CSCI-5709 ADVANCED WEB SERVICES

ASSIGNMENT – 2

Name	Benny Daniel
Banner #	B00899629
Course Instructor	Shehzeen Huda
Teaching Assistant	Srikrishna Sasidharan

1 – Application Details

Expense Tracker:

Time is precious for people, especially those working in a fast-paced environment. As such, our fast lives do not allow us to journal our expenses in a manner which we might find useful and efficient. Expense management is vital for individuals to gain better insights into their spending habits and keep their expenditures in check. After individuals set a budget, tracking expenses is essential to stay on track with the budget. In addition, modern-day credit card systems enable people to spend money without restrictions, and one's financial debts may skyrocket if they are unable to pay the bills. Even the people who are not victims of this vicious cycle would want to save and invest and manage their finances efficiently [1]. Next, noting down debts and reminding people to settle them is a dreary and time-consuming task. Individuals would rather log debts and send reminders with the click of a button than to manually note details and contact relevant individuals. We aim to alleviate overheads associated with regular Expense Management by providing a fluid and seamless experience to users, because of which, they can manage, track, and analyze their expenses with ease. The primary objective of developers of the application is to eradicate manual labor as much as possible and automate expense management and settlement to a large extent. This would also mean that payers would get reimbursed appropriate amounts with a minimal number of transactions. Expense Tracker application is built for people who like to manage their finances, including spending in a group. This application is going to act as a full-featured investment manager which will provide users insights based on their expenditures and will create charts displaying their monthly cash flow with an ability to break down expenses by category and dig deeper into any concerning spending habits [1]. We desire users to have a pleasant experience with the application while saving time and effort, therefore user convenience and an intuitive design are among our top priorities for this application.

The feature that I chose from our Project, for this assignment, is the In-app payment integration module. Current expense tracking applications are used in conjunction with payment vendors such as Venmo and PayPal to process transactions. If a user's country supports PayPal or Venmo, they will be directed to the respective application to make a payment, whereas if their country doesn't support either, it's just cash [5]. When we conceptualized the application and perceived its features, we agreed to develop an In-app payment portal where Users can input the details of payees and process payments instantaneously. We have used Stripe [6], an online payment processing and commerce solution for payers to accept or make payments.

2 - Application Details

2.1 Target User Insight

Expense Tracker has a large scope for utilization by individuals with diverse needs. These individuals have been categorized into the 3 User Personas, as follows:

Expense analysts – These are individuals who desire to use the application to analyze their expenditures to date and derive valuable insights based on which they can act upon and optimize their expenses and spending trends [1].

Bursars – Flatmates and bill splitters are the typical 'Bursars' in our application. They will use the app to stay on top of individual or group expenses and settle them on time [1].

Debt settlers – Money lenders and debtors come under this category of User Persona in our application. Debt settlers will input debts that are owed to or by them, in the application and send reminders to relevant individuals, prompting them to settle their debts, without the need to make phone calls [1].

We have designed our application in such a way that users do not need to have any prerequisite knowledge save the basic knowledge of logging into a web application and navigating through the various pages on the website.

Usability Mapping:

The following figures [4], represent the Usability Mapping for the In-app payment interface.

Figure 1 describes the path taken by a user to furnish a payee's details, transaction amount and initiate a payment.

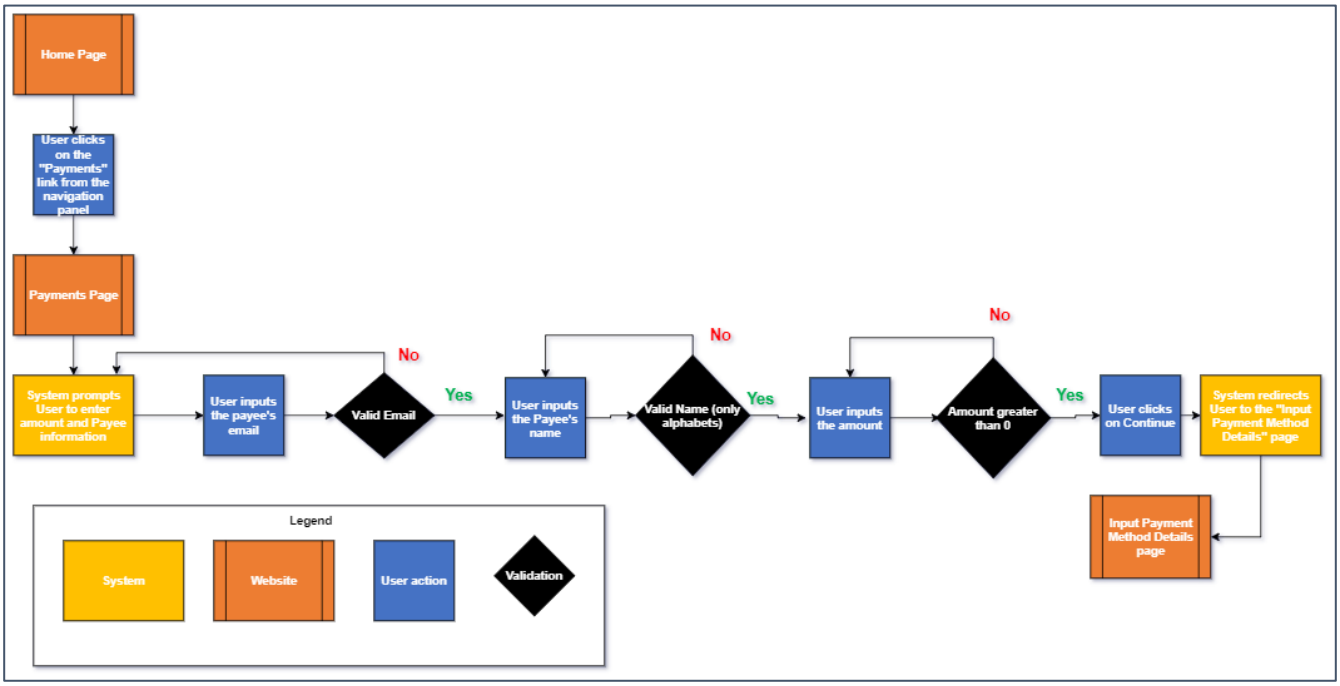

Figure 1: Task Flow – Initiate a Payment

Figure 2 describes the path taken by a user to furnish card details to proceed with a payment.

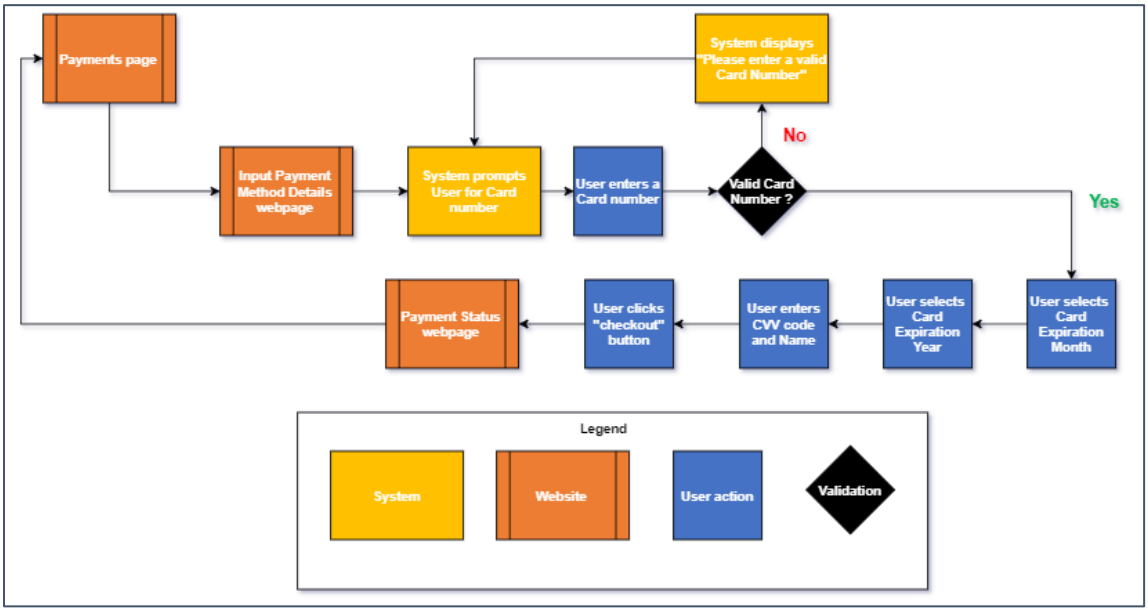

Figure 2: Task Flow - Input Payment Method Details

Figure 3 represents the path taken by a user to view the status of the most recent transaction.

