A PROJECT ON

Cross Platform Expense Tracking Progressive Web App

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING (CSE ML+AI)

Submitted by:

Harsh Chaturvedi **2014669**

Under the Guidance of

Dr. Parul Madan

Associate Professor

Project Team ID: MP22MLAI19

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand
MAY-2023

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the Project Report entitled **"Cross Platform Expense Tracking Web App"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering and submitted in the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun is an authentic record of my own work carried out during a period from **August-2022 to May-2023** under the supervision of **Dr. Parul Madan**, **Associate Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University).

The matter presented in this dissertation has not been submitted by me/us for the award of any other degree of this or any other Institute/University.

Harsh Chaturvedi 2014669 signature

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Signature Signature

Supervisor **Head of the Department**

External Viva

Name of the Examiners: **Signature with Date**

1.

2.

Abstract

A cross-platform expense tracking web app, meticulously crafted utilizing the powerful capabilities of Next.js, React and Firebase, stands as a reliable solution that empowers users to effortlessly manage their financial journeys. With its intuitive interface and robust functionality, this remarkable application enables users to not only set budgets with precision but also create multiple wallets to meticulously track expenses across various accounts, thereby providing comprehensive oversight and control over their financial resources. Whether it's personal expenses, business expenditures, or any other financial endeavor, this versatile app ensures a seamless and efficient experience, empowering individuals to make informed decisions, achieve financial goals, and navigate their monetary landscape with utmost confidence.

Acknowledgement

Any achievement, be in scholastic or otherwise does not depend solely on the individual effort but on the guidance, encouragement and co-operation of intellectuals, elders and friends. A number of personalities in their own capacity have helped me in carrying out this project work.

Our sincere thanks to project guide **Dr. Parul Madan, Associate Professor,** Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for his valuable guidance and support throughout the course of project work and for being a constant source of inspiration.

We extend our thanks to **Prof. (Dr.) Guru Prasad M.S.**, Project coordinator, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for his valuable suggestions throughout all the phases of the Project Work.

We are extremely grateful to **Prof. (Dr.) D. P. Singh**, HOD of the Computer Science and Engineering Department, Graphic Era (Deemed to be University), for his moral support and encouragement.

We thank the **management of Graphic Era (Deemed to be University)** for the support throughout the course of our Bachelor's Degree and for all the facilities they have provided.

Last, but certainly not least we thank all teaching and non-teaching staff of Graphic Era (Deemed to be University) for guiding us in the right path. Most importantly we wish to thank our parents for their support and encouragement.

Harsh Chaturvedi 2014669

Table of Contents

List of Figures

Chapter 1

Introduction

In the following sections, a brief introduction and the problem statement for the work has been included.

1.1 Project Introduction

Expense tracking is something everyone does in their life. They are often done in old-school registers, or using the notes app. Some people invest in expense tracking apps, only to be limited by their platform availability and stuck with paying large amounts to application subscriptions. But, there is no proper way to track expenses, that is also free and available everywhere. This project aims to solve this problem by creating a web app that can be used to track expenses.

This project uses modern web technologies alongside a sophisticated design language and animations to deliver a clear, consistent and visually stunning User Interface and User Experience, with accessibility features to account for the differently abled.

The project is about creating a web application that will help users to track their expenses. The application will be a single page application that will be built using ReactJS, TailwindCSS, SASS(SCSS), React Spring, React Router, React Redux, Autoprefixer, PostCSS, Lodash, Axios, and JQuery, with VitePWA plugin, all bundled together using WebPack, and Vite as the build tool. The application will be hosted on Firebase Hosting.

The application will be using Firebase as a backend, Firebase Authentication for user authentication, Firebase Firestore for storing user data, Firebase Storage for storing user images, Firebase Analytics for tracking user activity, Firebase Crashlytics for tracking application crashes.

1.2 Problem Statement

The problem statement for the present work can be stated as follows:

A Cross-Platform Web App for tracking expenses and budgets, and syncing it across Devices, with ability to analyse spending patterns, view trends, gain suggestions to better manage

spending and improve spending habits, and export reports and expenses in various formats to share across services.

1.3 Objectives

The proposed work objectives are as follows:

1. To survey about the possible features and capabilities of a expense tracking platform that a user requires to effectively manage spending and budgeting tasks.
2. To design a responsive web application using modern design language and focus on user experience using design tools and web technologies.
3. To propose features that a target demographic would desire to be present in the expense tracker, and new novel ways of storing and displaying expense data.
4. To create a PWA using React.js, application routing and corresponding custom components using Typescript, Framer Motion, TailwindCSS, React Query, Sass etc.
5. To compare and contrast the user experience and feature set from user expectations versus the actual delivered product.

