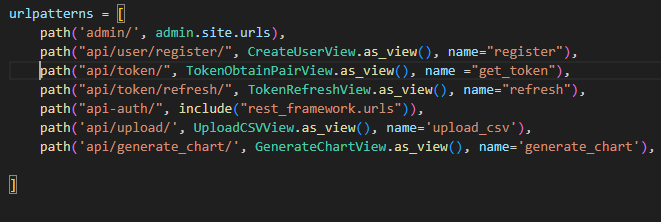
**Code Walkthrough**

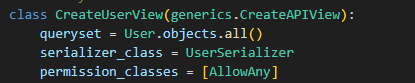
Defining endpoints for user authentication, CSV File Upload in urls.py



Authentication:

Set up of JWT Token Authentication in settings.py, default authentication class form “rest\_framework\_simplejwt” package. Using built in views like TokenObtainPairView, TokenRefreshView

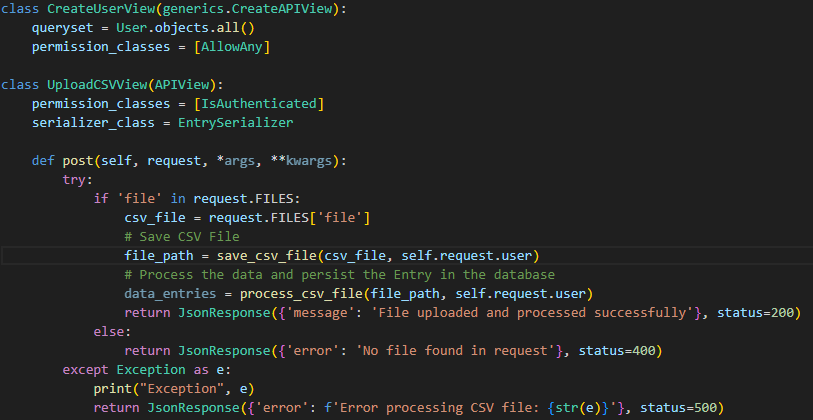
Registration of users in views.py, serializes the input data in User class using UserSerializer, and anyone can access this feature regardless if they are logged in or not

****

Saving CSV file and Creating Entry entities

Only Authenticated users can access feature with the permission\_classes, in views.py

1. Status 500 return if invalid file was uploaded
2. Save\_csv\_file takes in the file and saves it in media/uploads
3. Process\_csv\_file checks if it’s a valid csv file and creates Entry entity based on the data in each row of the csv file
4. Returns status 200 when successful



3

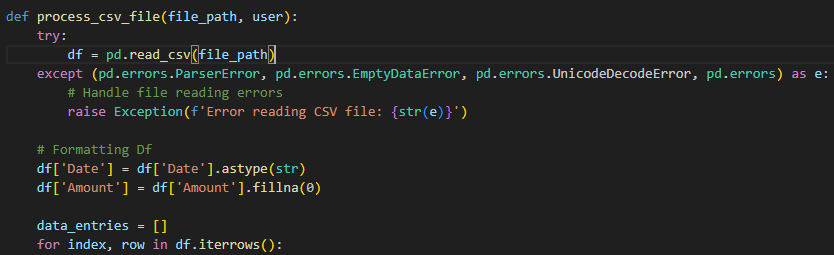
2

1

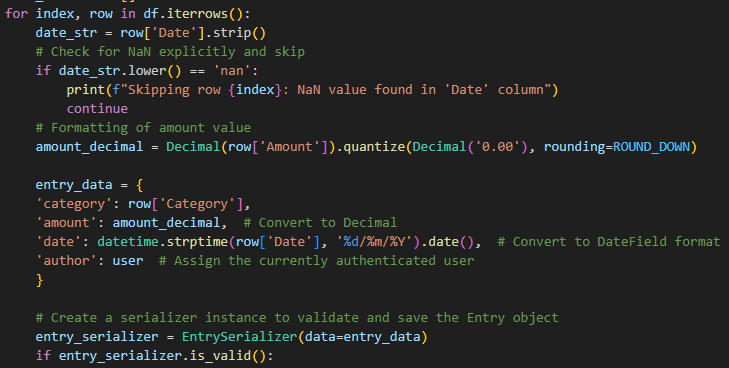
Explaination of process\_csv\_file:

Tries to use read\_csv method with pandas, if unable to read, exception is raised

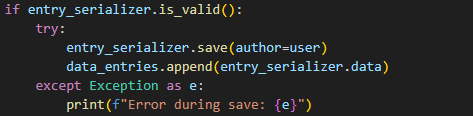
Ensure Date fields are registered as string type



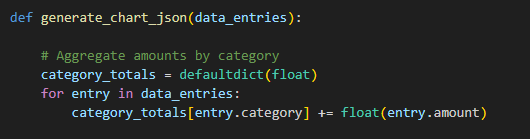
Skip the iteration of rows that are not valid and convert the data to appropriate data types that can be persisted



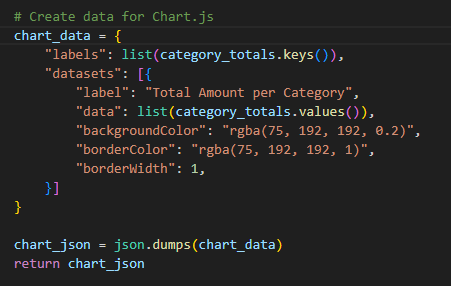
Serializes the data into Entry entities and save it to the database, catching any exceptions if required



