МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЦИРУЛЬНИК МАКСИМ СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 2025 р.

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ФІНАНСОВОГО СЕРВІСУ НА ОСНОВІ АРХІТЕКТУРИ FULLSTACK

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Потапова Н. А., доцент кафедри
інформаційних технологій,
к. е. н., доцент
_____

Оцінка: _____/_____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Цирульник М. С. Розробка мобільного додатку фінансового сервісу на основі архітектури Fullstack. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено питання, пов'язані з розробкою мобільного додатку фінансового сервісу на основі архітектури FullStack. В рамках роботи було створено інтерфейс додатку у хмарному редакторі Moqups, розроблено діаграму класів, реалізовано консольну та графічну версії з використанням об'єктно-орієнтованого підходу.

Ключові слова: мобільний додаток, архітектура, об'єктно-орієнтований підхід, інтерфейс, тестування.

69 с., 32 рис., 40 джерел.

ABSTRACT

Tsyrulnyk M. S. Development of a Mobile Finansial Servise Application based on the Fullstack Architecture. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2025.

In the qualification (bachelor's) work, issues related to the development of a mobile application of a financial service based on the FullStack architecture were investigated. As part of the work, the application interface was created in the cloud editor Moqups, a class diagram was developed, and a console and graphical version was implemented using an object-oriented approach.

Keywords: mobile application, architecture, object-oriented approach, interface, testing.

69 p., 32 figures, 40 sources.

ЗМІСТ

ВСТУП

У сучасному світі, де панує цифровізація, фінансові послуги стають дедалі доступнішими завдяки мобільним додаткам [17]. Банківські операції, інвестиції, бухгалтерський облік та особисте управління фінансами все більше переходять у цифровий формат [4], що створює великий попит на ефективні, безпечні та зручні у використанні рішення. Однією з ключових технологій, що забезпечує функціонування таких систем, є архітектура FullStack, яка інтегрує Frontend (клієнтську сторону) та Backend (серверну логіку) в єдиний, робочий продукт.

Стрімкий розвиток фінансових технологій (FinTech), що супроводжується зростаючою конкуренцією у секторі фінансових послуг [17], вимагає інтеграції новаторських рішень. Мобільні додатки, такі як Monobank, Revolut та PayPal, демонструють ключову важливість зручності використання [22], швидкості транзакцій та безпеки. Використання архітектури FullStack дає змогу розробникам створювати системи, які є масштабованими, адаптивними та ефективними, тим самим відповідаючи вимогам користувачів, що постійно змінюються.

Основною *метою цієї бакалаврської роботи* є розробка мобільного додатку, що функціонує як фінансова послуга, використовуючи методологію FullStack. Додаток повинен запропонувати користувачам зручний спосіб управління фінансами, а також продемонструвати можливості сучасних Web- та мобільних технологій у фінансовому секторі.

Об'єкт дослідження – процес розробки мобільного додатку фінансового сервісу з використанням технологій повного циклу розробки, здатний надавати користувачу такі послуги як: ведення обліку доходів і витрат, аналіз фінансової статистики, безпечні транзакції, зручний інтерфейс для користувачів.

Предметом дослідження є методи, моделі та технології розробки процесів, які реалізує функціонал мобільного додатку фінансового сервісу.

В межах поставленої мети були сформульовані *основні завдання*:

1. Провести аналіз та виявити основні тенденції розвитку ринку фінансових додатків та існуючих FullStack-рішень.

2. Обґрунтувати вибір стеку технологій (frontend, backend, база даних).

3. Спроєктувати архітектуру додатку та UX/UI-дизайну.

4. Виконати програмну реалізацію клієнтської та серверної частини мобільного додатку.

5. Провести тестування продукту на працездатність та безпеку.

Структура роботи складається з вступу, трьох розділів, списку джерел та додатків. В першому розділі досліджено теоретичні основи проєктування мобільних додатків фінансового сервісу. В другому розділі акцентовано увагу на виборі технології розробки та ідентифікація процесів функціонування мобільного додатку. В третьому розділі наведено результати програмної реалізації клієнтської та серверної частини мобільного додатку.

Практична цінність даної розробки полягає в тому, що створено готове рішення з засобами протоколів безпеки даних., яке може бути задіяним у процеси інтеграції у наявну банківську інфраструктуру.

Апробацію результатів даної роботи проведено на VI Всеукраїнській науково-практичній конференції «Прикладні інформаційні технології» (м. Вінниця, 22 травня 2025 року) в доповіді «Концепція розробки мобільного додатку фінансового сервісу», на конференції Proceedings ITTAP’2023: 3rd International Workshop on Information Technologies: Theoretical and Applied Problems (м. Тернопіль, 22–24 листопада 2023 року), на конференції ITTAP’2022: 2nd International Workshop on Information Technologies: Theoretical and Applied Problems (м. Тернопіль, 22–24 листопада 2022 року).

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ ДОДАТКІВ ФІНАНСОВОГО СЕРВІСУ

1.1 Вплив цифровізації на створення додатків фінансового сервісу

Внаслідок переходу на цифрові технології в усіх сферах діяльності компаній сфера фінансових технологій (FinTech) набуває ключового значення в удосконаленні фінансового сектору [17]. Термін «фінансові технології» зараз охоплює будь-які інновації, які впливають на спосіб ведення бізнесу [17]: від розробки цифрових валют до вдосконалення процесу бухгалтерського обліку. Після настання інтернет-революції та розвитку мобільного інтернету і смартфонів фінансові технології отримали стрімкий розвиток. Поняття Fintech, яке в першу чергу асоціювалося з комп'ютерними рішеннями, розробленими для внутрішніх операцій банків і компаній, сьогодні охоплює широкий спектр технологічних рішень у фінансовому секторі - від приватних до корпоративних і державних фінансів. Сучасний ринок Fintech можна розглядати як систему, що об'єднує різних учасників фінансового ринку, включаючи фінансові технологічні стартапи, регуляторів, банки, міжнародні фінансові організації, а також компанії, що займаються розробкою і впровадженням фінансових технологій [17].

З розвитком новітніх комп'ютерних та інформаційних технологій обробка великих обсягів даних, прогнозування та моделювання пов'язані з великою кількістю факторів. Це пов'язано з поширенням програмних продуктів у фінансовій галузі. Сучасні рішення, такі як бізнес-аналітика, штучний інтелект, хмарні обчислення та управління великими даними, використовуються фінансовими компаніями для вдосконалення управління та отримання стратегічних переваг на ринку. Сектор фінансових технологій за останні роки набув значного росту, а прогнозований обсяг інвестицій у FinTech до 2028 року планується збільшити на рівні $335,2 млрд [17]. Це характеризується показником

середнього росту на рівні 19-25%».

Після 2008 року, перенісши світову фінансову кризу, банківська система почала прилаштовуватись до нових умов господарювання. Регулятори фінансової діяльності посилили вимоги до капіталізації банків, розробили стандарти управління ризиками та здійснили посилення вимог до процедур KYC (визнання клієнта) та AML (боротьба з відмиванням грошей). В цей самий час набули стрімкого розвитку інформаційні технології.

Інтернет став відомим майже скрізь. Поширення смартфонів і мобільних додатків відбулося навіть у тих країнах, які не мають розвинутої економіки. Стрімко розвивалися соціальні мережі. Створюють інноваційні продукти та послуги з новими стандартами якості, швидкості обслуговування споживачів такі компанії як-от: Amazon, Apple, Facebook, Google, Microsoft,

Україна продемонструвала позитивні зрушення, піднявшись з 47 на 45 місце серед 131 країни в рейтингу Global Innovation Index. Особливо помітним є зростання у категоріях «освіта» - на 20 позицій та «високотехнологічний розвиток R&D» - на 10 позицій. Ці позитивні зміни свідчать про великий потенціал України та її готовність до впровадження інноваційних технологій. За останні п’ять років український фінансовий сектор досяг значного прогресу в розвитку фінансових технологій.

Ці фактори зумовили появу на українському фінансовому ринку нових учасників – організацій, які спроможні виконувати традиційні банківські функції, пов’язані з виконанням платежів та грошових переказів. Ці організації, відомі як фінтех-компанії, застосовують інноваційні технології високого рівня та створюють споживачам привабливі умови при обслуговуванні інструментів фінансового сервісу, пропонуючи більш низькі ціни. Крім того, вони розробляють нові платіжні інструменти та технічні рішення. Наприклад, такі фінтех-компанії, як "iPay.ua", "Portmone.com", "GlobalMoney", "City24", "Укркарт" та інші спеціалізуються на грошових переказах; інші компанії, такі як

Moneyveo, Global Credit, CreditKasa та інші, надають онлайн-позики. Серед них також є мікрофінансові організації та сервіси P2P і P2B кредитування. Низка фінтех-компаній (Exmo, Kuna, BTC Trade UA, Tyme, IBox, Bitcoin, Ethereum, Litecoin) пропонують послуги з торгівлі валютою та криптовалютами. Ці фінтех-компанії використовують передові технології для надання споживачам фінансових продуктів за нижчими цінами, а також для створення нових платіжних інструментів і технічних рішень. Збільшення кількості власників смартфонів сприяє активізації процесів в інтернеті, що призводить до зростання популярності безготівкових платежів, електронних гаманців та інших онлайн-платежів [4]. За даними Ukrainian FinTech Directory 2020 [19], більшість фінтех-компаній працюють за рахунок власних коштів засновників. Додатковим фактором, що впливає на розвиток фінтех-ринку, є ініціативи державних органів зокрема банку України та Міністерства цифрової трансформації. Протягом 2021-2022 років вони активно розробляли проєкти, які мали вплив на фінтех-індустрію. Стали важливими для українського фінтех-ринку заходи:

– створення Open Banking API HUB в Україні;

– перехід необанків від пілотних проєктів до операційного режиму;

– випуск NFT (не взаємозамінних токенів) українських художників;

– стипендії у криптовалюті та проект з виплатою зарплати у цифровій гривні.

Державні органи, які проводять регулювання фінансового ринку, формують законодавчу базу збалансованого розвитку фінтех-компаній України. Національний банк України розробив та затвердив Стратегію розвитку фінтеху до 2025 року, яка направлення на створення та функціонування повноцінної фінтех-екосистеми з інноваційними фінансовими послугами [4].

Українська асоціація фінтех та інноваційних компаній [19] презентувала у 2024 році щорічне дослідження (каталог фінтех-компаній 2024), за даними якого fintech-ринок України нараховує 256 компаній, а загальний обєм ринку

становить $1,2 млрд (рис. 1.1).

Рисунок 1.1 –Український фінтех в цифрах [19].

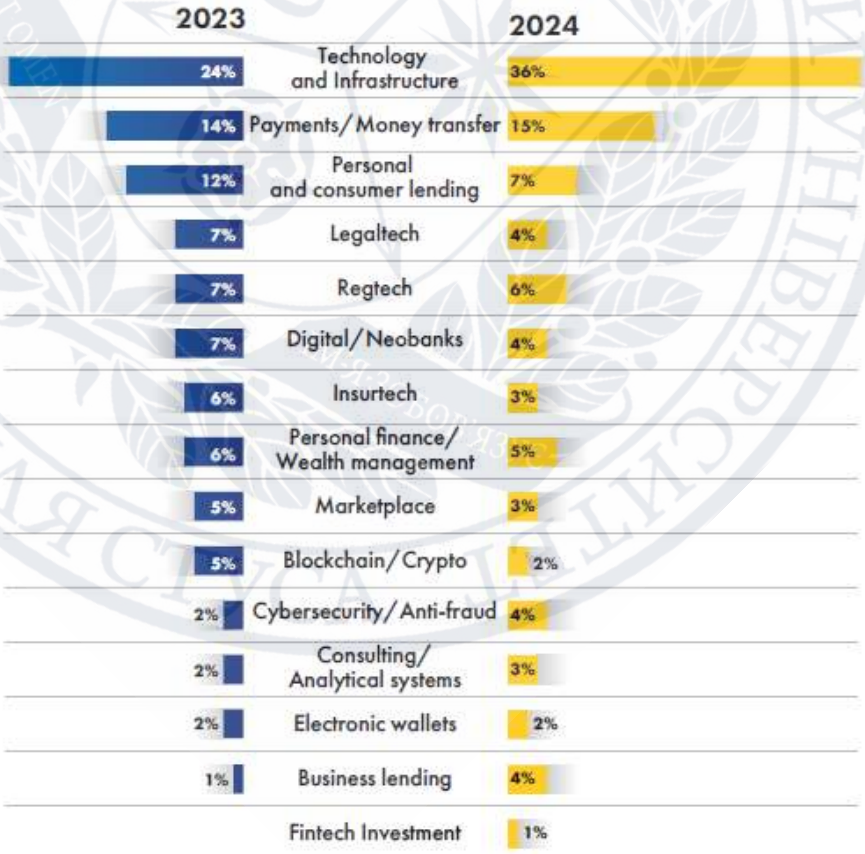


Рисунок 1.2 – Розподіл за сферами діяльності [19].

Найбільше зростання спостерігається у сфері технологічної інфраструктури (рис. 1.2) за даними [4]. У 2024 року її частка становить 36% від опитаних компаній, що на 12% більше, ніж 2023 року, коли вона складала 24%. Ця тенденція підтверджує продовження попиту на IT-рішення для фінансових установ, зокрема у напрямку їх цифрової трансформації. Платіжні сервіси і перекази 2024 року зберегли другу позицію з часткою 15 %, що лише на 1% більше, ніж 2023 року (14%). Незначне зростання свідчить про стійкість платіжного ринку України, попри виклики, пов’язані з війною.

Рисунок 1.3 – Головні технології серед український фінтех-компаній [19].

Найбільш розповсюдженою технологією серед фінтех-компаній вже кілька років поспіль залишається API. Значно зросло використання хмарних сервісів у 2024 року до 44%. Таке стрімке зростання не безпідставне, одним з ключових аспектів хмарних сервісів є здатність забезпечувати високу масштабованість і

гнучкість, дозволяючи компаніям швидко адаптуватися до змінних умов ринку і потреб споживачів. Хмарні рішення сприяють оптимізації бізнес-процесів, шляхом автоматизації й інтеграції різноманітних систем і програм. Крім цього, вони відіграють значну роль у стратегіях кібербезпеки, надаючи можливості для безпечного зберігання й обробки даних, а також їх відновлення у випадку непередбачуваних подій, що є актуальним як ніколи.

Набирає обертів щодо використання технологія блокчейн, яка може «мати такий же великий вплив (на суспільство), як і інтернет» [4].

Майже одноголосно 94% опитаних компаній [19] вважають найбільш перспективною технологією штучний інтелект. Також, серед перспективних технологій для українського ринку є технологію API (62%), хмарні сервіси (53%), роботизовану автоматизацію процесів RPA (robotic process automation) (40%) та блок-чейн (38%).

Рисунок 1.4 – Найбільш перспективні напрямки розвитку фінансово-технологічного сектору в Україні на найближчі 2 роки

Найактуальніші тренди, появу яких можна очікувати у майбутньому, опубліковані у звіті Української асоціації фінтеху та інноваційних компаній у

2025 р. [19] (рис. 1.4). За словами екпертів, найбільшою перспективою розвитку фінансово-технологічного сектору в Україні в найближчі два роки є напрями штучного інтелекту, відкритий банкінг та кібербезпека. У п’ятірку трендів входять військові технології та діджитал банкінг, який замінив цифрове кредитування у порівнянні з попереднім роком.

1.2 Особливості конкурентного ринку мобільних додатків фінансового сервісу

Мобільні додатки для управління персональними фінансами — це сучасні інструменти, створені для упорядкування, моніторингу та ефективного контролю грошових потоків [22]. Вони стали необхідністю для всіх, хто прагне оптимізувати власний бюджет, знайти баланс між витратами та заощадженнями і впевнено прямувати до своїх фінансових цілей.

У зв’язку з стрімким прогресом цифрових технологій такі додатки відкривають нові горизонти у спрощенні фінансових операцій. Їх зручність полягає у постійному доступі через смартфон, що дозволяє користувачам тримати фінансові питання під контролем у будь-який момент і в будь-якому місці. Просунуті системи безпеки на базі шифрування даних та багатофакторної автентифікації гарантують надійний захист конфіденційної інформації. Регулярне застосування мобільних фінансових додатків допомагає сформувати відповідальний підхід до управління коштами, що, у свою чергу, сприяє досягненню фінансової стабільності.

