Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРЕСИВНИЙ WEB-ЗАСТОСУНОК ДЛЯ СПІЛЬНОГО ТА ОСОБИСТОГО УПРАВЛІННЯ ФІНАНСАМИ

Виконав: студент 4 курсу, групи 3КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Киринюк І.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. КН
Сілагін О.В.
(прізвище та ініціали)
«07» 06 2024 р.
Рецензент: к.т.н., доц. каф. САІТ
Жуков С.О.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав.кафедри Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«07» 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Киринюку Іллі Сергійовичу

1. Тема роботи: Прогресивний WEB-застосунок для спільного та особистого управління фінансами.
Керівник роботи: Сілагін Олексій Віталійович, к.т.н., доц.
затверджені наказом вищого навчального закладу "11" 03 2024 року №80

2. Термін подання студентом роботи 27.05.2024

3. Вихідні дані до роботи:
Кількість користувачів на один акаунт – не менше 5 чоловік;
Час відповіді на запит користувача – 3 секунди;
Тип веб-застосунку – прогресивний;
Адаптивний інтерфейс користувача;
Мова програмування – з можливістю реалізації технології прогресивного застосунку;

4. Зміст текстової частини (перелік питань, які потрібно розробити):
Обгрунтування доцільності створення прогресивного WEB-застосунку для спільного та особистого управління фінансами;
Аналіз предметної області з управління особистими фінансами;
Розробка архітектури та проєктування прогресивного WEB-застосунку для спільного та особистого управління фінансами;
Програмна реалізація прогресивного WEB-застосунку для спільного та особистого управління фінансами; Тестування прогресивного WEB-застосунку для спільного та особистого управління фінансами.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
UML-діаграма активності;UML-діаграма послідовностей; Загальний вигляд інтерфейсу користувача; Приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Сілагін О. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обгрунтування доцільності створення прогресивного WEB-застосунку для спільного та особистого управління фінансами.	06.05.24	09.05.24	
2	Аналіз предметної області з управління особистими фінансами.	08.05.24	12.05.24	
3	Розробка архітектури та проєктування прогресивного WEB-застосунку для спільного та особистого управління фінансами.	12.05.24	16.05.24	
4	Програмна реалізація прогресивного WEB-застосунку для спільного та особистого управління фінансами.	16.05.24	23.05.24	
5	Тестування прогресивного WEB-застосунку для спільного та особистого управління фінансами.	23.05.24	27.05.24	
6	Оформлення додатків.	27.05.24	29.05.24	
7	Розробка інструкції користувача.	29.05.24	02.06.24	
8	Оформлення матеріалів до захисту БДР.	02.06.24	06.06.24	

Студент _____ **Киринюк І.С.**
(підпис) *(прізвище та ініціали)*

Керівник роботи _____ **Сілагін О. В.**
(підпис) *(прізвище та ініціали)*

АНОТАЦІЯ

Киринюк І.С. Прогресивний WEB-застосунок для спільного та особистого управління фінансами. Бакалаврська дипломна робота складається з 85 сторінок формату А4, на яких є 36 рисунків, список використаних джерел містить 20 найменувань.

У даній бакалаврській дипломній роботі розглянуто стан справ стосовно проблеми управління фінансами. Проведено аналіз існуючих програм-аналогів, наведено переваги та недоліки, розроблено постановку задачі на дослідження,. Вибрано архітектуру, спроектовано інтерфейс та функціональность. Виконано обгрунтування вибору мови програмування, середовища розробки та фреймворку. Результатом є розроблений прогресивний веб-застосунок для керування фінансами.

Ключові слова: програмування, веб-застосунок, прогресивний веб-застосунок, управління фінансами.

ABSTRACT

Kirinyuk I.S. Progressive WEB-application for joint and personal financial management. The bachelor's thesis consists of 85 pages of A4 format, which contains 36 figures, the list of references contains 20 titles.

In this bachelor's thesis, the state of affairs in relation to the problem of financial management is considered. An analysis of existing analogous programs was carried out, advantages and disadvantages were presented, and a research problem statement was developed. The architecture was chosen, the interface and functionality were designed. The choice of programming language, development environment and framework is justified. The result is an advanced web application for financial management.

Keywords: programming, web application, progressive web application, financial management.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі набуває широкого вжитку використання цифрових рішень замість традиційних. Нещодавні події дуже сильно вплинули на прискорення та впровадження даного процесу, а також відсікла будь які сумніви щодо доречності у їх використанні, адже має позитивний вплив на розвиток економіки та добробут населення . Відбувається етап цифровізації, який також вплинув на рішення проблеми контролю фінансів. Використання програмних засобів для задоволення даної проблеми має ряд переваг, адже це зручно, швидко та завжди під рукою, чи то смартфон, чи комп'ютер або інший цифровий гаджет і це також надавання відразу статистики та усіх необхідних даних, що цікавлять користувача.

Актуальність розробки застосунку для спільного та особистого управління фінансами має неабиякий потенціал та буде мати попит серед користувачів, адже більшість існуючих рішень мають обмеження, які частково вирішуються за додаткову плату, проте комплексного рішення що являє собою повноцінний функціонал, який би хотів мати користувач, не має. А з використанням сучасних технологій це можливо зробити та надати споживачу повноцінний продукт без будь яких додаткових чи прихованих оплат, до того ж створити функції, що не представлені у жодному з популярних рішень.

Метою є розширення функціональних можливостей застосунку для управління фінансами з використанням технології прогресивного WEB-застосунку. Для досягнення поставленої мети необхідно реалізувати завдання що наведені нижче:

1. Обгрунтування доцільності створення прогресивного WEB-застосунку для спільного та особистого управління фінансами;
2. Аналіз предметної області з управління особистими фінансами;
3. Розробка архітектури та проєктування прогресивного WEB-застосунку для спільного та особистого управління фінансами;

4. Програмна реалізація прогресивного WEB-застосунку для спільного та особистого управління фінансами;

5. Тестування прогресивного WEB-застосунку для спільного та особистого управління фінансами.

Об’єктом дослідження є процес управління фінансами, що охоплює як особисті витрати, так і спільні бюджети.

Предметом дослідження є алгоритми та програмні засоби для управління фінансами.

Практична цінність отриманих результатів дослідження:

- Розроблено алгоритм взаємодії прогресивного WEB-застосунку з базами даних Cloud Firestore;
- Реалізовано механізм офлайн-доступу до функцій застосунку, що дозволяє користувачам переглядати дані та вносити нові без постійного підключення до мережі.

Апробація роботи. Роботу апробовано на наступній конференції: LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024) [1].

Публікації. Матеріали одержані в процесі розробки бакалаврської дипломної роботи, опубліковані в тезах конференції “ LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)” [1].

Впровадження. Результати роботи планується до впровадження в розробці торгового комплексу “Корона” і представлені в додатку Д.

1 ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ ДЛЯ СПІЛЬНОГО ТА ОСОБИСТОГО УПРАВЛІННЯ ФІНАНСАМИ

1.1 Аналіз предметної області з управління особистими фінансами

Історія особистих фінансів починається з появою дисциплін, таких як економіка сім'ї та споживча економіка, які викладалися протягом багатьох років. Перше дослідження в цій галузі було проведене Хейзел Кірк у 1920 році, що сприяло розвитку споживчої економіки та економіки сім'ї. Іншими піонерами у вивченні поведінки споживачів та домогосподарств були Маргарет Рід та Герберт Саймон. З моменту створення професійних організацій, таких як Асоціації фінансового консультування та освіти з питань планування та Академії фінансових послуг, вивчення особистих фінансів стало більш систематичним. Кілька американських університетів також запровадили освітні програми з фінансів, залучивши багато уваги до цієї галузі. Після фінансової кризи 2008 року була створена Консультативна рада з питань фінансової спроможності, щоб сприяти фінансовій грамотності серед населення. Важливим завданням стала розробка стандартів фінансової освіти [2].

Особисті фінанси є важливою частиною у житті людини для забезпечення фінансової стабільності, а також для досягнення фінансових цілей. Особисті фінанси - це те, що охоплює керування особистими коштами, включно з інвестиціями та заощадженнями. Він охоплює бюджетування, банківську справу, страхування, іпотеку, інвестиції, а також пенсійне, податкове та спадкове планування. Часто цим терміном позначають цілу індустрію, у якій надають фінансові послуги приватним особам і домогосподарствам, консультують їх про фінансові та інвестиційні можливості [3].

У сучасному світі дедалі більше зростає проблема в управлінні коштами, що призводить до накопичення величезної кількості боргів. Наприклад, у лютому 2024 Федеральний резервний банк – регіональний банк в Сполучених Штатах Америки,

надав звіт [4], що представлений у вигляді графіків, один з яких наведено на рисунку 1.1. По графіку видно зростаючу тенденцію з кожним роком, а також щодо більш детального опису, то борг домогосподарств збільшився на $3,4 трлн з грудня 2019 року, тобто до початку рецесії. Крім того, з третього кварталу 2023 року до четвертого збільшилися такі залишки:

- Залишки на кредитних картах: на 50 мільярдів доларів;
- Автокредити: на $12 млрд;
- Споживчі кредити та магазинні картки: зросли на $25 млрд;
- Загальне нежитлове кредитування: на 89 мільярдів доларів;
- Іпотечні кредити: на 112 мільярдів доларів;
- Студентські кредити залишилися незмінними, на рівні близько $1,6 трлн.

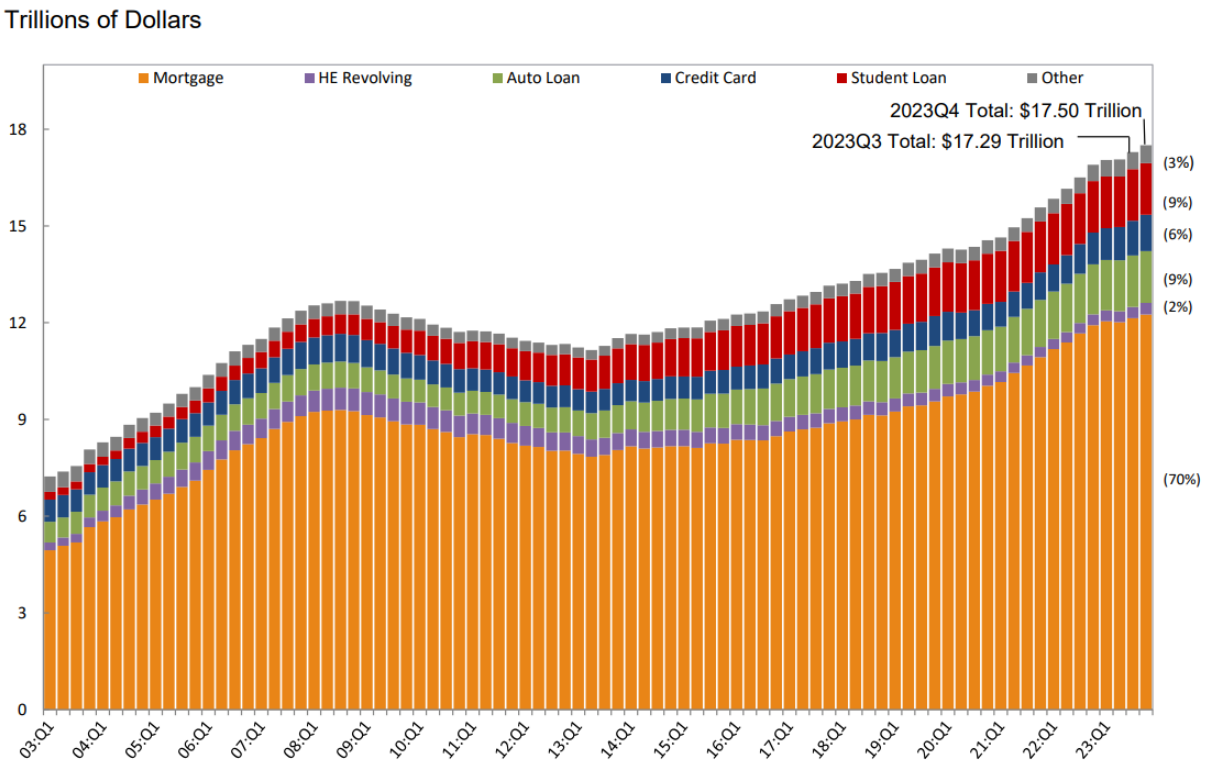


Рисунок 1.1 - Федеральний резервний банк Нью-Йорка. Квартальний звіт про борги та кредити домогосподарств за 2023 рік

Як висновок, американці беруть дедалі більше боргів та кредитів для задоволення своїх потреб, що вкотре підкреслює необхідність в управлінні

особистими фінансами, а також на фоні інфляції, що з’їдає купівельну спроможність та також спричиняє за собою зріст цін.

Зазвичай виділяють п’ять сфер особистих фінансів, це – дохід, витрати, заощадження, інвестиції та захист [5]. Коротко про кожен з них:

- Доходи – основа, або ж, стартова лінія у контексті особистих фінансів. Це вся сума грошових надходжень, яку ви отримуєте, виконуючи роботу чи отримання дивідендів або будь які інші джерела. Які потім розподіляють на чотири інших сфери.

- Витрати – відтік грошових надходжень, зазвичай те, на що йде основна частина доходу. До неї можна віднести і оренду, і комунальні послуги, і розваги, тобто будь яку діяльність яка безповоротно забирає грошовий дохід. Саме вміти керувати адекватно та правильно витратами – є найголовнішим завданням у керуванні фінансами. Адже, якщо витрати будуть переважати дохід, то це призводить до оформлення кредитів, взяття у борг і тому подібне, що має ганебний вплив на фінансове становище людини.

- Заощадження – по суті, це дохід, що залишився після витрат. Кожен повинен прагнути мати хоч якусь частину заощаджень, рекомендується мати так звану “фінансову подушку безпеки” – це певна кількість заощаджень, що здатні перекрити витрати періодом 3, 6 або 12 місяців життя. Зазвичай використовують для перекриття непередбачуваних витрат, втрати доходу, проблем зі здоров’ям, або ж якась інша кризова ситуація. Розраховується або для індивідуальної або для сімейної потреби. Великі заощадження, окрім подушки, мають здебільшого негативний наслідок для утримувача, адже вони будуть втрачати свою цінність і тому потрібно їх, наприклад, інвестувати.

- Інвестування – це придбання активів, переважно такі як акції та облігації, з метою прибільшення прибутку. Ціллю є отримання більшої суми, ніж була вкладена, тим самим змусюючи “працювати” гроші на себе. Так, ця сфера має певні ризики та складнощі, проте воно повинно мати сенс у кожного.

- Захист – це порядок дій, спрямованих на захист себе від нещасних випадків пов’язаних з різними сферами життя. Тим самим забезпечуючи свою фінансову стабільність.

Одним із популярних методів складання бюджету є правило 50/30/20. Правило 50/30/20 - це техніка бюджетування, яка передбачає розподіл грошей на три основні категорії, виходячи з вашого доходу після сплати податків (тобто вашої зарплати): 50% - на потреби, 30% - на бажання і 20% - на заощадження та виплати боргів. Це правило було всебічно висвітлено тодішнім професором (нині сенатором) Елізабет Уоррен та її донькою Амелією Уоррен Тьягі у книзі "Все, чого ви варті: Кінцевий грошовий план на все життя" [6]. Детальніше про кожну категорію правила:

- 50% потреби – це витрати, що пов’язані з базовою потребою людини для виживання та належного добробуту. Це такі повсякденні витрати як проїзд у громадському транспорті чи просто транспорт, комунальні послуги, харчування, охорона здоров’я, тощо.
- 30% бажання – це також витрати, проте такі що роблять життя комфортнішим, але відсутність яких не вплине суттєво на життя. У кожного це індивідуальне, наприклад, похід у кафе, кінотеатр або ж придбання підписки на сервіси чи шопінг, тощо.
- 20% заощадження – це те що залишилось після потреб і бажань, і те що зазвичай відкладають на погашення боргів (якщо такі є) або ж на певну ціль.

Отже, потрібно якийсь інструмент, який би дозволив контролювати сфери особистих фінансів та подальший розподіл, наприклад, за розглянутим вище правилом.

1.2 Аналіз існуючих програмних рішень для управління фінансами

Є певна кількість програм, сайтів та мобільних додатків, що дозволяють виконати задачу керування фінансами та подальшій аналітиці, проте ці програмні продукти мають як переваги, так і недоліки. Розглянемо декілька відомих рішень,

що набули широкого вжитку посеред користувачів та мають непогані відгуки та високі оцінки в магазинах програм у порівнянні з конкурентами.

“Бюджет: облік витрат, планування”[7] – один із популярних додатків для мобільних пристроїв, але лише ті що працюють на платформі Android, що вже є надзвичайно великим недоліком у контексті доступності та кросплатформеності. Проте із переваг, це наявність україномовного інтерфейсу, а також синхронізації між будь якою кількістю пристроїв, проте за платну підписку. Вигляд додатку зображено на рисунку 1.2.

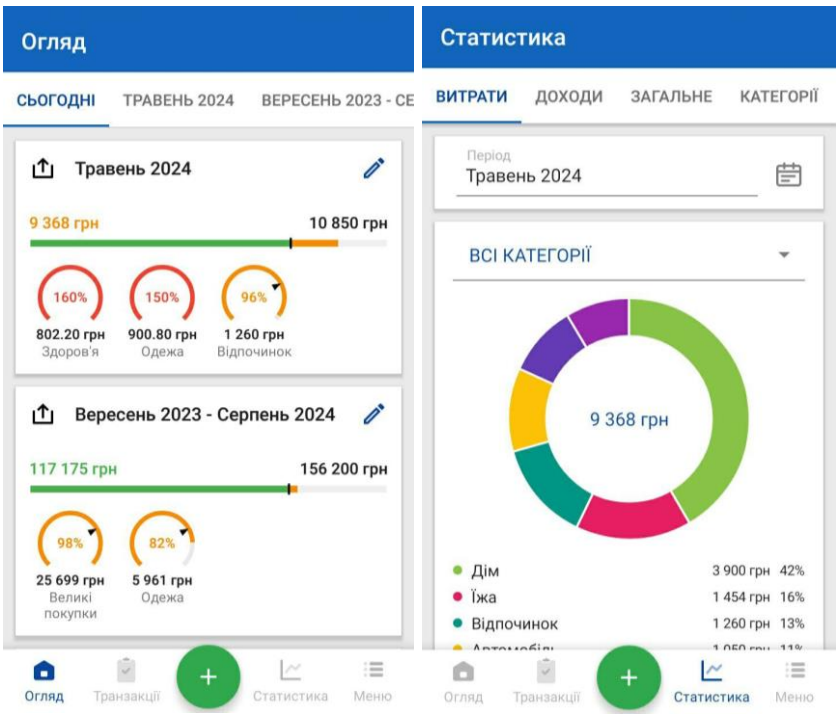


Рисунок 1.2 – Знімок екрана вигляду програми “Бюджет: облік витрат, планування” на Android пристрої

“Money Pro”[8] – програмний продукт, що має найбільшу різноманітність інструментів, що дозволяє ефективно виконувати контроль фінансів. Він доступний на усіх популярних платформах, як окремо встановлювана програма. Із недоліків – майже увесь функціонал за підписку, включаючи синхронізацію та спільний контроль фінансів, також нещодавні відгуки вказують на проблему синхронізації у

користувачів та втрату даних, що не можна повернути. Вигляд програми на платформі Windows зображено на рисунках 1.3-1.4.

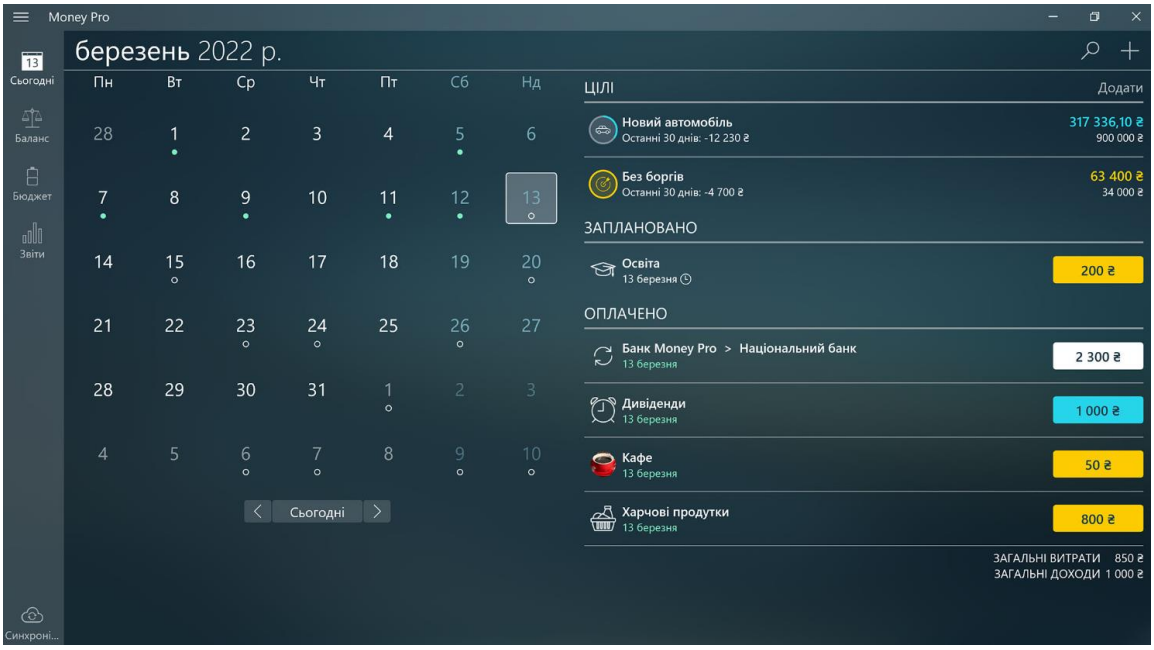


Рисунок 1.3 - Знімок екрана вигляду вкладки “Сьогодні ” програми “Money Pro” на комп’ютері

Рисунок 1.4 - Знімок екрана вигляду вкладки “Бюджет ” програми “Money Pro” на комп’ютері

“Goodbudget”[9] – сумнівний додаток із точки зору українського користувача через відсутність україномовного інтерфейсу та місцевої валюти, проте додаток згадується у багатьох виданнях як один із хороших фінансових менеджерів. Проте з останніх відгуків помітно, що користувачі незадоволені застрілим та інтуїтивно незрозумілим інтерфейсом (рис. 1.5). Також є жорсткі обмеження стосовно кількості девайсів та акаунтів навіть з підпискою.

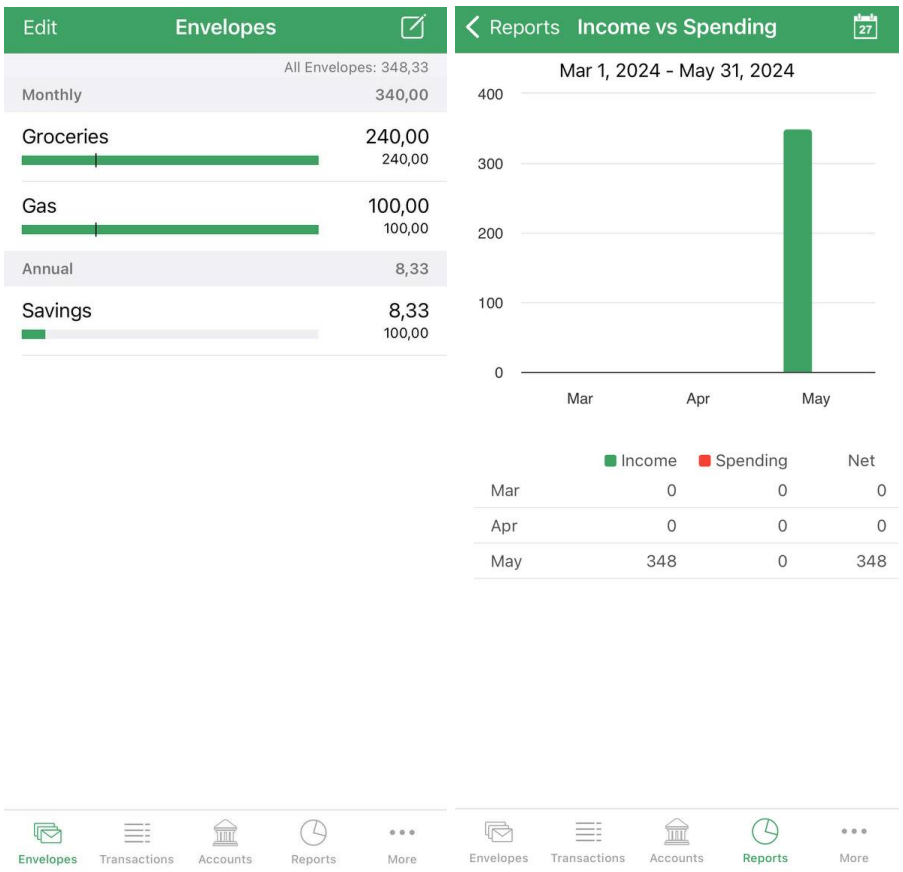


Рисунок 1.5 – Знімок екрана вигляду програми “ Goodbudget ” на IOS пристрої

“Monefy”[10] – додаток, що має дуже багато завантажень, а також приємний користувацький інтерфейс (рисунок 1.6). Щодо переваг – то це синхронізація між пристроями, а також веб-сайт, що не було у попередньо розглянутих аналогів, а також цікаву систему розподілу фінансів між іншими учасниками. Із недоліків – немає локалізації і також з кожним оновленням безкоштовна версія втрачає свій функціонал.

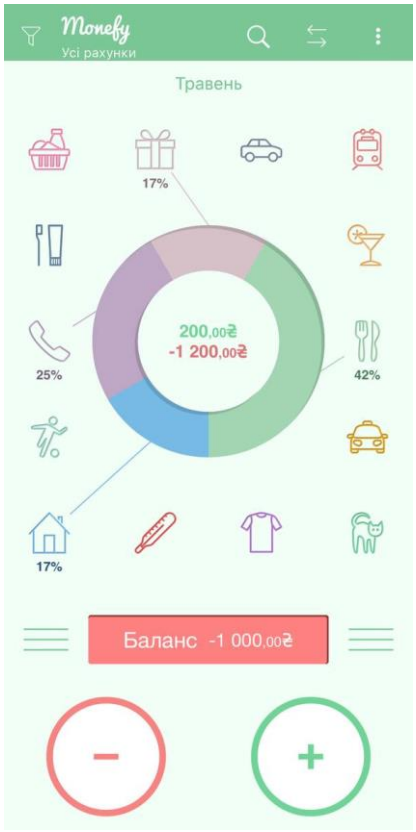


Рисунок 1.6 - Знімок екрана вигляду програми “ Monefy” на Android пристрої

Із розглянутих подібних програмних рішень можна дійти висновку, що кожна програма володіє різними функціональними можливостями, що впливають на вибір користувача. Одним суттєвим недоліком для усіх них є відсутність роботи у офлайн режимі, і це веде до того що користувач не завжди буде мати змогу виконати запис операції чи переглянути дані без підключення до мережі, тим самим може взагалі забути вписати свої витрати чи дохід. Також із ключевих моментів, це те що лише один додаток має підтримку у браузері та працює як веб-сайт, тоді як усі інші працюють лише на мобільних пристроях й потребують окремого завантаження, а також синхронізації, що зазвичай присутня лише за кошти.