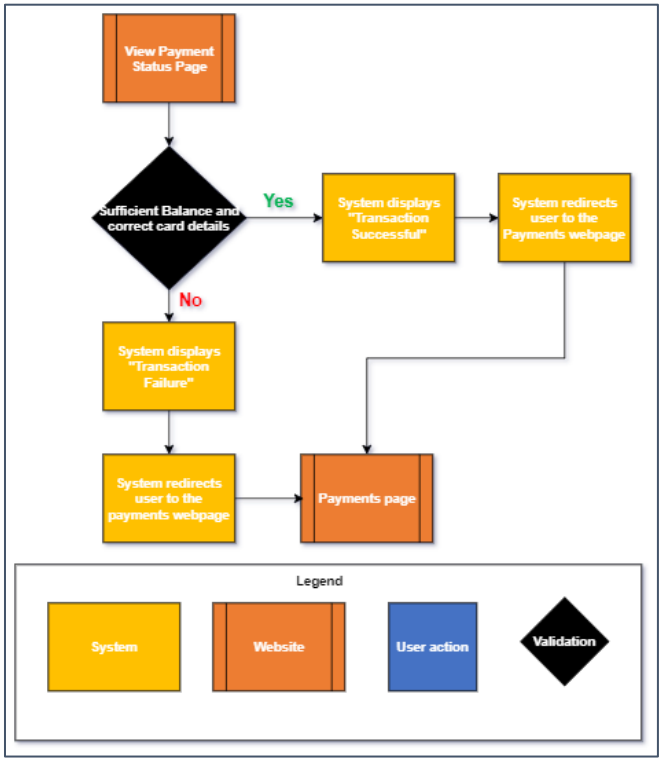

Figure 3: Task Flow - View Payment Status

Figure 4 represents the path taken by a user to view a history of all transactions initiated by them in the past.

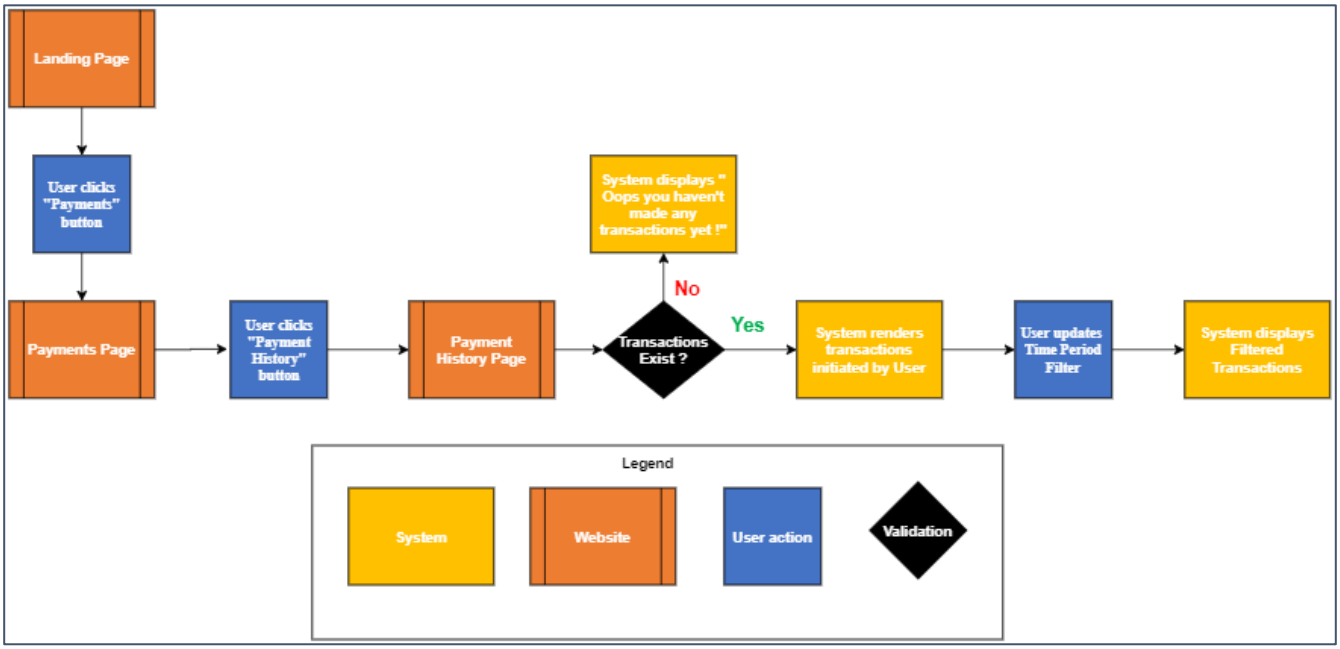

Figure 4: Task Flow - View Payment History

2.2 User-Centered Design Approach

Considering the goals and frustrations of the user personas in our application, we decided to develop an application that's simple, elegant and which fulfils all the major needs of our personas. Most of our User personas require an application which can optimize the time spent by users on expense management and bill settlement. A primary requirement from the user personas was the automation of repetitive tasks. In addition, the personas desired an uncomplicated user interface. In summary our personas require an application which alleviates the need for them to manually keep track of expenditure, bills, debts, and other data. As such we have developed and rendered concise visualizations and automates solutions such as debt simplification [5], which aggregates all existing debts and displays the resultant sum that each user owes another, to ensure that users have a seamless experience while using the app.

Low Fidelity Prototypes (Payment Module) [1,3]***:***

Figure 5 represents the Payment Initiation screen in the application.

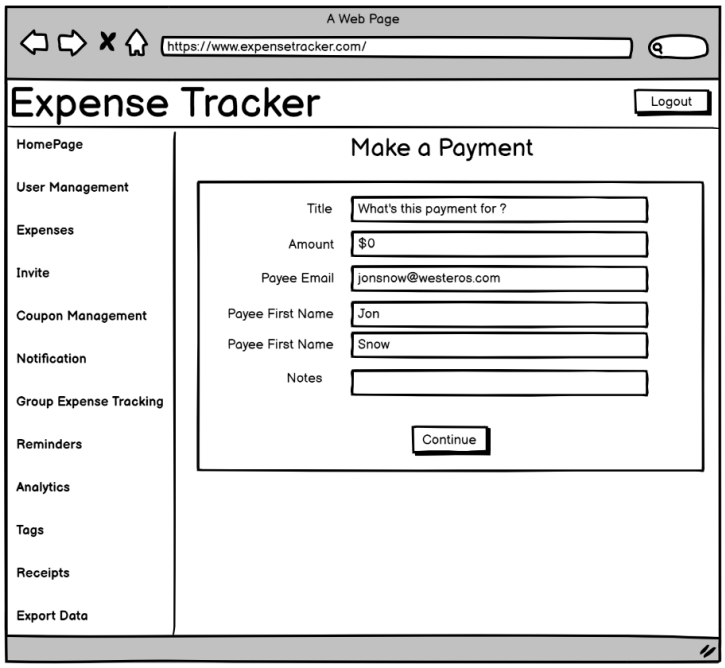

Figure 5: Wireframe - Initiate a Payment

Figure 6 represents the screen where Users input their card details to make a payment.

