**Software Engineering (DLMCSPSE01): EXPENSE TRACKER**

**Table of Contents**

[1. Conception Phase 3](#_Toc184847338)

[1.1 Project Profile 3](#_Toc184847339)

[1.2 Software Development Methodology 3](#_Toc184847340)

[1.3 Requirements Analysis 4](#_Toc184847341)

[1.3.1 Functional Requirements 4](#_Toc184847342)

[1.3.2 Non-Functional Requirements 4](#_Toc184847343)

[1.3.3 Glossary 4](#_Toc184847344)

[1.4 System Design 5](#_Toc184847345)

[1.4.1 Technology and Tools 5](#_Toc184847346)

[2. Development Phase/Reflection Phase 6](#_Toc184847347)

[2.1 Framework and Tools Setup 6](#_Toc184847348)

[2.2 Third-Party Library Selection 7](#_Toc184847349)

[2.3 Requirement Refinement 7](#_Toc184847350)

[2.4 Component Implementation 7](#_Toc184847351)

[2.5 Consideration of Available Resources 7](#_Toc184847352)

[2.6 Identifying Potential Problems, Risks, and Opportunities for Improvement 7](#_Toc184847353)

[2.7 Monitoring Status and Progress 7](#_Toc184847354)

[2.8 Portfolio Page Submission 7](#_Toc184847355)

[2.8.1 Implementation Summary 7](#_Toc184847356)

[3. Finalization phase 8](#_Toc184847357)

[3.1 Self-reflection and backend development 14](#_Toc184847358)

# 1. Conception Phase

## 1.1 Project Profile

**Project Major Scope and Objectives**

The project goal is to develop a coherent web application such as an expense tracker which has helped to store and manage numerous financial transactions. This web application is very beneficial for users such as users can view, add, editing, deleting and uploading bulk transactions. This expense tracker application also helps users to keep their all financial transactions, data validation and uploading CSV files.

**Target Group**

Here our target group is end-users such as businesses or persons who have an interest in recording or monitoring their financial operations. The major target groups are accountants, managers as well as owners of small business.

**Risks**

* Validate and process data accurately, especially for bulk CSV uploads.
* Secure user data and protect against unauthorized access.
* Maintain high performance when handling large transaction volumes.
* Provide clear error messages and manage incorrect inputs gracefully.

**Project Plan**

For the backend, the developer will use node which is a javascript and runs on a local browser and express which is a javascript framework which makes it easier to set up a backend server and listen to the user request on a local machine and respond based on the request along with typescript. External Api is used for currency conversion. Along with this database, the developer will use a relational database mainly Postgres due to its faster CRUD(Create, Read, Delete, Update) operations on a dataset and set up the connection using the credentials provided by Postgres on the first installation which is accessed through Mikro ORM. Also, the database can be managed using PG-Admin where the admin can see the database that consists of the raw data.

## 1.2 Software Development Methodology

Here the developer uses agile methodology such as iterative progress, collaboration and adaptability. Requirements are gathered in a dynamic way with feedback from the user in real time. Sprints are also used in development cycles which are planning, coding, testing and deployment. Frameworks such as React and Node.js as well as databases such as PostgreSQL improve modularity and scalability. Agile encourages versatility, and frequent communications, and is consistent with current development methodologies for software release.

## 1.3 Requirements Analysis

### 1.3.1 Functional Requirements

**Transaction Management**

* Users can view, add, edit, and delete individual transactions as well as there are options for bulk operations such as uploading CSV files and bulk deletion.

**Data Validation**

* Secure transaction data adheres to rules with real-time error messages for invalid inputs.

**Pagination and Navigation**

* Users have the ability to navigate via transactions with pagination controls such as options to jump to the first or last page.

**Currency Conversion**

* With the help of this application, users can automatically calculate and display transaction amounts in INR using an external API, based on the transaction's date and original currency.

**Soft Deletion**

* Here we also use a soft delete feature such as marking transactions as inactive instead of permanently removing them from the database.

### 1.3.2 Non-Functional Requirements

**Performance**

* The system has the ability to bulk uploads and operations efficiently without significant delays.

**Scalability**

* Support future growth, such as increasing the number of transactions or users.

**Security**

* Secure data integrity and prevent unauthorized access with secure authentication and validation.

**Usability**

* Provide an intuitive and user-friendly interface for seamless interaction.

**Reliability**

* Guarantee the system functions consistently with minimal downtime or errors.

### 1.3.3 Glossary

**Transaction**

* A record of financial activity including date, description, amount and currency.

**Pagination**

* A navigation system divides content into various pages for easier browsing.

**Soft Delete**

* A method of marking records as inactive without permanently removing them from the database.

**CSV File**

* A text file format will be used for storing tabular data which has been allowing bulk uploads of transactions.

**API (Application Programming Interface)**

* A service enabling data exchange, such as currency conversion, between systems.

