МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЄМЕЛЬЯНОВА АНАСТАСІЯ ОЛЕКСАНДРІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

ЗАСТОСУНОК ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ТА КОНТРОЛЮ ОСОБИСТИХ ФІНАНСІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
О. В. Зелінська, доцент кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____
(Підпис)

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

Голова ЕК: _____
(Підпис)

Вінниця — 2024

АНОТАЦІЯ

Ємельянова А.О. Застосунок інтелектуального аналізу та контролю особистих фінансів. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (магістерській) роботі розроблено застосунок, який допомагає здійснювати контроль за особистими фінансами, а також на підставі грошових операцій виконує аналітику поточного фінансового стану за допомогою елементів інтелектуального аналізу даних.

Ключові слова: веб-застосунок, електронний контроль фінансів, дані, інтелектуальний аналіз даних, React.js, JavaScript.

83 с., 37 рис., 4 дод, 60 джерел.

Yemelianova A. Application`s development of intellectual analysis and control of personal finances. Specialty 122 «Computer Science», Programme «Computer data processing technologies». Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

In the qualification (master's) work, an application was developed that helps to control personal finances. And based on monetary transactions an analysis of the current financial situation is performed using elements of intelligent data analysis.

Keywords: web application, electronic financial control, data, intelligent data analysis, React.js, JavaScript.

83 pages, 37 figures, 4 appendixes., 60 sources.

ЗМІСТ

ВСТУП

Гроші відіграють майже не першочергову роль у житті людини, тому відстежування власних фінансів є надзвичайно важливим і актуальним питанням у сучасному світі. Знання того, скільки грошей має людина, які витрати вона повинна зробити та скільки грошей очікувати, допомагає планувати своє життя та забезпечувати фінансову стабільність.

Відсутність контролю над особистими фінансами може призвести до низки проблем, таких як накопичення боргів, витрат, що перебільшують дохід, неспроможність забезпечити себе необхідними товарами та послугами, нестабільність фінансів тощо.

Додатки для контролю за фінансами можуть допомогти зрозуміти, куди йдуть гроші, де можна зекономити та як краще розпланувати свій бюджет. Вони дозволяють стежити за курсами валют, відстежувати заборгованості та кредитні картки, вести облік витрат і доходів, а також забезпечують єдиний доступ до фінансової інформації.

Онлайн-додатки для управління фінансами надають допомогу з аналізом власних фінансів та виявляють тенденції в споживанні. Більшість додатків дозволяють створювати категорії витрат, що дозволяє отримувати більш детальну інформацію про витрати та зберегти гроші на несуттєві речі. Оскільки більшість операцій виконується автоматично, без необхідності вносити кожну транзакцію вручну, веб-застосунки для контролю за фінансами також можуть допомогти зберегти час.

Із переваг використання веб-застосунків для контролю і аналізу власних фінансів можна виділити наступні аспекти:

- Веб-додатки можна використовувати з будь-якого пристрою, який має доступ до Інтернету.
- Більшість веб-застосунків забезпечують користувачів простим, інтуїтивно зрозумілим і зручним інтерфейсом, що спрощує ведення

обліку витрат та доходів. А також вони можуть містити корисні функції, такі як графіки та діаграми, що допомагають візуалізувати фінанси.

- Веб-додатки можуть автоматично сортувати та класифікувати витрати, що дозволяє значно зменшити час, необхідний для ведення обліку.
- Дані у веб-додатках зберігаються в хмарі або можуть зберігатись у резервних копіях. Це означає, що вони залишаться безпечними, навіть якщо дані втратяться чи пошкодяться або пристрій вийде з ладу чи втратиться.
- Багато веб-застосунків забезпечують аналітику витрат, що дозволяє зрозуміти, де і як витрачаються кошти і допомагає приймати рішення про раціональне використання власних коштів.

Метою роботи є розробка застосунку, який полегшує ведення та контроль особистих фінансів із елементами інтелектуального аналізу даних на основі мови програмування JavaScript та її популярної і широко використовуваної інтерфейсної бібліотеки React.js.

Об’єктом дослідження є процес розробки додатку із реалізацією методів інтелектуального аналізу фінансів.

Предметом дослідження є створення веб-застосунку інтелектуального аналізу та контролю особистих фінансів.

Для здійснення зазначеної мети служать наступні задачі:

1. Огляд питання і розгляд основних понять, що стосуються дослідження.
2. Аналіз додатків із схожим функціоналом (або додатків аналогів).
3. Створення і дослідження функціоналу додатку.
4. Визначення методів для виконання інтелектуального аналізу даних.
5. Вибір інструментів розробки.
6. Програмна реалізація застосунку.

Наукова новизна полягає у застосуванні методів аналізу даних для підсумування, прогнозування фінансових тенденцій і рекомендацій на основі

фінансової історії, а також розробці інтуїтивного та зручного інтерфейсу, який допомагає користувачам легко контролювати свої фінанси і аналізувати їх.

Основні положення і результати магістерської роботи були представлені у наступних публікаціях:

- «Вісник Хмельницького національного університету. Технічні науки №1» від 12 липня 2023 року. Стаття на тему: «Інформаційна система ведення реєстру клієнтів банку», с. 94-99;
- «Вісник студентського наукового товариства Донецького національного університету імені Василя Стуса. Том 2» від 1 грудня 2023 року. Стаття на тему: «Проєктування застосунку інтелектуального аналізу та контролю особистих фінансів», с. 193-197.

Кваліфікаційна (магістерська) робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Загальний обсяг кваліфікаційної роботи становить 83 сторінки. Обсяг основної частини роботи складає 70 сторінок.

РОЗДІЛ 1
ОГЛЯД І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття фінансової грамотності та важливість контролю за власними фінансами

Фінансова грамотність відноситься до знань, розуміння та навичок, пов'язаних з управлінням грошима, фінансами та економічною діяльністю. Це включає у себе розуміння основних фінансових понять, вміння планувати бюджет, керувати особистими фінансами, приймати розумні фінансові рішення та розуміти ризики та можливості, пов'язані зі споживанням, заощадженнями та інвестиціями [1].

Основні поняття, пов'язані з фінансовою грамотністю, включають [2, 3]:

- Бюджетування – вміння створювати бюджети, розподіляти гроші на різні категорії витрат та керувати витратами в межах призначеного ліміту.
- Закономірності витрат – розуміння, як розподіляються витрати, включаючи фіксові витрати (наприклад, квартплата, комунальні послуги), змінні витрати (наприклад, їжа, розваги) та непередбачувані витрати (наприклад, непередбачені ремонти або медичні витрати).
- Кредити та позики – розуміння процесу отримання кредиту, враховуючи відсоткові ставки, погашення боргу та ризики пов'язані з позичанням грошей.
- Інвестиції – розуміння основних принципів інвестування, включаючи ризики, дохідність та диверсифікацію портфеля інвестицій.
- Пенсійне забезпечення – розуміння систем пенсійного забезпечення, включаючи різні типи пенсійних планів та важливість розуміння пенсійних перспектив.

- Податки – розуміння основних принципів оподаткування, включаючи податки на дохід, податки на нерухомість та інші податки, що стосуються особистих фінансів.

Фінансова грамотність дозволяє людині зрозуміти, як працюють фінансові системи, банківські послуги, страхування, інвестиції та інші аспекти особистих фінансів. Вона також надає особі необхідні знання для ефективного планування свого майбутнього, накопичення коштів на особисті цілі, забезпечення фінансової безпеки та уникнення фінансових проблем.

Також варто зазначити те, що фінансова грамотність є важливою навичкою, оскільки допомагає людям забезпечити фінансову стабільність, здійснювати раціональне використання своїх ресурсів, максимізувати свій фінансовий потенціал і забезпечувати краще майбутнє для себе та своєї родини.

Одними із головних причин слідкувати за власними фінансами є [2]:

- Безпека і уникнення боргів – уникнення ризику втрати грошей, крадіжок, шахрайства, боргів, зменшення ризику погіршення фінансової ситуації та інших фінансових проблем. Свідоме планування витрат і розумне управління грошима допомагають уникнути непотрібних позик і кредитних зобов'язань, які можуть стати причиною фінансових проблем.
- Планування майбутнього – контроль фінансів дозволяє людині розробити план на майбутнє, визначити свої фінансові цілі та розробити стратегію, яка дозволить їй досягти цих цілей. Це може включати покупку житла, погашення позик, створення резервного фонду, інвестування в освіту або пенсійне забезпечення. Контроль над фінансами допомагає крок за кроком рухатися до цих цілей.
- Дисципліна – допомога у самодисципліні відносно своїх витрат та у визначенні пріоритетів у своєму бюджеті.

- Економія – допомога у зниженні несуттєвих витрат та збільшенні можливостей для інвестування і заощадження. Контроль за власними фінансами відкриває можливості для ефективного використання грошей через інвестиції. Заощадження і інвестування в прибуткові активи допомагає збільшити капітал та створити додаткові джерела доходу на майбутнє.
- Фінансова свобода – можливість здійснювати свої фінансові цілі, отримувати більшу свободу вибору та забезпечувати свою майбутню фінансову стабільність.
- Емоційний комфорт – контроль за власними фінансами допомагає зменшити стрес, пов'язаний з фінансовими турботами. Знання про свою фінансову ситуацію та уміння управляти грошима забезпечують більшу впевненість і спокій.

Контроль за власними фінансами є основою фінансового благополуччя і добробуту. Він допомагає створити стабільну фінансову основу, забезпечити майбутню безпеку і досягнути фінансових цілей.

1.2 Мобільний додаток як сучасна система для обліку фінансів

Мобільний веб-додаток – це програмне забезпечення, яке розроблене для використання на мобільних пристроях, таких як смартфони чи планшети. Він доступний через мобільний браузер або може бути встановлений на пристрій з використанням магазину додатків (наприклад, App Store або Google Play).

Мобільні веб-додатки можуть бути розроблені з використанням веб-технологій, таких як HTML, CSS і JavaScript. Вони можуть мати доступ до різних функцій пристрою, таких як камера, GPS, геолокація, датчики руху та інші. Це дозволяє створювати потужні та інтерактивні додатки зі зручним інтерфейсом для користувачів.

Основні переваги мобільних веб-додатків включають [6]:

1. Кросплатформеність. Мобільні веб-додатки можуть працювати на різних платформах, таких як iOS і Android, з використанням однакового коду. Це дозволяє економити час і зусилля при розробці для різних платформ.
2. Доступ через веб-браузер. Мобільні веб-додатки запускаються через веб-браузер на мобільному пристрої, що дозволяє користувачам отримувати доступ до них без необхідності встановлення окремої програми з магазину додатків.
3. Оновлення без необхідності встановлення. Оновлення мобільних веб-додатків можуть бути виконані безпосередньо на сервері, що дозволяє користувачам отримувати оновлення без необхідності встановлення нової версії додатку.
4. Адаптивний дизайн. Мобільні веб-додатки можуть бути розроблені з використанням адаптивного дизайну, що дозволяє їм пристосовуватися до різних розмірів екранів та пристроїв, забезпечуючи оптимальне відображення та взаємодію з користувачем.
5. Більша доступність. Випливаючи із попереднього пункту про адаптивність, можна зробити наступний висновок про те, що мобільні веб-додатки можуть бути доступні з будь-якого мобільного пристрою з доступом до Інтернету, що робить їх доступними для більшої аудиторії користувачів.
6. Зручне розповсюдження. Мобільні веб-додатки можуть бути поширені через посилання або через магазини додатків, що дозволяє швидко і легко розповсюджувати додатки серед користувачів.
7. Офлайн-режим. Деякі мобільні веб-додатки можуть підтримувати режим роботи без доступу до Інтернету, що дозволяє користувачам використовувати їх функціонал, навіть якщо вони не мають підключення до мережі.

8. Менші вимоги до пам'яті. Мобільні веб-додатки зазвичай потребують менше пам'яті на пристрої, порівняно зі звичайними мобільними додатками, що дозволяє зберігати більше даних і програм на пристрої.

Однак, мобільні веб-додатки також мають свої обмеження, такі як обмежена доступність до функцій пристрою та можливість обмеженого доступу до Інтернету в разі відсутності підключення.

1.3 Переваги та недоліки використання додатків інтелектуального аналізу в управлінні фінансами

Можливості додатків, що допомагають вести облік власних фінансів, насправді, безмежні. Усе залежить від креативності і вмінь тих, хто займається їхньою розробкою. Але все таки, аналізуючи купу фінансових трекерів, можна виділити низку основних спільних рис, що також мають відношення до переваг:

1. Запис всіх доходів і витрат. Можна вводити дані вручну або імпортувати їх з банківських рахунків та кредитних карт.
2. Категоризація витрат за різними категоріями, наприклад, їжа, транспорт, розваги тощо. Додаток також може дозволяти ставити мітки до транзакцій для легшого пошуку і фільтрації.
3. Створення місячних або тижневих бюджетів і встановлення лімітів витрат для кожної категорії. Це зручно для відстежування прогресу і отримання сповіщень про наближення межі бюджету.
4. Надання детальних звітів і графіків, що відображають фінансову активність. Можна аналізувати витрати за категоріями, переглядати загальний стан фінансів та відстежувати зміни з часом.
5. Нагадування про регулярні платежі, строкові платежі, бюджетні межі та інші фінансові події.
6. Захист даних. Мобільні додатки для контролю за фінансами зазвичай мають захист даних у вигляді встановлення паролю для входу у додаток, шифрування даних тощо.

7. Деякі додатки дозволяють синхронізувати дані між різними пристроями, що дозволяє використовувати застосунок на різних пристроях і мати доступ до оновленої інформації.

Хоча додатки відстеження фінансів можуть бути корисними для контролю та управління особистими фінансами, вони також мають деякі недоліки, про які варто знати.

Першим важливим пунктом є безпека даних. Використання додатків для відстеження фінансів може потребувати надання особистих фінансових даних, таких як банківські реквізити, паролі, номери карток тощо. Це може створити потенційний ризик для безпеки даних, особливо якщо додаток не має високого рівня шифрування або захисту.

Далі можна виділити залежність від Інтернету. Більшість додатків відстеження фінансів потребують з'єднання з Інтернетом для синхронізації даних або отримання оновлень. Це може створювати нестабільність у використанні додатка, якщо з'єднання з мережею недоступне або має обмежений обсяг трафіку.

Важливим також є питання приватності, оскільки деякі додатки можуть збирати та використовувати дані користувачів для рекламних цілей або передавати їх третім сторонам. Це може порушувати конфіденційність користувача і викликати питання щодо захисту особистої інформації.

Ще можна виділити обмежені можливості безкоштовних версій. Багато додатків мають безоплатні версії, але вони можуть бути обмежені за функціональністю або кількістю операцій, які можна здійснити. Щоб отримати повний функціонал, користувачам найчастіше доводитися купувати платну версію.

Також у деяких випадках, особливо якщо додаток працює на кількох платформах (наприклад, на смартфоні і комп'ютері), можуть виникати проблеми з синхронізацією даних або оновленнями. Це може призвести до втрати даних або незручностей у використанні додатка.

Останнє, що можна виділити із недоліків це невідповідність банківським стандартам. Деякі додатки можуть не повністю відповідати стандартам безпеки та інших правил, встановлених банками чи фінансовими установами, що може вплинути на безпеку фінансових операцій і застосування додатку для критичних фінансових потреб.

1.4 Огляд існуючих систем управління фінансами та інструментів аналізу

На сьогодні є безліч веб та мобільних додатків для обліку особистих фінансів [7]. Розглянемо декілька із них.

1. Monefy [8] (рис. 1.1).

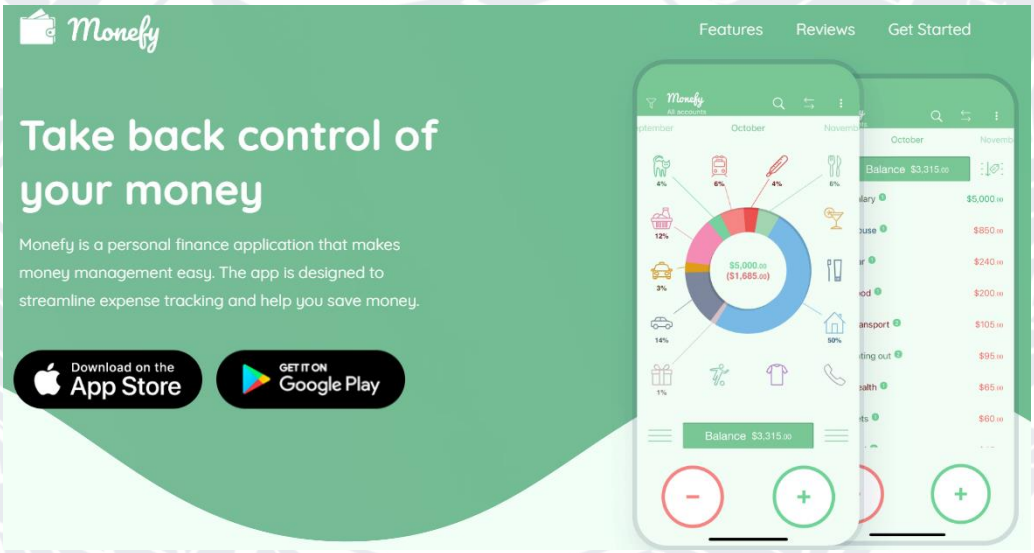

Рис. 1.1. Офіційна сторінка додатку Monefy

До переваг додатка можна віднести те, що він простий у використанні, має зручний інтерфейс і можливість синхронізації із іншими пристроями. Застосунок надає можливість відображення інформації про фінанси користувача у вигляді графіків та діаграм, що дозволяє зрозуміти, як розподіляються витрати на різні категорії та на що витрачається найбільше грошей. Також у додатка присутня можливість експорту даних, тобто користувач може експортувати свої фінансові дані з додатку у форматі CSV.