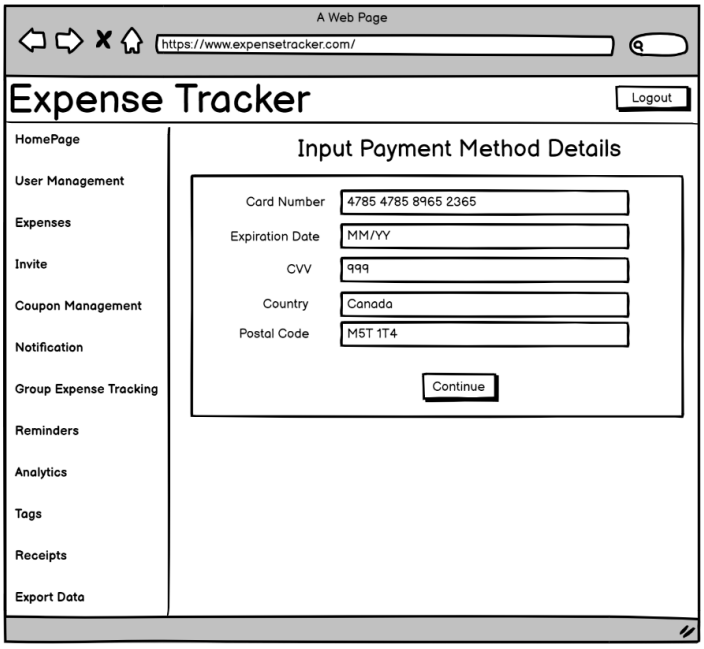

Figure 6: Wireframe - Input Payment Method Details

Figure 7 represents the Payment Status screen in the application.

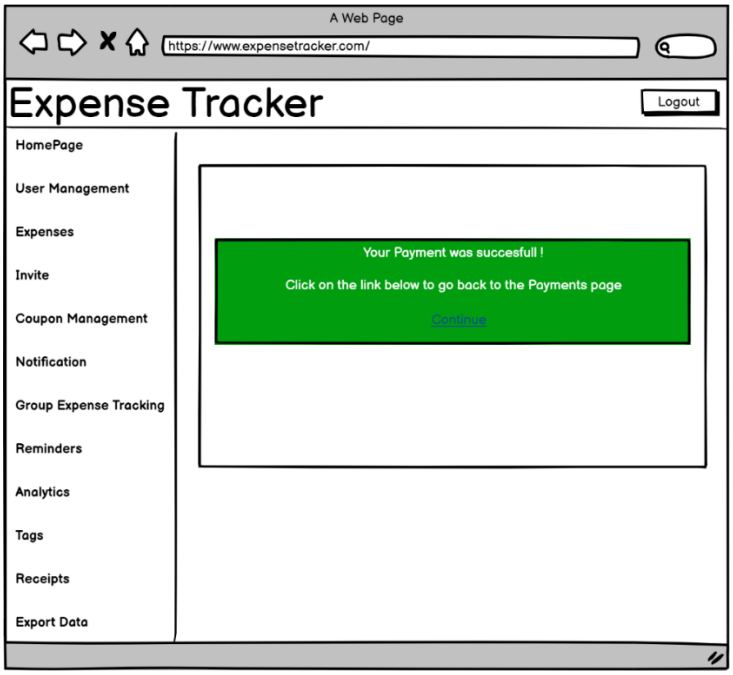

Figure 7: Wireframe - Payment Status

Figure 8 represents the Payment History screen in the application.

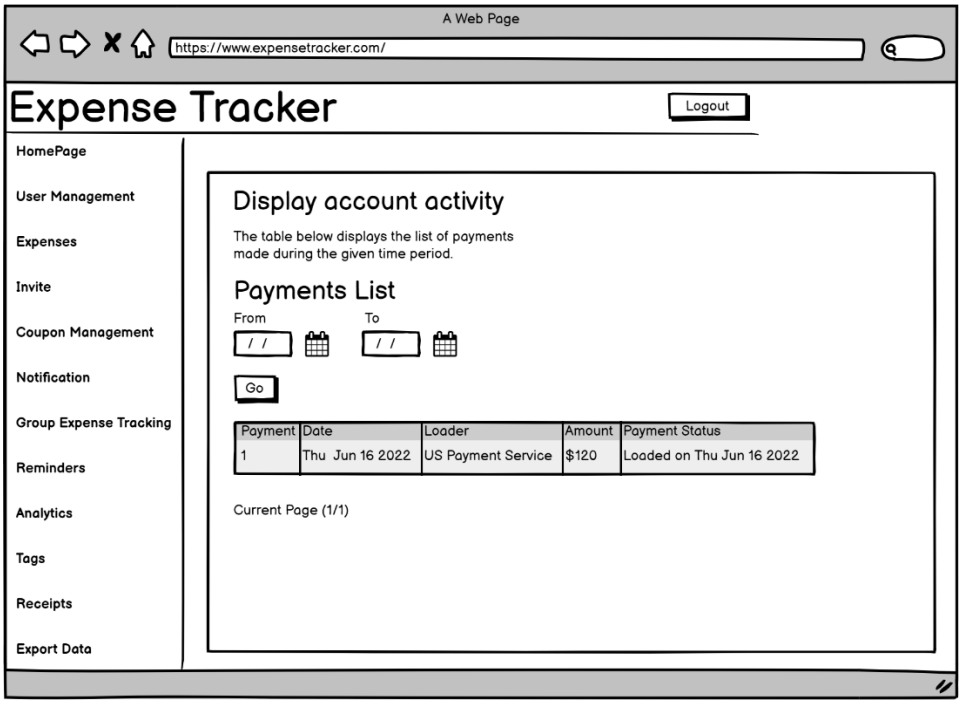

Figure 8: Wireframe - Transaction History

User Interface:

Figure 9 represents the User Interface where Payers can enter the details of Payees and the amount paid through the transaction.

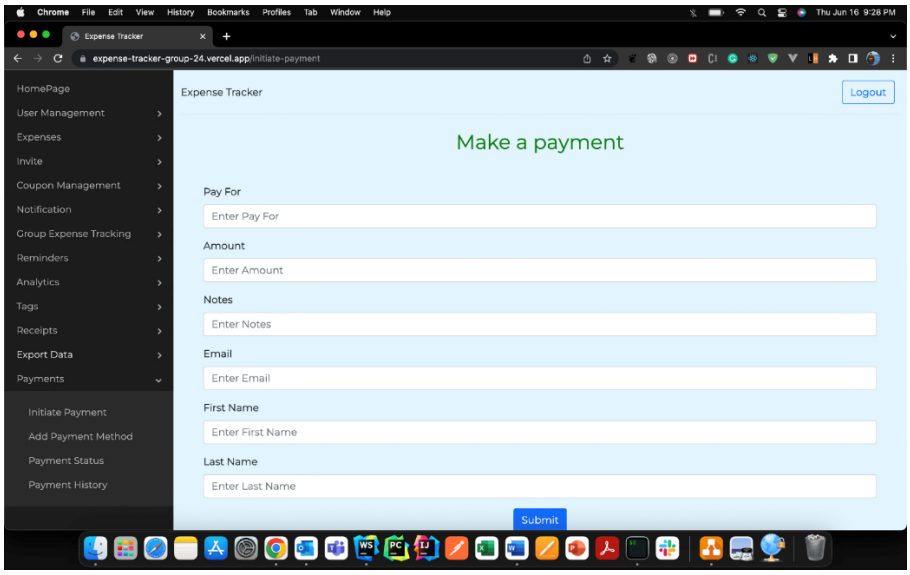

Figure 9: Initiate a Payment

Figure 10 exemplifies the page where users can enter the card information and proceed with the transaction.

