МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система контролю потоків фінансів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Проценко О.Б.**

Студента групи ІН.мз – 92с **Кононенко А.П.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Кононенко Анастасії Павлівни

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система контролю потоків фінансів*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд популярних технологій розробки веб додатків; 2)Постановка задачі; 3) Огляд та вибір технологій та програм; 4) Проектування застосунку; 5) Розробка алгоритму додатку; 6) Програмна реалізація.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.			
2.			
3.			
4.			
5.			

Студент – дипломник _____
(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 63 стор., 21 рис., 1 додаток, 16 джерел.

Об’єкт дослідження – інформаційна система розроблення мобільного застосунку для роботи зі сповіщеннями.

Мета роботи — пошук, аналіз та впровадження необхідних технологій, вибір універсального методу розробки; розробка веб додатку.

Результати — створено та запущено сервер; розроблений веб додаток, що дозволяє зручно та швидко виконувати свої функції. Веб додаток протестовано за допомогою розробки Google на швидкість та якість роботи та отримано найвищі результати. Додаток орієнтований на всі найпопулярніші у світі браузери.

REACT.JS, REDUX, TYPESCRIPT, NEXT.JX, FIREBASE API, AUTHENTICATION, MYSQL, MVC, ВЕБ ЗАСТОСУНОК

ЗМІСТ

ВСТУП

Інтернет з кожним днем охоплює все більше сфер життєдіяльності людини. У сучасному світі величезний відсоток людей щодня користується Інтернетом та різними веб-сайтами, адже доступ до різного роду інформації став простіше і швидше. Веб-розробники створюють такі сайти, якими можуть користуватися юзери в глобальній мережі. Існують такі види розробників веб-сайтів:

- Front-end розробники

Інтерфейс – це те, що відображаються на веб-сайті у браузері, включаючи наповнення та елементи інтерфейсу користувача, такі як панель навігації. Front-end розробники використовують HTML, CSS, JavaScript та їх відповідні фреймворки, щоб забезпечити ефективне та логічне відображення вмісту та маніпулювати даними, які надійшли з сервера.

- Back-end розробники

Back-end стосується даних програми, які знаходяться на сервері. Він створює, зберігає, видаляє та редагує дані у базі даних та передає їх до інтерфейсу програми. Для роботи з даними back-end розробники використовують мови програмування, такі як Java, Python та Ruby.

- Full-stack розробники

Full-stack розробникам зручно працювати як з front-end, так і з back-end.

Для будь-якого програмного продукту технологія, яка використовується для його створення, безпосередньо впливає на його продуктивність. Тип продуктивності, яка потрібна програмному забезпеченні, визначає технологію його створення. Тому під час створення веб-програми спочатку слід визначити інструменти, які їй потрібні. Оскільки зараз дуже багато варіантів технологічних стеків у 2020 році, вибір правильного технологічного стеку для розробки веб-додатків є вирішальним фактором.

Мета виконання роботи – оволодіння сучасною методологією розробки інформаційного та програмного забезпечення інформаційної системи на основі використання різних технологій, ознайомлення з сучасними технологіями, архітектурою програмних продуктів, новітніми методами розробки кросбраузерних додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Класифікація веб-додатків

Веб-додаток – це прикладна програма, яка зберігається на віддаленому сервері та доставляється через Інтернет через інтерфейс браузера.

Веб-додатки можуть бути розроблені для найрізноманітніших потреб, і ними може користуватися кожен: від організації до приватної особи. Зазвичай веб-додатки можуть включати веб-пошту, онлайн-калькулятори або магазини електронної комерції. До деяких веб-програм може отримати доступ лише певний браузер, однак більшість з них доступні незалежно від браузера.

Веб-програми не потрібно завантажувати, оскільки вони отримують доступ через мережу. Користувачі можуть отримати доступ до веб-програми через веб-браузер, такий як Google Chrome, Mozilla Firefox, Safari та ін.

Щоб веб-програма працювала, їй потрібен веб-сервер, сервер додатків та база даних. Веб-сервери управляють запитами, що надходять від клієнта, тоді як сервер додатків виконує запитане завдання. База даних може використовуватися для зберігання будь-якої необхідної інформації [14].

Зазвичай веб-програми мають короткий цикл розробки, і їх можна створювати за допомогою невеликих команд розробників. Більшість веб-програм написані на JavaScript, HTML5 та каскадних таблицях стилів (CSS). Клієнтське програмування зазвичай використовує ці мови, які допомагають створювати інтерфейс програми. Програмування на стороні сервера робиться для створення сценаріїв, які використовуватиме веб-програма. Такі мови, як Python, Java та Ruby, зазвичай використовуються в програмуванні на стороні сервера.

Нижче наведено деякі з типів веб-сайтів:

1) Домашні сторінки

Домашня сторінка є головним центром веб-сайту і служить обличчям бренду. Домашня сторінка допомагає відвідувачам сайту дістатися до різних

частин сайту, а також може мати логічну послідовністю переходів між сторінками. Оскільки більшість людей звертаються через домашню сторінку, саме тут дизайн має найбільше значення.

2) Веб-сайти журналів

Веб-сайт журналу містить статті, фотографії та відео, які мають інформаційний та освітній характер. За останні двадцять років журнальна індустрія змінилася з платформи, що публікує лише друк, на цифровий формат. Тип веб-сайту журналу добре працює для інформаційних веб-сайтів, зокрема публікацій університетів та різних організацій.

3) Веб-сайти електронної комерції

Веб-сайт електронної комерції – це місце для онлайн-покупок, де користувачі можуть придбати товари чи послуги у компанії. Надійна веб-сторінка електронної комерції дозволяє легко переглядати товари, фільтрувати за категоріями, виділяти спеціальні продажі та робити покупки.

4) Блог

Блог містить регулярно оновлювані статті, фотографії та відео. Блоги починалися з більш випадкового особистого контенту порівняно з журналами. Зараз надзвичайно часто у великих брендів та підприємств є власний блог. Додавання експертного змісту покращує загальну довіру до компанії чи окремої людини. Блоги також надають матеріали для публікацій у соціальних мережах та електронних кампаній.

5) Портфоліо веб-сайтів

Портфоліо веб-сайт дозволяє творчим професіоналам продемонструвати свої найкращі роботи. Це ідеально підходить для художників, письменників, дизайнерів, кінематографістів, виробників меблів та ін.

6) Лендинг

Лендинг – це певний тип сторінки, створений для маркетингової кампанії, який спонукає відвідувачів до певних дій. Вміст цільової сторінки має бути

обмеженим і спрямовувати на заклик до дії, який потрібен виконати користувач.

7) Каталог та контактні сторінки

Каталог або сторінка контактів – це місце, де користувачі можуть зв’язуватися з представниками компанії чи іншими персонами. Цей тип веб-сайтів добре працює, коли потрібно скласти список підприємств чи людей в організації. Наприклад, у каталозі місцевих ресторанів представлені закусочні з меню, діапазонами цін, номером телефону та відгуками [15].

1.2 Технології розробки веб-додатку

Створення веб-сайту непросте. Існує багато веб-сайтів з хорошим наповненням та функціоналом, але дизайн негарний. Або навпаки, сайт з хорошим дизайном не має достатнього обслуговування. Веб-розробка – це не тільки впровадження кодів на веб-сайті; веб-дизайн також відіграє важливу роль у процесі розробки.

Важливі процеси під час розробки додатку:

1) Збір інформації

Більшість людей ігнорують цей важливий крок у процесі розробки. Щоб переконатись, що веб-дизайн та наповнення сайту відповідає вимогам, необхідний збір інформації.

2) Планування

Після збору достатньо інформації необхідно створити мапу сайтів та каркасних карток. Карта сайту складається з інформацією, зібраною на першому етапі. Основним мотивом мапи сайту є створення зручного для користувача веб-сайту та створення структури сайту. Каркас забезпечує візуальний опис сайту. Крім цього, вирішіть, які функції потрібні сайту Ця функція включає вхід, підписку на електронну пошту, адміністратора, чат в режимі реального часу та багато іншого.

3) Дизайн

Веб-дизайн – це ключова частина успіху веб-додатку. Веб-дизайн створюється відповідно до цільової аудиторії. Веб-сайт, який розроблений для школи, абсолютно відрізняється від сайту для товарів. Наступне про що слід пам’ятати, – це тема, кольоровий контраст, розміщення тексту, зображення, відео тощо. Дизайн-макет структурує сторінку систематично, щоб вона виглядала привабливою.

4) Розробка

На цьому етапі розробник веб-сайту розробляє дизайн кодів. Веб-розробник використовуватиме коди на сайті, щоб він працював і працював безперебійно. Це найважливіший крок у розвитку, оскільки графічний дизайн на попередньому етапі оживає. Відповідно до мап сайтів, домашня сторінка розробляється спочатку.

5) Тестування

Це ще одна важлива частина процесу веб-розробки. Кожні сторінки та посилання повинні перевірятися перед запуском сайту, щоб переконатися, що нічого не порушено. Необхідно перевірити кожну форму, сценарій та запустити програмне забезпечення для перевірки правопису, щоб знайти можливі помилки друку. Також корисно використовувати валідатори коду, щоб переконатися, що код відповідає чинним стандартам веб-розробки.

Також на цьому етапі сайт перевіряється на:

- швидкість веб-сайту;
- сумісність між браузерами;
- тести на екрані.

1.3 Сучасні технології розробки на прикладі фреймворків

У наш час бізнес повинен мати сильну присутність в Інтернеті, і для цього створення веб-сайту має велике значення. Інтернет та цифровий світ постійно розвиваються та зростають вражаючими темпами. Завдяки цій

безперервній еволюції та зростанню сьогодні все робиться набагато по-іншому, з можливостями для експериментів.

Фреймворки відіграють важливу роль у галузі веб-розробки та веб-додатків. На ринку доступно багато фреймворків. Кожен фреймворк має свої переваги та недоліки. Тож вибір найкращої основи, з якою слід працювати, може бути складним.

Нижче представлений список найпопулярніших фреймворків веб-розробки, а на Рисунку 1.1 на діаграмі показано тенденції використання фреймворків:

1) Ruby on Rails

Досить популярно називаний RoR, Ruby on Rails сьогодні став одним із фаворитів серед веб-розробників. З моменту запуску в 2005 році RoR все ще є абсолютно безкоштовним у користуванні, має відкритий код і працює на Linux.

2) Symfony

Це одна з найбільш стабільних фреймворків і ідеально підходить для використання у складних проектах. Використовуючи цю структуру, розробники отримують можливість створювати веб-сайти, які можуть змінюватися в міру того, як вимоги бізнесу змінюються з часом.

3) Angular JS

Один з найпопулярніших та найвідоміших фреймворків – Angular.js походить від цифрового гіганта доби, Google. По суті, це фреймворк з відкритим кодом JavaScript, який допомагає створювати окремі веб-сторінки, використовуючи архітектурний шаблон MVC (Model-Controller-View). Це насправді не повний фреймворк, але можна розглядати його як інтерфейсну структуру, яка чудово підходить для роботи з веб-сторінками.

4) React.js

Бібліотека JavaScript з відкритим кодом, React.js, розроблена Facebook та підтримується масовою спільнотою розробників. Незважаючи на те, що це інструмент розробки веб-сайтів для електронної комерції, React.js особливо корисний при розробці інтерфейсу користувача для веб-програм.

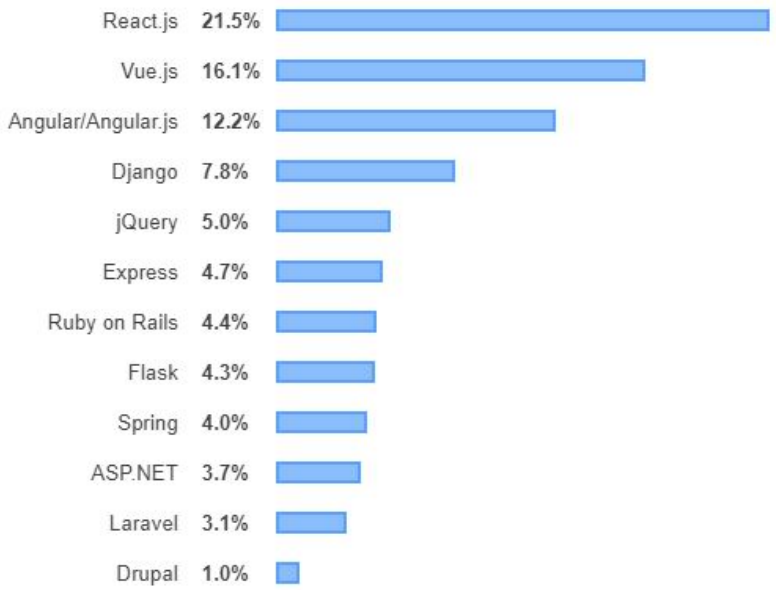

Рисунок 1.1 – Рейтингова діаграма популярності фреймворків

1.4 Принципи програмування

Найперший та найголовніший принцип проектування - DRY (Don’t repeat yourself), означає не писати повторюваний код, а використовувати абстракцію для абстрактних повсякденних речей в одному місці [10].

Якщо блок коду використовується більш ніж у двох місцях, необхідно подумати про те, щоб зробити його окремим методом. Перевага цього об’єктно-орієнтованого принципу проектування полягає в обслуговуванні, зменшенню кількості коді та зменшення часу роботи програміста.