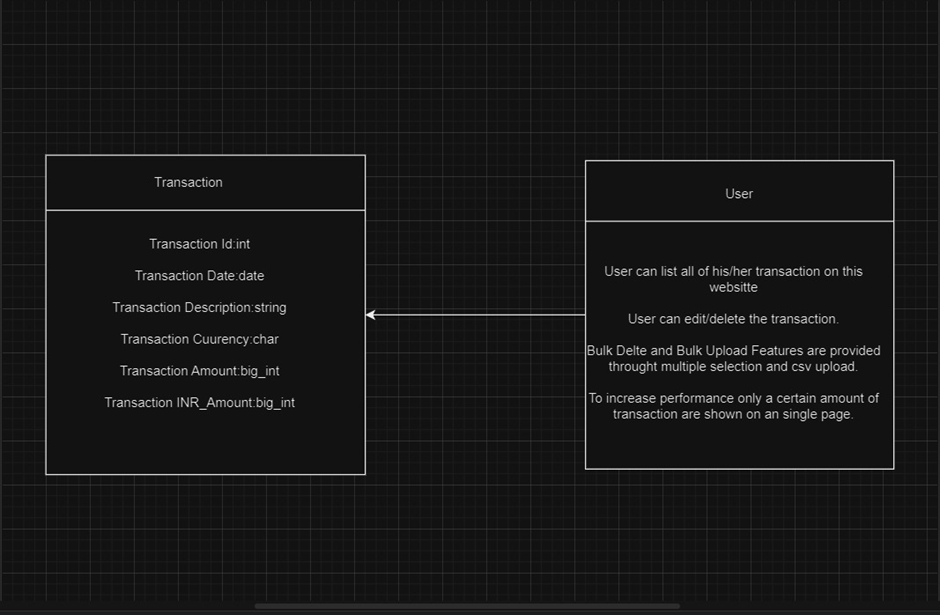
## 1.4 System Design

The application has a user transaction mechanism of viewing, adding, editing, and deleting transactions as well. It has the feature of CSV upload and currency conversion implemented in the software.

### 1.4.1 Technology and Tools

* Frontend: React (TypeScript) for building interactive UI.
* Backend: Node.js with Express for server-side logic.
* Database: PostgreSQL with Mikro ORM for efficient data management.
* API: Currency conversion API for INR calculation.
* Tools: PG-Admin for database management, Moment.js for date validation.

**UML diagram**

****

**Figure 1: UML**

**1.5 Summary**

React is selected for the frontend because it is a dynamic and scalable library, while TypeScript provides type-safety which makes the frontend more readily maintainable and increases the user experience. Node.js with Express is a perfect solution for backend processing and handling API requests quickly and efficiently. Mikro ORM with PostgreSQL ensures that all transactions are well handled in a very efficient manner. Currency Conversion API, the PostgreSQL utility PG-Admin enables efficient management of the database and Moment.js provides valid date formats for smooth operations, achieves real-time exchange rates.

# 2. Development Phase/Reflection Phase

The construction of the Expense Tracker web application starts which must remain compliant with the demands and operational for implementation, support and upgrade. The development phase consists of several activities that refer to the planning, application implementation and fine-tuning. Below is a detailed breakdown of the activities during this phase:

## 2.1 Framework and Tools Setup

Implemented tools and frameworks from the Conception Phase: Frontend – React; Backend – Node.js, Express; Database – PostgreSQL; ORM – Mikro ORM; Currency conversion API. This makes it possible to check that all the necessary environment variables for building and running the application are set.

## 2.2 Third-Party Library Selection

Mostly third-party library sources and APIs such as recent Moment.js for managing dates and actual Currency Conversion API for the conversion of currency to INR. This allows for coherent handling of dates and currency calculations within the application.

## 2.3 Requirement Refinement

The requirements elicited during the Conception Phase are subjected to further analysis in order to establish their clarity and feasibility. This may require adjusting the functional and non-functional requirements depending on the feedback received and the progress of the project.

## 2.4 Component Implementation

The fundamental components and modules that include the transaction management features including view, add, edit and delete, data validation, CSV upload feature and pagination are developed. Also, the currency conversion feature is implemented, as well as the soft delete feature.

## 2.5 Consideration of Available Resources

Time constraints are one of the major factors managed by dividing the task into smaller units. Here the developer mainly focuses on each aspect at a time such as for frontend design and backend API creation. The various kinds of technology considerations such as the framework and tools are outlined in the below section,

* React.js for the front end.
* Node.js/Express for the backend.
* PostgreSQL for the database.

This approach has been helping us to align with the project scope and deadlines.

## 2.6 Identifying Potential Problems, Risks, and Opportunities for Improvement

* Validation risks for example; invalid data input or CSV upload issues are mostly handled by having robust validation procedures and continuous testing of all steps.
* Taking feedback from users of the system, areas that need improvement are found enabling betterment in UI/UX design, as well as performance enhancement.

## 2.7 Monitoring Status and Progress

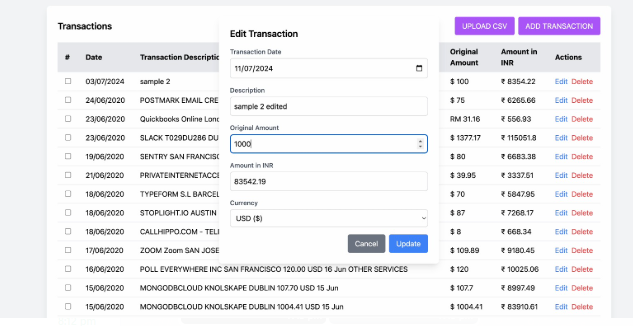
Maintenance checks are conducted to regularly review the progress to make sure it is on track. Documentation is usually updated to reflect every new change as well as every new progress made.

## 2.8 Portfolio Page Submission

### 2.8.1 Implementation Summary

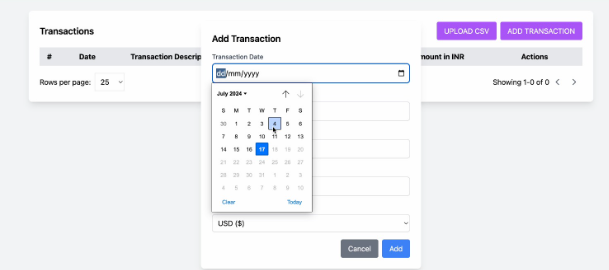
The expense tracker application has a React front end with TypeScript for managing transactions such as creating, updating, deleting and bulk importing through CSV. Initialization with default currency, dynamic currency conversion using an API, and exception handling mechanisms for wrong input regarding the pagination parameters are also implemented. The backend is developed using Node.js, Express, and PostgreSQL with CRUD functionality, soft delete functionality, and CSV import-export functionality. Database operations are made easy by Mikro ORM. Validation checks help in maintaining the quality of data while a third-party API helps in the conversion of currency. The application is designed with a user interface that is friendly and responsive with the ability to update in real-time and post error messages that facilitate proper transaction processing.