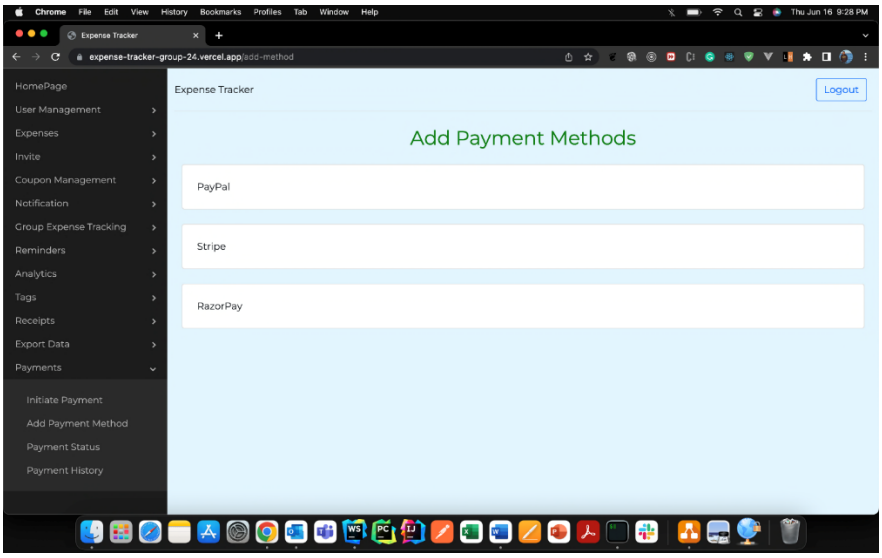

Figure 10: Add Payment Methods

Figure 11 is an example of the page where users view the status of the transaction initiated by them.

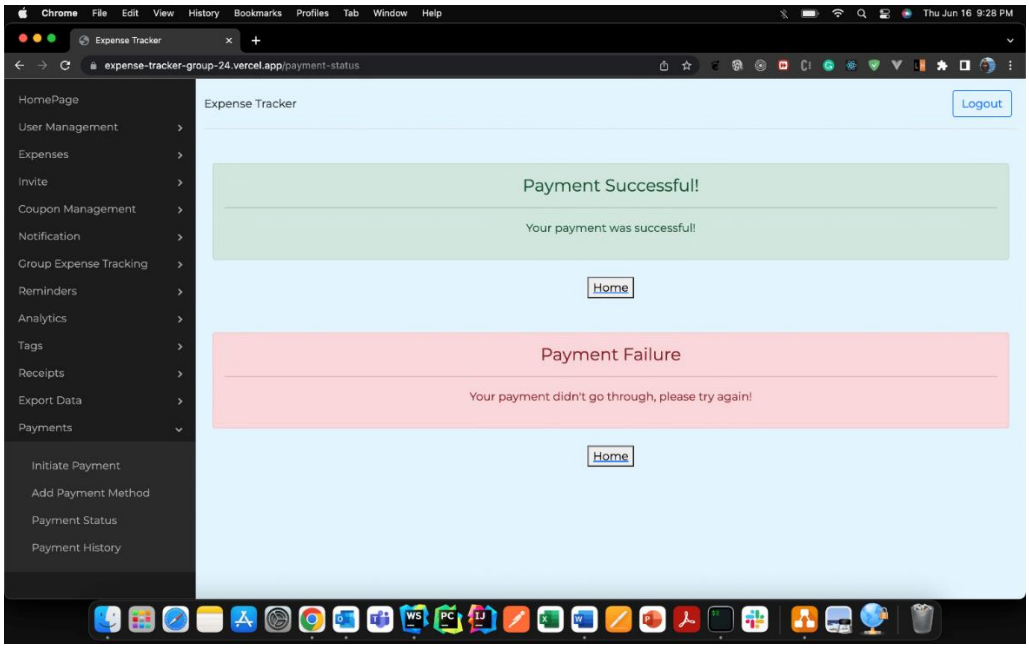

Figure 11: Payment Success and Failure

Figure 12 represents the page where Payers can view a history of all transactions that were initiated by them in the past.

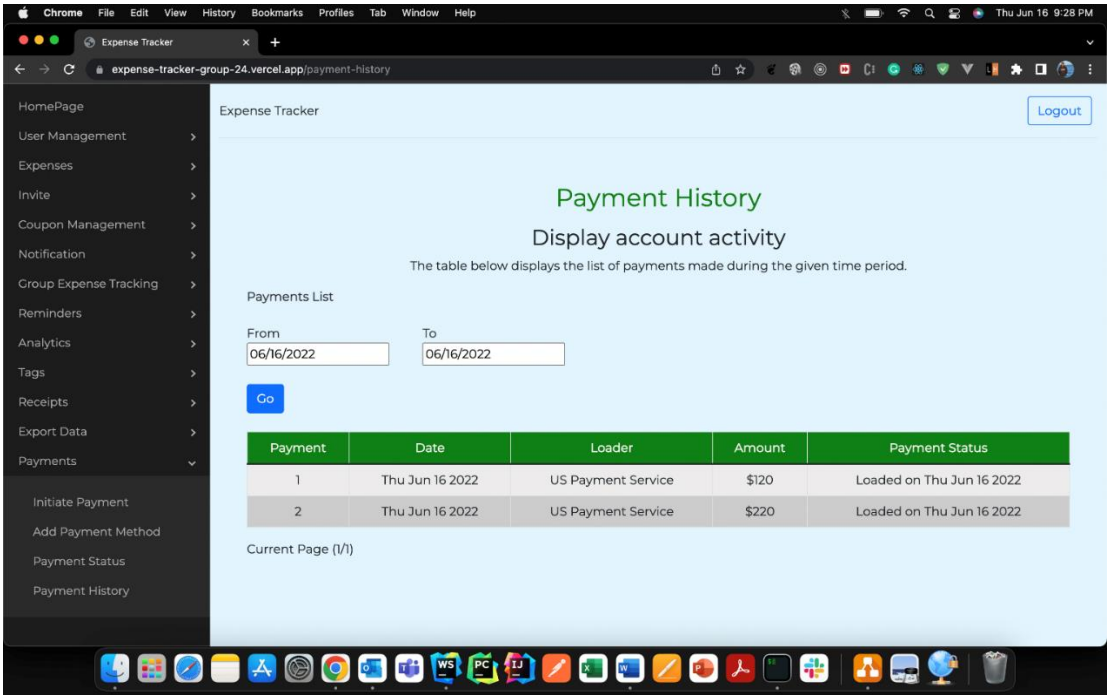

Figure 12: Payment History

2.3 Application Architecture:

Figure 13 represents the architecture of the Expense Tracker application. Our application's architecture is built on a conventional MVC model. JavaScript, HTML, and CSS will be used to create our client tier (View), which will be built on the ReactJS framework. The user will interact with this level of the architecture to access the functionalities of our application. The MVC design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects. The model contains only the pure application data; it contains no logic describing how to present the data to a user. The view presents the model's data to the user. The controller exists between the view and the model. It listens to events triggered by the view and executes the appropriate reaction to these events. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view. Node.js and Express.js will be used to create the Business Logic Tier (Controller), which serves as the Application Server and a communication link between the Client Tier and Database Tier. This tier will deliver HTML pages to the user's device, accept user HTTP requests, and respond appropriately. Our Database Tier (Model) will be hosting MySQL. This is where we will store all the crucial data our application needs to function.