1.3 Постановка задачі на дослідження

Із розгляду аналогів можна виділити ключові особливості та функції, що мають бути обов’язково присутні у розроблювальному застосунку, а саме

синхронізація, кросплатформеність, доступність – що були не задовільними у розглянутих програмах. І саме доречність використання технології Progressive Web App дозволить закрити усі потреби. Адже прогресивні WEB-застосунки здатні забезпечити високий рівень доступності на різних пристроях без окремого встановлення програм, що вже є значною перевагою та стимулом до використанням саме такого шляху реалізації. Також спільний контроль фінансів і синхронізація пристроїв може бути як базова інтегрована функція у Progressive Web App, що надасть вільний доступ користувачеві з будь якого пристрою. А також офлайн режим, що також є одним із головних переваг саме цієї технології, який дозволить використовувати програму без підключення мережі Інтернет, при чому без будь яких обмежень у роботі з базою даних.

Підбиваючи підсумки, застосунок повинен мати наступне:

1. Додавання доходу та витрат. Користувач може легко додавати записи про доходи та витрати через відповідні форми з можливістю вказувати суму та категорію транзакції.

2. Зберігання усієї історії транзакцій. Уся історія фінансових операцій зберігається в базі даних, яку користувач може переглядати, з такими вказівками як дата, категорія, тип (дохід/витрати).

3. Детальна аналітика доходів та витрат. Автоматичний підрахунок та відображення загальної суми доходів та витрат за певний період. Графіки та діаграми для візуалізації фінансових даних. Розподіл витрат на категорії для більш детального аналізу.

4. Інтуїтивно зрозумілий інтерфейс для користувача. Простий та зрозумілий дизайн, який легко освоїти новим користувачам. Логічне розташування елементів управління та швидкий доступ до основних функцій.

5. Застосунок адаптується до будь-якого розміру екрану, забезпечуючи зручне використання на смартфонах, планшетах та комп'ютерах.

6. Режим офлайн та можливість встановлення на пристрої. Застосунок повинен працювати без доступу до Інтернету, зберігаючи дані локально, як це передбачено для прогресивних WEB-застосунків.

Отже, у роботі потрібно реалізувати застосунок, що надаватиме можливість юзеру виконувати менеджмент своїх фінансів із певною аналітикою. Він повинен мати змогу виконувати необхідні операції, такі як внесення витрат або доходу, що є вхідними даними, та отримати на основі цього історію транзакцій, загальний баланс, а також аналітичні дані на основі введення, які можна буде відфільтрувати за певний період.

Додатково реалізувати список бажань, або ж цілей, відображеного у вигляді карточок, які можна редагувати, змінюючи при цьому дані щодо суми накопичення, та добавляти і відповідно відобразити для наочності шкалу прогресу, що показує процент накопиченої суми відносно загальної.

1.4 Висновки до розділу 1

У даному розділі проведено аналіз предметної області з управління особистими фінансами, розглянуто історію розвитку дисципліни, починаючи з робіт таких піонерів, як Хейзел Кірк, Маргарет Рід та Герберт Саймон, так і до сьогодення. Особисті фінанси охоплюють бюджетування, інвестиції, заощадження, страхування та податкове планування. У сучасному світі зростає боргове навантаження на домогосподарства, що підтверджує актуальність ефективних інструментів для фінансового планування та контролю. Проаналізовано основні сфери особистих фінансів і правило бюджетування 50/30/20.

Розглянуто існуючі програмні рішення для управління особистими фінансами, такі як "Бюджет: облік витрат, планування", "Money Pro", "Goodbudget" та "Monefy". Виявлено, що більшість із них мають обмеження та суттєві недоліки.

На основі аналізу поставлено задачу на дослідження: розробити новий застосунок для управління особистими фінансами з використанням технології прогресивного веб-застосунку, який має включати додавання доходів та витрат, зберігання історії транзакцій, детальну аналітику, інтуїтивно зрозумілий інтерфейс, адаптивний дизайн, офлайн-режим та синхронізацію між пристроями. Додатково буде реалізовано функцію списку бажань або цілей.

2 РОЗРОБКА АРХІТЕКТУРИ ТА ПРОЄКТУВАННЯ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ ДЛЯ СПІЛЬНОГО ТА ОСОБИСТОГО УПРАВЛІННЯ ФІНАНСАМИ

2.1 Вибір архітектури WEB-застосунку

Вибір правильної архітектури є критично важливим аспектом під час розробки будь-якого програмного забезпечення, включно з веб-застосунками та мобільними додатками. Правильно спроектована архітектура закладає міцний фундамент для подальшого розвитку та масштабування проекту, забезпечуючи ефективність, гнучкість та довговічність рішення. Архітектура визначає загальну структуру системи, розподіл компонентів, їхню взаємодію та принципи роботи. Це своєрідний каркас, на якому будується увесь додаток. Від якості цього каркаса залежить, наскільки легко буде додавати нові функції, розширювати можливості, інтегрувати з іншими системами та масштабувати продукт у майбутньому.

Щоб веб-застосунок був дійсно корисним та ефективним, він повинен бути розроблений з використанням правильного архітектурного підходу. У React проектах доцільно використовувати архітектурний підхід, заснований на концепції компонентів. Компоненти є самостійними, автономними блоками коду, кожен із яких відповідає за певну логіку та візуальне відображення. Рекомендовані підходи для структурування такого проєкту:

- Розбиття на компоненти за функціональністю: необхідно розділити додаток на невеликі перевикористовувані компоненти, кожен з яких відповідає за певну функціональність. Таким чином, вони легко підтримуються та можуть повторно використовуватися в різних частинах програми.
- Ієрархічна структура компонентів: впорядкувати компоненти у ієрархічну структуру, де батьківські компоненти містять або "рендерять" дочірні компоненти. Це допомагає організувати передачу даних та управління станом.

- Розподіл логіки та відображення: розділити компоненти на дві частини: функціональну логіку та відображення. Це можна зробити, використовуючи підхід "контейнерів" і "презентаційних" компонентів.
- Модульний імпорт/експорт: використовувати механізм ES6 модулів для імпорту/експорту компонентів, допоміжних функцій та констант. Це дозволяє краще структурувати код і полегшити повторне використання.
- Використання React Router: Для додатків з декількома сторінками використовувати React Router для навігації та структурування роутингу в програмі.
- Управління станом: для керування станом додатка можна використовувати бібліотеки, такі як Redux, MobX або Context API з React Hooks. Вони допомагають централізувати та спростити роботу зі станом.
- Структура каталогів: організовувати каталоги відповідно до типу файлів або функціональних областей (компоненти, сервіси, утиліти тощо). Такий підхід полегшує пошук та підтримку коду.

Крім того, дотримання принципів компонентного підходу, таких як повторне використання, інкапсуляція та поділ проблеми, допоможе створити кращу архітектуру додатка React. Це сприятиме легшому масштабуванню, підтримці та розширенню проекту в майбутньому.

Для роботою з базою даних, використовувати REST - це архітектурний підхід для розробки веб-додатків та інтерфейсів прикладного програмування. Його ідея полягає у побудові системи на основі передачі ресурсів із представницьким станом. Цей термін було введено Роєм Філдінгом у 2000 році в його дисертаційній роботі. З того часу REST став одним з найпопулярніших способів створення веб-інтерфейсів. Веб-сервіс, що відповідає архітектурному стилю REST, називається REST API, або RESTful API [11].

REST не є протоколом чи стандартом, а радше архітектурним стилем. Під час розробки API розробники можуть реалізувати принципи REST різними шляхами. Однак, для того, щоб інтерфейс вважався RESTful, він повинен дотримуватись певних керівних принципів та обмежень, притаманних цьому архітектурному

стилю. Ці принципи необхідно враховувати при проектуванні та реалізації веб-сервісів. Усього цих принципів шість:

1. Єдиний інтерфейс – даний принцип загальності спрощує архітектуру системи та покращує видимість взаємодій.

2. Клієнт-Сервер – клієнт-сервер розділяє інтереси, дозволяючи клієнту і серверу розвиватися незалежно, покращуючи портативність і масштабованість.

3. Безстанність – кожен запит має містити всю необхідну інформацію, оскільки сервер не зберігає попередній контекст.

4. Кешування – відповіді повинні бути помічені як кешовані або не кешовані, що дозволяє повторно використовувати відповідні дані для подібних запитів.

5. Багатошарова система - архітектура складається з ієрархічних шарів, де кожен компонент взаємодіє лише з найближчим шаром.

6. Код на вимогу (необов'язково) – REST дозволяє завантажувати і виконувати код на стороні клієнта, зменшуючи кількість попередньо реалізованих функцій.

Багато розробників React поєднують REST API для основних CRUD операцій та GraphQL для більш складних випадків використання, які вимагають ефективного отримання та маніпулювання даними.

Отже, REST API залишається актуальним варіантом для React проектів, особливо для простих або традиційних веб-додатків. Вибір підходу залежить від вимог проекту, наявних ресурсів та досвіду команди розробників. Більше того, REST забезпечує незалежність від конкретної мови програмування чи платформи, що дає розробникам свободу вибору найкращих інструментів. Стандартизований інтерфейс на основі HTTP методів робить взаємодію між клієнтом та сервером інтуїтивно зрозумілою. А безстановий підхід та використання URI для ідентифікації ресурсів полегшує кешування та контроль над станом додатку.

Нарешті, REST сприяє чіткому розподілу обов'язків між клієнтською та серверною частинами, спрощуючи розробку та тестування компонентів окремо. Це особливо важливо для великих проектів, таких як прогресивний веб-застосунок для спільного та особистого управління фінансами.

Отож, з використанням даних архітектур, буде отримано фінансовий додаток, який легко масштабується для обслуговування зростаючої кількості користувачів, працює однаково добре на різних платформах та пристроях, забезпечує швидку обробку транзакцій та ефективний обмін даними.

2.2 Проектування інтерфейсу та функціональності WEB-застосунку

Розробка прогресивного веб-застосунку (PWA) для управління фінансами на React буде розбита на кілька компонентів, розподілених між фронтендом (клієнтською частиною) і бекендом (серверною частиною), з використанням окремих модулів для взаємодії з базою даних та сервіс-воркером для реалізації PWA-функціоналу. Розглянемо кожен компонент і його функціональність.

Фронтенд (Клієнтська частина):

1. Головна сторінка:

- Компонент балансу. Функціонал: можливість додавати нові транзакції, які представляють собою доходи або витрати. Вибір категорії для кожної транзакції. Реалізація: форма для введення даних транзакції, кнопка для збереження нової транзакції, відображення поточного балансу користувача.
- Компонент транзакцій. Функціонал: відображення списку усіх транзакцій, які ввів користувач та можливість перегляду деталей кожної транзакції. Реалізація: відображення списку транзакцій у вигляді таблиці або списку.
- Компонент списку бажань. Функціонал: додавання нових бажань та встановлення цілей збереження і можливість відслідковувати прогрес досягнення цілей. Реалізація: форма для додавання нового бажання і відображення списку бажань з можливістю встановлення цілей.
- Компонент фонового зображення. Функціонал: можливість змінити вигляд головної сторінки, вибравши іншу тему або стиль. Реалізація:

вибір доступних обгорток або можливість завантаження власної і застосування обраної обгортки до інтерфейсу.

- Компонент статистики. Функціонал: відображення графіків та діаграм для аналізу фінансових даних. Реалізація: використання бібліотек для побудови графіків (наприклад, Chart.js або D3.js).

2. Статистика:

- Компонент статистики. Функціонал: відображення графіків та діаграм для аналізу фінансових даних за певний період та фільтрація даних за допомогою діапазону дат. Реалізація: форма для вибору діапазону дат та відображення різних графіків на основі вибраних фільтрів.

3. Налаштування:

- Компонент налаштування. Функціонал: зміна особистих даних користувача, а також зміна налаштувань, таких як фонове зображення на головній сторінці або баланс. Реалізація: форми для введення нових даних користувача.

Бекенд (Серверна частина):

1. Модуль для взаємодії з базою даних. Функціонал: реалізація CRUD (створення, читання, оновлення, видалення) операцій з базою даних.

2. Сервіс-воркер для PWA. Функціонал: реалізація кешування, щоб забезпечити офлайн доступ до додатку та обробка подій мережі і оновлення кешування для покращення продуктивності.

Ці компоненти можна реалізувати з використанням бібліотеки React разом з іншими технологіями. Така архітектура дозволяє ефективно організувати код та забезпечити швидку роботу додатку як на клієнтському, так і на серверному рівні, а також надає можливість розширення функціоналу в майбутньому.

Використання Firestore Database для прогресивного веб-застосунку є доцільним через кілька ключових аспектів, зокрема можливість роботи в офлайн режимі, яку забезпечує бібліотека firebase/firestore. Firestore є сучасною хмарною базою даних від Firebase, яка надає розробникам потужні інструменти для створення масштабованих і надійних веб-застосунків. Однією з основних переваг

Firestore є її вбудована підтримка офлайн режиму. Це означає, що користувачі можуть продовжувати взаємодіяти з веб-застосунком навіть без підключення до інтернету. Ця бібліотека зберігає дані локально на пристрої користувача, дозволяючи виконувати CRUD операції (створення, читання, оновлення, видалення) без з'єднання з мережею. Коли з'єднання відновлюється, вона автоматично синхронізує локальні зміни з сервером, що забезпечує узгодженість даних і безперебійний користувацький досвід.

Ця функціональність особливо важлива для розроблюваного застосунку, що повинен забезпечувати високу продуктивність та доступність незалежно від стану мережі. Завдяки Firestore, є можливість створювати застосунки, які працюють плавно і без збоїв навіть у випадках тимчасової втрати з'єднання. Це підвищує зручність для кінцевих користувачів, які можуть працювати з даними, заповнювати форми або переглядати контент без перерв.

Крім того, Firestore надає можливості реального часу для синхронізації даних між користувачами, що є ще однією суттєвою перевагою для PWA. Це дозволяє створювати інтерактивні та динамічні веб-застосунки з оновленням даних у режимі реального часу, що значно покращує взаємодію користувача із застосунком.

Ще однією важливою перевагою Firestore є його безшовна інтеграція з іншими сервісами Firebase, такими як аутентифікація, хмарні функції та аналітика. Це забезпечує єдину екосистему для розробки, що спрощує керування застосунком та його розвиток [13].

Таким чином, використання Firestore Database для прогресивного веб-застосунку є доцільним рішенням завдяки його можливостям офлайн режиму, синхронізації даних у реальному часі та інтеграції з іншими інструментами Firebase.

2.3 Розробка UML діаграм WEB-застосунку

UML (Unified Modeling Language) - це уніфікована мова моделювання, призначена для створення абстрактних моделей систем. Вона є відкритим

стандартом і застосовується для визначення, візуалізації, проєктування та документування програмних систем [12].

Користуючись різними типами UML діаграм, можна краще розібратись у структурі та функціональності веб-застосунку. Ось як кожен тип діаграми може бути застосований до веб-застосунку:

- Діаграма варіантів використання (Use Case Diagram) – дозволяє описати, які функції доступні для користувачів та як вони взаємодіють з системою. Це допоможе зрозуміти потреби користувачів та визначити основні функції веб-застосунку.
- Діаграма класів (Class Diagram) – відображає структуру системи, включаючи класи, їх атрибути та взаємозв'язки. Це допоможе організувати код та розуміти логіку програми.
- Діаграма послідовності (Sequence Diagram) – показує послідовність обміну повідомленнями між об'єктами системи. Це особливо корисно для розуміння взаємодії між різними компонентами системи, такими як фронтенд і бекенд.
- Діаграма активності (Activity Diagram) – описує послідовність дій, які виконуються в системі. Це допомагає зрозуміти процеси та переходи між ними, такі як обробка запитів користувачів та оновлення даних.
- Діаграма компонентів (Component Diagram) – відображає фізичну структуру системи та залежності між її компонентами. Це допомагає в організації та розумінні архітектури веб-застосунку, зокрема визначення фронтенду, бекенду, бази даних та інших компонентів.

Діаграма активності, що наведена на рисунку 2.1, ілюструє різні кроки, які користувач може виконати у системі фінансового управління, а також наслідки цих дій на систему.

Користувач відкриває додаток і система перевіряє, яку дію він хоче виконати. Якщо користувач хоче переглянути свої транзакції, система відображає список доступних транзакцій. У випадку, якщо користувач бажає додати нову транзакцію, він вводить необхідні дані, і система зберігає цю транзакцію у базі даних.

Якщо користувач бажає побачити статистику своїх фінансів, система відображає статистичні дані, такі як графіки доходів і витрат. І, нарешті, якщо користувач бажає керувати своїм списком бажань, система дозволяє йому переглядати і змінювати цей список, наприклад, додаючи або видаляючи бажання.

Ця діаграма не лише показує послідовність дій користувача, але й відображає, як ці дії впливають на систему і як система реагує на дії користувача, забезпечуючи відповідну функціональність та взаємодію.

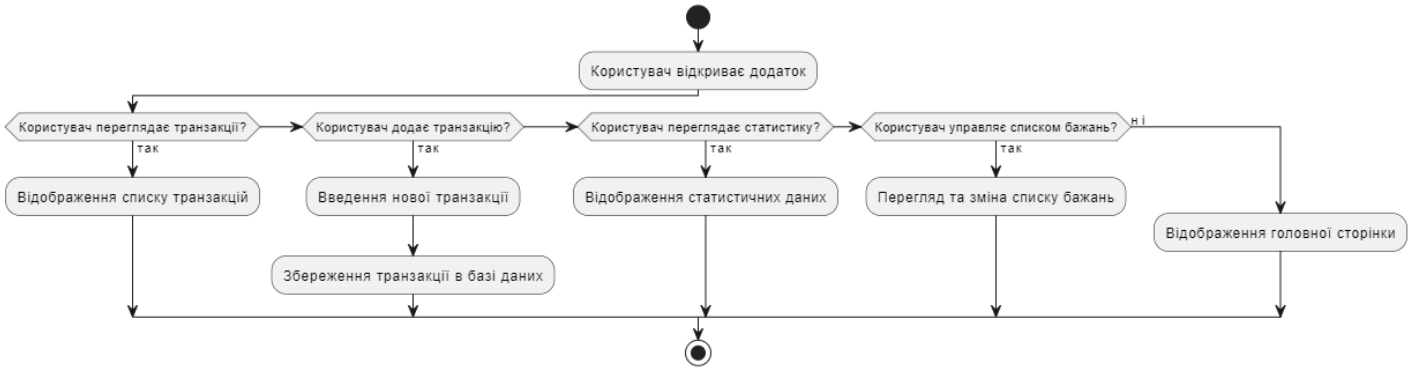


Рисунок 2.1 – UML-діаграма активності

Діаграма послідовності на рисунку 2.2 показує взаємодію компонентів системи фінансового управління при виконанні дій користувача.:

1. Користувач відкриває веб-застосунок і переглядає список транзакцій: веб-застосунок запитує сервер, сервер звертається до бази даних і повертає дані для відображення.

2. Користувач додає нову транзакцію: вводить дані, веб-застосунок надсилає їх на сервер, сервер зберігає в базі даних і повідомляє про успіх, веб-застосунок інформує користувача.

3. Користувач переглядає статистику: відкриває вкладку, веб-застосунок запитує сервер, сервер отримує дані з бази і повертає їх, веб-застосунок відображає статистику.

4. Користувач управляє списком бажань: відкриває вкладку, веб-застосунок запитує сервер, сервер звертається до бази даних і повертає дані, веб-застосунок відображає список бажань.

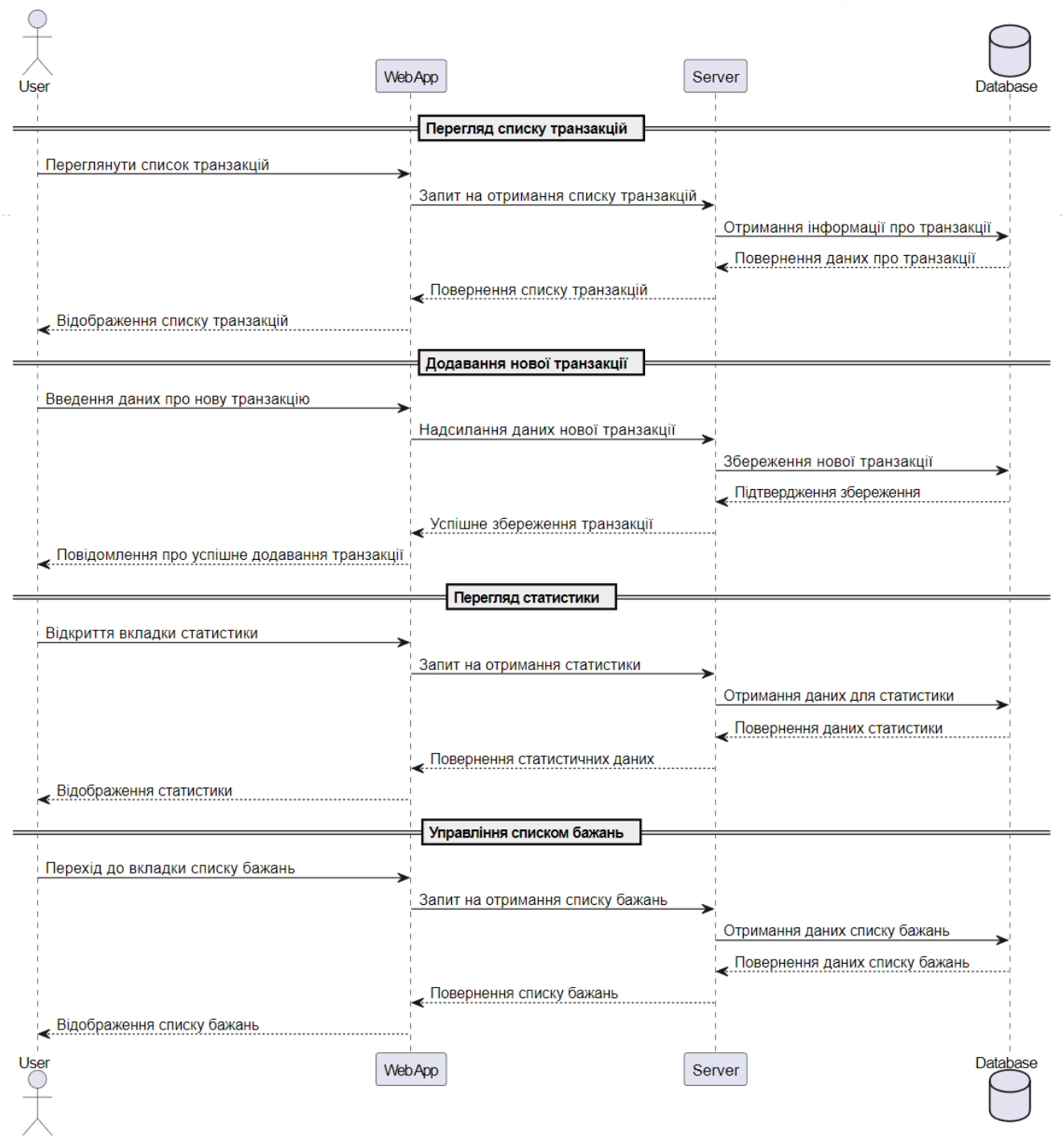


Рисунок 2.2 UML-діаграма послідовностей

2.4 Висновки до розділу 2

У даному розділі було здійснено вибір архітектури для прогресивного веб-застосунку, зосереджуючись на підході, заснованому на компонентах, а також на архітектурному стилі REST. Правильний вибір архітектури має ключове значення для успіху проекту, адже від цього залежить не лише ефективність роботи, але й можливість подальшого розвитку, інтеграції та масштабування додатку.

Проектування інтерфейсу та функціональності веб-застосунку включає детальний опис основних компонентів програми, їхнього призначення та способів реалізації. Фронтенд частина складається з кількох ключових сторінок: головної, статистики та налаштувань, кожна з яких містить специфічні компоненти, що забезпечують основні функції користувача, такі як додавання та перегляд транзакцій, управління списком бажань, аналіз фінансових даних та налаштування персональних параметрів. Бекенд частина для взаємодії з базою даних та реалізації функціоналу прогресивного веб-застосунку за допомогою сервіс-воркера, який забезпечує офлайн доступ та покращення продуктивності. Використання Cloud Firestore є обґрунтованим вибором для прогресивного веб-застосунку завдяки його підтримці офлайн режиму, можливостям синхронізації даних у реальному часі та інтеграції з іншими сервісами Firebase.

Для кращого розуміння структури та функціональності веб-застосунку було використано UML діаграми. Діаграма активності ілюструє різні кроки, які користувач може виконати у системі, та наслідки цих дій на систему, а діаграма послідовностей показує взаємодію компонентів при виконанні різних дій користувача.

Загалом, обраний підхід до архітектури, ретельне проектування інтерфейсу та функціональності, а також побудова UML діаграм закладають міцний фундамент для створення ефективного, гнучкого та масштабованого прогресивного веб-застосунку для управління фінансами. Це забезпечить не лише комфортне користування для кінцевих користувачів, але й полегшить підтримку та розвиток проекту в майбутньому.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ ДЛЯ СПІЛЬНОГО ТА ОСОБИСТОГО УПРАВЛІННЯ ФІНАНСАМИ

3.1 Обгрунтування вибору мови програмування

Розробка веб-застосунку потребує ретельного вибору мови програмування та середовища розробки, оскільки це безпосередньо впливає на ефективність, масштабованість та зручність використання кінцевого продукту. Також необхідно враховувати вимоги та особливості реалізації застосунку, щоб його розробка на обраній мові та у відповідному середовищі була здійсненною.

До розгляду обрано три мови програмування, що підходять для реалізації застосунку: JavaScript, Python, та Java. Вибір цих мов зумовлений їхньою популярністю та надійністю для розробки веб-застосунків.

Згідно з щорічним опитуванням DOU "Рейтинг мов програмування 2024" [14], ці три мови є лідерами серед IT-спеціалістів з України (рис. 3.1). JavaScript, хоча й залишається лідером серед інших мов програмування, проте показує тенденцію до зменшення використання в останні роки через стрімкий ріст популярності TypeScript. Цей спад можна пояснити тим, що TypeScript, як надмножина JavaScript, пропонує додаткові можливості для типізації, що підвищує надійність коду. Java також демонструє зниження популярності, але все ще залишається важливою мовою для багатьох корпоративних застосунків. На відміну від Java та JavaScript, Python, після періоду зниження популярності, знову набуває значного поширення серед розробників завдяки своїй простоті, універсальності та широкому спектру застосувань (рис. 3.2). Значне зростання популярності Python також пов'язане з його широким використанням у сфері штучного інтелекту та машинного навчання. Нижче коротко розглянуто кожну з них.

