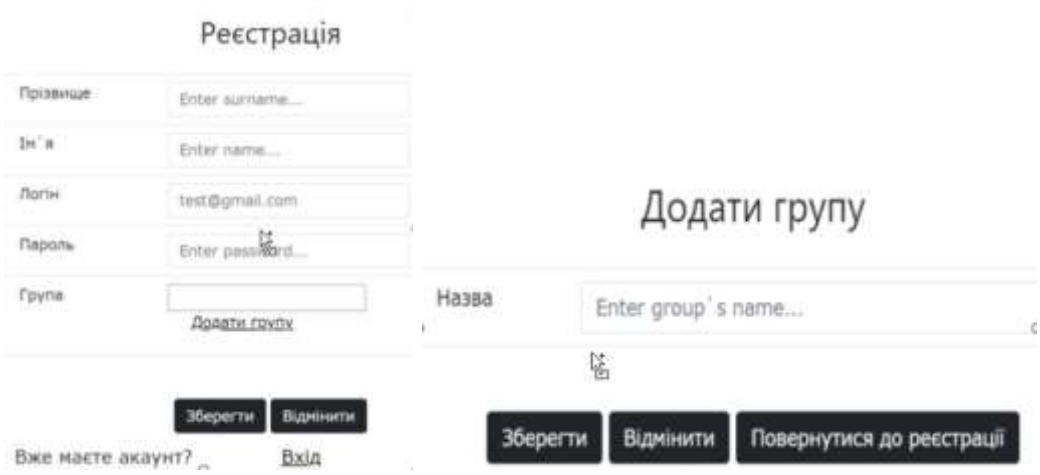


Рис. 3.6. Реєстрація нового користувача та створення групи

Перед тим як зареєструватися, користувачу необхідно вибрати групу або створити свою власну. Після успішної реєстрації, користувач має увійти до системи. Для цього, натиснувши кнопку "Увійти", він переходить на відповідну сторінку, де в формі необхідно ввести логін та пароль. У випадку введення неправильних даних, буде виведено відповідне повідомлення.

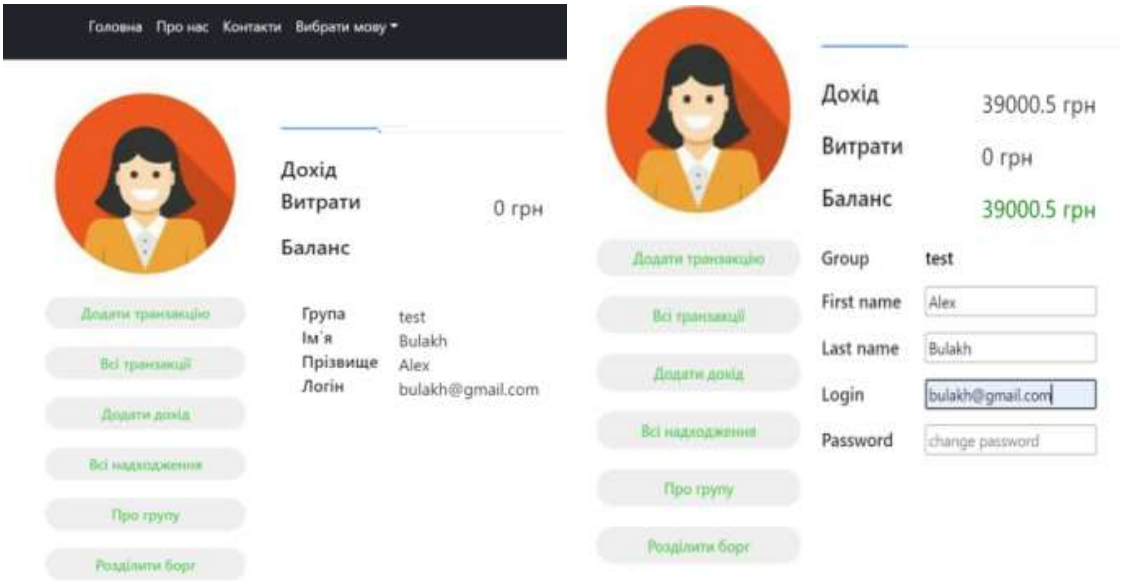


Рис. 3.7. Особистий кабінет користувача та редагування інформації про фінансові витрати

Після успішного входу в систему, відбувається перехід до особистого кабінету, де доступна вся інформація про користувача. При завершенні роботи можна вийти з системи, натиснувши кнопку "Вихід" на панелі меню.