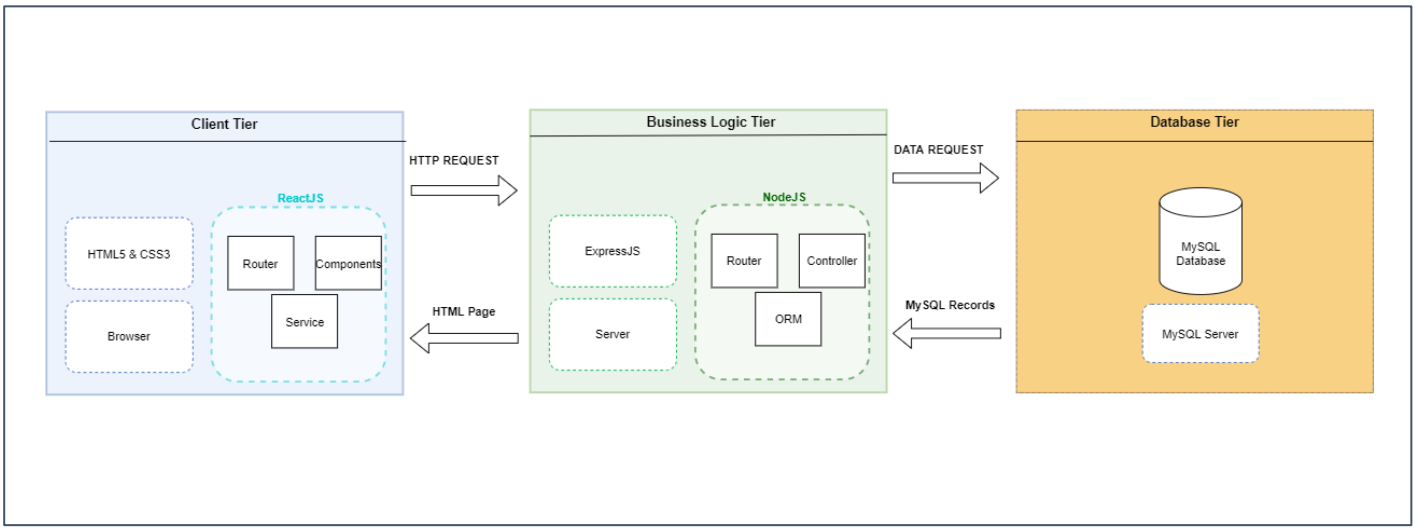

Figure 13: Expense Tracker - Architecture Diagram

Front End:

We have used the following frameworks and libraries for the development of the User Interfaces in our application.

Framework:

ReactJS – A component-based, declarative and stateful framework used to render interactive and complex User Interfaces.

Libraries:

1. Bootstrap – An open-source CSS framework directed at responsive, mobile-first front-end web development. It features numerous HTML and CSS templates for UI interface elements.
2. SweetAlerts – A responsive, customizable, accessible replacement for JavaScript's popup boxes.
3. React Chart-JS – React component for Chart.js, a charting library, to render customized charts in a web application.
4. Emotion – A library designed for writing CSS styles with JavaScript.
5. Redux - An open-source JavaScript library for managing and centralizing application state.
6. React Toastify – A open-source library for adding and rendering notifications in web applications.

Back End:

We have primarily used NodeJS and ExpressJS for the development of the back end of our application.

Frameworks:

1. NodeJS – A 'Promise based' HTTP client for the browser and node.js.
2. ExpressJS - A back-end web application framework for NodeJS that provides broad features for building web applications. It's a layer built on the top of the NodeJS that helps manage servers and routes.

Libraries:

1. Axios [7] – An open-source CSS framework directed at responsive, mobile-first front-end web development. It features numerous HTML and CSS templates for UI interface elements.
2. Stripe [6] - React Stripe.js is a thin wrapper around Stripe Elements. It's used to make use of Stripe elements in ReactJS.

Database:

MySQL - An open-source relational database management system used for storing data relevant to the application.

2.4 Application Workflow:

Interaction Design:

Scenarios [1]***:***

Scenario 1:

A Payer desires to add a new or alternate payment method to the System. The User navigates to the Payment tab and initiates a payment. The System directs the User to the payment method page. The User

enters the details of the Payment method, such as Card Number, Expiration Date and Card Verification Value (CVV). The System accepts the details of the payment method from the User and proceeds with the transaction.

Persona: Debt Settler

Feature: Input Payment Method details

Need: Add details of payment method which can be used to make payments through the application

Context: Payer inputs details of a Payment method after initiating a payment in the System

Scenario 2:

A Payer wants to make a payment to a Payee. The Payer navigates to the Payments tab and makes a payment to a Payee. The Payee either receives the sum credited by the Payer or no transaction occurs if the Payer does not have a sufficient balance to proceed with the transaction.

Persona: Debt Settler

Feature: Initiate a Payment

Need: Settle a debt or a payment through the application

Context: Payer uses the app to initiates a Payment to a Payee, the System processes the Transaction

Scenario 3:

A Payer desires to view the status of recent payment. The User navigates to the Payments site and makes a payment to a Payee. The System renders a status (Success /Failure) corresponding to the transaction initiated by the Payer.

Persona: Debt Settler

Feature: View Payment Status

Need: View the status of a Payment after it has been initiated

Context: Payer initiates a Payment to a Payee and views the status of the transaction

Scenario 4:

An Expense Analyst navigates to the Payments tab and selects the option to view Payment History. The System displays a list of Payments made by the User, corresponding to the filter inputs.

Persona: Expense Analyst

Feature: View Payment History

Need: View historical data relevant to payments

Context: A Payer updates filters and views historical payments in the application. The filters include Time Period and Amount thresholds.

Use Cases [1]***:***

1. In-App Payment Interface

1.1 Initiate a payment

Assumption: User is already authenticated and is about to initiate a payment and has a valid payment method (Debit / Credit Card).

1. User selects the "Payments" link on the Navbar, to make a Payment.
2. System displays the Payments webpage.
3. User inputs the Payee's email.
4. System authenticates the email.
   4.1. User has entered an invalid email
   4.2. System displays "Please enter a valid email" message.
   4.3. System prompts user to try again.
      4.3.1. User enters a valid email address.
5. User inputs the Payee's name.
6. System authenticates the Payee's name.
   6.1. User has entered an invalid name.
   6.2. System displays "Name should contain only alphabets" message.
   6.3. System prompts user to try again.

6.3.1. User enters a valid name.

7. User inputs an amount to be paid.
8. User reviews the transaction details.
9. User clicks on “Continue”.
10. System redirects user to the “Input Payment Method Details” webpage.

1.2 Input Payment Method Details

Assumption: User has initiated a payment from the Payments page and has been redirected to the “Input Payment Details” page.

1. System displays the “Input Payment Details” webpage.
2. User is prompted for the details of the card.
3. User enters the card number and navigates to the next field.
4. System authenticates the card number.
   4.1. User had entered an invalid card number.
   4.2. System displays “Please enter a valid card Number” message.
   4.3. System prompts user to try again.
      4.3.1. User enters a valid card number.
5. User selects a Month, corresponding to the Expiry Month on the card, from the dropdown and navigates to the next field.
6. User selects a Year, corresponding to the Expiry Year on the card, from the.
7. User enters the CVV on the back of the card.
8. User clicks on “Checkout”.
9. System processes payment and redirects user to the payment status page.

1.3 View Payment Status

Assumption: The user had recently initiated a transaction.

1. User is redirected from the Payments webpage to the “View Payment Status” webpage after initiating a payment and furnishing car details.
   1.1. User’s account has insufficient balance to complete the transaction.

1.2. System displays a “Transaction Failure” message.

1.3. System redirects User to the Payments webpage.

2. Transaction was successful and System displays a “Transaction Successful” message.
3. System redirects user to the “Payments” page.

1.4 View Payment History

1. User clicks on the link to the “Payments” webpage from the Navigation panel.
2. System renders the “Payments” webpage.
3. User clicks on the “Payment History” button.
4. System renders the “Payment History” page.

4.1. User has not made any Transactions.

4.2. System displays a “Oops looks like you have no transactions yet!” message.

5. System loads the list of transactions made by the User.
6. User updates the Filter corresponding to Time Period
7. User updates the Filter corresponding to the Lower limit of the Amount Thresholds.
8. User updates the Filter corresponding to the Upper limit of the Amount Thresholds.
9. System updates the list of transactions made by the User.

User Click Streams:

Figure 14 represents the user mouse clicks when the user attempts to initiate a payment.