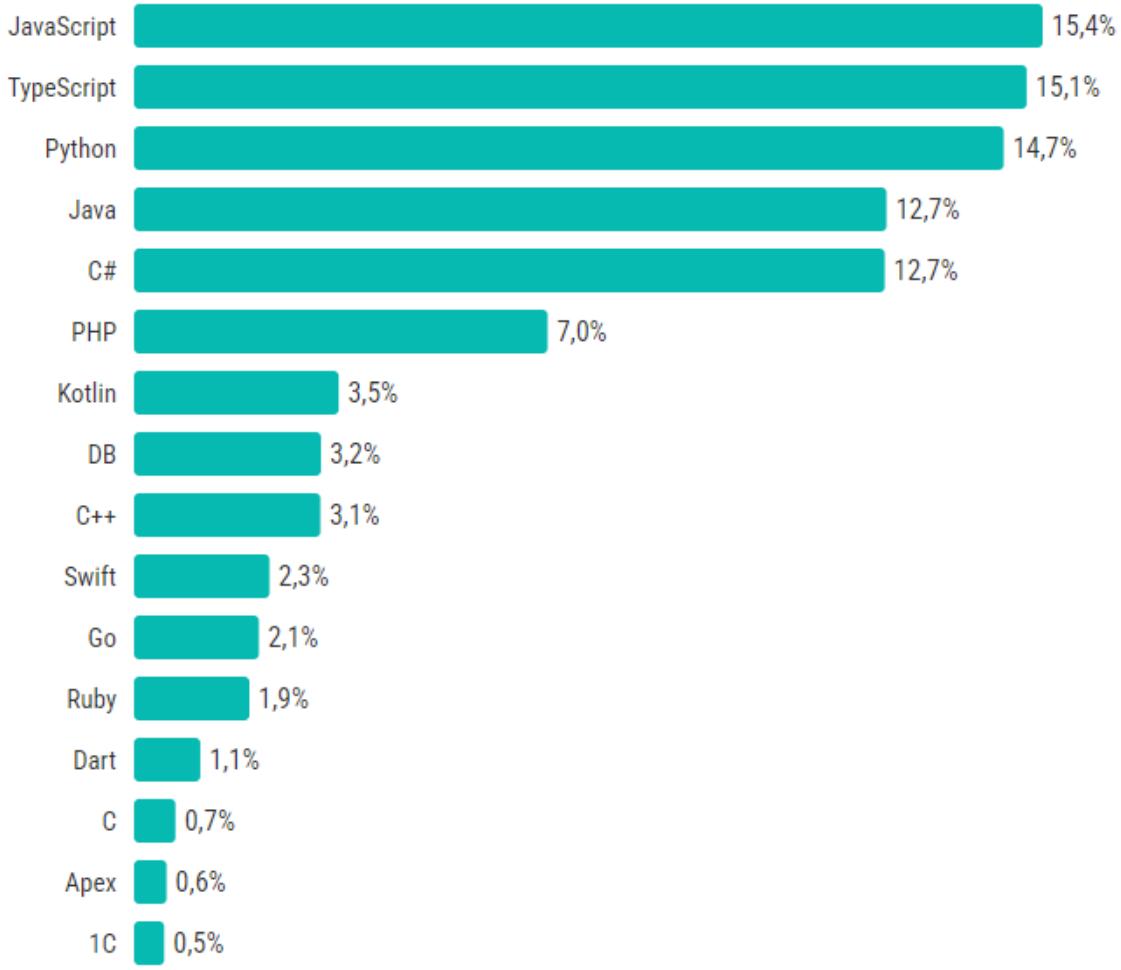


Рисунок 3.1 – Рейтинг мов програмування DOU станом на 2024 рік

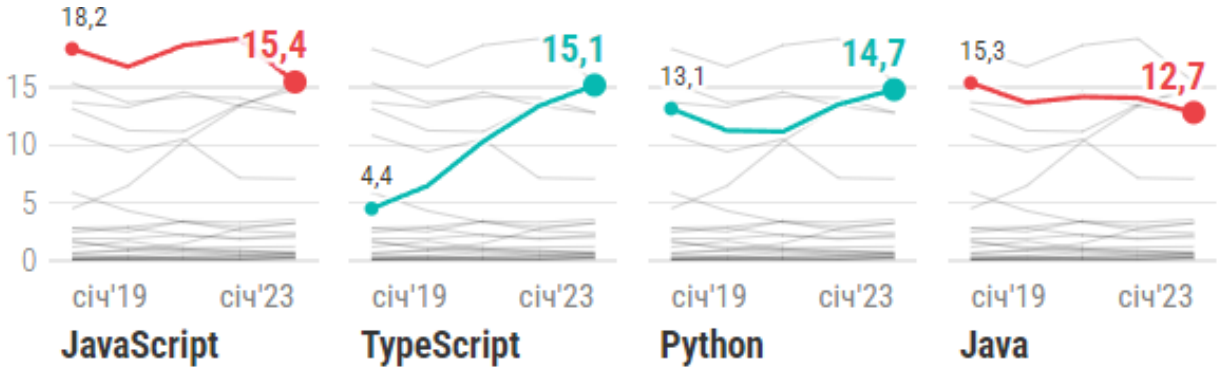


Рисунок 3.2 – Тенденція використання мов програмування згідно опитування DOU за 2019-2023 роки

JavaScript є одним з найпопулярніших інструментів для реалізації веб-застосунків завдяки своїй універсальності та широким можливостям. Використовуючи JavaScript, розробники можуть створювати динамічні й

інтерактивні інтерфейси користувача, оскільки ця мова виконується безпосередньо в браузері. Крім того, завдяки платформі Node.js, JavaScript може використовуватися і на стороні сервера, що дозволяє створювати повноцінні бекенд-системи. Велика екосистема бібліотек і фреймворків, таких як React, Angular та Vue.js, значно спрощує процес розробки, забезпечуючи високу продуктивність та зручність у використанні.

Python також є потужним інструментом для розробки веб-застосунків, особливо завдяки своїй простоті і читабельності коду. Ця мова ідеально підходить для швидкої розробки і тестування завдяки фреймворкам, таким як Django та Flask, які надають зручні та ефективні інструменти для створення застосунків. Python відомий своєю універсальністю, що дозволяє використовувати його не тільки для веб-розробки, але й для аналітики даних, автоматизації, а також розробки рішень у сфері штучного інтелекту та машинного навчання.

Java є однією з найстаріших і найнадійніших мов програмування, що широко використовується у великих корпоративних системах і веб-застосунках. Вона забезпечує високу продуктивність, безпеку та масштабованість, що робить її ідеальною для розробки складних та ресурсомістких веб-додатків. Завдяки фреймворкам, таким як Spring та JavaServer Faces, Java надає потужні інструменти для створення бекенд-систем та інтеграції з різними сервісами. Хоча популярність Java дещо знизилась останніми роками, вона все ще залишається важливим інструментом для розробників, які працюють над великими проектами, що вимагають надійності та стабільності.

Отже, для створення прогресивного веб-застосунку по управлінню фінансами оптимальним рішенням є JavaScript. Ця мова дозволяє розробляти як клієнтську, так і серверну частину завдяки Node.js, використовувати потужні бібліотеки (React, Angular, Vue.js) та створювати progressive web app з високою швидкістю завантаження, офлайн-режимом і кросплатформеністю.

3.2 Обґрунтування вибору бібліотек, фреймворків та середовища програмування для розробки

У процесі вибору інструментів для розробки прогресивного веб-застосунку для спільного та особистого управління фінансами важливу роль відіграє вибір фреймворку або бібліотеки для створення інтерфейсу користувача. Три найбільш популярні рішення у цій сфері — React, Angular та Vue.js. Кожне з них має свої унікальні переваги та особливості, які впливають на зручність розробки, продуктивність та масштабованість застосунку. Згідно щорічного опитування StackOverflow “2023 Develoeper Survey” [15], вони входять до ТОП-8 “Web frameworks and technologies”.

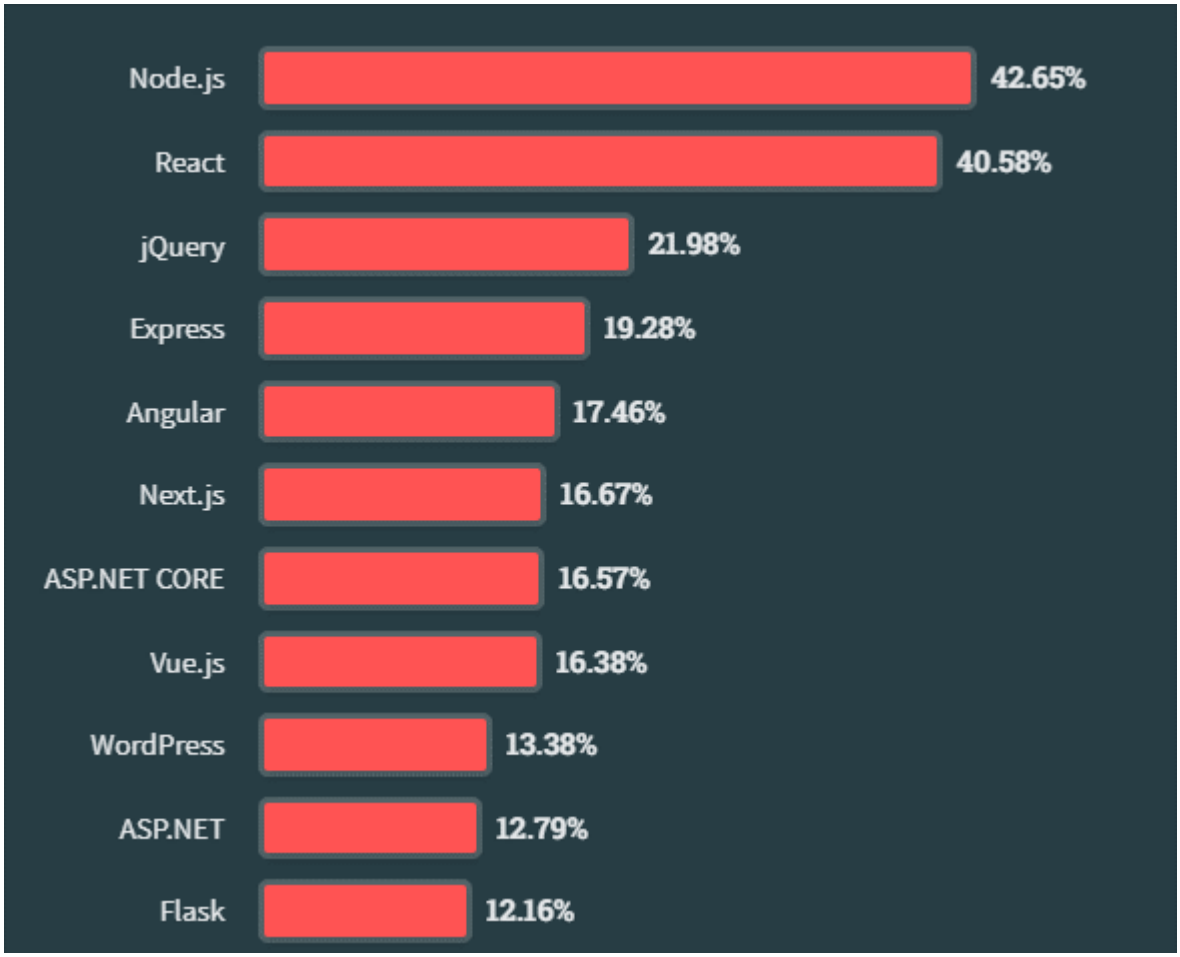


Рисунок 3.3 – Рейтинг веб-фреймворків та технологій StackOverflow за 2023 рік

React – це бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона орієнтована на компонентний підхід, що сприяє повторному використанню коду і спрощує його підтримку. React використовує віртуальний DOM для підвищення продуктивності, що дозволяє ефективно оновлювати лише ті частини сторінки, які змінюються. React має велику спільноту розробників та багату екосистему, включаючи інструменти для серверного рендерингу (Next.js) та мобільної розробки (React Native) [16].

Angular – це повноцінний фреймворк, розроблений компанією Google. Angular надає широкий набір інструментів для розробки масштабованих веб-додатків, включаючи двостороннє прив'язування даних, вбудовану підтримку модулів, форм та маршрутизації. Angular використовує реальний DOM, що може впливати на продуктивність при великій кількості змін у сторінці. Фреймворк також має строгу структуру, що може бути як перевагою, так і недоліком залежно від вимог проекту [17].

Vue.js – це прогресивний фреймворк для створення користувацьких інтерфейсів. Vue.js поєднує найкращі риси React та Angular, пропонуючи простий синтаксис та гнучкість. Vue.js використовує віртуальний DOM і підтримує компонентний підхід, що дозволяє створювати продуктивні та зручні у підтримці застосунки. Vue.js також має добре розвинену екосистему і підтримку, але менш поширений у корпоративному середовищі порівняно з React та Angular [18].

Зважаючи на особливості кожного фреймворку, для реалізації прогресивного веб-застосунку для спільного та особистого управління фінансами оптимальним вибором є React. Його компонентний підхід, висока продуктивність завдяки віртуальному DOM та широка екосистема інструментів роблять його ідеальним рішенням для створення інтерактивних та масштабованих веб-додатків. React також забезпечує велику гнучкість у розробці та підтримці коду, що є важливим для швидкого реагування на зміни у вимогах користувачів та ринку.

Вибір правильного середовища розробки є критично важливим для успішної реалізації програмного проекту. Це середовище забезпечує необхідні інструменти, бібліотеки та фреймворки, які спрощують процес кодування, тестування та

розгортання додатку. Від правильного вибору залежать продуктивність розробника, якість коду, швидкість впровадження змін та загальна ефективність розробки. Тому ретельний аналіз вимог проекту, особливостей різних середовищ та їх сумісності з технологічним стеком є необхідним для мінімізації ризиків та максимізації результатів. Обраним було Visual Studio Code, який є безсумнівним лідером посеред інших (рис. 3.4).

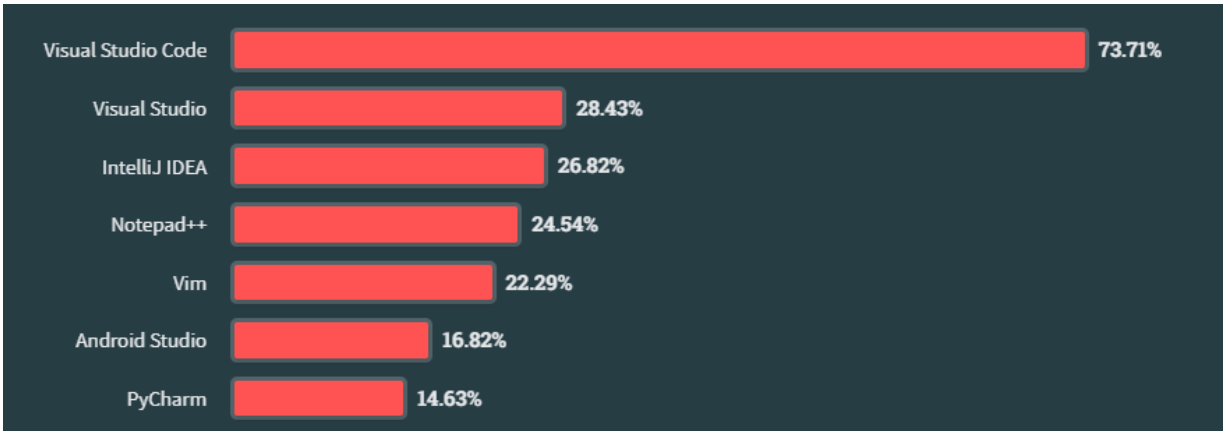


Рисунок 3.4 - Рейтинг інтегрованих середовищ розробки StackOverflow за 2023 рік

Visual Studio Code (VS Code) є ідеальним рішенням для розробки веб-застосунків на React. Це середовище розробки має вбудовану підтримку Node.js та npm, що полегшує керування залежностями React та виконання команд. Потужний редактор коду забезпечує підсвічування синтаксису JSX, автодоповнення, рефакторинг та навігацію по коду. Численні розширення, такі як React Extension Pack, Prettier та ESLint, спрощують розробку на React. Вбудований відладчик дозволяє налагоджувати React-компоненти та визначати помилки. Функція Live Share надає можливість кільком розробникам співпрацювати в режимі реального часу над одним проектом. VS Code також дозволяє персоналізувати інтерфейс, тему, ярлики та плагіни відповідно до ваших вподобань. На рисунку 3.5 наведено процес розробки.

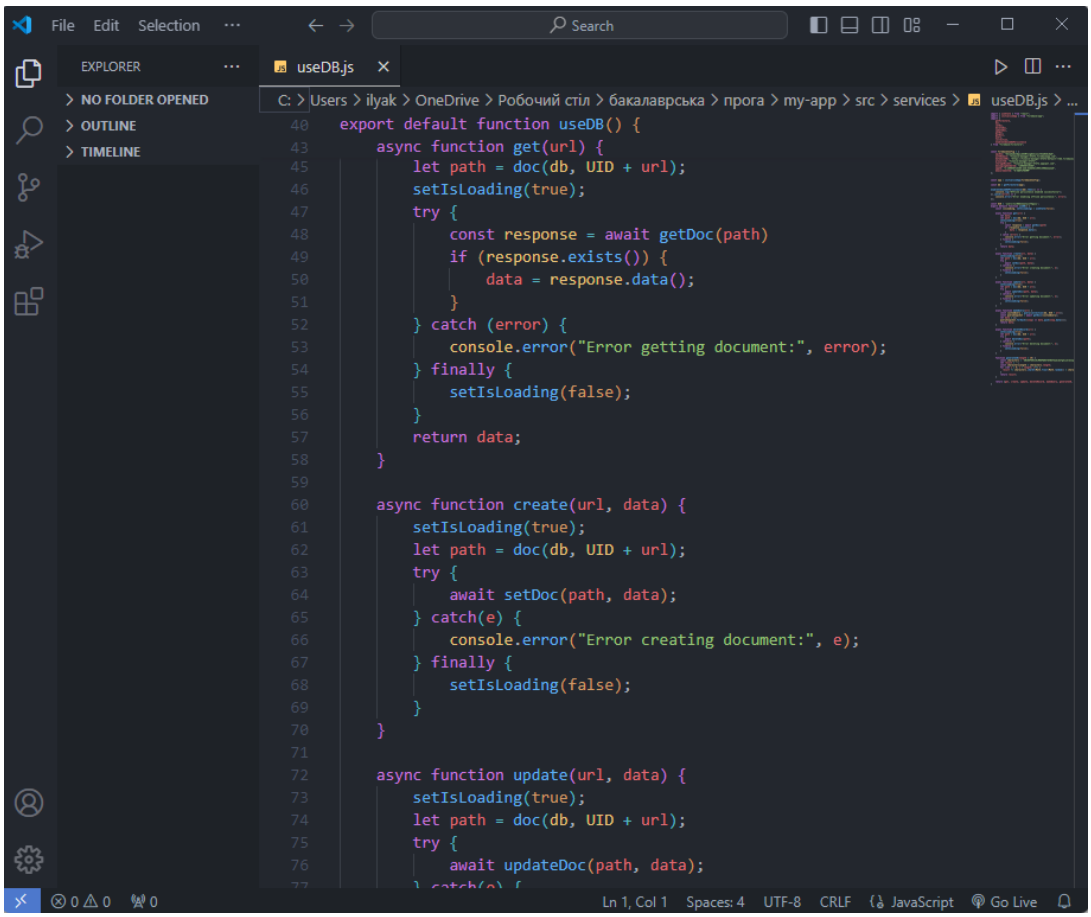


Рисунок 3.5 – Вигляд інтерфейсу середовища розробки VS Code

3.3 Реалізація інтерфейсу прогресивного WEB-застосунку

Успішна розробка прогресивного веб-застосунку для спільного та особистого управління фінансами вимагає ретельного планування та поетапного підходу до створення продукту. Одним з ключових аспектів цього процесу є розробка макету перед додаванням логіки за допомогою React. Створення макету є важливим першим кроком, оскільки він дозволяє візуалізувати структуру та вигляд кінцевого продукту, визначити його основні функціональні елементи та забезпечити зручність користувацького інтерфейсу. Макетування допомагає уникнути помилок, які можуть виникнути на пізніших етапах розробки, забезпечує чітке розуміння вимог до дизайну та функціональності.

Тобто, послідовне створення макету та подальше додавання логіки за допомогою React є оптимальним підходом для розробки веб-застосунку, що

забезпечує високу якість кінцевого продукту, зручність для користувачів та ефективність роботи.

Застосунок повинен мати три сторінки: “Головна”, “Статистика” та “Налаштування”. Звідси повинна бути навігація що дозволить між ними перемикатись (рис. 3.4).

Рисунок 3.4 – Зовнішній вигляд навігації

На кожній сторінці основні компоненти будуть організовані у вигляді невеликих блоків. На головній розміщено: компонент для керування балансом буде мати вигляд, що на рисунку 3.5, компонент що відображає нещодавні транзакції на рисунку 3.6, список бажань - на рисунку 3.7 та фонове зображення і статистика.

Сторінка “Статистика” матиме три блоки з різними графіками і також фільтр дат (рис. 3.8). Також сторінка “Налаштування” міститиме три блоки з наступними функціями: “Зміна імені”, “Зміна тла головної сторінки”, “Зміна балансу” (рис. 3.9).

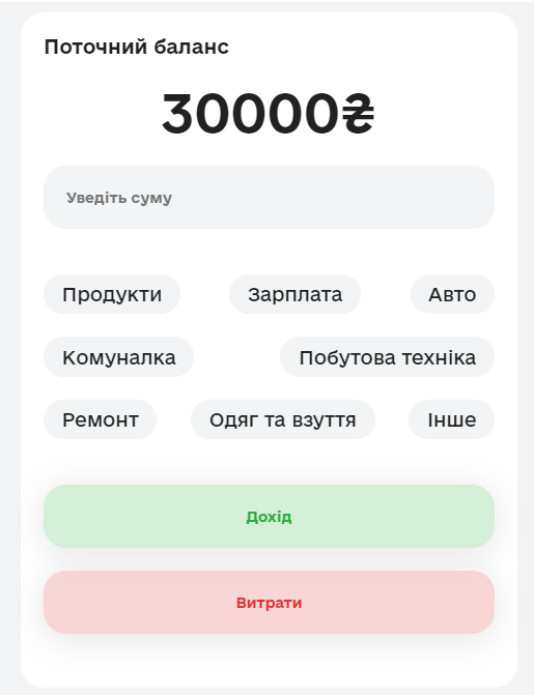


Рисунок 3.5 – Зовнішній вигляд компонента для керування балансом

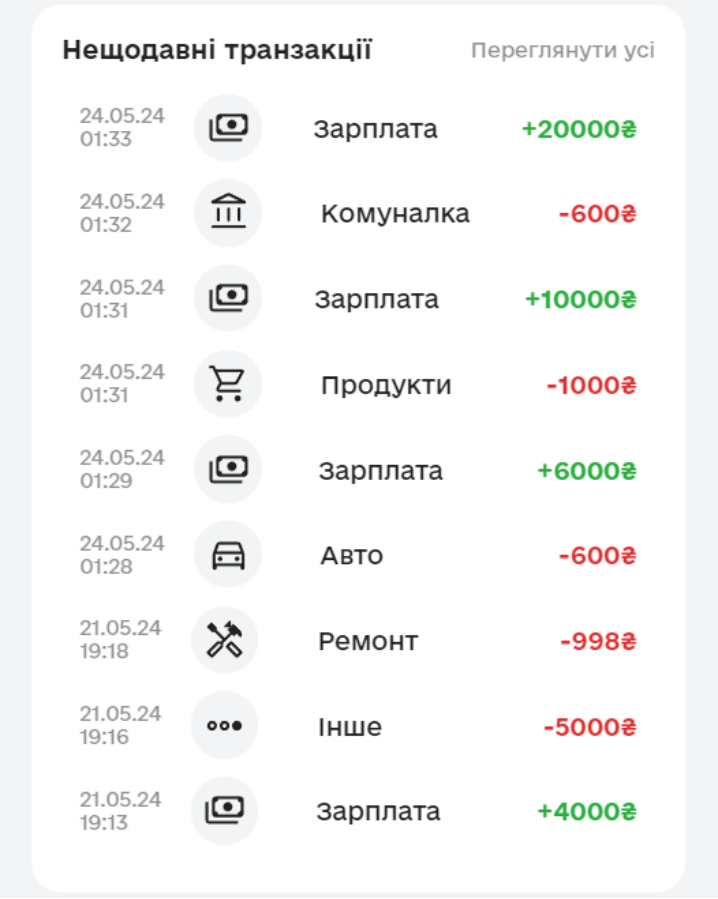


Рисунок 3.6 – Зовнішній вигляд компонента для відображення нещодавніх транзакцій

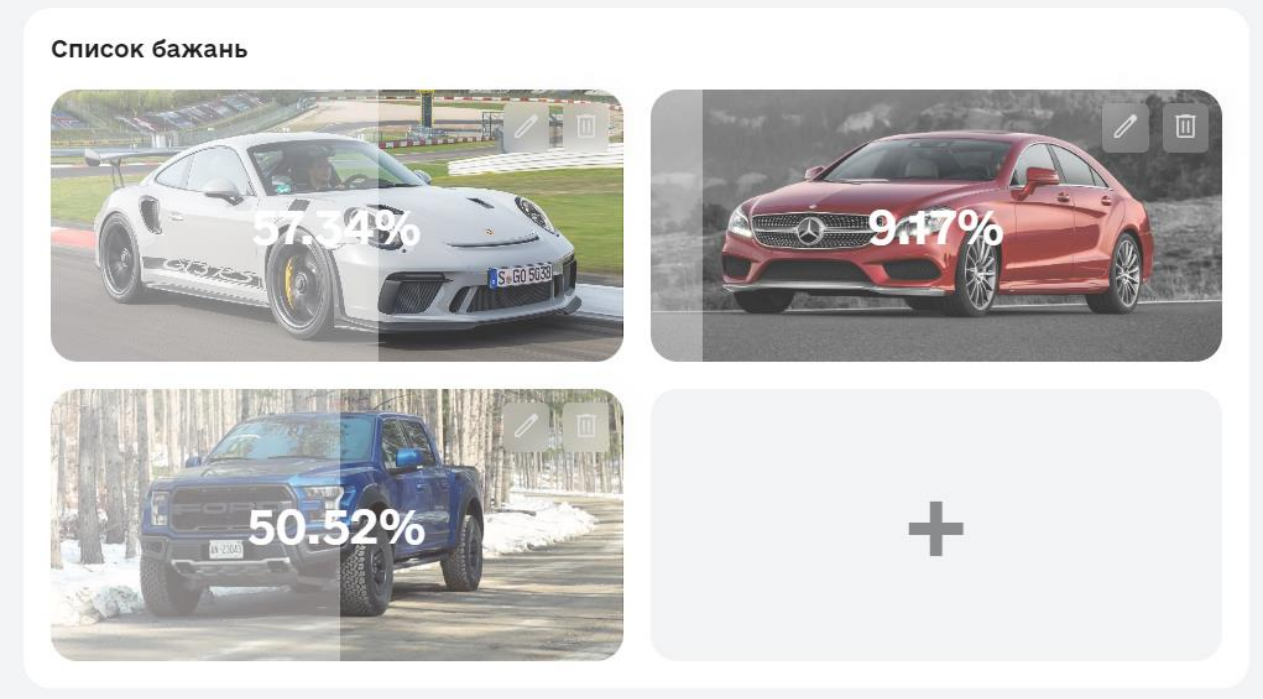


Рисунок 3.6 – Зовнішній вигляд компонента для керування списком бажань

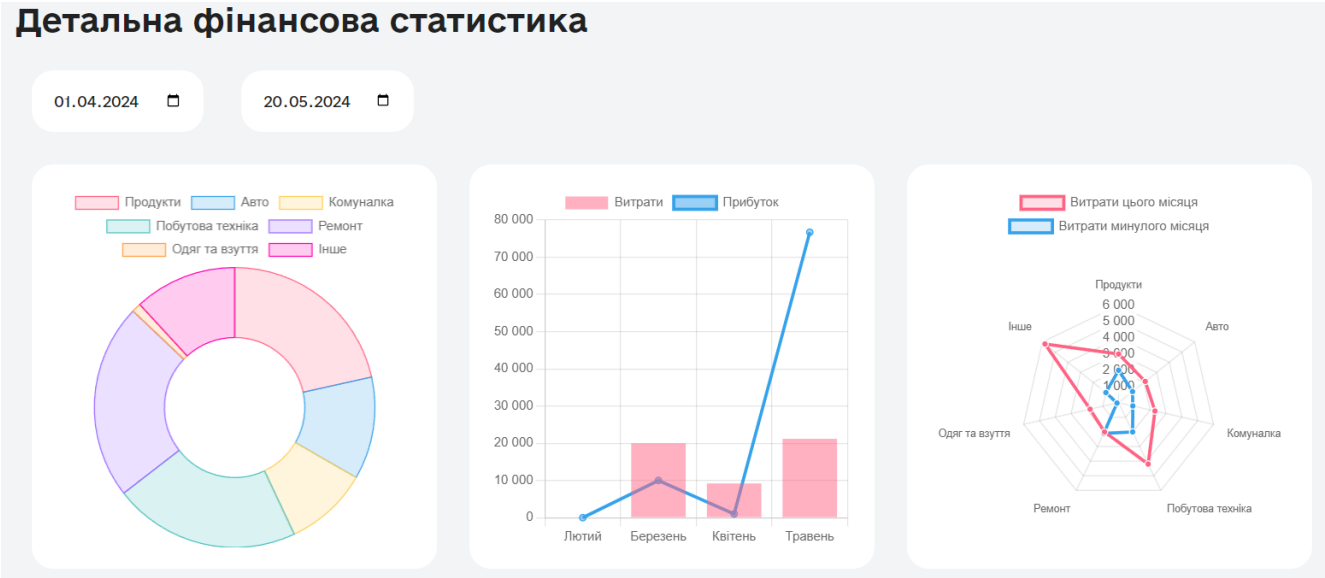


Рисунок 3.8 – Зовнішній вигляд сторінки “Статистика”

