Finantes

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Team Members:
Member 1 -
zaidr@iastate.edu

Member 2 -
jayson04@iastate.edu

May 11, 2025

1. Introduction

Overview of the problem, motivation, users, and goals. State whether the project is original or inspired.

In developing Finantes, our goal was to address several common challenges people face when managing their finances– particularly those who enjoy setting multiple financial goals. Often, individuals have to juggle between applications to figure out where they stand financially, which can be both cumbersome and overwhelming. Finantes solves this problem by gathering all your essential financial information into one convenient web-application. This makes managing finances simpler and more efficient which will allow users to easily take control of their finances. There are many other financial applications out there, but Finantes is our original take on simplifying finances.

2. Project Description

Explain major features, user flow, CRUD operations and entities affected.

Users will start on the landing page where they will see our selection of different tiers(Demo, Basic, and Premium). Users can check out our Demo by clicking on the “Try Demo” button which will take them to a sample/example of the different tools we provide to the users.

In the navbar on the top right corner users can find a log-in and sign-up button. By signing up and creating an account, users will be granted the Basic tier which allows them to do: Full Budget Planning, Investment Tracking, and CSV Import and Export.

On the dashboard page, users will be able to access the settings page by clicking the “Settings” button on the top right. Users will be able to set custom themes, add and edit their personal information, set passwords, set their preferences, and change what tier they want.

If users want to go premium they can do so by clicking the “Go Premium” button on the landing page or by changing their active tier in the settings. The Premium tier offers real-time market data, advanced analytics, and customizable themes.

Users can also browse credit card options by clicking the “Browse Credit Cards” button on the landing page. There, the user will be able to filter credit cards by company and view the different benefits each card provides.

Lastly, the user can view our Team Information page by selecting the “About Us” button which will pull up our name and contact information.

3. File and Folder Architecture

Frontend Architecture

- frontend/
  - src/
    - assets/ Images and external data
    - components/ React components
    - pages/ React components
    - main.jsx, Main react component that connects everything
    - AuthContext.jsx Context with custom state hook
    - DataContext.jsx Context with custom state hook
- backend/
  - routes/ Part of the servers routes exported as part of a router
  - utils/ JWT authentication and creation, DB connection
  - .env Environment variables needed
  - server.js Main server component that connects everything
- Documents/ Planning sketches, Software Architecture Document, final report, and video.

The backend will give the frontend JWT tokens to use with each request. For each request, the custom middleware will take the token, decrypt it, and put it into req.user. This way, the frontend does not need to send which user it is each time, nor need verification. The JWT access token expires every few minutes, with the refresh token expiring every few days. The user's information will be displayed to them throughout the application

Backend Architecture

The backend will use a RESTful API architecture. The following are the current endpoints we will utilize:

- api/
  - auth/
    - register (POST)
    - login (POST)
    - logout (POST)
    - refresh-token (POST)
    - verify (GET)
  - budgets/ (GET, POST)
    - _id (PUT, DELETE)
  - goals/ (GET, POST)
    - _id (PUT, DELETE)
  - income/ (GET, POST)
    - _id (PUT, DELETE)
  - investments/ (GET, POST)
    - _id (PUT, DELETE)
  - regular-expenses/ (GET, POST)
    - _id (PUT, DELETE)
  - upcoming-expenses/ (GET, POST)

    - _id (PUT, DELETE)
  - users/ (GET)
    - update (PUT)
    - password (PUT)
    - email (PUT)

Each authentication (auth) category will send HTTP-only cookies to the client except for verify. For the users category, we take the cookies, break them back down into userID and email, and we execute the request. The password endpoint requires the current password to be entered. Each of the endpoints for the other 6 categories has GET, POST, PUT, DELETE requests done the same.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

Starting off on the backend, the request is made to register from /api/auth/register, where a user document is created. First, we use bcrypt to hash the user's password, then we generate the userID using UUID version 4. We then store this information, along with the user's personal information and a theme and color defaulting to light and blue, respectively. We then create a user object, which we pass to JWT to create two different tokens: the access token and the refresh token. We then add the refresh token to the document and send it all to the database. On successful completion, we send HTTP Only cookies to the frontend with same-site set to strict. The access token is set to expire every couple of minutes, while the refresh token expires every week. On the frontend side, the Access token is used to verify that the user is allowed to make a request, and sends the userID with every request to get the user-specific information. The login route uses bcrypt to compare the hashed password with the input password, pushes a new refresh token, and sets the cookies.

On the frontend, the user manipulates their data in three main ways: registration, editor, and settings. The user registers an account on the /auth?tab=regsister endpoint, which will have them input their name, email, phone number, and password. They will then be sent to the dashboard, where the navbar will have updated to show new options: Edit Data and Settings. On the Edit Data page, there are tabs that allow users to manipulate each category they saw on the dashboard. The initial load of this page is a GET request, the same as the dashboard, and the users will be able to do POST, PUT, and DELETE requests here. In the settings page, there are similar tabs for each category of settings. Users interact with various inputs such as radio buttons, file uploads, and drop-downs. With the exception of the Data Management section in the preferences tab, each part of the Settings page updates the user collection. The Data Management section has an Import, Export, and Clear All section, which allows users to bulk manipulate their data. The exported data does not include items from the user's collection and is exported in a JSON file.