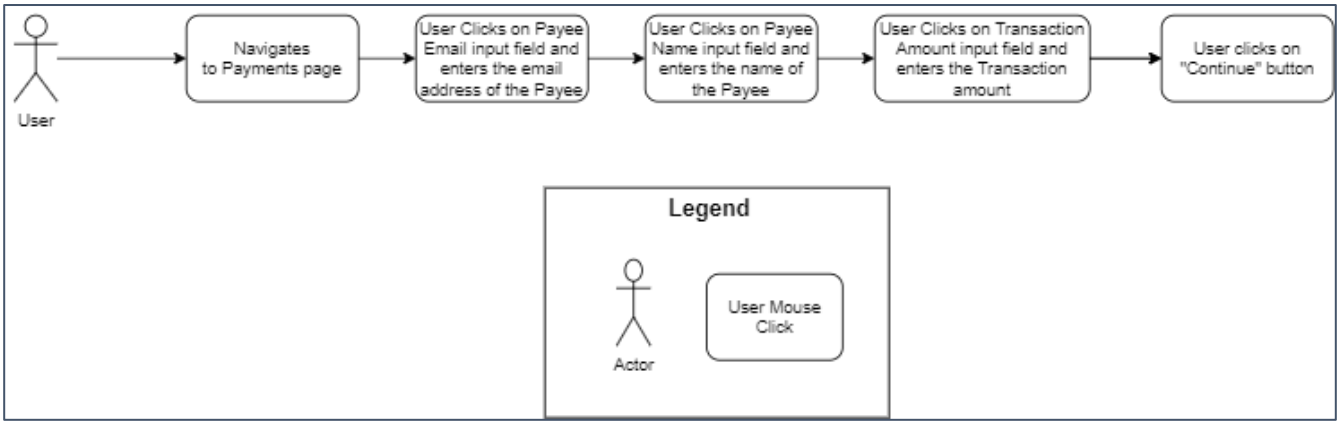

Figure 14: Initiate a Payment - User Click Stream

Figure 15 represents users' mouse clicks when they attempt to enter the details of their payment method.

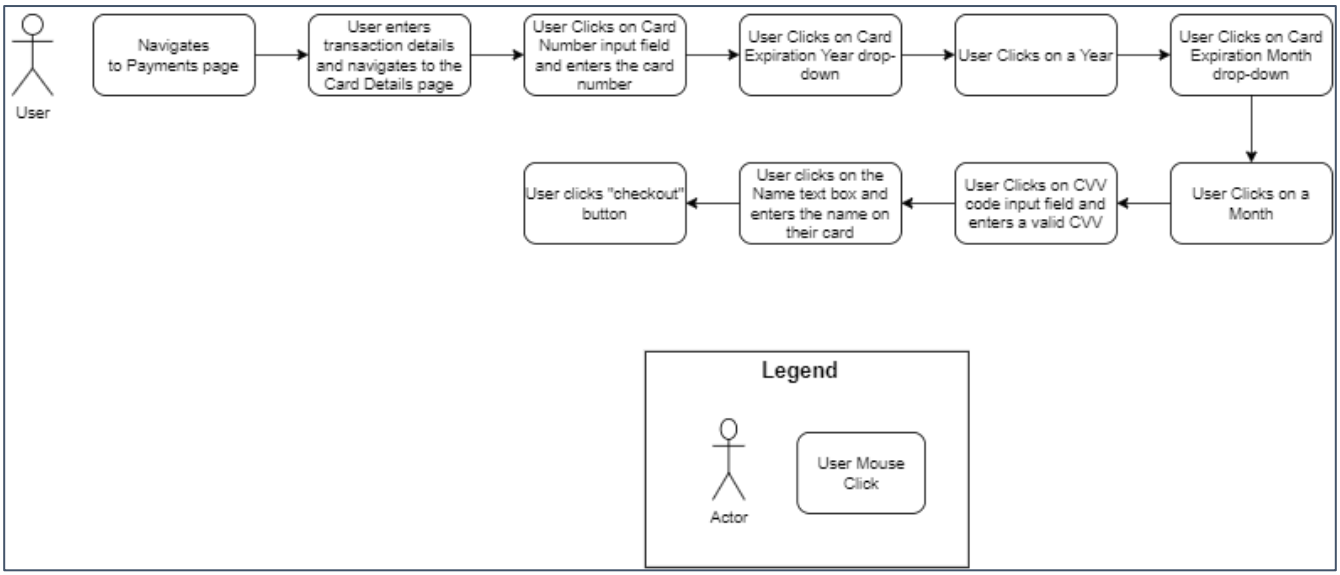

Figure 15: Enter Payment Method Details - User Click Stream

Figure 16 represents users' mouse clicks after they are redirected to the 'payment status' page.

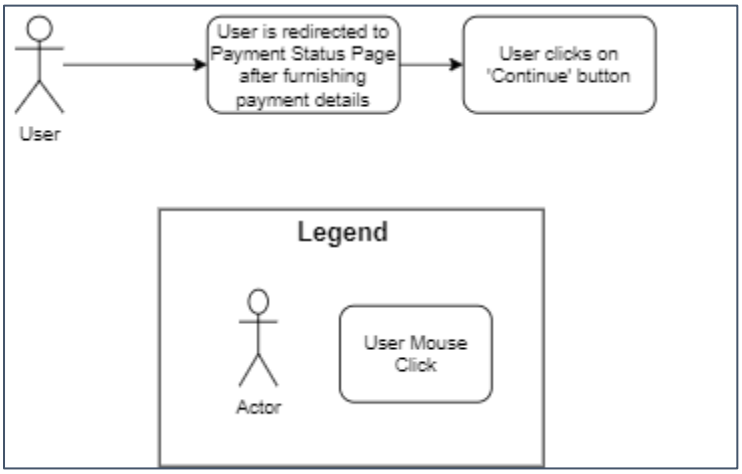

Figure 16: View Payment Status - User Click Stream

Figure 17 represents users' mouse clicks when they attempt to view a history of all the payments initiated by them.

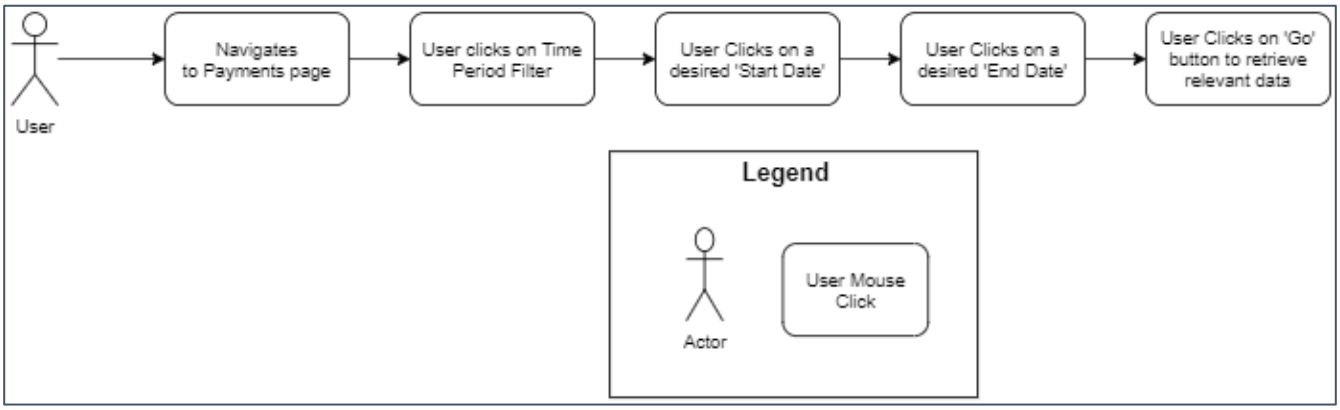

Figure 17: Access Payment History - User Click Stream

Overall Application Workflow:

Figure 18 represents the overall workflow of our application. The application communicates with APIs to obtain data and display it to the user. The backend can be connected to third-party APIs or a database server to obtain required records, process it, and send it back to the requester. When an HTTP request is made, the front-end will be triggered and ReactJS will pick up the request [8]. The request will be passed internally in ReactJS with Route sending a request for the view to View/Template.

The View will submit a request to the Controller. The Controller will then create an HTTP request to a RESTful (Representational state transfer) endpoint to the Server Side, which is Express/Node.js. The request will then be passed internally with Express/Node.js from its Route to the Controller/Model.

The Controller/Model will make a request to interact with the Database Server that hosts MySQL. The MySQL server will process the request and respond the callback to ExpressJS/NodeJS. ExpressJS/NodeJS sends a response to the ReactJS Controller. The Controller would then respond with a view which the User interacts with.