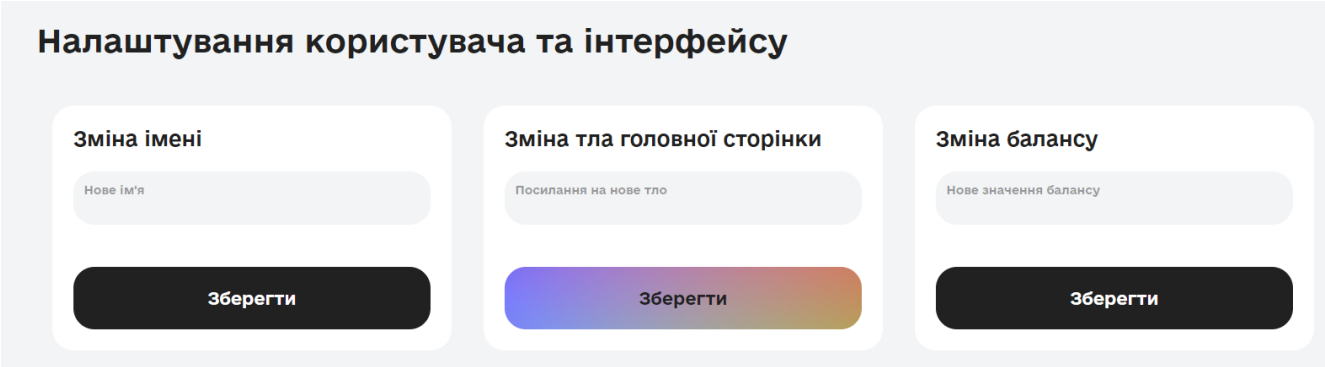


Рисунок 3.9 – Зовнішній вигляд сторінки “Налаштування”

Отож, розроблено інтерфейс ключових сторінок. Відповідно до запланованої структури, розроблено макети для таких сторінок: "Головна", "Налаштування" та "Статистика". Завершення розробки цих макетів дозволяє перейти до наступного етапу — реалізації логіки застосунку за допомогою React, що забезпечить інтерактивність та динамічність веб-застосунку.

3.4 Реалізація компонентів прогресивного WEB-застосунку

Перед тим як перейти до безпосередньої розробки прогресивного веб-застосунку для спільного та особистого управління фінансами, важливо

підготувати всі необхідні компоненти та визначити основні етапи роботи. Початкові етапи розробки включають створення макетів сторінок та налаштування середовища розробки. Макети сторінок дозволяють візуалізувати майбутній інтерфейс і забезпечують основу для додавання функціональної логіки. Налаштування середовища розробки, включаючи вибір відповідних інструментів та технологій, таких як React та Visual Studio Code, забезпечує ефективність та продуктивність роботи над проектом. Тепер, коли основні підготовчі етапи завершено, можна перейти до безпосередньої реалізації функціоналу застосунку.

Для початку створено файлову структуру, що зображено на рисунку 3.10, згідно компонентного підходу, де “components” – папки з компонентами, що містять у собі .js файл та файл стилів .scss, “resources” – папка з медіафайлами (фото, іконки і тому подібне), “services” – папка з власними хуками (хук для роботи з базою даних). Далі “розрізаємо” макет на відповідні компоненти, і у кожній відповідній папці вносимо шматок коду із макету.

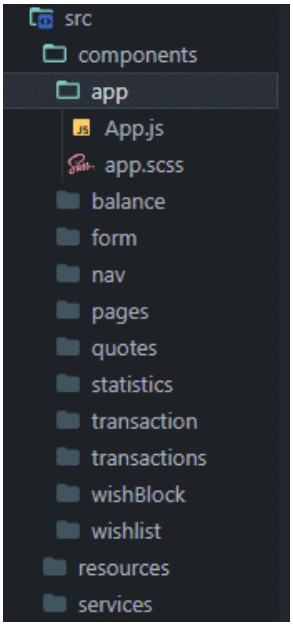


Рисунок 3.10 – Структура проєкту

Із основних моментів програмної реалізації – є написання власного хуку для зручності використання баз даних у проєкті. Для початку необхідно імпортувати бібліотеки з Firebase для роботи з Firestore. Далі виконати нашалтування додатку

Firebase за допомогою конфігурації `firebaseConfig`, ініціалізується Firestore, і вмикається офлайн-доступ через наступну команду - enableIndexedDbPersistence(db).

Оголошується функція `useDB`, яка повертає набір функцій для роботи з базою даних. Функція `get` отримує документ за заданим URL, використовуючи `getDoc`. Функція `create` створює новий документ за допомогою `setDoc`, а `update` оновлює існуючий документ через `updateDoc`. Функція `makeQuery` виконує запит до колекції і повертає масив документів. Функція `deleteRecord` видаляє документ за допомогою `deleteDoc`. Кожна із цих функцій використовує зручні інструменти, що надає бібліотека “firebase/firestore”, також містить конструкцію try/catch для відловлення помилок при зверненні до баз даних та async/await для виконання асинхронного коду, а саме запит до баз даних. Ці функції дозволяють виконувати CRUD-операції з Firestore, підтримуючи офлайн-доступ до даних.

Також, особливим є реалізація Service Worker, що виконує функцію проміжного шару між веб-додатком, браузером та мережею (якщо є доступ до інтернету). Їх основне призначення - забезпечити якісний досвід роботи в офлайн-режимі, перехоплювати мережеві запити та реагувати на них відповідним чином залежно від доступності інтернет-з'єднання, а також оновлювати ресурси, що знаходяться на сервері. Крім того, сервіс-воркери надають можливість отримувати push-сповіщення та взаємодіяти з API для фонової синхронізації даних.

Код у файлі “sw.js” реєструє Service Worker для веб-додатку. Під час встановлення (подія 'install') він відкриває кеш з ім'ям 'my-app-cache' і додає в нього ресурси з масиву `urlsToCache`. Під час події 'fetch' він перехоплює запити на мережу і намагається знайти відповідь у кеші. Якщо відповідь знайдена, він повертає її, якщо ні, то виконує мережевий запит і повертає результат або, в разі помилки, повертає кешовану сторінку 'offline.html'. Під час активації (подія 'activate') він видаляє старі кеші, які не знаходяться в `cacheWhitelist`.

Також для прогресивного WEB-застосунку необхідним є створення файлу “manifest.json”, що містить метадані про додаток, такі як ім'я, короткий опис,

іконки, кольори теми, стартова сторінка та інші деталі. Ця інформація дозволяє платформам, таким як операційні системи мобільних пристроїв або браузери, адаптувати відображення та поведінку додатку.

3.5 Тестування прогресивного WEB-застосунку

Для виявлення помилок у застосунку використано ручне тестування, тобто таке що не потребує використання автоматизованих інструментів, або ж сценаріїв. Воно дозволяє виявити проблеми сумісності, зручності використання та інші проблеми, які не можуть бути визначеними автоматизованим тестуванням [20]. Для початку запущено сайт на локальному хості і виконано тест на адаптивність: вигляд на комп’ютері з роздільною здатністю 1920*1080 на рисунку 3.11, на телефоні 375*667 на рисунку 3.12. З рисунків видно, що навігація на мобільних пристроях та планшетах відображається внизу, а також кожен блок займає усю довжину екрану, на відміну від комп’ютера. Жоден з елементів на виступає за границі екрану та не з’являється горизонтальна смуга прокручування. Тобто, тестування на адаптивність пройдено.

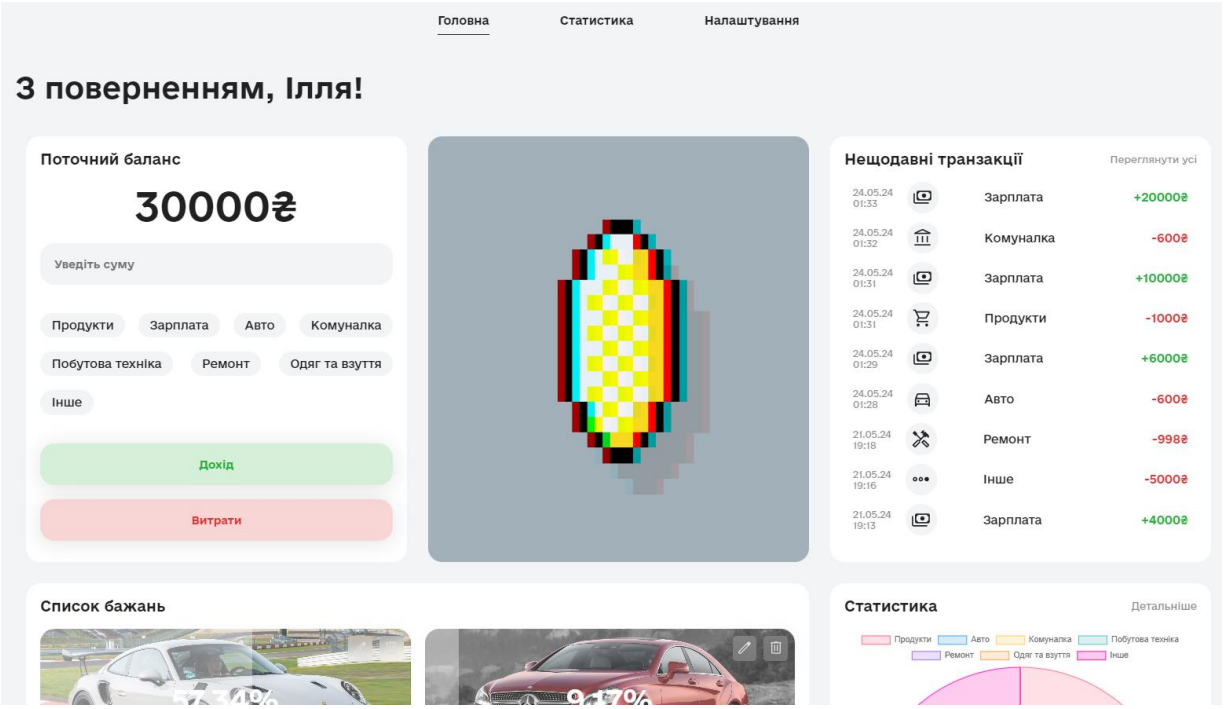


Рисунок 3.11 – Вигляд веб-ресурсу на комп’ютері

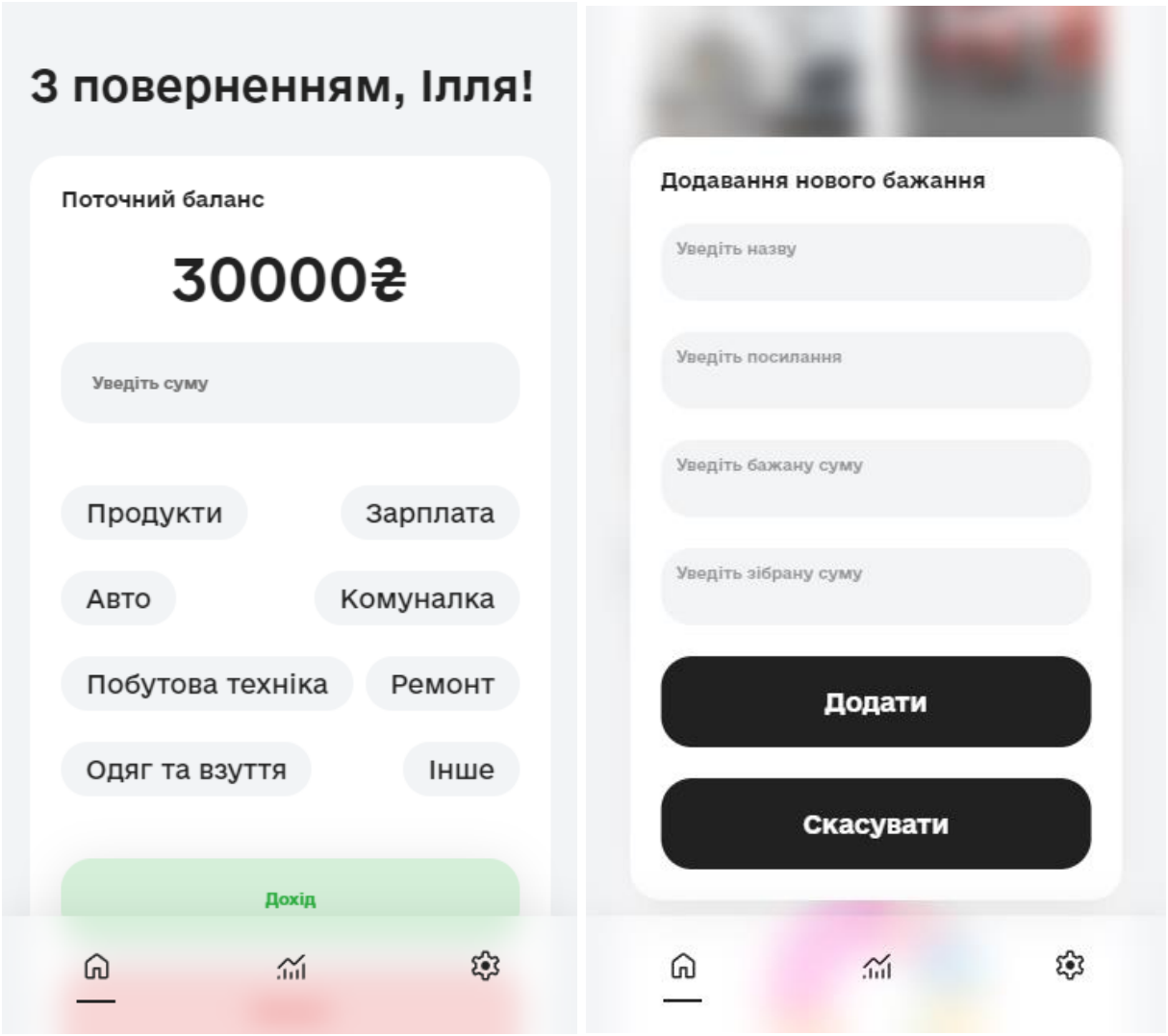


Рисунок 3.12 – Вигляд веб-ресурсу на мобільному пристрої

Далі, виконано тестування додавання доходу та витрат, зміну балансу та відображення у нещодавніх транзакціях. Для цього добавлено один дохід у розмірі 5.000 гривень та дві витрати: ремонт – 1000 гривень, продукти – 500 гривень. Початкова сума була 30.000 гривень (рис. 3.13), кінцева стала 33.500 гривень (рис. 3.14), що свідчить про правильну калькуляцію балансу. Також у нещодавніх транзакціях з’явилось три нових записи сьогоднішьою датою (27.05 на момент тестування).

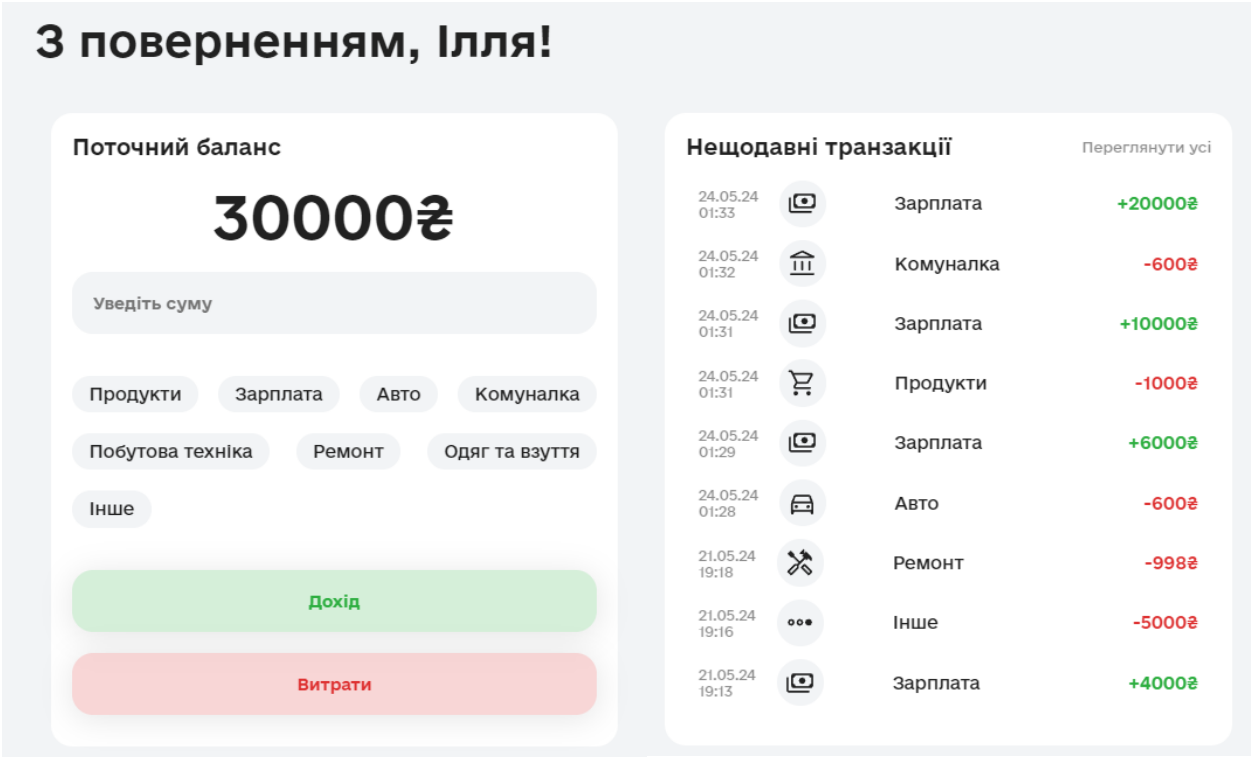


Рисунок 3.13 – Вигляд компонентів до внесення нових транзакцій

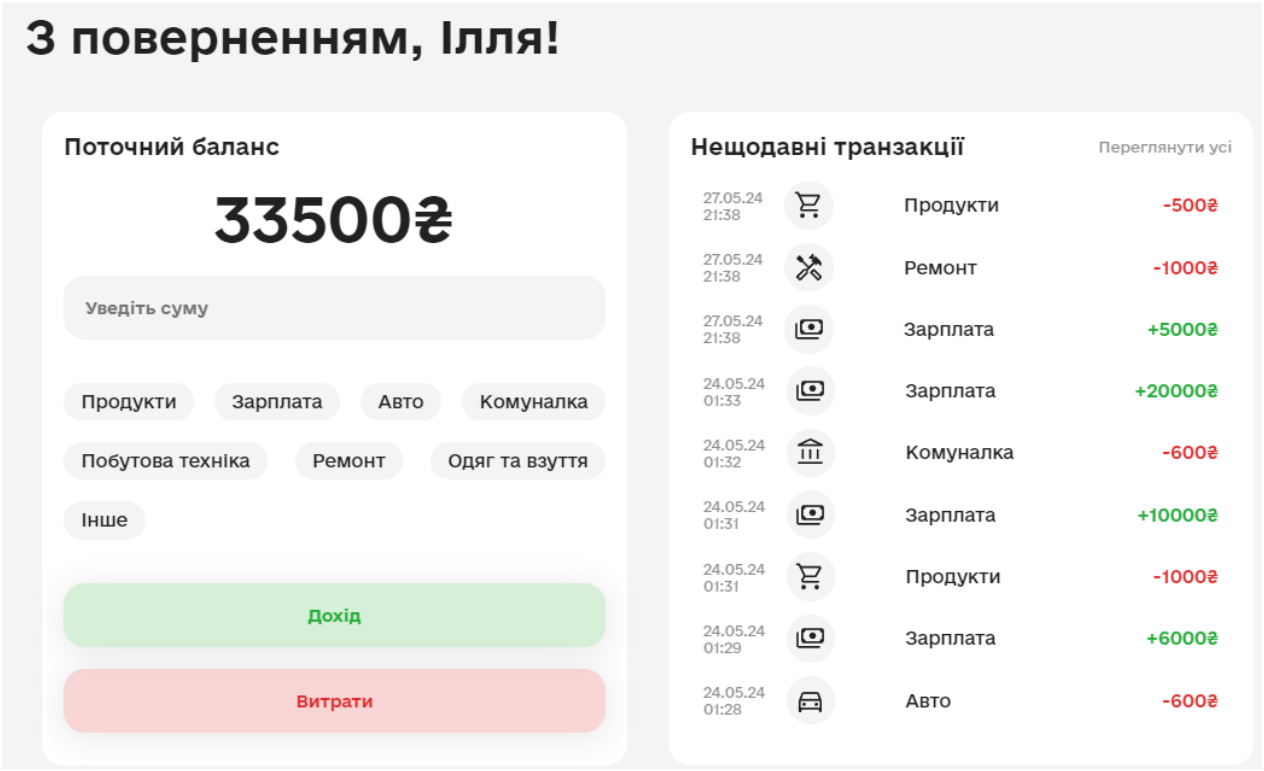


Рисунок 3.14 – Вигляд компонентів після внесення нових транзакцій

Наступним, проведено тестування для списку бажань. Виконано функцію додавання нового, зміна балансу та видалення, початковий вигляд на рисунку 3.15.