Із недоліків можна виділити наявність реклами та обмежені можливості. Додаток має обмежену кількість функцій порівняно з іншими додатками для відстеження фінансів. Також безкоштовна версія застосунку містить не усі необхідні функції. Наприклад, синхронізація даних з іншими пристроями, доступна лише у платній версії додатку. Мінусом є також відсутність можливості зберігати фотографії чеків. Monefy не дозволяє користувачам зберігати фотографії чеків, що може бути незручним для тих, хто хоче зберігати документальне підтвердження своїх витрат.

2. Wallet [9] (рис. 1.2).

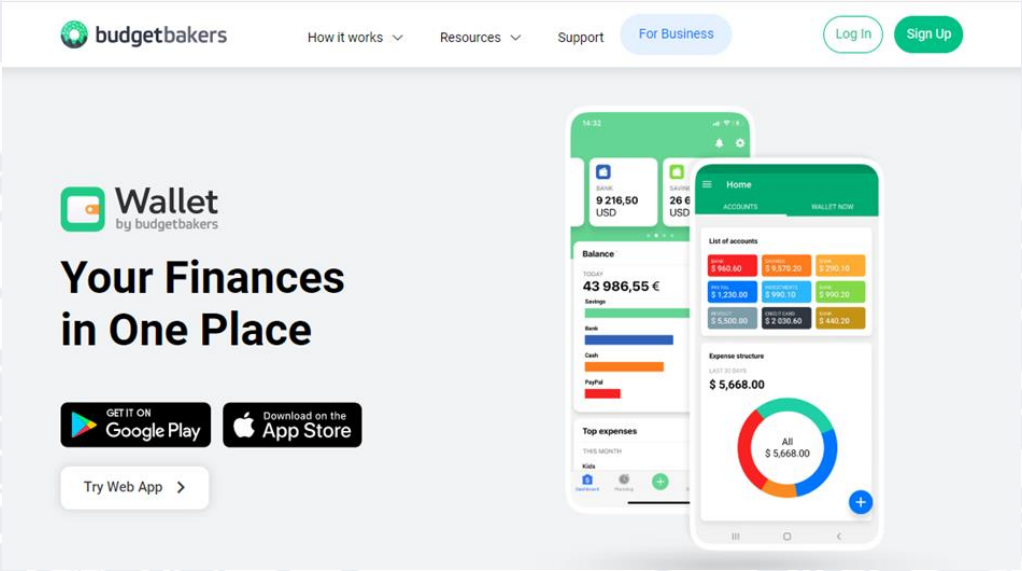

Рис. 1.2. Офіційна сторінка додатку Wallet

До плюсів додатку можна віднести: зручний та інтуїтивно зрозумілий інтерфейс, різноманітні можливості ведення обліку та імпорту даних і захист даних. Також застосунок має додаткові функції, такі як створення бюджету, графіки доходів та витрат, що дозволяє користувачам легко відстежувати свої фінансові потоки.

До мінусів можна віднести те, що безкоштовна версія додатку Wallet має обмеження щодо кількості доданих транзакцій та категорій. Додаток не має можливості автоматичного імпорту даних з банківських рахунків. Також

відсутні підтримки кількох користувачів, автоматичне оновлення курсу валют, можливість додати фотографію чека, а також примітки чи коментарі до транзакцій.

3. Zenmoney [10] (рис. 1.3).

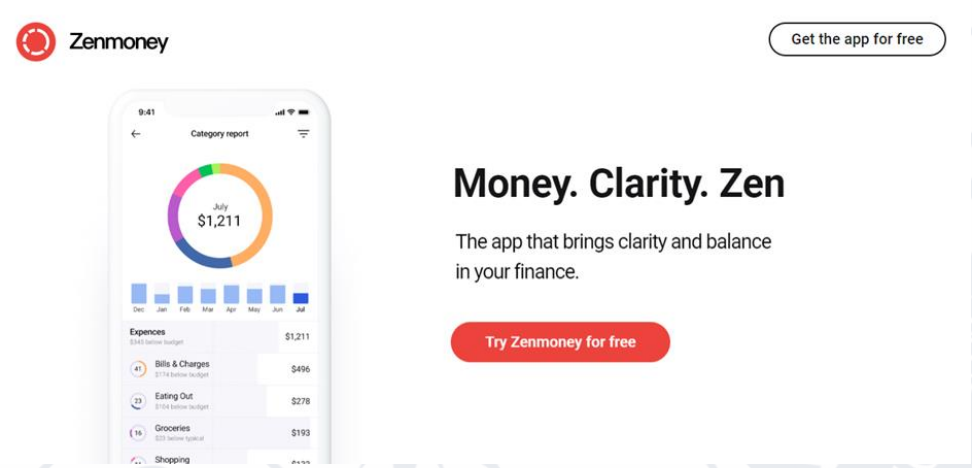

Рис. 1.3. Офіційна сторінка додатку Zenmoney

Сильними сторонами додатка є його інтеграція з більш ніж 30 банками, підтримка кількох валют, синхронізація між пристроями, можливість вести облік боргів та позик, що дуже зручно для тих, хто має регулярні платежі. А також можливість створювати користувацькі категорії.

Слабкими сторонами Zenmoney є те, що безкоштовна версія містить дуже обмежений функціонал і налаштування сповіщень. Також відсутні можливості додавання фотографій чека, прогнозування витрат та імпорту файлів у певному форматі.

4. 1Money [11] (рис. 1.4).

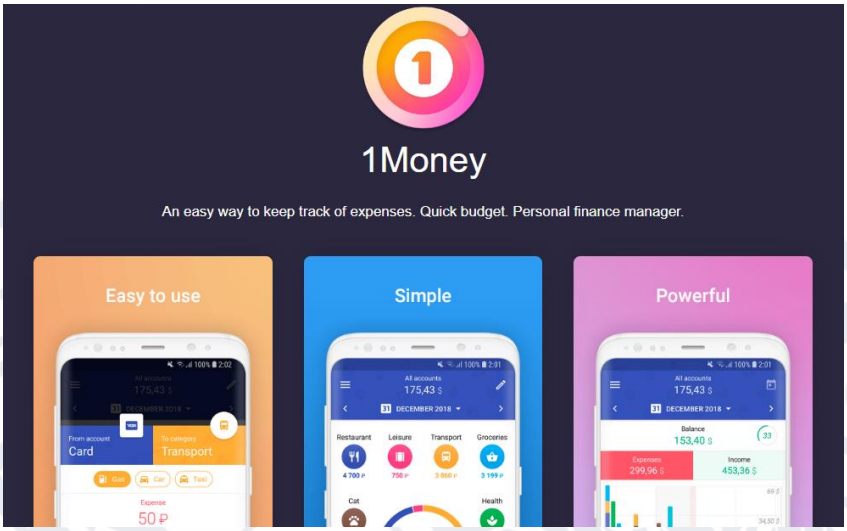

Рис. 1.4. Офіційна сторінка додатку 1Money

Із переваг додатку можна виділити його інтуїтивно зрозумілий і простий інтерфейс, можливість створення різних банківських рахунків, підтримку імпорту даних та просте планування бюджету.

До недоліків можна віднести обмежений функціонал безкоштовної версії, особливо аналізу фінансових даних, відсутність деяких функцій, які присутні в інших додатках для відстеження фінансів. А також відсутність автоматичного імпорту і синхронізації в режимі реального часу. Важливими мінусами є обмежена підтримка мов і суттєва відмінність між iOS-версією та Android-версією. Перша, в свою чергу, має значно більший функціонал.

Отже, розглянувши на теорії і практиці та проаналізувавши додатки із схожим функціоналом, постало завдання створити власний застосунок, який вирішуватиме проблеми із простим, ненав’язливим, мінімалістичним інтерфейсом та спрощеною і зрозумілою навігацією всередині додатку. Також який міститиме такий набір функцій та варіацій візуалізації даних, що зможе задовільнити усі базові потреби користувача навіть у обмеженій, безкоштовній версії додатку.

Висновок до розділу 1

У першому розділі було виконано огляд питання, а саме розкрито тему важливості контролю за власними фінансами. Розглянуто поняття фінансової грамотності та досліджено зв’язок між максимально результативним відстеженням грошових потоків та мобільних фінансових трекерів. Також проаналізовано існуючі системи управління фінансами із схожим функціоналом, що є важливим для подальшого розкриття теми у наступних розділах кваліфікаційної роботи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ЗАСТОСУНКУ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

2.1 Функціональні можливості розроблюваного веб-застосунку

Веб-застосунок для контролю за особистими фінансами повинен надавати користувачу зручні інструменти для відстеження своїх доходів та витрат, аналізу своєї фінансової ситуації та планування своїх витрат на майбутнє.

Реалізація веб-додатку відбуватиметься за принципом «Mobile first», тобто першочергово створюватиметься мобільна веб-версія, після повного завершення якої буде можлива розробка під екрани планшетів та комп'ютерів.

Веб-застосунок на початкових етапах реалізації буде позиціонуватись як MVP (Minimal Viable Product – мінімально життєздатний продукт) [12] і включатиме наступні функції (рис. 2.1):

1. Додавання доходів та витрат – користувач може вводити свої доходи та витрати в додаток, щоб мати повний огляд своєї фінансової ситуації.
2. Категоризація доходів та витрат – користувач може розподіляти свої доходи та витрати за категоріями (наприклад, їжа, транспорт, розваги, оплата кредиту тощо), для доступнішого аналізу того, куди ідуть гроші.
3. Аналіз фінансів – додаток може надавати користувачеві статистику про його фінанси (наприклад, середній дохід за місяць, середні витрати на різні категорії, загальні витрати за рік тощо), щоб допомогти йому зрозуміти свої фінансові потреби та вирішити, як краще розподіляти свої гроші.
4. Графіки та діаграми – додаток може відображати інформацію про фінанси користувача у вигляді графіків та діаграм, щоб зрозуміти, як розподіляються його витрати на різні категорії та на що він витрачає найбільше грошей. Це покращує сприйняття інформації через її візуальне подання.

5. Можливість експорту даних – користувач може експортувати свої фінансові дані з додатку у форматі, який йому зручний (наприклад, CSV, Excel тощо), щоб зберегти свої дані та аналізувати їх у власному розумінні.
6. Додаткові функції.

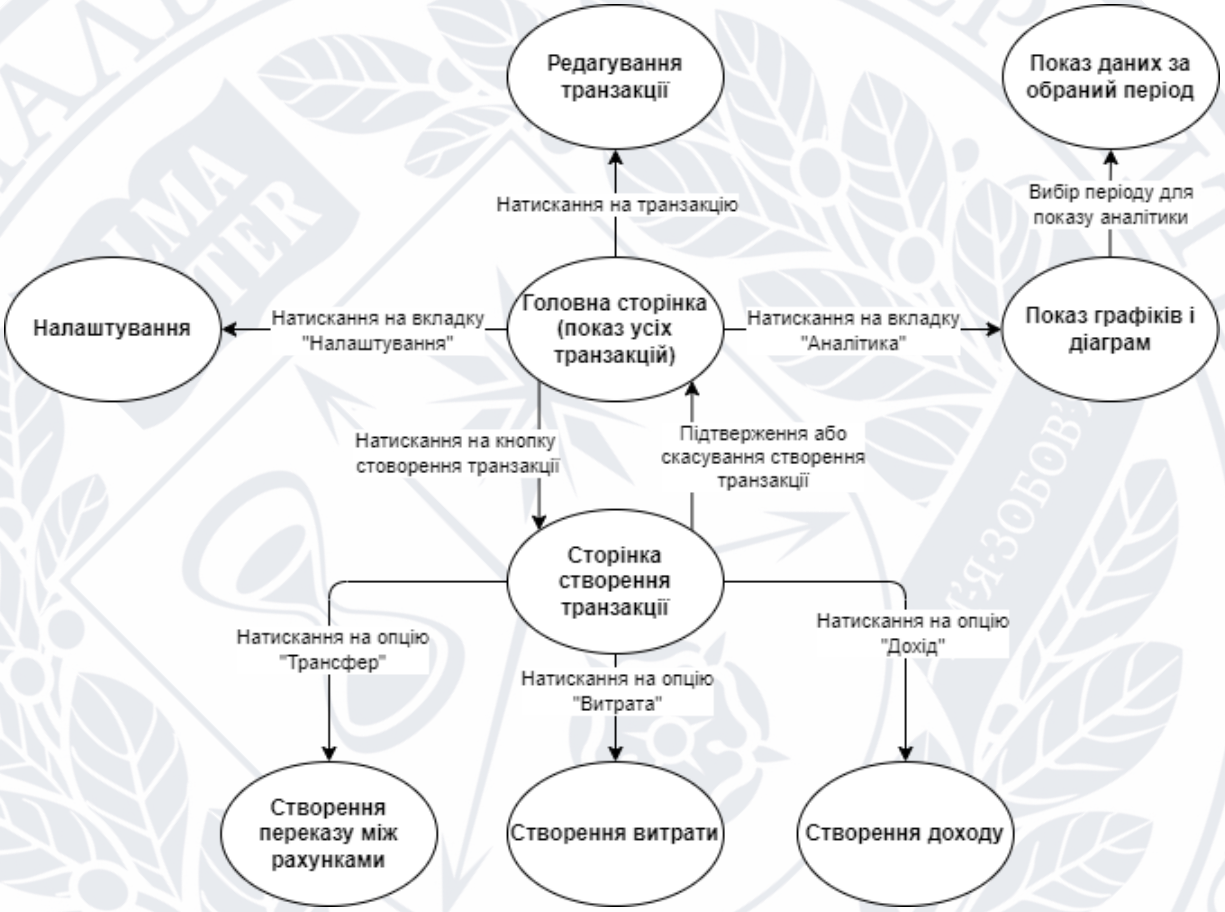

Рис. 2.1. Основні функціональні можливості застосунку

2.2 Проектування інтелектуального аналізу фінансових даних

Суть інтелектуального аналізу даних полягає у обробці різноманітної інформації значних обсягів для виявлення моделей, неочевидних закономірностей і тенденцій, що допомагають приймати рішення. Іншими словами, це процес виділення з даних неструктурованої і неявної інформації та представлення її у вигляді, придатному для подальшого використання [13]. Зазвичай важко виявити результативні висновки при традиційному перегляді даних, тому що зв’язки можуть бути складними або ж через надмірний обсяг

даних. Створення моделі інтелектуального аналізу даних є частиною більшого процесу, який включає в себе все: від формулювання запитань про дані до побудови моделі для відповідей на ці запитання та розгортання моделі у робочому середовищі [14].

Інтелектуальний аналіз даних іноді також розглядають як синонім до аналізу даних. Однак найчастіше його розглядають як специфічний аспект аналізу даних, що автоматизує аналіз великих масивів даних і виявляє інформацію, яку неможливо виявити іншими методами. Далі ця інформація може бути використана в обробці даних та інших аналітичних додатках [15].

Механізм аналізу даних включає кілька кроків і методів, які допомагають отримати інсайти з великого обсягу даних. Загальний опис механізму даних зображений на рис. 2.2.

Рис. 2.2. Механізм аналізу даних

Розглянемо більш детально кожен крок процесу інтелектуального аналізу даних [16-18].

Спочатку необхідно зібрати дані з різних джерел, таких як бази даних, файли, API тощо. Цей крок може включати збір структурованих даних (наприклад, числа, таблиці) та/або неструктурованих даних (текст,

зображення, відео). Далі дані потрібно очистити та підготувати для подальшого аналізу. Це включає видалення некоректних або відсутніх значень, виправлення помилок, нормалізацію даних тощо. Для кращого розуміння даних важливо візуалізувати їх у вигляді графіків, діаграм або інших візуальних елементів. Це допомагає виявити патерни, тренди та взаємозв'язки між даними.