Chapter 2

Literature Survey/ Background

3.1 Figma Symbols and Toolkit

In order to establish a consistent visual design language across platforms, and allow for better reusability, we begin by creating a Figma Toolkit following the design ideology and specifications of our application and our vision for user experience and interface design. Figma toolkit and symbols are essential components for designers who work on digital interfaces, websites, and applications. They provide the following advantages while prototyping and wireframing our application design:

a) Consistency: Figma toolkit and symbols enable designers to maintain consistency across their designs. Designers can create and reuse assets such as buttons, icons, and forms, which ensures that the design elements are consistent throughout the project.

b) Efficiency: Figma toolkit and symbols save designers a lot of time and effort. Instead of creating new design elements from scratch, designers can use pre-existing assets, which allows them to focus on other aspects of the design.

c) Collaboration: Figma toolkit and symbols facilitate collaboration between designers and team members. The assets can be shared and updated in real-time, which ensures that everyone is working with the latest version of the design.

d) Scalability: Figma toolkit and symbols make it easy to scale designs. When working on larger projects or designs with multiple screens, designers can reuse the same assets across different screens, which saves time and ensures consistency.

e) Design Systems: Figma toolkit and symbols are essential components of a design system. Design systems are a collection of reusable components, guidelines, and best practices that enable designers to create consistent and cohesive designs.

We created a few simple usable component symbols for Navigation View and Tabs, Scaffold, Sidebars, six cohesive set of Headings alongside a base and emphasis text styles, Buttons, Radio Controls, a drop-down select element alongside its combo-dropdown counterpart, file upload input, text and number based inputs, segmented pickers, segmented text fields, long text inputs, Lists, List.Headings, List.Sections with header and footer, Toolbars for top and bottom control strip, and Stacks with spacing, margin and paddings. We maintained consistent design attributes across these components, making use of a generous amount of Gaussian Backdrop Blur, Drop Shadow and Object Lighting and Particle effects to have a clear differentiation in different layering and depth in a virtual Z-axis, and in order to provide emphasis to different elements, gaining user attention and providing a call to action.

3.2 Figma Design and Wireframe

Creating an application layout and design in Figma can be a challenging but rewarding process. Figma is a powerful digital design tool that offers many features and tools that can help us create stunning layouts and designs. We will follow the following steps to creating our application layout and design in Figma.

3.2.1: Understand the User and Business Needs

Before starting the design process, it is essential to understand the user and business needs. The application design should address the specific needs of the user while aligning with the business goals. Conducting user research and gathering requirements can help us understand the user needs, pain points, and expectations. This information will help in designing an application that meets user needs while fulfilling the business requirements.

3.2.2: Create a Wireframe

Once the user and business needs are understood, we can begin to create a wireframe. Wireframes are a low-fidelity representation of the design, focusing on the layout, structure, and functionality of the application. Figma offers several wireframe templates that we can use

to create a basic structure for the application. The wireframe should include all the essential elements, such as the navigation, buttons, and content areas.

3.2.3: Design the Layout

With the wireframe as a foundation, we can start designing the layout of the application. The layout should be designed with the user in mind, making sure that the most critical information is easy to find and accessible. We can then use Figma's grid system to ensure that the layout is consistent and aligned. The layout should also be designed with responsiveness in mind, making sure that it looks good on different devices and screen sizes.

3.2.4: Add Visual Design Elements

Once the layout is designed, we can now start adding visual design elements such as color, typography, and images. The visual design should align with the branding guidelines and be consistent throughout the application. Figma offers many features that can help us create a cohesive visual design, such as styles and components.

3.2.5: Prototype and Test

With the layout and visual design in place, we can create a prototype of the application. The prototype should allow the user to interact with the application and test its

functionality. Figma's prototype feature allows the us to create clickable prototypes and test them on different devices and screen sizes.

Henceforth, designing an application layout and design in Figma requires a thorough understanding of the user and business needs, and can be used to easily perform the tasks of wireframing, designing the layout, adding visual design elements, and prototyping and testing. Figma offers many features that can help the us create a stunning and responsive application that meets the user's needs and aligns with the business goals.

3.3 React Components, Contexts and Hooks

Instead of randomly creating components in-place for reusable elements, creating React components, contexts, and hooks for an application can significantly improve its functionality, reusability, and maintainability. In this essay, we will explore the steps involved in creating React components, contexts, and hooks for our application.

3.3.1 Define the Component's Purpose

The first step in creating a React component is to define its purpose. A component should be designed to solve a specific problem or implement a specific feature. Defining the component's purpose helps in making the component reusable and easily maintainable.

3.3.2 Create the Component

Once the component's purpose is defined, we can create it using JSX, which is a syntax extension to JavaScript that allows us to write HTML-like syntax for elements and components. The component should be designed to take in props, which are arguments passed to a component. Props allow the component to be reusable and customizable.

3.3.3 Create the Context

Context is a feature in React that allows us to share data between components without passing props manually at every level. Creating a context involves defining the data to be shared and creating a provider and consumer for the context. The provider is responsible for providing the data to the components, and the consumer is responsible for consuming the data.

3.3.4 Create the Hook

Hooks are a feature in React that allows us to use state and other React features in functional components. Hooks provide a way to share stateful logic between components without using higher-order components or render props. Creating a hook involves defining the state and logic to be shared and exporting the hook to be used in other components.

3.3.5 Implement the Components, Contexts, and Hooks in the Application

Once the components, contexts, and hooks are created, they can be implemented in the application. The components can be used in different parts of the application, and the contexts and hooks can be used to share data and logic between the components. Implementing the components, contexts, and hooks in the application involves importing them into the necessary components and using them to improve the functionality and reusability of the application.

Creating React components, contexts, and hooks can significantly improve the functionality, reusability, and maintainability of an application. The steps involved in creating React components, contexts, and hooks include defining the component's purpose, creating the component, creating the context, creating the hook, and implementing them in the application. By following these steps, we can create a robust and scalable application using React.