Для створення натиснуто відповідний блок зі знаком “+”, де отримано форму, що зображена на рисунку 3.16 та внесено відповідні дані нового бажання і отримано результат на рисунку 3.17

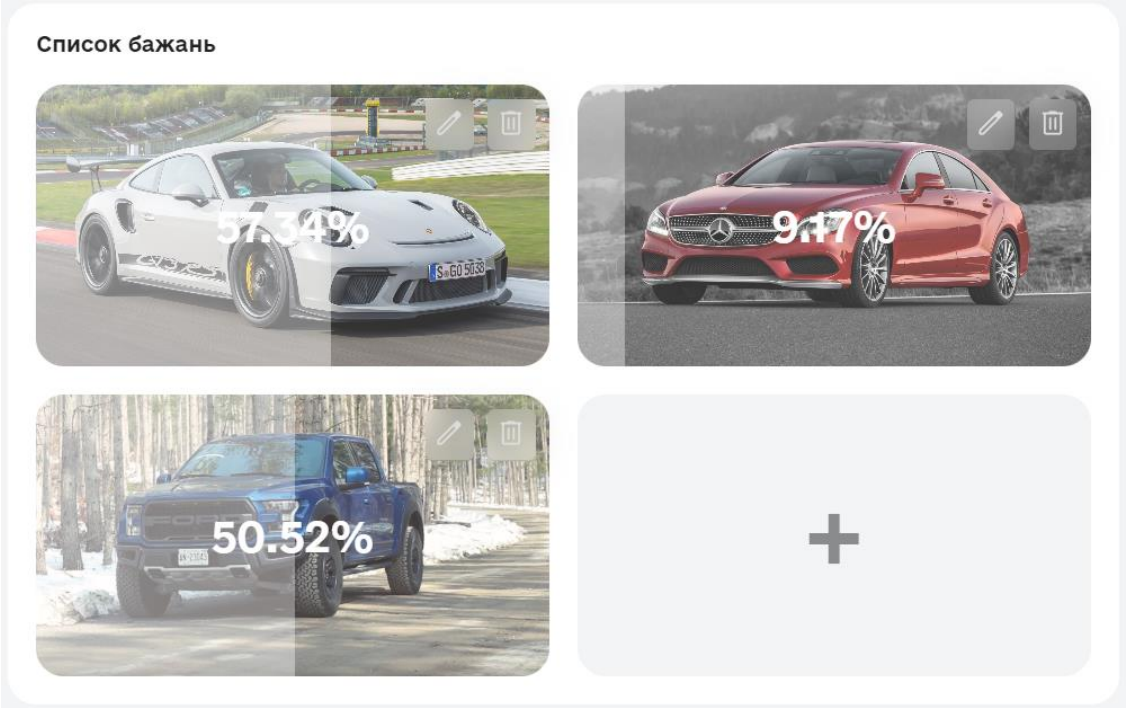


Рисунок 3.15 – Початковий вигляд списку бажань

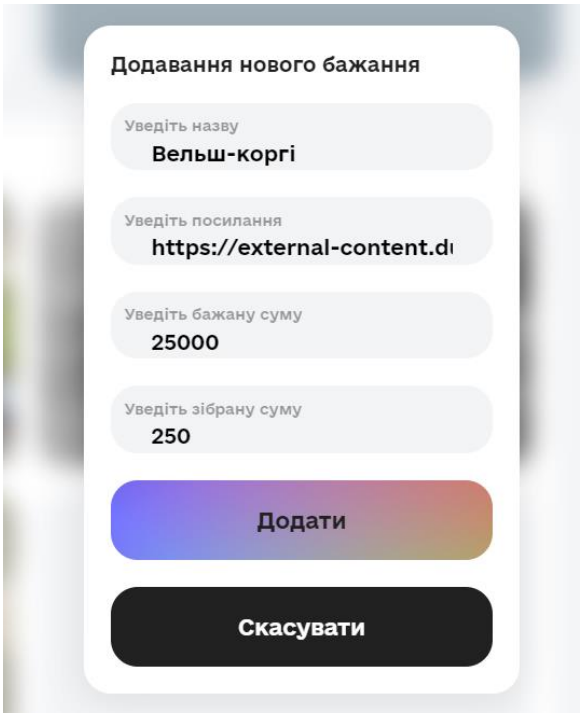


Рисунок 3.16 – Вигляд форми для створення нового бажання з внесеними даними

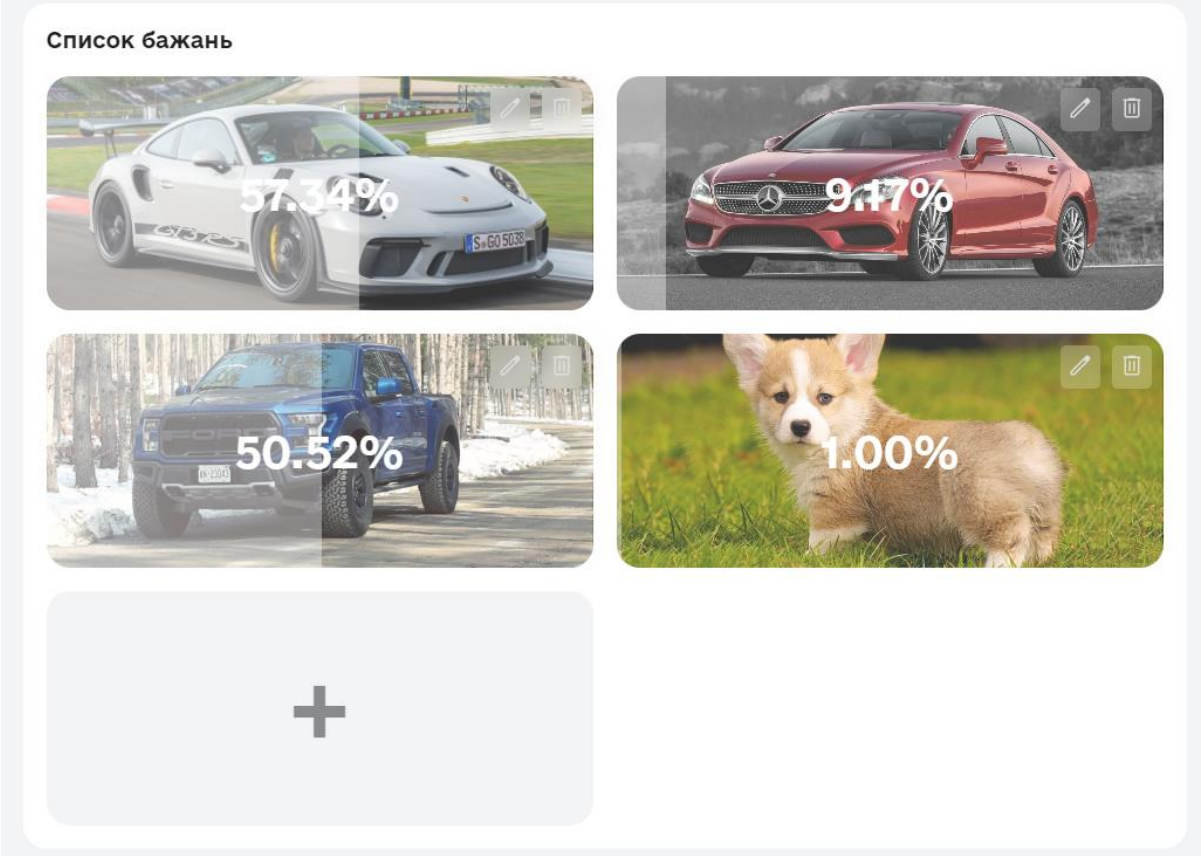


Рисунок 3.17 – Вигляд списку бажань після додавання нового

Отримано на новому бажанні 1.00% прогресу через те що, сума для досягнення цілі 25.000, а зібрана 250 – що становить 1% від загальної. Тепер натиснуто кнопку редагування, де отримано відповідну форму та внесено у неї 23.000 гривні, що дало результат на рисунку 3.18. Після чого протестовано кнопку видалення на прикладі другої карточки з червоним авто, та отримано наступний результат, що на рисунку 3.19.

Рисунок 3.18 – Зміна прогрес-бару відповідно до внесеної суми

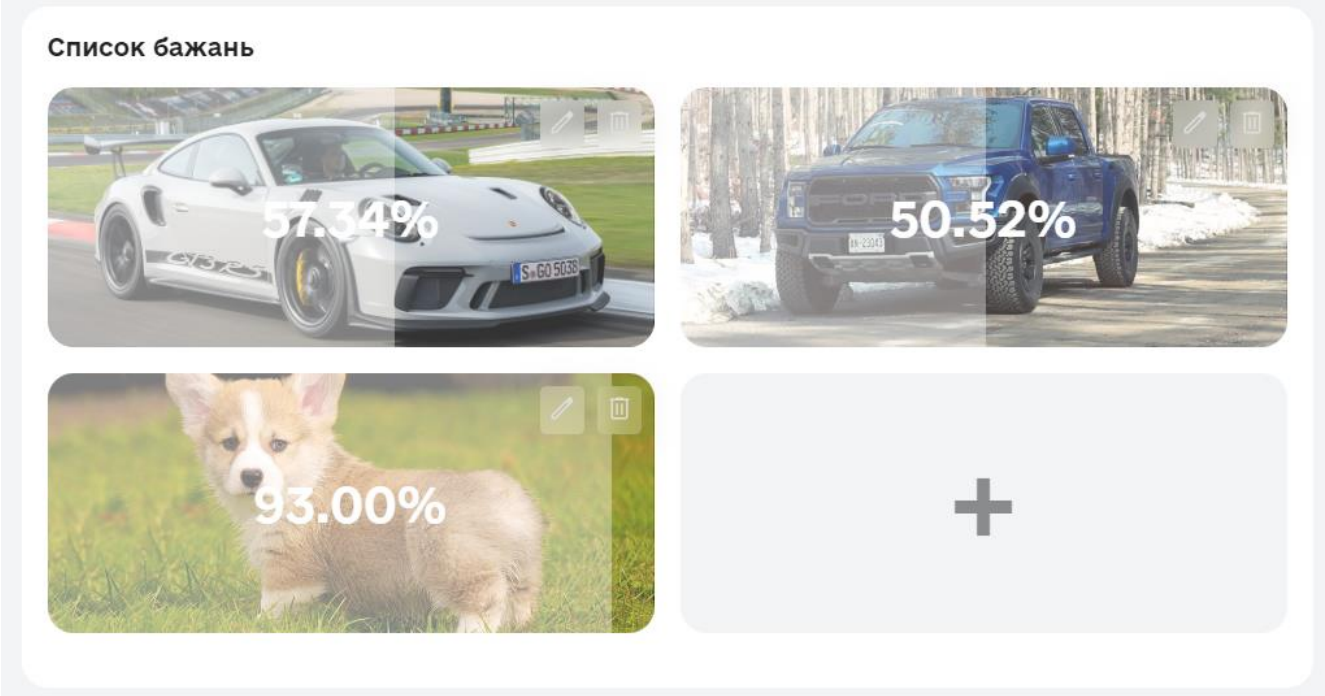


Рисунок 3.19 – Оновлений список бажань після видалення

Отже, функція добавлення нової карточки працює, редагування працює та обчислюється коректно, після видалення будь якої карточки, усі інші оновлюють своє місцерозташування та не виникає проблем з верстkою.

Наступним кроком тестування була перевірка модуля статистики. Було проведено тестування роботи фільтру по діапазону дат, коректності відображення даних та оновлення статистики відповідно до нових транзакцій. На рисунку 3.20 представлено вигляд статистики за замовчуванням У попередньому тестуванні, яке показане на рисунку 3.14, видно, що 27.05 було проведено лише три транзакції. Щоб перевірити роботу фільтру, діапазон дат було змінено на цей день. Як результат, отримано кругову діаграму (рис. 3.21), яка коректно відображає витрати за цей день. Для подальшого тестування було внесено ще одну витрату на суму 200 гривень за комунальні послуги. Після цього кругова діаграма (той же рисунок) автоматично оновилася, відобразивши нові дані. Це підтвердило, що система правильно обробляє та візуалізує зміни у фінансових даних у режимі реального часу.

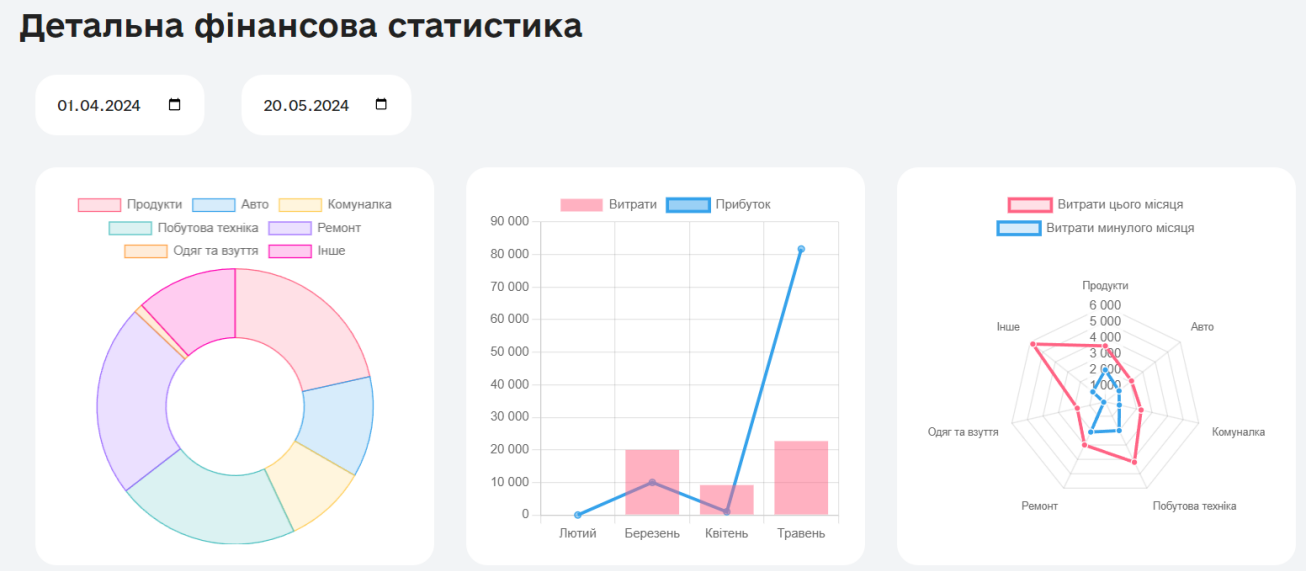


Рисунок 3.20 – Початковий вигляд сторінки зі статистикою

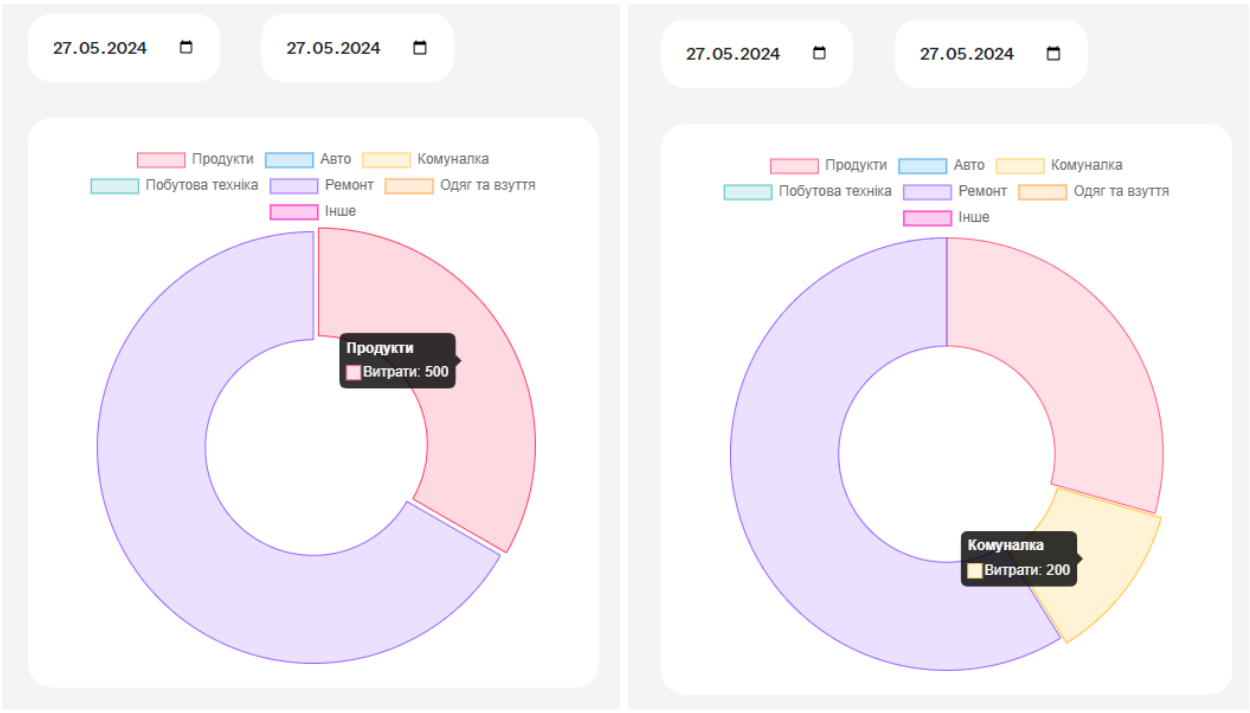


Рисунок 3.21 – Вигляд кругової діаграми за 27.05

На графіку витрати/прибуток за місяць (рис. 3.22) – витрати у травні складають 22.898 гривень, а прибуток – 81.700 гривень. Нехай добавимо нову витрату на одяг у розмірі 5.000 гривень. Тоді на тому ж рисунку видно зміни, що витрати складають уже 27.898 гривень, прибуток не змінився. Те ж саме і з

радарною діаграмою витрат по категоріям цього і минулого місяця, де на рисунку 3.23 зображено до та після внесення витрати на одяг.

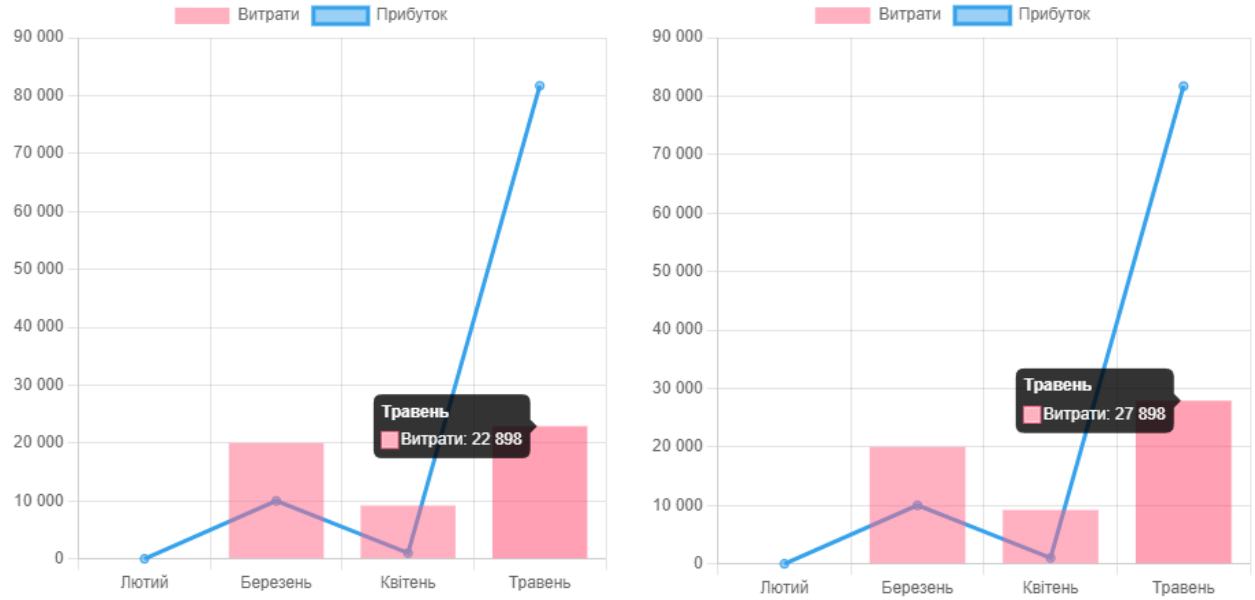


Рисунок 3.22 – Вигляд графіку витрати/прибуток до та після внесення змін

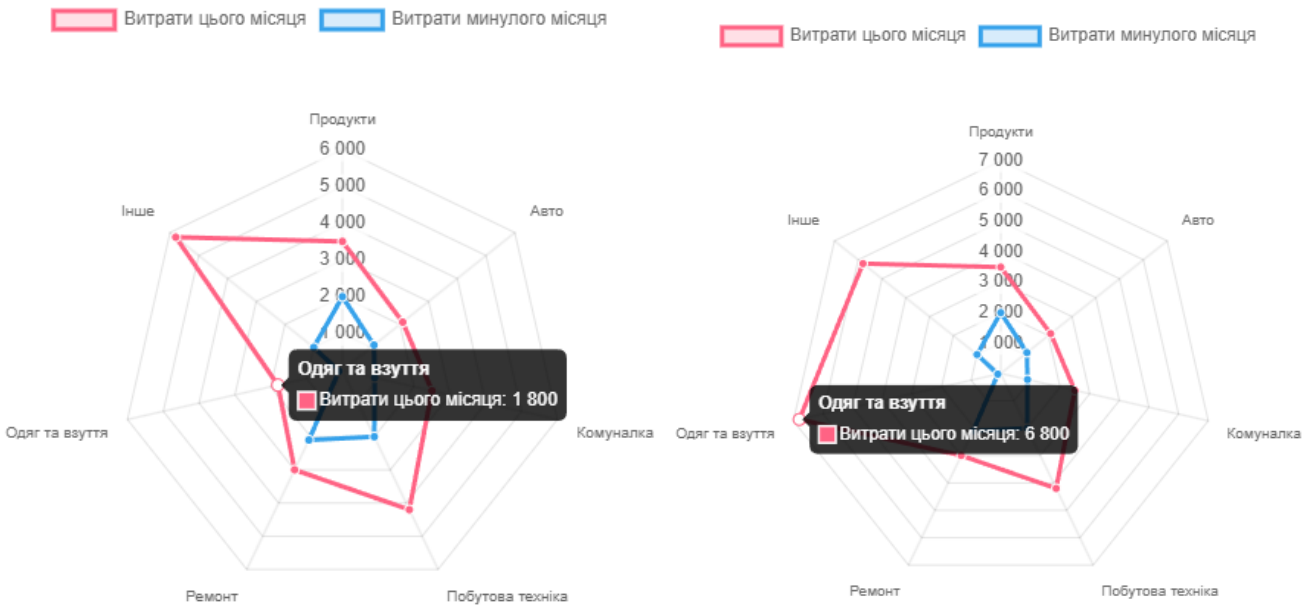


Рисунок 3.23 – Вигляд радарної діаграми до та після внесення змін

Останнім етапом, це тестування безпосередньо роботи ресурсу як прогресивного веб-застосунку, тобто це встановлення і відображення як звичайної

програми та робота офлайн. Отож, при переході на сайт, пропонує його встановити (рис. 3.24) після натискання згоди – на робочому столі з’являється іконка ресурсу при клікові на який отримано результат що на рисунку 3.25. Тобто, веб-ресурс тепер нагадує звичайну програму, що доступна по кліку на іконку, проте “під капотом” все ж таки працює браузер.

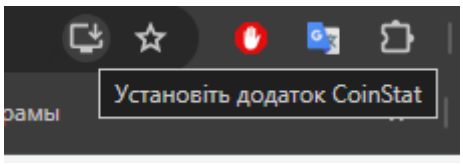


Рисунок 3.24 – Підказка, що притаманна прогресивним веб-застосункам

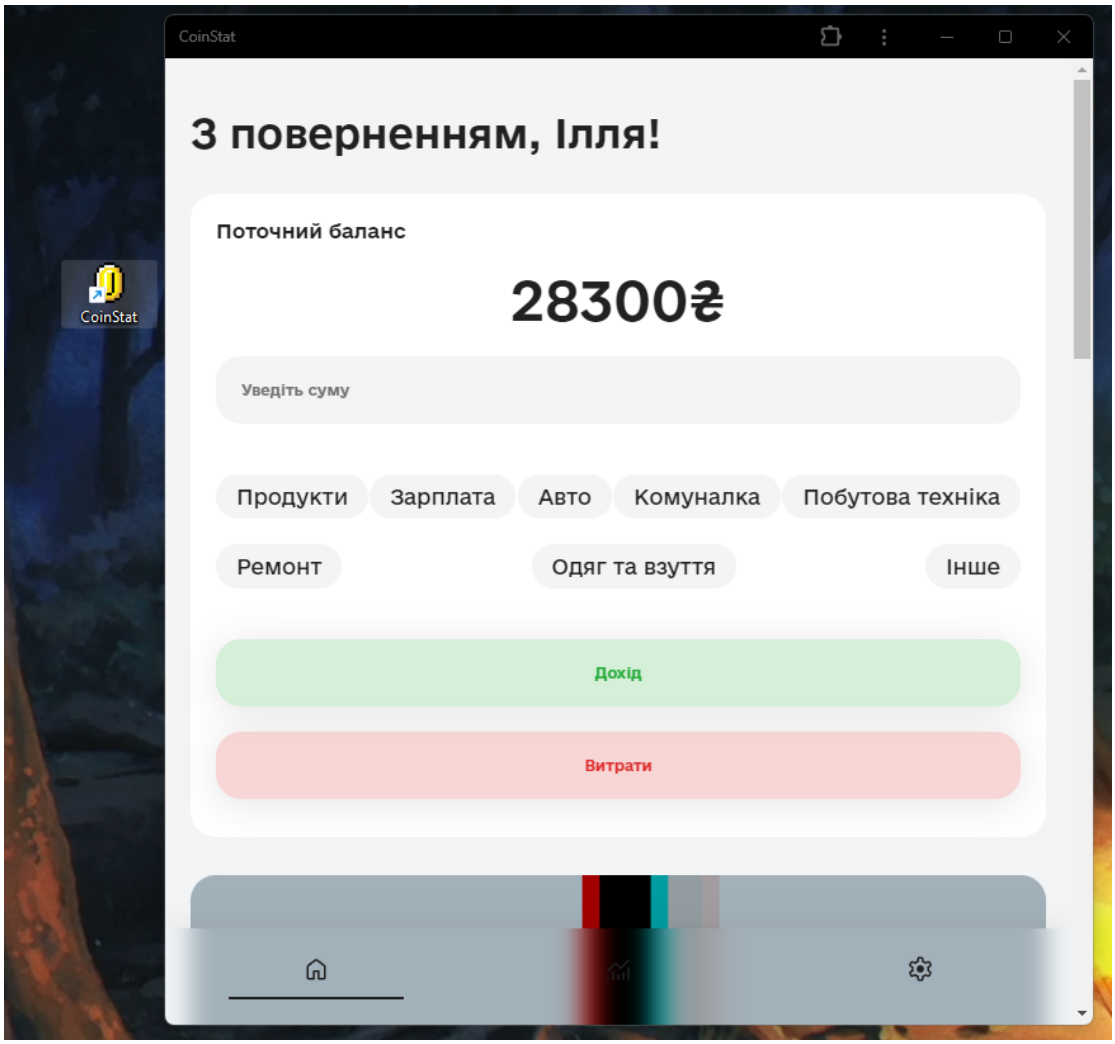


Рисунок 3.25 – Вигляд встановленого веб-ресурсу на комп’ютері, де видно іконку та вікно застосунку

Залишився офлайн-режим, для цього через інструменти розробки у вкладці “Network” вимикаємо з’єднання, імітуючи відсутність інтернет зв’язку. Вносимо витрати 7.000 гривень на продукти та повертаємо підключення до Інтернету (рис. 3.26). Після чого оновлюємо сторінку і отримано те, що витрати на продукти залишись і записались до бази даних.