4.2. React Component Structure

Inside main.jsx is where all the routes are, as well as where the root is created. Typically, you would have StrictMode as the outermost layer, then just have RouterProvider inside of it, but we created our own providers and context. After StrictMode, we have AuthProvider, which is used for Authentication and to pass down context information about Authentication, such as the user information, if the check for tokens is loading, and if the user is logged in. There are also some helpful methods inside, such as checkAuth, logout, and refreshToken. Next, we have the

DataProvider, which is used to fetch the user data from 6 of the collections. It will sanitize all the data, and it will pass down its context to everything in the router. After the built-in router components, we have our root layout. The root layout adds the navigation bar and the outlet, which lets all the other routes exist simultaneously with the navigation bar. The navigation bar will update dynamically when the location pathname is changed, changing the buttons on the left-hand side along with adding user status. After all of those components, we get to the main pages. Each page has its own pathname tied to it, along with more components that are split up to make code easier to read. These components are split mainly by tabs, such as each settings page having a different component per tab, and authentication and editor are the same. There are also components split by layout on the page. The best example of this is in the dashboard, where each card has its own component.

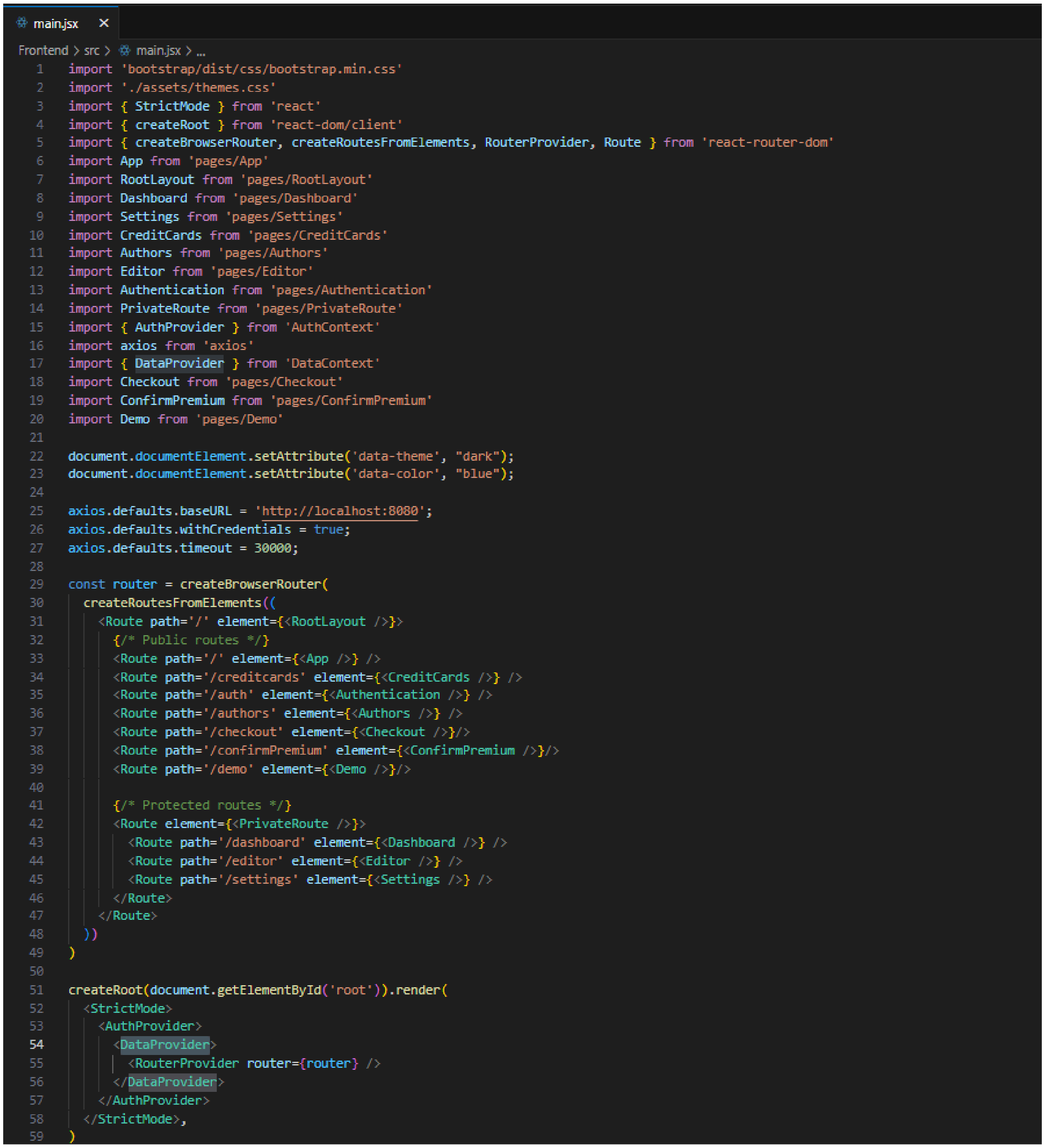
4.3. Database Interaction

We use MongoDB Atlas to store all the information. The user's profile information, including the email, userID, password, active refresh tokens, name, phone number, theme, color, and currency type is stored in the user collection. The user collection also has 5 indexes, 4 to make lookups faster and 1 to kill expired refresh tokens. Every collection has the userID index for a fast lookup, and _id index for unique IDs. There are also strict validation rules on the user collection, making sure that each field is filled out before the document is added, with correct typing.

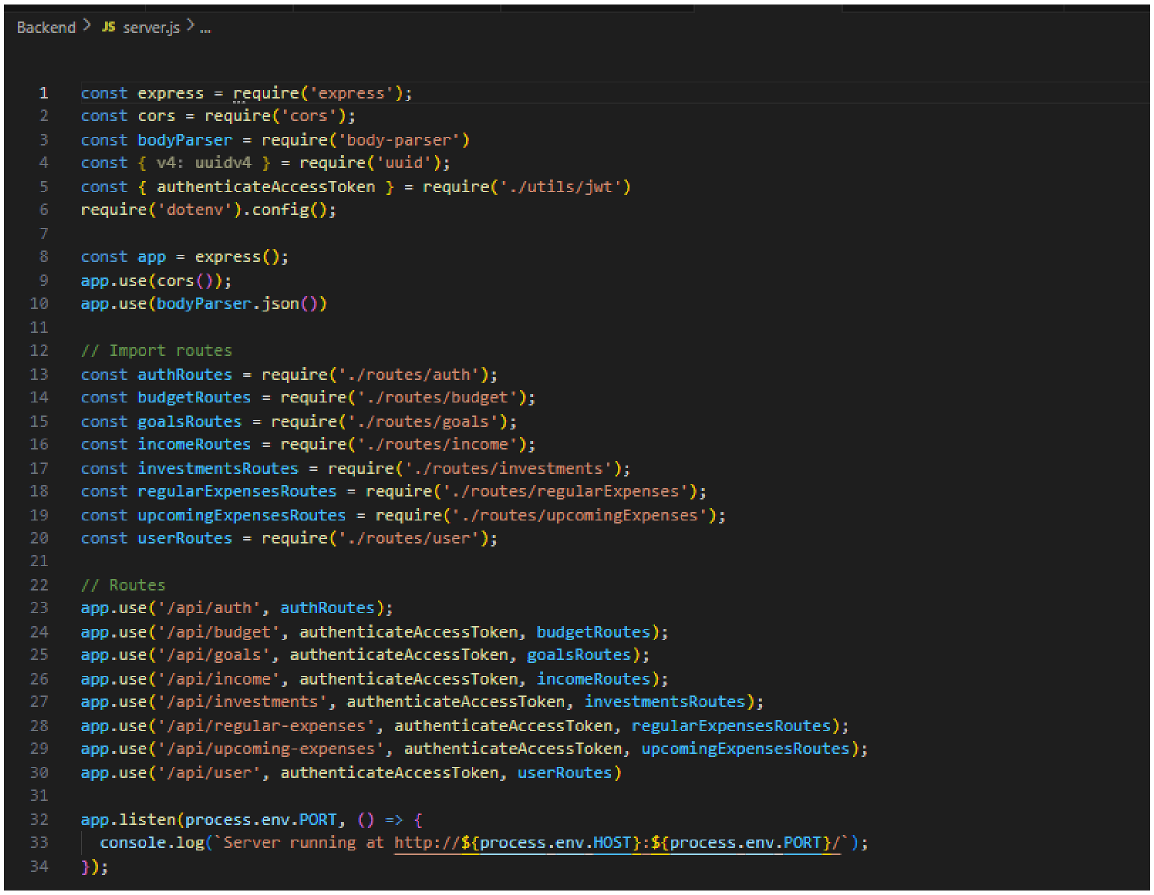
Each collection, except the credit card collection, has a userID field, storing the user's generated UUID to connect back to them when they want to access their data. These collections include the budget, goals, income, investments, regular expenses, and upcoming expenses. Each of them has relevant fields, such as target amounts, dates, shares, values, types, and more.

The credit card collection is used on the credit card viewing page. It is not protected, so anyone can use it whether they have an account or not. It includes a name, company, three to four benefits, and an image URL.