YAGNI («You are not gonna need it»; з англ. - «Вам це не знадобиться») - процес і принцип проектування ПО, при якому в якості основної мети та / або цінності декларується відмова від надмірної функціональності, - тобто відмова додавання функціональності, в якій немає безпосередньої потреби [9].

KISS (англ. keep it simple, stupid — «не ускладнюй, дурню» або більш ввічливий варіант англ. keep it short and simple — «роби коротше і простіше») — процес і принцип проектування, при якому простота системи декларується як основна мета та/або цінність [8].

1.5 Кросбраузерна сумісність у веб-дизайні

Для доступу в мережу Інтернет існує безліч браузерів, найпопулярнішими з яких є Google Chrome, Firefox, Safari, Opera та інші. Вибір дуже великий та підходить для користувачів, які можуть вибрати браузер, який найкраще відповідає їхнім потребам та уподобанням. Однак це створює деякі проблеми для веб-дизайну.

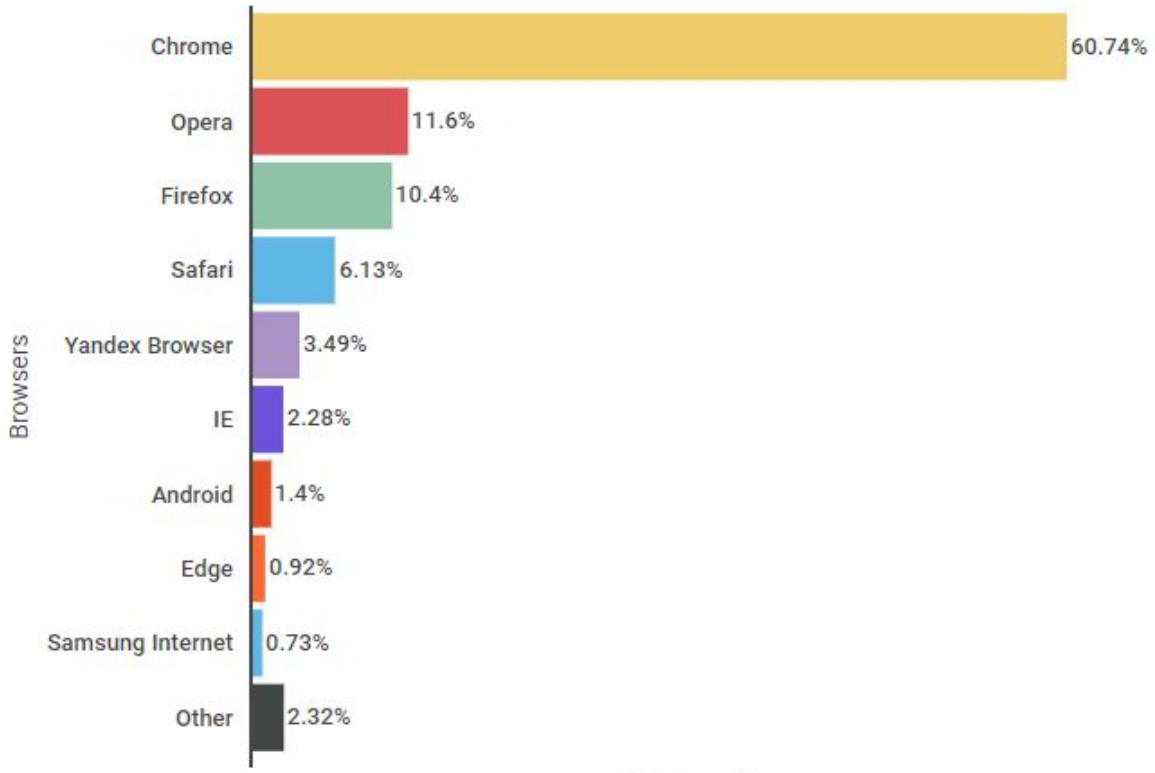

Рисунок 1.2 – Рейтинг найпопулярніших браузерів на 2020 рік [11]

Зокрема, не можна проектувати веб-сайт для одного браузера, оскільки він може виглядати по-іншому, коли користувач відкриє його в іншому браузері. Створення веб-сайту, який працює для всіх ваших потенційних відвідувачів, означає розробку веб-сайту, який працює в декількох браузерах. Ця функція, яка називається кросбраузерною сумісністю, вимагає продуманого веб-дизайну та розробки.

Нижче представлені правила, яким необхідно слідувати для того, щоб зробити сайт кросбраузерним.

1) Написання простого коду

Простота кодування є основним принципом незалежно від того, який тип веб-сайту розробляється. Впорядкування коду створює швидший веб-сайт із меншою кількістю помилок. Крім того, це полегшує виправлення коду або перенесення веб-сайту в інше місце, якщо це необхідно.

2) включити DOCTYPE у код

DOCTYPE – це інструкція на початку коду, яка повідомляє браузерам, якою мовою ви використовували свій код. Без вказівки DOCTYPE браузери повернуться до свого режиму читання коду, що є способом, який браузери використовують, щоб зробити старий веб-дизайн сумісним із сучасними браузерами.

3) Перевірка та валідація коду

Сумісність між браузерами легше досягти, якщо для початку код надійний і не містить помилок. Щоб забезпечити якість коду, його необхідно перевірити, перш ніж додавати його на сайт.

4) Тестування веб-сайту у різних браузерах

Тестування є важливим кроком у виявленні будь-яких потенційних проблем сумісності та їх вирішенні до того, як веб-сайт з’явиться у мережі.

1.6 Постановка задачі

Створити веб-додаток, який буде:

- коректно працювати на всіх найпопулярніших браузерах (Google Chrome, Firefox, Safari, Opera);
- мати такі розділи, як баланс, категорії витрат, зміни, діаграми доходів та витрат та конвертер валют;
- мати гнучку та вдало організовану структуру даних;

- мати просту та зрозумілу користувачу навігацію;
- мати привабливий дизайн;
- мати високу швидкість завантаження та працювати без перешкод та довгого очікування.

2 МЕТОДИ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Клієнтська частина

JavaScript-бібліотека для розробки користувальницького інтерфейсу React.js

Екосистема JavaScript дуже динамічна. Регулярно з’являються нові інструменти та бібліотеки, кожна з яких трохи відрізняється від решти, і у користувача є широкий вибір. Фреймворки не є винятком.

React вже досяг мети і є популярним і використовується багатьма великими компаніями, включаючи, звичайно, Facebook, але також Netflix, AirBNB, DropBox, IMDb, PayPal, Tesla Motors, Walmart та багато інших [1].

React має безліч переваг в порівнянні з іншими фреймворками, основними з яких є:

1) Легкий у вивченні, простий у використанні

React простий у вивченні та простий у використанні, а також постачається документація, навчальні посібники та навчальні ресурси. Той, хто має знання JavaScript, може зрозуміти і почати використовувати React за кілька днів.

2) Багаторазові компоненти

React базується на компонентах. Все починається з дрібних речей, які потім використовуються для створення більших речей, які використовуються для створення додатків. Кожен компонент має свою логіку та керує власним рендерингом і може бути використаний повторно там, де вони він потрібен. Повторне використання коду допомагає спростити розробку та підтримку додатків.

3) Віртуальний DOM

Однією з справді важливих частин React є віртуальна DOM. Зазвичай, коли розробляється програма, яка має багато взаємодії з користувачами та оновлення даних, розробнику слід ретельно продумати, як структура програми вплине на ефективність [13].

У React для кожного об'єкта DOM існує відповідний "віртуальний об'єкт DOM". Віртуальний об'єкт DOM – це представлення об'єкта DOM, як легка копія.

Віртуальний DOM-об'єкт має ті самі властивості, що і реальний DOM-об'єкт, але йому не вистачає реальної сили безпосередньо змінити те, що на екрані.

Маніпулювання DOM відбувається повільно. Маніпулювання віртуальним DOM відбувається набагато швидше, тому що на екрані нічого не малюється.

4) Легше писати за допомогою JSX

JSX це технологія, яка була представлена React.

Хоч React може працювати без використання JSX, це все ж ідеальний спосіб роботи з компонентами, так що React багато в чому виграє, застосовуючи JSX.

Використання JSX синтаксису – це написання того ж самого декларативного синтаксису, але тільки тим, чим повинен бути компонент UI.

Препроцессор JSX

JSX – це XML / HTML-подібний синтаксис, що використовується React, який розширює ECMAScript, щоб текст, подібний XML / HTML, міг співіснувати з кодом JavaScript / React [3]. Синтаксис призначений для використання препроцесорами (тобто транпіляторами, такими як Babel) для перетворення HTML-подібного тексту, знайденого у файлах JavaScript, у стандартні об'єкти JavaScript, які механізм JavaScript проаналізує.

В основному, використовуючи JSX, можна писати стислі HTML / XML-подібні структури (наприклад, DOM-подібні деревоподібні структури) в тому ж файлі, що і код JavaScript, тоді Babel перетворить ці вирази на фактичний код JavaScript. На відміну від минулого, замість того, щоб вставляти JavaScript у HTML, JSX дозволяє розміщувати HTML у JavaScript.

Мова програмування TypeScript

TypeScript (TS) – це мова програмування з відкритим кодом, створена корпорацією Майкрософт [4]. Він асоціюється з JavaScript (JS) завдяки тому, що він компілюється з ним. Це означає, що код, написаний на TypeScript, автоматично перекладається на JavaScript. Простіше кажучи, TS – це “обгортка” JS. Вони створюються для прискорення та полегшення роботи програміста. Перевірка типу – безумовно, найбільша перевага TypeScript. Його використання усуває багато випадково створених помилок, які виникають внаслідок ненадійних типів даних. Завдяки цьому програміст може зосередитись на розробці програми і не витрачати час без потреби на пошук та аналіз отриманих помилок.

JSLint - це статичний аналізатор коду з веб-інтерфейсом для програм на мові JavaScript, перевіряючий їх відповідність стандартам оформлення коду [5].

Фреймворк для веб розробки Material UI

На сьогоднішній день матеріальний інтерфейс є найпопулярнішим інтерфейсом інтерфейсу користувача для React, а бібліотека пропонує безліч готових до використання компонентів. Бібліотека складається з компонентів для макета, навігації, введення, зворотного зв'язку тощо. Інтерфейс користувача Material заснований на Material Design, мові дизайну, яку Google виклав спочатку, але зараз широко застосовується у спільноті розробників інтерфейсів [7].

Матеріальний дизайн був спочатку анонсований в 2014 році і побудований на попередньому дизайні Google Now на основі карт [8]. Material Design – це випробувана в боях мова дизайну, яка постачається з підтримкою сучасних стандартних стандартів розробки, таких як швидкість реагування, тематизація, перевага для мобільних пристроїв, і створена так, щоб бути дуже настроюваною.

React фреймворк Next.js

Просто, Next.js – це фреймворк React для розробки односторінкових програм Javascript [6]. Нижче представлені деякі з численних переваг цієї основи:

1) Server Side Rendering (SSR)

Компоненти React, які складають орієнтовану на користувача частину веб-сайту, спочатку відображаються на стороні сервера. Це означає, що після доставки HTML-коду клієнту (браузеру користувача) нічого іншого не повинно відбуватися, щоб користувач міг читати вміст на сторінці. Це робить час завантаження сторінки набагато швидшим для користувача.

2) Автоматичне розбиття коду

Next.js є досить розумним, щоб завантажувати лише Javascript і CSS, необхідні для будь-якої даної сторінки. Це дозволяє значно швидше завантажувати сторінки, оскільки браузер користувача не повинен завантажувати Javascript та CSS, які йому не потрібні для певної сторінки, яку користувач переглядає. Це підвищує продуктивність, оскільки браузер користувача завантажує менше, і користувач отримує вигоду від швидшого перегляду вмісту сторінки.

3) Hot Module Replacement (HMR)

Це менш важливо для кінцевих користувачів програми, але дуже важливо для розробників. HMR дозволяє розробникам бачити будь-які зміни, внесені ними під час розробки, переглядати в додатку одразу після їх внесення. Однак, на відміну від традиційних методів "живого перезавантаження", він перезавантажує лише ті модулі, які насправді змінилися, зберігаючи стан, в якому перебувало додаток, і значно скорочуючи час, необхідний для того, щоб побачити зміни в дії [12]. Зрештою, позитивним впливом для наших клієнтів є те, що нам потрібно менше часу для розвитку, оскільки є ефективність розвитку, яку слід отримати.

2.2 Серверна частина

Firebase – це платформа для розробки мобільних та веб-додатків, яка підтримується Google, щоб допомогти розробникам забезпечити більш багатий досвід роботи з додатками [2]. Firebase керує власною інфраструктурою із гарним набором інструментів, що спрощують робочий процес розробника,

надаючи їм набори для розробки. Ці набори інструментів взаємопов’язані, масштабовані та інтегровані зі стороннім програмним забезпеченням для подолання складних проблем.