# 3. Finalization phase



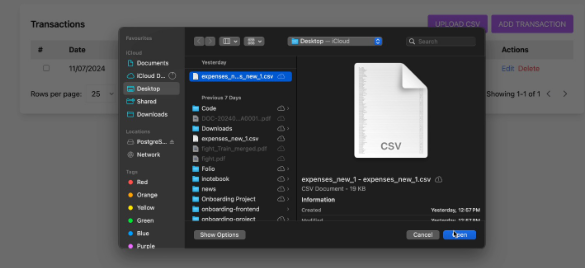
**Figure 2: Transaction Details**

The main page of the website shows the transactions of users listed by user. Here the user along with seeing the transaction can add the transaction and delete the transaction along with editing a particular transaction. Also user is provided with the feature of uploading the csv file for bulk addition of transactions The left and right buttons are provided for transition between pages and the extreme left is for going to page no 1 and the extreme right takes the user to the last page.



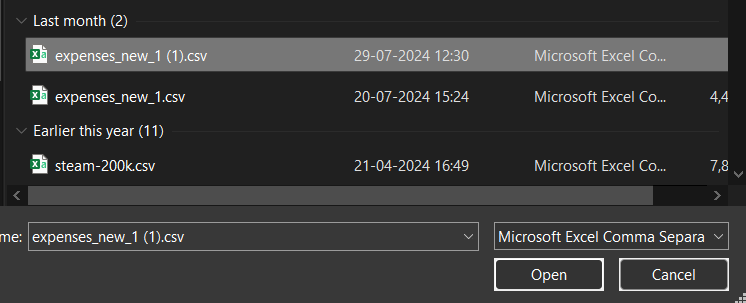
**Figure 3: Add transaction option**

Here the user can add the transaction by providing the transaction description, original amount, currency of the transaction from the drop down and date of the transaction. Now if the user either clicks on the cancel or close button nothing happens but if the user clicks on the save button then that particular transaction gets saved in the database and the frontend is populated with the new transaction.



**Figure 4: CSV uploading option**

This app provides the user with the facility to upload the data in bulk through its csv feature where the user can select the csv file that they want to upload and after successful validation of the data in csv file that particular data is inserted into the database then the user is able to see the transaction in the frontend UI.



**Figure 5: Importing CSV in Web application**

Here the user can select the csv file and after clicking on open that csv file gets uploaded and data is populated in both frontend and backend.

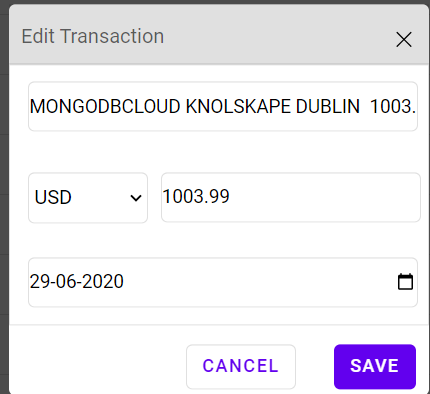


**Figure 6: Transaction description**

Mainly there are four main columns date represents the date of the transaction, description contains the expanded details of the transaction original amount denotes the amount of original currency whereas the amount in INR denotes the converted amount from the currency provided by the user and the original amount and then using the currency conversion rate of the date on transaction happened the INR amount is fetched through external api.



**Figure 7: Buttons**



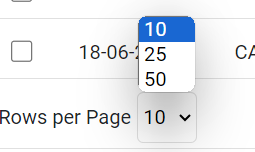
**Figure 7: Editing transaction**

After clicking on editing a transaction this window appears where the user gets pre-filled the data of that transaction and now can choose to modify some or all the fields. if the user clicks on the close or cancel button then original details are preserved in both the frontend and backend. But if the user clicks on the save button then new updated details start to appear for that particular transaction in both frontend and backend. If the user clicks on the delete button then that particular transaction gets deleted from the frontend and backend. To not have harsh delete we have implemented a soft delete feature which basically has extra information about a particular transaction which tells if the user has deleted that transaction or not.