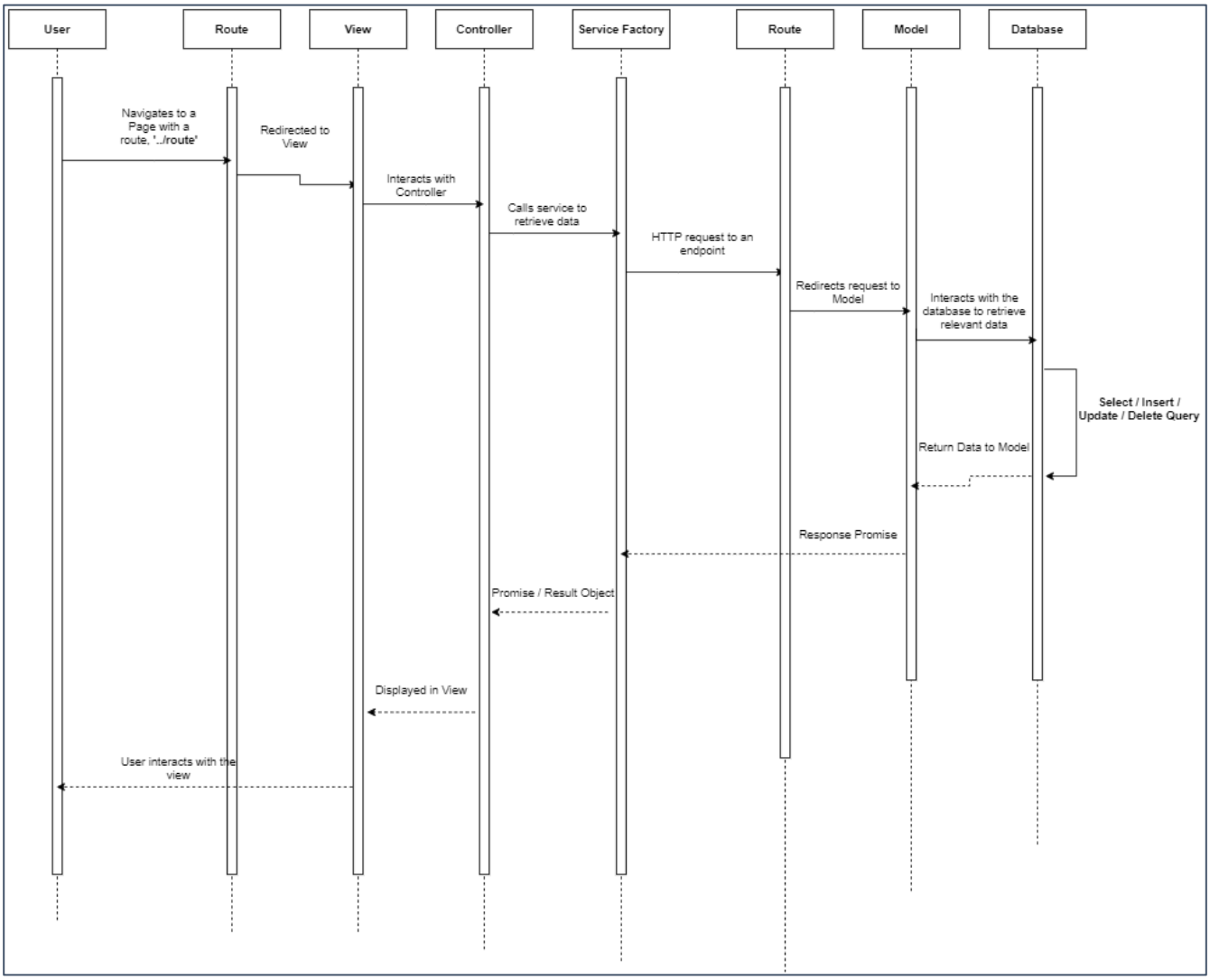

Figure 18: Overall Application Workflow

In-App Payment Feature - Process Flow:

Figure 19 represents the entire API process flow of the In-app Payment feature when a user initiates a transaction. Every transaction in a contemporary Stripe integration makes use of an item called a 'PaymentIntent'. It indicates a user's intention to get payment. This object keeps track of the actions users take to fulfilling that intent. A Payment Intent starts with the status 'requires_payment_method' [9]. To proceed further, Stripe needs details about the customer's payment method either a card number or credentials for some other payment system.

An integration represents these details using an API object called a 'PaymentMethod'. The next status is 'requires_confirmation'. Users must confirm that they intend to pay and that they intend to do it using the method they provided, this conformation occurs when users click on the 'Pay' button in the application. When the customer clicks Pay or otherwise confirms their intent, an integration notifies

Stripe with an API call. The intent's subsequent state is called the 'processing state', and at this point Stripe attempts to process the payment. Then, Stripe updates the intent's state with the outcome: either succeeded or back to 'requires_payment_method' if it fails.

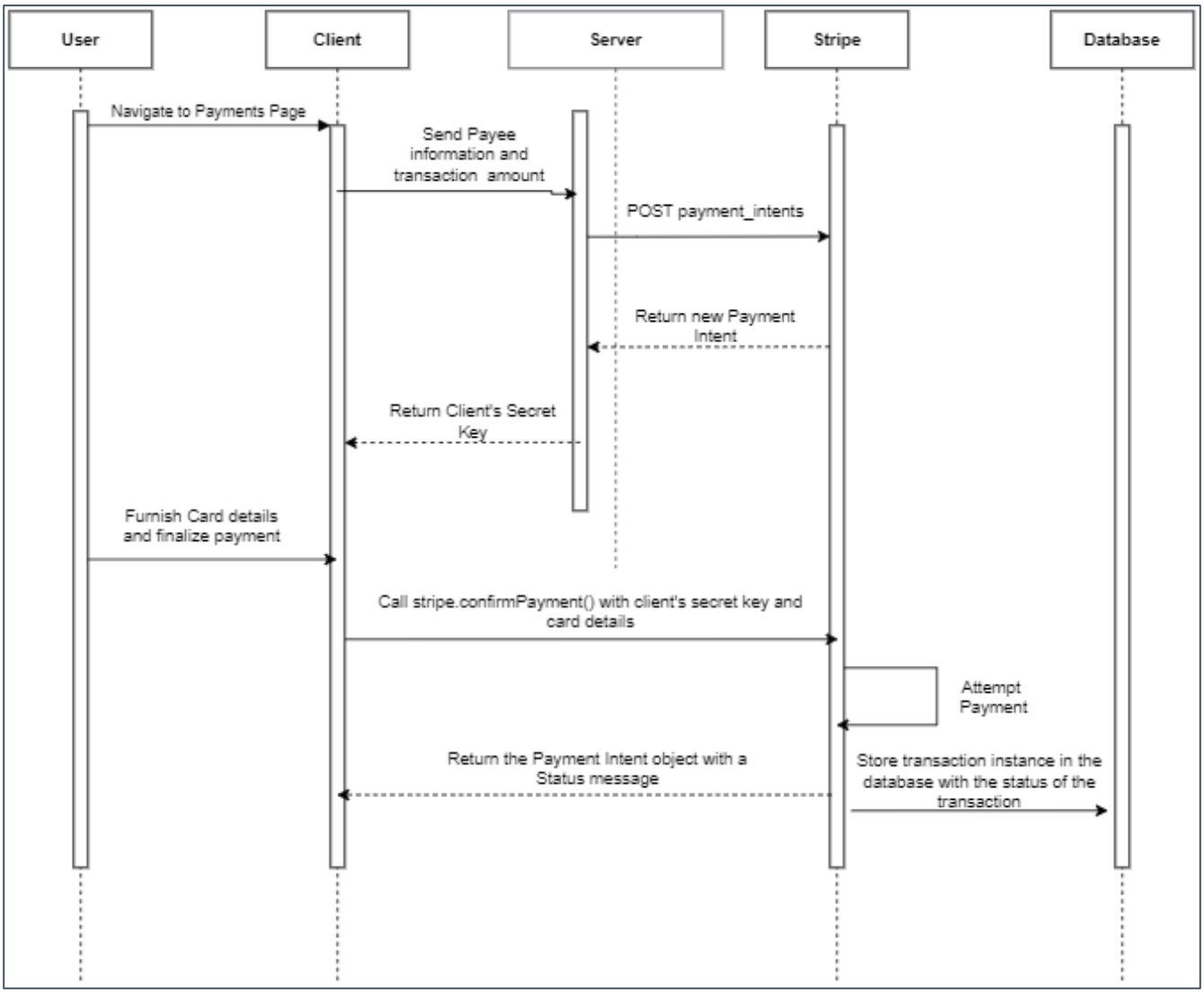

Figure 19: Payment Integration with Stripe - Process Flow

2.5 Folder Structure:

Figures 20-23 represent the directory structure of our application. The directories in our application are primarily split into two folders, namely, 'frontend' and 'backend' [10]. The 'frontend' directory contains

the code for all the ReactJS components. Each component has its own dedicated directory for holding all relevant scripts. In addition to directories for components, the frontend directory also contains a directory for the Redux module, as shown in Figure 22. The backend directory contains directories for the controllers, routes and models used in the server side scripts.