Платформа складається з великого набору інструментів розробки. База даних Realtime і Cloud Firestore можуть зберігати структуровані за документами дані та синхронізувати відповідні програми за мілісекунди щоразу, коли відбувається перетворення даних. Це означає, що як програма, так і її база даних слухають один одного, надаючи користувачеві реактивну роботу програми. А хмарні функції Firebase можуть навіть розширити цю функціональність. Ці функції дозволяють розробнику писати бекенд-код для реагування на події, що відбуваються на платформі Firebase, без необхідності мати справу з будь-якими серверами.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Практична реалізація

Щоб почати роботу, потрібно створити React-проект. Це можна зробити за допомогою команди create-react-app, але так як буде необхідно надалі використовувати Next.jx, то найпростіший спосіб розпочати роботу з Next.js – за допомогою create-next-app. Цей простий інструмент дозволяє швидко розпочати створення нової програми Next.js з усіма налаштуваннями:

```
asiia@Anastasiias-MacBook-Pro Job % create-next-app expenses
```

Prettier – це інструмент для автоматичного форматування коду. Мета Prettier – надати можливість розробникам не думати про стиль під час написання коду, не потрібно перевіряти непослідовні бази кодів у пошуках "помилок". Розробники пишуть код, Prettier його форматує.

Husky запобігає поганому `git commit` та `git push`!

Необхідно встановити Prettier та Husky за допомогою команди:

```
asiia@Anastasiias-MacBook-Pro expenses % npm install prettier husky --save-dev
```

Далі створюється файл .prettierrc з кодом зі списком правил форматування:

```
{
 "printWidth": 120,
 "tabWidth": 2,
 "semi": false,
 "singleQuote": true,
 "jsxSingleQuote": true
}
```

Наступним додається код у файл package.json:

```
"husky": {
 "hooks": {
   "pre-commit": "pretty-quick --staged"
 }
}
```

За допомогою даних налаштувань форматування буде виконуватися завжди перед виконанням `git commit`.

Одразу після створення проекту необхідно визначитись з його архітектурою. На високому рівні правильна структура папок має три основні цілі:

1. Це допомагає розробникам швидко і впевнено знаходити те, що їм потрібно.
2. Це допомагає розробникам зберігати файли в структурі каталогів у потрібному місці.

Тобто далі необхідно створити структуру директорій проекту для зручного та логічного розміщення файлів:

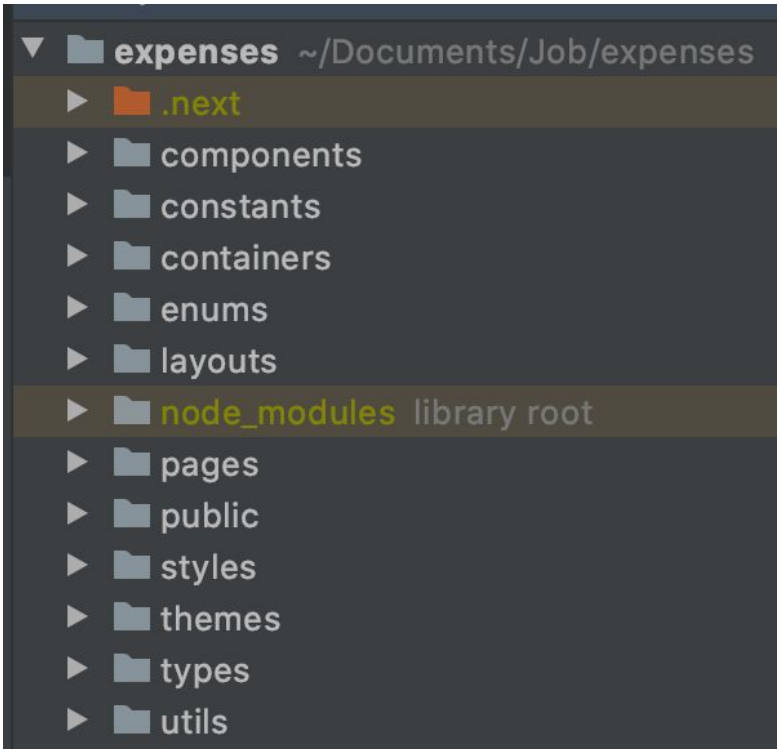

Рисунок 3.1 – Структура директорій проекту

У проекті використовується фреймворк NextJS, який допомагає полегшити навігацію між сторінками. Для створення сторінки необхідно лише додати файл з необхідним ім’ям сторінки та розширенням “js” до папки “pages”. На рисунку 3.2 наведено всю структуру сторінок проекту. Наприклад файл “exchange.js” відповідає сторінці“/exchange”, а запустивши даний додаток сторінку можна знайти за посиланням http://localhost:3000/exchange.\

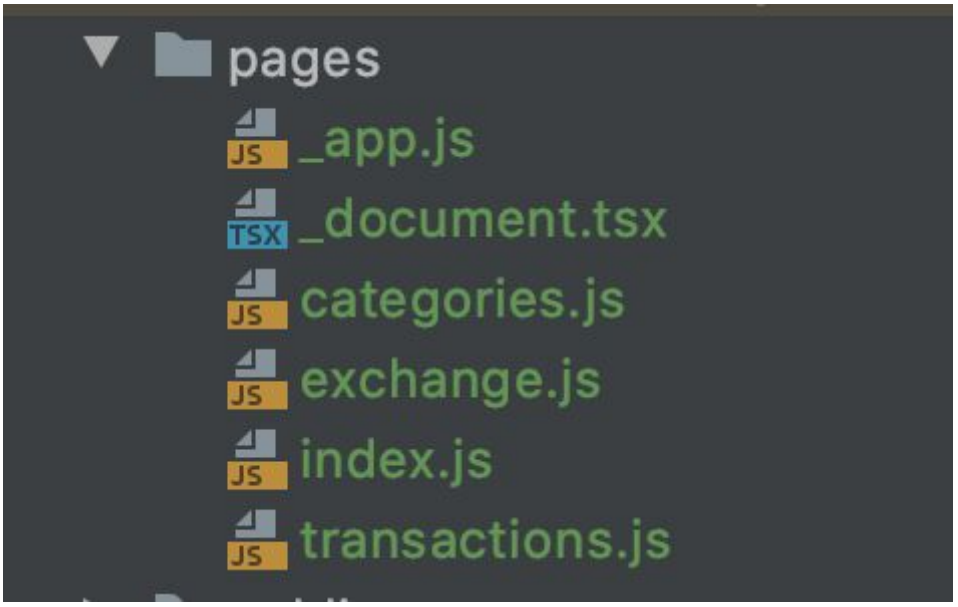

Рисунок 3.2 – Структура папки “pages”

Для того щоб переміщатись між сторінками NextJS є декілька варіантів:

1. Використання хука useRouter з бібліотеки “next/router” для отримання доступу до об’єкта маршрутизатора всередині будь-якого функціонального компонента у програмі. Нижче приведена частина коду з використанням даного способу:

```
import { useRouter } from 'next/router'
...
const router = useRouter()
const redirect = () => {
     router.push(‘/categories’)
}
```

2. Використання компонента Link з біблиотеки “next/link”. Нижче приведений приклад використання даного компоненту у компоненті меню навігації Navigation.

```
<Link href='/categories'>
 <Tooltip title='Categories' enterDelay={450}
placement='right'>
   <IconButton
     style={{ ...styles.iconButton, ...(router.pathname ===
'/categories' ? styles.selected : {}) }}
   >
     <LocalOfferIcon style={{ color: 'white' }} />
   </IconButton>
 </Tooltip>
</Link>
```

3. Хок withRouter, який віддає об’єкт маршрутизації до пропсів компонента. У проекті не використовувався.

Папка “components” містить у собі всі компоненти, які використовуються у проекті. Для покращення навігації для розробника створено вкладені папки та компонента названі відповідно до їх функціоналу. Наприклад, файл index.js у папці Categories відповідає контенту сторінки “categories” та має на одному рівні папку “components”, яка відповідає за зберігання лише тих компонентів які використовуються для даної сторінки (Рисунок 3.3). Файли, які використовуються для декількох компонентах знаходяться у папці “common”.

Рисунок 3.3 – Структура папки “components”

Папка “containers” містить у собі контейнери, які предоставляють дані для виводу або обробки даних сторінкам проекту.

Для того щоб надати компоненту необхідну інформацію необхідно створити контекст, провайдер, виконати певні дії та запити у провайдері та передати всю отриману інформацію контекту, далі обгорнути компонент у провайдер та отримати дані.

Наприклад TransactionsContextProvider надає такі дані як transactions - масив даних всіх транзакцій, createTransaction - функція для створення транзакції, isLoading - булеве значення, яке означає чи готові дані для відображення.

Нижче наведений приклад створення провайдеру:

```
…
export const TransactionsContext = React.createContext({
 transactions: [],
})
export const TransactionsContextProvider = ({ children }) => {
…
const transactions = []; //do some request
 return (
   <TransactionsContext.Provider value={{ transactions }}>
     {children}
   </TransactionsContext.Provider>
 )}
```

У даному проекті був використаний такий принцип розробки як DRY (Don’t repeat yourself). Саме тому у проекті так багато перевикористовуваних компонентів. Гарним прикладом даного принципу є головний лейаут проекту, який знаходиться у папці “layouts”. Мета цього компоненту - створити місце, де будуть поєднані всі спільні між усіма сторінками компоненти, у даному випадку - це хедер та бокове меню навігації.

Папка “public” також являється необхідною у NextJs. Тут зберігаються всі медіа-файли проекту, у даному випадку іконки та зображення.

Папка “styles” містить всі глобальні стилі, а папка “themes” зберігає всі властивості, які часто використовуються у проекті, наприклад кольори та розміри шрифтів.

Для того щоб запустити додаток необхідно виконати скрипт на Рисунку 3.4.

```
asiia@Anastasiias-MacBook-Pro expenses % npm run dev

> expenses@0.1.0 dev /Users/asiia/Documents/Job/expenses
> next dev

ready - started server on http://localhost:3000
event - compiled successfully
```

Рисунок 3.4 – Результат запуску додатку

Для перегляду сайту необхідно перейти за посиланням http://localhost:3000.

Для створення бази даних додатку було обрано платформу від Google - Firebase. Для цього необхідно створити новий проект та додати до його бази даних колекції. Структура даних бази зображена на Рисунку 3.5.

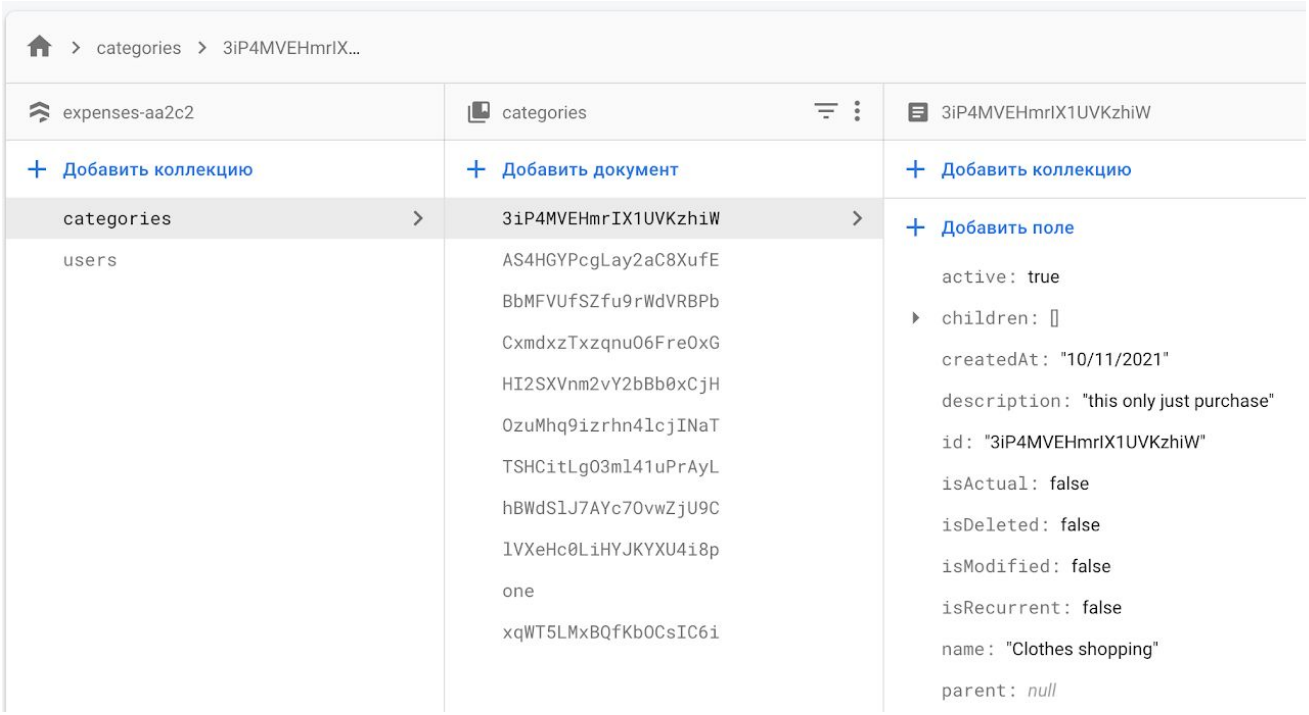

Рисунок 3.5 – Приклад колекції “Categories”

Для того щоб отримати доступ до бази даних необхідно встановити пакет firebase та firebase-tools, підключити Firebase до проекту, додати конфігурацію та ініціалізувати проект:

```
const firebaseConfig = {
   apiKey: "AIzaSyCE4umNjza7Ds-0rhfkjPzp11GzqC5pgTA",
   authDomain: "expenses2-6d5fd.firebaseapp.com",
   databaseURL: "https://expenses2-6d5fd.firebaseio.com",
   projectId: "expenses2-6d5fd",
   storageBucket: "expenses2-6d5fd.appspot.com",
   messagingSenderId: "1002104909562",
   appId: "1:1002104909562:web:1810b027e487f624c52343"
}

if (!firebase.apps.length) {
   firebase.initializeApp(firebaseConfig)
}
```

Тепер додаток має доступ до бази даних. Для того щоб отримати дані необхідно виконати функцію на ефекті useEffect, яка отримує доступ до колекції “`categories`”. Функція, передана в useEffect, буде запущена після того, як рендер буде зафіксований на екрані.

```
import * as React from 'react'
// libs
import firebase from "firebase"
// styles
import '../styles/globals.css'

function MyApp({ Component, pageProps }) {
   const getFirebaseData = async () => {
       const db = firebase.firestore();
       const result = await db.collection("categories")
           .get()
           .then(function(querySnapshot) {
               const snapshots = []

               querySnapshot.forEach(function(doc) {
                   snapshots.push(doc.data())
               });

               return snapshots
           })
           .catch(function(error) {
               console.log("Error getting documents: ", error);
           });

       console.log(result, "result")
   }

 React.useEffect(() => {
     getFirebaseData()
 }, [])

 return <Component {...pageProps} />
}

export default MyApp
```

Далі необхідно задати правила для доступу кожної з колекцій Firebase для читання, редагування та додавання даних.