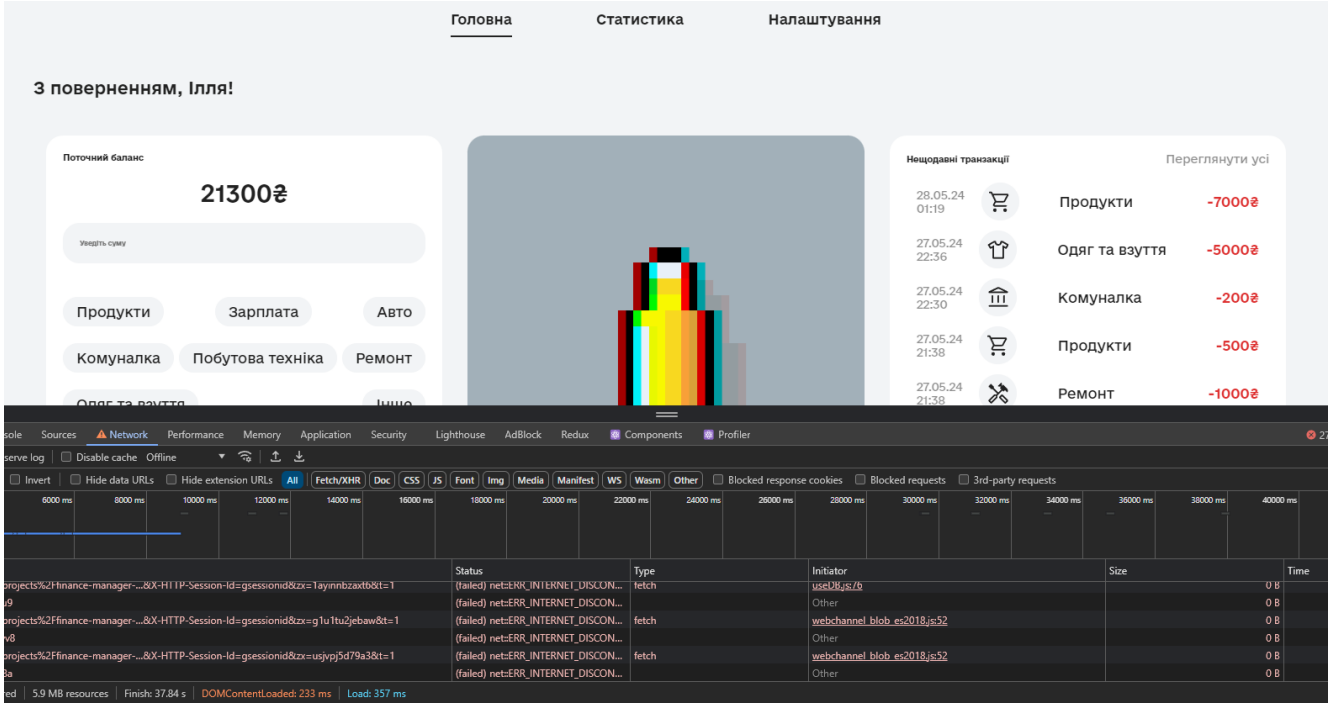


Рисунок 3.26 – Процес тестування офлайн-режиму застосунку

Отже, застосунок виконує усі свої функції правильно та відповідно до поставлених вимог, у тому числі нововведенні з результатів аналізу схожих програм.

3.6 Висновки до розділу 3

Обрано мову програмування, яка має вирішальне значення для успішної розробки веб-застосунку. Після аналізу популярності та відповідних можливостей, було вирішено використовувати JavaScript. Ця мова забезпечує високу гнучкість завдяки можливості розробки як клієнтської, так і серверної частини застосунку за допомогою Node.js. Окрім того, JavaScript має велику екосистему бібліотек і

фреймворків, що спрощують процес розробки та забезпечують високу продуктивність.

Також було обрано бібліотеку React. Її компонентний підхід, висока продуктивність завдяки віртуальному DOM та багата екосистема інструментів роблять її оптимальним рішенням для створення інтерактивних та масштабованих веб-додатків. Як середовище розробки було обрано Visual Studio Code, яке забезпечує зручні інструменти для розробки, тестування та відладки коду.

Було розроблено макети трьох основних сторінок: "Головна", "Статистика" та "Налаштування" та виконано подальшу програмну реалізацію, шляхом розбиття макету на компоненти й реалізації логіки та відображення певної частини на стороні користувача, з використанням обраної мови, бібліотеки та середовища розробки.

Проведено тестування, що є критично важливим етапом для виявлення помилок і забезпечення коректної роботи застосунку, а саме ручне тестування, що включало перевірку адаптивності дизайну на різних пристроях, тестування функцій додавання доходів і витрат, керування списком бажань, а також перевірку коректності відображення статистики. Результати тестування показали, що застосунок працює коректно, елементи інтерфейсу адаптуються до різних розмірів екранів, а всі функції виконуються відповідно до вимог.

Загалом розробка прогресивного веб-застосунку для управління фінансами вимагала ретельного вибору мови програмування, фреймворків та інструментів для розробки, а також послідовного підходу до створення інтерфейсу і реалізації логіки застосунку. Використання JavaScript та React забезпечило високу продуктивність і гнучкість розробки, а проведене тестування підтвердило коректність роботи застосунку. Цей підхід дозволив створити надійний і зручний для користувачів продукт, що відповідає сучасним вимогам і стандартам розробки веб-додатків.

ВИСНОВКИ

Усі завдання, поставлені перед бакалаврською дипломною роботою, виконані в повному об’ємі, а саме:

– Обгрунтовано доцільність створення прогресивного WEB-застосунку для спільного та особистого управління фінансами;

– Проаналізовано предметну область з управління особистими фінансами;

– Розроблено архітектуру та виконано проєктування прогресивного WEB-застосунку для спільного та особистого управління фінансами;

– Здійнесно програмну реалізацію прогресивного WEB-застосунку для спільного та особистого управління фінансами;

– Виконано тестування прогресивного WEB-застосунку для спільного та особистого управління фінансами.

У ході виконання бакалаврської дипломної роботи було проаналізовано предметну область стосовно управління фінансами та існуючі програмні рішення, де виконано детальне порівняння та виділено їхні переваги та недоліки. На основі цього втілено постановку задачі на дослідження.

Наступним кроком було визначено архітектуру для майбутнього застосунку, спроектовано інтерфейс та визначено основні його функції, і також розроблено UML-діаграми. Обрано мову програмування JavaScript, бібліотеку React та середовище програмування Visual Studio Code для безпосередньої реалізації застосунку. Розроблено інтерфейс та компоненти, після чого виконано тестування додатку.

Мету роботи з розширення функціональних можливостей застосунку для управління фінансами з використанням технології прогресивного WEB-застосунку досягнуто за рахунок надання користувачеві більше функцій у порівнянні з аналогами такі як: офлайн-режим, необмежена кількість пристроїв, підтримка будь якого пристрою що має інстальований браузер, відсутність необхідності завантаження з маркетів, використання хмарного сховища, відсутність будь яких додаткових оплат за певні функції.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Киринюк І.С., Сілагін О.В. Аналіз передумов розробки прогресивного вебзастосунку для спільного та особистого управління фінансами. Матеріали конференції “LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)”, Вінниця, 2024 [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20415

2. Taxipa K. Xipa. Personal Finance: Past, Present and Future [Електронний ресурс] – Режим доступу: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1522299

3. Вілл Кентон. What Is Personal Finance, and Why Is It Important? [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/p/personalfinance.asp

4. Household debt and credit [Електронний ресурс] Federal Reserve Bank of New York. – Режим доступу: https://www.newyorkfed.org/medialibrary/interactives/householdcredit/data/pdf/HHDC_2023Q4

5. What Are the 5 Areas of Personal Financial Management? [Електронний ресурс] Marietta Wealth. – Режим доступу: https://www.mariettawealth.com/what-are-the-5-areas-of-personal-financial-management/

6. Келлі Енн Сміт, Дженн Андервуд. What Is The 50/30/20 Rule? [Електронний ресурс] – Режим доступу: https://www.forbes.com/advisor/banking/guide-to-50-30-20-budget/

7. Бюджет: облік витрат, плануваня [Електронний ресурс] – Режим доступу: https://www.apptronic.net/en (дата звернення: 22.05.2024). – Назва з екрана.

8. Money Pro [Електронний ресурс] – Режим доступу: https://money.pro/ (дата звернення: 22.05.2024). – Назва з екрана.

9. Goodbudget [Електронний ресурс] – Режим доступу: https://goodbudget.com/ (дата звернення: 22.05.2024). – Назва з екрана.

10. Monefy [Електронний ресурс] – Режим доступу: https://monefy.me/ (дата звернення: 22.05.2024). – Назва з екрана.

11. Splitwise [Електронний ресурс] – Режим доступу: https://www.splitwise.com/ (дата звернення: 22.05.2024). – Назва з екрана.

12. Локеш Гупта. What is REST? [Електронний ресурс] – Режим доступу: https://restfulapi.net/

13. Unified Modeling Language [Електронний ресурс] Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language

14. Developer documentation for Firebase [Електронний ресурс] Firebase. – Режим доступу: https://firebase.google.com/docs

15. Рейтинг мов програмування 2024 [Електронний ресурс] DOU. – Режим доступу: https://dou.ua/lenta/articles/language-rating-2024/

16. 2023 Developer Survey [Електронний ресурс] Stackoverflow. – Режим доступу: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe

17. React [Електронний ресурс] React dev. – Режим доступу: https://react.dev/

18. What is Angular? [Електронний ресурс] Angular dev. – Режим доступу: https://angular.dev/overview

19. What is Vue? [Електронний ресурс] Vue Mastery. – Режим доступу: https://vuejs.org/guide/introduction.html

20. Моххамед Ірфан. Top Manual Testing Interview Questions & Answers [Електронний ресурс] – Режим доступу: https://www.thetestingsquad.in/2023/01/30-software-testing-interview-questions.html?m=1#google_vignette

ДОДАТКИ

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Прогресивний WEB-застосунок для спільного та особистого управління фінансами

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФПТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,81% Схожість 1,19%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Киринюк І.С.

Керівник роботи Сілагін О.В.

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА

1. Після переходу у застосунок, користувачу відкривається головна вітальна сторінка, що зображена на рисунку Б.1, де йому доступні наступні функції: керування балансом, перегляд транзакцій, керування списком бажань, коротка статистика по витратам.

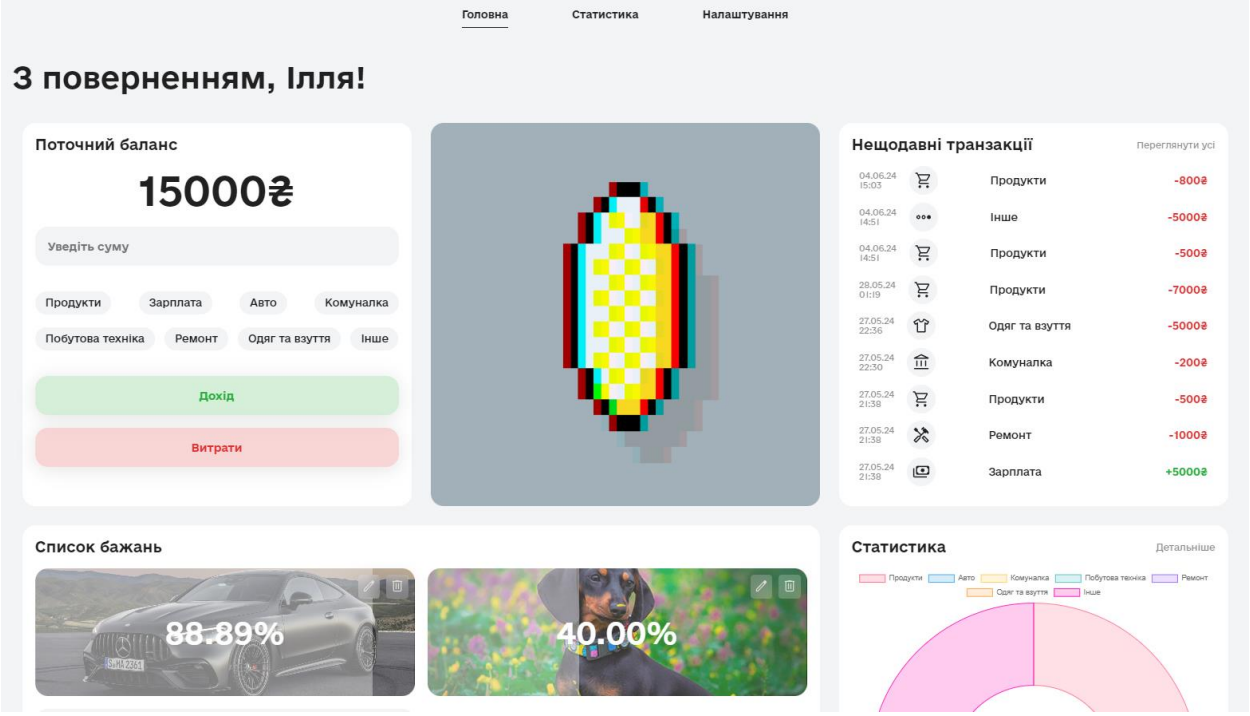


Рисунок Б.1 – Зовнішній вигляд головної сторінки

2. Для керування балансом скористайтеся наступним блоком, який зображено на рисунку Б.2. У цьому блоці потрібно ввести необхідну суму грошей, яку необхідно витрати або добавити. Далі обрати відповідну категорію зі списку, яка найкраще описує тип транзакції, наприклад, "Зарплата" для доходу або "Їжа" для витрат. Після цього натиснути кнопку, що відповідає типу транзакції: "Дохід" для додавання коштів до балансу або "Витрати" для їх списання. Ці дії автоматично оновлять ваш загальний баланс і збережуть нову транзакцію в системі для подальшого аналізу та перегляду.

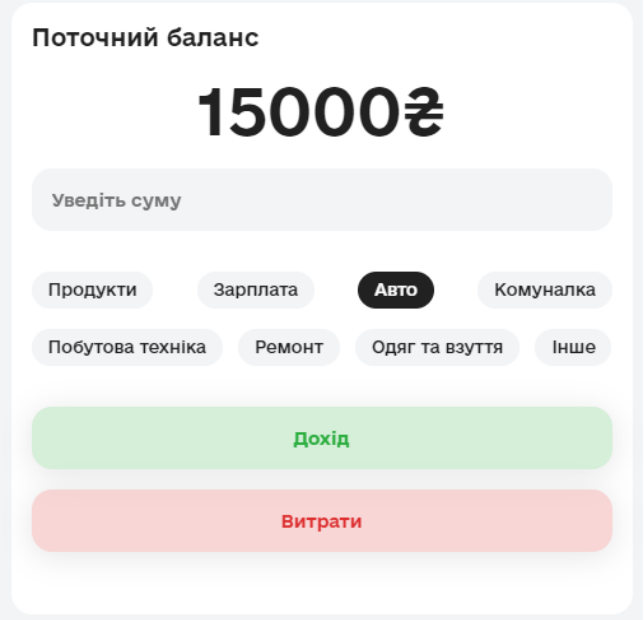


Рисунок Б.2 – Зовнішній вигляд блоку для керування балансом

3. Усі зміни балансу записуються у транзакції, що знаходяться у блокові, який зображений на рисунку Б.3. У ньому відображається останні 10 транзакцій, щоб переглянути усі, необхідно натиснути відповідну кнопку “Переглянути усі”.

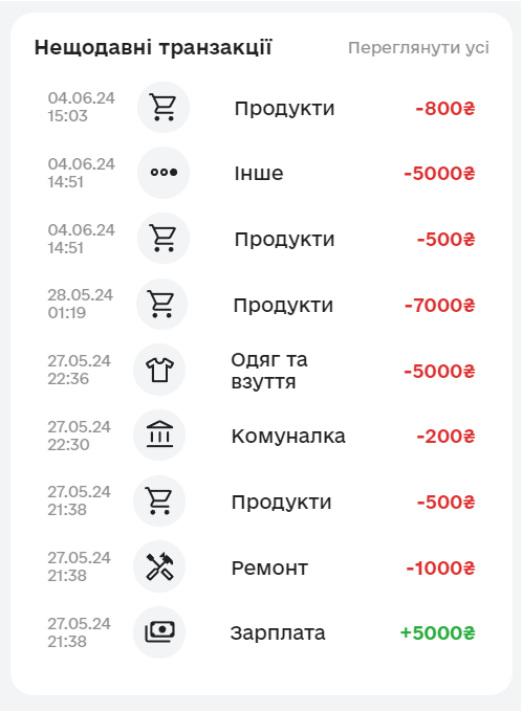


Рисунок Б.3 – Зовнішній вигляд блоку для перегляду транзакцій

4. Для керування списком бажань, необхідно скористатись блоком, який зображено на рисунку Б.4. Щоб добавити нове бажання, необхідно натиснути сірий блок зі знаком “+”, після чого відкриється форма (рис Б.5), де необхідно увести відповідні дані. Для редагування зібраної суми – необхідно натиснути кнопку на відповідному бажанні із зображенням олівця, для видалення – кнопка із зображенням урни для сміття.

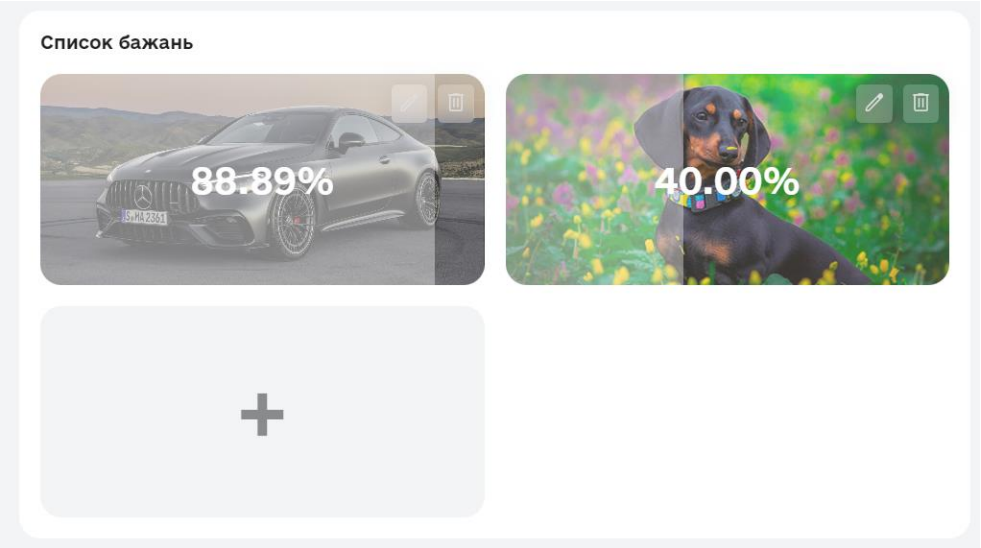


Рисунок Б.4 – Зовнішній вигляд списку бажань

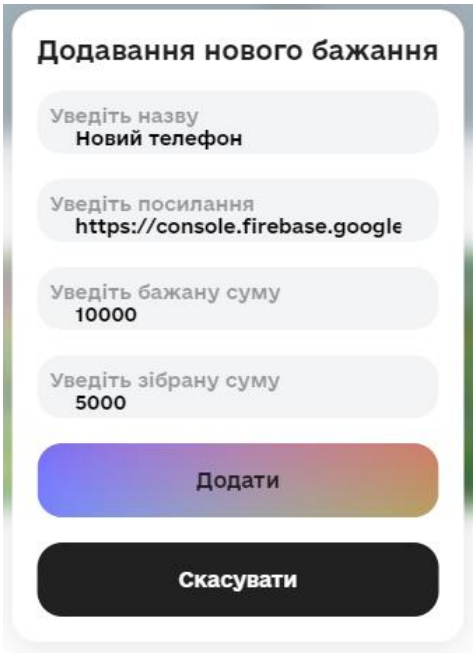


Рисунок Б.5 – Зовнішній вигляд форми для додавання нового бажання

5. Для статистики на головній сторінці є відповідний блок, що зображено на рисунку Б.6, це кругла діаграма, що показує витрати по категоріям за поточний місяць. Для перегляду інших діаграм – необхідно натиснути кнопку “Детальніше”, після чого користувача відправляє на сторінку статистики (рис. Б.7). На даній сторінці є три графіки: перший – кругова діаграма, а над нею діапазон дат, змінюючи діапазон дат - змінюються дані на круговій діаграмі (на рисунку Б.7, зображено кругову діаграму для діапазону чисел 01.04.2024 – 20.06.2024); на другому графіку зображено відношення витрат до прибутку за останні 4 місяці; на третьому – пелюсткова діаграма, що зображує витрати по категоріям цього місяця і минулого.

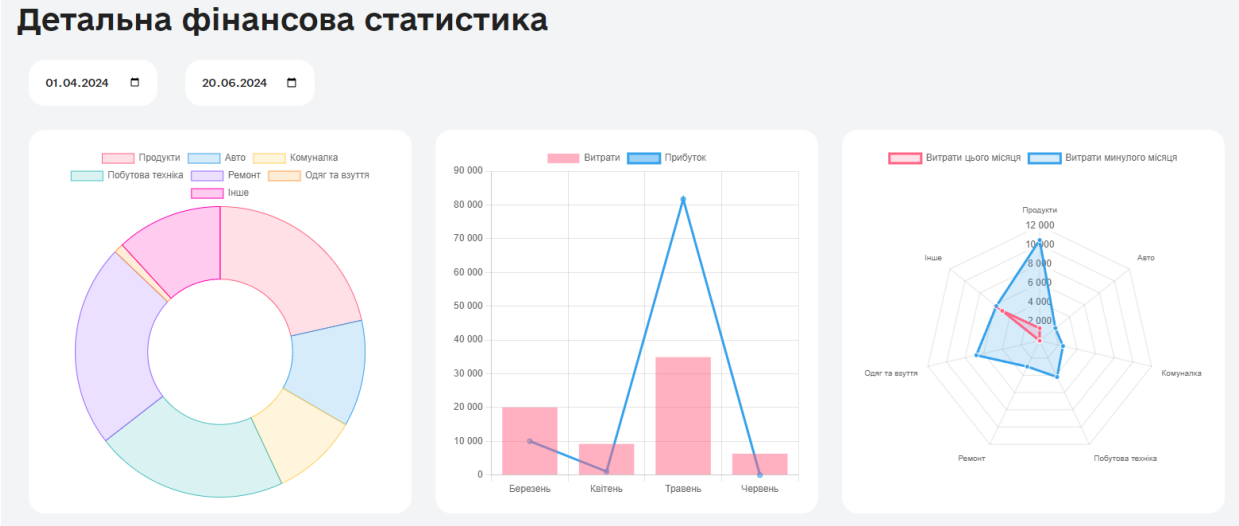


Рисунок Б.7 – Зовнішній вигляд сторінки “Статистика”

6. Сторінка налаштувань, зображена на рисунку Б.8, дозволяє користувачам змінювати різні параметри. На цій сторінці можна налаштувати тло головної сторінки (блок з картинкою на головній сторінці), змінити ім’я користувача або баланс. Для цього потрібно внести відповідні зміни у текстові поля та натиснути кнопку “Зберегти”. Зміни будуть збережені та застосовані до вашого облікового запису.

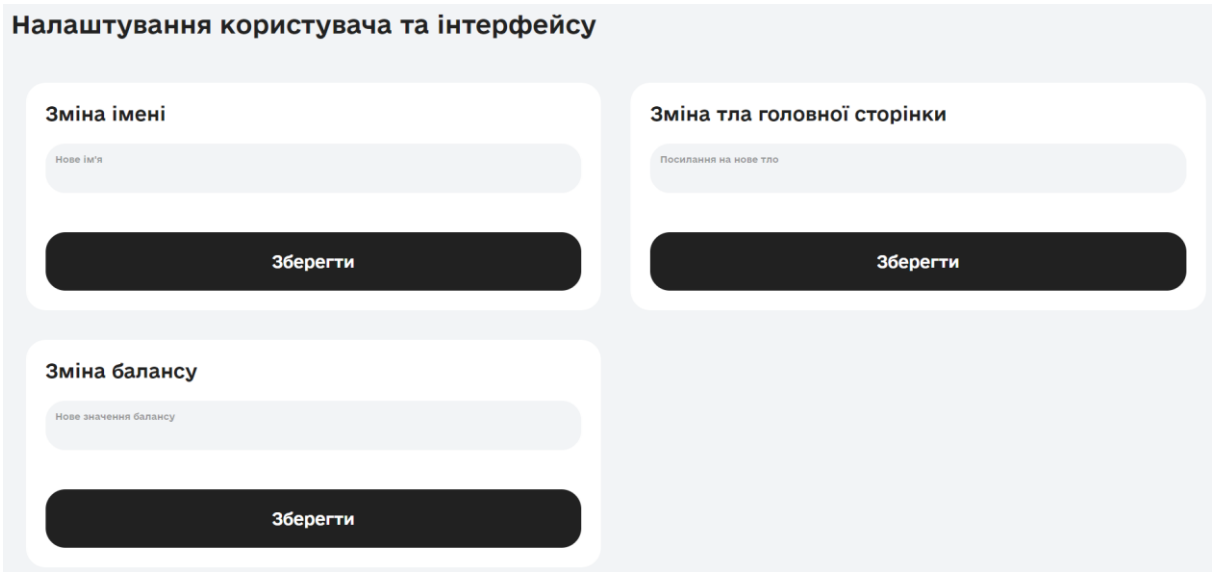


Рисунок Б.8 – Зовнішній вигляд сторінки “Налаштування”

7. Щоб перемикатися між згаданими трьома сторінками, необхідно скористтатись навігаційним меню, зображеним на рисунку Б.9. Це меню дозволяє легко переходити між різними розділами застосунку для зручного користування.