3.4 React Routing and Protected Routes

3.4.1 React Routing

React Routing is a process of creating and managing different views or pages in a single-page application. It allows the user to navigate between different components without refreshing the page. React Router is a popular library for implementing routing in React applications.

The first step in implementing React Routing is to define the routes in the application. Routes are defined using the Route component provided by the React Router library. Each Route component is associated with a specific URL path and renders a specific component when the URL matches the path.

For example, suppose we want to create a route for a login page. In that case, we can define a Route component with the path "/signin" and render the Login component when the URL matches the path.

Once the routes are defined, we can use the Link component provided by React Router to create links between the different components. Links allow the user to navigate between different components without refreshing the page.

3.4.2 Protected Routes

Protected Routes are routes that require authentication before accessing them. In other words, they are routes that should only be accessed by authorized users. Implementing protected routes involves creating a higher-order component that checks whether the user is authenticated before rendering the component.

The first step in implementing a Protected Route is to create an authentication service that checks whether the user is authenticated. This can be done using tokens or cookies, which are stored in the user's browser.

Next, we can create a higher-order component that wraps around the component that requires authentication. The higher-order component checks whether the user is authenticated, and if not, redirects the user to the login page. If the user is authenticated, the higher-order component renders the protected component.

For example, suppose we want to create a Protected Route for a dashboard component. In that case, we can create a higher-order component that checks whether the user is authenticated and redirects the user to the login page if not. If the user is authenticated, the higher-order component renders the Dashboard component.

React Routing and Protected Routes are essential concepts in front-end web application development that help in building robust and secure applications. React Router is a popular library for implementing routing in React applications, and Protected Routes can be implemented using higher-order components. By using these concepts, we can create a seamless user experience and ensure the security of our applications.

3.5 React Pages with Layout and Route Navigation and Elements

Creating a React app with different pages using components created earlier involves implementing React routing to navigate between the pages and rendering the previously created components on the respective pages. Here are the steps involved:

Step 1: Set Up a New React App: The first step is to create a new React app using create- react-app or any other tool of your choice. Once the app is set up, install the necessary dependencies such as React Router and any other components required for the app.

Step 2: Create the Required Components: The next step is to create the components required for the app. These components should be designed to solve specific problems or implement specific features. Each component should be designed to be reusable and customizable by taking in props.

Step 3: Implement React Routing React Routing is essential for navigating between the different pages of the app. To implement routing, create a routes.js file that defines the different routes and the respective components to be rendered when the routes are accessed.

For example, if you have a Home component and a Sign In component, you can define the routes in the routes.js file as follows:

```
import React from 'react';
import { Route, Switch } from 'react-router-dom';
import Home from './Pages/Home';
import SignIn from './Pages/SignIn;
const Routes = () => {
  return ( <Switch>
            <Route exact path="/" component={requireAuth(Home)} />
            <Route exact path="/signin" component={SignIn} />
      </Switch> );
};
11


export default Routes;
```

This code defines two routes, one for the Home component and one for the SignIn component.

Step 4: Implement Navigation Once the routes are defined, implement navigation by adding links to the different pages of the app. This can be done using the Link component provided by React Router.

For example, to add a link to the Home component, you can add the following code to the Home component:

```
import React from 'react';
import { Link, Outlet } from 'react-router-dom'; import {
  Scaffold,
  Page,
  List,
  Chip,
  Stack
} from "../components"
import { Sidebar } from "../fragments"
import AppContext from "../contexts/AppContext"
export default function Home() {
const { fetchTransactions } = useContext(AppContext) return (
    <Scaffold sidebar={<Sidebar />}>
      <Page>
<List.View>
{ fetchTransactions().map(transaction => { <List.Card
leading={<Icon data={ transaction.icon } /> } heading={
`${transaction.currencySymbol}${transaction.amount}` }
            Subheading=(transaction.label)
```

```
            trailing={
              <Stack.Horizontal>
                <Chip
leading=(<Icon data={transaction.category.icon}/>)
color={transaction.category.color} >
                  { transaction.category.label }
                </Chip>
                <Chip
                  leading=(
<Icon data={fetchWallet(transaction.wallet).icon}/> )
color={fetchWallet(transaction.wallet).icon} )>
{ fetchWallet(transaction.wallet).name } </Chip>
</> )/>
}) } </Page>
      <Outlet />
    </Scaffold>
); };
```

This code adds the Home component.

Step 5: Render Components on Respective Pages Finally, render the previously created components on the respective pages. This can be done by importing the components into the routes.js file and passing them as props to the Route components.

For example, to render the Home component on the home page, add the following code to the routes.js file:

```
import React from 'react';
```

```
import { Route, Switch } from 'react-router-dom'; import Home from
'./components/Home';
import SignIn from './components/SignIn;
const Routes = () => {
  return (
    <Switch>
      <Route
      <Route
</Switch> );
};
exact path="/" component={Home} />
exact path="/signin" component={SignIn} />
export default Routes;
```

This code passes the Home component as a prop to the Route component for the home page.

Creating a React app with different pages using components created earlier involves implementing React routing to navigate between the pages and rendering the previously created components on the respective pages. By following these steps, we can create a robust and scalable front end using React.

Chapter 3

Software Design

3.1 Backend

This app uses Firebase as its backend to provide user authentication, database and serverless functions, and Vercel’s cloudfront for hosting the Next.js webapp.