Існує безліч мобільних додатків, що допомагають планувати особист іфінанси, кожен з яких може похвалитися власним набором функцій [22]: від базового обліку витрат до складного фінансового планування і навіть інвестування. Вибір залежить виключно від особистих потреб та фінансових цілей, які ви ставите перед собою. Розглянемо найпопулярніші мобільні фінансові додатки на українському ринку [22].

Money Manager – це мобільний додаток для управління фінансовими витратами. У даному додатку відображаються основні вкладення користувача відсортовані за часовими періодами, а в документах транзакцій відображаються рахунок, тип, сума, категорія та інша інформація. Також є візуалізація у вигляді кругових діаграм і графіків, які призначені для моніторингу бюджету. Існує функція управління картами та планування бюджету за окремими категоріями. Є можливість встановити бюджет по категоріях. Зручність програми передбачає те, що інформацію можна переглядати через комп'ютер [22].

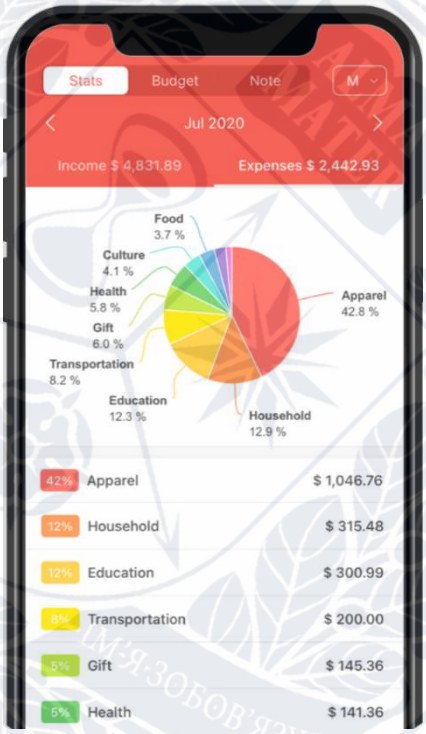


Рисунок 1.5 –Вікно мобільного додатку Money Manager

Monefy – додаток призначений для легкого й швидкого відстежування витрат та доходів. Додаток має простий інтерфейс та зручний дизайн, який орієнтований як на початківців, так і на досвідчених користувачів. Однією з основних переваг Monefy є наявність діаграм, які призначені для візуалізації витрат. Це дозволяє користувачам відслідковувати, куди витрачаються їх гроші

(рис. 1.6). Monefy синхронізовує дані через Dropbox, що дає змогу отримати доступ до власних фінансових даних з різних пристроїв. Додаток підтримує кілька мов та валют і дозволяє сформувати статистику планового бюджету, а не усіх доходів [22].

Рисунок 1.6 – Вікна мобільного додатку Monefy

Сервіс Wallet [22] налаштований на автоматичне оновлення щоденних витрат. Це досягається синхронізацією додатка з банком, в якому знаходиться основний рахунок, а також існує можливість переглядати щотижневі звіти про витрати, керувати боргами і відслідковувати рахунки; формувати фінансові транзакції по різних категоріях. Такими категоріями можуть бути: витрати на харчування, транспорт, дозвілля тощо. Такий підхід допомагає проводити повний контроль над фінансами і розуміти, куди витрачаються гроші.

Сервіс Wallet формує детальні звіти з графіками для аналізу фінансової звітності операцій (рис. 1.7). Це допомагає обґрунтовує процес прийняття фінансових рішень. Додаток також підтримує можливість спільного управління

фінансами, що дозволяє ділитися бюджетом і транзакціями з родиною або друзями.

Рисунок 1.7 – Вікна мобільного додатку сервісу Wallet

Багатофункціональний мобільний додаток Money Lover [22] для управління особистими фінансами допомагає користувачам планувати бюджет, контролювати свої витрати та доходи.

Додаток Money Lover дозволяє додавати фінансові операції за категоріями: розваги, транспорт, харчування, рахунки. Додаток дозволяє планувати бюджети на різні категорії витрат й відстежувати їхнє виконання. Можна побачити, наскільки близько до перевищення бюджету в кожній категорії.

Додаток показує прогрес й пропонує функцію нагадувань про оплату рахунків для уникнення пропущених платежів, штрафів; генерує детальні звіти про доходи та витрати.

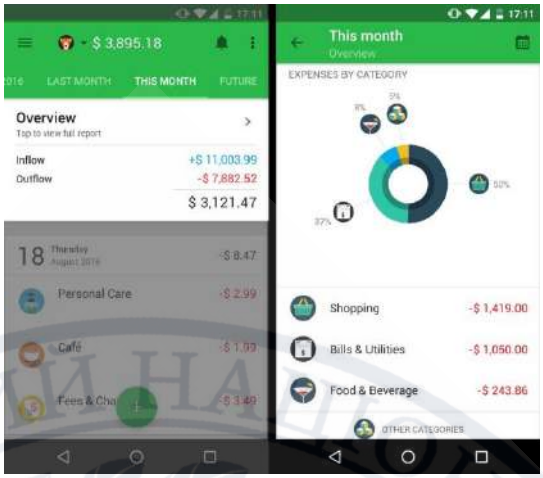


Рисунок 1.8 – Вікна мобільного додатку Money Lover

У додатку Mobills [22] всі витрати розділені на категорії, а основну інформацію за певний період можна переглядати у вигляді діаграми або графіка.

Додаток дозволяє встановлювати ліміти витрат за кожною статтею бюджету й періодично переглядати ліміт. Mobills дозволяє встановити фінансову мету з необхідною сумою і контролювати терміни її досягнення. Додаток може надсилати сповіщення про терміни платежів та синхронізується з хмарним сховищем й експортує інформацію про витрати й доходи в Excel, PDF.

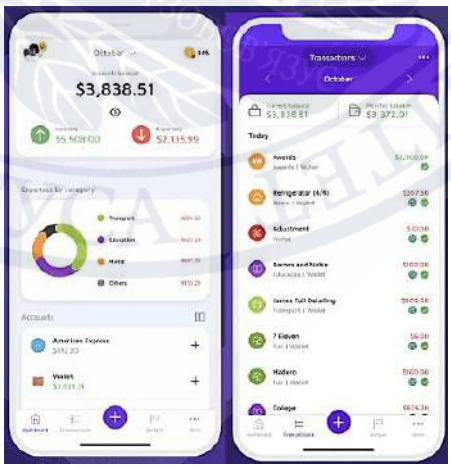


Рисунок 1.9 – Вікна мобільного додатку Mobills

1.3 Концептуальні підходи в проектуванні додатків фінансового сервісу

Створення додатків FinTech є поєднанням передового досвіду процесів розробки програмного забезпечення, методологій і передових технологій для створення фінансових додатків, адаптованих як для мобільних, так і для веб-платформ [20]. Процес об'єднує відносини взаємодії з користувачем і фінансову експертизу для створення програм, орієнтованих на фінансові потреби користувачів та інтуїтивно зрозумілих і практичних [20]. Етапами розробки фінансової програми є:

– визначення мети додатку, що охоплює встановлення мети програми або конкретизацію проблем, які розв'язує ця програма;

– створення дизайну інтерфейсу користувача, що включає проектування інтерфейсу користувача програми, визначення і наповнення його функціями;

– розробка бекенд-інфраструктури, яка передбачає побудову бекенд-інфраструктури програми або формулювання та впровадження бізнес-логіки, яка має відрізняти її від існуючих програм FinTech;

– проведення тестування та перевірки, в результаті чого проводиться ретельна оцінка усіх технічних складових програми та перевірка її функціональності шляхом реального тестування користувачами з метою задоволення їх потреб.

При розробці додатків FinTech орієнтуються на використання передових та інноваційних технологій, зокрема блокчейну, машинного навчання, штучного інтелекту, хмарних обчислень тощо. Таке по'єднання технологічних стеків направлене на забезпечення конфіденційності, безпеки, продуктивності та безперебійного виконання фінансових послуг і транзакцій.

Низка ключових функцій робить додаток FinTech життєздатним і привабливим рішенням для користувачів [20]. Такі функції спільно підвищують безпеку, функціональність і зручність програми, сприяючи широкому визнанню.

Критичними функціями, які можливо інтегрувати у програму FinTech, охоплюють:

1) Надійні заходи безпеки. Безпека програми має першочергове значення орієнтуючись на суворі правила, які регулюють світовий фінансовий сектор. В умовах керування конфіденційною фінансовою інформацією, застосування суворих заходів безпеки є необхідним. Для запровадження вдосконалених протоколів безпеки, використовують такі технології, як блокчейн, біометрія, шифрування, двофакторна автентифікація.

2) Безпроблемна інтеграція платіжного шлюзу. Ключовим елементом в побудові додатків FinTech є онлайн-платежі та безперебійна інтеграція платежів. Включивши платіжні функції у додаток, користувачі отримають вигоду при керуванні своїми заощадженнями та витратами.

3) Використання машинного навчання. Використання ШІ та методик машинного навчання є ключовим для інноваційних додатків FinTech.

4) Інтуїтивно зрозумілі інформаційні панелі. Важливим елементом є спрощення фінансового управління за допомогою використання додатку. Основна ідея полягає в тому, що необхідно надати користувачам інтуїтивно зрозумілу інформаційну панель. Така панель має візуально відображати витрати користувачів, історію платежів або показники аналітики акцій. Такі візуальні елементи дають змогу користувачам зрозуміти власні фінансові звички, сприяючи прийняттю обґрунтованих рішень.

5) Голосова інтеграція. Користувачів все більше приваблюють голосові сервіси, що є наслідком поширення використання голосових помічників (таких як Alexa та чат-ботів) Увімкнувши у програмі сервіс голосової взаємодії, користувачі можуть працювати з її функціями без фізичної доступності до програми.

1.4 Сутність безпеки персональних даних користувачів мобільного додатку

Одним з найскладніших завдань для банків та інших постачальників фінансових послуг є ідентифікація та перевірка особи користувачів [20]. Біометрія, яка включає розпізнавання голосу, відбитків пальців або обличчя, змінює спосіб надання фінансових послуг. Ця тенденція не тільки відповідає вимогам перевірки KYC, але й вирішує нагальні проблеми споживачів. Складність введення паролів і кодів утримує велику кількість клієнтів від використання нових цифрових фінансових рішень. Впровадження біометричного єдиного платіжного інтерфейсу [20] призводить до збільшення обсягу цифрових платіжних транзакцій.

Персональні дані (ПД) – це будь-яка інформація, яка прямо або опосередковано ідентифікує фізичну особу [20]. У контексті мобільних додатків до них належать:

- контактні дані (ім’я, email, телефон);
- фінансова інформація (платіжні дані, історія транзакцій);
- геолокація, IP-адреса, MAC-адреса пристрою;
- дані про активність (логіни, паролі, історія використання додатку).

Витоки персональних даних призводять до:

- шахрайства (крадіжка коштів, соціальна інженерія);
- порушення приватності (несанкціонований збір даних).

Основні загрози для персональних даних у мобільних додатках:

– незахищене зберігання інформації: застосування незашифрованих локальних сховищ (SharedPreferences, SQLite без шифрування);

– небезпечні API: вразливості back-end-у (SQL-ін’єкції, недостатня аутентифікація);

– перехоплення трафіку: відсутність HTTPS або слабкі алгоритми шифрування.

– надмірні дозволи: запит доступу до камери, мікрофона, геоданих без необхідності;

– Hardcoding конфіденційних даних: API-ключі, паролі в коді додатку;

– зловмисні атаки: встановлення шкідливих модифікованих версій додатку (трояни, кейлогери); підроблені екрани входу для викрадення облікових даних.

Для захисту персональних даних у мобільних додатках необхідно:

– мінімізувати даних: збирати лише необхідні персональні дані; видаляти дані після завершення їх використання;

– шифрувати дані при передаванні: використовувати протокол TLS 1.2/1.3, заборона використовувати протокол HTTP;

– шифрувати дані при зберіганні: AES-256 для локальних даних, хешування паролів (bcrypt, Argon2);

– контроль доступу: двофакторна автентифікація (2FA).

РОЗДІЛ 2
ТЕХНОЛОГІЇ РОЗРОБКИ ТА ІДЕНТИФІКАЦІЯ ПРОЦЕСІВ ФУНКЦІОНУВАННЯ МОБІЛЬНОГО ДОДАТКУ ФІНАНСОВОГО СЕРВІСУ

2.1 Технології програмування при розробці мобільного додатку

Існують багато технологій створення мобільних додатків та середовищ розробки: Android Studio (мова програмування Java, Kotlin), Framework Xamarin для Visual Studio (мова програмування C#), Flutter (мова програмування Dart). Технологія створення Web-додатків з технологією React.js також дозволяє створювати крос-платформлені додатки, тому далі буде розглянуто більш детально стек технологій: TypeScript, JavaScript, eslint, prettier, json, npm(node-package-module), React.js, CSS, DOM.

Структура проєкту – набір файлів та папок, які утворюють повноцінний проєкт та надають можливість писати новий код.

React.js – це JavaScript бібліотека, що використовується для створення інтерфейсів користувача (UI) [5].

Бібліотека – це набір готових інструментів (функції, методи тощо), які допомагають розв’язувати певні завдання.

React допомагає створювати та керувати тим, як користувачі бачать і взаємодіють із веб-сторінками або додатками. React це SPA.

SPA (Single Page Application) – це веб-додаток, який завантажує тільки одну сторінку і динамічно змінює її вміст без необхідності перезавантаження всієї сторінки.

SPA використовує AJAX (Asynchronous JavaScript And XML) – це підхід до побудови веб-додатків, в якому JavaScript відповідає за завантаження та оновлення лише тих даних, які прямо зараз потрібні користувачу.

Це дозволяє уникнути перезавантаження повної сторінки, якщо

користувачу потрібно оновити лише частину даних з інтерфейсу:

– Встановлення React.js.

npx – це нова команда NPM, яка дозволяє зручне створення нового проекту на базі певних технологій: *npx create-react-app my-app my-app. Це* команда термінала, яка використовується для створення нового проекту на React;

– "react": це бібліотека React, яка надає можливість створення і управління компонентами та інтерфейсом вашого додатка;

– "react-dom": wей пакет допомагає відображати React-компоненти в реальному DOM браузера;

– "react-scripts": we набір скриптів та налаштувань, які допомагають вам розробляти, тестувати та збирати ваш React-додаток. Він містить Webpack, Babel та інші інструменти;

– "web-vitals": цей пакет допомагає вимірювати різні важливі показники продуктивності додатка, такі як час завантаження, відгук користувача та інші;

– "@testing-library/jest-dom": цей пакет надає розширені матчери та функції для покращення тестування в бібліотеці Jest. Він дозволяє писати більш зрозумілі й зручні тести;

– "@testing-library/react": це одна з найпопулярніших бібліотек для тестування React-додатків. Вона надає функції для ефективного та простого тестування компонентів, включаючи взаємодію з DOM та симуляцію подій.

– "@testing-library/user-event": цей пакет розширює можливості тестування, дозволяючи симулювати реальні дії користувача, такі як клікання, введення тексту, фокус тощо.

Компонент– це невелика, автономна частина інтерфейсу користувача, яка визначається за допомогою JavaScript-функції та може містити розмітку (HTML-подібний код) і логіку для відображення та взаємодії з іншими частинами програми.

Компоненти це основна концепція в React. Це як будівельні блоки, з яких складається весь інтерфейс. Вони допомагають організовувати код, зробити його більш зрозумілим і перевикористовуваним.

Іменований експорт – це можливість експортувати та імпортувати багато елементів з одного модуля.

Використовується, коли є великий файл, в якому знаходяться багато незалежних невеликих компонентів

Експорт за замовчуванням – це можливість експортувати один основний елемент, який складається з інших елементів у файлі

У файлі може бути не більше одного дефолтного експорту, але може бути скільки завгодно іменованих експортів.

JSX (JavaScript XML) – це розширення синтаксису JavaScript, яке дозволяє включати HTML-подібний код безпосередньо в JavaScript-код, що спрощує роботу з React-елементами

Файли з форматом розширенням jsx мають переваги:

– допомагають одразу розпізнати, що файл містить код JSX;

– середовища розробки надають кращу підсвітку синтаксису;

– деякі інструменти вимагають мати JSX файли.