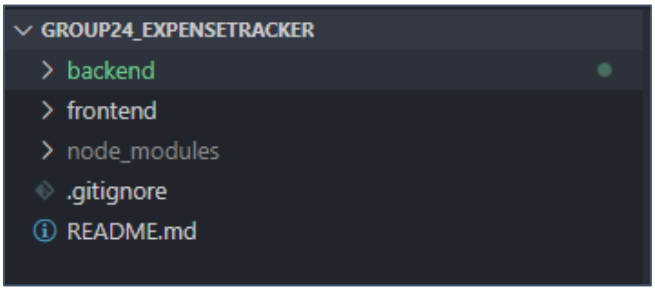

Figure 20: Complete Directory Structure

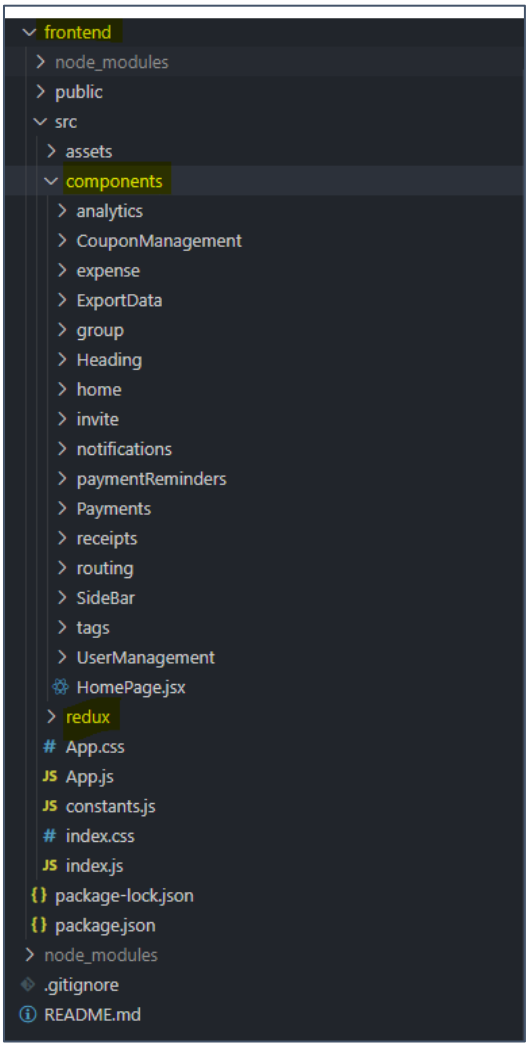

Figure 21: frontend directory structure

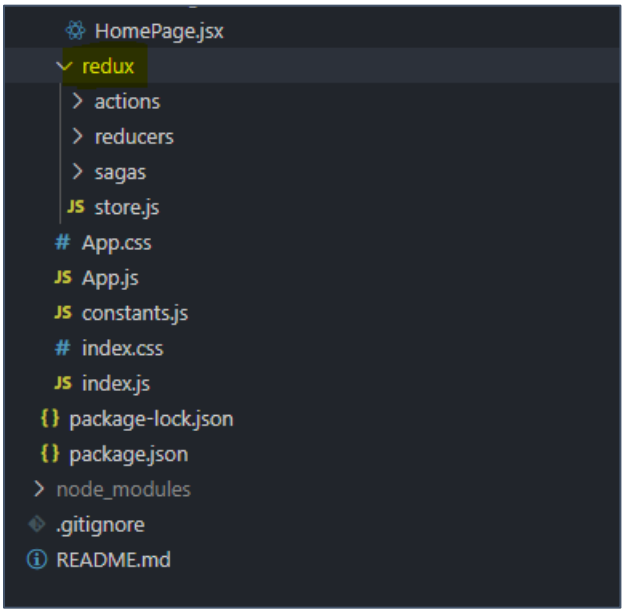

Figure 22: Redux directory

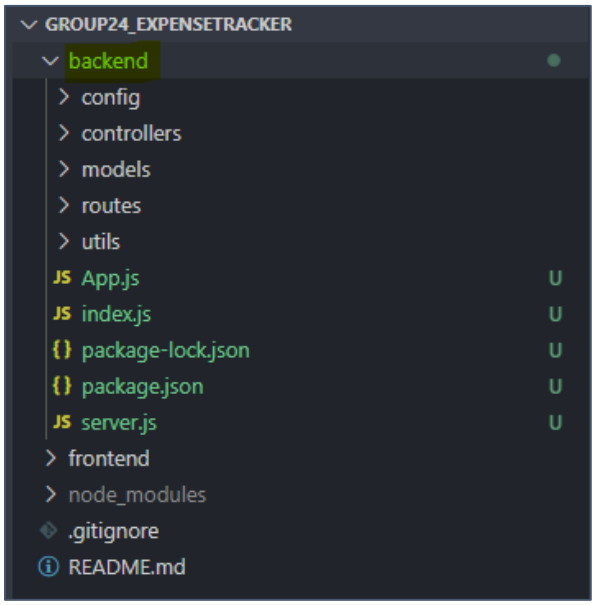

Figure 23: backend directory structure

References

[1] U. Abhishek, Y. Vatsal, T. Benny, V. Devarshi, V. Ayush, T. Smith "CSCI 5709 Project Proposal Document." Summer Term, Dalhousie University, [online document], 2022. [Accessed June 25, 2022]

[2] "Split expenses with friends.", Splitwise, 2022. [Online]. Available: https://www.splitwise.com/. [Accessed: 26- Jun- 2022]

[3] "Balsamiq Cloud", Balsamiq.cloud, 2022. [Online]. Available: https://balsamiq.cloud/. [Accessed: 17- Jun- 2022]

[4] "Flowchart Maker & Online Diagram Software", App.diagrams.net, 2022. [Online]. Available: https://app.diagrams.net/. [Accessed: 27- Jun- 2022]

[5] "Is Splitwise the Best Bill Splitting App for Travelers? [Review]," *www.pilotplans.com*, Sep. 20, 2014. https://www.pilotplans.com/blog/splitwise-review (accessed Jun. 27, 2022).

[6] "Stripe | Payment Processing Platform for the Internet," *stripe.com*, Aug. 19, 2014. https://stripe.com/en-ca (accessed Jun. 26, 2022).

[7] axios, "axios/axios," *GitHub*, May 31, 2019. https://github.com/axios/axios (accessed Jun. 26, 2022).

[8] "MEAN Stack," *Programmatic Ponderings*, Jul. 09, 2015. https://programmaticponderings.com/tag/mean-stack/ (accessed Jun. 29, 2022).

[9] "Stripe API reference – The PaymentIntent object – curl," *stripe.com*, Oct. 19, 2016. https://stripe.com/docs/api/payment_intents/object (accessed Jun. 26, 2022).

[10] N. Limane, "How to structure your Express and Node.Js project," *DEV Community*, Jan. 14, 2022. https://dev.to/nermineslimane/how-to-structure-your-express-and-nodejs-project-3bl (accessed Jun. 27, 2022).