Leveraging the immense capabilities of Firebase, this app harnesses a wide array of features such as Cloud Firestore, authentication, and cloud functions to deliver an unparalleled experience. By utilizing Cloud Firestore, the app securely stores and organizes essential data, including expenses, wallet details, and user information, in a highly scalable and easily accessible manner. With the robust user authentication provided by Firebase, users can confidently safeguard their personal data while enjoying a seamless login and registration process.

Moreover, the app makes intelligent use of Firebase's cloud functions, enabling it to perform complex calculations and manipulate database values in real-time. This dynamic functionality ensures that users receive accurate and up-to-date financial insights, empowering them to make informed decisions regarding their expenses and budgets.

Additionally, Firebase's synchronization capabilities play a pivotal role in keeping the app's data in perfect harmony across multiple devices and platforms. Changes made to expenses, wallet details, or user information are seamlessly propagated, guaranteeing a consistent and synchronized experience for users, regardless of the device they use to access the app.

Furthermore, Firebase's comprehensive suite of features goes beyond storing and syncing data. The app takes advantage of other functionalities provided by Firebase, such as real-time notifications, analytics, and remote configuration, to enhance the overall user experience. From receiving timely notifications about budget thresholds to gaining valuable insights through

analytics, users can leverage these additional features to optimize their financial management practices and make the most of the app's capabilities.

In summary, by harnessing the power of Firebase's Cloud Firestore, authentication, cloud functions, and various other functionalities, this app sets a new standard in expense tracking, delivering a seamless, secure, and feature-rich solution that empowers users to take control of their finances with ease and confidence.

3.2 Frontend

Incorporating the cutting-edge technologies of Next.js 13 and React server components, this expense tracking web app takes user experience to new heights by seamlessly integrating server-side rendering (SSR). By leveraging the power of Next.js 13's React server components, the app delivers blazing-fast initial page loads, ensuring that users can swiftly access their expense tracking dashboard and valuable financial insights.

Furthermore, the app embraces a file system-based routing approach, allowing for intuitive and organized routing management. This efficient routing system simplifies the development process, making it easier to navigate between different pages and components while maintaining a clean and maintainable codebase.

To optimize performance and enhance development productivity, the app utilizes the turbo-packed module bundler. This state-of-the-art bundler intelligently analyzes dependencies, enabling lightning-fast module bundling, reducing load times, and delivering an optimized user experience.

Native tailwind CSS support further enhances the app's styling capabilities. By leveraging the power and flexibility of Tailwind CSS, the app can effortlessly create custom and responsive designs, ensuring a visually appealing and consistent interface across different devices and screen sizes.

In order to streamline the loading process and improve overall performance, the app incorporates lazy loading of fonts. This technique ensures that only the necessary fonts are loaded, minimizing unnecessary network requests and enhancing the app's loading speed.

Additionally, the app prioritizes optimized image rendering. By employing best practices for image optimization, such as lazy loading, responsive image loading, and utilizing appropriate image formats, the app ensures that images are delivered efficiently, reducing bandwidth usage and improving overall performance.

To optimize performance and enhance development productivity, the app leverages the turbo-packed module bundler, a state-of-the-art bundling tool. With its intelligent analysis of dependencies, the turbo-packed bundler ensures lightning-fast module bundling, significantly reducing load times and delivering an optimized user experience. This optimization leads to improved performance and faster interactions, contributing to a seamless expense tracking workflow.

Native tailwind CSS support further amplifies the app's styling capabilities, empowering developers to effortlessly create custom and responsive designs. By harnessing the power and flexibility of Tailwind CSS, the app achieves a visually appealing and consistent interface across various devices and screen sizes. This native support enhances the efficiency of styling implementation, resulting in a polished and engaging user interface.

In order to streamline the loading process and enhance overall performance, the app incorporates lazy loading of fonts. By implementing this technique, the app selectively loads fonts only when necessary, minimizing unnecessary network requests and significantly improving loading speed. This optimization ensures a swift and responsive user experience, eliminating unnecessary delays and enhancing the app's overall performance.

Additionally, the app prioritizes optimized image rendering by adhering to best practices in image optimization. Techniques such as lazy loading, responsive image loading, and utilizing

appropriate image formats are employed to deliver images efficiently. By reducing bandwidth usage and optimizing image loading, the app achieves improved performance, faster page rendering, and a more responsive user interface.

By leveraging Next.js 13 with React server components, file system-based routing, turbo-packed module bundling, native Tailwind CSS support, lazy loading of fonts, and optimized image rendering, this expense tracking web app delivers an exceptional user experience characterized by fast loading times, seamless navigation, visually appealing designs, and overall superior performance.

Chapter 4

Requirements and Methodology

4.1 Hardware Requirements

- A computer with a Multi-Core Processor (Core i5 4550 or Later)
- A 3D Rendering capable GPU with support for OpenGL and WebGL 3.0 or later (GTX 770 or later, GTX 1080 or higher for Hi-Dpi and Hi-Resolution monitors)
- 8GB DDR2 RAM or greater (16GB DDR4 Preffered)
- An Internet Connection with speeds of 1Mbps or greater (30Mbps preferred)

4.2 Software Requirements

- A JavaScript-enabled Web Browser (Safari 12, Chrome 110, Edge Chromium 110, Firefox 112 or later preferred, Internet Explorer not compatible)
- macOS Mojave or Later for Mac app.
- Windows 10 or Later for Windows app.
- Android 10 or later for Android app.
- iOS 12 or later for iOS app.