У особистому кабінеті кнопка "Додати транзакцію" перенаправляє на сторінку форми для додавання нової транзакції. Для регулярних транзакцій у формі відкриваються додаткові поля. У разі неправильного введення даних, буде надано відповідне повідомлення. Після успішного введення з'явиться діалогове вікно, що про це повідомить.

Якщо користувач бажає редагувати особисті дані, при натисканні на кнопку "Редагування" відкривається сторінка редагування, де можна змінити відповідні поля. Кнопка "Додати надходження" у особистому кабінеті також перенаправляє на сторінку форми для додавання прибутку. У разі неправильного введення даних, буде надано відповідне повідомлення. Після успішного введення з'явиться діалогове вікно, що про це повідомить.

Кнопка "Всі транзакції" перенаправляє на сторінку з таблицею всіх транзакцій користувача. Список можна відфільтрувати за періодом, а також видалити або редагувати окремі записи.

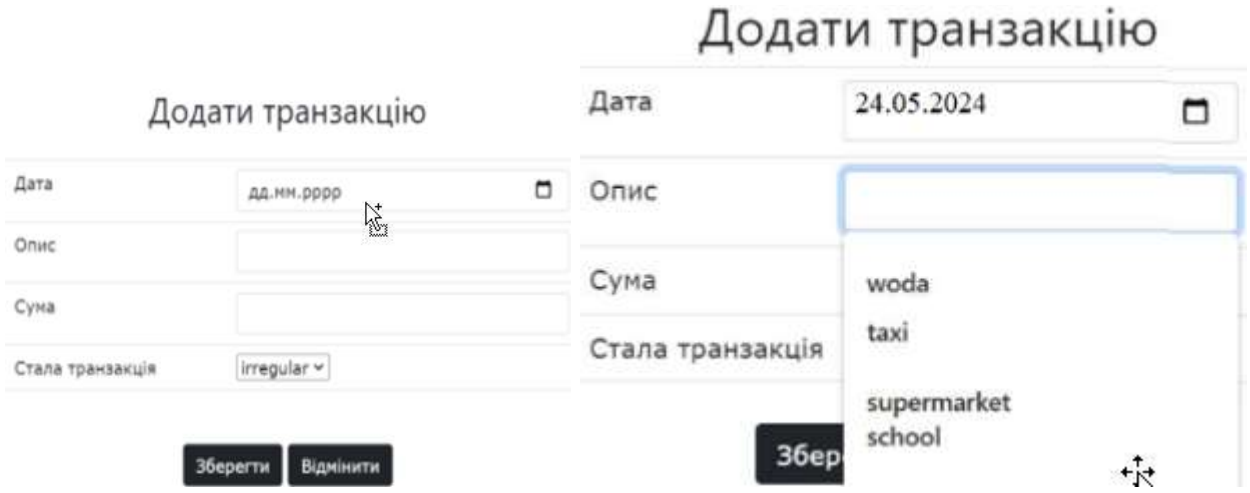


Рис. 3.8. Форма для додавання транзакції та список передбачуваних витрат

Кнопка "Всі надходження" на сторінці перенаправляє на таблицю, де відображені всі надходження користувача. Ви можете відфільтрувати список за періодом та редагувати або видаляти записи. З особистого кабінету є можливість перейти на сторінку з інформацією про групу, де також можна переглянути діаграми витрат та доходів групи. Навігація для перегляду транзакцій та надходжень по групі розташована на панелі меню.

На сторінці групи, натиснувши кнопку "Всі транзакції" або "Всі надходження", ви зможете переглянути таблиці з відповідними транзакціями або надходженнями групи. Ви можете відфільтрувати їх за датою, а також виконувати редагування або видалення за потребою.

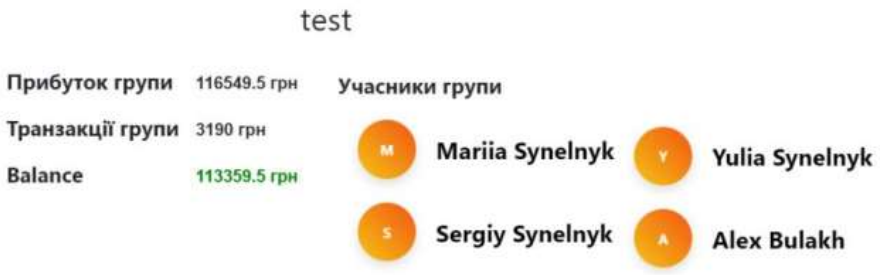


Рис. 3.9. Загальна сторінка групи

Також з особистої сторінки або сторінки групи можна розподілити борги, заповнивши відповідну форму для групи. Якщо у користувача є борги, його баланс відображається червоним кольором.