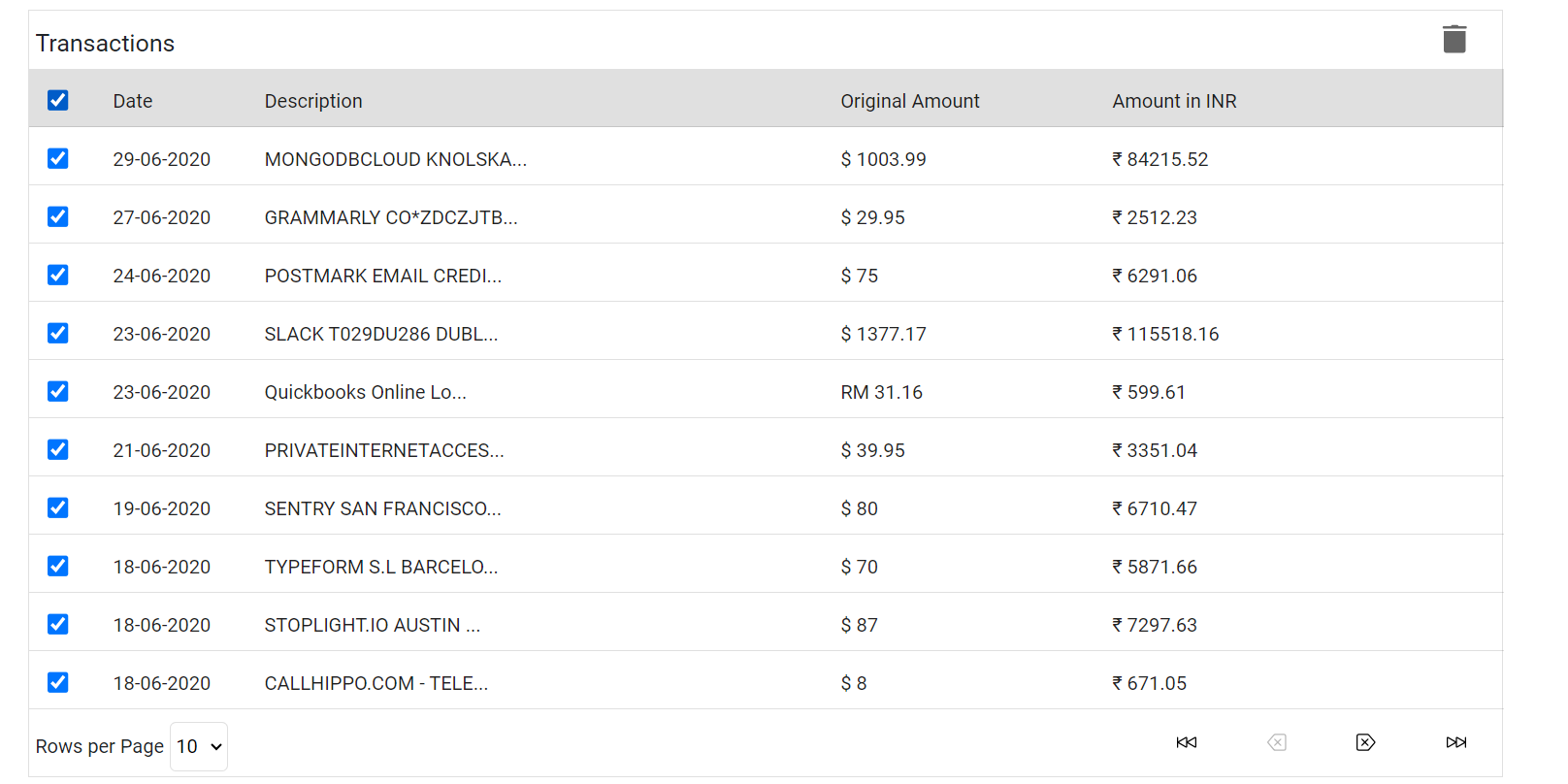
**Figure 8: Buttons**

These buttons allow the user to navigate between the pages so if the user is on page no 1 and then by default going to left option is disabled as there is no backward same logic happens when the user is on the last page for going right button extreme left takes the user instantly to page no1 and extreme right button takes the user instantly to the last page.



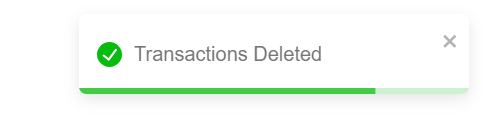
**Figure 9: Rows per page**

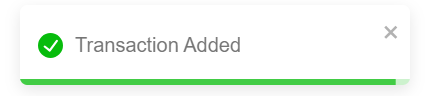
From this drop-down list user can select the amount of entries that need to be shown per page which has the option of 10-25-50.

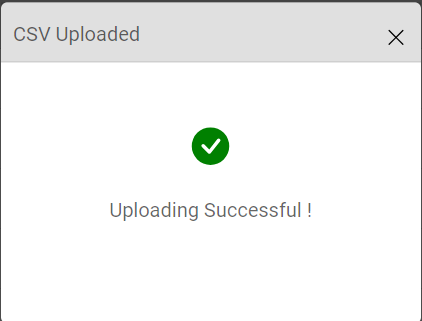


**Figure 10: Bulk delete**

If the user clicks on the bulk delete button at the top left then all the entries of that page get selected and a delete button appears on the right-hand side. After clicking the delete button all the selected entries get deleted from frontend and backend.

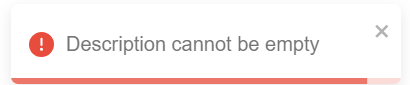
  
**Figure 11: Message the user gets after the successful deletion of an entry**

  
**Figure 12: Message the user gets after adding a transaction successfully.**

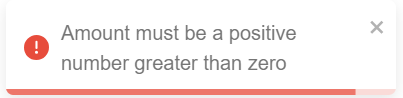


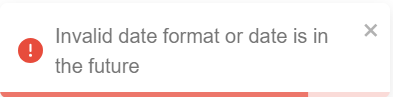
**Figure 13: Message shown after successful insertion of csv data into frontend.**

**Edge Cases or Test Cases for Error Handling**



**Figure 14: Message that appears when user leaves the transaction description empty**

  
**Figure 15: Message the user gets when he enters any other entry beside 0 or a positive number in the amount section.**

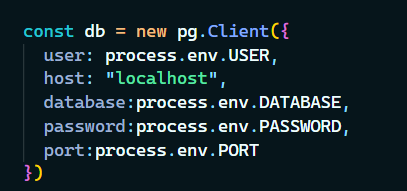


**Figure 16: Message which appears when the user has entered an invalid date format or selected a future date**

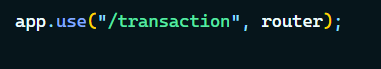
## 3.1 Self-reflection and backend development

For the frontend, I have used the React framework which runs on localhost on port no 5432 of the local machine. The language used is typescript in which component rendering concept is used. The use of the useState feature in react makes it possible to render some of the components that require the user input for showing them in UI(for example if the user clicks on add transaction only then that add transaction model should appear in the screen). All the components are separate from each other and the project is highly customisable increasing the overall performance of the web app. Also, Mikro ORM is used along with Postgres to increase the performance of the web app and make it write simpler queries for fetching and inserting data into the database.