Для отримання інсайтів з даних застосовуються різні аналітичні методи, такі як статистичний аналіз, класифікація, кластеризація, прогнозування, асоціативний аналіз тощо. Вони допомагають розуміти залежності, знаходити важливі змінні та зробити прогнози.

Після аналізу даних слід проаналізувати результати та зробити висновки. Це може включати інтерпретацію виявлених патернів, пояснення знайдених залежностей та прийняття рішень на основі отриманих інсайтів. Остаточні результати аналізу даних можуть бути використані для прийняття рішень, планування дій, вдосконалення процесів, прогнозування майбутніх подій та впровадження стратегій у сфері фінансів або будь-якій іншій галузі.

Однак, варто зазначити, що розглянутий вище механізм аналізу даних є загальною процедурою і може варіюватися в залежності від конкретних вимог, методів аналізу та використовуваних інструментів.

Фінансовий аналіз особистих фінансів – це процес оцінки і вивчення фінансового стану та діяльності конкретної особи або домогосподарства. Метою цього аналізу є розуміння того, як ефективно управляти грошима, як вони витрачаються та як можна поліпшити фінансовий стан. Дослідження фінансового стану особистих фінансів може включати в себе використання різних методів та інструментів, які допомагають збирати, аналізувати та розуміти фінансову інформацію.

До загальних методів дослідження фінансового стану можна віднести:

- Бюджетування та відслідковування витрат, що включають у себе розробку детального бюджету, в якому визначаються планові доходи та

витрати. А також використання спеціальних мобільних або веб-додатків для створення та відстеження бюджету.

- Аналіз витрат та доходів шляхом групування витрат за категоріями для отримання уявлення, на що саме витрачаються гроші і створенням дохідної структури через розгляд різних джерел доходів та їх внесок у загальний дохід.
- Створення і аналіз фінансової звітності, куди можна віднести створення балансу (розгляд власності та зобов'язань для визначення фінансового стану) і створення звіту про доходи та витрати (для аналізу фінансової продуктивності та здатності до заощадження).
- Оцінка інвестицій та доходів із активів, а саме: визначення ризику і доходності інвестицій та аналіз доходу з активів для визначення, як ефективно працюють різні активи.
- Розрахунок коефіцієнтів фінансового забезпечення – оцінки здатності забезпечити фінансові потреби за рахунок наявних ресурсів.
- Аналіз та планування доходів шляхом розробки стратегій збільшення доходів і розгляду можливостей покращення фінансової продуктивності через нові джерела доходів.
- Визначення фінансових цілей через постановку і створення стратегій для їх досягнення.
- Психологічний аналіз фінансів через вивчення психології витрат – розуміння та управління емоційними аспектами фінансових рішень. А також через аналіз психологічних факторів заощадження – визначення впливу психологічних факторів на звички заощадження.

Використання цих методів може допомогти створювати повні, точні та аналітичні звіти щодо фінансового стану особистих фінансів та розробляти стратегії для його покращення.

Особливо зручно досліджувати свій фінансовий стан за допомогою фінансових трекерів та програм для контролю фінансів, які можуть

використовувати різні формули та показники для аналізу фінансового стану. Розглянемо декілька загальних формул, які можуть застосовуватися у таких програмах:

1. Загальний дохід – сума всіх отриманих доходів за певний період. Це може включати заробітну плату, проценти від інвестицій, дивіденди та інші джерела доходу.
2. Загальні витрати – усі витрати, здійснені за певний період, такі як витрати на їжу, житло, транспорт, розваги, платежі за кредити тощо.
3. Чистий прибуток – розраховується як різниця між загальним доходом та загальними витратами. Він вказує на суму грошей, яка залишається після оплати всіх витрат:

$$\text{Чистий прибуток} = \text{Загальний дохід} - \text{Загальні витрати} \quad (2.1)$$

4. Зміна стану рахунку – відображення змін стану рахунку відносно часу. Допомагає відслідковувати збільшення або зменшення фінансового стану:

$$\text{Стан рахунку} = \text{Початковий баланс} + \text{Чистий прибуток}, \quad (2.2)$$

де Початковий баланс – це стартовий стан рахунку в певний часовий момент.

Чистий прибуток – різниця між загальним доходом і загальними витратами (формула 1.1) протягом періоду часу, який аналізується.

5. Співвідношення витрат до доходів – визначення відношення витрат до загального доходу і вказування на те, скільки відсотків доходу витрачається:

$$\text{Співвідношення витрат до доходів} = \frac{\text{Витрати}}{\text{Дохід}} \times 100\% \quad (2.3)$$

6. Відсоток збережень – відсоток чистого прибутку, який виділяється на збереження або інвестування:

$$\text{Відсоток збережень} = \frac{\text{Чистий прибуток}}{\text{Загальний дохід}} \times 100\% \quad (2.4)$$

7. Відсоток витрат на категорію (доля витрат) – цей показник визначає відсоток загальних витрат, які спрямовуються на оплату певної категорії (оплата комунальних послуг, продуктів, проїзду, тощо) в порівнянні з загальними витратами:

$$\text{Доля витрат} = \frac{\text{Витрати на категорію}}{\text{Загальні витрати}} \times 100\% \quad (2.5)$$

8. Рентабельність – відсоток прибутку відносно загального доходу:

$$\text{Рентабельність} = \frac{\text{Чистий прибуток}}{\text{Загальний дохід}} \times 100\% \quad (2.6)$$

9. Середній чек – визначення середньої вартості кожної транзакції.

$$\text{Середній чек} = \frac{\text{Загальний дохід}}{\text{Кількість транзакцій}} \quad (2.7)$$

10. Доля доходу від різних джерел – підсумування надходжень із джерел, з яких отримується дохід, такий як заробітна плата, інвестиції, пасивний дохід тощо.
11. Динаміка доходів у часі – зміна доходу протягом певного періоду.
12. Динаміка витрат у часі – зміна витрат протягом певного періоду.
13. Динаміка збережень – зміни збережень протягом часу.

2.3 Вибір технологій та інструментів для розробки застосунку

При створенні застосунку інтелектуального аналізу та контролю особистих фінансів використовуються сучасні та широко застосовувані технології розробки, які обирає більшість розробників веб-застосунків через свою зручність, доступність, швидкість та продуктивність. Основними технологіями і базовими блоками, на яких побудований додаток, є середовище розробки Visual Studio Code, мова програмування JavaScript і її інтерфейсна бібліотека React.js, а також бібліотека для візуалізації результатів Chart.js, що з кожним днем все більше набуває своєї популярності.

2.3.1 Середовище розробки Visual Studio Code

Visual Studio Code (або VS Code) [19] – це безкоштовний, відкритий редактор вихідного коду, який розроблений компанією Microsoft на основі Electron Framework для операційних систем Windows, Linux та macOS і призначений для роботи з різноманітними мовами програмування та технологіями. Visual Studio Code став дуже популярним серед розробників завдяки своїй широкій функціональності, активній спільноті та можливості налаштовувати його під конкретні потреби розробників.

Згідно з опитуванням 2023 року, проведеним Stack Overflow, Visual Studio Code є найпопулярнішим середовищем розробки у світі [20, 21] (рис. 2.3).

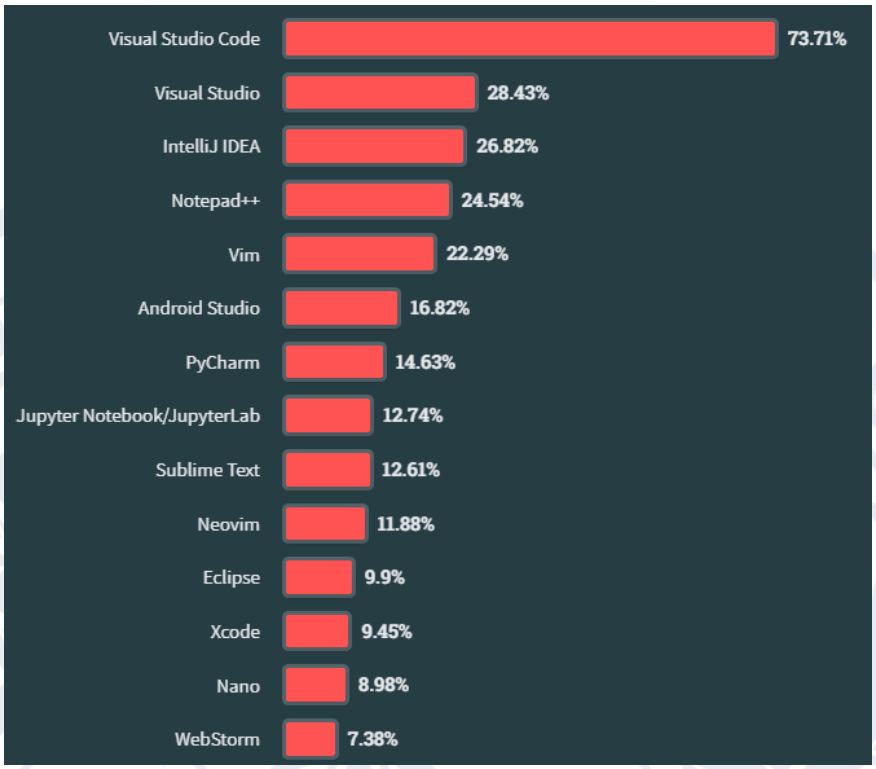

Рис. 2.3. Найпопулярніші мови програмування у світі за 2023 рік згідно з опитуванням StackOverflow [20]

І хоча VS Code часто класифікують як редактор вихідного коду, його можна використовувати як повноцінне інтегроване середовище розробки (IDE) [22]. VS Code надає потужні інструменти для розробки програмного забезпечення, підтримує велику кількість розширень та розширюється за допомогою плагінів, що дозволяє налаштовувати редактор під свої потреби. У даного редактору коду можна виділити ряд наступних ключових особливостей [23-25]:

1. Підтримка багатьох мов програмування – Visual Studio Code пропонує чудову підтримку широкого діапазону мов програмування, включаючи JavaScript, C++, C#, TypeScript, Pug, PHP, Python, XML, F#, DockerFile, CoffeeScript, Java, HandleBars, R, Objective-C, PowerShell, Visual Basic, Markdown, JSON, тощо. Він містить такі функції, як підсвічування синтаксису, автозавершення коду та інструменти для певної мови, що

робить його чудовим вибором для розробників, які працюють із кількома мовами.

2. Компактність та продуктивність – VS Code є дуже легким і швидким редактором, що споживає мало системних ресурсів. Він запускається швидко і працює ефективно навіть на менш потужних комп'ютерах.
3. Інтегрований термінал – VS Code має вбудований термінал, що дає можливість виконувати команди і сценарії безпосередньо в редакторі, що, в свою чергу, спрощує розробку та відладку.
4. Intelli-Sense (або автодоповнення коду) – те, що може виявити, якщо якийсь фрагмент коду залишився незавершеним. Крім того, загальний синтаксис змінних і оголошення змінних створюються автоматично. Наприклад, якщо в програмі використовується певна змінна, і користувач забув її оголосити, intelli-sense оголосить її для користувача. Дана особливість значно спрощує написання коду і допомагає уникнути помилок.
5. Розширюваність – Visual Studio Code пропонує багато функцій, які роблять його потужним редактором коду. Деякі з ключових функцій включають інтеграцію з Git, SVN або Mercurial для відстеження і керування змінами в кодовій базі. Також існує безліч інструментів налагодження та розширення, які додають підтримку для конкретних мов, фреймворків та інструментів розробки, тим самим дозволяючи ефективно і зручно налаштувати робочий процес. Також VS Code має широкі можливості настроювання робочого процесу, а саме налаштовування інтерфейсу і комбінацій клавіш на свій смак.
6. Спільна робота та обмін даними – редактор підтримує спільну роботу над проектами через різні сервіси для обміну даними та інтеграції з хмарними платформами. Наприклад, за допомогою Live Share можна з легкістю спільно працювати над кодом разом із іншими розробниками в режимі реального часу, навіть якщо вони знаходяться в іншій частині світу.

7. Велика спільнота розробників – у Visual Studio Code є велика спільнота розробників, які створюють і підтримують розширення та плагіни, які додають нові функції до редактора.

Незважаючи на безліч переваг, Visual Studio Code також має деякі недоліки та обмеження [23, 24]:

1. Важкий для початківців – Visual Studio Code може бути важкуватим для початківців, які вперше знайомляться з програмуванням і можуть не потребувати всіх функцій, які він пропонує. Щоб навчитися ефективно користуватися редактором, може знадобитися деякий час.
2. Вимагає додаткової конфігурації – інколи може здаватися, що VS Code вимагає більшої конфігурації, ніж інші редактори коду. Це може бути недоліком для розробників, які віддають перевагу більш оптимізованому налаштуванню.
3. Специфічність мов програмування – обмеженість або відсутність підтримки деяких специфічних або старих мов програмування може становити проблему для розробників, які працюють з менш популярними мовами.
4. Споживання ресурсів – незважаючи на легкість під час запуску, VS Code може стати ресурсномістким, особливо з кількома встановленими розширеннями. Занадто велика кількість встановлених розширень може призвести до сповільнення роботи редактора. Також для великих проектів із великою кількістю файлів VS Code може вимагати значного обсягу оперативної пам'яті, що може стати проблемою для розробників із старими або менш потужними комп’ютерами.
5. Залежність від розширень сторонніх розробників – якщо розробник забере своє розширення з репозиторію або припинить його підтримку, це може створити проблеми для користувачів, які залежать від цього розширення.

Але попри недоліки, Visual Studio Code все ще залишається одним із найпопулярніших інструментів для розробки програмного забезпечення завдяки своїй широкій функціональності, активній спільноті та безкоштовній ліцензії.

2.3.2 Мова програмування JavaScript та бібліотека React.js

JavaScript (далі JS) [26] – це легка, інтерпретована або своєчасно скомпільована мова програмування з першокласними функціями. Хоча дана мова програмування найбільш відома як мова сценаріїв для веб-сторінок, багато середовищ без браузерів також використовують її, наприклад Node.js, Apache CouchDB і Adobe Acrobat. Часто люди плутають JavaScript із мовою програмування Java — JavaScript не є «інтерпретованою Java». І «Java», і «JavaScript» є товарними знаками або зареєстрованими товарними знаками Oracle у США та інших країнах. Однак ці дві мови програмування мають дуже різний синтаксис, семантику та використання [26].

JavaScript [26] – це багатопарадигмальна, однопотокова, динамічна мова програмування на основі прототипу, яка підтримує об’єктно-орієнтований, імперативний і декларативний стилі (наприклад, функціональне програмування).

На відміну від більшості мов програмування, мова JS не має поняття введення або виведення. Вона призначена для роботи як мова сценаріїв у хост-середовищі, яке має забезпечити механізми для спілкування із зовнішнім світом. Найпоширенішим хост-середовищем є браузер [27].

JavaScript існує вже майже три десятиліття і її універсальність, здатність обслуговувати Front-end та Back-end розробку, зробила її основою інструментів більшості розробників [28].

Згідно з опитуванням, проведеним Stack Overflow у 2023 році, JavaScript на даний момент також є найпопулярнішою мовою у світі [29] (рис. 2.4).

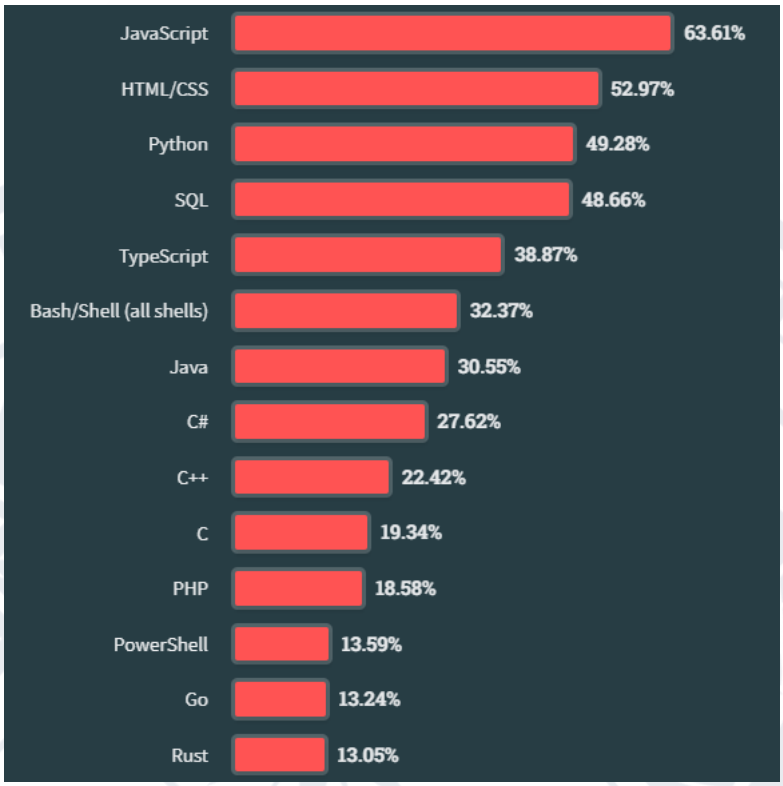

Рис. 2.4. Найпопулярніші мови програмування у світі за 2023 рік згідно з опитуванням StackOverflow [29]

JavaScript має широке застосування у багатьох сферах розробки, зокрема [27]:

- Веб-розробка: додавання інтерактивності і поведінки до статичних сайтів, для чого в 1995 році і було винайдено JavaScript.
- Веб-програми: із розвитком технологій, браузери настільки вдосконалилися, що для створення надійних веб-програм потрібна була мова програмування. Наприклад, увесь детальний перегляд карти Google Maps можливий лише завдяки JS, який використовує інтерфейси прикладного програмування (API), які забезпечують додаткову потужність коду.
- Серверні програми: за допомогою Node.js JavaScript пройшла шлях від клієнтської частини до серверної, де Node.js є найпотужнішим на стороні сервера.