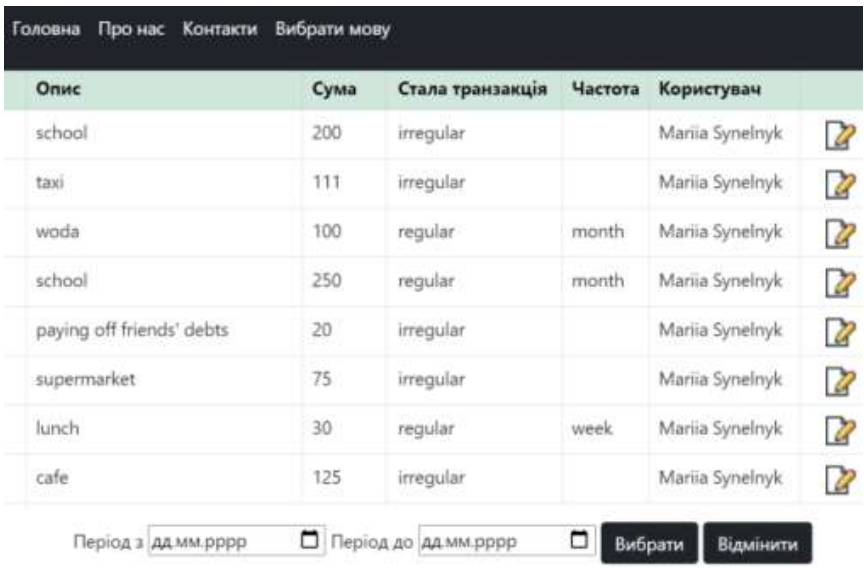


Рис. 3.10. Всі транзакції користувача

Якщо баланс групи від’ємний, жоден з користувачів не може додати транзакцію.

ВИСНОВКИ ДО РОЗДІЛУ 3

Було описано послідовність виконання програмної частини дослідження, а також подано програмний код. Надано огляд файлів даних та інтерфейсу програми, а також наведено блок-схему роботи додатку. Проведено компіляцію коду та тестування його працездатності. Була перевірена правильність обробки введених даних, реакція програми на помилкові введення та загальна стабільність. У результаті тестування виявлено деякі незначні помилки та можливості для поліпшення, які можуть бути враховані у майбутньому розвитку додатку.

ВИСНОВКИ

1. Початковий акцент у дослідженні зроблено на значущості автоматизованого відстеження витрат за допомогою смартфона. Викладено основні принципи контролю та прогнозування фінансових витрат. Підкреслено, що фінансове планування має базуватися на реалістичних прогнозах та вирішувати грошові питання, встановлюючи чіткі терміни вирішення. Результатом бюджетування повинні бути інформативні дані, які регулярно аналізуються для ухвалення обґрунтованих рішень.

2. Був проведений огляд наявних аналогів програм для контролю фінансових витрат на ринку. Наприклад, додаток Andromoney відомий своєю популярністю та високим рейтингом у Google Play. Він має інтуїтивно зрозумілий і зручний інтерфейс, що дозволяє вести облік фінансових операцій, створювати звіти та управляти бюджетом. Подібні програми надають користувачам широкий спектр можливостей для особистої бухгалтерії та фінансового планування. Наприклад, «Finance PM» дозволяє планувати операції щодо витрат, доходів або переказів на певну дату з різноманітними варіантами періодичності, такими як щоденні, щотижневі, щомісячні повторення, а також в конкретні дні тижня чи місяця. Плановані транзакції можна переглянути в розділі "Найближчі операції", а всі минулі – у розділі "Грошові переміщення".

3. У другій частині дослідження було виконане обґрунтування вибору засобів проєктування та докладно описані особливості операційної системи Android 10. Однією з ключових переваг платформи Android є її відкритість. Операційна система Android базується на відкритому вихідному коді та доступна для вільного використання. Це дає розробникам можливість отримати доступ до вихідного коду і зрозуміти, як реалізовані функції та властивості додатків.

4. Дана характеристика технологій JavaScript, React Native, Redux, Kotlin, Firebase. За допомогою дерева цілей розглянуто функціональність

розробки та її перспективність. Зроблено концептуальне проєктування інформаційної системи, розробку структури бази даних та use case діаграми.