Синтаксичний схожий на HTML

JSX виглядає дуже схоже на HTML, що полегшує перехід для розробників зі знанням HTML.

```
return (
<div>
<h1>Title</h1>
<p>Content</h1>
</h1>
);
```

Хуки (hooks) – це спеціальні функції в бібліотеці React, які надають можливість використовувати різні функціональні можливості у компонентах,

пов’язаних з управлінням станом та іншими аспектами React.js.

Назва «хуки» походить від англійського слова "hook", яке означає "зацепка" або "крюк".

Це ім’я відображає призначення хуків – вони дозволяють "зацепитися" за внутрішній стан компонента та реагувати на зміни цього стану або виконувати інші дії.

React Router – це бібліотека для маршрутизації на стороні клієнта в односторінкових додатках (SPA) на React.js

Маршрутизація – це процес визначення та управління тим, як веб-додаток React відображає різні компоненти на сторінці в залежності від URL-адреси, яку користувач вказує в браузері

Основні особливості React Router:

– дозволяє виконувати маршрутизацію без перезавантаження сторінки. При клікання за посиланням IIPI_ змінюється, і відповідний компонент відображається без перезавантаження сторінки;

– підтримує вкладену маршрутизацію, що означає, що можна вкладати компоненти-маршрути в інші компоненти, створюючи складні структури сторінок;

– можна визначити різні маршрути для різних IP-адрес або параметризувати маршрути для передачі даних між компонентами;

– дозволяє встановлювати обмеження на доступ до деяких маршрутів, щоб захистити їх від несанкціонованого доступу.

Routes – це компонент, який використовується для визначення маршрутів у React-додатку. Можна вказати шляхи до різних сторінок та пов’язані з ними компоненти для відображення

TypeScript – це мова програмування, яка є розширенням JavaScript і надає можливість використовувати статичну типізацію та іншу нову функціональність [16].

Код, написаний на TypeScript, транслюється в JavaScript.

Транспіляція – це процес, у якому одна мова програмування перетворюється на іншу мову програмування.

Файли з кодом мови TypeScript мають розширення .ts

JavaScript – це:

– високорівнева мова програмування, яка надає зручні технології для спрощення написання складного коду;

– ООП мова, яка структурує код за допомогою об’єктів, які мають свої властивості, функції та модифікації;

– мова, яка спочатку працювала у браузері, додає інтерактивність на веб-сторінки, взаємодіє з HTML та CSS, відправляє та завантажує дані з сервера

– мова, яка згодом стала працювати на будь-якому пристрої, на який встановлена платформа Node.js, яка дозволяє будувати будь-які серверні, мобільні, ПК програми, скрипти;

– мова з підтримкою асинхронності, що дозволяє виконувати різний код паралельно без блокування інших частин коду;

– мова, яка має динамічну, слабку типізацію, але з мовою-доповнення Typescript, вона стає мовою суворої типізації;

– найпопулярніша мова з комерційного використання та має велику спільноту розробників, які розвивають цю мову;

– мова, яка має великий набір готових вбудованих функцій, а також великий набір технологій від інших розробників;

– мова, яка була створена для початківців.

Node.js – це програмна платформа, серверне середовище з відкритим кодом, яке працює на всіх актуальних ОС.

Node.js – працює на движку V8 та виконує код JavaScript поза веб-браузером.

CSS (Cascading Style Sheets, каскадна таблиця стилів) – це мова для опису стилів та їх значень до HTML тегів.

NPM (Node package manager) – це менеджер пакетів Nodejs, який дає можливість встановлювати потрібні технології

Пакет / Залежність (package, dependence) – це код певної технології, який завантажуємо в наш проект

Node modules – це спеціальна папка в Node.js-проектах, де зберігаються всі зовнішні бібліотеки та залежності, встановлені через npm (Node Package Manager)

Вміст папки node_modules:

– бібліотеки (наприклад, react, lodash, axios) – кожна має свою підпапку;

– транзитивні залежності – бібліотеки, від яких залежать інші бібліотеки;

– метадані (package.json, README.md, ліцензії тощо) для кожної бібліотеки.

Структура NPM проекта:

– package-lock.json – список всіх пакетів та зв’язків;

– node_modules – папка з вихідним кодом всіх пакетів;

– package.json – конфігурація NPM проекту.

Ініціалізація виконується командою

Перевірка працездатності NodeJS та NPM та дізнатися версію NPM та NodeJS можна командою:

```
npm -v nodejs -v node -v
```

Перевірка працездатності NPM та інтернет з’єднання виконується командою:

```
npm ping
```

Команда ініціалізації NPM проекту в поточній папці:

```
npm init
```

Структура проєкта наведена на рис. 2.1.

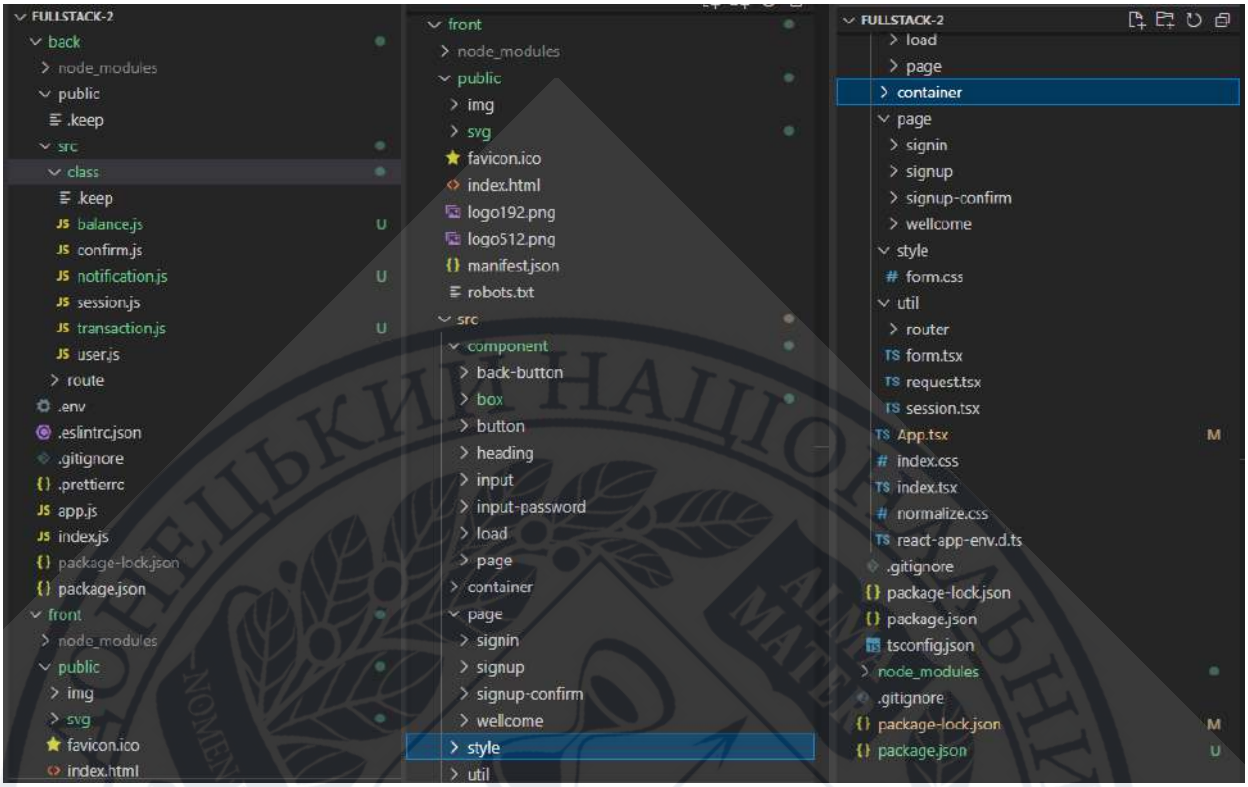


Рисунок 2.1 – Структура проєкту React-Bank

Node_modules - зберігає встановлені бібліотеки. Спочатку даний модуль не буде відображений у стуктурі проекту. Для того щоб данні модулі встановились потрібно в обох частинах (front-end та back-end) в консолі ввести команду (рис 2.2) npm install.

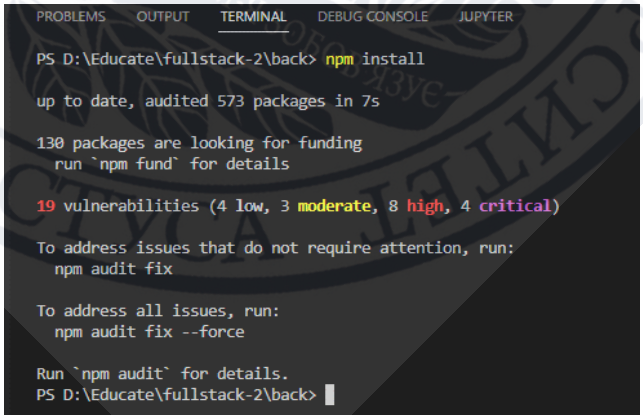


Рисунок 2.2 – Встановлення Node_modules у BackEnd та FrontEnd частинах за допомогою команди ‘npm install’

```
PS D:\Educate\fullstack-2\front> npm install

up to date, audited 1511 packages in 29s

245 packages are looking for funding
  run `npm fund` for details

30 vulnerabilities (3 low, 11 moderate, 15 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS D:\Educate\fullstack-2\front>
```

Рисунок 2.2 – Встановлення Node_modules у BackEnd та FrontEnd частинах за допомогою команди ‘npm install’(продовження)

Папка public - містить публічні файли (рис.2.3)

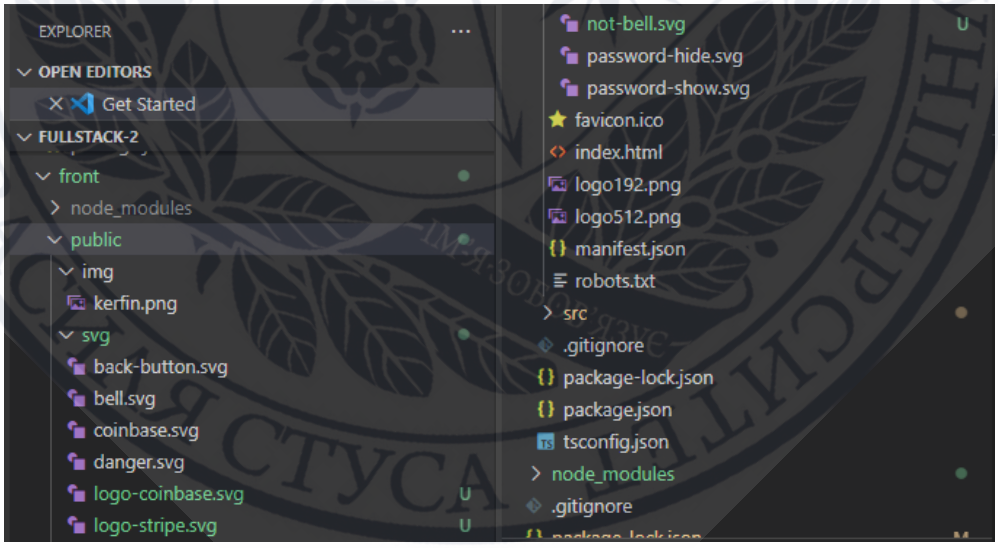


Рисунок 2.3 – Вміст файлів папки public

route – містить файли, які визначають шляхи, за якими знаходяться сторінки, та визначають які дані передаються на сторінки (рис.2.4).

Рисунок 2.4 – Вміст файлів папки route

Для запуску node.js сервера застосовуються файли з кодом app.js та index.js (рис. 2.4).

Деякі файли відносяться до системи контролю версій Git, GitHub.

Файл ‘.eslintrc.json’ (рис.2.4) відповідає за налаштування технології для підказок.

Файл .prettierrc (рис.2.4) відповідає за налаштування технології для форматування.

Файл .gitignore (рис.2.4) відповідає за налаштування файлів, які ігноруються GIT-системою.

Файли package.json та package-lock.json (рис.2.4) – файли від NPM

Взаємодія з сервером відбувається за допомогою ендпоїнтів. Ендпоїнт (end

points) – це конкретний URL, по якому сервер віддає HTML сторінку та/або будь-які інші дані з сервера. Структура найпростішого ендпоїнту наведена нижче:

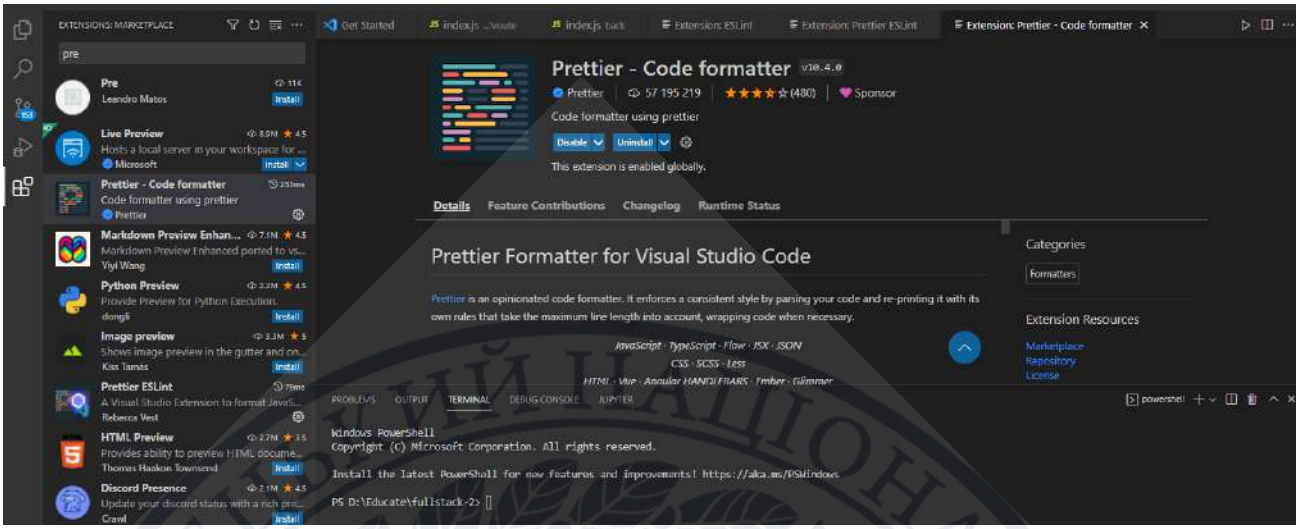
```
router.get("<PATH>", function (req, res, next) {
    res.render("<HTML FILE NAME>", <DATA OBJECT>);
});
```

де:

- <PATH> - шлях на сервері до HTML документа
- <HTML FILE NAME> - назва файлу HTML з папки views
- <DATA OBJECT> - об’єкт даних, які передаються з сервера

Також потрібно інсталювати технології ESLint, Prettier, Prettier ESLint та встановити значення “onFocusChange” для властивості “FileAutoSave” (рис. 2.5) для швидкого збереження відформатованого коду.

Рисунок 2.5 – Інсталювання необхідних технологій та параметрів.Частина 1.