**Backend**

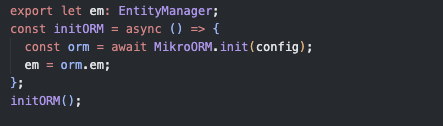


**Figure 17: db configuration for connecting the database to the backend**

****

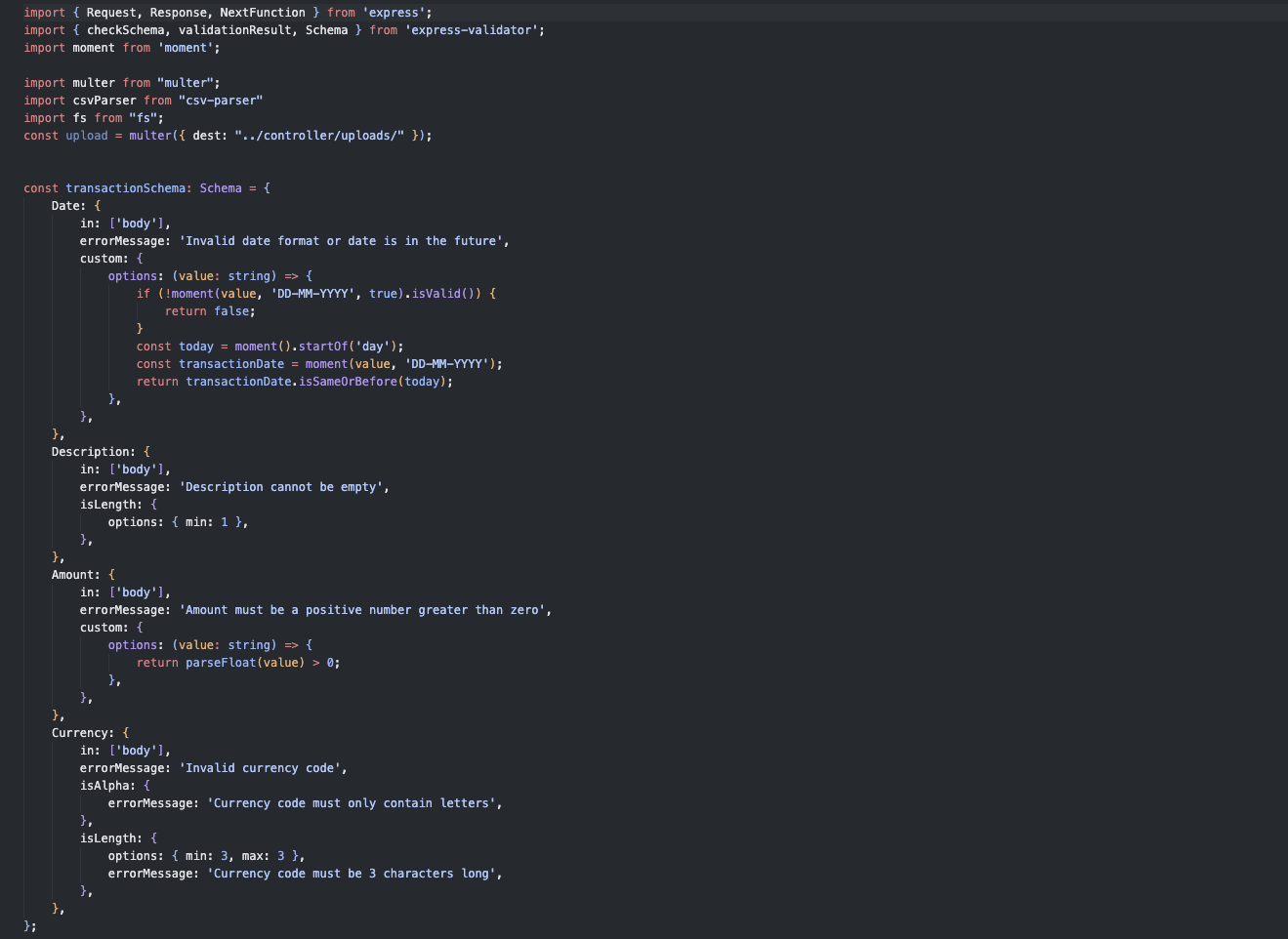
**Figure 18: Router**

Through this router, all the requests get processed.



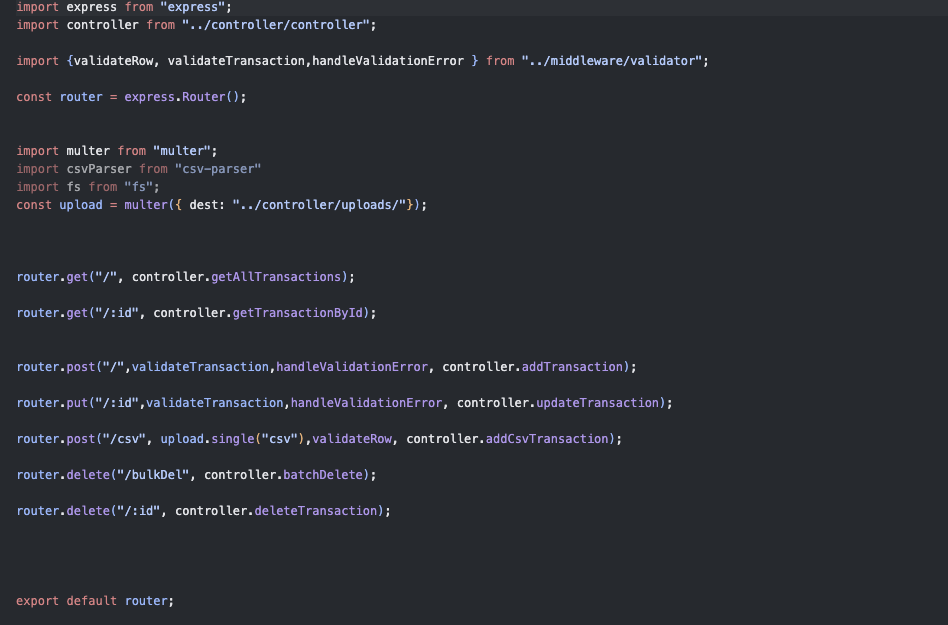
**Figure 19: Mikro ORM**

Initialization of the Mikro ORM for setting up the connection between the database and the backend.