5. Практична частина проекту була реалізована у вигляді веб-додатку, спрямованого на ведення та контроль витрат у родині. Користувачі мають можливість відстежувати як свої власні, так і загальні фінансові операції. Цей веб-додаток має зручний та адаптивний інтерфейс, що спрощує навігацію для всіх типів користувачів, навіть тих, хто не є досвідченими користувачами Інтернету.

6. Тестування програми охоплювало різноманітні сценарії використання та введення різних типів даних. Ми перевірили правильність обробки введених даних, реакцію програми на помилкові введення та загальну стабільність її роботи. У ході тестування було виявлено кілька незначних помилок та можливостей для поліпшення, які можна буде врахувати в подальшому розвитку програми.

7. Огляд та тестування закінченого додатку дозволили зрозуміти його загальну якість та готовність до використання. Виявлено, що додаток є функціональним, корисним і здатним задовольнити основні потреби користувачів у контролі фінансів. Однак, для досягнення ще більшого успіху, можна розглянути додаткові можливості для покращення та розвитку, такі як розширення функціональності, підтримка інших платформ або вдосконалення інтерфейсу користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритми та структури даних / уклад. О. В. Щербаков, Ю. Е. Парфьонов, В. М. Федорченко. Харків: ХНЕУ ім. С. Кузнеця, 2019. 58 с.
2. Аршулик Т. В. Дослідження методів інтерполяції зображень при створенні мультимедійного контенту. *Мультимедійні технології в освіті та інших сферах діяльності,* К.: НАУ, 2019. С. 13.
3. Об’єктивне порівняння iOS і Android в 2019 році [Електронний ресурс]. Режим доступу: https://journal.ilounge.ua/ua/review/obektivnoe-sravnenie-ios-i-android-v-2019-godu. Дата звернення – 03.06.2024
4. Бондаренко О. Безпека web-додатків. *Актуальні проблеми та їх аналіз.* 2019. № 3. С. 28–36.
5. Бондаренко О. Безпека web-додатків. *Актуальні проблеми та їх аналіз.* 2020. № 3. С. 28–36.
6. Голян В. В., Кравченко О. К., Порівняння моделей життєвих циклів програмного забезпечення з метою виявлення найефективнішого. – Харків: ХНУ, стаття, 2019. 8 с.
7. Zhmuts'ka N. S. Pobudova veb-dodatkiv na osnovi mikroservisnoi arkhitektury: Materialy XLVIII naukovo-tekhnichnoi konferentsii pidrozdiliv VNTU. Vinnytsia. 2019. 5 s.
8. Zhuravliov O. V., Simachov O. A. Statystychne doslidzhennia rynku IT-posluh v Ukraini. Statystyka Ukrainy. 2018. № 4. S. 25–33.
9. Izonin I. V. Metod zbil'shennia rozdil'noi zdatnosti zobrazhen' na osnovi shtuchnykh neironnykh merezh. Visnyk Lvivs'koho derzhavnoho universytetu bezpeky zhyttiediial'nosti. 2019. № 11. S. 47–56.
10. Klymash M. M., Shpur O. M. Monitorynh dostupnosti veb-servisu v rozpodilenykh infokomunikatsiinykh systemakh. Visnyk Universytetu "Ukraina" Seriia Informatyka, obchysliuvalna tekhnika. 2020. T. 1. №. 28. S. 35–44.
11. Lutskiv A. M. Arkhitektury vysokoproduktyvnykh system opratsiuvannia velykykh danykh. Zbirnyk tez dopovidei Naukovo-tekhnichna VI

Naukovo-tekhnichna konferentsiia "Informatsiini modeli, systemy ta tekhnolohii", 12–13 hrudnia 2018 roku. T.: TNTU, 2018. S. 75.

12. Melikhova T. O. Rozroblka prohrami audytu vytrat na vyrobnytstvo dlia pidvyshchennia finansovoi bezpeky pidpryiemstva. Ekonomika ta derzhava. 2018. № 1. S. 69–75.

13. Rudnichenko, M. Prohramna rozrobka systemy obroky ta filtratsii rastrovykh hravychnykh zobrazhen'. Informatsiini tekhnolohii ta suspilstvo, 2021. №1 (1), S. 51–58.