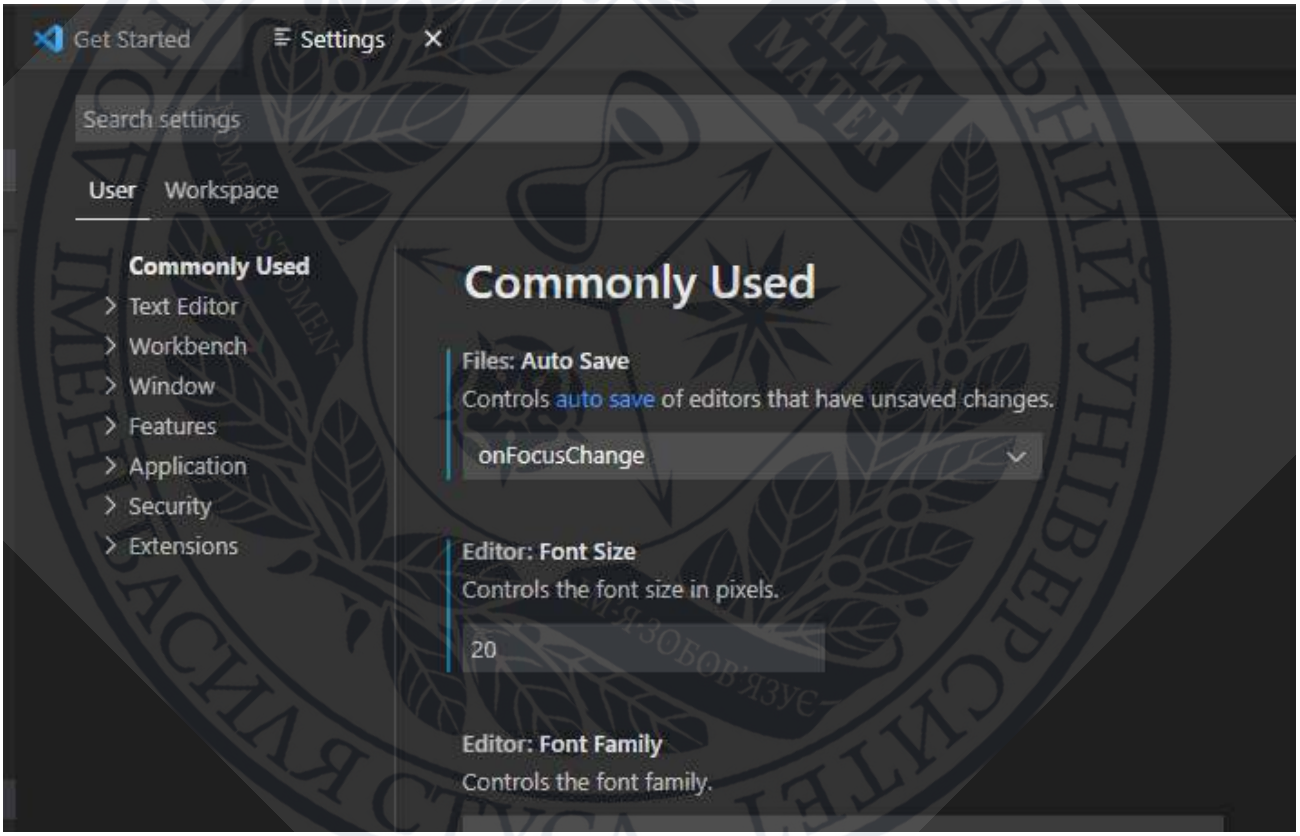


Рисунок 2.5 – Інсталювання необхідних технологій та параметрів. Частина 2.

CSS (Cascading Style Sheets, каскадна таблиця стилів) – це мова для опису стилів та їх значень до HTML тегів.

Властивість – це конкретний параметр, який надає певний ефект стилізації для HTML тегу, та має свій список значень

Каскадність – це принцип роботи CSS, який полягає в здатності перезаписувати та додавати нові властивості до тегів

Успадкування – це принцип роботи CSS, який полягає в здатності успадковувати CSS властивості від батьківських тегів

Є 4 варіанта як можна додати CSS код до певного тегу HTML:

– через окремий CSS файл;
– через тег <style>;
– через атрибут style;
– через JavaScript.

Селектор – це конструкція, яка збирає в одному місці всі потрібні властивості та вказує до яких тегів підключити їх

Структура правопису селектора наведено на рис. 2.6.

Рисунок 2.6 – Структура правопису селектора

Normalize.css – це файл CSS, який забезпечує кросбраузерність стилів та видаляє непотрібні нативні.

Також інколи такий CSS файл називають Reset.css.

У проєкті будуть використані такі типи селекторів:

1) Селектор класу (Class Selectors) – це селектор, який надає стилі для всіх тегів, які мають певний клас, який був вказаний у селекторі.Селектор класу

починається з крапки і за ним слідує ім’я. В HTML для встановлення класу потрібно вказати атрибут class. Приклад використання селектору класу:

```
.heading {
    display: flex;
    flex-direction: column;
    gap: 12px;
    justify-content: center;
    align-items: center;
    margin-bottom: 32px;
}
```

2) Селектор тегу – це селектор, який надає стилі для певного тегу, який використовується у будь-якому місці HTML документа. Якщо вказаний тег в HTML документі дублюється, то стилі будуть надані для всіх тегів, які відповідають селектору. Приклад використання селектору:

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI",
"Roboto", "Oxygen",
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans",
"Helvetica Neue",
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
```

3) Дочірній селектор – це селектор, який надає стилі для всіх тегів, які є дочірніми тегами головного тегу. Для цього слід вказати знак більше (">"), щоб показати відношення між батьківським та дочірнім тегом. Приклад використання дочірнього селектору:

```
.heading > div {
    font-weight: bold;
}
```

```
.heading > p {
    color: #939199;
}
```

4) Сестринський селектор – це селектор, який надає стилі для сестринського тегу, який йде одразу після основного тегу. Для цього слід вказати плюс ("+"), щоб показати відношення між основним тегом та сестринським тегом. Приклад використання сестринського селектору:

```
.wellc + p {
    color: white;
}
```

5) Селектор першої дитини – це селектор, який вибирає перший тег, якщо він є першою дитиною всередині батьківського тегу. Приклад використання селектору першої дитини наведено нижче:

```
.wellcome > div:first-child {
    padding-top: 100px;
    background-image: url(./background.png);
    background-repeat: no-repeat;
    background-position: center;
    width: 100%;
    height: 583px;
    border-radius: 24px;
}
```

6) Селектор псевдоклас – це ключове слово, додане до звичайного селектора, яке визначає його особливий стан. Селектор псевдокласу додається через двокрапку. Приклад використання селектору псевдокласу:

```
a:hover {
  outline: 0;
}
```

7) Селектор останньої дитини – це селектор, який вибирає останній тег, якщо він є останньою дитиною всередині батьківського тегу. Приклад використання селектору останньої дитини:

```
.wellcome > div:last-child {
    display: flex;
    flex-direction: column;
    gap: 12px;
    justify-content: flex-end;
    padding-bottom: 33px;
}
```

Link – це компонент, який використовується для створення навігаційних посилань у React-додатку. Цей компонент дозволяє змінювати URL, не перезавантажуючи сторінку, що дає можливість створювати односторінкові додатки (SPA).

```
<Link to="/recovery" className="link">
  Restore
</Link>
```

Властивості компоненту Link наведені на рис. 2.7.

Назва	Опис
to	Визначає URL-адресу, на яку потрібно перейти, або можна вказати значення Partial<Path>
reloadDocument (необов'язковий)	Властивість reloadDocument можна використовувати, щоб пропустити маршрутизацію на стороні клієнта та дозволити браузеру обробляти перехід у звичайному режимі (наче це було <a href>).
relative (необов'язковий)	• "route" — вказує, що шлях у посиланні буде відносно ієрархії маршрутів. • "path" — вказує, що шлях у посиланні буде відносно поточного шляху, на якому ви знаходитесь.
replace (необов'язковий)	Це булеве значення, яке вказує, чи потрібно замінити поточний запис в історії навігації новим, замість додавання нового запису.

Рисунок 2.7 – Властивості компоненту Link

Редактор коду – це програма для написання, зберігання, запуску та

тестування коду мови програмування

Сучасні редактори надають широкий асортимент інструментів, які полегшують і прискорюють процес розробки коду. Буде використано редактор коду Visual Studio (VS) Code.

2.2 Ідентифікація процесів реєстрації та авторизації користувачів

Аутентифікація (Authentication, auth) — це процес встановлення ідентичності користувача з метою переконатися, що особа, яка намагається отримати доступ, дійсно є тим, за кого вона себе видає, таїр/іає відповідні права на це. Аутентифікація включає такі операції:

- реєстрація акаунту;
- підтвердження реєстрації;
- вхід в акаунт;
- відновлення доступу до акаунту;
- створення ключа (токену) для підтвердження дій.

На рис. 2.8 зображено частину яка відповідає до аутентифікації.

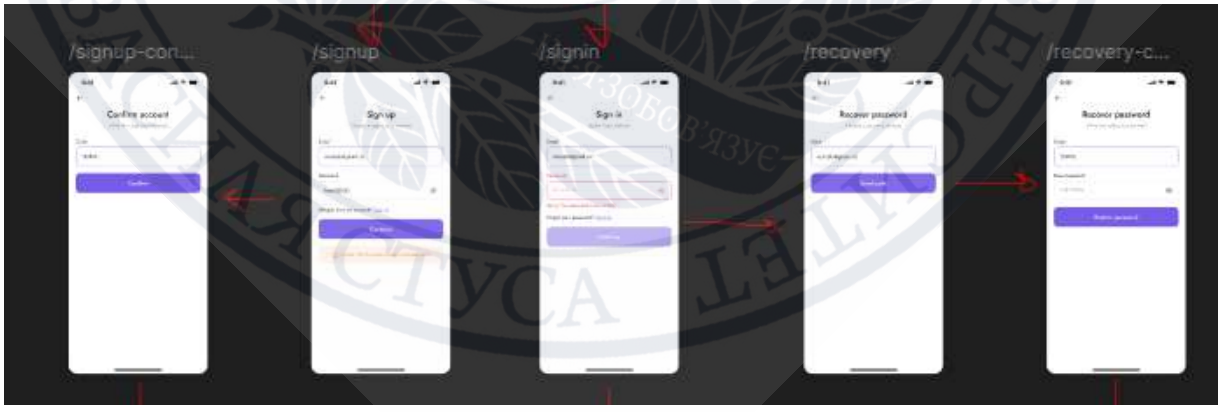


Рисунок 2.8 – Макет сторінок “signup-confirm”, “signup”, “signin”, “recovery”, “recovery-confirm”

Сторінка “SignUp” (рис. 2.9) містить кнопку “history back (←) ”, яка

повертає користувача на попередню сторінку, заголовок реєстрації, компонент form-item, який складається з самого поля та текстом помилки, кнопки “Continue”, alert-повідомлення, валідації (логіку відправки даних на сервер), кнопки у вигляді ока, для приховування/показу паролю, кнопку-посилання на сторінку “SignIn” у випадку, якщо користувач вже був раніше зареєстрований.

Регулярний вираз (regular expression) – це вбудований об’єкт-сутність, яка надає вбудовані функції для роботи з регулярними виразами в JavaScript.

Приклад використання регулярного виразу наведено нижче:

```
export const REGEXP_EMAIL = new RegExp(
  /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/g,
)
export const REGEXP_PASSWORD = new RegExp(
  /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-Z]).{8,}$/g,
)
    if (name === FORM_ALERT_NAME.PASSWORD) {
      if (!REGEXP_PASSWORD.test(String(value))) {
        return {
          [name]: FIELD_ERROR.PASSWORD,
          [FIELD_STATUS.DISABLED]: FIELD_STATUS.DISABLED,
        };
      }
    }
    if (name === FORM_ALERT_NAME.EMAIL) {
      if (!REGEXP_EMAIL.test(String(value))) {
        return {
          [name]: FIELD_ERROR.EMAIL,
          [FIELD_STATUS.DISABLED]: FIELD_STATUS.DISABLED,
        };
      }
    }
```

Валідація (Validation) – це процес перевірки даних на відповідність певним критеріям, правилам або стандартам. Вона використовується для забезпечення коректності, безпеки та цілісності інформації. Валідація застосовується для форми на вебсайтах та для перевірки email, пароля, номера телефону тощо.

Приклад валідації функції “validate”:

```
export const validate = (name: string, value: any) => {
    if (String(value).trim().length < 1) {
```

```
            return {
              [name]: FIELD_ERROR.IS_EMPTY,
              [FIELD_STATUS.DISABLED]: FIELD_STATUS.DISABLED,
            };
          }
        export const validateAll = (obj: { [key: string]: any }):
ALERT_NAME => {
          const obj1: { [key: string]: any } = {};
          for (const key in obj) {
            const value = validate(key, obj[key]);
            if (value[key]) {
              obj1[key] = value[key];
              obj1[FIELD_STATUS.DISABLED] = FIELD_STATUS.DISABLED;
            } else {
              obj1[key] = null;
            }
          }

      const alert = validate(fieldName, value);
        const handleSubmit = async (e: FormEvent<HTMLFormElement>)
=> {
          e.preventDefault();
          const val = validateAll(formData);
          setStateAlert(val);

          if (val.disabled === "") {
            dispatchRequest({
              type: REQUEST_ACTION_TYPE.PROGRESS as keyof typeof
REQUEST_ACTION_TYPE,
            });
            try {
              const res = await
fetch("http://localhost:4000/signin", {
                method: "POST",
                headers: {
                  "Content-Type": "application/json",
                },
                body: JSON.stringify(formData),
              });

              const data = await res.json();
              if (res.ok) {
                dispatchRequest({
```

```
                type: REQUEST_ACTION_TYPE.RESET as keyof typeof
REQUEST_ACTION_TYPE,
              });

              dispatch({
                type: ActionTypes.LOGIN,
                payload: {
                  token: data.session.token,
                  user: data.session.user,
                },
              });

              saveSession({
                token: data.session.token,
                user: data.session.user,
              });