4.4. Code Snippets

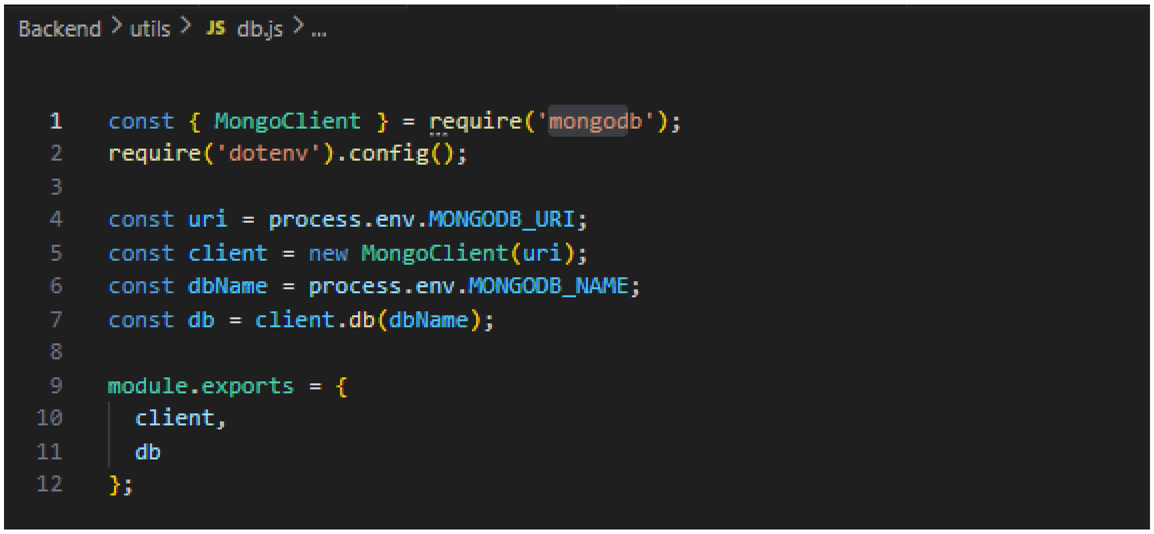


```
import 'bootstrap/dist/css/bootstrap.min.css'
import './assets/themes.css'
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import { createBrowserRouter, createRoutesFromElements, RouterProvider, Route } from 'react-router-dom'
import App from 'pages/App'
import RootLayout from 'pages/RootLayout'
import Dashboard from 'pages/Dashboard'
import Settings from 'pages/Settings'
import CreditCards from 'pages/CreditCards'
import Authors from 'pages/Authors'
import Editor from 'pages/Editor'
import Authentication from 'pages/Authentication'
import PrivateRoute from 'pages/PrivateRoute'
import { AuthProvider } from 'AuthContext'
import axios from 'axios'
import { DataProvider } from 'DataContext'
import Checkout from 'pages/Checkout'
import ConfirmPremium from 'pages/ConfirmPremium'
import Demo from 'pages/Demo'

document.documentElement.setAttribute('data-theme', "dark");
document.documentElement.setAttribute('data-color', "blue");

axios.defaults.baseURL = 'http://localhost:8080';
axios.defaults.withCredentials = true;
axios.defaults.timeout = 30000;

const router = createBrowserRouter(
  createRoutesFromElements((
    <Route path='/' element={<RootLayout />}>
      {/* Public routes */}
      <Route path='/' element={<App />} />
      <Route path='/creditcards' element={<CreditCards />} />
      <Route path='/auth' element={<Authentication />} />
      <Route path='/authors' element={<Authors />} />
      <Route path='/checkout' element={<Checkout />}/>
      <Route path='/confirmPremium' element={<ConfirmPremium />}/>
      <Route path='/demo' element={<Demo />}/>

      {/* Protected routes */}
      <Route element={<PrivateRoute />}>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/editor' element={<Editor />} />
        <Route path='/settings' element={<Settings />} />
      </Route>
    </Route>
  ))
)

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <AuthProvider>
      <DataProvider>
        <RouterProvider router={router} />
      </DataProvider>
    </AuthProvider>
  </StrictMode>,
)
```



Backend > server.js > ...

```javascript
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser')
const { v4: uuidv4 } = require('uuid');
const { authenticateAccessToken } = require('./utils/jwt')
require('dotenv').config();

const app = express();
app.use(cors());
app.use(bodyParser.json())

// Import routes
const authRoutes = require('./routes/auth');
const budgetRoutes = require('./routes/budget');
const goalsRoutes = require('./routes/goals');
const incomeRoutes = require('./routes/income');
const investmentsRoutes = require('./routes/investments');
const regularExpensesRoutes = require('./routes/regularExpenses');
const upcomingExpensesRoutes = require('./routes/upcomingExpenses');
const userRoutes = require('./routes/user');

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/budget', authenticateAccessToken, budgetRoutes);
app.use('/api/goals', authenticateAccessToken, goalsRoutes);
app.use('/api/income', authenticateAccessToken, incomeRoutes);
app.use('/api/investments', authenticateAccessToken, investmentsRoutes);
app.use('/api/regular-expenses', authenticateAccessToken, regularExpensesRoutes);
app.use('/api/upcoming-expenses', authenticateAccessToken, upcomingExpensesRoutes);
app.use('/api/user', authenticateAccessToken, userRoutes)

app.listen(process.env.PORT, () => {
  console.log(`Server running at http://${process.env.HOST}:${process.env.PORT}/`);
});
```