14. Teslenko O. K., Kisilchuk B. Ia. Formuvannia kliuchiv alhorytmu shyfruvannia na bazi liniinykh struktur. Prykladna matematyka ta komp'iutynh. Kyiv, NTUU "KPI", 14–16 lystopada 2019 r.

15. Tkach Yu. M. Tendentsii rozvytku suchasnoho kiberprostoru ta yoho zakhystenosti v umovakh informatsiinoho protyborstva. Bezpeka informatsii. 2020. T. 26 (2). S. 74–80.

16. Fillips B., Stuart K., Marsikano K. Android. Prohramuvannia dlia profesiionoliv. 2019. 688 s.

17. Шматко О. В. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посіб. / О. В. Шматко, А. О. Поляков, В. М. Федорченко. Харків: НТУ «ХПІ», 2018. 284 с.

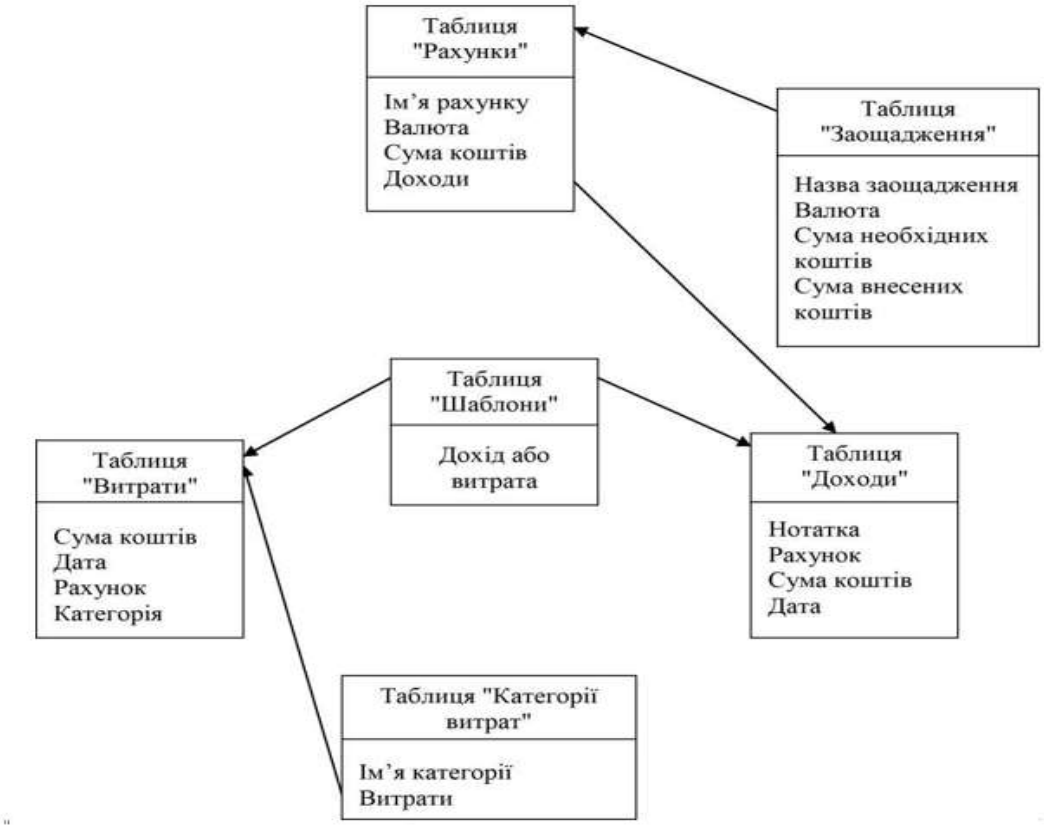
18. Розробка веб-додатків, мобільних додатків та порталів: [Електронний ресурс]. Режим доступу: http://ittel.com.ua/informacijni-texnologiyi/ rozrobka-mobilnih-dodatkiv/ Дата звернення – 03.06.2024.

19. Технології створення мобільних додатків [Електронний ресурс]. – Режим доступу: http://group-global.org/publication/63343-tehnologiisozdaniya-mobilnyh-prilozheniy. Дата звернення – 02.06.2024.