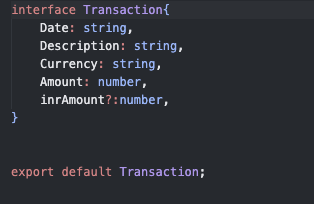
**Figure 20: Validator file**

This is the validator file used to validate the data data being sent to the backend. From this, the date is checked to avoid being in the future, the description does not have any special characters and the amount should not be negative. If there is any mismatch in the data, rather than sending the request to the router a response message is sent to the frontend containing the error message.



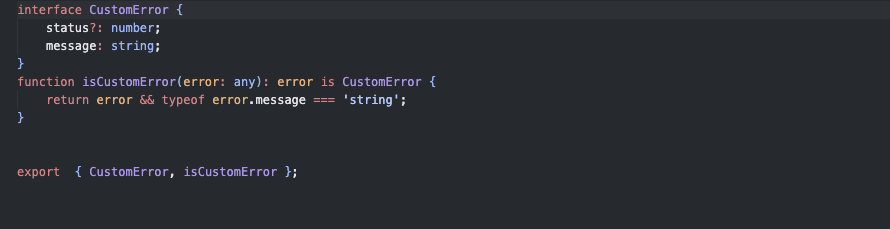
**Figure 21: Handling Fronted request**

Here we handle all the frontend requests that are sent to the backend get requests are meant for fetching the transaction to display on the frontend post requests are made to insert the data into the database as well as delete requests are made to delete the data from the database. Put requests are made to update a particular transaction and before processing the request a validation happens for the data being sent by the user (like amount is positive or not, the checking date is in the proper format, description is not empty) and if the validation fails then that particular request is not processed further and an appropriate message is sent on the response to the user.

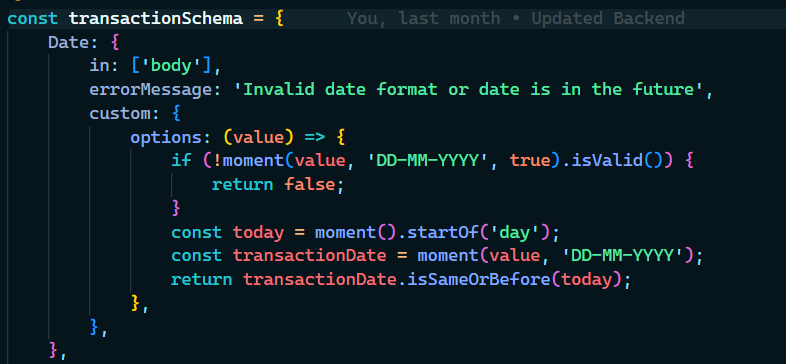


**Figure 22: TypeScript**

Since we are using typescript, this is the type defined for the transaction data coming from the database.

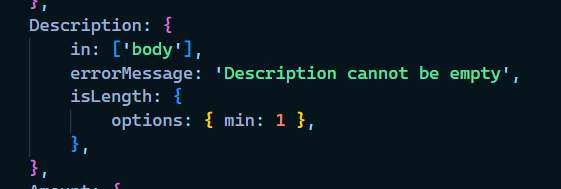


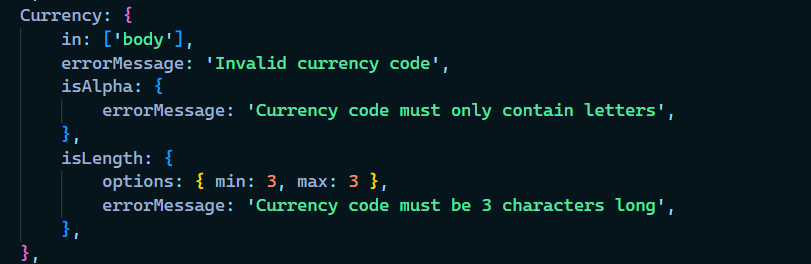
**Figure 23: This is the type of error message which is shown to the user when invalid data is passed.**