- Ігри: JavaScript також допомагає створювати ігри. Поєднання JS і HTML5 робить JavaScript популярною і у розробці ігор. За допомогою бібліотеки EaseJS можна реалізувати різноманітні рішення для роботи із насиченою графікою.
- Розумні годинники: JavaScript використовується у всіх можливих пристроях і програмах. Бібліотека PebbleJS, яка має застосування в програмах для розумних годинників, допомагає налаштовувати додатки, яким для роботи необхідний доступ до Інтернету.
- Мистецтво: художники та дизайнери можуть створювати все, що завгодно, за допомогою JS, щоб малювати на HTML5 полотні (Canvas), а також для покращення звуку за допомогою бібліотеки p5.js.
- Машинне навчання: бібліотеку JS ml5.js можна використовувати у веб-розробці для машинного навчання.
- Мобільні програми: особливості JavaScript роблять її потужним інструментом для створення мобільних програм. Використовуючи фреймворк React Native, можна створювати мобільні програми для різних операційних систем.

Принцип роботи JavaScript коду полягає у наступному: він або вбудовується безпосередньо у веб-сторінку, або посилається на нього через окремий файл формату .js. Коли користувач відвідує веб-сторінку, його браузер запускає сценарій разом із кодом HTML і CSS, створюючи функціональну сторінку, що відображається у вкладці браузера. Сценарій завантажується на машини користувачів, де і проходить процес обробки. Це відрізняється від серверної мови, у якій сервер обробляє сценарій перед тим, як надсилати його в браузер. Зустрічаючи блок JS коду, веб-браузер поступово оброблює його зверху донизу. Оскільки він дуже чутливий до порядку, необхідно посилатись на об’єкти чи змінні в блоці перед тим, як змінювати їх. Наявність змінних без значень призведе до невизначеної помилки [30].

Перевагами у використанні мови програмування JavaScript є наступні аспекти [30]:

1. Простота – проста структура спрощує вивчення та впровадження JS. Також вона працює швидше, ніж деякі інші мови. Помилки легко виявляти та виправляти.
2. Швидкість – JavaScript виконує сценарії безпосередньо у веб-браузері без попереднього підключення до сервера чи компілятора. Крім того, більшість основних браузерів дозволяють JS компілювати код під час виконання програми.
3. Універсальність – JavaScript сумісний з іншими мовами програмування, такими як PHP, Perl і Java. Це також робить data science та машинне навчання доступними для розробників.
4. Популярність – у вільному доступі безліч ресурсів і форумів, які допоможуть новачкам із обмеженими технічними навичками та знаннями у JavaScript.
5. Навантаження на сервер – ще одна перевага роботи на стороні клієнта полягає в тому, що JS зменшує кількість запитів, які надсилаються на сервер. Перевірку даних можна виконати через веб-браузер, а оновлення стосуються лише певних розділів веб-сторінки.
6. Оновлення – команда розробників JavaScript і ECMA International постійно оновлюють і створюють нові фреймворки та бібліотеки, забезпечуючи їх актуальність у галузі.

Як і будь-яка інша мова програмування, JavaScript має обмеження, які необхідно враховувати. Нижче наведено деякі недоліки, з якими можна зіштовхнутись, працюючи із JS [27, 30]:

1. Сумісність браузера – різні веб-браузери по-різному інтерпретують код JavaScript, що спричиняє неузгодженість. Тому для коректної роботи програмного коду слід тестувати свій JS сценарій у всіх популярних веб-браузерах, включаючи їхні старіші версії.

2. Ризики у безпеці – код JavaScript, який виконується на стороні клієнта, вразливий до використання безвідповідальними користувачами. Наприклад, JavaScript можна використовувати для отримання даних за допомогою AJAX або шляхом маніпулювання тегами, які завантажують дані. Атаки даного типу називаються атаками міжсайтового сценарію. Вони вводять JS код, який не є частиною сайту, у браузер користувача, одержуючи потрібну інформацію.
3. Відладка і перевірка типів – хоча деякі HTML редактори підтримують відладку, але все ж вони менш ефективні, ніж інші редактори. Оскільки веб-браузери не показують жодних попереджень про помилки, знайти проблему інколи може бути складно. Також оскільки JS слабко типізована мова, то може бути важко знайти помилки, пов’язані із неправильними типами.
4. Складність – щоб оволодіти даною мовою сценаріїв, програмісти повинні мати досконалі знання про всі концепції програмування, основні мовні, клієнтські та серверні об’єкти, інакше їм буде важко писати розширені сценарії за допомогою JavaScript.

JavaScript є однією з найбільш важливих мов програмування для сучасної веб-розробки і має безліч застосувань у різних областях інформаційних технологій. Вивчення і розуміння JS може бути корисним для будь-якого розробника, який бажає працювати у веб-індустрії або створювати мобільні додатки.

ReactJS (React, React.js) [31] – це JavaScript бібліотека, розроблена компанією Meta для створення динамічних та інтерактивних додатків і кращого дизайну UI/UX для веб- і мобільних додатків [32]. Це інтерфейсова бібліотека на основі компонентів, яка відповідає лише за рівень перегляду архітектури контролера перегляду моделі (Model View Controller або MVC). React використовується для створення модульних користувацьких інтерфейсів і сприяє розробці повторно використовуваних компонентів інтерфейсу

користувача, які відображають динамічні дані [33]. Також React відомий створенням односторінкових додатків (SPA) і мобільних додатків.

React не є фреймворком – він навіть не є ексклюзивним для Інтернету. Він використовується з іншими бібліотеками для відтворення в певних середовищах. Наприклад, React Native можна використовувати для створення мобільних додатків [34].

Основною метою React є мінімізація помилок, які виникають, коли розробники створюють інтерфейси користувача. Це робиться за допомогою компонентів – самодостатніх, логічних фрагментів коду, які описують частину інтерфейсу користувача. Ці компоненти можна об'єднати разом, щоб створити повноцінний інтерфейс користувача. React абстрагує більшу частину роботи рендерингу, зосереджуючи увагу на дизайні інтерфейсу користувача [34].

Бібліотека вперше з'явилася у травні 2013 року і зараз є однією з найбільш часто використовуваних інтерфейсних бібліотек для веб-розробки [35].

Згідно з опитуванням, проведеним Stack Overflow у 2023 році, React посідає друге місце після Node.js у рейтингу найпопулярніших веб-фреймворків та технологій [36] (рис. 2.5).

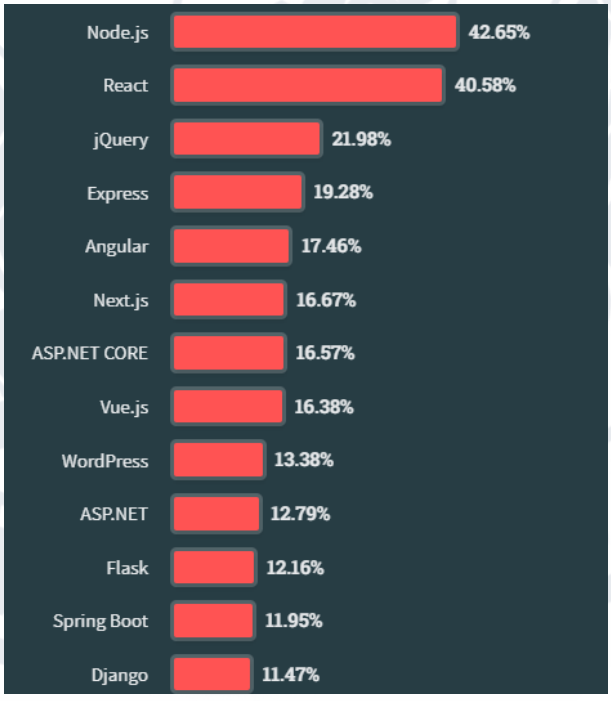

Рис. 2.5. Найпопулярніші веб-фреймворки та технології у світі за 2023 рік згідно з опитуванням StackOverflow [36]

ReactJS використовує декларативну парадигму, яка дозволяє програмам бути ефективними та гнучкими. Він ефективно оновлює та відтворює потрібний компонент у міру зміни даних і створює прості подання для кожного стану в програмі. Декларативне подання полегшує налагодження та передбачуваність коду. Кожен компонент програми React відповідає за рендеринг окремого фрагмента HTML-коду, який можна використовувати декілька разів. Можливість вкладати одні компоненти у інші дозволяє створювати складні програми із простих «будівельних блоків» [33].

Особливостями React є [34]:

- JSX (розширення синтаксису JavaScript)
- Віртуальний DOM (Virtual DOM)
- Односпрямований потік даних
- Продуктивність
- Розширення
- Умовні твердження
- Компоненти
- Простота

JSX (JavaScript XML – розширення синтаксису JavaScript) – це комбінація HTML і JavaScript, що дозволяє вставляти об’єкти JavaScript в елементи HTML. Даний синтаксис обробляється у викликах JavaScript React Framework. Він розширює ES6, щоб текст, подібний до HTML, міг співіснувати з React кодом JavaScript. JSX не підтримується браузерами, тому компілятор Babel транскомпілює JSX код у JS код [31, 37].

Віртуальний DOM (Virtual Document Object Model) – це представлення об’єкта DOM у вигляді полегшеної копії. Зазвичай фреймворки JavaScript оновлюють весь DOM одночасно, що робить веб-програму дуже повільною. Для швидкої маніпуляції із DOM деревом React використовує віртуальний DOM (Virtual DOM) – точну копію справжнього DOM. Щоразу, коли у веб-

програмі відбуваються будь-які зміни, весь інтерфейс користувача повторно відображається у віртуальному представленні DOM. Потім відбувається перевірка різниці між попереднім представленням DOM і новим DOM. Після цього справжній DOM оновлює лише ту частину, яка нещодавно змінювалася, а все інше залишається без змін. Даний процес робить програму швидшою та не витрачає пам’ять і називається терміном reconciliation [32, 37, 38].

Односпрямований потік даних (або одностороннє зв’язування даних) – дані в React рухаються лише в одному напрямку – зверху вниз, тобто від батьківських компонентів до дочірніх. Властивості у дочірньому компоненті не можуть повертати дані до свого батьківського компоненту, але можуть мати зв’язок із ними для зміни станів відповідно до наданих вхідних даних. Перевагами одностороннього зв’язування даних є те, що відбувається кращий контроль у всій програмі. Якщо потік даних йде в іншому напрямку, то це вимагає додаткових функцій і це все через те, що компоненти мають бути незмінними, а дані в них не можуть бути змінені. Даний принцип робить код більш гнучким, що призводить до підвищення ефективності [32, 37].

Продуктивність – за допомогою віртуального DOM, реальний DOM виконується в пам’яті. Тому можна створювати окремі компоненти, завдяки чому DOM працює плавніше, швидше і оновлюється без зайвих втрат з боку продуктивності [29].

Розширення – React має багато розширень, які можна використовувати для створення повноцінних програм інтерфейсу користувача. Він підтримує розробку мобільних додатків і забезпечує рендеринг на стороні сервера. React добре розширюється за допомогою Flux, Redux, React Native тощо, що допомагає створювати гарні інтерфейси користувача [32].

Умовні твердження – JSX дозволяє писати умовні оператори. Дані в браузері відображаються відповідно до умов, наданих усередині JSX [32] (рис. 2.6).

```
if (isLoggedIn) {
   <p> Hello { username }! </p>
}
else {
   <p> Hello unknown user! </p>
}
```

Рис. 2.6. Синтаксис умовного рендерингу

Компоненти – React ділить веб-сторінку на компоненти. Кожен компонент є частиною дизайну інтерфейсу користувача, який має власну логіку та дизайн. Таким чином, логіка компонентів, написана на JavaScript, полегшує і пришвидшує роботу, а також може використовуватися повторно, одночасно допомагаючи підтримувати код під час роботи над великими проектами [32].

Простота – як неодноразово зазначалось вище, React заснований на компонентах, що робить код придатним для повторного використання. React використовує JSX, який є комбінацією HTML і JavaScript, що робить код легким для написання, розуміння і налагодження [32, 37].

Одними із ключових переваг React є [35, 39, 40]:

1. Virtual DOM – підвищення продуктивності завдяки reconciliation.
2. SEO-friendly – традиційні фреймворки JavaScript мають проблеми з SEO. Пошукові системи, як правило, мають проблеми з читанням програм, які містять багато JavaScript коду. React долає цю проблему, допомагаючи розробникам легко орієнтуватися в різних пошукових системах. Все через те, що програми React.js можуть запускатись на сервері, а віртуальний DOM відображатиметься та повертатиметься до браузера як звичайна веб-сторінка.
3. Легке створення динамічних додатків – створювати динамічну веб-програму тільки за допомогою HTML було складно, оскільки це вимагає складного кодування, але React вирішив цю проблему та спростив її. Він забезпечує менше кодування та надає більше функціональних можливостей за допомогою JSX.

4. Легке тестування коду – React пропонує область, де розробник може тестувати та налагоджувати свій код за допомогою вбудованих інструментів.
5. Велика спільнота розробників і легкість у освоєнні – React має достатню кількість документації, посібників і навчальних ресурсів. Будь-який розробник із досвідом JavaScript може легко зрозуміти та почати створювати веб-додатки за допомогою React за кілька днів.
6. Можна використовувати з іншими фреймворками.

Окрім переваг, звісно ж, необхідно згадати і про недоліки у роботі із React [39, 40]:

1. Використання JSX – даний підхід має свої переваги, але деякі члени спільноти розробників вважають JSX перешкодою, особливо для нових розробників. Розробники скаржаться на його складність у кривій навчання.
2. Неякісна документація – недолік, характерний для технологій, що постійно оновлюються. Технології React оновлюються та прискорюються настільки швидко, що бракує часу робити належну документацію. Щоб хоч якось змінити ситуацію, розробники самостійно пишуть інструкції з розвитком нових випусків та інструментів у своїх поточних проектах.
3. View part або частина перегляду – React охоплює лише рівні інтерфейсу користувача програми і нічого більше. Тому все одно потрібно обрати деякі інші технології, щоб отримати повний набір інструментів для проектної розробки. React – це бібліотека, а не фреймворк, і немає офіційних бібліотек для обробки загальних функцій інтерфейсних додатків, як наприклад маршрутизації, HTTP-запитів тощо. React не має багатьох базових інструментів «із коробки». Навіть якщо розробник використовує лише найпопулярніші та найпоширеніші бібліотеки, то може виникнути конфлікт React версій між версією бібліотеки та

версією програми React (здебільшого йдеться про застарілі програми). Якщо розробник хоче підтримувати свою програму в актуальному стані з версією React, то ймовірно може знадобитися оновлення бібліотек сторонніх розробників, які живуть власним життям. Подібно до того, як React має чудову зворотну сумісність, деякі бібліотеки (навіть дуже популярні) можуть зазнати деяких критичних змін.

4. Відсутність домовленостей щодо написання коду між розробниками – React не нав'язує жодних домовленостей щодо розробки. Знайомство нових розробників із застарілим проектом займає більше часу, оскільки дуже часто розробники JavaScript створюють React програми різними способами. Тому існує висока ймовірність того, що розробник зіткнеться зі застарілим проектом із правилами кодування, з якими він раніше не перетинався. Це змушує команду розробників витрачати час на обговорення деяких загальних правил розробки в проекті. А коли команда змінюється багато разів, це може призвести до безладного коду, оскільки всі мають різні звички та дотримуються різних умов.