              if (data.session.user.isConfirm) {
                navigate("/balance");
              } else {
                navigate("/signup-confirm");
              }
            } else {
              dispatchRequest({
                type: REQUEST_ACTION_TYPE.ERROR as keyof typeof
REQUEST_ACTION_TYPE,
                payload: data.message,
              });
            }
          } catch (error: any) {
            dispatchRequest({
              type: REQUEST_ACTION_TYPE.ERROR as keyof typeof
REQUEST_ACTION_TYPE,
              payload: error.message,
            });
          }
        }
      };
```

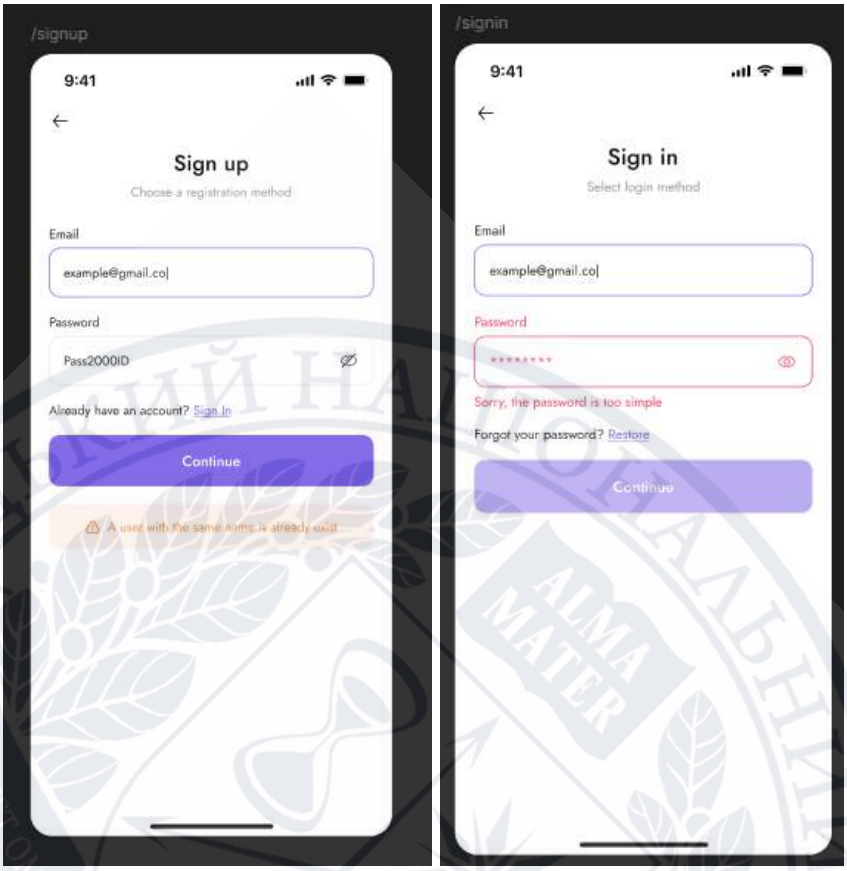


Рисунок 2.9 – Приклад роботи валідації

Рисунок 2.10 – Сторінка підтвердження реєстрації

Сторінка «відновлення паролю» (рис. 2.11) та «підтвердження відновлення паролю» (рис. 2.12)

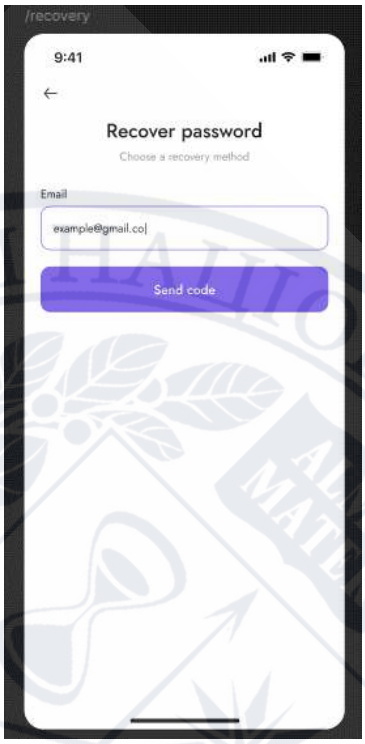


Рисунок 2.11 – Макет сторінки «відновлення паролю»

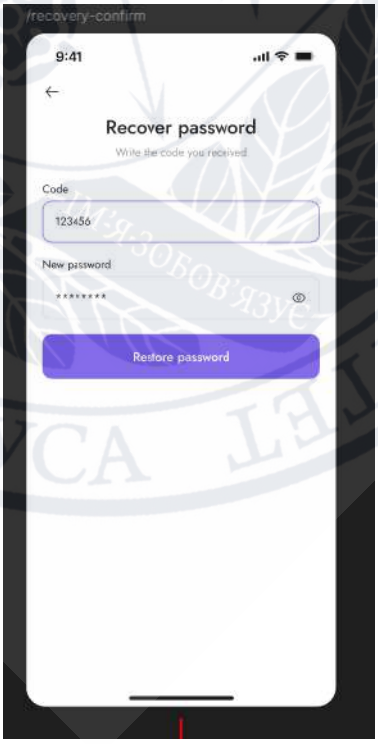


Рисунок 2.12 – Макет сторінки «Підтвердження відновлення паролю»

На рис. 2.12 показано макет сторінки «Підтвердження відновлення паролю».

2.3 Функціональні вимоги мобільного додатку фінансового сервісу

Мобільний додаток фінансового сервісу повинен мати:

– сторінку балансу з можливістю поповнення балансу,

– переказати кошти іншому користувачу,

– подивитись транзакції,

– налаштування акаунту, ноифікації,

– загальної аутентифікації (реєстрація, авторизація, підтвердження акаунту та відновлення)

– наявність «вітальної» сторінки, яку користувач бачитиме, коли захидить на головну (index) сторінку

– яскраву, контрастну графіку та іконки.

Дизайн сторінки “Wellcome”, яка завантажується при запуску додатку наведено на рис. 2.13.

Якщо користувач натискає на кнопку “SignIn” та коректно вводить свої дані, він потрапляє на сторінку Балансу.

Якщо немає токену, то користувач може подивитись дану сторінку за допомогою натискання кнопки “SignUp” або “SignIn”. Відповідно при натисканні кожної з цих кнопок будуть відкриватись окремі сторінки (“SignUp” або “SignIn”). На сторінках присутній компонент back-button (←), який повертає на попередню сторінку. Є компонет заголовку з описом. Є компонент вводу (input). Є компонент (input-password) з підтримкою кнопки «показу паролю». Також на сторінці реєстрації є посилання на сторінку входу. Кнопка “Continue” відправляє запит на реєстрацію. Є Alert-оголошення. Коли користувач зареєструвався відправляється запит на сервер на створення користувача. Користувач

створюється. Далі генерується код підтвердження, в якому знаходиться інформація про ідентифікатор користувача, для розуміння який це користувач. На сторінці «Підтвердження акаунту» вводиться код для підтвердження акаунту. При реєстрації генерується токен. Якщо зареєстрований користувач коректно вводить свої облікові дані, то далі потрапляє на сторінку Балансу.

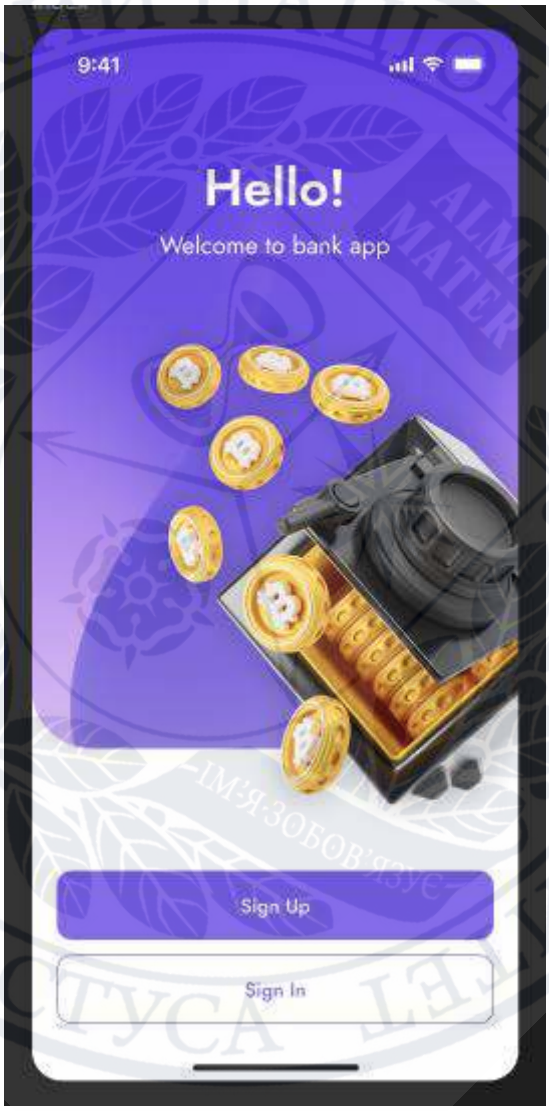


Рисунок 2.13 – Дизайн вітальної сторінки “Welcome”

Сторінка авторизації містить поля для вводу email та password. Далі після

натискання кнопки “Continue” відбувається вхід користувача в акаунт.

Є можливість відновлення акаунту. Відкривається відповідна сторінка (“Recovery”). Вводиться email. Далі після натискання кнопки “Send code” генерується код з інформацію про ідентифікатор користувача, який хоче відновити доступ до акаунту. Вводиться згенерований код, вводиться пароль та натискається кнопка “Restore password” («Відновлення паролю»).

Генерується токен. Оновлюється контекст аутентифікації та відбувається перехід на сторінку балансу. Є заголовок, який містить кнопку-посилання на сторінку налаштувань. На сторінці «Налаштування» є форма для зміни email, де вводиться email та старий пароль та зміна паролю (введення старого та нового паролю) та кнопка «виходу» з акаунту. В запиті передається токен який дозволяє ідентифікувати користувача та зрозуміти в якому користувачі потрібно змінити email та пароль.

На сторінці балансу присутня верстка шапки (з відображенням балансу, налаштування, нотифікації). На сторінці Нотифікацій відображається список нотифікацій (при певних «маніпуляціях» з акаунтом) (вхід, реєстрація, відновлення, оновлення, зміна даних, поповнення, вивід, переказ і т.д) створюється об’єкт нотифікації, який відображається у якості списку нотифікацій.

Є область для відображення поточного балансу. Є 2 кнопки («поповнення» та «відправка» грошей).

Поповнення відбувається наступним чином. Вводиться сума, далі натискається кнопка платіжної системи, далі відправляється запит на поповнення суми через платіжну систем. Користувач отримує поповнення на свій баланс.

При натиснення “Sent” відкривається сторінка відправки грошей, де вводяться данні пошти користувача, сума та відправляються гроші.

Створюються транзакції на «списання» (у користувача який надіслав гроші) та «поповнення» грошей (у користувача який отримав кошти).

При поповненні через платіжну систему з’являється транзакція поповнення через платіжну систему.

На сторінці Балансу є список транзакцій (в порядку від найновіших). При натисненні на будь-яку транзакцію відкривається сторінка з транзакціями з більш детальною інформацією.

Мобільна ОС (Mobile operating system) – це операційна система для смартфонів, планшетів або інших мобільних пристроїв.

Мобільний додаток (Mobile application) – є спеціально розробленим додаток, який працює в середовищі мобільної платформи (iOS, Android).

Мобільний сайт (Mobile website) – це сайт; адаптований для перегляду і функціонування на мобільному пристрої.

Інтерфейс користувача (User Interface) – це сукупність засобів, методів і правил взаємодії з будь-якою системою, яка керується людиною.

Макет (Mockup) – є графічним файлом, який складається з найдрібніших картинок-шарів елементів загального малюнка, і потрібен для повноти уявлення про реалізацію додатка.

Етапи розробки мобільних додатків:

1) Розробка технічної документації. Визначення основних завдань та потреб користувача має задовольняти додаток. Складається детальний опис функціоналу додатку.

2) Розробка інтерфейсу користувача. Як користувач буде користуватись додатком. Розробка прототипу додатку. Розробка схем (макетів) основних екранів.

3) Створення концепції дизайну. Створення декількох варіантів.

4) Створення макетів всіх екранів.

5) Розробка. На цьому етапі відбувається з’єднання клієнтської та серверної частин. Створення першої версії програми.

6) Тестування.

7) Налагодження. Виправлення певних помилок під час процесу тестування.

8) Регресійне тестування.

9) Створення іконки програми.

10) Запуск на ринку мобільних додатків. Включає наступні етапи:

– завантаження файлу програми;

– розміщення інформаційних матеріалів;

– розгляд заяви адміністрацією та затвердження її на ринку.

11) Тестування нових версій, регресійне тестування.

12) Встановлення .apk додатків на Android.

2.4. Обґрунтування підходів та методів тестування

Перед впровадженням додатка необхідно виконати ряд важливих умов:

1. Провести тестування на різних пристроях. Дана процедура спрямована на переконання коректності роботи додтаку на різних смартфонах. Для цього потрібно кілька пристроїв різних виробників, побажанням є наявність різних версій Android і розширення екрану. У разі, якщо додаток не протестований на 3-4 найпопулярніших пристроях, викладати його на Google Play є недоцільним. Для встановлення додатку на телефон з метою перевірки його функціональності, необхідно перейти в папку *app\build\outputs* та знайти файл app-debug.apk . Цей файл встановлюють на телефон через програму fileManager.

2. Перевірка підтримки різних розширень екрану. Проводиться перевірка того, як виглядає додаток на екрані, розмір якого відрізняється від розміру екрану емулятора за замовчуванням. Необхідним є створення кількох емуляторів з різними розмірами екрану і провести запуск додатку на кожному з них.

3. Локалізація. Локалізація спрямована на розширення аудиторії і

передбачає переклад програми на різні мови. Локалізацію проводять після створення програми, коли відбулось виправлення всіх помилок і більше не планується додавати нові рядкові ресурси.

4. Піктограма. Для її створення необхідно підготувати файл піктограми і помістіть його в каталог *res/drawable*. Подальшому змінюють атрибут *android: icon* елемента <*application*> у файлі маніфесту: <*application anroid: icon="@ drawable / my_icon" ...*

5. Посилання на магазин. Для перенаправлення користувача на GooglePlay (наприклад, для купівлі повноцінної версії програми), необхідно використовувати URI, наприклад:

```
market://search?q:імя_програм";
```

Існує основні чотири підходи в тестуванні мобільних додатків: на основі емуляції, в хмарі, на базі пристроїв і з використанням краудсорсингу.

Емулятори допомагають виявити більшість помилок на найбільш ранній стадії життєвого циклу розробки програмного забезпечення. Мобільні емулятори використовують для імітації поведінки смартфона/планшета у тому випадку, коли в наявності немає необхідних комплектуючих або потрібний пристрій зайнято іншим тестувальником. Спочатку вони розроблялися для тестування, тому є частиною SDK розробника. Запуск емуляторів на комп'ютерах або серверах, більш потужних пристроях, ніж смартфон є достатньо зручним способом у використанні. Проте, через такий підхід деякі помилки можуть бути не виявлені або викликати неправильне тлумачення, як-от час відгуку або продуктивність програми.

Важливим фактором є наявність низької ціни процедури. Тобто, тестувати на емуляторі дешевше в порівнянні з купівлею нового мобільного пристрою. Мінусами даного підходу є помилкові спрацьовування, обмежений набір жестів і багато іншого, що неможливо протестувати без мобільного пристрою. Емулятори не можуть охопити всі наявні проблеми, які можуть виникнути під

час безпосередньої взаємодії з користувачем. Тому, проведення реального тестування є важливою складовою.

Щоб досягти кращого результату тестування, використовують змішаний підхід – за допомогою емулятора та пристрою. Хмара пристроїв – це мобільне середовище, яке включає реальні пристрої Android та iOS з різними комбінаціями версій ОС, розмірами екрану, оперативної пам'яті та багато іншого. Хмарні пристрої дозволяють безпроблемно провести великомасштабне мобільне тестування.

Тестування в хмарі не залежить від місця розташування мобільних пристроїв і дозволяє будь-де до них підключатися. Пристрої підтримують паралельне тестування, підходять для швидкої розробки, записують результати і доступні 24/7. Для дотримання безпеки обирають приватну хмару замість загальнодоступної.

У реальній хмарі пристроїв тестувальник на кожному пристрої може протестувати призначений для користувача інтерфейс, виміряти продуктивність програми, спостерігати за тим як працює додаток з розрядженою батареєю, а також коли він знаходиться в автономному режимі і т. д.

Тестування мобільних додатків на базі пристроїв показує найкращі результати, оскільки програмне забезпечення відображує реальне використання кінцевого користувача. При тестуванні на реальних мобільних пристроях враховуються всі особливості ОС і якість обслуговування мережі (QoS).

Такий підхід до тестування є фінансово витратний, оскільки всі пристрої необхідно купувати. З урахуванням того, що ринок смартфонів зростає і оновлюється – це не є вигідним вкладенням. Тобто, фінансова сторона в цьому підході є мінусом.

Краудсорсінг є методом для перевірки функціональності програми на більш пізніх етапах розробки метою відтворення усіх варіантів використання. В процес тестування тимчасово залучаються фрілансери. Існують спеціальні

майданчики, на яких є користувачі, готові тестувати програмне забезпечення, а також ті, кому необхідно тестування [20]. Таким чином замовник може протестувати додаток саме для своєї цільової аудиторії.

Тестувальники можуть використовувати свої власні пристрої для тестування програми або отримати доступ до емуляторів пристроїв через краудсорсінговую платформу тестування, в залежності від вимоги замовника.

Такий підхід надає змогу отримати зворотний зв'язок щодо поліпшення UX і забезпечення зручності використання. До недоліків краудсорсингу відносяться проблеми, які виникають з вимогами конфіденційністю додатка, який тестується. Також проблемою є ненадійність тестувальників і виконання їх роботи.

Кожен підхід в тестуванні має свої переваги і недоліки мобільних додатків. Емулятори гарні для тестування користувальницького інтерфейсу і початкового контролю якості, реальні пристрої необхідні для тестування продуктивності, а хмарне тестування – хороший спосіб перевірити додаток на великій кількості пристроїв і операційних систем. Краудсорсінг дозволяє перевірити ПЗ в різних умовах, які наближаються до реальних.

Сервіси для бета-тестування для iOS [20]: TestFlight, TestFairy, HockeyApp, Crashlytics, для Android [20]: Alpha/Beta testing, TestFairy, HockeyApp, Crashlytics, для Windows: HockeyApp.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ ФІНАНСОВОГО СЕРВІСУ

3.1 Розробка інтерфейсу мобільного додатку

У відповідності до розділу 2 проєкт складається з окремих частини (клієктську FrontEnd, яка відповідає за візуальну складову та серверну BackEnd (відповідальну за функціональну складову)). Структура проєкту була наведена на рис. 2.1. Відповідно до принципів розробки та створеної архітектури для данної системи, в першу чергу було здійснено розгортання середовища розробки Visual StudioCode, та створено репозиторій на GitHub.

Visual Studio Code було завантажено з офіційного сайту Microsoft. Після чого, слідуючи інструкціям майстра встановлення, та завершення інсталяції, було здійснено початкове налаштування середовища.

Далі, зареєструвавшись на офіційному сайті GitHub, на панелі управління створюємо новий проект та налаштовуємо його у відповідності до раніше зазначених потреб. Після створення проекту, необхідно скопіювати URL репозиторію для його подальшого використання.

Після успішного встановлення середовищ розробки та налаштування GitHub, необхідно в терміналі здійснити ініціалізацію усіх необхідних пакетів згаданих у другому розділі, при побудові архітектури системи. Для цього, було використано вказаний на рисунку 3.1 перелік команд в терміналі.

Встановлення необхідних пакетів для розробки. Команда «npm install» є однією з найчастіше використовуваних команд npm і є ключовою для управління залежностями у проєктах. Вона використовується для встановлення пакетів (бібліотек або модулів) з npm (Node Package Manager) до проєкту.

Візуальний інтерфейс розроблявся у середовищі Figma Desctop.

3.2 Програмна реалізація функціональних блоків проєкту мобільного додатку фінансового сервісу

Спочатку необхідно створити порожній git-репозиторій та клонувати заготовку FullStack-проєкту і «перепідв'язати» її.

Далі необхідно створити сторінку Wellcome. Для цього у Front-End частині у папці "src/page" створити папку "Wellcome" в якій необхідно створити 2 файли – "index.tsx" для верстки сторінки за допомогою TypeScript та "index.css" для верстки стилів. Необхідно завантажити картинку «заднього фону» з Figma.

Створимо в папці Component компоненти "back-button", "balance-item", "button", "button-pay", "heading", "input", "input-password", "load" та "page" з файлами "index.css" та "index.tsx" у кожній з них.

Кнопка-компонент "back-button":

– містить імпорти: import React from "react" – імпорт React для роботи з JSX;

– import "./index.css" – імпорт CSS стилів для цього компонента.

Компонент BackButton:

– оголошується як функціональний компонент TypeScript (React.FC);

– не приймає жодних пропсів (пусті дужки ());

– повертає JSX розмітку;

Розмітка:

`<div>` з класом back-button:

– має обробник події onClick, який викликає window.history.back();

– вміщує <img> з іконкою кнопки «Назад»;

Обробник кліку:

`window.history.back()` – властивий метод браузера, який переводить користувача на попередню сторінку в історії переглядів.

Працює аналогічно до кнопки "Back" у браузері.

Елемент img:

Властивість `src="../../../svg/back-button.svg"` показує шлях до SVG іконки;

`alt="<"` – альтернативний текст (в даному випадку - символ "<");

`width="25" height="25"` – розміри іконки;

6. Стилі (CSS):

- компонент back-button містить властивість cursor: pointer яка змінює курсор на вказівник при наведенні;
- властивість transition: opacity 0.7s робить плавну зміна прозорості за 0.7 секунди;
- відступи: margin-top, margin-bottom, margin-left

`back-button:hover:`

зміна прозорості до 70% при наведенні;

Приклад коду з проєкту:

```
import { ReactComponent as BackIcon } from './back-button.svg';
```

Даний компонент є простим, але ефективним рішенням для навігації «повернення» в додатку. Може бути легко інтегрований у будь-яку частину додатку, де потрібна така функціональність.

Компонент balance-item:

– містить імпорти:

o import React, { FC } from "react" – імпорт React та типу FunctionComponent;

o import "./index.css" – імпорт CSS стилів;

– інтерфейс пропсів:

```
interface ComponentProps {
  img: string;          // URL іконки
  date: string;         // Дата операції
```

```
    title: string;          // Заголовок операції
    type: string;           // Тип операції
    sum?: number | null;    // Сума (опціонально)
    isPositive?: boolean | null; // Чи позитивна сума (для стилізації)
}
```

Структура компонента:

- основний контейнер (bal-item) з властивостями:
  - flex-контейнер з розміщенням по краях;
  - білий фон з закругленими кутами;
- лівий блок (bal-item__block):
  - іконка у круглому контейнері (48×48px);
  - текстовий контент з: заголовком (h4) та рядком метаданих (дата, тип);
- правий блок (bal-item__sum):
  - відображення суми;
  - умовна стилізація (зелений колір для позитивних значень);
- особливості:
  - умовний рендеринг: додає «+» для позитивних сум;
  - мемоізація: React.memo для оптимізації;
  - адаптивність: Гнучкі розміри через flexbox;
  - семантичний HTML: Використання strong для сум.
- стилізація:
  - створення круглої іконки з фоном #f7f7f7, радіусом 50%;
- типографіка:
  - жирний заголовок (font-weight: bold);
  - сірий текст для метаданих (#939199)
  - роздільник: круглий елемент 4×4px між датою і типом

Компонент button:

– містить імпорти import { FC } from "react" – імпорт типу FunctionComponent з React;

– import "./index.css" – імпорт CSS стилів для кнопки

– інтерфейс пропсів:

```
interface ComponentProps {
  text: string;        // Текст кнопки
  className:  string;        //  Клас  для  вибору  стилю (primary/white/red)
  disabled?: string;   // Опціональний стан disabled
}
```

– основні характеристики:

- розміри: ширина (353px) та висота (56px);
- тип: кнопка “submit” (type="submit");

– стилі:

- flex-контейнер з центруванням вмісту;
- закруглені кути (border-radius: 12px);
- плавна анімація opacity при наведенні;

– варіанти стилізації:

- властивість primary: фіолетовий фон (#775ce5), білий текст;
- white: фіолетова рамка, фіолетовий текст;
- red: червона рамка, червоний текст;

– стан disabled (неактивності):

- знижена прозорість (opacity: 0.5);
- стандартний курсор (не pointer);
- додається клас “disabled”.

Кнопка-компонент “button-pay”:

– містить імпорти: import { FC, MouseEventHandler } from "react" – імпорт

FunctionComponent та типу для обробника кліків;

– import "./index.css" – імпорт стилів для компонента;

Інтерфейс пропсів:

```
interface ComponentProps {
  title: string; // Назва платіжної системи
  logo: string; // URL маленької іконки
  img: string;  // URL великого зображення
  onClick?: MouseEventHandler<HTMLButtonElement>; // Обробник кліку
  disabled?: string; // Стан disabled
}
```

Структура компонента:

кореневий елемент:

– семантичний <button>;

– класи “payment__sys” + disabled (якщо активовано);

– обробник події onClick;

лівий блок:

– маленька іконка (19×19px);

– назва платіжної системи;

правий блок:

– велике зображення (161×21px) - логотип системи

Компонент заголовку “heading”:

– містить імпорти: import { FC } from "react" – імпорт типу FunctionComponent;

– import "./index.css" – імпорт CSS стилів

Інтерфейс пропсів:

```
interface ComponentProps {
  title: string;        // Головний заголовок
  description?: string; // Опціональний підзаголовок
```

```
  className?: string;   // Клас для стилізації заголовка
}
```

Структура компонента:

- кореневий контейнер (div.heading):
  - flex-контейнер з колонковим напрямком;
  - центрування по вертикалі та горизонталі;
  - відступи між елементами 12px;
  - нижній відступ 32px;
- заголовок (div):
  - жирний шрифт (font-weight: bold);
  - приймає додаткові класи через проп className;
- опис (p):
  - сірий колір тексту (#939199).

Рендериться тільки при наявності пропсу description.

Проєкт містить спеціальні класи:

- “.wellc” містить великий білий текст (28px);
- “.auth” – середній розмір тексту (24px);
- .private – менший розмір тексту (20px).

Приклад використання компонента Heading:

```
<Heading
  title="Welcome Back!"
  description="Please enter your details"
  className="wellc"
/>
<Heading
  title="Private Office"
  className="private"
/>
```

Даний компонент ідеально підходить для:

– заголовків сторінок;

– форм авторизації;

– привітальних повідомлень;

– секцій інтерфейсу

Даний компонент поєднує простоту використання з гнучкістю стилізації, що робить його ідеальним вибором для створення послідовного дизайн-системи.

Компонент-вводу “input”:

Містить імпорти:

– ChangeEvent, FC з React - для типізації подій і компонента;

– стилі з двох файлів: index.css (основні) та form.css (додаткові);

Інтерфейс пропсів:

```
typescript
interface ComponentProps {
  label?: string;  // Опціональний текст мітки
  type: string;    // Тип інпута (text, password, email тощо)
  name: string;    // Атрибут name для форми
  placeholder: string; // Плейсхолдер
  onChange?: (event: ChangeEvent<HTMLInputElement>) => void; //
```

Обробник змін:

```
  value?: string;  // Поточне значення
  message?: string | null; // Повідомлення про помилку
}
```

3. Структура компонента:

зовнішній контейнер (`div.field`):

– flex-контейнер з колонковим напрямком;

– відступ 8px між елементами;

мітка (`label.field__label`):

– відображається тільки якщо передано `label`;

– прив’язана до input через `htmlForm={name}`

поле вводу *(input.field__input):*

– фіксовані розміри (353×56px);

– світло-сіра рамка *(#e9e8eb);*

– білий фон;

– закруглені кути (*12px*);

– внутрішні відступи (*16px* з боків)

Містить повідомлення про помилку:

– відображення при наявності message;

– додавання класу error (червона рамка) до input.

Стани:

– фокус: фіолетова рамка *(#775ce5)*;

– помилка: червона рамка (клас *error*).

Даний компонент є універсальним полем вводу для:

– форм реєстрації/авторизації;

– профілів користувачів;

– платіжних даних;

– пошукових інтерфейсів.

Поєднує чітку структуру, гнучкість через пропси та професійну стилізацію, що робить його ідеальним вибором для будь-яких форм у додатку.

Компонент вводу-паролю “input-password”:

Імпорти:

– *import { ChangeEvent, FC, useState } from "react"* – імпорт необхідних типів і хуків;

– *import "./index.css"* – імпорт основних стилів;

– *import "../../style/form.css"* – імпорт додаткових стилів форми.

2. Інтерфейс пропсів:

typescript

```
interface ComponentProps {
  label: string;          // Текст мітки для поля вводу
  name: string;           // Атрибут name для input
  placeholder: string;    // Плейсхолдер
  onChange?: (value: string) => void; // Обробник зміни значення
  value?: string;         // Поточне значення
  message?: string | null; // Повідомлення про помилку
}
```

3. Логіка компонента:

– стан *showPassword:* контролює відображення пароля (true – текст, false – зірочки).

Оновлюється через *setShowPassword*

Обробники подій:

– *handleInputChange:* передає значення вводу батьківському компоненту;

– *handleToggleClick*: перемикає видимість пароля.

Рендер:

– *мітка (label) для доступності;*

– *поле вводу (input) з умовним типом (password/text);*

– *іконка перемикача видимості пароля;*

– *повідомлення про помилку (при наявності).*

4. Стилізація:

Позиціонування:

– відносне позиціонування обгортки;

– абсолютне позиціонування іконки (праворуч).

Іконки:

– дві SVG-іконки (показати/приховати пароль);

– різні розміри для різних станів.

Помилка:

– містить червону рамку при наявності повідомлення про помилку.

Код компонента Page:

```
import { ReactNode } from "react";
import "./index.css";
interface ComponentProps {
  children: ReactNode;
}
const Component: React.FC<ComponentProps> = ({ children }) => {
  return <div className="page">{children}</div>;
};
export default Component;
```

Стилі (index.css):

```
.page {
  width: 353px;
  height: 852px;
  margin: 0 auto;
  border-radius: 24px;
  background: #fff;
  box-shadow: 0px 0px 30px 0px rgba(0, 0, 0, 0.25);
}
```

Детальний аналіз:

Імпорти:

– `import { ReactNode } from "react"` – імпорт типу для дочірніх елементів;

– `import "./index.css"` – імпорт стилів для сторінки

Інтерфейс пропсів:

```
interface ComponentProps {
  children: ReactNode; // Дочірні елементи, які будуть відрендерені всередині сторінки
}
```

Основні характеристики:

– розміри: фіксовані `width (393px)` і `height (852px)` – стандартні розміри для мобільних пристроїв;

– позиціонування: центрування за допомогою властивості `margin: 0 auto.`

Візуальні ефекти:

– білий фон (`#fff`);

– закруглені кути (`24px`);

– тінь (`box-shadow`) для ефекту "піднятої" сторінки.

Призначення:

– виступає в якості контейнера для всієї сторінки додатка;

– імітує вигляд мобільного додатка на десктопі;

– забезпечує послідовність дизайну всіх сторінок.

Даний компонент є базовим будівельним блоком для:

– мобільних додатків на React;

– візуального відокремлення контенту;

– забезпечення послідовності дизайну;

– створення ефекту «мобільного додатка» на десктопі.

Простота і універсальність компонента роблять ідеальним вибором для створення консистентного інтерфейсу.

3.3. Особливості налаштування серверної частини мобільного додатку

Нижче наведено перелік ключових особливостей налаштування серверної частини мобільного додатку:

– використання Firebase як “Backend-as-a-Service”. Проєкт використовує Firebase як повноцінну серверну платформу, що дозволяє обійтися без власного серверного коду на Node.js чи інших технологій:

– Firebase Authentication – реалізована автентифікація користувачів через електронну пошту/пароль. Конфігурація знаходиться в src/firebase.js, де ініціалізуються сервіси Firebase з використанням конфігураційних даних.

– Firestore Database – використовується для зберігання даних користувачів, транзакцій та іншої банківської інформації. Налаштування підключення до Firestore також міститься в src/firebase.js.

Особливості конфігурації:

– Environment Variables – проєкт використовує змінні оточення(замінити на інше слово) для зберігання конфіденційних даних, таких як API-ключі Firebase. Вони зберігаються в .env файлі (який не включений до репозиторію через .gitignore);

– безпека API – усі запити до Firebase захищені правилами безпеки, які, ймовірно, налаштовані в консолі Firebase (у самому коді проекту ці правила не представлені).

API-взаємодія містить специфічні методи взаємодії:

– у src/api/ знаходяться модулі для роботи з API (наприклад, transactionsApi.js, userApi.js), які використовують Firebase SDK для виконання CRUD-операцій;

– для роботи з транзакціями використовуються спеціалізовані методи, такі як “getTransactionsByUserId”, “createTransaction”;

– реалізована базова обробка помилок при взаємодії з Firebase, що видно в методах API, де використовуються блоки try/catch.

Оптимізація для мобільних пристроїв:

– офлайн-робота – Firebase SDK налаштований для підтримки офлайн-роботи (можливість кешування даних і синхронізації при відновленні з’єднання);

– швидкість відгуку – використовуються оптимізовані запити до Firestore з обмеженням кількості даних, що отримуються (наприклад, лімітування

кількості транзакцій при першому запиті).

Серверна частина мобільного додатку React Bank побудована на принципах сучасного «Serverless підходу» з використанням Firebase як основної платформи. Це дозволяє:

- мінімізувати витрати на розробку власного серверного коду;
- забезпечити масштабованість рішення;
- використовувати вбудовані механізми безпеки Firebase;
- швидко розгортати оновлення.

Основним недоліком такого підходу є залежність від Firebase і потенційні обмеження платформи при складних банківських операціях.

Таким чином, розроблений мобільний додаток є прикладом сучасного, ефективного, масштабованого рішення для реалізації фінансових сервісів, з орієнтацією на безпеку, зручність і мобільність.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було реалізовано повноцінний цикл створення мобільного додатку фінансового сервісу, починаючи від аналізу теоретичних основ FinTech-сектору, і завершуючи розробкою клієнтської та серверної частини продукту. За результатами даного дослідження можна зробити наступні висновки:

1. Проведено теоретичний аналіз сучасних тенденцій у сфері фінансових технологій (FinTech), що є основою для розробки мобільних додатків фінансового сервісу. Внаслідок проведеного аналізу доведено, що цифровізація суттєво трансформує фінансовий сектор, сприяючи розвитку FinTech-компаній, які запроваджують інноваційні платформи для фінансових послуг, покращують доступність, швидкість та зручність сервісів для користувачів.

2. Ринок FinTech в Україні демонструє позитивну динаміку, що підтверджується зростанням кількості фінтех-компаній, впровадженням Open Banking API, запуском необанків, проєктами в сфері криптовалют і державною підтримкою в межах Стратегії розвитку FinTech до 2025 року. Конкурентне середовище мобільних додатків активно розвивається: провідні рішення, такі як Money Manager, Monefy, Wallet, Money Lover, Mobills, пропонують різні функції – від простого обліку витрат до комплексного фінансового планування з аналітикою, візуалізацією та синхронізацією даних.

3. В межах дослідження функціональних властивостей мобільного додатку фінансового сервісу обгрунтовано вибір стеку технологій для реалізації мобільного додатку: React.js, TypeScript, Node.js, CSS, JSX та інших технологій, які забезпечують крос-платформенність, інтерактивність та масштабованість додатку.

4. Функціональні можливості додатку охоплюють повний цикл обслуговування користувача: реєстрація, авторизація, перегляд балансу,

поповнення, переказ коштів, перегляд історії транзакцій, налаштування профілю та сповіщення. Це свідчить про високий рівень продуманості та зручності взаємодії з додатком.

5. Інтерфейс користувача був розроблений згідно з принципами UI/UX-дизайну, з використанням Figma для створення макетів і React для реалізації компонентів. Це забезпечило модульність, гнучкість та легкість масштабування інтерфейсу. Усі компоненти адаптовані до мобільних пристроїв і відповідають сучасним вимогам зручності та функціональності.

6. Серверна частина побудована з використанням хмарної платформи Firebase, що дозволяє уникнути створення власного сервера [20]. Впроваджено: безпечну автентифікація користувачів; базу даних для зберігання транзакцій та даних користувачів [20]; environment Variables — для безпечного зберігання ключів; офлайн-режим та кешування, що підвищує надійність додатку на мобільних пристроях.

7. Проєкт реалізований відповідно до принципів Serverless-архітектури, що забезпечує масштабованість, високу швидкість розгортання та зменшує витрати на підтримку інфраструктури. Слабкою стороною обраної архітектури є потенційна залежність від функціоналу Firebase, що може створити обмеження в майбутньому при реалізації складніших бізнес-логік.

Отже, за результатами дослідження був створений функціональний і безпечний мобільний додаток фінансового сервісу. Подальші перспективи полягають у розширенні функціоналу, зокрема інтеграції з банківськими API, криптогаманцями, чат-ботами та AI-аналітикою фінансової поведінки користувача.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. 10 застосунків і програм для роботи з грошима. URL: https://bazilik.media/10-zastosunkiv-i-prohram-dlia-roboty-z-hroshyma (date of access: 30.05.2025).
2. Adedeji O. Full-Stack Flask and React: Learn, Code and Deploy Powerful Web Applications with Flask 2 and React 18. Packt Publishing, Limited, 2023.
3. Biswas N. TypeScript Basics. Berkeley, CA : Apress, 2023. URL: https://doi.org/10.1007/978-1-4842-9523-6 (date of access: 30.05.2025).
4. Boduch A., Derks R. React and React Native: A Complete Hands-On Guide to Modern Web and Mobile Development with React. js, 3rd Edition. Packt Publishing, Limited, 2020. 526 p.
5. Digital tiger: the Market Power of Ukrainian IT (research for 2024). URL: https://itukraine.org.ua/files/DigitalTiger2024.pdf (date of access: 30.05.2025).
6. Fintech в Україні 2025: проблеми, тренди, інновації. URL: https://mezha.media/articles/chim-zhive-ukrajinskiy-fintech-sektor-300651/ (date of access: 29.05.2025).
7. Freeman A. Essential TypeScript 4. Berkeley, CA : Apress, 2021. URL: https://doi.org/10.1007/978-1-4842-7011-0 (date of access: 30.05.2025).
8. Getting started with React - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started (date of access: 29.05.2025).
9. GitHub - maximtsyrulnyk/react-bank-new: repository for Final Project (Junior FullStack Developer). GitHub. URL: https://github.com/maximtsyrulnyk/react-bank-new.git (date of access: 29.05.2025).
10. Introducing JSON. URL: https://www.json.org/json-en.html(date of access: 29.05.2025).
11. Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-

started/introduction-to-nodejs. (date of access: 29.05.2025).

12. Introduction to the DOM. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction(date of access: 29.05.2025).

13. JSON – Introduction. URL: https://www.w3schools.com/js/js_json_intro.asp (date of access: 29.05.2025).

14. Mangabo K. Full Stack Django and React: Get Hands-On Experience in Full-stack Web Development with Python, React, and AWS. Packt Publishing, Limited, 2023.

15. Node.js Tutorial. URL: https://www.w3schools.com/nodejs/(date of access: 29.05.2025).

16. Qiu J. Test-Driven Development with React and TypeScript. Berkeley, CA : Apress, 2023. URL: https://doi.org/10.1007/978-1-4842-9648-6 (date of access: 30.05.2025).

17. Quick Start – React. React. URL: https://react.dev/learn (date of access: 29.05.2025).

18. Real-World Next. js: Build Scalable, High-Performance, and Modern Web Applications Using Next. js, the React Framework for Production. de Gruyter GmbH, Walter, 2022. URL: https://react.dev/learn(date of access: 29.05.2025).

19. Remotely Manage the Power of Iot Devices / S. Tsyrulnyk et al. Èlektronnoe modelirovanie. 2024. Vol. 46, no. 5. P. 74–91. URL: https://doi.org/10.15407/emodel.46.05.074 (date of access: 29.05.2025).

20. Tsyrulnyk S., Tromsiuk V., Bogach I., Tsyrulnyk M., Potapova N. Programmable Power Unit Supplies Based on Quick Charge Technology. CEUR Workshop Proceedings. 2023. Vol. 3628. 3. 136-150. URL: https://ceur-ws.org/Vol-3628/paper11.pdf(date of access: 29.05.2025).

21. Tsyrulnyk S., Tsyrulnyk M., Potapova N., Semenov A. and Tromsyuk V. The climate control system using ESP8266 and Arduino IoT Cloud. CEUR Workshop Proceedings. 2022. Vol. 3309. P. 462-477. Code 185325. URL: https://ceur-

ws.org/Vol-3309/paper27.pdf (date of access: 29.05.2025).

22. Tsyrulnyk S. Mobile applications and online wi-fi monitoring platforms of weather stations. Open educational e-environment of modern university. 2020. No. 9. P. 181–192. URL: https://doi.org/10.28925/2414-0325.2020.9.15 (date of access: 29.05.2025).

23. Tsyrulnyk S. M., Tkachuck V. M., Tsyrulnyk M. O. Prototyping IOT project at WOKWI service. 16th IC Measurement and Control in Complex Systems, Vinnytsia, Ukraine, 15–17 November 2022. Vinnytsia, 2022. URL: https://doi.org/10.31649/mccs2022.03 (date of access: 29.05.2025).

24. Tsyrulnyk, M. Tsyrulnyk, V. Tkachuk, O. Kylymchuk. Remotely manage the power of iot devices. Electronic Modeling. 2024, 46(5):74-91. https://doi.org/10.15407/emodel.46.05.074

25. What is TypeScript? URL: https://www.typescriptlang.org/(date of access: 29.05.2025).

26. Zammetti F. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker. Apress, 2020. 395 p.

27. Zhytar M. Fintech market in Ukraine: features, ways and prospects of development. European scientific journal of Economic and Financial innovation №1(13) 2024, p.4-11. DOI: http://doi.org/10.32750/2024-0101(date of access: 29.05.2025).

28. Білас О.Є. Якість програмного забезпечення та тестування. Львів : Львівська політехніка, 2011. 216 с.

29. Бородкіна І. Л., Бородкін Г. О. Web-технології та Web-дизайн : застосування мови HTML для створення електронних ресурсів. К: Видавництво Ліра-К, 2020. 212 с.

30. Десять найперспективніших напрямів розвитку фінтеху в Україні. URL: https://fintechinsider.com.ua/desyat-najperspektyvnishyh-napryamiv-rozvytku-fintehu-v-ukrayini/ (date of access: 29.05.2025).

31. Каталог фінтех-компаній України 2024. URL: https://fintechua.org/

32. Пасічник В.В. Веб-дизайн. Підручник. Львів : «Магнолія-2006», 2025. 336 с.

33. Пастернак І.І., Костик А.Т. Інструментальні засоби веб-технологій. Львів : «Магнолія-2006», 2025. 336 с.
34. Проскурович О. В., Бойчук В. А. Комп'ютерні технології економічного аналізу. Навч. посібник. Львів: Новий Світ, 2024. 312 с.
35. Розробка фінтех-додатків: тенденції, особливості та перспективи. URL: https://stfalcon.com/uk/blog/post/fintech-app-development-trends-features-perspective (date of access: 29.05.2025).
36. Сенів М. М., Яковина В. С. Безпека програм та даних. Навчальний посібник. Львів : Львівська політехніка, 2015. 256 с.
37. Стратегія цифрового розвитку інноваційної діяльності України на період до 2030 року. URL: https://winwin.gov.ua/ (date of access: 29.05.2025).
38. Фінансова грамотність у вашій кишені: огляд мобільних додатків для управління особистими фінансами. URL: https://banker.ua/uk/projects/oglyad-dodatkiv-dlya-upravlinnya-osobistimi-finansami
39. Цирульник М. С., Цирульник С. М. Розробка додатку погода засобами фреймворка XAMARIN. Матеріали II Міжнародної науково-практична конференції «Прикладні аспекти сучасних міждисциплінарних досліджень». ДонНУ імені Василя Стуса, м. Вінниця, 24 листопада 2023 року URL: https://jpasmd.donnu.edu.ua/article/view/14817 (date of access: 29.05.2025).
40. Цирульник М. Текст створений за допомогою штучного інтелекту та відредагований автором. ChatGPT [GPT-4o mini]. 2025. URL: https://chat.openai.com (дата звернення: 01.04.2025).

ДОДАТКИ

ДОДАТОК А

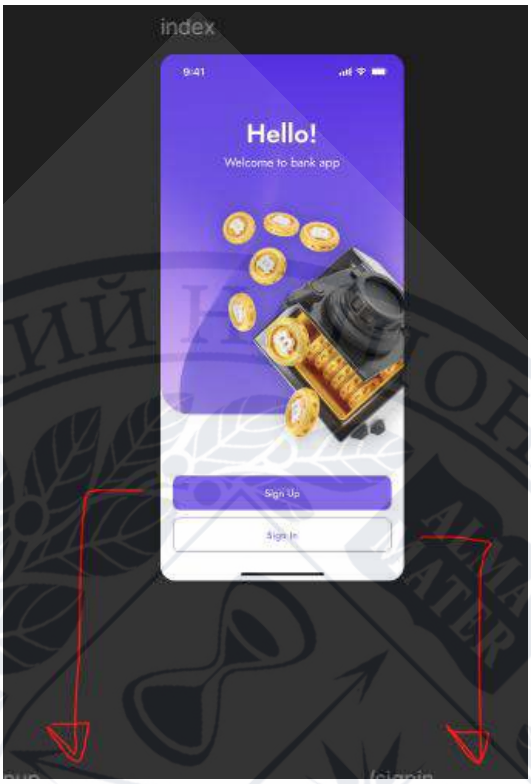


Рисунок А.1 –Дизайн сторінки “Wellcome”

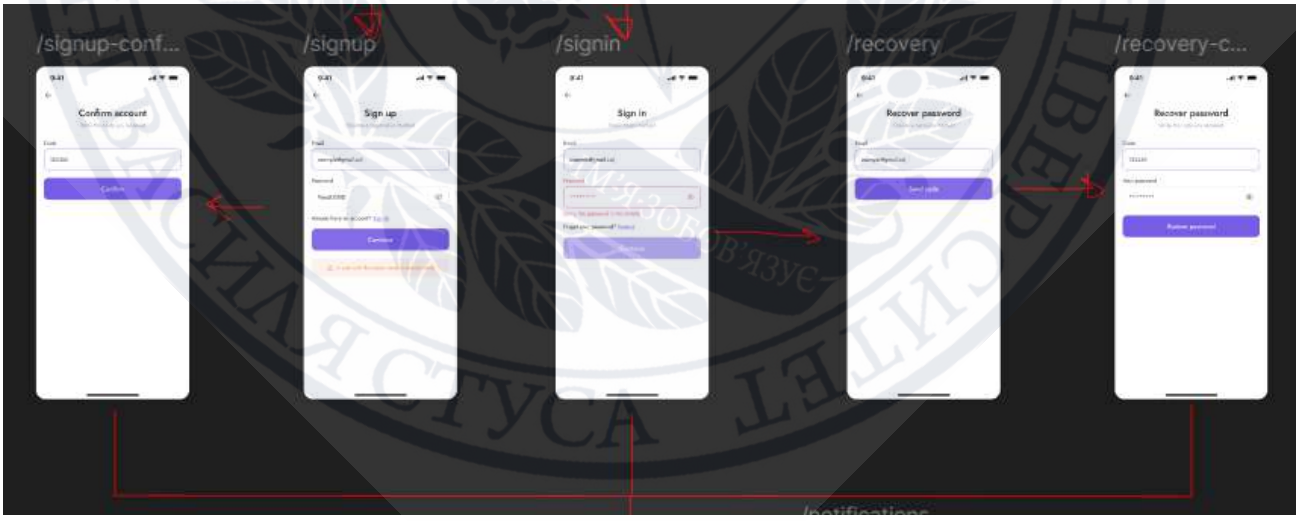


Рисунок А.2 – Дизайн сторінок “SignUp-Confirn”, “SignUp”, “SignIn”, “Recovery”, “Recovery-Confirm”

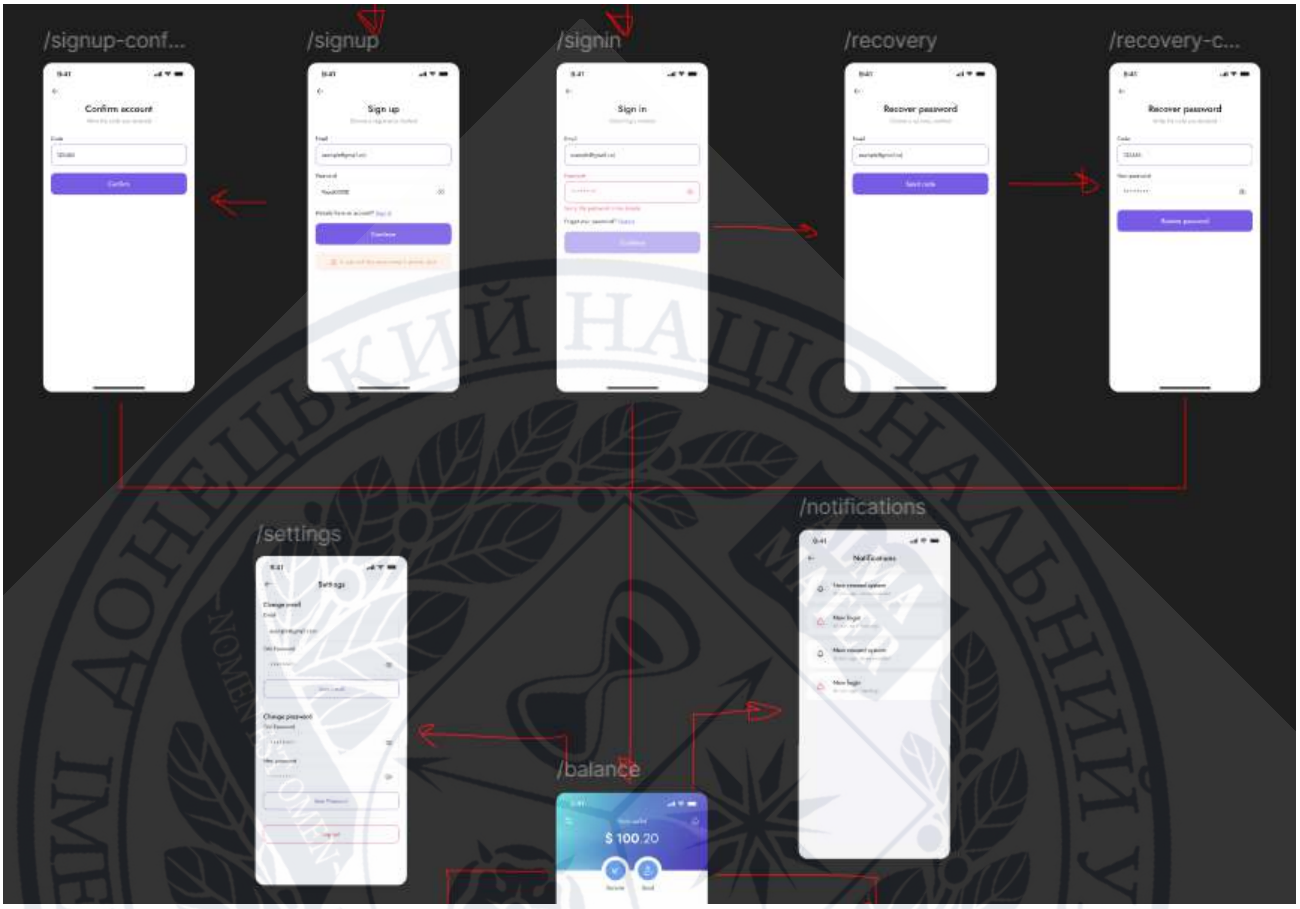


Рисунок А.3 – Дизайн сторінок “SignUp-Confirn”, “SignUp”, “SignIn”, “Recovery”, “Recovery-Confirm”, “Settings”, “Notifications”, “Balance”

Рисунок А.4 – Дизайн сторінок “Settings”, “Notifications”, “Balance”, “Receive”, “Send”, “Transaction”

Рисунок А.5 – Зовнішній виглад головної сторінки “Wellcome”

ДОДАТОК Б