Приклад правил для колекції “categories” зображений на Рисунку 3.6

Simulated read allowed DETAILS CLEAR

```
{
  "rules": {
      "posts":
      {
        "$pid" : {
          ".read": "data.child('uid').val() === auth.uid",
          ".write": "auth.uid == data.child('uid').val() || auth.uid == newData.child('ui
        }
      },
      "user-posts":
      {
        "$uid": {
              ".read": "auth.uid == $uid",
              ".write": "auth.uid == $uid"
            }
      },
      "users" : {
        "$uid": {
              ".read": "auth.uid == $uid",
              ".write": "auth.uid == $uid"
            }
      }

}
```

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
          request.time < timestamp.date(2020, 12, 25);
    }
  }
}
```

Рисунок 3.6 – Приклад правил колекції “Categories”

За допомогою Styled Components можна писати простий CSS в JavaScript файлі. Це означає те, що можна використовувати весь функціонал CSS, навіть такий як медіа запити, псевдо-селектори, вкладення тощо в JavaScript.

Styled-components використовує шаблонні рядки для стилізації компонентів. Це означає те, що коли визначаються стилі, насправді створюється звичайний React компонент, який має стилі, які вже прикріплені до нього.

Приклад використання styled-components у проекті:

```
export const Wrapper = styled.div`
 display: flex;
 align-items: center;
 justify-content: space-between;
 border: 1px solid black;
 padding: 20px 10px;
`
```

За допомогою компонентів з бібліотеки Material-UI дуже просто використовувати елементи Material Design у React додатку. У даному додатку було використано багато компонентів з бібліотеки, такі як Toolbar, AppBar, Typography, Grid, TextField, Card, CardActions, CardContent, CardMedia, Button та інші.

Приклад використання компонента Button у модальному вікні авторизації:

```
<Button
 variant='contained'
 color='primary'
 fullWidth
 disabled={loading}
 type='submit'
 style={{ margin: '8px 0' }}
>
 {mode === Mode.SIGN_IN ? 'Login' : 'Sign Up'}
</Button>
```

Даний додаток було протестовано за допомогою спеціальної розробки від Google на якість додатку та отримано найкращі результати (Рисунок 3.7).

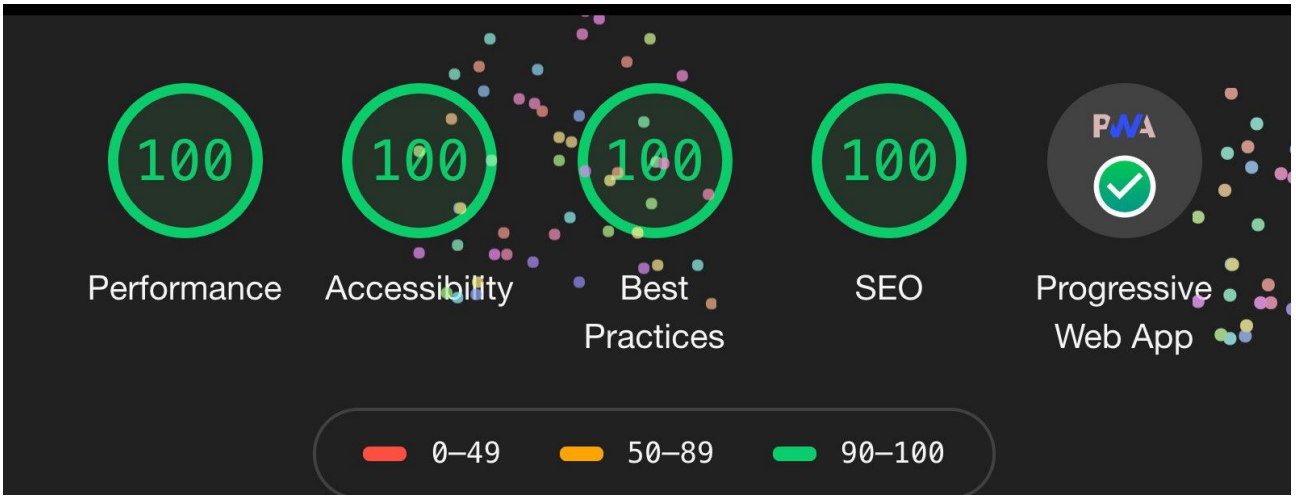

Рисунок 3.7 – Результати аналізу додатка

Webpack – це статичний пакет модулів для додатків JavaScript. Він бере весь код із програми та робить його придатним для використання у веб-браузері. Модулі – це багаторазові фрагменти коду, створені на основі JavaScript додатка, node_modules, зображення та стилі CSS, які упаковані для легкого використання на веб-сайті. Webpack відокремлює код на основі того, як він використовується у додатку, і завдяки цій модульній структурі відповідальності стає набагато легше керувати, налагоджувати, перевіряти та тестувати код. Завдяки Webpack було оптимізовано вихідний код програми до розміру файлу 1MB, що підвищує швидкість завантаження додатку (Рисунок 3.8).

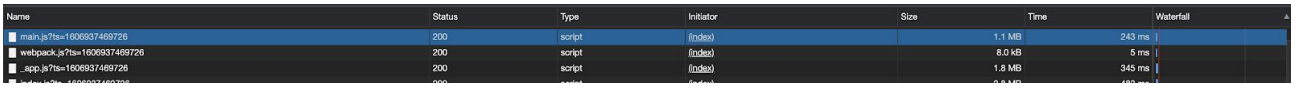

Рисунок 3.8 – Вихідний файл програми у вкладці “Network”

3.2 Алгоритм розробки програмного додатку

Для створення додатку необхідно чітко розуміти термін “транзакція”.

Під транзакцією зазвичай мають на увазі цикл взаємодії об'єктів протягом короткого відрізка часу, який включає запит, виконання завдання і відповідь.

Алгоритм додатку починається з аутентифікації користувача. Після того, як юзер ввів свої логін та пароль додаток запам’ятовує введені дані, створює унікальний токен авторизації, який містить зашифровану інформацію про данного користувача, та записує його до локального сховища сторінки. За допомогою даного токену додаток може розуміти чи давати дозвіл на будь-які дії з транзакціями. Без авторизації юзер не зможе переглядати ніяку інформацію і завжди матиме модальне вікно з реєстрацією або логіном.

Коли юзер успішно ввійшов у систему він може починати користуватися додатком. Будь-яке маніпулювання даних – це робота з базою даних. Під час створення транзакції юзер вирішити є це дохід або витрата і надалі заповнювати форму відповідно до обраного варіанту. Якщо введене значення поля не відповідає правилу – під полем з’явиться тест помилки та її суть. Якщо значення хоча б одного поля некоректне – кнопка відправки буде недоступна. Після підтвердження форми дані відправляються до Cloud Firestore та зберігаються у базі даних. Якщо операція пройшла успішно, то користувач побачить вікно оповіщення зеленого кольору з текстом “Your transaction was successfully saved” (“Транзакцію успішно збережено”). У випадку коли Firebase відхилив запис до колекції або виникла помилка то після спроби виконання запису запит видасть помилку і клієнт побачить вікно сповіщення червоного кольору з текстом “Your transaction was failed” (“Транзакцію відхилено”) а також повідовлення, яке отримано під час виконання запиту. У даному модальному вікні можна обрати 2 варіанти: “Cancel” (“Відмінити”) “Create new transaction” (“Створити нову транзакцію”). Після вибору потрібного варіанту діалогове вікно закривається і, у випадку вибору другого варіанту, відкривається форма для створення нової транзакції.

Під час створення транзакції користувач може обрати опціональне поле “Recurrent transaction” (“Повторювана транзакція”). Після вибору даного поля необхідно ввести дані про те, як часто дана транзакція буде повторюватись (кожні X років, місяців, неділь або днів). Це означає, що кожен зазначений період користувач буде отримувати нотифікації про те, що система хоче додати

нову транзакцію. Якщо юзер підтверджує дану операцію, то система автоматично додасть до бази даних нові дані на основі тих, що було введено під час створення першої транзакції, лише буде змінено дату створення. Схематично алгоритм зображено на Рисуноку 3.9.

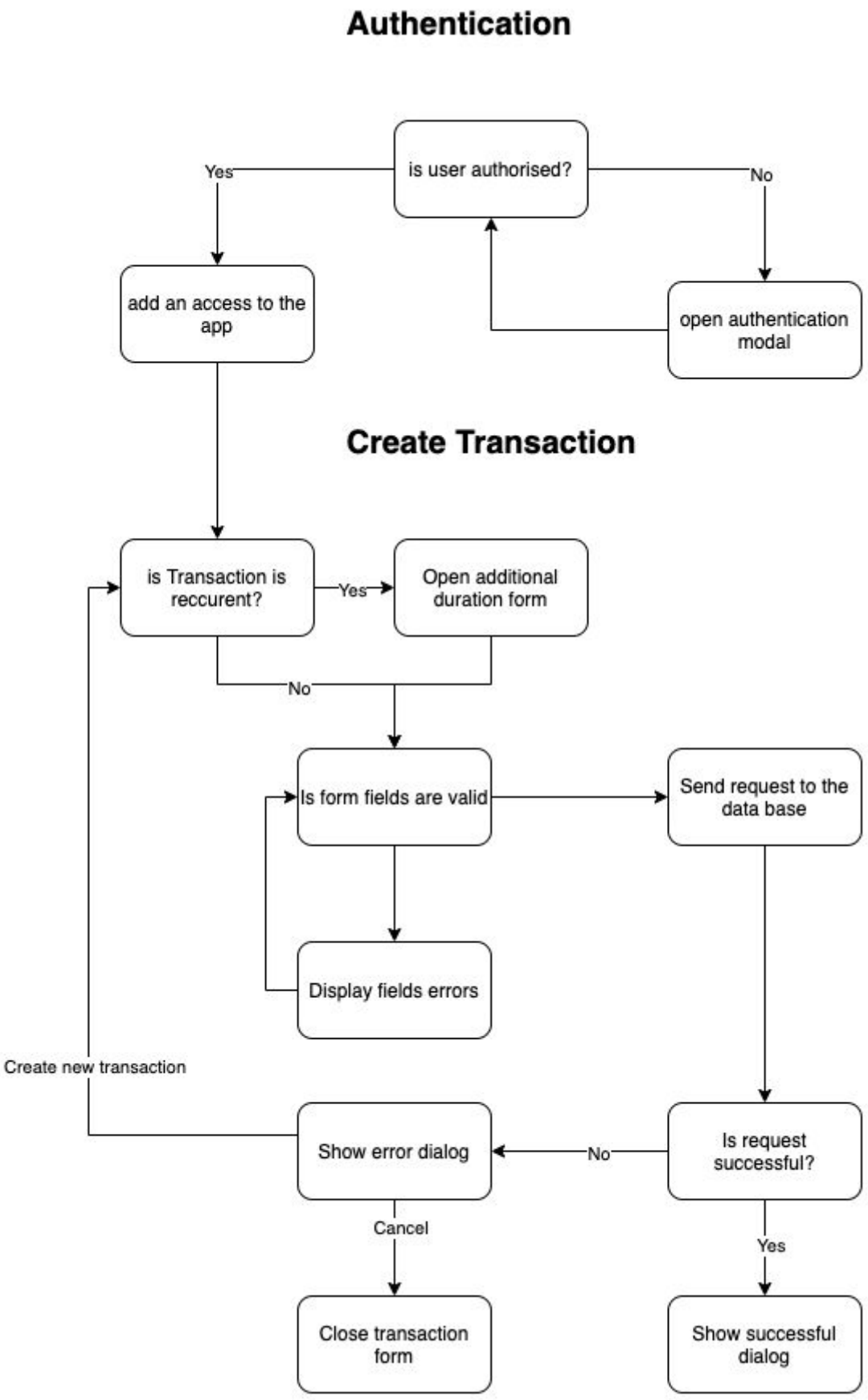

Рисунок 3.9 – Алгоритм у вигляді блок схеми

Після того користувач відкрив додаток та всі дані було завантажено, якщо він вже не авторизований, тобто не має токену авторизації у локальному сховищі браузеру, він побачить модальне вікно для авторизації. На рисунку 3.5 зображено вигляд модального вікна для логіну.

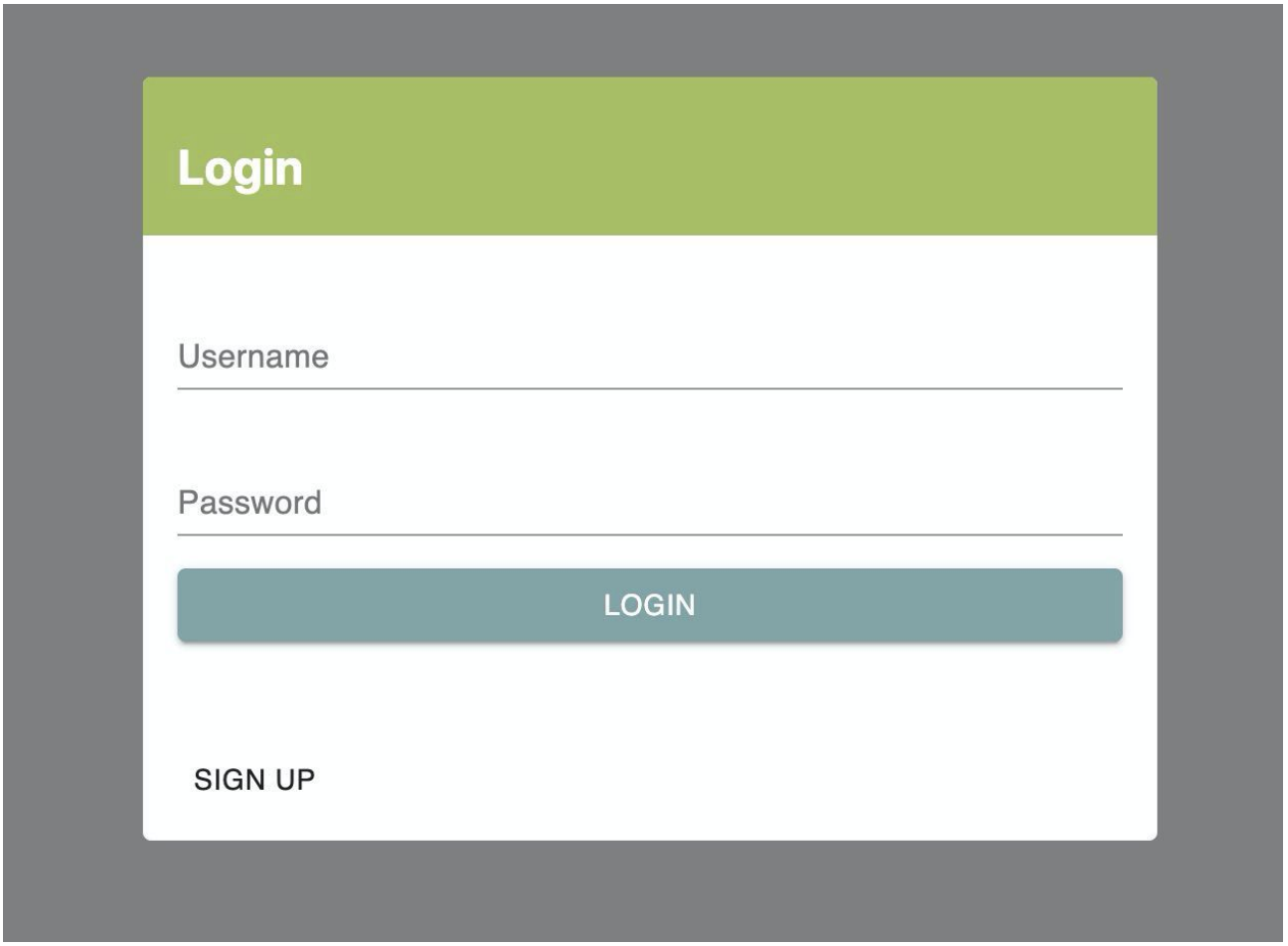

Рисунок 3.10 – Модальне вікно логіну

Якщо користувач ще не має аккаунту у даному додатку, то він може обрати пункт “sign up” для реєстрації. На Рисунку 3.11 зображений крок реєстрації.

Дизайн схожий логіном для зручності юзера та поле для вводу паролю приховує введені символи для безпеки та захисту особистих даних.

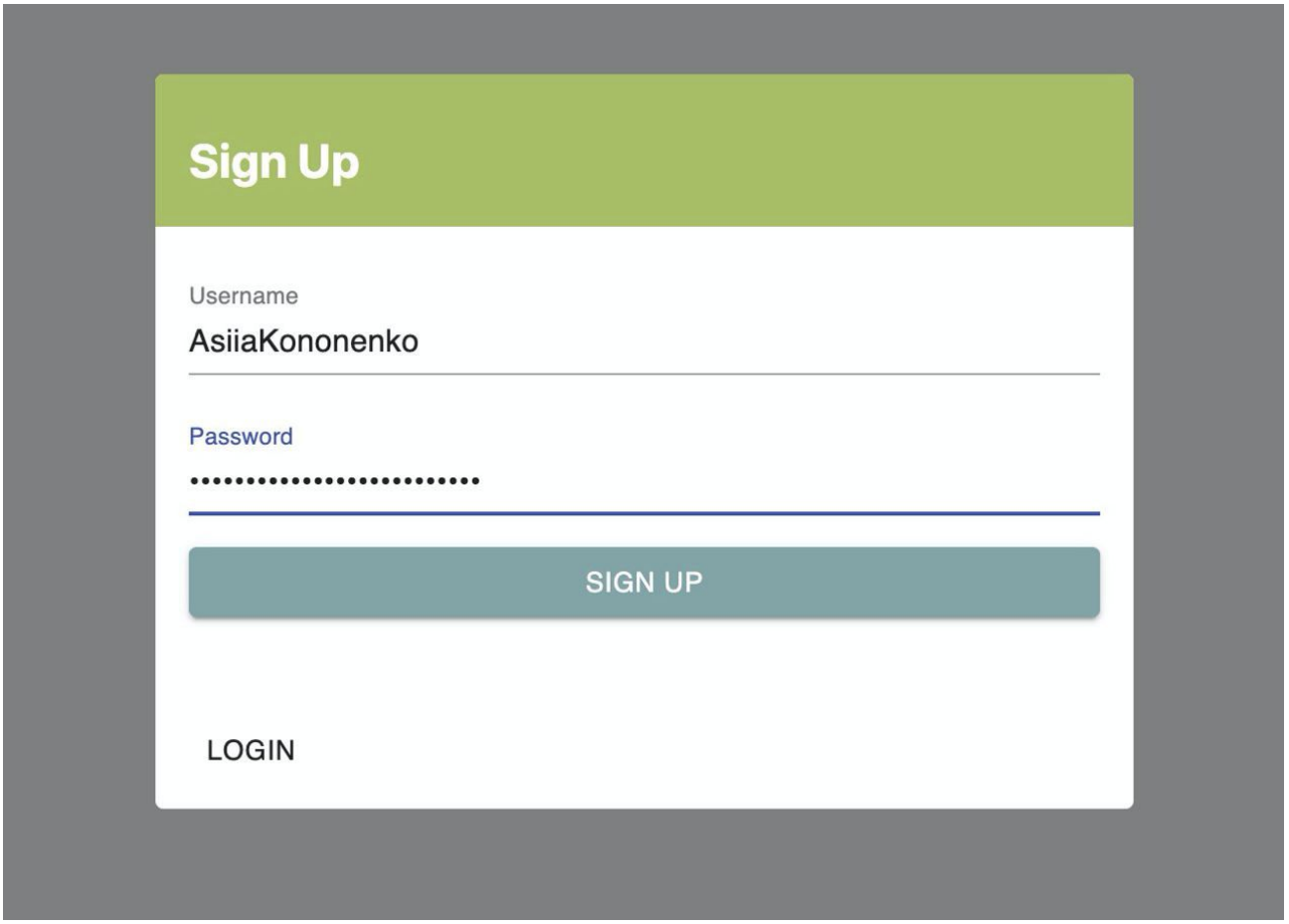

Рисунок 3.11 – Модальне вікно реєстрації

Після того як користувач закінчив роботу та не має намірів заходити до додатку найближчими годинами, то рекомендується виходити зі свого аккаунта, для забезпечення безпеки та конфіденційності даних. Для цього користувач може натиснути кнопку “Logout” у хедері на іконці юзера (Рисунок 3.12).

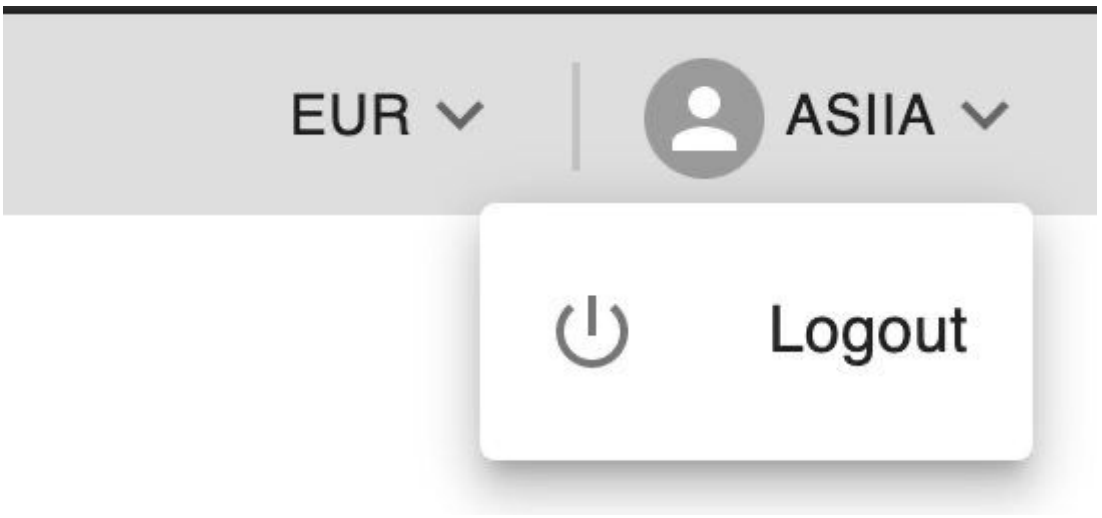

Рисунок 3.12 – Кнопка виходу користувача із власного аккаунту

Після того як користувач авторизувався, він може користуватися повним функціоналом програми.

По-перше, необхідно обрати валюту в якій користувач буде створювати запити та зберігати інформацію. На рисунку 3.13 показано селектор з великим вибором різних валют.

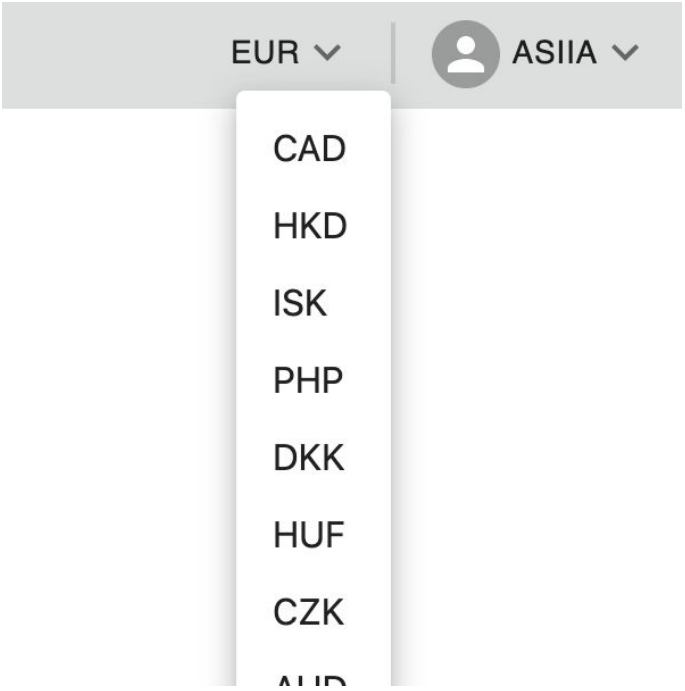

Рисунок 3.13 – Селектор з опціями різних валют

Додатковим функціоналом додатку є конвертер валют. Це необхідно для зручності користувача управляти різними доходами чи витратами у відмінній від головної валюти. На Рисунку 3.14 зображено приклад використання конвертеру. Якщо юзер отримав 1000 рублів, то він з легкістю зможе конвертувати дану суму у євро та зберегти коректну суму.

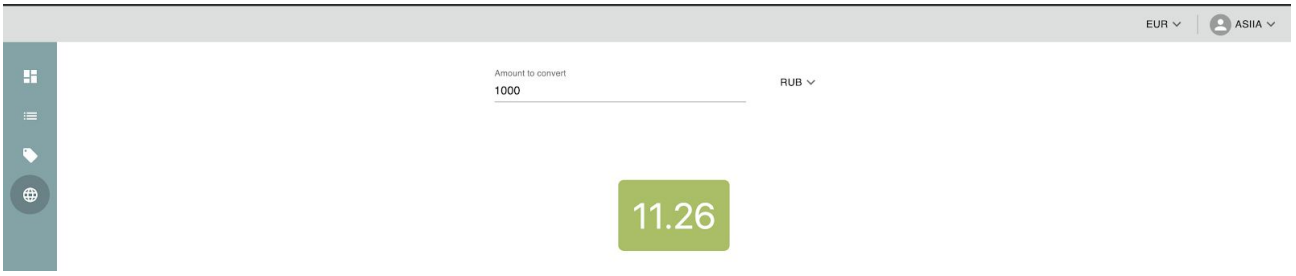

Рисунок 3.14 – Сторінка конвертеру валют