Але, незважаючи на недоліки, React і досі залишається однією з найпопулярніших бібліотек для розробки веб-інтерфейсів і використовується великою кількістю компаній та розробників для створення сучасних та ефективних веб-додатків. Його активна спільнота і багато розширень роблять його потужним інструментом для веб-розробки.

2.3.3 Chart.js і react-chartjs-2 – інструменти для візуалізації результатів аналізу даних

Chart.js [41] – це популярна, безкоштовна JavaScript бібліотека із відкритим кодом для створення інтерактивних графіків та діаграм на веб-сайтах і у веб-додатках. Вона надає простий і зручний інтерфейс для візуалізації даних у вигляді різних типів графіків. Розмір бібліотеки складає лише 60 Кб, тому це досить легка і невелика JS бібліотека. Графіки мають

досить сучасний вигляд та кольорову гаму, а також підтримують анімацію під час першого відмальовування графіку [42].

Chart.js є однією із найпростіших бібліотек візуалізації для JavaScript, яка підтримує наступні вбудовані типи діаграм [41-44]:

- діаграма розсіювання;
- лінійна діаграма;
- гістограма;
- кругова або секторна діаграма;
- кругла секторна діаграма (пончикова);
- бульбашкова діаграма;
- діаграма областей;
- радарна діаграма;
- змішана діаграма, тощо.

Під час роботи із Chart.js потрібно лише вказати, де відобразити графік і вказати його тип. Після цього треба надати Chart.js дані, мітки та деякі інші налаштування, а рештою займатиметься вже сама бібліотека [44].

До особливостей бібліотеки можна віднести наступне [42, 44]:

1. Chart.js – це безкоштовна бібліотека з відкритим кодом, яку підтримує спільнота і яку можна використовувати в режимі офлайн або онлайн.
2. Надає широкий вибір різноманітних вбудованих діаграм, що дозволяє вибирати найкращий спосіб представлення даних в залежності від конкретного завдання. Усі типи діаграм можна змінювати та анімувати, а також всі діаграми підтримують адаптивність.
3. Можливість створювати користувацькі діаграми. Для збільшення функціональності можна використовувати плагіни.
4. Підтримка всіх сучасних браузерів.

5. Добре організована документація, яка містить детальну інформацію про кожну діаграму. Це полегшує розуміння та використання навіть для користувачів початківців.

React-chartjs-2 [45] – це бібліотека, яка є обгорткою (wrapper) для використання бібліотеки Chart.js в додатках, які створені з використанням бібліотеки React. Основна ідея react-chartjs-2 полягає у тому, щоб зробити процес використання Chart.js більш зручним та сумісним із екосистемою React.

Особливостями бібліотеки є [45]:

1. Зручний інтерфейс – можна вставляти графіки Chart.js прямо в компоненти React як звичайні React компоненти. Це дозволяє краще інтегрувати графіки з іншими елементами інтерфейсу.
2. Автоматичні оновлення – коли стан React компонентів змінюється, графіки, створені за допомогою react-chartjs-2, автоматично оновлюються, що спрощує роботу з динамічними даними.
3. react-chartjs-2 дозволяє налаштовувати графіки та діаграми Chart.js за допомогою параметрів та опцій, що передаються через властивості (props) компонента.
4. Підтримка усіх можливостей Chart.js.
5. Активна спільнота і підтримка.

Отже, бібліотека Chart.js дуже корисна для візуалізації даних на веб-сайтах та веб-додатках, і вона широко використовується розробниками для покращення інтерфейсу користувача та зрозуміння даних. А бібліотека react-chartjs-2 робить використання Chart.js більш дружньою для React розробників, дозволяючи легко створювати і керувати графіками та діаграмами у проектах екосистеми React.

Висновок до розділу 2

У другому розділі кваліфікаційної роботи описано функціонал веб-додатку. Розглянуто і описано механізми збору та аналізу даних, а також усіх необхідних обчислень, що використовуватимуться для аналізу даних та їхньої візуалізації на підставі фінансових операцій. Також проведений і визначений вибір інструментів та технологій, необхідних для реалізації застосунку шляхом детального огляду тенденцій розвитку, переваг, обмежень і особливостей використання.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ТА КОНТРОЛЮ ОСОБИСТИХ ФІНАНСІВ

3.1 Конфігурація проекту і налаштування додаткових залежностей

Create React App (CRA) [46] – зручне середовище для створення невеликих React проектів. Це інструмент командного рядка (command line interface, CLI), який допомагає створити новий проект на основі React з мінімальними зусиллями щодо налаштувань і конфігурації. Він розроблений командою React для полегшення старту нових проектів та спрощення роботи розробників. Create React App не опрацьовує логіку бекенду або логіку баз даних, а лише надає команди для побудови фронтенду, що дозволяє використовувати його із будь-яким бекендом [47].

До особливостей Create React App можна віднести наступне [46]:

- CRA створює проект React з базовою структурою та налаштуваннями. Можна одразу починати писати React код, не витрачаючи час на налаштування веб-пакетів та інших інструментів.
- CRA дотримується сучасних стандартів розробки веб-додатків. Він використовує Webpack для збирання та оптимізації коду, Babel для трансляції сучасного JavaScript у старіші версії, а також підтримує середовище розробки з підтримкою Hot Module Replacement (HMR) для швидкого оновлення змін під час розробки.
- CRA включає в себе підтримку тестування за допомогою бібліотеки Jest та засобів для тестування користувальницького інтерфейсу, таких як React Testing Library.
- CRA автоматично налаштовує додаток як прогресивний веб-додаток (Progressive Web App, PWA), що дозволяє йому працювати в автономному режимі та забезпечувати кращу продуктивність і взаємодію з користувачем.

- CRA автоматично оптимізує та стискає зображення, що додаються до проекту, для покращення швидкості завантаження сторінок.
- Після завершення розробки проекту, його можна легко розгорнути на хостинговому сервісі, такому як GitHub Pages, Netlify, Vercel тощо.
- CRA створює початкові сторінки, які допомагають почати розробку свого додатку.

Для створення Create React App проекту необхідно мати наступні мінімальні вимоги: Node.js більше версії 8.10 та npm версія не менше 5.6 [47]. Щоб створити новий CRA проект достатньо написати у командному рядку декілька команд, наведених на рисунку 3.1.

```
npx create-react-app finance-tracker
cd finance-tracker
npm start
```

Рис. 3.1. Створення CRA проекту

Перша команда відповідає за створення CRA проекту під назвою «finance-tracker». Наступна команда виконує перехід у директорію створеного проекту. Остання команда відповідає за запуск проекту у режимі розробки.

Кожен npm пакет і Node.js проект мають файл package.json із метаданими для проекту. Файл знаходиться в кореневому каталозі кожного Node.js пакета і з'являється після виконання команди npm init. Файл package.json містить описові та функціональні метадані про проект, наприклад назву, версію та залежності. Файл надає менеджеру пакетів npm різну інформацію, щоб допомогти ідентифікувати проект і обробити залежності [48].

Одним із важливих полів файла package.json є властивість залежностей (dependencies). Це об'єкт, який називається залежностями, і складається з усіх пакетів, які використовуються в проекті, із їхніми номерами версій. Щоразу, коли розробник встановлює будь-яку бібліотеку, необхідну для свого проекту, то її можна знайти в об'єкті залежностей [49]. Наприклад, після створення

Create React App проекту, у полі залежностей відображаються деякі наступні властивості, наведені на рисунку 3.2.

```
"dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.10.0",
    "react-scripts": "5.0.1",
  }
```

Рис. 3.2. Фрагмент залежностей CRA проекту

Також важливою властивістю є залежності розробки (devDependencies). Він складається з усіх пакетів із номером версії, які використовуються в проекті на етапі його розробки, а не в робочому чи тестовому середовищі. Щоразу, коли необхідно встановити будь-яку бібліотеку, яка потрібна лише на стадії розробки, то її можна знайти в об’єкті devDependencies [49].

Для розробки фінансового додатку, що є складовою даної кваліфікаційної роботи, залежностей CRA недостатньо, тому далі розглянемо усі пакети, необхідні для повноцінної розробки веб-застосунку:

1. Пакети JavaScript бібліотеки Redux для управління станом додатку [50].
2. Chart.js і react-chartjs-2 для побудови різноманітних графіків на піставі аналізу даних у React додатку (див. пункт 2.3.3).
3. Пакети для стилізації додатку за допомогою Material UI (MUI) [51].
4. JSON Server – імітація бекенд серверу, фіктивний API на етапі розробки [52].
5. Пакет axios для відправки запитів на сервер [53].
6. React Router для маршрутизації і переключення між сторінками додатку [54].
7. І багато інших залежностей, необхідних для розробки, таких як пакетів для перевірки синтаксису коду, генерації унікальних ключів, пакетів для оголошення змінних тощо.

Отже, повний лістинг із усіма потрібними для розробки проекту встановленими залежностями наведений у Додатку А.

3.2 Структура проекту та імітаційної бази даних

На рисунку 3.3 зображена структура проекту веб-додатку для контролю особистих фінансів.

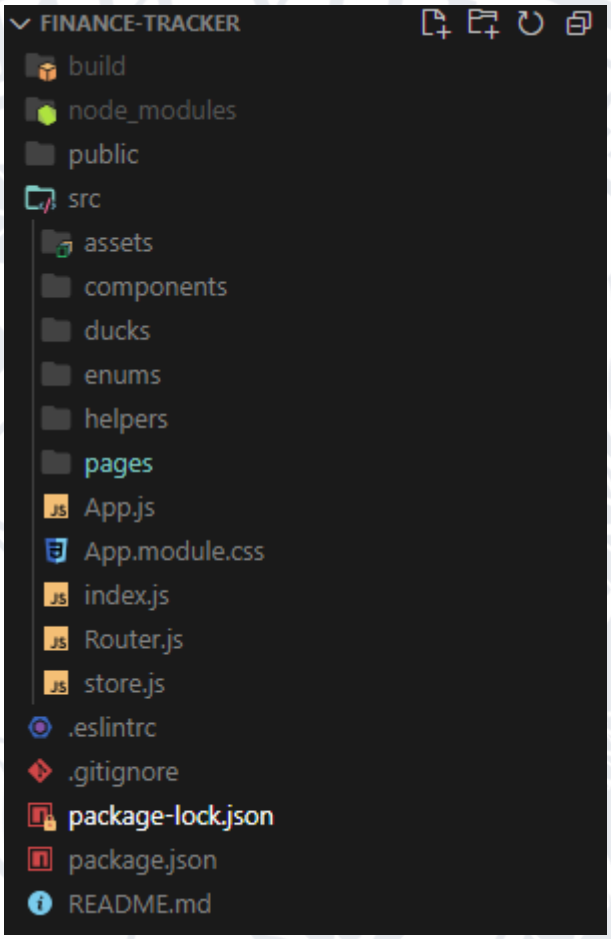

Рис. 3.3. Структура проекту розроблюваного веб-застосунку

Розглянемо детальніше структуру розроблюваного проекту.

Каталог build містить робочу збірку програми.

Каталог node_modules, який створений npm, який є способом відстеження кожного пакета, який інсталюється локально через package.json. Дана папка може розглядатись як кеш для зовнішніх модулів, від яких залежить проект. При виконанні команди npm install, модулі завантажуються з Інтернету і копіюються у каталог node_modules, де Node.js буде шукати потрібний модуль при імпорті безпосередньо в коді програми без вказання конкретного шляху. Папку node_modules можна будь-коли повністю відтворити з нуля,

перевстановивши всі залежні модулі (які мають бути вказані в папках проекту і package.json).

Папка public, яка містить в собі статичні файли, такі як index.html – шаблон сторінки, який є необхідним і який не можна видаляти, вміст якого наведений на рисунку 3.4.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-
scale=1" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <title>Finance tracker</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;
500;600&family=Roboto:wght@300;400;500;700&display=swap"
rel="stylesheet">
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this
app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Рис. 3.4. Вміст корневого файлу index.html

А також, окрім index.html у папці public містяться js файли, зображення тощо, які не потрібно обробляти збирачами проектів, такими як Webpack. Файли в цій папці копіюються та вставляються так, як вони знаходяться, безпосередньо в папку збірки. Також у цій папці міститься файл db.json, який імітує базу даних для роботи із JSON Server.

У каталозі src розміщуються всі папки і файли проекту:

- assets – каталог із зображеннями;
- components – каталог, що містить усі «будівельні блоки», з яких складається веб-застосунок;
- ducks – відповідає за роботу із Redux сховищем і містить редуктори (reducers) і дії (actions);
- enums – містить файли із константними даними, а саме масивами даних;
- helpers – містить функції, які виконують необхідні обчислення, які не стосуються відображення компонентів на сторінці;
- pages – каталог, який містить у собі папки із усіма сторінками, які є у застосунку;
- App.js, App.module.css – головні файли проекту;
- index.js – вхідна точка JavaScript проекту;
- Router.js – файл, у якому описаний механізм переходу між сторінками;
- store.js – Redux сховище.

Також можна помітити файли у безпосередньо кореневій папці проекту (root folder). Це файли, які відповідають за загальну конфігурацію проекту. Про package.json було згадано у попередніх пунктах кваліфікаційної роботи. А, наприклад, файл .gitignore повідомляє системі контролю версій Git, які файли потрібно ігнорувати під час надсилання проекту до репозиторію GitHub.

React Router [54] – стандартна бібліотека для маршрутизації в React. Вона забезпечує навігацію між представленнями різних компонентів у React додатку, дозволяє змінювати URL-адресу браузера, а також підтримує синхронізацію інтерфейсу користувача із URL-адресою [55].

Основними компонентами React Router є [55]:

- BrowserRouter – це реалізація маршрутизатора, яка використовує API історії HTML5 (pushState, replaceState та подію popstate), щоб підтримувати синхронізацію інтерфейсу користувача Із URL-адресою. Це батьківський компонент, який використовується для зберігання всіх інших компонентів.
- Routes – це новий компонент, представлений у React Router v6 та оновлений компонент. Основною перевагою Routes над Switch із старіших версій React Router є те, що маршрути вибираються на основі найкращого збігу, а не по порядку.
- Route – це умовно показаний компонент, який відображає деякий інтерфейс користувача, якщо його шлях збігається з поточною URL-адресою.
- Link – використовується для створення посилань на різні маршрути та реалізації навігації у програмі. Він працює як тег прив'язки HTML.

Реалізація маршрутизації між сторінками описується у файлі Router.js проекту (див. п. 3.1).

Для початку потрібно створити сторінки, тобто React компоненти. Усі сторінки розміщуються у папці pages проекту (див. п. 3.1). Наразі це наступні компоненти: головна сторінка (Home), сторінки створення операції (CreateOperation), показу аналітики (Analytics) і загальної інформації по поточних рахунках (Accounts). Також для обробки некоректних шляхів необхідно створити компонент, який повідомлятиме користувача про неправильну адресу (компонент ErrorBoundary).

Далі у файлі Router.js потрібно включити створені компоненти до маршутизації додатку. Виконується це наступним чином:

1. Створюються маршрути, що наведені на рисунку 3.5.

```
<Routes>
  <Route
    index element={<Home setShowBottomNav={setShowBottomNav} />}
    errorElement={<ErrorBoundary />}
  />
     ...
  <Route
    path="/operation"
    element={<CreateOperation
setShowBottomNav={setShowBottomNav} />}
    errorElement={<ErrorBoundary />}>
    <Route
      path=":transactionId"
      element={<CreateOperation />}
      errorElement={<ErrorBoundary />}
    />
  </Route>
  <Route path="/*"
    element={<ErrorBoundary setShowBottomNav={setShowBottomNav}
/>} />
</Routes>
```

Рис. 3.5. Фрагмент створення маршрутів

2. Створюється функція Router(), яка повертає компонент маршруту, що продемонстровано на рисунку 3.6.

```
const Router = ({setShowBottomNav}) => {
  return <Routes>
    ...
  </Routes>
}
```

Рис. 3.6. Фрагмент створення функції, що повертає створені маршрути

Параметр `setShowBottomNav`, що передається у функцію не має відношення до створення роутингу, а відповідає за відображення навігаційної панелі на сторінках.

3. Останнім кроком залишається підключення створеної маршутизації у додаток. Це відбувається за допомогою BrowserRouter, що показано на рисунку 3.7.

```
import { BrowserRouter } from "react-router-dom";
import App from "./App";
<BrowserRouter>
    <App />
</BrowserRouter>
```

Рис. 3.7. Фрагмент підключення маршрутизації

Як зазначалось раніше, BrowserRouter є батьківським компонентом або ж обгорткою, тому необхідно ним обгорнути компонент, який є головною точкою додатку і містить в собі усі сторінки і компоненти.

Отже, ось таким чином відбувається створення переходу між сторінками React веб-додатку за допомогою React Router v6.