```
List sortedList = people.stream()
.sorted(Comparator.comparingInt(Person::getAge))
.toList();
```

Back/index.js

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */
const app = require('./app')
const debug = require('debug')(
  'template-express-live-reload:server',
)
const http = require('http')
/**
 * Get port from environment and store in Express.
 */
const port = normalizePort(process.env.PORT || '4000')
app.set('port', port)
/**
 * Create HTTP server.
 */
const server = http.createServer(app)
/**
 * Listen on provided port, on all network interfaces.
 */
```

```
server.listen(port)
server.on('error', onError)
server.on('listening', onListening)
/**
 * Normalize a port into a number, string, or false.
 */
function normalizePort(val) {
  const port = parseInt(val, 10)
  if (isNaN(port)) {
    // named pipe
    return val
  }
  if (port >= 0) {
    // port number
    return port
  }
  return false
}
/**
 * Event listener for HTTP server "error" event.
 */
function onError(error) {
  if (error.syscall !== 'listen') {
    throw error
  }
  const bind =
    typeof port === 'string'
      ? 'Pipe ' + port
      : 'Port ' + port
  // handle specific listen errors with friendly messages
```

```
  switch (error.code) {
    case 'EACCES':
      console.log(bind + ' requires elevated privileges')
      process.exit(1)
      break
    case 'EADDRINUSE':
      console.log(bind + ' is already in use')
      process.exit(1)
      break
    default:
      throw error
  }
}
/**
 * Event listener for HTTP server "listening" event.
 */
function onListening() {
  const addr = server.address()
  const bind =
    typeof addr === 'string'
      ? 'pipe ' + addr
      : 'port ' + addr.port
  debug('Listening on ' + bind)
  console.log(
    'Listening on ' + 'http://localhost:' + addr.port,
  )
}
```

Back/route/auth.js

```
// Підключаємо технологію express для back-end сервера
const express = require('express')
// Створюємо роутер - місце, куди ми підключаємо ендпоїнти
const router = express.Router()
const { User } = require('../class/user')
const { Confirm } = require('../class/confirm')
const { Session } = require('../class/session')
const { Balance } = require('../class/balance')
const { Notification } = require('../class/notification')
// ================================================================
router.post('/signup', function (req, res) {
  const { email, password } = req.body
  console.log(email, password)
  if (!email || !password) {
    return res.status(400).json({
      message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const user = User.getByEmail(email)
    if (user) {
      return res.status(400).json({
        message: 'Помилка. Такий користувач вже існує',
      })
    }
    const newUser = User.create({ email, password })
    Balance.create(newUser.id)
    const session = Session.create(newUser)
    Confirm.create(newUser.email)
    return res.status(200).json({
```

```
      message: 'Користувач успішно зареєстрований',
      session,
    })
  } catch (err) {
    return res.status(400).json({
      message: 'Помилка створення користувача',
    })
  }
})
// ================================================================
router.post('/signup-confirm', function (req, res) {
  const { code, token } = req.body
  console.log(code, token)
  if (!code || !token) {
    return res.status(400).json({
    message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const session = Session.get(token)
    if (!session) {
      return res.status(400).json({
        message: 'Помилка. Ви не увійшли в акаунт',
      })
    }
    const email = Confirm.getData(Number(code))
    if (!email) {
      return res.status(400).json({
        message: 'Код не існує',
      })
```

```
    }
    if (email !== session.user.email) {
      return res.status(400).json({
        message: 'Код не дійсний',
      })
    }
    const user = User.getByEmail(session.user.email)
    user.isConfirm = true
    session.user.isConfirm = true
    return res.status(200).json({
      message: 'Ви підтвердили свою пошту',
      session,
    })
  } catch (err) {
    return res.status(400).json({
      message: err.message,
    })
  }
})
// ========================================================
router.post('/recovery', function (req, res) {
  const { email } = req.body
  if (!email) {
    return res.status(400).json({
      message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const user = User.getByEmail(email)
    if (!user) {
```

```
      return res.status(400).json({
        message: 'Користувач з таким email не існує',
      })
    }
    Confirm.create(email)
    return res.status(200).json({
      message: 'Код для відновлення паролю відправлено',
    })
  } catch (err) {
    return res.status(400).json({
      message: err.message,
    })
  }
})
// =====================================================
router.post('/recovery-confirm', function (req, res) {
  const { password, code } = req.body
  console.log(password, code)
  if (!code || !password) {
    return res.status(400).json({
      message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const email = Confirm.getData(Number(code))
    if (!email) {
      return res.status(400).json({
        message: 'Код не існує',
      })
    }
```

```
    const user = User.getByEmail(email)
    if (!user) {
      return res.status(400).json({
        message: 'Користувач з таким email не існує',
      })
    }
    user.password = password
    console.log(user)
    const session = Session.create(user)
    if (!session.user.isConfirm) {
      Confirm.create(session.user.email)
    }
    const notification = Notification.create({
      type: 'Warning',
      title: 'Відновлення акаунту',
      img: '../../../svg/not-danger.svg',
      id: user.id,
    })
    if (!notification) {
      return res.status(400).json({
        message: 'Помилка створення нотифікації',
      })
    }
    return res.status(200).json({
      message: 'Пароль змінено',
      session,
    })
  } catch (err) {
    return res.status(400).json({
      message: err.message,
```

```
    })
  }
})
// ========================================================
router.post('/signin', function (req, res) {
  const { email, password } = req.body
  if (!email || !password) {
    return res.status(400).json({
      message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const user = User.getByEmail(email)
    if (!user) {
      return res.status(400).json({
        message:
          'Помилка. Користувача з таким email не існує',
      })
    }
    if (user.password !== password) {
      return res.status(400).json({
        message: 'Помилка. Пароль не підходить',
      })
    }
    const session = Session.create(user)
    if (!session.user.isConfirm) {
      Confirm.create(session.user.email)
    }
    const notification = Notification.create({
      type: 'Warning',
```

```
      title: 'Вхід в акаунт',
      img: '../../../svg/not-danger.svg',
      id: user.id,
    })
    if (!notification) {
      return res.status(400).json({
        message: 'Помилка створення нотифікації',
      })
    }
    return res.status(200).json({
      message: 'Ви увійшли',
      session,
    })
  } catch (err) {
    return res.status(400).json({
      message: err.message,
    })
  }
})
router.post('/settings-change-email', function (req, res) {
  const { email, password, token, user } = req.body
  if (!email || !password || !token || !user) {
    return res.status(400).json({
      message: "Помилка. Обов'язкові поля відсутні",
    })
  }
  try {
    const sessionUser = Session.get(token)
    if (!sessionUser) {
      return res.status(400).json({
```

```
    message: 'Токен не вірний',
  })
}
const newUser = User.getById(user.id)
if (!newUser) {
  return res.status(400).json({
    message: 'Користувач з таким id не існує',
  })
}
if (
  String(user.email).toLowerCase() !== newUser.email
) {
  return res.status(400).json({
    message: 'Email не вірний',
  })
}
if (String(password) !== newUser.password) {
  return res.status(400).json({
    message: 'Password не вірний',
  })
}
newUser.email = email
sessionUser.user.email = email
const notification = Notification.create({
  type: 'Warning',
  title: 'Зміна пошти',
  img: '../../../svg/not-danger.svg',
  id: user.id,
})
if (!notification) {
```

```
      return res.status(400).json({
        message: 'Помилка створення нотифікації',
      })
    }
    return res.status(200).json({
      message: 'Email змінено',
      sessionUser,
    })
  } catch (err) {
    return res.status(400).json({
      message: err.message,
    })
  }
})
// ====================================================
router.post(
  '/settings-change-password',
  function (req, res) {
    const { oldPassword, newPassword, token, user } =
      req.body
    if (!oldPassword || !newPassword || !token || !user) {
      return res.status(400).json({
        message: "Помилка. Обов'язкові поля відсутні",
      })
    }
    try {
      const sessionUser = Session.get(token)
      if (!sessionUser) {
        return res.status(400).json({
          message: 'Токен не вірний',
```

```
    })
  }
  const newUser = User.getById(user.id)
  if (!newUser) {
    return res.status(400).json({
      message: 'Користувач з таким id не існує',
    })
  }
  if (
    String(user.email).toLowerCase() !== newUser.email
  ) {
    return res.status(400).json({
      message: 'Email не вірний',
    })
  }
  if (String(oldPassword) !== newUser.password) {
    return res.status(400).json({
      message: 'Password не вірний',
    })
  }
  newUser.password = newPassword
  const notification = Notification.create({
    type: 'Warning',
    title: 'Зміна пароля',
    img: '../../../svg/not-danger.svg',
    id: user.id,
  })
  if (!notification) {
    return res.status(400).json({
      message: 'Помилка створення нотифікації',
```

```
        })
      }
      console.log(newUser)
      return res.status(200).json({
        message: 'Пароль змінено',
      })
    } catch (err) {
      return res.status(400).json({
        message: err.message,
      })
    }
  },
)
// =====================================================================
// Підключаємо роутер до бек-енду
module.exports = router
```

transaction.js

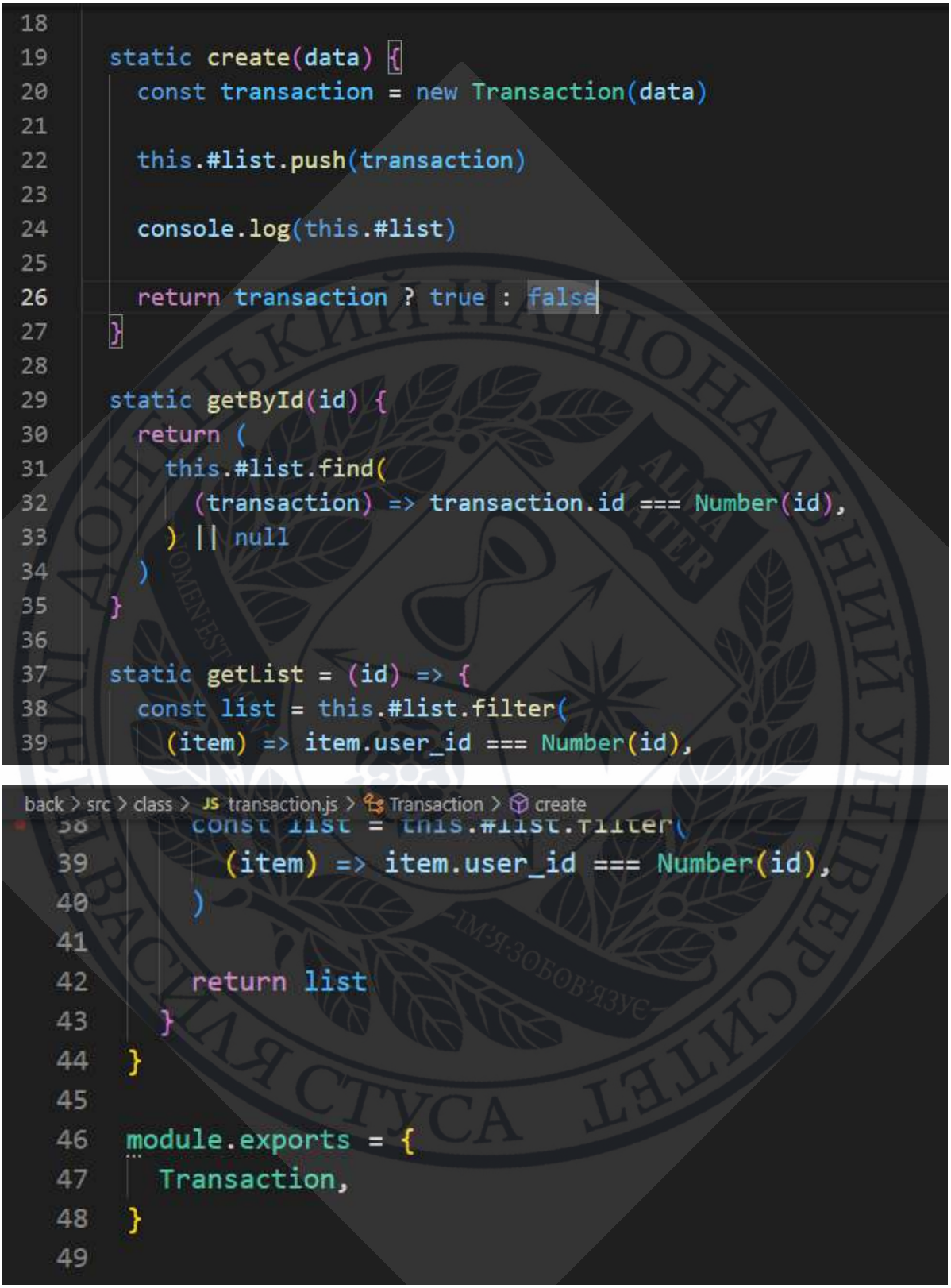
```
class Transaction {
  static #list = []

  static #count = 1

  constructor({ id, email, type, sum, title, img }) {
    this.id = Transaction.#count++

    this.email = String(email).toLowerCase()
    this.type = type
    this.sum = Number(sum)
    this.title = title
    this.img = img
    this.user_id = id

    this.date = new Date().getTime()
  }
```



```
18
19    static create(data) {
20      const transaction = new Transaction(data)
21
22      this.#list.push(transaction)
23
24      console.log(this.#list)
25
26      return transaction ? true : false
27    }
28
29    static getById(id) {
30      return (
31        this.#list.find(
32          (transaction) => transaction.id === Number(id),
33        ) || null
34      )
35    }
36
37    static getList = (id) => {
38      const list = this.#list.filter(
39        (item) => item.user_id === Number(id),
```



```
back > src > class > JS transaction.js > Transaction > create
39          (item) => item.user_id === Number(id),
40        )
41
42        return list
43      }
44    }
45
46    module.exports = {
47      Transaction,
48    }
49
```

Рисунок Б.1 – Лістинг класу “Transaction”

User.js

```
class User {
  static #list = []

  static #count = 1

  constructor({ email, password }) {
    this.id = User.#count++

    this.email = String(email).toLowerCase()
    this.password = String(password)

    this.isConfirm = false
  }
```

```
  static create(data) {
    const user = new User(data)

    console.log(user)

    this.#list.push(user)

    console.log(this.#list)

    return user
  }
```

```
        saveSession({
          token: data.session.token,
          user: data.session.user,
        });
```

```
import { saveSession } from "../../util/session";
```

```
        saveSession({
          token: data.sessionUser.token,
          user: data.sessionUser.user,
        });
```

```
import { saveSession } from "../../util/session";
```

```
        saveSession({
          token: data.session.token,
          user: data.session.user,
        });
```

```
import { saveSession } from "../../util/session";
```

```
        saveSession({
          token: data.session.token,
          user: data.session.user,
        });
```

```
import { saveSession } from "../../util/session";
```

```
  static getByEmail(email) {
    return (
      this.#list.find(
        (user) =>
          user.email === String(email).toLowerCase(),
      ) || null
    )
  }
```

```
3  export const saveSession = (value: object) => {
4    try {
5      sessionStorage.setItem(SESSION_KEY, JSON.stringify(value));
6    } catch (er) {
7      console.error("Error saving session:", er);
8      sessionStorage.clear();
9    }
10 };
```

```
36   static getById(id) {
37     return (
38       this.#list.find((user) => user.id === Number(id)) ||
39       null
40     )
41   }
42
43   static getList = () => this.#list
44 }
45
46 module.exports = {
47   User,
48 }
```

Рисунок Б.2 – Методи класу “User” (продовження)

back/src/route/index.js

back > src > route > JS index.js > ...

```
1   // Підключаємо роутер до бек-енду
2   const express = require('express')
3   const router = express.Router()
4
5   // Підключіть файли роутів
6   // const test = require('./test')
7   // Підключіть інші файли роутів, якщо є
8
9   // Об'єднайте файли роутів за потреби
10
11  // router.use('/', test)
12  // Використовуйте інші файли роутів, якщо є
13
14  router.get('/', (req, res) => {
15    res.status(200).json('Hello World')
16  })
17
18  const auth = require('./auth')
19  const balance = require('./balance')
20  // const user = require('./user')
21
22  router.use('/', auth)
23  router.use('/', balance)
24  // router.use('/', user)
25
26  // Експортуємо глобальний роутер
27  module.exports = router
```

Рисунок Б.3 –Методи класу “User” (продовження)

ДОДАТОК В

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)