Generating Chart Data, summing all the entries with the same category together

****

Storing the data with the correct key values pairs required for chart.js, returning chart data

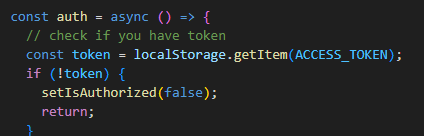
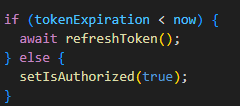
****

**React Front End:**

Tailwind CSS was used for styling, toastify was used to show success / error messages, react chartjs-2 was used to display charts. JWT helps to decode token

There are 3 main folders, components consist of reuseable UI Components, Layout consists of components that are used throughout the application, and pages represent different pages of an application

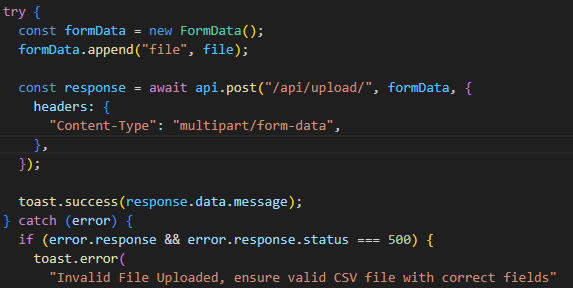
Protected Route component is used to ensure authorization is valid, upon login, the access and refresh token are set in local storage and when a protected site is accessed, it checks at the validity of token by running auth function

****

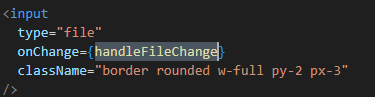
Upload Component: State variable stores the file, and setFile updates the “file” state



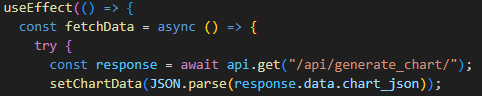
Attempt to make API request to upload the first, if fail, error returned depending on status code



Allows for user to input a file that updates the state and attempts to upload when button is pressed



**Generating Chart Data:** useEffect() Make the API call and update state when the component mounts



Bar chart is rendered using “react-chartjs-2”, the json return have all the information required to display, such as border/ background colour, display chart if it is present with y and x axis labelled





**UI Component: Loading**

**Loading Indiactors:** Spinner Component is created using ClipLoader class from react-spinners. UseState() is set to false by default when the page loads and state is changed to true during function calls



Spinner is only conditionally rendered if loading is true, and loading is false once function completes



WebApp Architecture:

1. Modular Monolithic Approach: The application adopts a modular monolithic architecture where different functionalities like authentication (JWT-based), file upload, and data visualization (chart generation) are logically separated into distinct components or modules. Each module handles specific tasks such as user authentication, handling CSV file uploads, and generating charts based on stored data.
2. Backend Communication via API: The frontend communicates with the backend server through API calls, adhering to RESTful principles. This separation allows for clear delineation between client-side and server-side responsibilities.
3. Frontend-Backend Segregation: The frontend (built with React) is segregated from the backend (Django-based REST API), ensuring loose coupling between the user interface and server logic. This segregation facilitates easier adaptation and modification of the frontend without affecting the backend, enhancing flexibility in design and updates.
4. JWT Authentication: JWT (JSON Web Token) authentication provides secure and authentication for users. It allows the frontend to manage user sessions effectively by storing tokens locally and refreshing them when necessary. This approach ensures that user sessions remain secure and validated across different parts of the application, and users can only view the sections they are authorized to view
5. Component-Based Development: The frontend development follows a component-based approach, leveraging React's capabilities to create reusable UI components such as the ProtectedRoute for managing authenticated routes and Spinner for indicating loading states. This modularity enhances code reusability and maintainability.

**Readme**

What the Program Does:

1. Users are allowed to register or login to their accounts
   1. You can use a prebuilt account, username: jackytan, password: 12345, if preferred
2. After logging in, users can view their financial data of their previous transactions
3. If they have any new transactions, they can upload a CSV which will be saved and financial data will be extracted
4. The home page will be reupdated with the new chart data of their collated transactions based on category
5. CSV file should be in the following format:
   1. Columns: “Date, Category, Amount”
   2. Each row represent each entry with these columns filled

This web application consists of a Django backend and a React frontend, in order to run the application, follow these steps.

1. In VSC, open folder CVPD\_web\_app\_Tan\_Jacky, and open the terminal in VSC
2. Navigate to Backend Directory:
   1. cd backend
3. Activate Virtual Environment
   1. venv\Scripts\activate
4. Install Dependencies (if required):
   1. pip install -r requirements.txt
5. Apply Migrations (if required):
   1. python manage.py makemigrations
   2. python manage.py migrate
6. Run Django Development Server:
   1. python manage.py runserver
7. Ensure you have node.js installed
8. Open a new Terminal and navigate to Frontend Directory:
   1. cd frontend
9. Install Dependencies
   1. npm install
10. Start the Development Server:
    1. npm run dev

Frontend will be run at <http://localhost:5173/> while backend will be run at <http://127.0.0.1:8000/>