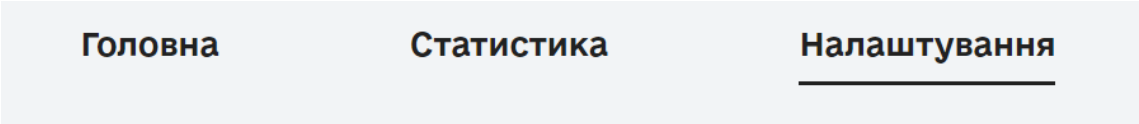


Рисунок Б.9 – Зовнішній вигляд навігації

ДОДАТОК В
ЛІСТИНГ ПРОГРАМИ

useDB.js

```
import { useState } from "react";
import { initializeApp } from "firebase/app";
import {
    getFirestore,
    doc,
    setDoc,
    deleteDoc,
    updateDoc,
    getDoc,
    getDocs,
    query,
    collection,
    enableIndexedDbPersistence
} from "firebase/firestore";


const firebaseConfig = {
};


const app = initializeApp(firebaseConfig);

const db = getFirestore(app);

enableIndexedDbPersistence(db).then(() => {
    console.log("Offline persistence enabled successfully");
}).catch((error) => {
    console.error("Error enabling offline persistence:", error);
});

const UID =’’;
export default function useDB() {
    const [isLoading, setIsLoading] = useState(false);

    async function get(url) {
        let data;
        let path = doc(db, UID + url);
        setIsLoading(true);
        try {
            const response = await getDoc(path)
            if (response.exists()) {
                data = response.data();
            }
        } catch (error) {
            console.error("Error getting document:", error);
        } finally {
```

```
        setIsLoading(false);
    }
    return data;
}

async function create(url, data) {
    setIsLoading(true);
    let path = doc(db, UID + url);
    try {
        await setDoc(path, data);
    } catch(e) {
        console.error("Error creating document:", e);
    } finally {
        setIsLoading(false);
    }
}

async function update(url, data) {
    setIsLoading(true);
    let path = doc(db, UID + url);
    try {
        await updateDoc(path, data);
    } catch(e) {
        console.error("Error updating document:", e);
    } finally {
        setIsLoading(false);
    }
}

async function makeQuery(url) {
    const customQuery = query(collection(db, UID + url));
    const querySnapshot = await getDocs(customQuery);
    let data = [];
    querySnapshot.forEach((snap) => data.push(snap.data()));
    return data;
}

async function deleteRecord(url) {
    setIsLoading(true);
    let path = doc(db, UID + url);
    try {
        await deleteDoc(path);
    } catch(e) {
        console.error("Error deleting document:", e);
    } finally {
        setIsLoading(false);
    }
}

function generateID(length = 20) {
    const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
```

```
        let result = '';
        const charactersLength = characters.length;
        for (let i = 0; i < length; i++) {
          result += characters.charAt(Math.floor(Math.random() * charactersLength));
        }
        return result;
    }

    return {get, create, update, deleteRecord, makeQuery, generateID, isLoading};
}
```

App.js

```
import {BrowserRouter as Router, Route, Routes} from 'react-router-dom';

import Nav from '../nav/Nav';
import MainPage from '../pages/MainPage';
import StatisticsPage from '../pages/StatisticsPage';
import SettingsPage from '../pages/SettingsPage';

import './app.scss';

function App() {
  return (
    <Router>
      <Nav />
        <Routes>
          <Route path='/' element={<MainPage />} />
          <Route path='/statistics' element={<StatisticsPage />} />
          <Route path='/settings' element={<SettingsPage />} />
        </Routes>
    </Router>
  );
}

export default App;
```

Balance.js

```
import { useRef, useState } from 'react';
import useDB from '../../services/useDB';

import './balance.scss';

function Balance({ handleChangeBalance, userBalance }) {
    const { create, generateID } = useDB();
    const [operationType, setOperationType] = useState('');
    const formRef = useRef(null);

    const handleFormSubmit = (e) => {
        e.preventDefault();
        const TodaysDate = formatDate(new Date());
        let formData = new FormData(formRef.current);
        formData.set('operationType', operationType);
```

```
    formData.set('date', TodaysDate);
    let data = Object.fromEntries(formData.entries());
    handleChangeBalance(+data.value, operationType);
    create(`/transaction/${Date.now()}`, data);

    formRef.current.reset();
  }

  return (
    <section className="container balance">
      <div className="container__header">
        <h2>Поточний баланс</h2>
      </div>
      <p className="balance__info">{userBalance}₴</p>
      <form onSubmit={handleFormSubmit} ref={formRef}>
        <input className="balance-input" name="value" type="number" placeholder='Уведіть
суму' required/>
        <input type="hidden" name="id" value={generateID()}></input>
        <div className="category">
          <input id="category1" type="radio" name="category" value="Продукти" required/>
          <label htmlFor="category1" className="radio-btn">Продукти</label>

          <input id="category2" type="radio" name="category" value="Зарплата" required/>
          <label htmlFor="category2" className="radio-btn">Зарплата</label>

          <input id="category3" type="radio" name="category" value="Авто" required/>
          <label htmlFor="category3" className="radio-btn">Авто</label>

          <input id="category4" type="radio" name="category" value="Комуналка" required/>
          <label htmlFor="category4" className="radio-btn">Комуналка</label>

          <input id="category5" type="radio" name="category" value="Побутова техніка"
required/>
          <label htmlFor="category5" className="radio-btn">Побутова техніка</label>

          <input id="category6" type="radio" name="category" value="Ремонт" required/>
          <label htmlFor="category6" className="radio-btn">Ремонт</label>

          <input id="category7" type="radio" name="category" value="Одяг та взуття" required/>
          <label htmlFor="category7" className="radio-btn">Одяг та взуття</label>

          <input id="category8" type="radio" name="category" value="Інше" required/>
          <label htmlFor="category8" className="radio-btn">Інше</label>
        </div>

        <input type="hidden" name="operationType" value="" />
        <button
          className="balance__button balance__button_income"
          onClick={() => setOperationType('income')}
        >
          Дохід
        </button>
```

```
            <button
              className="balance__button balance__button_spending"
              onClick={() => setOperationType('spending')}
            >
              Витрати
            </button>
          </form>
        </section>
    )
}

function formatDate(date) {
    const day = String(date.getDate()).padStart(2, '0');
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const year = String(date.getFullYear()).slice(-2);
    const hours = String(date.getHours()).padStart(2, '0');
    const minutes = String(date.getMinutes()).padStart(2, '0');

    return `${day}.${month}.${year} ${hours}:${minutes}`;
}


export default Balance;
```

Form.js

```
import { useRef, useEffect, useState } from 'react';
import useDB from '../../services/useDB';

import './form.scss';

function Form({ isFormOpen, handleChangeFormStatus, mode = 'add', initialData = {} }) {
    const { create, update } = useDB();
    const formRef = useRef(null);

    const handleFormSubmit = (e) => {
        e.preventDefault();
        let id = mode === 'add' ? Date.now() : initialData.id;
        let formData = new FormData(formRef.current);
        formData.set('id', id);
        let data = Object.fromEntries(formData.entries());
        console.log(data);
        if (mode === 'add') {
            create(`/wishlist/${id}`, data).then(()=>handleChangeFormStatus());
        } else {
            update(`/wishlist/${id}`, {'id': id, done: +data.addSum +
+initialData.done}).then(()=>handleChangeFormStatus());
        }
    };

    useEffect(() => {
        if (isFormOpen) {
            document.body.classList.add('body-no-scroll');
```

```
    } else {
      document.body.classList.remove('body-no-scroll');
    }

    return () => {
      document.body.classList.remove('body-no-scroll');
    };
  }, [isFormOpen]);

  const FormFilling = () => {
    return (
      isFormOpen ?
        <div className="popup__bg">
          <form       className="form       popup__window"       onSubmit={handleFormSubmit}
ref={formRef}>
            <h2>{mode === 'add' ? 'Додавання нового бажання' : 'Редагування бажання'}</h2>
            {mode === 'add' && (
              <>
                <div className='form__field'>
                  <label htmlFor="title">Уведіть назву</label>
                  <input id="title" type="text" name="title" required/>
                </div>
                <div className='form__field'>
                  <label htmlFor="link">Уведіть посилання</label>
                  <input id="link" type="text" name="link" required/>
                </div>
                <div className='form__field'>
                  <label htmlFor="sum">Уведіть бажану суму</label>
                  <input id="sum" type="number" name="sum" required/>
                </div>
                <div className='form__field'>
                  <label htmlFor="done">Уведіть зібрану суму</label>
                  <input id="done" type="number" name="done" required/>
                </div>
              </>
            )}
            {mode === 'edit' && (
              <>
                <h6>Зібрана сума: {initialData.done}</h6>
                <h6>Необхідна сума: {initialData.sum}</h6>
                <div className='form__field'>
                  <label htmlFor="addSum">Додати до зібраної суми</label>
                  <input id="addSum" type="number" name="addSum" required/>
                </div>
              </>
            )}
            <button type="submit" className='btn'>Додати</button>
            <button
              type="button"
              className='btn'
              onClick={handleChangeFormStatus}>Скасувати</button>
          </form>
```

```
            </div> : null
        );
    };

    return <FormFilling />;
}

export default Form;
```

Nav.js

```
import { NavLink } from 'react-router-dom';
import './nav.scss';

import homeIcon from '../../resources/img/icons/home.svg';
import monitoringIcon from '../../resources/img/icons/monitoring.svg';
import settingIcon from '../../resources/img/icons/setting.svg';

function AppHeader() {
    return (
        <>
            <nav className="nav">
                <ul className="nav__link">
                    <li><NavLink to='/' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}>Головна</NavLink></li>
                    <li><NavLink to='/statistics' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}>Статистика</NavLink></li>
                    <li><NavLink to='/settings' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}>Налаштування</NavLink></li>
                </ul>
            </nav>
            <nav className="nav_bottom">
                <ul className="nav__link">
                    <li><NavLink to='/' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}><img src={homeIcon} alt="home icon" /></NavLink></li>
                    <li><NavLink to='/statistics' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}><img src={monitoringIcon} alt="statistics icon" /></NavLink></li>
                    <li><NavLink to='/settings' style={({isActive}) => ({borderBottom: isActive ? '2px solid' : 'none'})}><img src={settingIcon} alt="settings icon" /></NavLink></li>
                </ul>
            </nav>
        </>

    )
}

export default AppHeader;
```

MainPage.js

```
import useDB from '../../services/useDB';
import { useState, useEffect } from 'react';
```

```
import Balance from '../balance/Balance';
import Quotes from '../quotes/Quotes';
import Transactions from '../transactions/Transactions';
import Wishlist from '../wishlist/Wishlist';
import Statistics from '../statistics/Statistics';
import Form from '../form/Form';

import './mainPage.scss';

function MainPage() {
   const url = '/';
   const [userData, setUserData] = useState();
   const [dbUpdateChecker, setDbUpdateChecker] = useState(false);
   const {get, update} = useDB();
   const [isFormOpen, setIsFormOpen] = useState(false);
   const [initialWishData, setInitialWishData] = useState(null);
   const [editMode, setEditMode] = useState(false);

   const {makeQuery} = useDB();
   const [transactionData, setTransactionData] = useState();

   useEffect(() => {
      get(url).then(data => setUserData(data));
      makeQuery('/transaction/').then((data) => {setTransactionData(data.reverse())});
   }, [dbUpdateChecker]);

   const onChangeBalance = (newVal, operationType) => {
      update(url, {balance: operationType === 'income' ? +userData.balance + newVal :
+userData.balance - newVal});
      setDbUpdateChecker(!dbUpdateChecker);
   }

   const onChangeFormStatus = () => {
      setIsFormOpen(!isFormOpen);
      setEditMode(false);
   }

   const onEditWish = (initialData) => {
      onChangeFormStatus();
      setInitialWishData(initialData);
      setEditMode(true);
   }

   return (
      <>
         <header className="header__main">
            <h1>З поверненням, {userData && userData.name}!</h1>
         </header>

         <main className="wrapper">
            <Balance
               handleChangeBalance={onChangeBalance}
```

```
            userBalance={userData && userData.balance}/>
          <Quotes
            backgroundUrl={userData && userData.background}
          />
          <Transactions
            transactionData={transactionData}
          />
          <Wishlist
            isChangedWishlist={isFormOpen}
            handleAddNewWish={onChangeFormStatus}
            handleEditWish={onEditWish}/>
          <Statistics
            transactionData={transactionData}/>
          <Form
            isFormOpen={isFormOpen}
            handleChangeFormStatus={onChangeFormStatus}
            mode={editMode ? 'edit' : 'add'}
            initialData={initialWishData}
             />
        </main>
      </>

  )
}

export default MainPage;
```

SettingsPage.js

```
import { useRef } from 'react';
import useDB from '../../services/useDB'

function SettingsPage() {
  const { update } = useDB();

  const nameFormRef = useRef(null);
  const bgFormRef = useRef(null);
  const balanceFormRef = useRef(null);

  const handleFormSubmit = (e, formRef) => {
    e.preventDefault();
    const formData = new FormData(formRef.current);
    let data = Object.fromEntries(formData.entries());

    update('/', data);

    formRef.current.reset();
  }

  return (
    <>
      <header className="header__main">
        <h1>Налаштування користувача та інтерфейсу</h1>
```

```
        </header>

        <main className="wrapper ">
          <section className="container form form_settings">
            <h3>Зміна імені</h3>
            <form      className="form__field"      ref={nameFormRef}      onSubmit={(e)      =>
handleFormSubmit(e, nameFormRef)}>
              <label htmlFor="profile-name">Нове ім'я</label>
              <input id="profile-name" name="name" type="text"></input>
              <button className="btn" type="submit">Зберегти</button>
            </form>
          </section>

          <section className="container form form_settings">
            <h3>Зміна тла головної сторінки</h3>
            <form      className="form__field"      ref={bgFormRef}      onSubmit={(e)      =>
handleFormSubmit(e, bgFormRef)}>
              <label htmlFor="profile-bg">Посилання на нове тло</label>
              <input id="profile-bg" name="background" type="text"></input>
              <button className="btn" type="submit">Зберегти</button>
            </form>
          </section>

          <section className="container form form_settings">
            <h3>Зміна балансу</h3>
            <form      className="form__field"      ref={balanceFormRef}      onSubmit={(e)      =>
handleFormSubmit(e, balanceFormRef)}>
              <label htmlFor="profile-balance">Нове значення балансу</label>
              <input id="profile-balance" name="balance" type="number"></input>
              <button className="btn" type="submit">Зберегти</button>
            </form>
          </section>
        </main>
      </>
    )
}

export default SettingsPage;
```

StatisticsPage.js

```
import { useState, useEffect } from 'react';

import useDB from '../../services/useDB';

import Statistics from '../statistics/Statistics';

function StatisticsPage() {
  const { makeQuery } = useDB();

  const [transactionData, setTransactionData] = useState();

  useEffect(() => {
```

```
    makeQuery('/transaction/').then((data) => {setTransactionData(data.reverse())});
  }, []);

  return (
    <Statistics
      transactionData={transactionData}
      flag={true} />
  )
}

export default StatisticsPage;
```

Quotes.js

```
import './quotes.scss';
import defaultBackground from '../../resources/img/bg.gif'
function Quotes({ backgroundUrl }) {

  return (
    <section
      className="container quotes"
      style={{
        backgroundImage: `url(${backgroundUrl || defaultBackground})`,
        backgroundSize: 'cover',
        backgroundRepeat: 'no-repeat',
        backgroundPosition: 'center',
      }}
    >
    </section>
  );
}

export default Quotes;
```

Statistics.js

```
import { useState, useEffect } from 'react';
import { NavLink } from 'react-router-dom';
import { Doughnut, Scatter, Radar } from 'react-chartjs-2';
import { Chart, registerables } from 'chart.js';
import './statistics.scss';

Chart.register(...registerables);

function Statistics({ transactionData, flag = false }) {
  const [startDate, setStartDate] = useState('01.04.24');
  const [endDate, setEndDate] = useState('20.05.24');
  const [dateForInput, setDateForInput] = useState({
    start: '2024-04-01',
    end: '2024-05-20',
  });
  const [dateRange, setDateRange] = useState([]);

  useEffect(generateDateRange, [startDate, endDate]);
```

```
const categories = [
    'Продукти',
    'Авто',
    'Комуналка',
    'Побутова техніка',
    'Ремонт',
    'Одяг та взуття',
    'Інше',
];

const dataMap = new Map(categories.map((category) => [category, 0]));
const prevMonthDataMap = new Map(categories.map((category) => [category, 0]));
const filteredDataMap = new Map(categories.map((category) => [category, 0]));

let monthlyIncome = new Array(12).fill(0);
let monthlySpending = new Array(12).fill(0);

function generateDateRange() {
    let startDateArr = startDate.split('.');
    let endDateArr = endDate.split('.');
    let tempArr = [];

    let startDay = +startDateArr[0];
    let startMonth = +startDateArr[1];
    let startYear = +startDateArr[2];

    let endDay = +endDateArr[0];
    let endMonth = +endDateArr[1];
    let endYear = +endDateArr[2];

    let startDateObject = new Date(`20${startYear}-${startMonth}-${startDay}`);
    let endDateObject = new Date(`20${endYear}-${endMonth}-${endDay}`);

    while (startDateObject <= endDateObject) {
        let day = startDateObject.getDate();
        let month = startDateObject.getMonth() + 1;
        let year = startDateObject.getFullYear().toString().slice(-2);

        let formattedDay = day < 10 ? `0${day}` : day;
        let formattedMonth = month < 10 ? `0${month}` : month;

        tempArr.push(`${formattedDay}.${formattedMonth}.${year}`);

        startDateObject.setDate(startDateObject.getDate() + 1);
    }

    setDateRange(tempArr);
}

transactionData?.forEach(({ category, operationType, value, date }) => {
    const [day, month, year] = date.split('.').map(Number);
```

```
    const currentMonth = new Date().getMonth() + 1;

    if (operationType === 'spending') {
      monthlySpending[month - 1] += +value;
      if (month === currentMonth) {
        dataMap.set(category, (dataMap.get(category) || 0) + Number(value));
      } else if (month === currentMonth - 1) {
        prevMonthDataMap.set(category, (prevMonthDataMap.get(category) || 0) + Number(value));
      }
      if (dateRange.includes(date.slice(0, 8))) {
        filteredDataMap.set(category, (filteredDataMap.get(category) || 0) + Number(value));
      }
    }

    if (operationType === 'income') {
      monthlyIncome[month - 1] += +value;
    }
  });

  const labels = Array.from(dataMap.keys());
  const dataValues = flag ? Array.from(filteredDataMap.values()) : Array.from(dataMap.values());

  const handleChangeDate = (e) => {
    const { name, value } = e.target;
    const dateArr = value.split('-').reverse();
    dateArr[2] = dateArr[2].slice(2, 4);
    const date = dateArr.join('.');
    if (name === 'start') {
      setDateForInput({ ...dateForInput, start: value });
      setStartDate(date);
    } else {
      setDateForInput({ ...dateForInput, end: value });
      setEndDate(date);
    }
  };

  const chartData = {
    labels,
    datasets: [
      {
        label: 'Витрати',
        data: dataValues,
        backgroundColor: [
          'rgba(255, 99, 132, 0.2)',
          'rgba(54, 162, 235, 0.2)',
          'rgba(255, 206, 86, 0.2)',
          'rgba(75, 192, 192, 0.2)',
          'rgba(153, 102, 255, 0.2)',
          'rgba(255, 159, 64, 0.2)',
          'rgba(255, 0, 179, 0.2)',
        ],
        borderColor: [
```

```
            'rgba(255, 99, 132, 1)',
            'rgba(54, 162, 235, 1)',
            'rgba(255, 206, 86, 1)',
            'rgba(75, 192, 192, 1)',
            'rgba(153, 102, 255, 1)',
            'rgba(255, 159, 64, 1)',
            'rgba(255, 0, 179, 1)',
        ],
        borderWidth: 1,
    },
  ],
};

const chartOptions = {
  responsive: true,
  plugins: {
    legend: {
      position: 'top',
    },
  },
  hoverOffset: 20,
};

const ScatterChart = () => (
  <Scatter
    datasetIdKey="id"
    data={data3}
    options={{
      scales: {
        y: {
          beginAtZero: true,
        },
      },
      maintainAspectRatio: false,
    }}
  />
);

const data2 = {
  labels: labels,
  datasets: [
    {
      label: 'Витрати цього місяця',
      data: Array.from(dataMap.values()),
      fill: true,
      backgroundColor: 'rgba(255, 99, 132, 0.2)',
      borderColor: 'rgb(255, 99, 132)',
      pointBackgroundColor: 'rgb(255, 99, 132)',
      pointBorderColor: '#fff',
      pointHoverBackgroundColor: '#fff',
      pointHoverBorderColor: 'rgb(255, 99, 132)',
    },
```

```
        {
          label: 'Витрати минулого місяця',
          data: Array.from(prevMonthDataMap.values()),
          fill: true,
          backgroundColor: 'rgba(54, 162, 235, 0.2)',
          borderColor: 'rgb(54, 162, 235)',
          pointBackgroundColor: 'rgb(54, 162, 235)',
          pointBorderColor: '#fff',
          pointHoverBackgroundColor: '#fff',
          pointHoverBorderColor: 'rgb(54, 162, 235)',
        },
      ],
    };

    const currentMonth = new Date().getMonth() + 1;
    const monthsToShow = [currentMonth - 3, currentMonth - 2, currentMonth - 1,
currentMonth].map(month => {
      if (month <= 0) month += 12;
      return month;
    });

    const spendingData = monthsToShow.map(month => monthlySpending[month - 1]);
    const incomeData = monthsToShow.map(month => monthlyIncome[month - 1]);

    const data3 = {
      datasets: [
        {
          type: 'bar',
          label: 'Витрати',
          data: spendingData,
          borderColor: 'rgb(255, 99, 132)',
          backgroundColor: 'rgba(255, 99, 132, 0.5)',
        },
        {
          type: 'line',
          label: 'Прибуток',
          data: incomeData,
          borderColor: 'rgb(53, 162, 235)',
          backgroundColor: 'rgba(53, 162, 235, 0.5)',
        },
      ],
      labels: monthsToShow.map(month => {
        const monthNames = ['Січень', 'Лютий', 'Березень', 'Квітень', 'Травень', 'Червень', 'Липень',
'Серпень', 'Вересень', 'Жовтень', 'Листопад', 'Грудень'];
        return monthNames[month - 1];
      }),
    };

    const RadarChart = () => (
      <Radar
        data={data2}
        options={{
```

```
        elements: {
          line: {
            borderWidth: 3,
          },
        },
        maintainAspectRatio: false,
      }}
    />
);

const DoughnutChart = () => <Doughnut data={chartData} options={chartOptions} />;

const StatisticsBlock = () => {
  if (flag) {
    return (
      <>
        <header className="header__main">
          <h1>Детальна фінансова статистика</h1>
        </header>
        <main className="full-statistics__wrapper">
          <section className="full-statistics__filter">
            <input
              type="date"
              name="start"
              value={dateForInput.start}
              onChange={handleChangeDate}
            />
            <input
              type="date"
              name="end"
              value={dateForInput.end}
              onChange={handleChangeDate}
            />
          </section>

          <section className="full-statistics__blocks">
            <div>
              <DoughnutChart />
            </div>
            <div>
              <ScatterChart />
            </div>
            <div>
              <RadarChart />
            </div>
          </section>
        </main>
      </>
    );
  } else {
    return (
      <section className="container statistics">
```

```
                <div className="container__header">
                    <h2>Статистика</h2>
                    <NavLink to="/statistics">Детальніше</NavLink>
                </div>
                <div className="statistics__blocks">
                    <DoughnutChart />
                </div>
            </section>
        );
    }
};

return <StatisticsBlock />;
}

export default Statistics;
```

Transaction.js

```
import shopCartIcon from '../../resources/img/icons/shopping-cart.svg';
import salaryIcon from '../../resources/img/icons/money.svg';
import carIcon from '../../resources/img/icons/car.svg';
import paymentsIcon from '../../resources/img/icons/payments.svg';
import repairIcon from '../../resources/img/icons/repair.svg';
import techniqueIcon from '../../resources/img/icons/technique.svg'
import clothesIcon from '../../resources/img/icons/tshirt.svg'
import otherIcon from '../../resources/img/icons/dots.svg'

function Transaction({category, value, operationType, date}) {
    let iconUrl;
    let iconDescr;
    switch(category) {
        case 'Продукти':
            iconUrl = shopCartIcon;
            iconDescr = 'shop cart';
            break;
        case 'Зарплата':
            iconUrl = salaryIcon;
            iconDescr = 'money';
            break;
        case 'Авто':
            iconUrl = carIcon;
            iconDescr = 'car';
            break;
        case 'Комуналка':
            iconUrl = paymentsIcon;
            iconDescr = 'utilities'
            break;
        case 'Побутова техніка':
            iconUrl = techniqueIcon;
            iconDescr = 'technique';
            break;
        case 'Ремонт':
```

```
            iconUrl = repairIcon;
            iconDescr = 'repair'
            break;
        case 'Одяг та взуття':
            iconUrl = clothesIcon;
            iconDescr = 'clothes'
            break;
        default:
            iconUrl = otherIcon;
            iconDescr = 'dots'
            break;
    }
    const TransactionBody = () => {
        if (operationType === 'income') {
            return (
                <>
                    <p className='transaction__date'>{date}</p>
                    <span className="transaction__icon"><img src={iconUrl} alt={iconDescr} /></span>
                    <p>{category}</p>
                    <p className="transaction__info_income">+{value}₴</p>
                </>
            )
        } else {
            return (
                <>
                    <p className='transaction__date'>{date}</p>
                    <span className="transaction__icon"><img src={iconUrl} alt={iconDescr} /></span>
                    <p>{category}</p>
                    <p className="transaction__info_spending">-{value}₴</p>
                </>
            )
        }
    }
    return (
        <div className="transaction__block">
            <TransactionBody />
        </div>
    )
}

export default Transaction;
```

Transactions.js

```
import { useEffect, useState } from 'react';

import Transaction from '../transaction/Transaction';
import './transactions.scss';

function Transactions({transactionData}) {

    const [isWindowOpen, setIsWindowOpen] = useState(false);
```

```
useEffect(() => {
    if (isWindowOpen) {
        document.body.classList.add('body-no-scroll');
    } else {
        document.body.classList.remove('body-no-scroll');
    }

    return () => {
        document.body.classList.remove('body-no-scroll');
    };
}, [isWindowOpen]);

const PopupTransaction = () => {
    return (
        <div className='popup__bg popup__transaction'>
            <div className='popup__window'>
                <h2>Усі транзакції</h2>
                {transactionData && transactionData.map((transact) => (
                <Transaction
                    key={transact.id}
                    category={transact.category}
                    value={transact.value}
                    operationType={transact.operationType}
                    date={transact.date}
                />
            ))}
            <button className="btn" onClick={() => setIsWindowOpen(false)}>Закрити</button>
            </div>

        </div>
    )
}
return (
    <section className="container transaction">
        <div className="container__header">
            <h2>Нещодавні транзакції</h2>
            <button onClick={() => setIsWindowOpen(true)}>Переглянути усі</button>
        </div>
        {transactionData && transactionData.slice(0, 9).map((transact) => {
        return (
            <Transaction
                key={transact.id}
                category={transact.category}
                value={transact.value}
                operationType={transact.operationType}
                date={transact.date}
            />
        )})}

        {isWindowOpen ? <PopupTransaction /> : null}

    </section>
```

```
    )
}

export default Transactions;
```

WishBlock.js

```
import deleteIcon from '../../resources/img/icons/delete.svg';
import editIcon from '../../resources/img/icons/edit.svg';

function WishBlock({id, done, sum, imgUrl, handleDeleteWish, handleEditWish}) {
    let remain = sum - done;
    let remainInPerc = (100*remain)/sum;
    let doneInPerc = 100 - remainInPerc;

    return (
        <div
            className="wishlist__block"
            style={{backgroundImage: `url(${imgUrl})`}}
        >
            <div className="wishlist__btns">
                <button
                    className="wishlist__btn"
                    onClick={()=>handleEditWish({id, done, sum})}
                ><img src={editIcon} alt='edit icon'/></button>
                <button
                    className="wishlist__btn"
                    onClick={()=>handleDeleteWish(id)}><img src={deleteIcon} alt='delete icon'/></button>
            </div>
            <div className="wishlist__progress-bar">
                <div                    className="wishlist__progress-bar_done"                    style={{width:
`${doneInPerc}%`}}></div>
                <p>{doneInPerc.toFixed(2)}%</p>
                <div                    className="wishlist__progress-bar_remain"                    style={{width:
`${remainInPerc}%`}}></div>
            </div>
        </div>
    )
}

export default WishBlock;
```