JSON Server [56] – інструмент для імітації серверу на етапі розробки. Він імітує RESTful API, використовуючи JSON файл як джерело даних. Завдяки серверу JSON фронтенд розробники можуть створювати фіктивні макети API без необхідності писати складний код на стороні сервера або якщо API ще не готовий [56].

Цей макет API надсилає запити до наданої кінцевої точки. Він відповідає на запити HTTP, і це робить його ідеальним для швидкої розробки для фронтенд розробників. JSON Server також дозволяє розробникам виконувати операції CRUD і зберігає дані у файлах JSON [56]. JSON складається із пари ключ-значення, приклад якого наведений на рисунку 3.8.

```
{
  "key1": "value1",
  "key2": 20,
  "key3": ["value1", "value2"]
}
```

Рис. 3.8. Приклад JSON коду

Файл JSON може мати два формати – формат масиву та формат об’єкта з вкладеними об’єктами, що продемонстровано на рисунку 3.9.

```
// формат масиву
[
  {
    "id": 1,
    "key1": "value1",
    "key2": 10
  },
  {
    "id": 2,
    "key1": "value1",
    "key2": 20
  }
]
// формат об’єкту із вкладеними об’єктами
{
  "group": {
    "obj1": {
      "key1": "value1",
      "key2": 30
    },
    "obj2": {
      "key1": "value1",
      "key2": 40
    }
  }
}
```

Рис. 3.9. Приклад двох JSON форматів – масиву та об’єкту із об’єктами

У випадку створення веб-додатку стеження за власними фінансами JSON файл має формат об’єкту із вкладеними об’єктами, що наведено на рис. 3.10.

```
{
  "balance": {
    "Card": 5000, "Cash": 6000
  },
  "operations": [
    {
      "id": "43288356-872d-476e-a0b0-b5783f2daf01",
      "account": "Card",
      "fromAccount": null,
      "toAccount": null,
      "category": "Adjustment",
      "amount": 100,
      "type": "Income",
```

```
      "note": "",
      "date":   "Mon   Sep   25   2023   17:18:15   GMT+0300   (за
східноєвропейським літнім часом)",
      "payee": ""
    },
    ...
  ]
}
```

Рис. 3.10. Фрагмент JSON файлу розроблюваного веб-додатку

JSON файл проекту містить два об’єкти: balance, який зберігає баланс по рахункам (готівка і картка) та operations – масивом, який зберігає транзакції у вигляді об’єктів. Кожна транзакція має наступні поля:

- id – унікальний ідентифікатор транзакції;
- account – назва рахунку, з якого здійснена транзакція;
- fromAccount – назва рахунку відправника при виконанні переказу між рахунками;
- toAccount – назва рахунку отримувача при виконанні переказу між рахунками;
- category – категорія (продукти, розваги, транспорт тощо);
- amount – вартість;
- type – тип транзакції: дохід, витрата або переказ;
- note – коментар або примітка до транзакції;
- date – дата здійснення транзакції;
- payee – місце витрати (назва закладу тощо) або назва місця прибутку (зарплата, повернення боргу, подарунок тощо)

Обов’язковими полями є id, amount, type і date. Інші поля є опціональними для заповнення.

3.3 Функціонал застосунку

У веб-застосунку інтелектуального аналізу та контролю особистих фінансів можна виконувати наступні дії над транзакціями:

1. створення транзакції;

2. редагування транзакції;

3. видалення транзакції.

В залежності від типу транзакції, сторінка має декілька варіантів відображення. На рисунку 3.11 зображені три варіанти сторінки створення операції при занесенні витрати, доходу і переказу між рахунками.

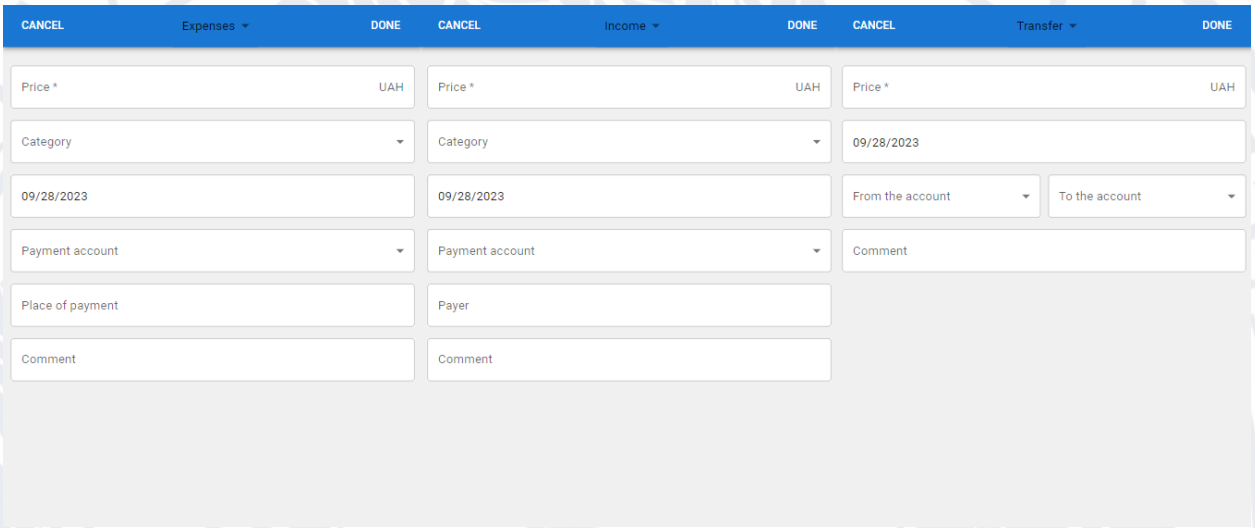

Рис. 3.11. Три варіанти сторінки створення операції в залежності від її типу

Незначні зміни у дизайні можна спостерігати між транзакціями витрат і доходів, оскільки там змінюється лише назва одного із полів вводу, а саме із «Place of payment» для витрати на «Payer» для доходу. Однак для проведення переказу між рахунками сторінка створення операції значно відрізняється від попередніх, оскільки там відсутні поля категорії та місця оплати, а поле для вибору рахунку розділене на два поля – рахунку-відправника і рахунку-отримувача.

Також є ще один варіант відображення сторінки, що стосується того моменту, коли користувач хоче внести зміни до власної транзакції (рис. 3.12).

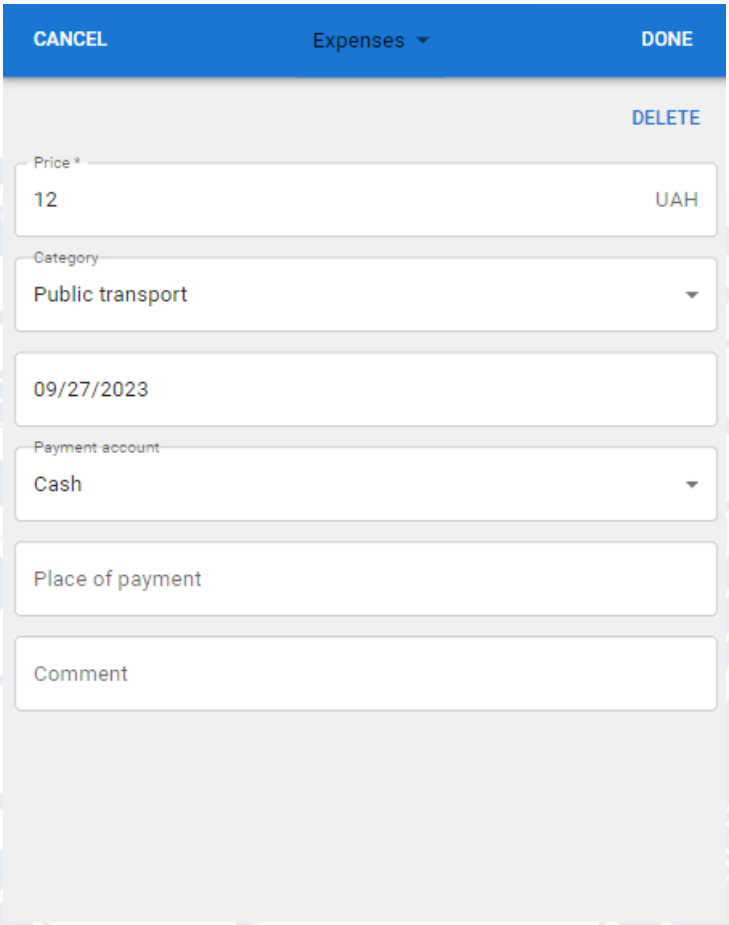

Рис. 3.12. Сторінка редагування транзакції

Як можна побачити із рисунку 3.12 вище, головною відмінністю сторінки редагування транзакції від інших є заповнені поля і наявність кнопки видалення операції.

Для збереження нової транзакції необхідно надіслати запит на сервер і змінити стан додатку за допомогою Redux.

Метод HTTP запиту POST відправляє дані на сервер, приклад застосування якого наведений на рисунку 3.13.

```
export const postOperation = (data) => {
    return {
        type: POST_TRANSACTION,
        payload: data,
    };
};
export const operationRequestError = (error) => {
    return {
        type: TRANSACTION_REQUEST_ERROR,
```

```
        payload: error,
    };
};
export const fetchPostNewTransaction = (operationData) => {
    return (dispatch) => {
        axios
            .post(`http://localhost:3001/operations`,
operationData)
            .then(res => { dispatch(postOperation(res)); })
            .catch(error => {
                console.log(error);

dispatch(operationRequestError(error.response.status));
            });
    };
};
```

Рис. 3.13. POST запит на сервер із даними про нову транзакцію

Метод HTTP запиту PUT створює новий ресурс або замінює представлення цільового ресурсу корисним навантаженням запиту, приклад застосування якого наведений на рисунку 3.14.

```
export const updateTransaction = (data) => {
    return {
        type: UPDATE_TRANSACTION,
        payload: data,
    };
};
export const fetchUpdateTransaction = (operationData) => {
    return (dispatch) => {
        axios

.put(`http://localhost:3001/operations/${operationData.id}`,
operationData)
            .then(res => { dispatch(updateTransaction(res)); })
            .catch(error => {
                console.log(error);

dispatch(operationRequestError(error.response.status));
            });
    };
};
```

Рис. 3.14. PUT запит на сервер із оновленими даними

Метод HTTP запиту DELETE видаляє вказаний ресурс, приклад застосування якого наведений на рисунку 3.15.

```
export const deleteTransaction = (data) => {
    return {
        type: DELETE_TRANSACTION,
        payload: data,
    };
};
export const fetchDeleteTransaction = (transactionId) => {
    return (dispatch) => {
        axios

.delete(`http://localhost:3001/operations/${transactionId}`)
            .then(
                dispatch(deleteTransaction(transactionId))
            )
            .catch(error => {
                console.log(error);

dispatch(operationRequestError(error.response.status));
            });
    }
}
```

Рис. 3.15. DELETE запит на сервер із вказаним ресурсом на видалення

Метод HTTP запиту GET запитує представлення зазначеного ресурсу. Запити із використанням GET слід використовувати лише для запиту даних, які не повинні включати будь-які дані. Приклад застосування GET запиту наведений на рисунку 3.16 і знаходить своє використання у додатку для відображення усіх транзакцій на домашній сторінці.

```
export const getOperations = (data) => {
    return {
        type: GET_TRANSACTIONS,
        payload: data,
    };
};
export const fetchGetTransactions = () => {
    return async (dispatch) => {
        try {
```

```
            const response = await
axios.get("http://localhost:3001/operations");
            const sortedEvents = response.data
                .slice()
                .sort(
                    (a, b) => new Date(b.date).getTime() - new
Date(a.date).getTime()
                );
            dispatch(getOperations(sortedEvents));
        }
        catch (error) {
            alert(error.message);
            console.log(error);

dispatch(operationRequestError(error.response.status));
        }
    };
};
```

Рис. 3.16. GET запит на сервер для отримання даних

Функції `fetchPostNewTransaction`, `fetchUpdateTransaction`, `fetchDeleteTransactio`, `fetchGetTransactions` наведені на рисунках 3.13-3.16 вище, відповідають за відправку або отримання даних із сервера. Також, варто зазначити, що у даних функціях відбувається взаємодія із Redux сховищем, а саме створення дій (actions) для відслідковування стану застосунку. Код реалізації редуктору (reducer), який виконує зміну стану операцій в Redux сховищі, наведений у Додатку В.

Метод HTTP запиту PATCH застосовує часткові зміни до певного ресурсу. Приклад застосування PATCH запиту наведений на рисунку 3.17 і використовується у додатку для оновлення балансу після маніпуляцій із транзакціями.

```
export const updateBalance = (data) => {
    return {
        type: UPDATE_BALANCE,
        payload: data,
    };
}
export const fetchUpdateBalance = (account, newBalance) => {
```

```
    const updateData = { [account]: newBalance };
    return (dispatch) => {
        axios
            .patch(`http://localhost:3001/balance`, updateData)
            .then(() => {
                dispatch(updateBalance({account, newBalance}));
            })
    };
};
```

Рис. 3.17. PATCH запит на сервер із частковою заміною ресурсу

Функція `fetchUpdateBalance` наведена у лістингу 3.17 вище, відповідає за відправку даних на сервер. Як і попередні функції, описані вище, дана функція взаємодіє із Redux сховищем, однак через свій окремий незалежний редуктор. Код реалізації редуктору (reducer), який виконує зміну стану балансу в Redux сховищі, наведений у Додатку Г.

Візуалізація результатів аналізу усіх транзакцій реалізовується за допомогою бібліотек Chart.js і react-chartjs-2 (див. п. 2.3.3).

У даному пункті розглянемо візуалізацію наступних аналітичних даних, розрахованих на підставі формул з пункту 2.2:

1. Стовбчаста гістограма загальних витрат по кожній категорії.
2. Кругова діаграма долі витрат по кожній категорії.
3. Діаграми областей динаміки витрат у часі.
4. Розрахунок рентабельності.

Розглянемо процес візуалізації створення стовбчастої гістограми, яка відображає загальні витрати по кожній категорії за певний проміжок часу (тиждень, місяць, рік). Для цього необхідно виконати наступні кроки:

1. Виконати фільтрацію усіх транзакцій за потрібний проміжок часу, що продемонстровано на рисунку 3.18.

```
export const filterDatesByCurrentPeriod = (data, period) => {
    const now = new Date();
    const past = new Date(now.getTime() - period * 24 * 60 * 60 *
1000);
    const start = past.getTime();
    const end = now.getTime();
    return data.filter(item => Date.parse(item.date) >= start &&
Date.parse(item.date) < end);
}
```

Рис. 3.18. Фільтрація транзакцій за потрібний проміжок часу

Функція `filterDatesByCurrentPeriod` отримує два параметри: усі транзакції (data) та потрібний період (period), який виступає числом, що позначає тиждень, місяць або рік.

Для визначення часових рамок, потрібно визначити кінцеву точку і початкову точку визначеного періоду. Для пришвидшення процесу фільтрації, крайні часові точки за допомогою Date методу getTime() перетворюємо на кількість мілісекунд для цієї дати з епохи, яка визначається як опівніч на початку 1 січня 1970 року за UTC. На виході отримуємо відфільтрований масив транзакцій за потрібний визначений проміжок часу.

2. Із отриманого відфільтрованого масиву із значеннями за визначений період, відокремити усі транзакції, що є витратами, розподілити по категоріям і підсумовувати ціни. Також прийняти до уваги можливість розподілу витрат по різним рахункам (рисунок 3.19).

```
export const filterDataToDisplay = (categories, accountType,
timePeriodTransactions) => {
    const filteredArr = [];
    if (accountType === "All") {
        for (let category of categories) {
            filteredArr.push(timePeriodTransactions
                .filter(item => item.type === "Expenses" &&
item.category === category)
                .map(elem => elem.amount)
                .reduce((accum, curr) => accum += curr, 0))
        }
```

```
    }
    else {
        for (let category of categories) {
            filteredArr.push(timePeriodTransactions
                .filter(item => item.type === "Expenses" &&
item.account === accountType && item.category === category)
                .map(elem => elem.amount)
                .reduce((accum, curr) => accum += curr, 0))
        }
    }
    return filteredArr;
}
```

Рис. 3.19. Фільтрація транзакцій за потрібний проміжок часу

Застосування імперативного підходу у парі із декларативним робить даний процес компактним та дуже простим у написанні і розумінні. У масив `filteredArr` записуються уже підраховані суми витрат по кожній категорії окремо.