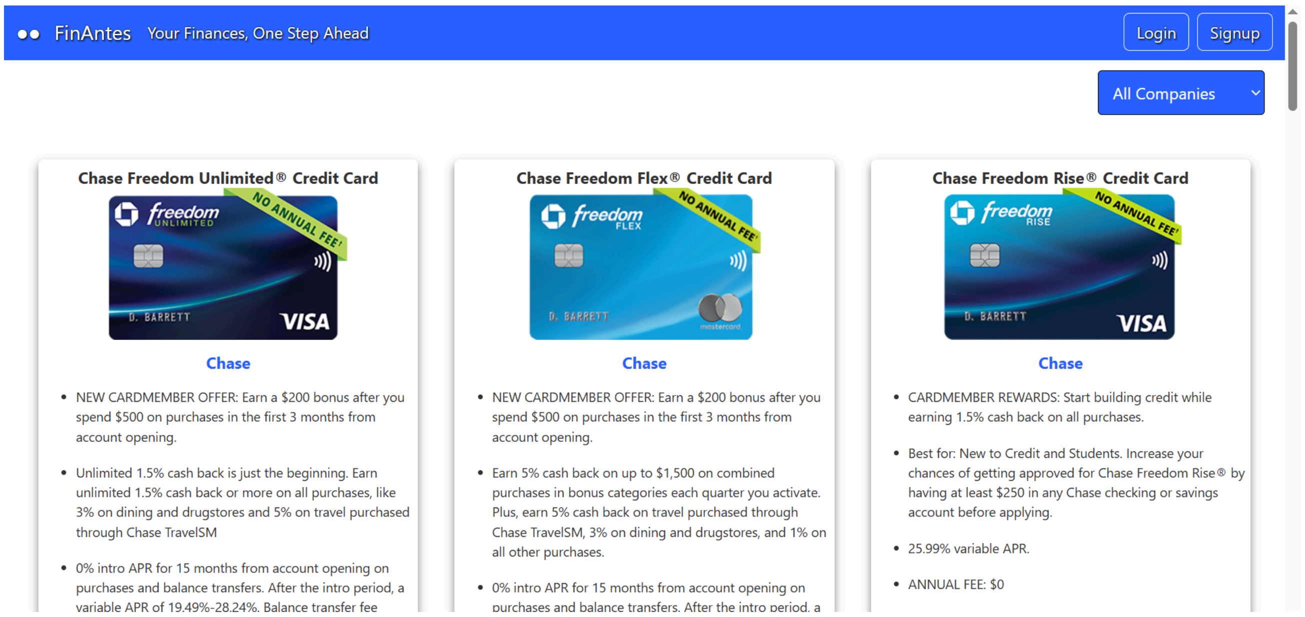


Backend > utils > db.js > ...

```javascript
const { MongoClient } = require('mongodb');
require('dotenv').config();

const uri = process.env.MONGODB_URI;
const client = new MongoClient(uri);
const dbName = process.env.MONGODB_NAME;
const db = client.db(dbName);

module.exports = {
    client,
    db
};
```

5. Web View Screenshots and Annotations

Insert and annotate full-page screenshots for 4 features (2 for each team member)
Description: What this page does and what the user can do.



Chase Freedom Unlimited® Credit Card

Chase

- NEW CARDMEMBER OFFER: Earn a $200 bonus after you spend $500 on purchases in the first 3 months from account opening.
- Unlimited 1.5% cash back is just the beginning. Earn unlimited 1.5% cash back or more on all purchases, like 3% on dining and drugstores and 5% on travel purchased through Chase TravelSM
- 0% intro APR for 15 months from account opening on purchases and balance transfers. After the intro period, a variable APR of 19.49%-28.24%. Balance transfer fee