Chapter 5

Coding / Code Templates

5.1 Expense Data Model

```
import { Timestamp } from "firebase/firestore";

export type Expense = {
  id: string;
  userId: string;
  amount: number;
  currency: number;
  title: number;
  recurring: boolean;
  recurringDurationDays: number;
  categoryId: string;
  categoryBudgetRemaining: number;
  date: Timestamp;
  merchant: string; // Cache in user doc
  walletId: string;
  walletRemaining: number;
  paymentMethodIds: [string];
  splitWith: [string];
  remaining: number;
  notes: string;
};
```

5.2 Wallet Data Model

```
export type WalletProps = {
  id: string;
  icon: string;
  title: string;
  color: string;
```

```
  balance: number;
  description: string;
  linkedAccountName: string;
  linkedDebitCards: string[];
  startingBalance: number;
  transactions: string[];
  minimumBalance: number;
  minimumPreferredBalance: number;
  isCredit: boolean;
  creditLimit: number;
  creditLimitHistory: string[];
};
```

5.2 Credit Report Data Model

```
export type CreditReport = {
  id: string;
  date: string;
  creditScore: number;
  creditBureau: string;
  factorOnTimePayments: number;
  factorCreditUtilization: number;
  factorCreditAge: number;
  factorCreditMix: number;
  factorNewCreditAccounts: number;
};
```

Chapter 6

Testing

6.1 Cloud Firestore Tests

```
import firebase from "firebase/app";
import "firebase/firestore";
import { render, waitFor } from "@testing-library/react";
import { FirestoreProvider, useFirestore } from "react-firestore";
import { createMemoryHistory } from "history";
import { Router } from "react-router-dom";

// Mock Firebase configuration and initialize Firestore
jest.mock("firebase/app", () => {
  return {
    initializeApp: jest.fn(),
    firestore: jest.fn(() => ({
      collection: jest.fn(),
      doc: jest.fn(),
      get: jest.fn(),
      add: jest.fn(),
      set: jest.fn(),
      update: jest.fn(),
      delete: jest.fn(),
    })),
  };
});

// Mock FirestoreProvider and useFirestore hook
jest.mock("react-firestore", () => {
  return {
    FirestoreProvider: jest.fn(({ children }) => children),
    useFirestore: jest.fn(() => firebase.firestore()),
```

```
  };
});

// Mock React Router history
const history = createMemoryHistory();

// Example component that interacts with Firestore
function MyComponent() {
  const firestore = useFirestore();

  const addData = async (data) => {
    await firestore.collection("myCollection").add(data);
  };

  const getData = async () => {
    const snapshot = await
firestore.collection("myCollection").get();
    return snapshot.docs.map((doc) => doc.data());
  };

  // ... other CRUD operations

  return (
    <div>
      <button onClick={() => addData({ name: "John" })}>Add
Data</button>
      <button onClick={getData}>Get Data</button>
      {/* ... render other components */}
    </div>
  );
}

// Unit test for Firebase Firestore CRUD operations
```

```
describe("Firebase Firestore CRUD operations", () => {
  it("should add data to Firestore collection", async () => {
    const { getByText } = render(
      <FirestoreProvider firebase={firebase}>
        <Router history={history}>
          <MyComponent />
        </Router>
      </FirestoreProvider>
    );

    const addButton = getByText("Add Data");
    addButton.click();

    await waitFor(() => {
      expect(firebase.firestore().collection).toHaveBeenCalledWith(
        "myCollection"
      );
      expect(firebase.firestore().add).toHaveBeenCalledWith({ name:
"John" });
    });
  });

  it("should get data from Firestore collection", async () => {
    const mockSnapshot = {
      docs: [
        { data: () => ({ name: "John" }) },
        { data: () => ({ name: "Jane" }) },
      ],
    };

    firebase.firestore().collection.mockReturnValue({
      get: jest.fn(() => Promise.resolve(mockSnapshot)),
    });
```

```
    const { getByText } = render(
      <FirestoreProvider firebase={firebase}>
        <Router history={history}>
          <MyComponent />
        </Router>
      </FirestoreProvider>
    );

    const getDataButton = getByText("Get Data");
    getDataButton.click();

    await waitFor(() => {
      expect(firebase.firestore().collection).toHaveBeenCalledWith(
        "myCollection"
      );
      expect(firebase.firestore().get).toHaveBeenCalled();
      expect(getByText("John")).toBeInTheDocument();
      expect(getByText("Jane")).toBeInTheDocument();
    });
  });
});
```