3. Відобразити отриманий масив результатів у вигляді стовбчастої гістограми. Даний процес зображений на рисунку 3.20.

```
const options = {
    responsive: true,
    plugins: { legend: { display: false }, },
};
export const CategoriesExpensesBarChart = ({ filteredDataArr })
=> {
    const data = {
        labels,
        datasets: [
            {
                data: filteredDataArr,
                backgroundColor: 'rgba(25, 118, 210, 0.7)',
            }
        ],
    };
    return <Bar options={options} data={data} />
};
```

Рис. 3.20. Побудова стовбчастої гістограми

У результаті отримуємо стовбчасту гістограму, яка показує витрати по кожній категорій за вказаний проміжок часу (рис. 3.21).

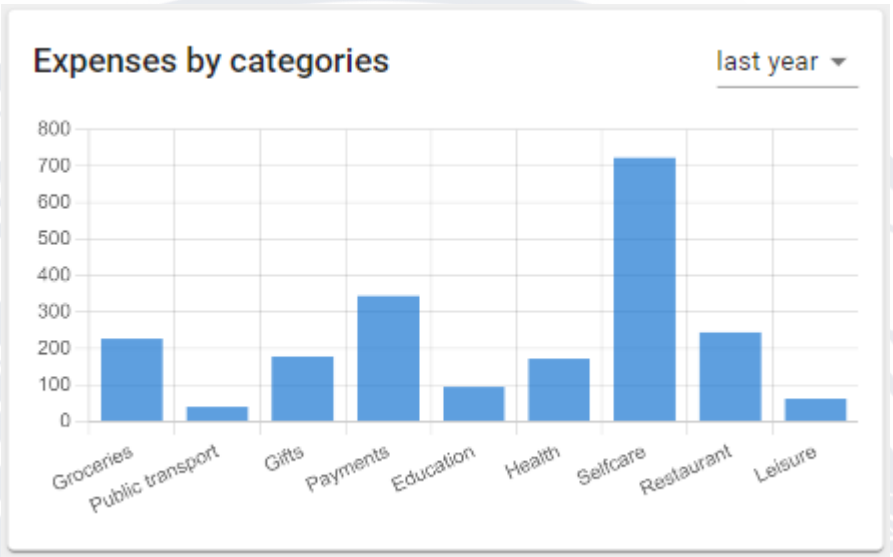

Рис. 3.21. Стовбчаста гістограма із витратами по кожній категорії

Далі розглянемо відображення даних, які були розглянуті вище при побудові стовбчастої гістограми, у вигляді кругової діаграми (Pie chart). Для побудови даної діаграми необхідно отримані дані по категоріям у вигляді грошових сум перевести у відсотки відносно загальної суми витрат, що показано на рисунку 3.22.

```
const generalSum = transactions
        .filter(item => item.type === "Expenses")
        .map(elem => elem.amount)
        .reduce((accum, curr) => accum += curr, 0);

const percentsArr = filteredDataArr && filteredDataArr.map(item
=> ((item / generalSum) * 100).toFixed(1));
```

Рис. 3.22. Розрахунок відсотку витрат по кожній категорії відносно загальних витрат

Спочатку відбувається підрахунок загальних витрат, після чого за допомогою формули розрахунку долі витрат (див. п. 2.2 формула 1.5) підраховується відсоток витрат на кожну категорію.

Відображуємо розраховані дані у вигляді кругової діаграми (рис. 3.23).

```
export const CategoriesExpensesPieChart = ({ filteredDataArr })
=> {
    ...
    const data = {
        labels,
        datasets: [
            {
                label: '%',
                data: percentsArr,
                backgroundColor: [
                    'rgba(255, 99, 132, 0.2)',
                    'rgba(54, 162, 235, 0.2)',
                    'rgba(255, 206, 86, 0.2)',
                    ...
                ],
                borderColor: [
                    'rgba(255, 99, 132, 1)',
                    'rgba(54, 162, 235, 1)',
                    'rgba(255, 206, 86, 1)',
                    ...
                ],
                borderWidth: 1,
            },
        ],
    };
    return <Pie data={data} />
}
```

Рис. 3.23. Побудова кругової діаграми

У результаті отримуємо кругову діаграму, яка показує долю витрат по кожній категорій за вказаний проміжок часу (рис. 3.24).

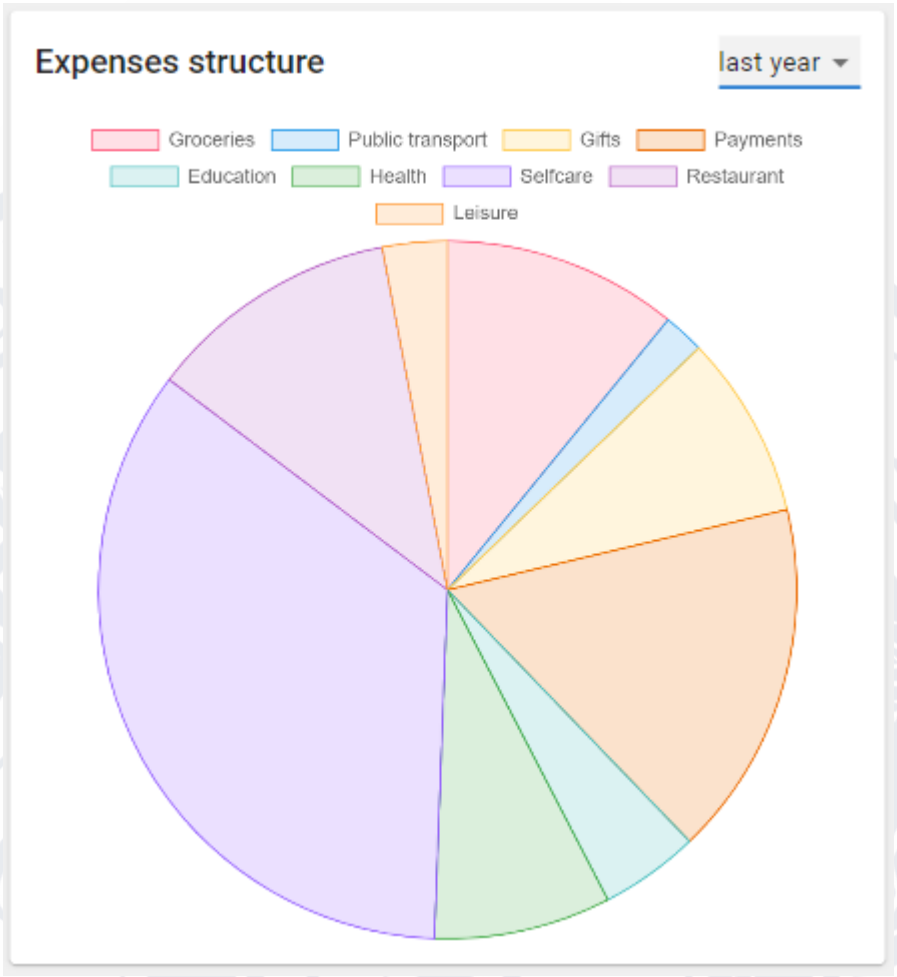

Рис. 3.24. Кругова діаграма із долею витрат по категоріям

Розглянемо створення діаграми областей, що відображає динаміку витрат у часі. Загальний алгоритм обрахунку наступний:

1. Створюємо масив із усіма датами, що входять у визначений період часу (рис. 3.25).

```
const secondIndexToSlice = timePeriodNum !== 365 ? 10 : 7;
const periodExpensesSum = [];
let datesArr = new Set();
if (account === "All") {
    timePeriodTransactions.map(item => item.type ===
transactionType
    && datesArr.add(item.date.slice(0, secondIndexToSlice)));
} else {
    timePeriodTransactions.map(item => item.type ===
transactionType && item.account === account
&& datesArr.add(item.date.slice(0, secondIndexToSlice))); }
```

Рис. 3.25. Створення масиву із датами

Дата приходить у форматі «2023-08-25 22:35:06» і змінна `secondIndexToSlice` відповідає за фільтрацію дати за місяцем (2023-08) або за днем (2023-08-25).

2. Підраховуємо суму усіх витрат по кожній даті (рис. 3.26).

```
for (let date of datesArr) {
    periodExpensesSum.push(timePeriodTransactions
        .filter(item => item.type === transactionType &&
item.date.slice(0, secondIndexToSlice) === date)
        .map(elem => elem.amount)
        .reduce((accum, curr) => accum += curr, 0)
    )
}
```

Рис. 3.26. Підрахунок витрат по кожній даті

3. Відображаємо підрахунки на графіку (рис. 3.27).

```
const options = {
    responsive: true,
    plugins: { legend: { display: false } }
};

export const TrendChart = ({ transactionType, timePeriodNum,
timePeriodTransactions, account }) => {
    ...
    const labels = datesArr;
    const data = {
        labels,
        datasets: [
            {
                fill: true,
                label: transactionType === 'General expense',
                data: periodExpensesSum,
                borderColor: 'rgb(255, 99, 132)',
                backgroundColor: 'rgba(255, 99, 132, 0.5)'
            },
        ],
    };
    return <Line options={options} data={data} />
}
```

Рис. 3.27. Відображення графіку областей

У результаті обчислень отримуємо діаграму областей, що відображає динаміку витрат у часі (рис. 3.28).

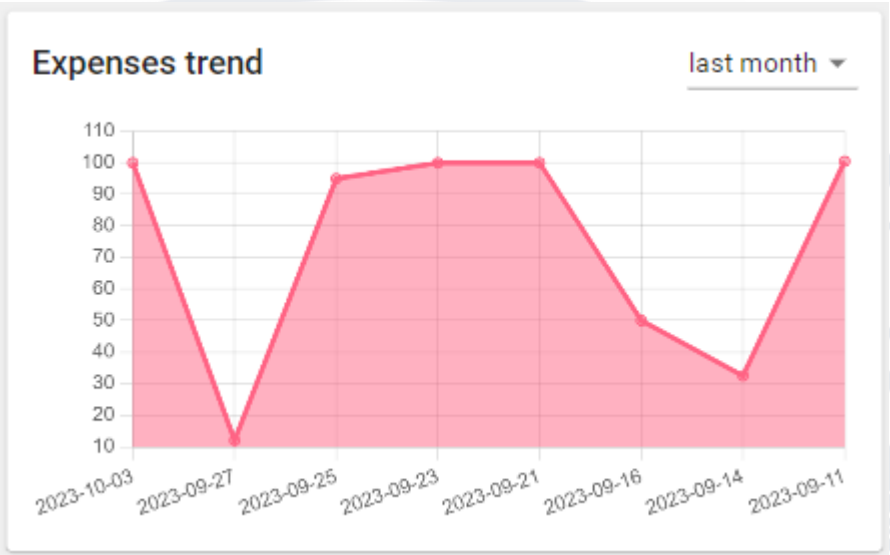

Рис. 3.28. Діаграма областей динаміки витрат у часі

І, наостанок, розглянемо обрахунок рентабельності (відсотку прибутку відносно загального доходу), що наведений на рисунку 3.29.

```
export const ProfitabilityCard = ({ account,
timePeriodTransactions }) => {
    const incomeByAccount = filterIncomeByAccount(account,
timePeriodTransactions)
        .map(item => item.amount)
        .reduce((accum, curr) => accum += curr, 0);

    const filteredExpensesByAccount = account === "All"
        ? timePeriodTransactions.filter(item => item.type ===
"Expenses")
        : timePeriodTransactions.filter(item => item.type ===
"Expenses" && item.account === account);

    const generalExpensesByAccount = filteredExpensesByAccount
        .map(item => item.amount)
        .reduce((accum, curr) => accum += curr, 0);

    const netProfit = incomeByAccount -
generalExpensesByAccount;
    const profitability = ((netProfit / incomeByAccount) *
100).toFixed(2);
```

```
    return (
        <Typography variant="h6">
            {profitability > 0 ? profitability : 0} UAH
        </Typography>
    )
}
```

Рис. 3.29. Розрахунок рентабельності

Для отримання рентабельності (див. п. 2.2 формула 1.6) необхідно розрахувати загальний дохід (`incomeByAccount`), загальні витрати (`generalExpensesByAccount`), та чистий прибуток (`netProfit`).

Результат обчислень, зображений на рисунку 3.30.

Рис. 3.30. Рентабельність за минулий місяць

3.4 Подальші перспективи розвитку веб-застосунку

Розроблюваний веб-застосунок має безліч перспектив у подальшому розвитку. У рамках кваліфікаційної роботи відбулась розробка продукту із базовим набором функцій для стабільного функціонування і демонстрації результатів виконання магістерської роботи. Необхідно лише розмістити веб-додаток на сервері (хості) та створити доменне ім’я, щоб він став доступним в мережі Інтернет.

До наступних етапів покращення і розширення функціональності веб-застосунку інтелектуального аналізу та контролю особистих фінансів можна віднести такі пункти:

1. Розширення аналітики, щоб додаток міг надавати більше аналітичних засобів для детального аналізу витрат, інвестицій та збережень.
2. Додавання функції прогнозування витрат і доходів, а саме розвиток алгоритмів прогнозування, який може допомогти користувачам

передбачати майбутні витрати і доходи на основі їхньої історії та патернів.

3. Інтеграція зі штучним інтелектом (AI) і машинним навчанням (ML), що може покращити аналітику та надавати персоналізовані рекомендації користувачам для ефективного управління фінансами.
4. Розробка версії додатку для комп'ютерів та пристроїв із більшим розширенням екрану, ніж у смартфонів.
5. Додавання можливості підключення до банківських акаунтів, кредитних карт, інвестиційних платформ та інших фінансових джерел, що може автоматизувати процес відстеження та аналізу фінансів користувача.
6. Підвищення безпеки і конфіденційності, що є надзвичайно важливими аспектами у фінансових додатках.

Також, варто зазначити, що розвиток функціоналу додатку для контролю за власними фінансами повинен базуватися на потребах та запитах користувачів, а також на нових технологіях та трендах у фінансовому секторі.

Висновок до розділу 3

У третьому розділі кваліфікаційної роботи було здійснено програмну реалізацію веб-застосунку інтелектуального аналізу та контролю особистих фінансів. Було показано процес конфігурації проекту та описано структуру проекту.

Продемонстровано ключові етапи програмної реалізації, такі як створення імітаційної бази даних, створення функціоналу, візуалізації результатів аналізу тощо.

Також чітко окреслено етапи майбутніх перспектив у розвитку даного веб-додатку.

ВИСНОВКИ

В результаті кваліфікаційної (магістерської) роботи реалізовано веб-застосунок інтелектуального аналізу та контролю особистих фінансів. Даний додаток є гарним помічником, який полегшує ведення та контроль власних фінансів та надає фінансову аналітику на підставі використання елементів інтелектуального аналізу даних.

Реалізація відбувалась у середовищі розробки Visual Studio Code із використанням мови програмування JavaScript. Було проведено дослідження багатьох різноманітних засобів та інструментів для реалізації даного веб-додатку, таких як React.js, Redux, Chart.js, React Router, Material UI, JSON Server та інших.

Перший розділ кваліфікаційної роботи був присвячений розкриттю теми важливості контролю власних фінансів. Досліджено зв’язок між максимально результативним відстеженням грошових потоків та мобільних додатків для обліку фінансів. Також відбувся аналіз існуючих систем управління фінансами із схожим функціоналом, що є фундаментом для розкриття теми у наступних розділах кваліфікаційної роботи.

У другому розділі роботи було розглянуто і описано механізми збору та аналізу даних, а також усіх необхідних обчислень, що використовуватимуться для аналізу даних та їхньої візуалізації на підставі фінансових операцій. Проведено вибір інструментів та технологій, необхідних для реалізації застосунку шляхом детального огляду тенденцій розвитку, переваг, обмежень і особливостей використання.

Третій розділ роботи присвячений програмній реалізації веб-застосунку. Продемонстровано процес конфігурації, структуру проекту та ключові етапи програмної реалізації. Також окреслено можливий розвиток і майбутні перспективи розвитку даного веб-додатку.