Chase Freedom Flex® Credit Card

Chase

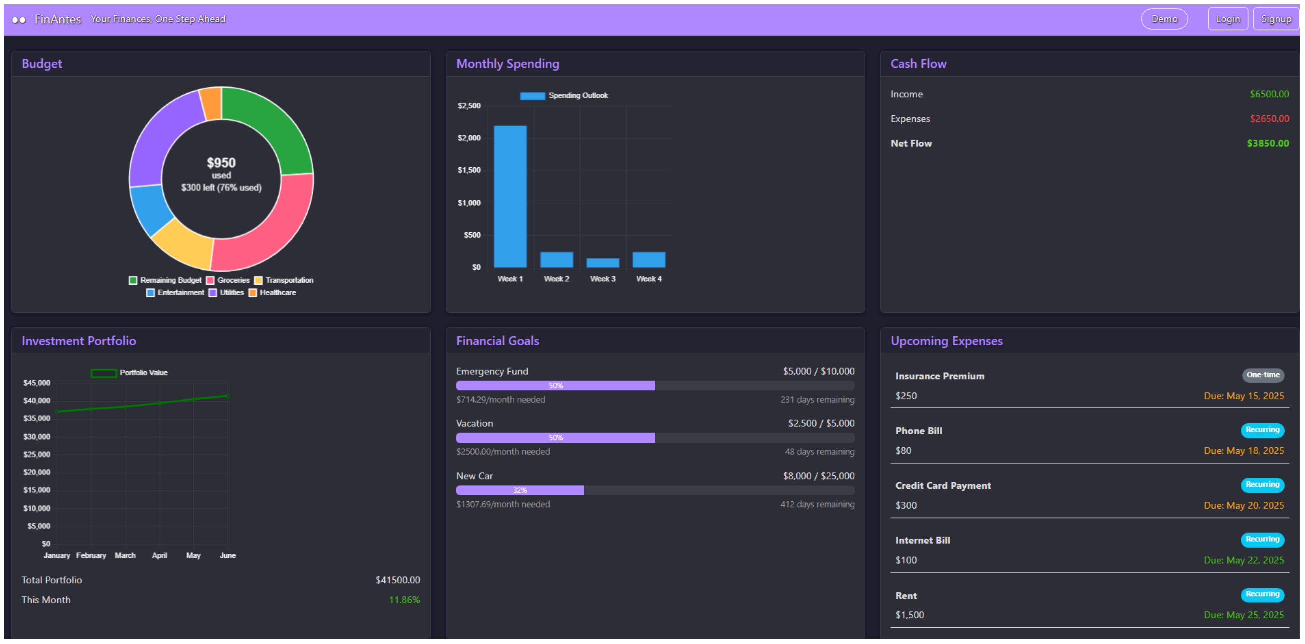
- NEW CARDMEMBER OFFER: Earn a $200 bonus after you spend $500 on purchases in the first 3 months from account opening.
- Earn 5% cash back on up to $1,500 on combined purchases in bonus categories each quarter you activate. Plus, earn 5% cash back on travel purchased through Chase TravelSM, 3% on dining and drugstores, and 1% on all other purchases.
- 0% intro APR for 15 months from account opening on purchases and balance transfers. After the intro period, a

Chase Freedom Rise® Credit Card

Chase

- CARDMEMBER REWARDS: Start building credit while earning 1.5% cash back on all purchases.
- Best for: New to Credit and Students. Increase your chances of getting approved for Chase Freedom Rise® by having at least $250 in any Chase checking or savings account before applying.
- 25.99% variable APR.
- ANNUAL FEE: $0

The Credit Cards page displays a collection of credit cards from different companies. Users can filter the cards by company via the dropdown menu on the top right. Users do not need an account to browse this page.