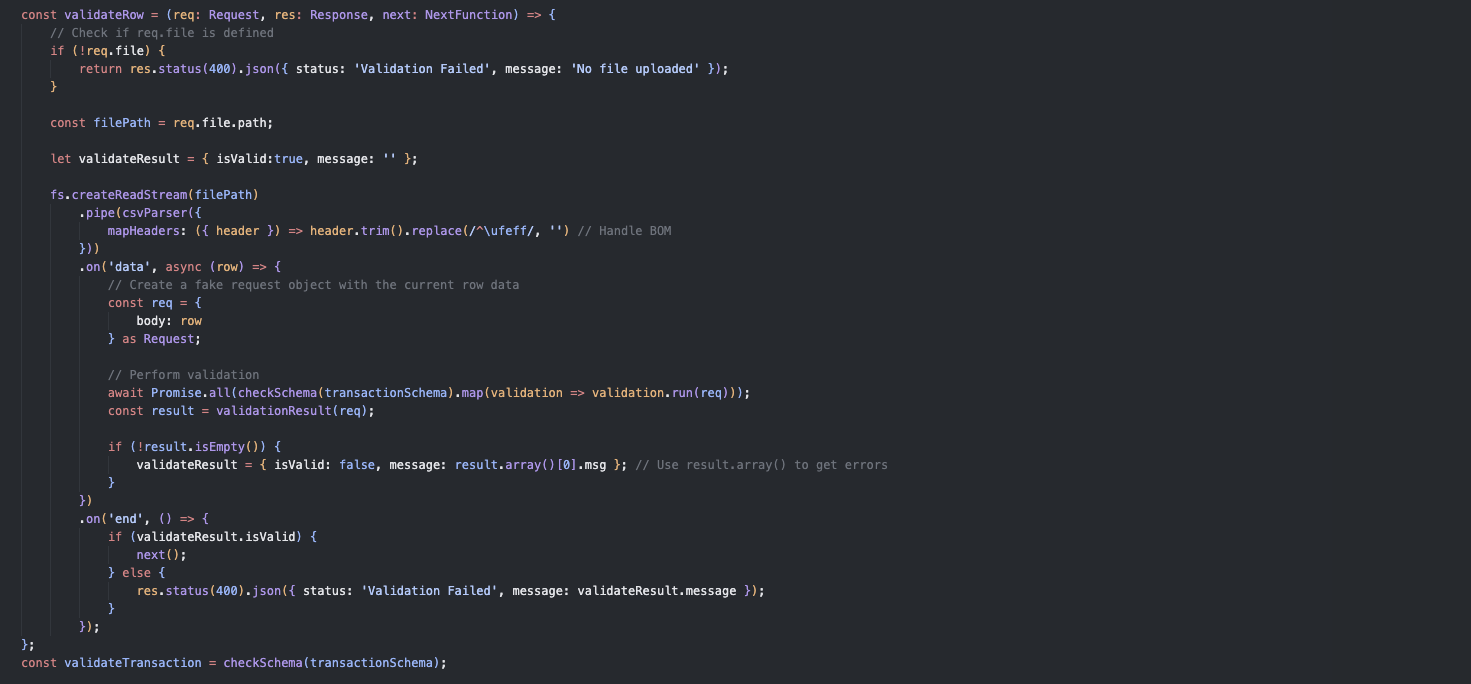


**Figure 24: Logic for date**

Logic for validation of Date.moment package is used to check that the date is not from the future and also for format checking with DD-MM-YYY format to be valid.

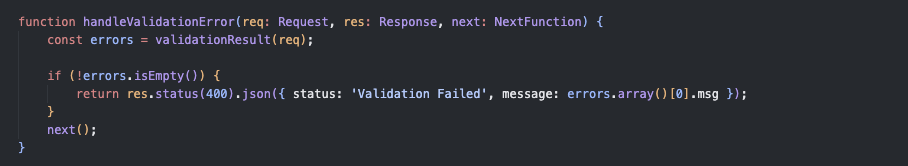
  
**Figure 25: Logic for checking the description length should be greater than 1(basically it can't be empty)**

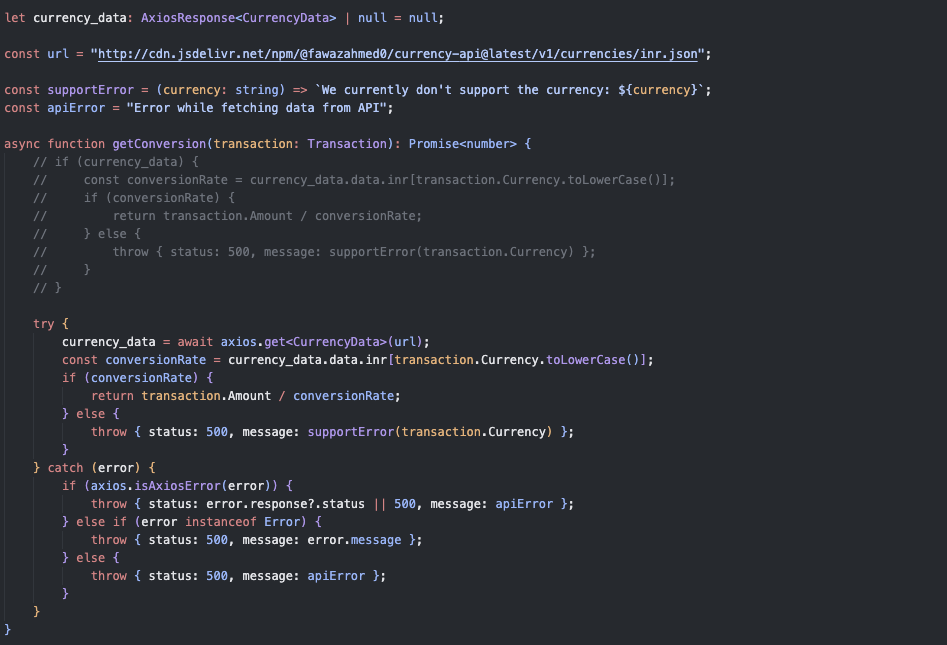
  
**Figure 26: Currency should be provided from the dropdown only and must be three characters**



**Figure 27: Logic for converting CSV data**

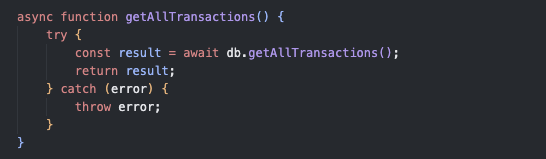
The logic for converting the csv data into the proper format by first checking the data field value and stopping inserting the data if any error data is found otherwise convert the data into an array and then insert it into the database.