Wishlist.js

```
import { useState, useEffect } from 'react';

import useDB from '../../services/useDB';

import WishBlock from '../wishBlock/WishBlock';
import './wishlist.scss';

function Wishlist({handleAddNewWish, isChangedWishlist, handleEditWish}) {
    const { makeQuery, deleteRecord } = useDB();
    const [wishlistData, setWishlistData] = useState();
```

```
    const [isDeleted, setIsDeleted] = useState(false)

    useEffect(() => {
        makeQuery('/wishlist/').then(data => {setWishlistData(data)});
    }, [isChangedWishlist, isDeleted])

    const handleDeleteWish = (id) => {
        deleteRecord(`/wishlist/${id}`);
        setIsDeleted(!isDeleted);
    }

    return (
        <section className="container wishlist">
            <div className="container__header">
                <h2>Список бажань</h2>
            </div>
            <div className="wishlist__blocks">
            {wishlistData && wishlistData.map((item) => (
                <WishBlock
                    key={item.id}
                    id={item.id}
                    done={item.done}
                    sum={item.sum}
                    imgUrl={item.link}
                    handleDeleteWish={handleDeleteWish}
                    handleEditWish={handleEditWish}
                />
            ))}
                <button onClick={handleAddNewWish} className="wishlist__block wishlist__block_add-
new">
                    <p>+</p>
                </button>
            </div>
        </section>
    )
}

export default Wishlist;
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

“ПРОГРЕСИВНИЙ WEB-ЗАСТОСУНОК ДЛЯ СПІЛЬНОГО ТА ОСОБИСТОГО УПРАВЛІННЯ ФІНАНСАМИ”

Виконав: студент 4 курсу, групи ЗКН-20б
спеціальності 122 – Комп’ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Киринюк І.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. КН
Сілагін О.В.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

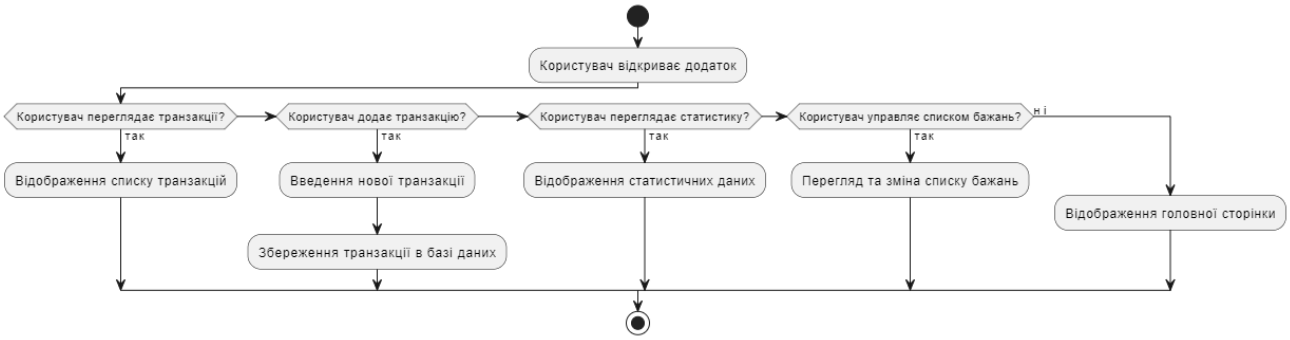


Рисунок Г.1 – UML-діаграма активності

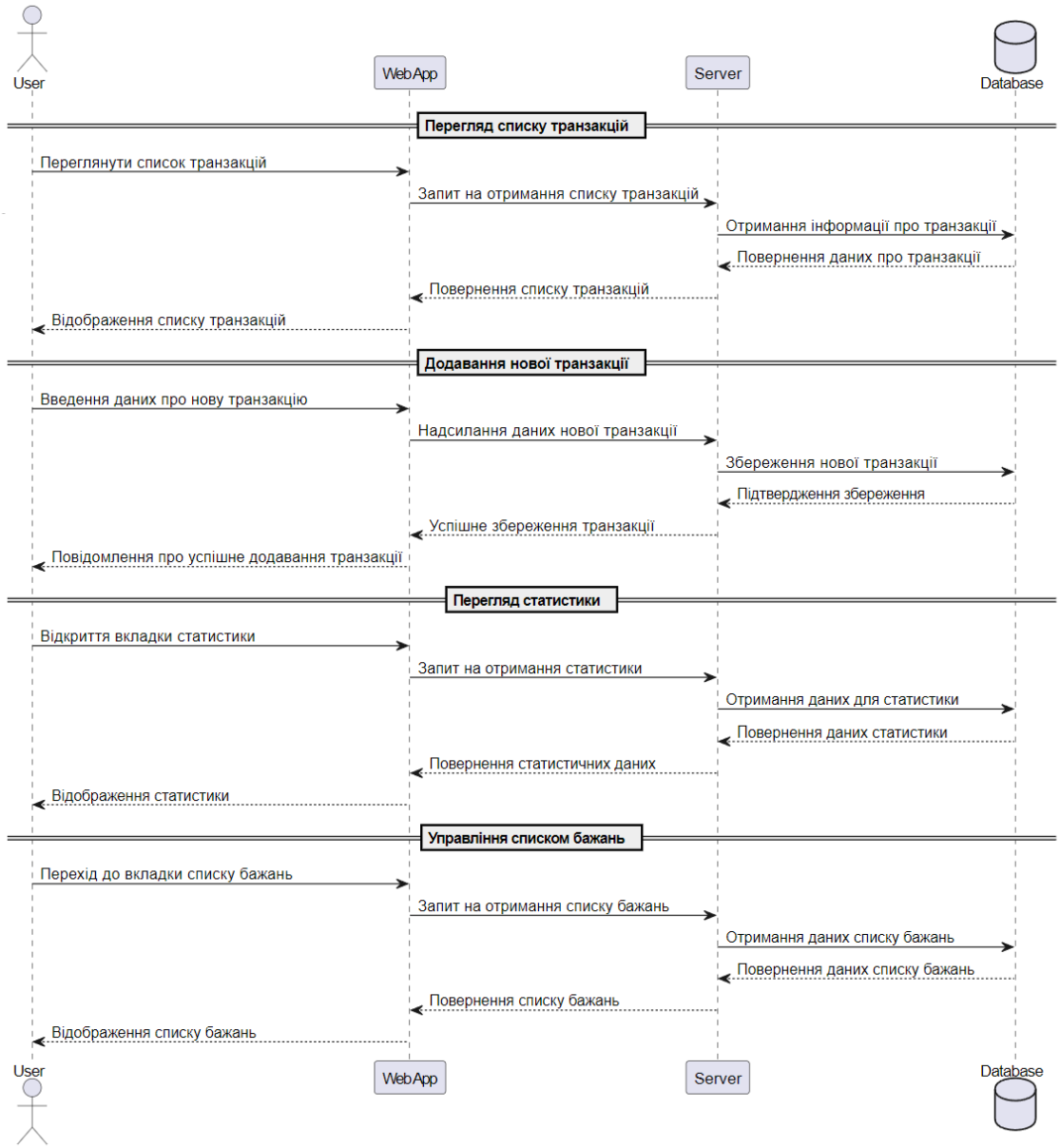


Рисунок Г.2 – UML-діаграма послідовностей

Рисунок Г.3 – Зовнішній вигляд навігації

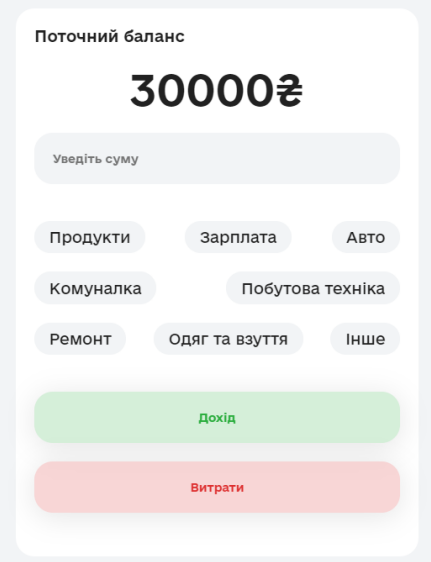


Рисунок Г.4 – Зовнішній вигляд компонента для керування балансом

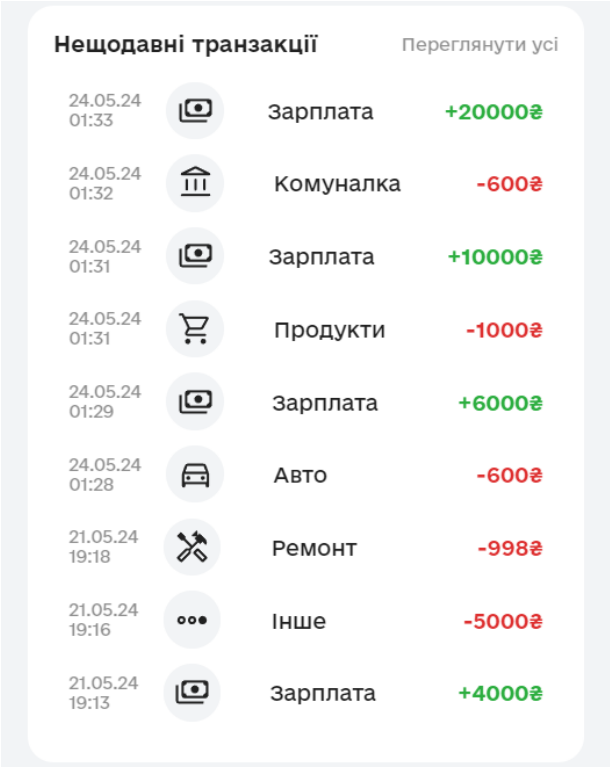


Рисунок Г.5 – Зовнішній вигляд компонента для відображення нещодавніх транзакцій

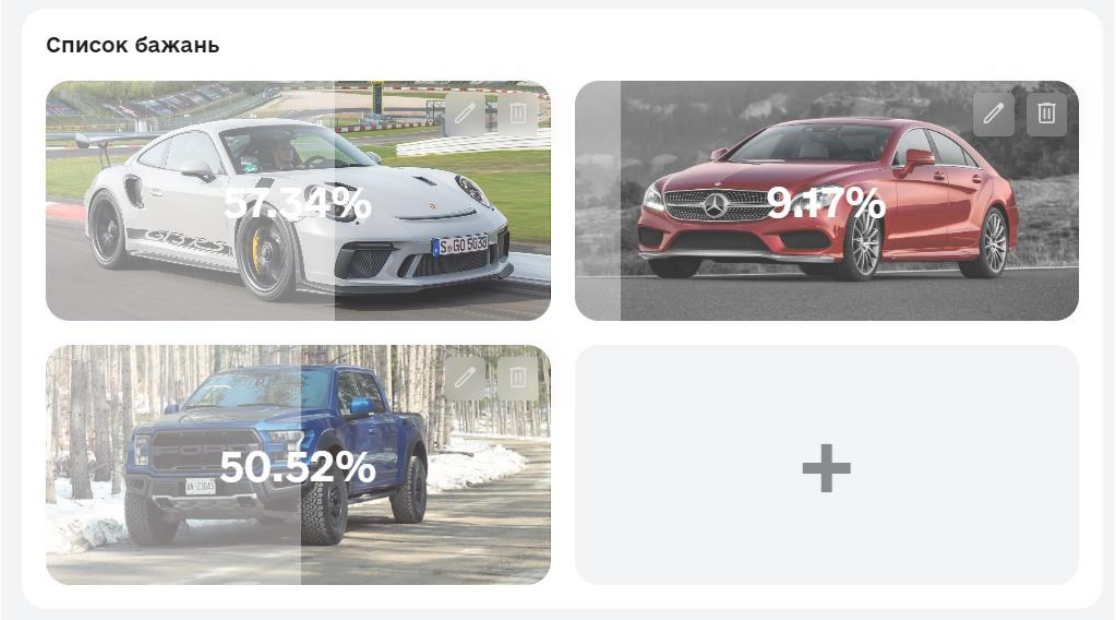


Рисунок Г.6 - Зовнішній вигляд компонента для керування списком бажань

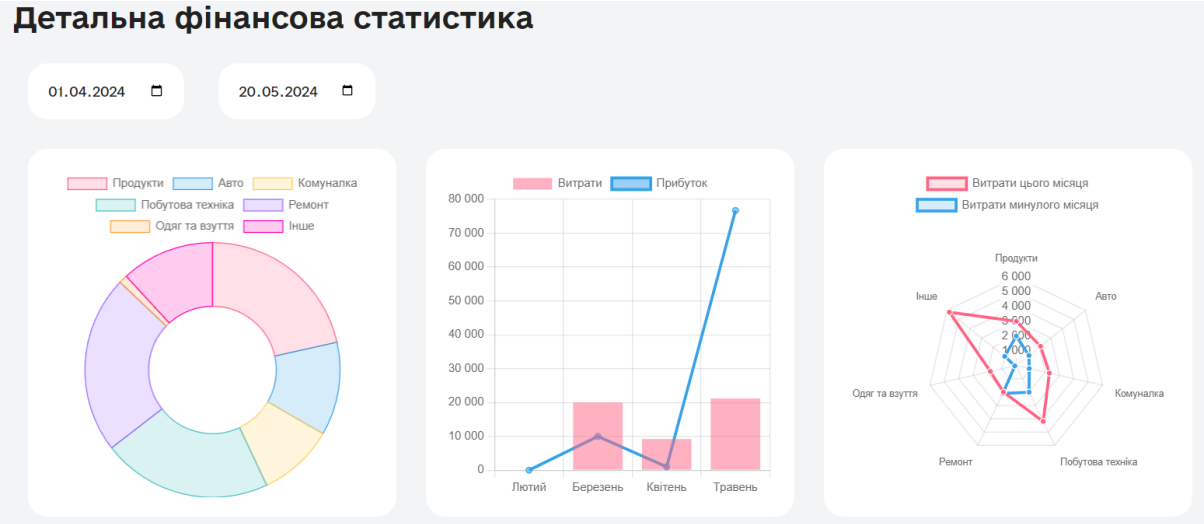


Рисунок Г.7 – Зовнішній вигляд сторінки “Статистика”

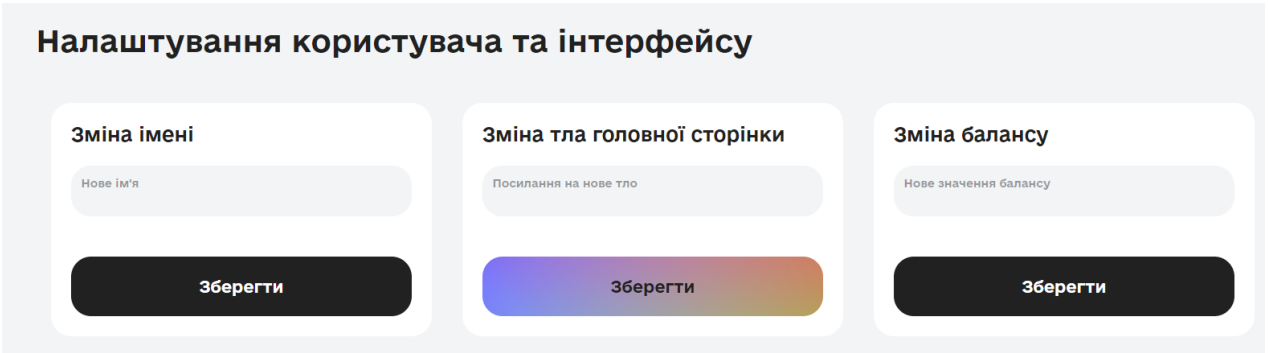


Рисунок Г.8 – Зовнішній вигляд сторінки “Налаштування”

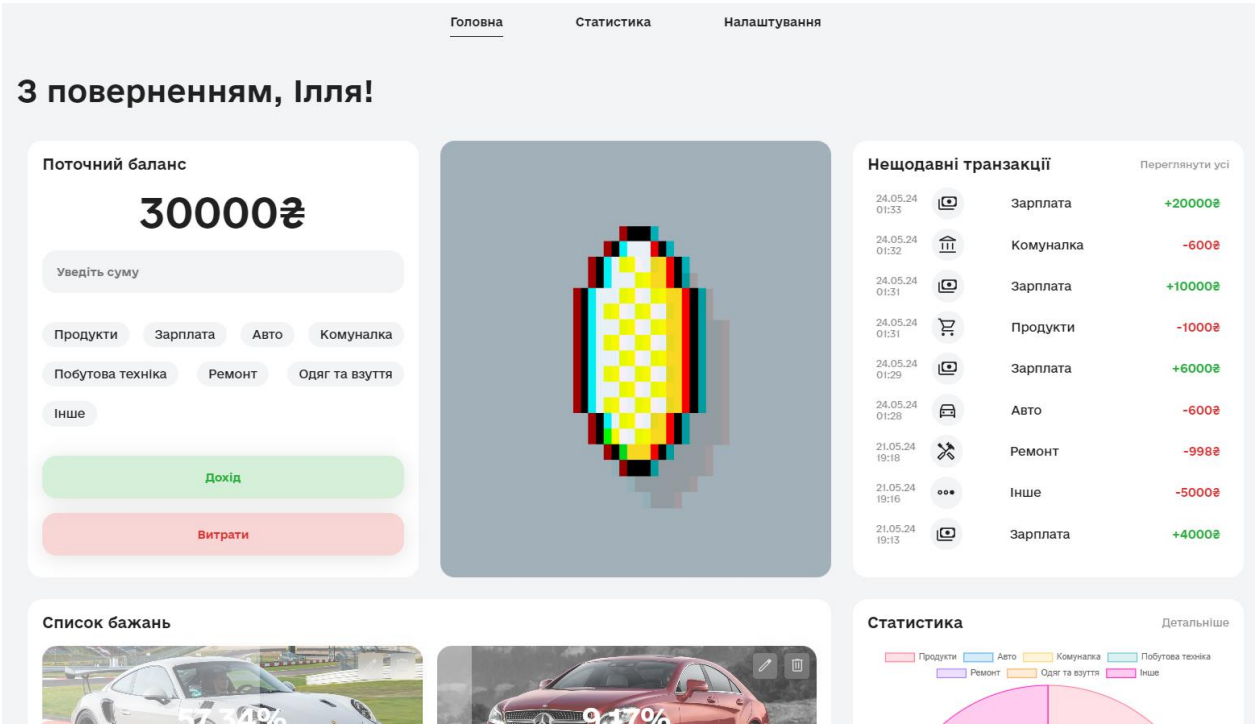


Рисунок Г.9 – Вигляд веб-ресурсу на комп’ютері

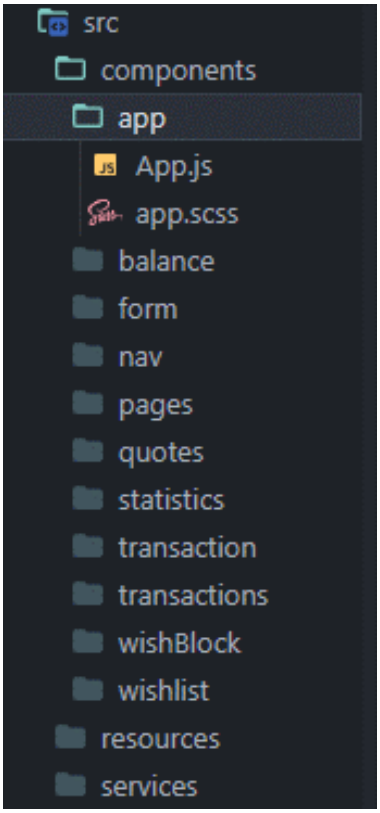


Рисунок Г.10 – Структура проєкту

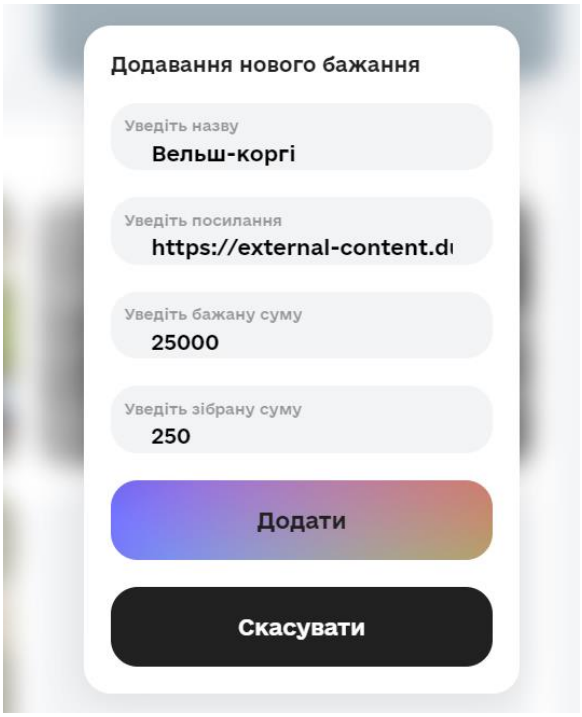


Рисунок Г.11 - Вигляд форми для створення нового бажання з внесеними даними

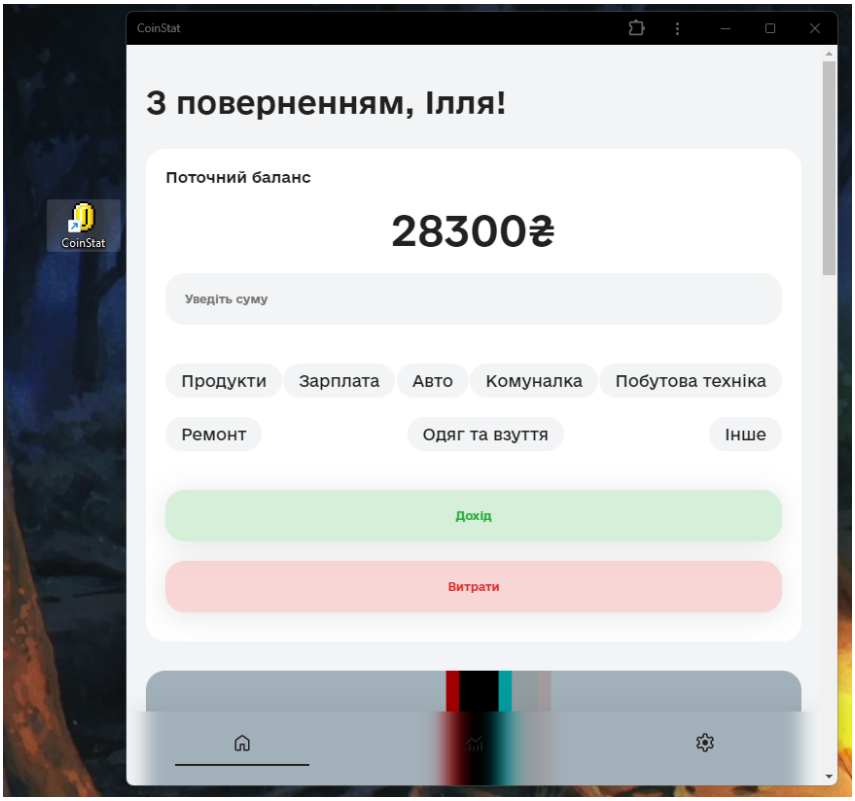
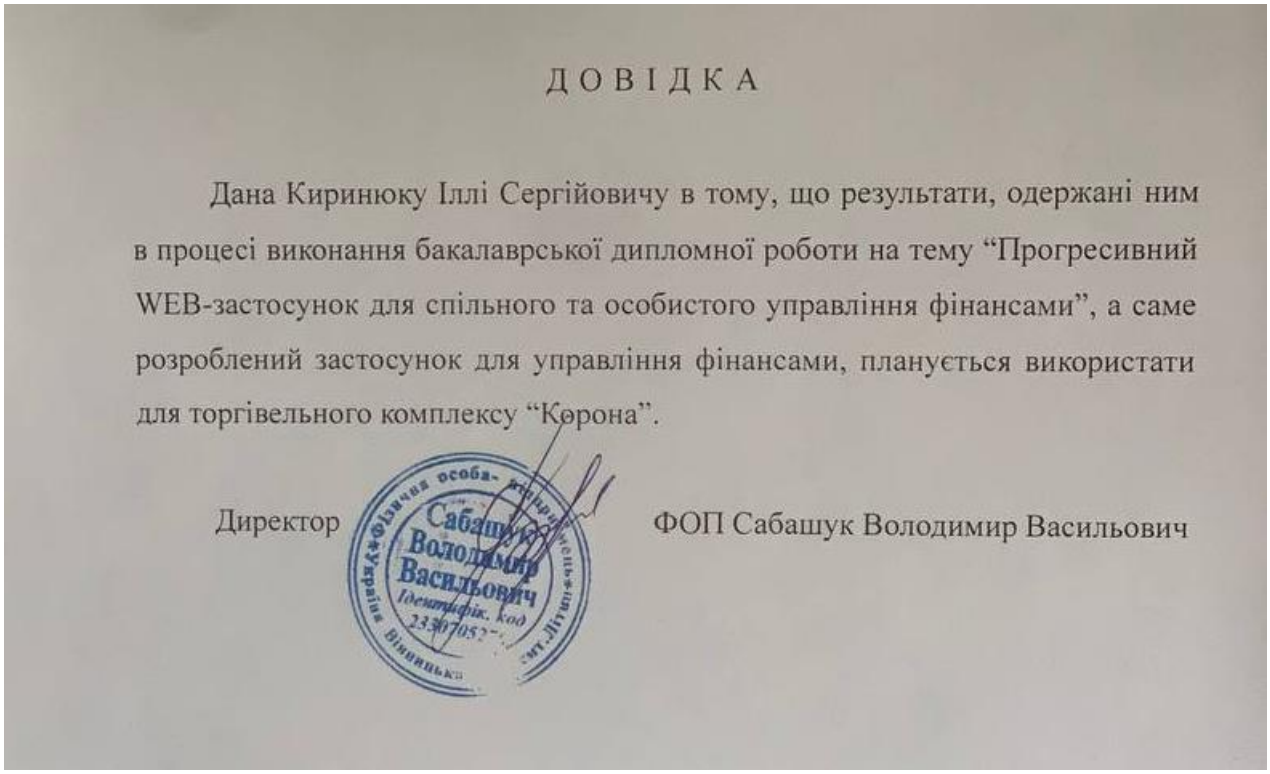


Рисунок Г.12 - Вигляд встановленого веб-ресурсу на комп’ютері, де видно іконку та вікно застосунку

ДОДАТОК Д
ДОВІДКА ПРО ВПРОВАДЖЕННЯ



ДОВІДКА

Дана Киринюку Іллі Сергійовичу в тому, що результати, одержані ним в процесі виконання бакалаврської дипломної роботи на тему “Прогресивний WEB-застосунок для спільного та особистого управління фінансами”, а саме розроблений застосунок для управління фінансами, планується використати для торгівельного комплексу “Корона”.

Директор ФОП Сабашук Володимир Васильович