Cash Flow

Income	$6500.00
Expenses	$2650.00
Net Flow	**$3850.00**

Total Portfolio	$41500.00
This Month	11.86%

Financial Goals

Emergency Fund $5,000 / $10,000
50%
$714.29/month needed 231 days remaining

Vacation $2,500 / $5,000
50%
$2500.00/month needed 48 days remaining

New Car $8,000 / $25,000
32%
$1307.69/month needed 412 days remaining

Upcoming Expenses

Insurance Premium One-time
$250 Due: May 15, 2025

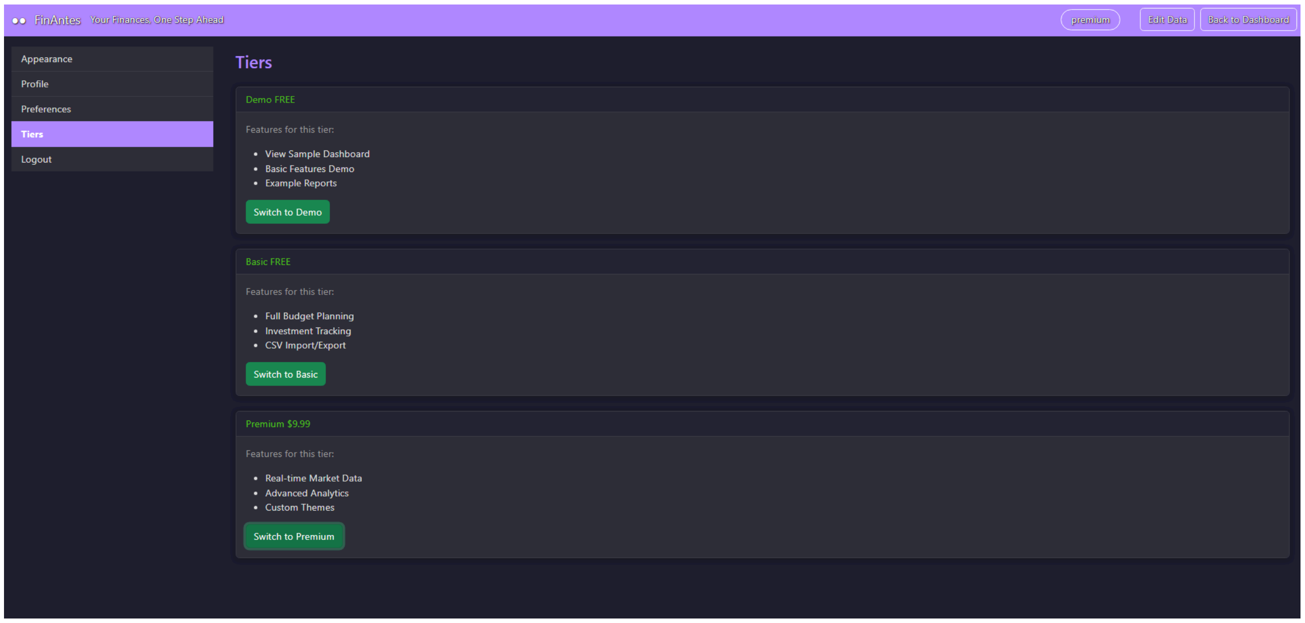
Phone Bill Recurring
$80 Due: May 18, 2025

Credit Card Payment Recurring
$300 Due: May 20, 2025

Internet Bill Recurring
$100 Due: May 22, 2025

Rent Recurring
$1,500 Due: May 25, 2025

The Dashboard page provides an overview of the user's financial status, displaying budget allocation, monthly spending trends, cash flow, investment portfolio performance, financial goals progress, and upcoming expenses. Users can visualize their financial data through interactive charts and progress bars. All financial metrics are updated in real-time based on the user's transaction data.


●● FinAntes Your Finances, One Step Ahead premium Edit Data Back to Dashboard

Appearance
Profile
Preferences
Tiers
Logout

Tiers

Demo FREE

Features for this tier:

- View Sample Dashboard
- Basic Features Demo
- Example Reports

Switch to Demo

Basic FREE

Features for this tier:

- Full Budget Planning
- Investment Tracking
- CSV Import/Export

Switch to Basic

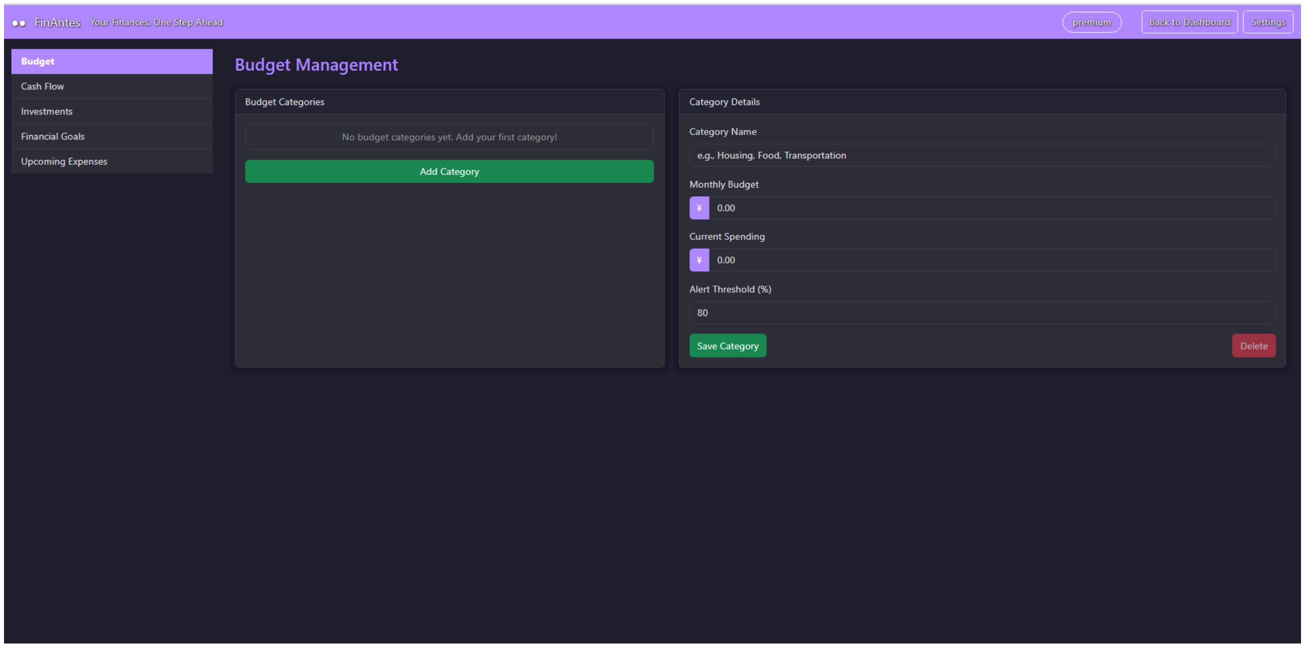
Premium $9.99

Features for this tier:

- Real-time Market Data
- Advanced Analytics
- Custom Themes

Switch to Premium

The Tier Selection page allows users to view and switch between different subscription levels: Demo, Basic, and Premium. Each tier displays its included features, with action buttons to switch between tiers. Users must have an account to access this settings page and manage their subscription preferences.