Chapter 7

Result

7.1 Screenshots

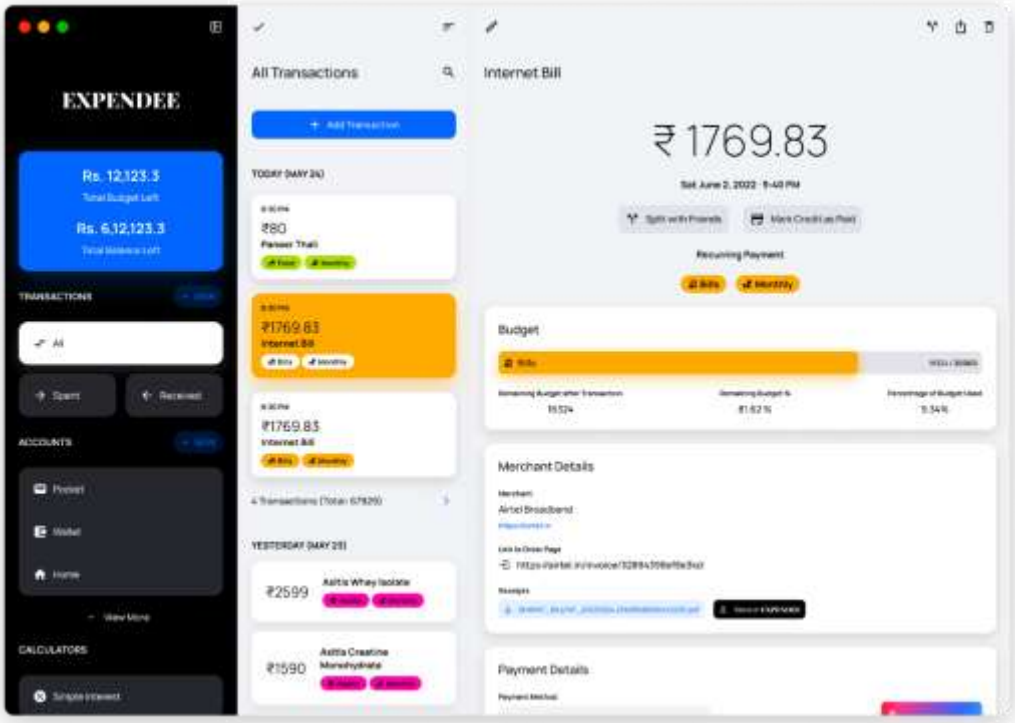

Fig 1.1 Transaction Details

Fig 2. View Options Menu

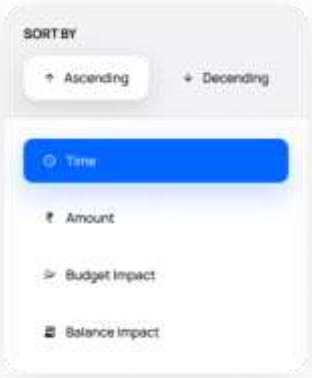

Fig 3. Sort Menu

Fig 4. Group Menu

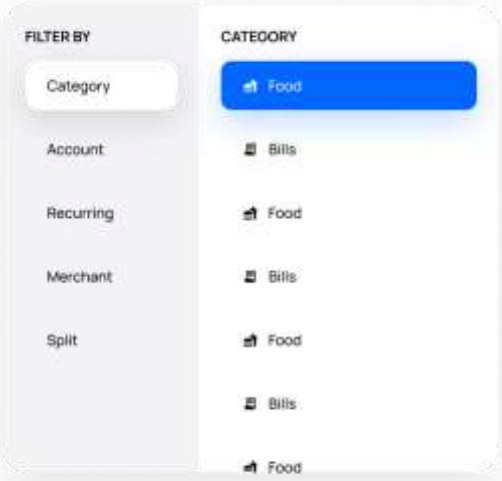

Fig 5. Filter Menu

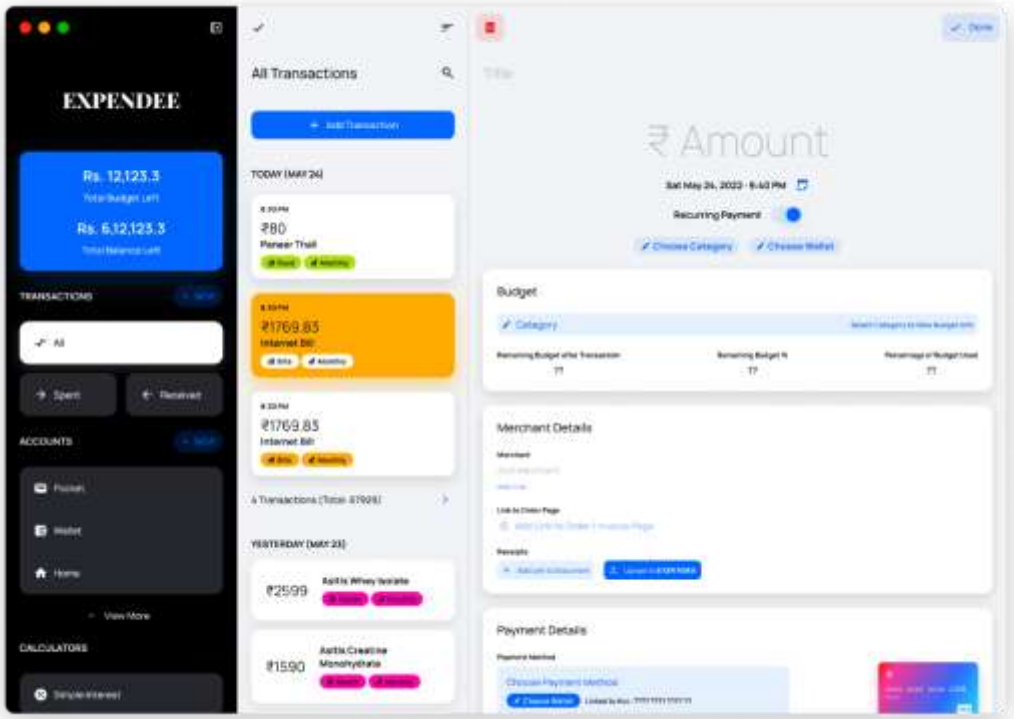

Fig 6. Create a New Expense

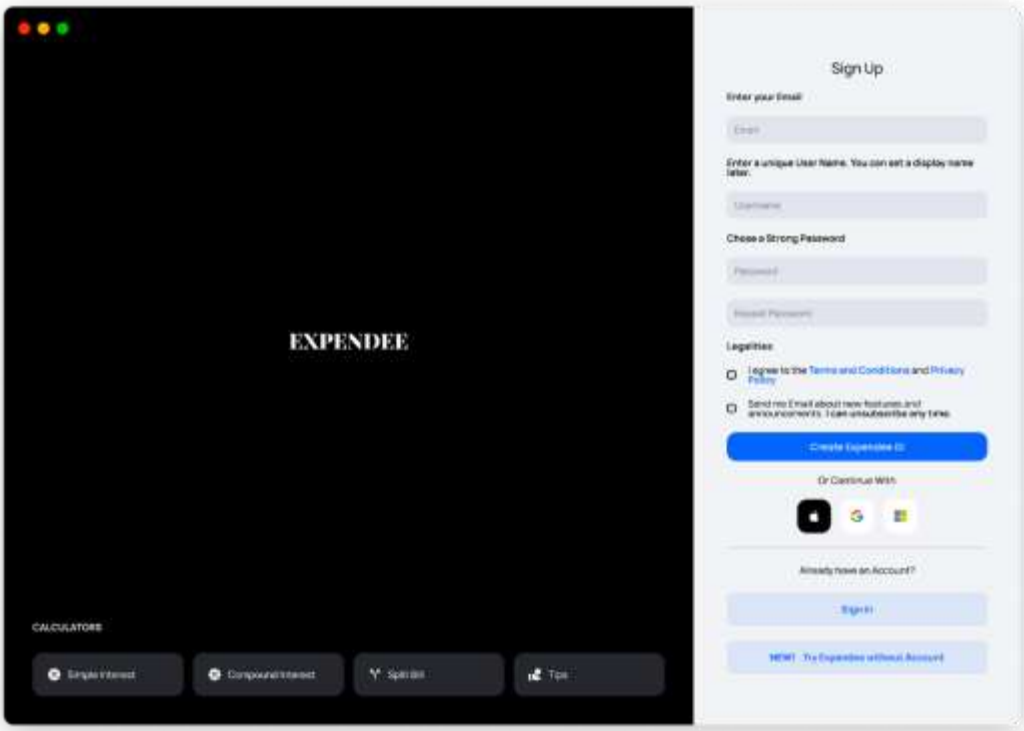

Fig 7. Sign In Page

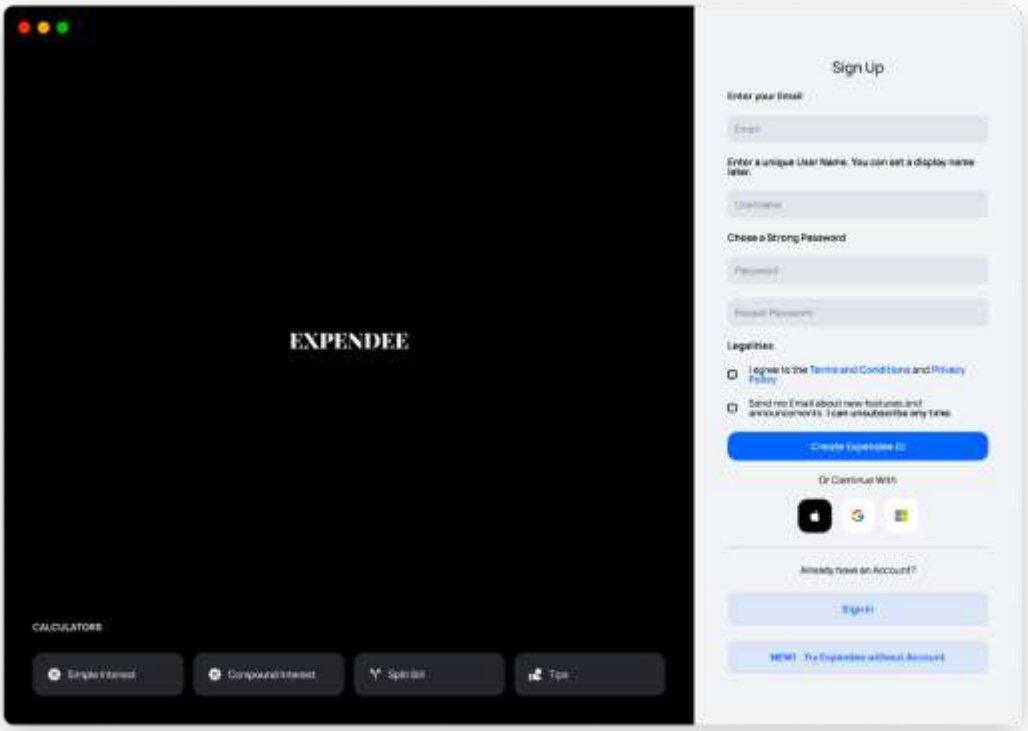

Fig 8. Sign Up Page

7.2 Result Summary

We developed an innovative Cross-Platform Web Application utilizing state-of-the-art web technologies to facilitate efficient expense and budget tracking. This advanced solution not only ensures seamless synchronization across multiple devices but also offers comprehensive analysis of spending patterns, enabling users to visualize trends and make informed decisions. Additionally, it provides invaluable suggestions for better managing expenses and improving spending habits. Furthermore, our application allows users to effortlessly export reports and expenses in diverse formats, facilitating easy sharing across various services, thereby enhancing collaboration and convenience.