На сторінці категорій користувач зможе додати необхідні категорії та підкатегорії. Це є корисним тоді, коли необхідно підрахувати скільки витрат та доходів юзер зробив у певній сфері свого життя. На рисунку 3.15 показано форму для створення категорії. Також користувач може створити підкатегорію для уточнення витрати

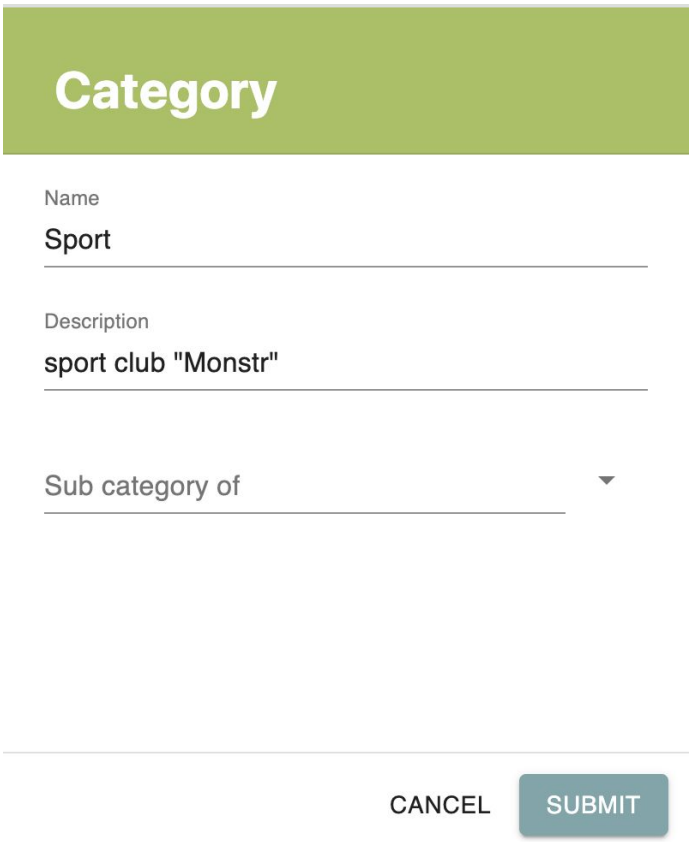

Рисунок 3.15 – Форма для створення категорії

Створити категорію як підкатегорію іншої можна лише зі списку вже існуючих категорій. На рисунку 3.16 зображено модальне вікно для вибору вже існуючих головних категорій. Зробити підкатегорію підкатегорії не можна.

Рисунок 3.16 – Модальне вікно для вибору головної категорії

Усі категорії та підкатегорії будуть зображені у списку зліва. Це зображено на рисунку 3.17. Як тільки користувач додастб нову категорію, вона одразу ж з'явиться у списку.

Рисунок 3.17 – Список категорій на їх підкатегорій

Також над списком категорій знаходиться пошуковий рядок. Пошук виконується таким чином - виявляється чи є будь-яке співпадіння букви у слові не зважаючи на регістр букв, не через словосполучення (Рисунок 3.18).

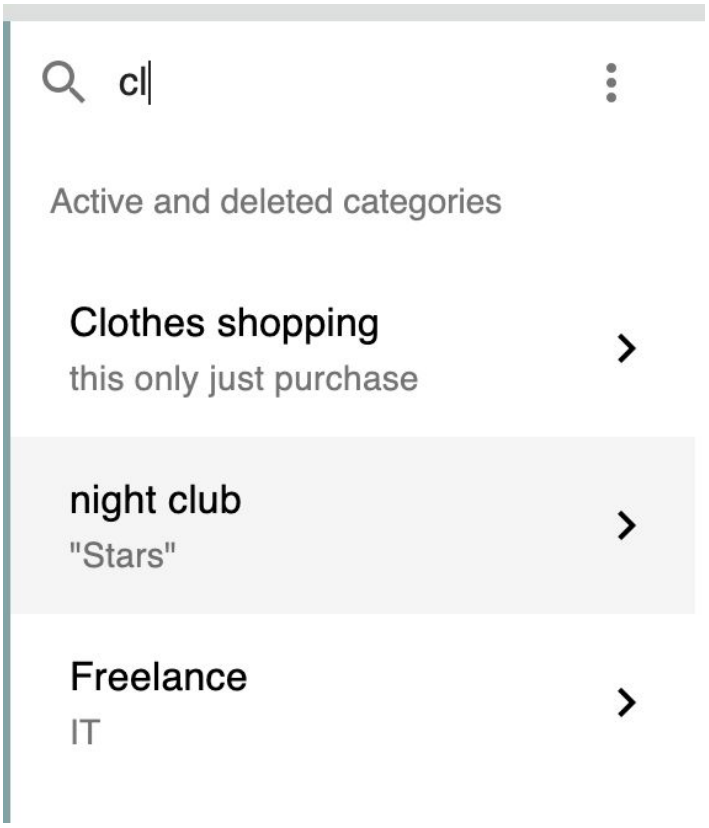

Рисунок 3.18 – Приклад пошуку категорії

Найголовнішим функціоналом додатку є створення транзакцій. Для цього необхідно перейти на сторінку “/transactions” та натиснути на іконку “+”. Після цього з'явиться форма для створення транзакції (Рисунок 3.19). Валюта будет предзаповнена як головна та без можливості змінення. Дату можно ввести вручну або обрати з викликаного модального вікна с календарем.

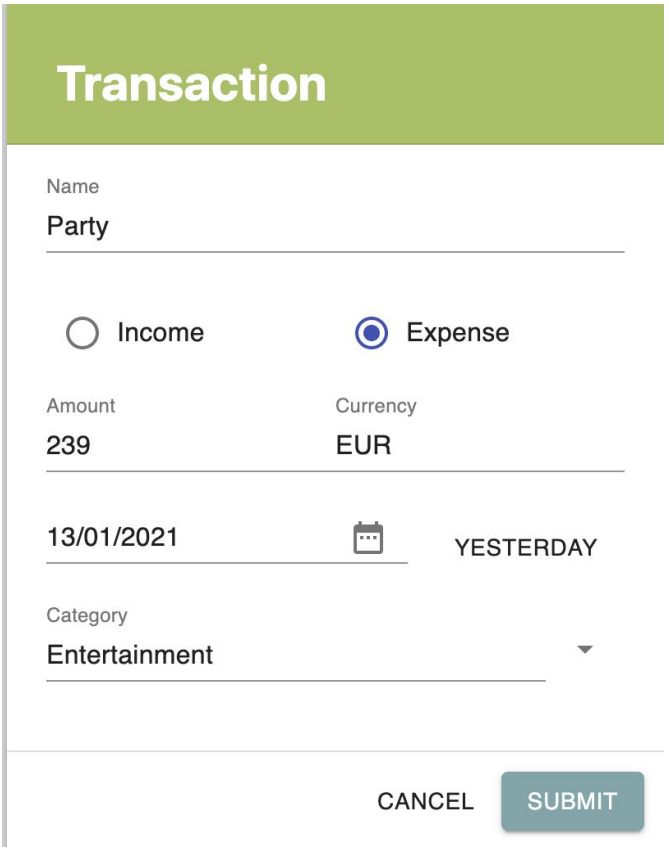

Рисунок 3.19 – Форма для створення транзакції

Після створення транзакції вона одразу відобразиться у списку транзакції. Також всі транзакції будут відсортовані за датою створення та лінійно відображено найважливішу інформацію за поточний місяц. Юзер може переглянути транзакції за попередні місяці, натиснувши кнопку “See previous month”. Червоним кольором позначено витрати та сума зі знаком “-”, а зеленим - дохід. Це необхідно для легкого сприйняття юзером інформації. У лівій частині показано весь список транзакцій та підраховано поточний баланс з детальною інформацією про витрати та дохід (Рисунок 3.20).

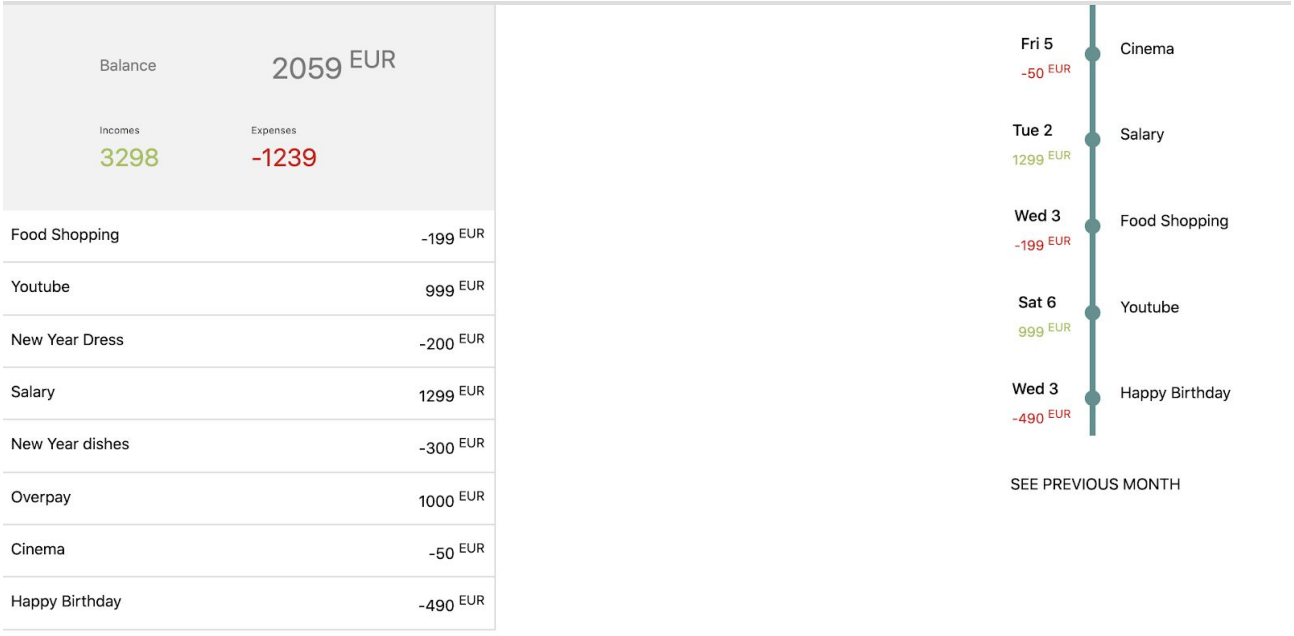

Рисунок 3.20 – Сторінка транзакцій

Після додавання транзакцій користувач може повернувшись до сторінки категорії та переглянути всі витрати чи дохід по категоріям просто вибравши потрібну категорію (Рисунок 3.21)

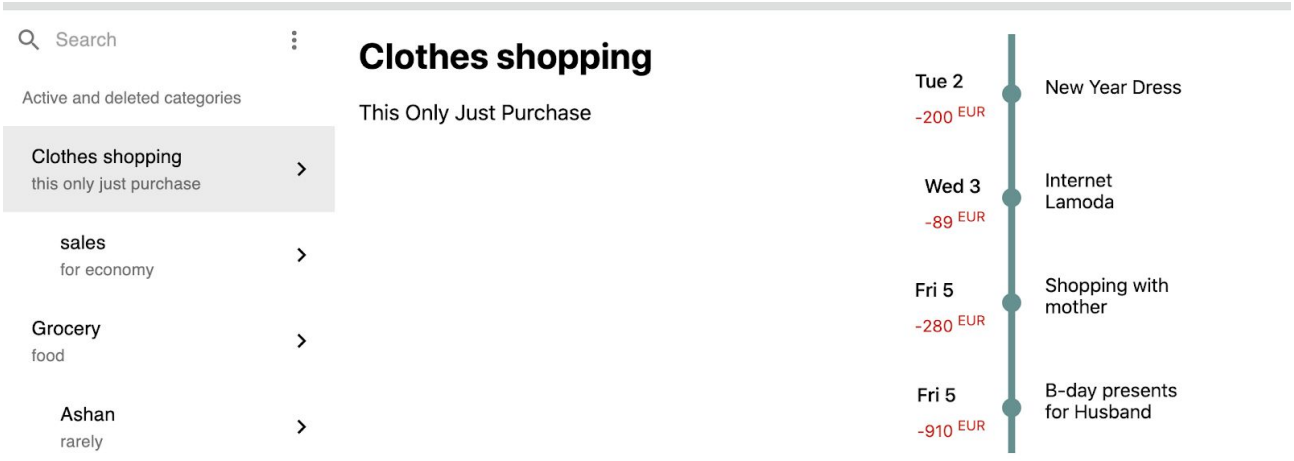

Рисунок 3.21 – Приклад деталей категорії з інформацією про її транзакції

ВИСНОВКИ

У результаті виконання практичної роботи було:

1) виконано порівняльний аналіз сучасних підходів до розробки крос-платформних веб-додатків;

2) розглянуто найсучасніші технології для розробки веб додатків та відібрано ті, які відповідають вимогам даного проекту;

3) детально розглянуто такі технології як React, Redux, Firebase, Styled Components, JSX, Typescript, Next.js, Material UI;

4) розглянуто різні подібні до практичного завдання ресурси;

5) детально вивчено переваги та вимоги кросбраузерної сумісності;

6) чітко визначено етапи створення та розробки даного веб-додатку;

7) знайдено недоліки та наступні шляхи розвитку додатку.

СПИСОК ЛІТЕРАТУРИ

1. React JS Documentation [Електронний ресурс] / Facebook Inc – 2020. – Режим доступу до ресурсу: https://reactjs.org.

2. Documentation [Електронний ресурс] // Google Developers. – 2020. – Режим доступу до ресурсу: https://firebase.google.com/docs.

3. Introducing JSX [Електронний ресурс] / Facebook Inc – 2020. – Режим доступу до ресурсу: https://reactjs.org/docs/introducing-jsx.html

4. TypeScript Documentation [Електронний ресурс] / Microsoft – 2020. – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/.

5. An extensible linter for the TypeScript language. // Palantir Technologies – 2018. – Режим доступу до ресурсу: https://palantir.github.io/tslint/.

6. The React Framework for Production [Електронний ресурс] // Vercel. – 2020. – Режим доступу до ресурсу: https://nextjs.org/.

7. React components for faster and easier web development. Build your own design system, or start with Material Design. [Електронний ресурс] // Material-UI. – 2020. – Режим доступу до ресурсу: https://material-ui.com/.

8. Принцип «KISS». [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://uk.m.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%C2%ABKISS%C2%BB.

9. YAGNI: You Ain’t Gonna Need It. [Електронний ресурс] // Alberto Salas. – 2020. – Режим доступу до ресурсу: https://medium.com/better-programming/yagni-you-aint-gonna-need-it-f9a178cd8e1.

10. Три ключевых принципа ПО, которые вы должны понимать. [Електронний ресурс] // Chris Peters. – 2020. – Режим доступу до ресурсу:https://habr.com/ru/post/144611/.

11. СТАТИСТИКА НАЙБІЛЬШ ПОПУЛЯРНИХ БРАУЗЕРІВ В СВІТІ. [Електронний ресурс] // Антон Юдін. – 2020. – Режим доступу до ресурсу: https://marketer.ua/ua/stats-of-browsers-2017/.

12. Урок по Next.js: SEO-Friendly React E-Commerce SPA. [Електронний ресурс] // WebForMyself. – 2019. – Режим доступу до ресурсу: https://webformyself.com/urok-po-next-js-seo-friendly-react-e-commerce-spa/.

13. How does the virtual DOM work in React!. [Електронний ресурс] // Ajinkya Chanshetty. – 2020. – Режим доступу до ресурсу: https://dev.to/_don_2/how-does-the-virtual-dom-work-in-react-1hin.

14. 10 Types of Web Applications to Simplify your Business. [Електронний ресурс] // intrepidwebstudio. – 2020. – Режим доступу до ресурсу: https://www.google.com/search?q=web+application+types&oq=web+application+types&aqs=chrome..69i57j0l2j0i22i30i395l5.14064j1j4&sourceid=chrome&ie=UTF-8.

15. Types of Web Applications. [Електронний ресурс] // gurutechnolabs. – 2020. – Режим доступу до ресурсу: https://www.gurutechnolabs.com/types-of-web-applications/.

ДОДАТКИ

Зміст файлу exchange.js:

```
import * as React from 'react'
import { Categories } from '../components/Categories'
import { MainLyout } from '../components/layouts'
import { CategoriesContextProvider } from
'../containers/categories'
import { UserContextProvider } from '../containers/login'
import { CurrenciesContextProvider } from
'../containers/currencies'
import { Exchange } from '../components/exchange'

const DashboardPage = () => (
 <UserContextProvider>
   <CategoriesContextProvider>
     <CurrenciesContextProvider>
       <MainLyout>
         <Exchange />
       </MainLyout>
     </CurrenciesContextProvider>
   </CategoriesContextProvider>
 </UserContextProvider>
)

export default DashboardPage
```

Зміст файлу login/index.js:

```
import * as React from 'react'
import firebase from 'firebase'
import { useState } from 'react'

export const UserContext = React.createContext({
 users: [],
 createUser: (userData: {}) => {},
 isLoading: false,
 currentUser: null,
 getUsers: () => {}
})

export const UserContextProvider = ({ children }) => {
 const [users, setUsers] = React.useState([])
 const [currentUser, setCurrentUser] = React.useState(null)
 const [isLoading, setIsLoading] = useState(false)

 const getUsers = async () => {
   const allUsers = []
   const db = firebase.firestore()
   await db
```

```
      .collection('users')
      .get()
      .then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
          allUsers.push({ ...doc.data(), id: doc.id })
        })
      })
      .catch(function(error) {
        console.log('Error getting documents  : ', error)
      })

    setUsers(allUsers)
    const userId = window.localStorage.getItem('authId')
    const user = users.find(us => {
      return us.id == '111' || userId
    })

    setCurrentUser(user)
  }

  React.useEffect(() => {
    getUsers()
  }, [])

  React.useEffect(() => {
    const userId = window.localStorage.getItem('authId')
    const user = users.find(us => {
      return us.id == '111' || userId
    })

    setCurrentUser(user)
  }, [users.length])

  const createUser = userData => {
    setIsLoading(true)
    const db = firebase.firestore()

    db.collection('users')
      .add(userData)
      .then(function() {
        getUsers().then(() => {
          const user = users.find(user => user.name ===
userData.name && user.password === userData.password)
          window.localStorage.setItem('authId', user?.id)
        })
        console.log('Document successfully written!')
        setIsLoading(false)
      })
  }

  return (
    <UserContext.Provider value={{ users, createUser, isLoading,
currentUser, getUsers }}>
```

```
      {children}
    </UserContext.Provider>
  )
}
```

Зміст файлу layouts/index.js:

```
import * as React from 'react'
import styled from 'styled-components'
import { Navigation } from '../Navigation'
import firebase from 'firebase'
import { Sizes } from '../../styles/sizes'
import { Welcoming } from '../CreateAccount'
import { UserContext } from '../../containers/login'
import Avatar from '@material-ui/core/Avatar'
import ExpandMore from '@material-ui/icons/ExpandMore'
import Person from '@material-ui/icons/Person'
import Button from '@material-ui/core/Button'
import ListItemText from '@material-ui/core/ListItemText'
import ListItemIcon from '@material-ui/core/ListItemIcon'
import PowerSettingsNewIcon from
'@material-ui/icons/PowerSettingsNew'
import List from '@material-ui/core/List'
import ListItem from '@material-ui/core/ListItem'
import Popover from '@material-ui/core/Popover'
import { CurrenciesContext } from '../../containers/currencies'

export const MainLyout = ({ children }) => {
  const firebaseConfig = {
    apiKey: 'AIzaSyBwulZPYnV_yU1C4ykJSxgf8pe3qJlLJKQ',
    authDomain: 'expenses-aa2c2.firebaseapp.com',
    databaseURL: 'https://expenses-aa2c2.firebaseio.com',
    projectId: 'expenses-aa2c2',
    storageBucket: 'expenses-aa2c2.appspot.com',
    messagingSenderId: '937087513563',
    appId: '1:937087513563:web:928138fff218775bea2dfe'
  }

  if (!firebase.apps.length) {
    firebase.initializeApp(firebaseConfig)
  }

  // state
  const [isPopoverOpen, setPopoverOpen] = React.useState(false)
  const [anchorEl, setAnchorEl] = React.useState(null)
  const [ratesAnchorEl, setRatesAnchorEl] = React.useState(null)
  const [isRatesPopoverOpen, setRatesPopoverOpen] =
React.useState(false)
  const [currentRate, setCurrentRate] = React.useState('')

  const { currentUser: user } = React.useContext(UserContext)
```

```
 const { rates, updateUserCurrencies } =
React.useContext(CurrenciesContext)

 React.useEffect(() => {
   if (user) {
     setCurrentRate(user.currency)
   }
 }, [user])

 const handleClick = (event = {}) => {
   setAnchorEl(event.currentTarget)
   setPopoverOpen(!isPopoverOpen)
 }

 const handleRatesClick = (event = {}) => {
   setRatesAnchorEl(event.currentTarget)
   setRatesPopoverOpen(!isRatesPopoverOpen)
 }

 const onLogout = () => {
   if (user) {
     window.localStorage.removeItem('authId')
   }
 }

 const onChooseRate = rate => {
   setRatesPopoverOpen(false)
   updateUserCurrencies(rate)
   setCurrentRate(rate)
 }

 return (
   <Wrapper>
     <Welcoming />
     <aside className='navigation'>
       <Navigation />
     </aside>
     <div id='content'>
       <Toolbar>
         <div className='left' />
         <div className='right'>
           {/*{account && !account.isLocal ? <SyncButton
className='showDesktop' /> : ''}*/}

           {/*{user ? <CurrencySelector display='code'
className='showDesktop' /> : ''}*/}
           <Button style={{ padding: '6px 16px' }}
onClick={handleRatesClick}>
             <span>{currentRate}</span>
             <ExpandMore color='action' />
           </Button>

           <hr className='showDesktop' />
```

```
            <Button style={{ padding: '6px 16px' }}
onClick={handleClick}>
              <div className='avatar'>
                <Avatar
                  style={{
                    height: 30,
                    width: 30,
                    fontSize: 14,
                    marginTop: 1,
                    background: 'rgba(0, 0, 0, 0.3)',
                    textTransform: 'uppercase',
                    color: 'white'
                  }}
                >
                  <Person />
                </Avatar>
                <span>{user?.name}</span>
              </div>
              <ExpandMore color='action' />
            </Button>
          </div>

          <Popover
            open={isRatesPopoverOpen}
            onClose={() => setRatesPopoverOpen(false)}
            anchorEl={ratesAnchorEl}
            anchorOrigin={{
              vertical: 'bottom',
              horizontal: 'right'
            }}
            transformOrigin={{
              vertical: 'top',
              horizontal: 'right'
            }}
          >
            <List>
              {Object.keys(rates)?.map(rate => (
                <ListItem button onClick={() =>
onChooseRate(rate)}>
                  {rate}
                </ListItem>
              ))}
            </List>
          </Popover>

          <Popover
            open={isPopoverOpen}
            onClose={() => setPopoverOpen(false)}
            anchorEl={anchorEl}
            anchorOrigin={{
              vertical: 'bottom',
              horizontal: 'right'
            }}
```

```
            transformOrigin={{
              vertical: 'top',
              horizontal: 'right'
            }}
          >
            <List>
              <ListItem button onClick={onLogout}>
                <ListItemIcon>
                  <PowerSettingsNewIcon />
                </ListItemIcon>
                <ListItemText primary='Logout' />
              </ListItem>
            </List>
          </Popover>
        </Toolbar>
      </div>
      <ChildrenWrapper>{children}</ChildrenWrapper>
    </Wrapper>
  )
}

// views
export const ChildrenWrapper = styled.div`
 margin-top: 50px;
 width: 100%;
`

export const Toolbar = styled.div`
 display: flex;
 justify-content: space-between;
 align-items: center;
 position: fixed;
 top: 0;
 left: 0;
 right: 0;
 z-index: 201;
 height: 50px;
 background: #dddede;

 .avatar {
   display: flex;
   align-items: center;

   span {
     margin-left: 5px;
   }
 }

 .showDesktop {
   border-color: rgba(0, 0, 0, 0.12);
 }

 .right {
```

```
    display: flex;
  }
`

export const Wrapper = styled.div`
 min-height: 100vh;
 display: flex;
 flex-grow: 1;
 flex-direction: row;
 overflow: hidden;

 .layout_fab_button {
    position: fixed !important;
    bottom: 10px;
    right: 10px;
    visibility: hidden;
    opacity: 0;
    transform: scale(0.9);
    transition: opacity 0.25s, transform 0.25s, visibility 0s
0.25s;

    &.show {
      visibility: visible;
      opacity: 1;
      transform: scale(1);
      transition: opacity 0.25s, transform 0.25s, visibility 0s;
    }
 }

 &.open {
    visibility: visible;
    opacity: 1;
    @media screen and (min-width: 601px) {
      transform: scale(1);
    }

    @media screen and (max-width: 600px) {
      transform: translateY(0px);
    }

    transition: visibility 0s 200ms, opacity 350ms 200ms, transform
350ms 200ms;
 }

 form.content {
    display: flex;
    flex-direction: column;
    height: 100%;
    header {
      background: var(--primary-color);
      padding: 30px 10px 20px 10px;
      h2 {
        margin: 0;
```

```
      color: white;
      font-size: 30px !important;
      width: 100%;
      padding: 0 20px;
    }
    border-bottom: solid 1px var(--divider-color);
  }

  div.form {
    padding: 2px 24px 20px 24px;
    flex-grow: 1;
    flex-shrink: 1;
    overflow: auto;
    -webkit-overflow-scrolling: touch;
  }

  footer {
    display: flex;
    justify-content: flex-end;
    padding: 12px 12px 12px 12px;
    border-top: solid 1px var(--divider-color);
    flex-shrink: 0;
  }
}

modalContent {
  z-index: 999;

  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;

  visibility: hidden;
  transition: visibility 0s 0.4s;

  &:before {
    content: '';
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    background: black;
    z-index: -1;
    opacity: 0;
    transition: opacity 0.3s 0.2s;
  }

  .modalContentCard {
    z-index: 200;
    width: 100%;
```

```
// Animation
opacity: 0;
transition: transform 0.3s, opacity 0.3s;

@media screen and (max-width: ${Sizes.FULL_SCREEN}px - 1px) {
  position: absolute;
  top: 30px;
  bottom: 0;
  left: 0;
  right: 0;
  transform: translateY(20px);
}