ДОДАТКИ

Додаток А

Структура бази даних



Діаграма форми «Статистика за весь час» та «Статистика за період»

TransactionsControlle

```
@Controller
@RequestMapping («ui / transactions»)
public class TransactionsController {
@GetMapping («/list»)
public String showTransactionsList (Model model){
model. addAttribute («user_id», 0);
return «transaction / list»;
}
@GetMapping («/debt / user/{user_id}»)
public String showTransactionsDebtByUser(@PathVariable String user_id, Model model){
model. addAttribute («user_id», user_id);
return «transaction / debt»;
}
@GetMapping («/list / user/{user_id}»)
public String showTransactionsByUser(@PathVariable String user_id, Model model){
model. addAttribute («user_id», user_id);
return «transaction / list»;
}
@GetMapping («/list / group/{group_id}»)
public String showTransactionsByGroup(@PathVariable String group_id, Model model){
model. addAttribute («group_id», group_id);
return «transaction / list»;
}
@GetMapping («/read/{transaction_id}»)
public String showTransactionsRead(@PathVariable String transaction_id, Model model){
model. addAttribute («transaction_id», transaction_id);
return «transaction / read»;
}
@GetMapping («/create»)
public String showTransactionsCreate (Model model){
Integer userId = ((Users) SecurityContextHolder. getContext(). getAuthentication(). getPrincipal()). getId_users();
model. addAttribute («user_id», userId);
return «transaction / create»;
}
@GetMapping («/update/{transaction_id}»)
public String showTransactionsUpdate(@PathVariable String transaction_id, Model model){
model. addAttribute («transaction_id», transaction_id);
return «transaction / update»;
}
```

IncomeRestController

```
@RestController
@RequestMapping («incomes»)
public class IncomeRestController {
/*
GET / incomes
GET / incomes / 1
POST / incomes
PUT / incomes / 1
DELETE / incomes / 1
*/
@Autowired
private IncomeService incomeService;
@GetMapping
public List<Income> getAllIncome() {
return incomeService. getAllIncome();
}
@GetMapping («/{id}»)
public Income getIncome(@PathVariable Integer id) {
return incomeService. getIncome (id);
}
@GetMapping («/sum»)
public Double getSum() {
Integer id = ((Users) SecurityContextHolder. getContext(). getAuthentication(). getPrincipal()). getId_users();
return incomeService. getSumAllIncomes (id);
}
@GetMapping («/sum-by-group»)
public Double getSumByGroup() {
Integer id = ((Users)
SecurityContextHolder. getContext(). getAuthentication(). getPrincipal()). getId_group(). getId_groups();
return incomeService. getSumUserGroupIncomes (id);
}
@PostMapping
public void addIncome(@RequestBody Income income) {
Users user = (Users) SecurityContextHolder. getContext(). getAuthentication(). getPrincipal();
income. setIdUser (user);
incomeService. addIncome (income);
}
@PutMapping («/{id}»)
public void updateIncome(@RequestBody Income income, @PathVariable Integer id) {
Users user = (Users) SecurityContextHolder. getContext(). getAuthentication(). getPrincipal();
income. setIdUser (user);
incomeService. updateIncome (income, id);
}
@DeleteMapping («/{id}»)
public void deleteIncome(@PathVariable Integer id) {
incomeService. deleteIncome (id);
```

```
}
```



```
@GetMapping («user/{user_id}»)
public List<Income> getIncomesByUser(@PathVariable Integer user_id) {
return incomeService. getIncomeByUser (user_id);
}
@GetMapping («group/{group_id}»)
public List<Income> getIncomesByGroup(@PathVariable Integer group_id) {
return incomeService. getIncomesByGroup (group_id);
}
@GetMapping («user/{user_id}/period»)
public List<Income> findAllByDateBetweenByIdUser(@PathVariable Integer user_id,@RequestParam String start,
@RequestParam String end) {
SimpleDateFormat format = new SimpleDateFormat();
format. applyPattern («yyyy-MM-dd»);
Date startDate = null;
Date endDate = null;
try {
startDate = format. parse (start);
endDate = format. parse (end);
} catch (ParseException e) {
e. printStackTrace();
}
return incomeService. findAllByDateBetweenByIdUser (user_id, startDate, endDate);
}
@GetMapping («group/{group_id}/period»)
public List<Income> findAllByDateBetweenByIdGroup(@PathVariable Integer group_id,@RequestParam String start,
@RequestParam String end) {
SimpleDateFormat format = new SimpleDateFormat();
format. applyPattern («yyyy-MM-dd»);
Date startDate = null;
Date endDate = null;
try {
startDate = format. parse (start);
endDate = format. parse (end);
} catch (ParseException e) {
e. printStackTrace();
}
return incomeService. findAllByDateBetweenByIdGroup (group_id, startDate, endDate);
}
}
```