We harnessed the capabilities of Next.js, a cutting-edge framework, to develop a full-stack web application that seamlessly combines the latest features of React. By extending React with Next.js, we were able to leverage its server-side rendering (SSR) capabilities, enabling us to

deliver pre-rendered pages for improved performance and enhanced search engine optimization (SEO).

In addition, we integrated powerful Rust-based JavaScript tooling into our development process, which resulted in lightning-fast builds. By utilizing the Rust ecosystem, we experienced significant performance gains, enabling us to deliver a highly responsive and efficient web application. The Rust-based JavaScript tooling also provided us with robust error handling and advanced debugging features, ensuring a smooth development experience.

One notable feature provided by Next.js is automatic code splitting, which optimizes the loading of JavaScript bundles by dynamically loading only the necessary code for each page. This results in faster initial page loads and improved overall performance, especially for larger applications.

Next.js also offers built-in support for serverless functions, allowing us to effortlessly create and deploy serverless APIs to handle various backend functionalities. This seamless integration streamlined our development process and empowered us to build scalable and flexible web applications.

Another noteworthy feature of Next.js is its powerful routing system. With dynamic routing capabilities, we were able to create dynamic and personalized pages based on user input or data from external sources. This enabled us to deliver customized experiences and implement complex routing logic with ease.

Overall, by utilizing Next.js and extending the latest React features, combined with the integration of powerful Rust-based JavaScript tooling, we achieved an exceptional development experience and delivered a performant, feature-rich web application.

We made a strategic decision to leverage the exceptional capabilities of Firebase for our project. Firebase played a pivotal role in enhancing our application by providing reliable and scalable cloud storage, seamless cloud functions, and robust user authentication.

By utilizing Firebase's cloud storage, we were able to securely store and retrieve data, ensuring the integrity and accessibility of our users' information. The scalable nature of Firebase's storage allowed us to accommodate growing data needs without compromising performance or reliability.

In addition, Firebase's cloud functions empowered us to execute server-side logic and perform complex computations effortlessly. This enabled us to offload time-consuming tasks from the client-side, resulting in a smoother user experience and improved application performance.

Firebase's user authentication capabilities provided a seamless and secure login experience for our users. With features such as email and password authentication, social media login integration, and multi-factor authentication, we were able to ensure the privacy and security of our users' accounts while offering them a convenient and hassle-free authentication process.

Furthermore, Firebase's extensive set of development tools and APIs simplified our development workflow, allowing us to focus on building core features rather than dealing with the intricacies of backend infrastructure. This streamlined approach accelerated our development timeline and enabled us to deliver a robust and feature-rich application to our users.

Overall, leveraging Firebase for cloud storage, cloud functions, and user authentication proved to be a wise decision. Its comprehensive suite of services provided us with a solid foundation, allowing us to create a secure, scalable, and efficient application while reducing development complexities. Firebase undoubtedly played a crucial role in elevating our project and ensuring a seamless user experience.

Chapter 8

Conclusion and Future Work

We conclude this project with its current state as a complete solution for managing budgets and tracking expenses, and consider this as a useful and necessary tool for helping someone manage finances better, and help build their good financial habits.

We are always open to user feedback and take a pride in creating a program that helps users solve their problems. One of the best ways to do that is to collect feedback and feature or quality requests from the userbase, and intent to spend most of the development time completing users requests for developments and fixes.

The project, in its current form is sufficiently developed to be released as a complete package to the public. It has been successfully used by a few testers and their feedback has been collected and acted upon successfully. A few of the features, such as uploading relevant files to each transaction is still missing, as it has been intentionally been held back to be later released under a paid plan that the user can attain by paying a monthly or yearly subscription, with different plans of different amounts that give user access to larger storge space, file types and features such as encrypting or sharing those files across users.

Another planned feature release is for an official page listing on Apple's iOS and Mac App Store, the Microsoft Store for Windows, a Flatpak version for Linux, and Play Store release for Android.

There are a vast number of opportunities that would allow us to take this project further in terms of technical and business aspects, and a roadmap for some of the work has already been planned.

References

[1] React.js Documentation. Accessed 26th September 2022. Available under MIT License.
https://reactjs.org/docs

[2] Firebase Documentation. Accessed 4th October 2022. Available under Creative Commons 2.0 License.
https://firebase.google.com/docs/

[3] TailwindCSS Documentation. Accessed 27th September 2022. Available under MITLicense.
https://tailwindcss.com/docs/installation

[4] React Design Patterns and Best Practices
Packt Publishing, ISBN: 978-1-78953-017-9

[5] Full Stack React
Packt Publishing, ISBN 978-1-83921-541-4