●● FinAntes Your Finances, One Step Ahead premium Back to Dashboard Settings

Budget
Cash Flow
Investments
Financial Goals
Upcoming Expenses

Budget Management

Budget Categories

No budget categories yet. Add your first category!

Add Category

Category Details

Category Name

e.g., Housing, Food, Transportation

Monthly Budget

¥ 0.00

Current Spending

¥ 0.00

Alert Threshold (%)

80

Save Category Delete

The Budget Management page enables users to create, edit, and delete budget categories for expense tracking. Users can add new categories by specifying the name, monthly budget amount, current spending, and alert threshold percentage. Users must have an account to access this editor page and manage their data.

6. Installation and Setup Instructions

Our database is hosted on MongoDB Atlas, making it easier for both parties to use and manipulate data without having to send files to each other. As stated in 4.3, we have 8 different collections.

On the backend, we have a .env file with our post, host, MongoDB information, and JWT secrets and expirations. We use the following dependencies: bcrypt, body-parser, cookie-parser, cors, dotenv, express, jsonwebtoken, mongodb, uuid. An important thing to note is that, since he uses HTTP-only cookies, we need to specify the origin when we add the cors middleware. The value of origin is the host and port we are running our React app on, which in our case was http://localhost:5173. We also specified credentials: true.

On the frontend, the only thing to note is the axios configuration in main.jsx. We specify the baseURL of the server, we set withCredential to true to send the HTTP Only cookies with every request, and we set a default timeout of 30 seconds (30000ms). We use the following dependencies: axios, bootstrap, chart.js, react, react-bootstrap, react-chartjs-2, react-dom, react-router-dom. We also created absolute pathing in our vite.config.js file to make it easier to import components.

To run since the database is always online thanks to MongoDB Atlas, we just need to run npm i in both the frontend and backend, then run nodemon . in the backend and npm run dev in the frontend.

7. Contribution Overview

Jayson Acosta	Zaid Rachman
Dashboard	Landing Page
Settings	Credit Cards Page
Editor	Authors Page
User Authentication	Intermediate Process Page
API Integration	Checkout Page
Data Synchronization	Confirmation Page
Demo Page	

8. Challenges Faced

One particular challenge was getting the credit card stylesheet to apply appropriately. Initially, it was imported as just a standard CSS file, but the styles somehow applied to all the other pages. We switched it to a module.css file to prevent it from causing issues to the different pages, which caused it to not render correctly on the credit cards page either. Finally, we made it an object and adjusted the jsx file accordingly until the styles were rendered as we wanted.

Another challenge was figuring out how to set up the Authentication on the frontend properly. During the midterm project, we had no authentication and everything was stored in local storage, but with the final, we added hashed passwords and JWT tokens. Since the authentication tokens expired so frequently, we needed to find a way to refresh them in the background without the user ever getting bothered. That's when we recalled that you could pass down context to each component, and have those components wait for the data. The second the user gets onto the landing page, we request the database to verify the token. If that does not work, we send another request to refresh the token. We know the user is not logged in if that also does not work. We initially attempted to check if the user had cookies, but this did not work once we switched to HTTP-only cookies to prevent XSS attacks. After lots of testing, we finished the AuthContext component, which lets everything inside its context use its custom state hook. This allows us to check if the user is logged in or if we are waiting for responses from the server. We were also able to create protected routes due to this, forcing users to sign in if they access the dashboard, editor, or settings pages.

9. Final Reflections

Summary of learning outcomes and things you would improve.

Zaid Rachman Response:

Throughout the development of this project, we gained valuable hands-on experience building a full-stack application from scratch using MongoDB, Express, React, and Node.js. This project helped us understand how these components interact and work together to create a well-structured user experience. Personally, I found the backend development particularly interesting and would like to further my understanding in that area.

Jayson Acosta Response:

Throughout the development process, we were able to learn a lot of different technologies. Through all the errors I encountered on this project, I have become more confident in my React and Express skills. I never used MongoDB before, but now I can confidently use it without worrying. I would like to learn more in-depth about Express and React, specifically middleware and custom hooks. This was my first time making a state hook in React and using context in the way I did. The backend was also the second biggest node application that I have made.