  
**Figure 28: Function to handle the logic of the situation when an error is found in the data given by the user.**

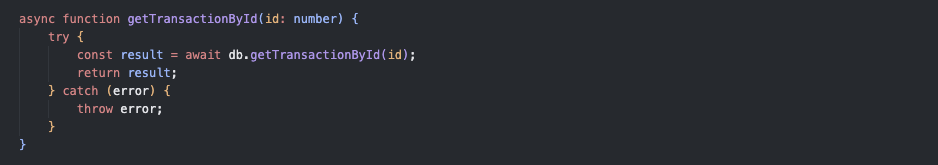


**Figure 29: Logic for currency converting**

This logic is for api that converts the currency along with the original amount to inr\_amount which is then inserted into the database. The transaction's original amount is divided by the currency conversion rate on the date of the transaction to obtain the inr\_amount which is then inserted into the database along with other transaction details.

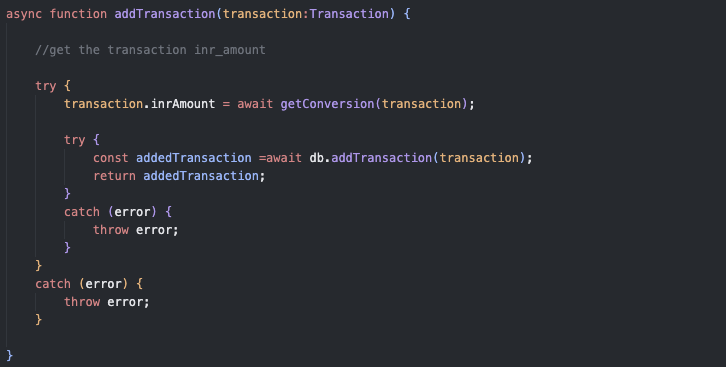


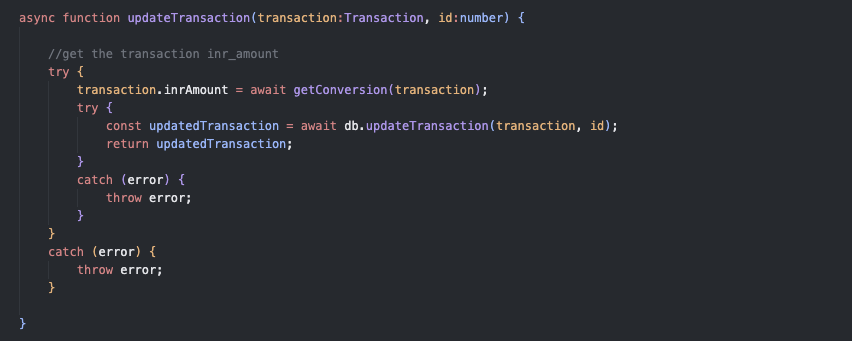
**Figure 30: This logic fetches all the transactions that are inserted into the database by the user which shows on the front when the user first loads the website.**



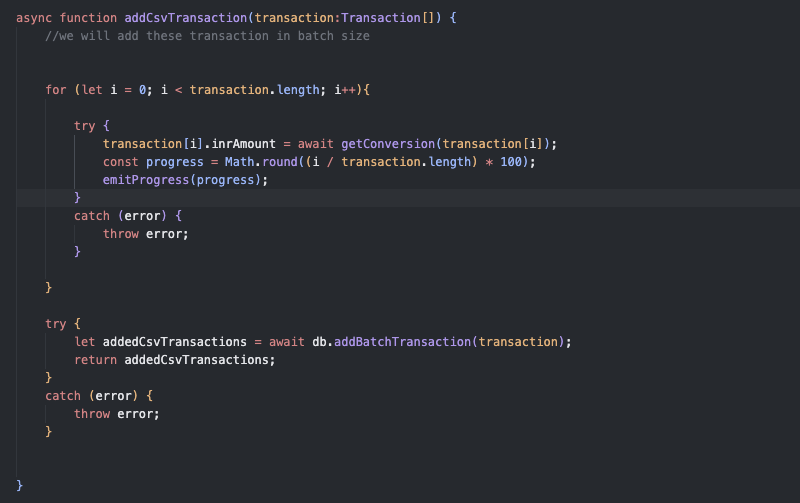
**Figure 31: Logic for fetching transaction details**

This logic gets the transaction by a particular unique ID of that transaction. This logic is used to fetch all the details of the transaction to be pre-filled when the user clicks on the edit for any transaction.

  
**Figure 32: This logic is for adding the transaction after all its fields pass the validation test**.

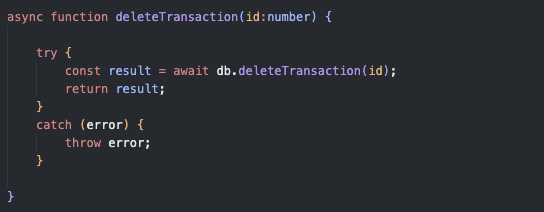


**Figure 33: This logic in the database file is to update a transaction.**

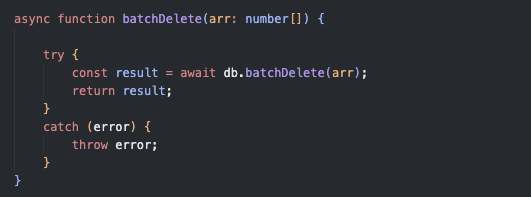


**Figure 34: Logic for handling CSV**

This logic handles the insertion of csv data where in arguments an array is accepted which contains all the csv data and then inserted one by one into the database also the emit progress shows the progress in the frontend to the user in terms of percentage.



**Figure 35: Logic deals with deleting a transaction**



**Figure 36: Logic for handling multiple entries**

This logic handles the case of deleting multiple entries from the database. It accepts an array consisting of unique IDs of the transaction that the user wants to delete and then all the transactions are deleted from the database one by one by iterating over the array. In concluding this web application implementation we can state that the expense tracker web application is coherent for storing and managing numerous financial transactions.