@media screen and (min-width: ${Sizes.FULL_SCREEN}px) {
  position: absolute;
  top: 0;
  bottom: 0;
  right: 0;
  width: 400px;
  transform: translateX(20px);
}

form.content {
  display: flex;
  flex-direction: column;
  height: 100%;
  header {
    background: var(--primary-color);
    padding: 30px 10px 20px 10px;
    h2 {
      margin: 0;
      color: white;
      font-size: 30px !important;
      width: 100%;
      padding: 0 20px;
    }
    border-bottom: solid 1px var(--divider-color);
  }

  div.form {
    padding: 2px 24px 20px 24px;
    flex-grow: 1;
    flex-shrink: 1;
    overflow: auto;
    -webkit-overflow-scrolling: touch;
  }

  footer {
    display: flex;
    justify-content: flex-end;
    padding: 12px 12px 12px 12px;
    border-top: solid 1px var(--divider-color);
```

```
          flex-shrink: 0;
        }
      }
    }

    &.open {
      visibility: visible;
      transition: visibility 0s;
      &:before {
        opacity: 0.2;
        transition: opacity 0.3s;
      }
      .modalContentCard {
        opacity: 1;

        @media screen and (max-width: ${Sizes.FULL_SCREEN}px - 1px)
{
          transform: translateY(0px);
        }
        @media screen and (min-width: ${Sizes.FULL_SCREEN}px) {
          transform: translateX(0px);
        }

        transition: transform 0.3s 0.1s, opacity 0.3s 0.1s;
      }
    }
  }
`
```

Зміст файлу themes/index.js:

```
import cyan from "@material-ui/core/colors/cyan";
import orange from "@material-ui/core/colors/orange";
import green from "@material-ui/core/colors/green";
import blue from "@material-ui/core/colors/blue";
import red from "@material-ui/core/colors/red";
import blueGrey from "@material-ui/core/colors/blueGrey";
import indigo from "@material-ui/core/colors/indigo";

const darktheme = {
 palette: {
   type: "dark",
   primary: blue,
   background: {
     default: "rgb(21, 32, 42)",
     paper: "rgb(28, 41, 55)",
   },
   transparent: {
     default: "rgba(21, 32, 42, 0%)",
     paper: "rgba(28, 41, 55, 0%)",
   },
   numbers: {
```

```
    red: red[400],
    blue: blue[400],
    green: green[400],
    yellow: "#FDD835",
  },
  brand: {
    nomadlist: "rgb(255, 71, 66)",
  },
  cardheader: "rgba(255, 255, 255, 0.12)",
  default: {
    primary: blue,
    main: blue[800],
  },
  dashboard: {
    primary: blue,
    main: blue[700],
  },
  transactions: {
    primary: cyan,
    main: cyan[800],
  },
  categories: {
    primary: green,
    main: green[800],
  },
  changes: {
    primary: orange,
    main: orange[900],
  },
  report: {
    primary: indigo,
    main: indigo[500],
  },
  settings: {
    primary: blueGrey,
    main: blueGrey[600],
  },
  search: {
    primary: blueGrey,
    main: blueGrey[600],
  },
  convertor: {
    primary: blueGrey,
    main: blueGrey[600],
  },
  nomadlist: {
    primary: red,
    main: "rgb(255, 71, 66)",
  },
},
typography: {
  useNextVariants: true,
  fontSize: 14,
```

```
    htmlFontSize: 16,
  },
};

export { darktheme };
```

Зміст файлу components/categories/index.js:

```
import React, { useState, useEffect } from 'react'
import { useTheme } from '@material-ui/styles'
import Card from '@material-ui/core/Card'
import Fab from '@material-ui/core/Fab'
import styled, { css } from 'styled-components'
import List from '@material-ui/core/List'
import ListItem from '@material-ui/core/ListItem'
import ListItemSecondaryAction from
'@material-ui/core/ListItemSecondaryAction'
import ListItemText from '@material-ui/core/ListItemText'
import ListSubheader from '@material-ui/core/ListSubheader'
import MoreVertIcon from '@material-ui/icons/MoreVert'
import SearchIcon from '@material-ui/icons/Search'
import { sortBy } from 'lodash'
import InputBase from '@material-ui/core/InputBase'
import IconButton from '@material-ui/core/IconButton'
import red from '@material-ui/core/colors/red'
import KeyboardArrowRight from
'@material-ui/icons/KeyboardArrowRight'
import UndoIcon from '@material-ui/icons/Undo'
import ContentAdd from '@material-ui/icons/Add'
import CategoryForm from './components/CategoryForm'
import { useRouter } from 'next/router'
import { CategoriesContext } from '../../containers/categories'
import { TransactionsContext } from
'../../containers/transactions'
import moment from 'moment'
import { FlexWrapper, GridWrapper } from
'../transactions/TransactionsContent'

const styles = {
 button: {
    float: 'right',
    marginTop: '26px'
 },
 listItem: {
    paddingLeft: '14px'
 },
 listItemDeleted: {
    paddingLeft: '14px',
    color: red[500]
 }
}
```

```
export const Categories = () => {
 const { categories } = React.useContext(CategoriesContext)
 const { transactions } = React.useContext(TransactionsContext)
 const [filteredCategories, setFilteredCategories] =
useState(categories)
 const [search, setSearch] = useState('')
 const [menu, setMenu] = useState(null)
 const [isOpen, setIsOpen] = useState(false)
 const [component, setComponent] = useState(null)

 const router = useRouter()
 const theme = useTheme()

 const [category, setCategory] = useState(() => (categories ?
categories.find(c => c.id === router.query.id) : null))

 const sortedT = sortBy(transactions, [
   t => {
     return moment(t.date)
   }
 ]).filter(t => t.category.id === category?.id || t.category.id
=== category?.parent)

 React.useEffect(() => {
   categories && !category && setCategory(categories.find(c =>
c.id === router.query.id))
 }, [router.query.id, categories.length])

 useEffect(() => {
   if (search) {
     setFilteredCategories(
       categories
         ? categories.filter(category => {
             const result = search
               .toLowerCase()
               .split('')
               ?.map(s =>
                 category.name
                   .toLowerCase()
                   .split('')
                   .includes(s)
               )

             return result.every(s => s === true)
           })
         : null
     )
   } else {
     setFilteredCategories(categories)
   }
 }, [search, categories])

 const _handleUndeleteCategory = category => {
```

```
    category.active = true
  }

  const handleOpenCategory = category => {
    const component = <CategoryForm category={category} onClose={()
=> setIsOpen(false)} />
    setComponent(component)
    setIsOpen(true)
  }

  const drawListItem = (categories, parent = undefined, indent = 0)
=> {
    return categories
      .filter(category => {
        if (!category.active) {
          return false
        }
        return search ? true : !category.parent
      })
      .map(c => {
        let result = []
        result.push(
          <ListItem
            button
            key={c.id}
            selected={category && category.id === c.id}
            style={{
              ...(c.active ? styles.listItem :
styles.listItemDeleted),
              ...{ paddingLeft: (theme?.spacing?.() || 6) * 4 *
indent + 24 }
            }}
            onClick={event => {
              setCategory(c)
              router.push({ pathname: '/categories/', query: { id:
c.id } })
            }}
          >
            <ListItemText primary={c.name}
secondary={c.description} />
            {c.active ? (
              <KeyboardArrowRight />
            ) : (
              <ListItemSecondaryAction>
                <IconButton onClick={() =>
_handleUndeleteCategory(c)}>
                  <UndoIcon />
                </IconButton>
              </ListItemSecondaryAction>
            )}
          </ListItem>
        )
        const childCat = categories.filter(i => i.parent === c.id)
```

```
        if (childCat.length > 0) {
          childCat.map(childCat =>
            result.push(
              <ListItem
                button
                key={childCat.id}
                selected={category && category.id === childCat.id}
                style={{
                  ...(childCat.active ? styles.listItem :
styles.listItemDeleted),
                  ...{ paddingLeft: (theme?.spacing?.() || 6) * 4 *
(indent + 1) + 24 }
                }}
                onClick={event => {
                  setCategory(childCat)
                  router.push({ pathname: '/categories/', query: {
id: childCat.id } })
                }}
              >
                <ListItemText primary={childCat.name}
secondary={childCat.description} />
                {childCat.active ? (
                  <KeyboardArrowRight />
                ) : (
                  <ListItemSecondaryAction>
                    <IconButton onClick={() =>
_handleUndeleteCategory(c)}>
                      <UndoIcon />
                    </IconButton>
                  </ListItemSecondaryAction>
                )}
              </ListItem>
            )
          )
        }

        return result
      })
  }

  return (
    <MainLayout>
      {isOpen && component && (
        <ModalContent>
          <ModalContentCard isOpen={isOpen} square>
            {component}
          </ModalContentCard>
        </ModalContent>
      )}
      <MainWrapper>
        <div className='layout_content wrapperMobile'>
          <TwoColumnsWrapper>
```

```
          <div className={(category ? 'hide ' : '') +
'layout_noscroll'}>
            <ContextSearch>
              <SearchIcon color='action' />
              <InputBase
                placeholder='Search'
                fullWidth
                value={search}
                onChange={event => setSearch(event.target.value)}
                style={{ margin: '2px 10px 0 10px' }}
              />
              <IconButton onClick={event =>
setMenu(event.currentTarget)}>
                <MoreVertIcon color='action' />
              </IconButton>
            </ContextSearch>

            {categories && !categories.length && (
              <div className='emptyContainer'>
                <p>No categories</p>
              </div>
            )}
            {categories && categories.length ? (
              <List
                className=' wrapperMobile'
                style={{ paddingBottom: 70 }}
                subheader={<ListSubheader
disableSticky={true}>Active and deleted
categories</ListSubheader>}
              >
                {drawListItem(filteredCategories)}
              </List>
            ) : (
              ''
            )}
          </div>
        </TwoColumnsWrapper>
      </div>

      <CatBlock>
        <CatWrapper>
          <div style={{ width: '300px' }}>
            <div className='title'>{category?.name}</div>
            <div className='desc'>{category?.description}</div>
          </div>
          <div>
            <div style={{ width: '500px', margin: '0 auto' }}>
              <GridWrapper>
                {sortedT.map(t => (
                  <>
                    <FlexWrapper isGreen={t.type === 'income'}>
                      <div>
```

```
                          <div
className='date'>{moment(t.date).format('ddd d')}</div>
                          <div className='curr'>
                            {t.type === 'income' ? '' : '-'}
                            {t.amount} <sup>{t.currency.name}</sup>
                          </div>
                        </div>
                      </FlexWrapper>
                      <div className='line dot' />
                      <FlexWrapper start>
                        <div>{t.name}</div>
                      </FlexWrapper>
                    </>
                  ))}
                </GridWrapper>
              </div>
            </div>
          </CatWrapper>
        </CatBlock>
      </MainWrapper>
      <Fab
        color='primary'
        className='show layout_fab_button main-button'
        aria-label='Add'
        disabled={!categories}
        onClick={handleOpenCategory}
      >
        <ContentAdd />
      </Fab>
    </MainLayout>
  )
}

export const CatWrapper = styled.div`
  display: flex;
`

export const CatBlock = styled.div`
  padding: 20px;
  .title {
    font-size: 30px;
    font-weight: bold;
  }

  .desc {
    margin-top: 20px;
    font-size: 18px;
    text-transform: capitalize;
  }
`
export const MainWrapper = styled.div`
  display: flex;
```

```
  .line {
    width: 24px;
    position: relative;
    &:after {
      content: '';
      border-left: solid 6px #5a9090;
      height: 100%;
      position: absolute;
      top: 0;
      bottom: 0;
      left: calc(50% - 3px);
    }
    &.dot {
      position: relative;
      &:before {
        content: '';
        height: 100%;
        position: absolute;
        top: 50%;
        left: 50%;
        margin-top: -2px;
        //margin-left: -2px;
        width: 16px;
        height: 16px;
        border-radius: 10px;
        background: #5a9090;
        transition: all 0.2s;
        left: calc(50% - 8px);
      }
    }
  }
`

export const ContextSearch = styled.div`
  background: var(--paper-color);
  border-bottom: solid 1px var(--divider-color);
  display: flex;
  align-items: center;
  padding: 0 10px;
  position: -webkit-sticky;
  position: sticky;
  top: 0;
  z-index: 100;
  min-height: 49px;
`

export const TwoColumnsWrapper = styled.div`
  width: 100%
  height: 100%;
  position: relative;
  overflow: auto;
  display: flex;
  flex-direction: row;
```

```
`

export const MainLayout = styled.div`
 width: 100%;
 min-height: 100%;
`

export const ModalContent = styled.div`
 z-index: 200;
 position: absolute;
 top: 50px;
 bottom: 0;
 left: 0;
 right: 0;

 h2 {
   margin: 0;
   padding: 0 20px;
   font-size: 30px;
 }
 :before {
   content: '';
   position: absolute;
   top: 0;
   bottom: 0;
   left: 0;
   right: 0;
   background: black;
   z-index: -1;
   opacity: 0.2;
   transition: opacity 0.3s;
 }
`

export const ModalContentCard = styled(Card)`
 position: absolute;
 top: 0;
 bottom: 0;
 right: 0;
 width: 400px;
 opacity: 0;
 ${({ isOpen }) =>
   isOpen &&
   css`
     opacity: 1;
     transition: transform 0.3s 0.1s, opacity 0.3s 0.1s;
   `}
`
```