У ході виконання роботи було досягнуто зазначеної мети та виконано усі етапи поставленої задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Косенков С. Що таке фінансова грамотність - основи для початківців + ТОП-9 порад, з чого почати вивчення фінансової грамотності [Електронний ресурс]. Сергій Косенков. 2023. Режим доступу до ресурсу: https://biznescat.com/informatsiia/74-shcho-take-finansova-gramotnist.html. (Дата звернення: 08.05.2023)
2. Тетянич Ю. Фінансова грамотність: вчимося, поки не пізно [Електронний ресурс]. Юлія Тетянич. 2021. Режим доступу до ресурсу: https://www.ukrinform.ua/rubric-yakisne-zhyttia/3343235-finansova-gramotnist-vcimosa-poki-ne-pizno.html. (Дата звернення: 08.05.23)
3. Жученя К. Фінансова грамотність: як забезпечити комфортне життя та безбідну старість [Електронний ресурс]. Катерина Жученя. 2021. Режим доступу до ресурсу: https://happymonday.ua/finansova-gramotnist. (Дата звернення: 08.05.23)
4. Бортко О. У чому відмінність вебдодатків від мобільних додатків [Електронний ресурс]. Олександр Бортко. 2022. Режим доступу до ресурсу: https://brander.ua/blog/u-chomu-vidminnist-vebdodatkiv-vid-mobilnykh-dodatkiv. (Дата звернення: 08.05.23)
5. Мобільний та веб-додаток. У чому різниця? [Електронний ресурс]. Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/mobilnij-ta-veb-dodatok-u-chomu-rizniczya/. (Дата звернення: 08.05.23)
6. Смирнов І. Мобільний веб-додаток: який тип обрати для розробки [Електронний ресурс]. Ілля Смирнов. 2023. Режим доступу до ресурсу: https://webcase.com.ua/uk/blog/cho-takoe-mobilnoe-web-prilozhenie/. Дата звернення: (08.05.23)
7. Чорноморченко Е. Добірка додатків для контролю фінансів [Електронний ресурс]. Елеонора Чорноморченко. 2022. Режим доступу до ресурсу: https://bazilik.media/dobirka-dodatkiv-dlia-kontroliu-finansiv/. (Дата звернення: 10.05.23)

8. Monefy [Електронний ресурс]. Режим доступу до ресурсу: https://monefy.me/. (Дата звернення: 10.05.23)
9. Wallet [Електронний ресурс]. Режим доступу до ресурсу: https://budgetbakers.com/. (Дата звернення: 10.05.23)
10. Zenmoney [Електронний ресурс]. Режим доступу до ресурсу: https://zenmoney.app/. (Дата звернення: 10.05.23)
11. 1Money [Електронний ресурс]. Режим доступу до ресурсу: https://www.1moneyapp.com/. (Дата звернення: 10.05.23)
12. Що таке MVP і навіщо він мені – поради для стартаперів [Електронний ресурс]. 2019. Режим доступу до ресурсу: https://uaspectr.com/2019/08/29/shho-take-mvp-i-navishho-vin-meni-porady-dlya-startaperiv/. (Дата звернення: 13.05.23)
13. Тема 1. Інтелектуальний аналіз даних (Data Mining). Особливості технології Data Mining та її відмінності від інших методів аналізу даних [Електронний ресурс]. Режим доступу до ресурсу: https://moodle.znu.edu.ua/pluginfile.php/486112/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%201.pdf. (Дата звернення: 30.11.23)
14. Інтелектуальний аналіз даних (data mining) [Електронний ресурс]. Режим доступу до ресурсу: https://sites.google.com/a/kubg.edu.ua/integrovani-sistemi/intelektualnij-analiz-danih-data-mining. (Дата звернення: 30.11.23)
15. ЩО ТАКЕ DATA MINING (АНАЛІЗ ДАНИХ)? [Електронний ресурс]. 2022. Режим доступу до ресурсу: https://futurenow.com.ua/shho-take-data-mining-analiz-danyh/. (Дата звернення: 30.11.23)
16. Аналіз даних [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85. (Дата звернення: 13.05.23)
17. Основи статистичного аналізу даних [Електронний ресурс]. Режим доступу до ресурсу: https://www.miyklas.com.ua/p/informatica/10-

klas/modeli-i-modeliuvannia-analiz-ta-vizualizatciia-danikh-326184/modeli-i-modeliuvannia-analiz-ta-vizualizatciia-danikh-325755/re-a43672ea-4fe7-4839-8746-f03a2572fea0. (Дата звернення: 13.05.23)

18. Основи статистики та аналізу даних [Електронний ресурс]. Режим доступу до ресурсу: https://socialdata.org.ua/manual/manual4/. (Дата звернення: 14.05.23)
19. Visual Studio Code [Електронний ресурс]. Режим доступу до ресурсу: https://code.visualstudio.com/. (Дата звернення: 14.05.23)
20. Integrated development environment [Електронний ресурс]. 2023 Developer Survey. 2023. Режим доступу до ресурсу: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-integrated-development-environment. (Дата звернення: 14.05.23)
21. Top 3 Code Editors for Programmers in 2023 [Електронний ресурс]. Medium. 2023. Режим доступу до ресурсу: https://javascript.plainenglish.io/top-3-code-editors-for-programmers-in-2023-fccf42c47e52. (Дата звернення: 14.05.23)
22. Sibisi M. Modern VS Code extension development: The basics [Електронний ресурс]. Mdu Sibisi. 2023. Режим доступу до ресурсу: https://snyk.io/blog/modern-vs-code-extension-development-basics/. (Дата звернення: 14.05.23)
23. Mir M. What are the advantages and disadvantages of using Visual Studio Code or Atom? [Електронний ресурс]. Muhammad Ahmed Mir. Medium. 2023. Режим доступу до ресурсу: https://medium.com/@ssc.ahmed.926748/what-are-the-advantages-and-disadvantages-of-using-visual-studio-code-or-atom-d3132bf1af85. (Дата звернення: 14.05.23)
24. Tiushka N. Unveiling The Differences: Visual Studio VS Visual Studio Code [Електронний ресурс]. Nazar Tiushka. 2023. Режим доступу до ресурсу: https://marketsplash.com/tutorials/visual-studio/visual-studio-vs-visual-studio-code/. (Дата звернення: 14.05.23)

25. Pedamkar P. Visual Studio Code [Електронний ресурс]. Priya Pedamkar. 2023. Режим доступу до ресурсу: https://www.educba.com/what-is-visual-studio-code/. (Дата звернення: 14.05.23)
26. JavaScript [Електронний ресурс]. MDN Web Docs. 2023. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript. (Дата звернення: 14.05.23)
27. Introduction to JavaScript [Електронний ресурс]. 2023. Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-javascript/. (Дата звернення: 14.05.23)
28. O'Grady B. What is JavaScript? [Електронний ресурс]. Brian O'Grady. Режим доступу до ресурсу: https://codeinstitute.net/global/blog/what-is-javascript-and-why-should-i-learn-it/. (Дата звернення: 14.05.23)
29. Programming, scripting, and markup languages [Електронний ресурс]. 2023 Developer Survey. 2023. Режим доступу до ресурсу: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages. (Дата звернення: 14.05.23)
30. Jordana A. What Is JavaScript? A Basic Introduction to JS for Beginners [Електронний ресурс]. Jordana. 2023. Режим доступу до ресурсу: https://www.hostinger.com/tutorials/what-is-javascript. (Дата звернення: 19.09.23)
31. React The library for web and native user interfaces [Електронний ресурс]. Режим доступу до ресурсу: https://react.dev/. (Дата звернення: 19.09.23)
32. What are the features of ReactJS ? [Електронний ресурс]. 2023. Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-are-the-features-of-reactjs/. (Дата звернення: 19.09.23)
33. ReactJS Introduction [Електронний ресурс]. 2023. Режим доступу до ресурсу: https://www.geeksforgeeks.org/reactjs-introduction/. (Дата звернення: 19.09.23)
34. Getting started with React [Електронний ресурс]. MDN Web Docs. 2023. Режим доступу до ресурсу: https://developer.mozilla.org/en-

US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started. (Дата звернення: 19.09.23)

35. Deshpande C. The Best Guide to Know What Is React [Електронний ресурс]. Chinmayee Deshpande. 2023. Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs. (Дата звернення: 19.09.23)
36. Web frameworks and technologies [Електронний ресурс]. 2023 Developer Survey. 2023. Режим доступу до ресурсу: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies. (Дата звернення: 19.09.23)
37. React Features [Електронний ресурс]. Режим доступу до ресурсу: https://www.javatpoint.com/react-features. (Дата звернення: 19.09.23)
38. Verma A. The Comprehensive Guide to React's Virtual DOM [Електронний ресурс]. Ayush Verma. Medium. 2021. Режим доступу до ресурсу: https://javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a. (Дата звернення: 19.09.23)
39. Pros and Cons of ReactJS [Електронний ресурс]. Режим доступу до ресурсу: https://www.javatpoint.com/pros-and-cons-of-react. (Дата звернення: 19.09.23)
40. Goralski C. Pros and Cons of React [Електронний ресурс]. Cezary Goralski. 2022. Режим доступу до ресурсу: https://thecodest.co/blog/pros-and-cons-of-react/. (Дата звернення: 19.09.23)
41. Chart.js [Електронний ресурс]. Режим доступу до ресурсу: https://www.chartjs.org/docs/latest/. (Дата звернення: 23.09.23)
42. 15 Best JavaScript Chart Libraries in 2023 [Електронний ресурс]. 2023. Режим доступу до ресурсу: https://www.atatus.com/blog/javascript-chart-libraries/. (Дата звернення: 23.09.23)
43. Chart.js [Електронний ресурс]. Режим доступу до ресурсу: https://www.w3schools.com/ai/ai_chartjs.asp. (Дата звернення: 23.09.23)

44. Chart.js - Introduction [Електронний ресурс]. Режим доступу до ресурсу: https://www.tutorialspoint.com/chartjs/chartjs_introduction.htm. (Дата звернення: 23.09.23)
45. react-chartjs-2 [Електронний ресурс]. Режим доступу до ресурсу: https://react-chartjs-2.js.org/. (Дата звернення: 23.09.23)
46. Create React App Set up a modern web app by running one command. [Електронний ресурс]. Режим доступу до ресурсу: https://create-react-app.dev/. (Дата звернення: 24.09.23)
47. Створюємо новий React-додаток [Електронний ресурс]. Режим доступу до ресурсу: https://uk.legacy.reactjs.org/docs/create-a-new-react-app.html. (Дата звернення: 24.09.23)
48. package.json Quick Start Guide [Електронний ресурс]. 2022. Режим доступу до ресурсу: https://phoenixnap.com/kb/package-json#:~:text=npm%20init%20command.-,The%20package.,the%20project%20and%20handle%20dependencies. (Дата звернення: 24.09.23)
49. Difference between dependencies, devDependencies and peerDependencies [Електронний ресурс]. 2022. Режим доступу до ресурсу: https://www.geeksforgeeks.org/difference-between-dependencies-devdependencies-and-peerdependencies/. (Дата звернення: 24.09.23)
50. Redux A Predictable State Container for JS Apps [Електронний ресурс]. Режим доступу до ресурсу: https://redux.js.org/. (Дата звернення: 19.09.23)
51. Move faster with intuitive React UI tools [Електронний ресурс]. Режим доступу до ресурсу: https://mui.com/. (Дата звернення: 23.09.23)
52. json-server [Електронний ресурс]. Режим доступу до ресурсу: https://www.npmjs.com/package/json-server. (Дата звернення: 24.09.23)
53. Axios Promise based HTTP client for the browser and node.js [Електронний ресурс]. Режим доступу до ресурсу: https://axios-http.com/. (Дата звернення: 24.09.23)

54. React Router [Електронний ресурс]. Режим доступу до ресурсу: https://reactrouter.com/en/main. (Дата звернення: 24.09.23)
55. ReactJS Router [Електронний ресурс]. 2023. Режим доступу до ресурсу: https://www.geeksforgeeks.org/reactjs-router/. (Дата звернення: 24.09.23)
56. Ofoegbu J. How to Use JSON Server for Front-end Development [Електронний ресурс]. Juliet Ofoegbu. 2023. Режим доступу до ресурсу: https://www.freecodecamp.org/news/json-server-for-frontend-development/. (Дата звернення: 24.09.23)
57. Зелінська О.В., Потапова Н.А., Волонтир Л.О. Інформаційні системи та технології в галузі. Навчальний посібник. Вінниця: ВНАУ, 2020. 263 с.
58. Бурєннікова Н.В., Зелінська О.В., Ушкаленко І.М., Бурєнніков Ю.Ю. Оптимізаційні методи та моделі. Навчальний посібник. Вінниця: ВНТУ, 2019. 114 с.
59. Зелінська О.В., Потапова Н.А., Ємельянова А.О. Інформаційна система по веденню реєстру клієнтів банку. Вісник Хмельницького національного університету: Технічні науки. Хмельницький: ХНУ, 2023. №1. С 94-99.
60. Зелінська О.В., Ємельянова А.О. Проектування застосунку інтелектуального аналізу та контролю особистих фінансів. Вісник студентського наукового товариства Донецького національного університету імені Василя Стуса. Том 2. Вінниця: ДонНУ імені Василя Стуса, 2023. Вип. 15. Т. 2. С 193-197.

ДОДАТКИ

ДОДАТОК А

Перелік встановлених залежностей проекту

```
"dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@fontsource/roboto": "^5.0.3",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.13.6",
    "@mui/styled-engine-sc": "^5.12.0",
    "@mui/x-date-pickers": "^6.9.0",
    "axios": "^1.4.0",
    "chart.js": "^4.4.0",
    "classnames": "^2.3.2",
    "dayjs": "^1.11.8",
    "json-server": "^0.17.3",
    "prop-types": "^15.8.1",
    "react": "^18.2.0",
    "react-chartjs-2": "^5.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^8.0.5",
    "react-router-dom": "^6.10.0",
    "react-scripts": "5.0.1",
    "redux": "^4.2.1",
    "uuid": "^9.0.0",
    "web-vitals": "^2.1.4"
  },
"devDependencies": {
    "eslint": "^8.48.0",
    "gh-pages": "^5.0.0",
    "prettier": "3.0.3",
    "redux-devtools-extension": "^2.13.9",
    "redux-thunk": "^2.4.2",
    "thunk": "^0.0.1"
  }
```

ДОДАТОК Б

Реалізація роутингу в додатку

```
import React from "react";
import { Routes, Route } from "react-router-dom";

import PropTypes from 'prop-types';

import Home from "./pages/Home";
import CreateOperation from "./pages/CreateOperation";
import Analytics from "./pages/Analytics";
import Accounts from "./pages/Accounts";

import { ErrorBoundary } from "./components/ErrorBoundary";

const Router = ({setShowBottomNav}) => {
  return <Routes>
    <Route
      index element={<Home setShowBottomNav={setShowBottomNav}
/>}
      errorElement={<ErrorBoundary />}
    />
    <Route
      path="/finance-tracker"
      element={<Home setShowBottomNav={setShowBottomNav} />}
      errorElement={<ErrorBoundary />}
    />
    <Route
      path="/home"
      element={<Home setShowBottomNav={setShowBottomNav}/>}
      errorElement={<ErrorBoundary />}
    />
    <Route
      path="/analytics"
      element={<Analytics setShowBottomNav={setShowBottomNav}
/>}
      errorElement={<ErrorBoundary />}
    />
    <Route
      path="/operation"
      element={<CreateOperation
setShowBottomNav={setShowBottomNav} />}
      errorElement={<ErrorBoundary />}>
      <Route
```

```
          path=":transactionId"
          element={<CreateOperation />}
          errorElement={<ErrorBoundary />}
        />
      </Route>
      <Route
        path="/accounts"
        element={<Accounts setShowBottomNav={setShowBottomNav} />}
        errorElement={<ErrorBoundary />}
      />
      <Route path="/*" element={<ErrorBoundary
setShowBottomNav={setShowBottomNav} />} />
    </Routes>
}

Router.propTypes = {
  setShowBottomNav: PropTypes.func
}

export default Router;
```

Додаток В

Редуктор зміни стану транзакцій у Redux сховищі

```
import {
  GET_TRANSACTIONS,
  POST_TRANSACTION,
  UPDATE_TRANSACTION,
  DELETE_TRANSACTION,
  TRANSACTION_REQUEST_ERROR
} from "./operationsAction";

const operationsReducer = (state = [], action) => {
  switch (action.type) {
    case GET_TRANSACTIONS:
      return action.payload;
    case POST_TRANSACTION:
      return {
        operations: [...state, action.payload],
      };
    case UPDATE_TRANSACTION: {
      const newState = state.filter((item) => item.id !==
action.payload.id);

      return [...newState, action.payload]
    }
    case DELETE_TRANSACTION: {
      const newState = state.filter((item) => item.id !==
action.payload);

      return newState;
    }
    case TRANSACTION_REQUEST_ERROR:
      return { errorStatus: action.payload };
    default:
      return state;
  }
};

export default operationsReducer;
```

Додаток Г

Редуктор зміни стану загального балансу в Redux сховищі

```
import {
    GET_BALANCE,
    UPDATE_BALANCE
} from "./balanceAction";

const balanceReducer = (state = [], action) => {
    switch (action.type) {
        case GET_BALANCE:
            return action.payload;
        case UPDATE_BALANCE: {
            const [account, newBalance] = action.payload;
            const newState = {
                ...state,
                [account]: newBalance
            }
            return newState;
        }
        default:
            return state;
    }
}

export default balanceReducer;
```

Додаток 2 до наказу
від «31» березня 2023 року
№119/05

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)