System Design Document

Project CashTag

EECS 3311 - Sprint 2

By: Priya Gill, Abrar Kassim, Eesha Fawad,
Melanie-Jane Mushayev, Ashal Iman

Table of Contexts

CRC Cards……..……3

System Interaction……..……...16

Architecture Overview……..……...16

Architecture Diagram……..……...17

Database Schemas……18

System Decomposition ……..……19

CRC cards

User & Authentication

Class name: User	
Responsibilities	**Collaborators**
- Store user information (id, firstname, lastname, email, createdAt)	- UserController
- Allow retrieval of all users	
- Allow creation of new users	
- Validate and maintain user data integrity	

Class Name: UserRepository	
Parent Class = JpaRepository, Sub Classes = None	
Responsibilities	**Collaborators**
- Manages storage and retrieval of User data	- User - UserController
- Use Spring Data JPA methods such as findAll()	

Class Name: UserController	
Responsibilities	**Collaborators**
- Retrieves user data from UserRepo	- User - UserRepository
- Handles the HTTP requests to interact with user data	
- Map routes through api/users	

Class Name: Login	
Responsibilities	**Collaborators**
- View component for the user log in page.	- UserController
- Request and save login fields such as email and password. - Syncs users once logged in if they have groups so invited members (non-owners) can view groups.	

Class Name: Sign Up	
Responsibilities	**Collaborators**
- View component for the user sign up page.	- UserController
- Shows a form where users can input their details (first and last name, email, and password) to create an account and get access to the site CashTag. This account will allow them to save their data and view it.	

Expenses

Class Name: ExpenseController	
Responsibilities	**Collaborators**
- Handles API requests for adding, deleting, or retrieving expenses.	- Expense, - ExpenseRepository, - Category, - ExpenseRepository,
- Has a separate paginated endpoint for UI views so not all expenses are loaded at once, and a get all expenses endpoint.	

Class Name: Expense	
Responsibilities	**Collaborators**
- Stores and manages the data related to an expense. - Has getters and setters for all Expense entity fields such as expense_id, user_id, category, amount, description, date.	- ExpenseController - Category

Class Name: ExpenseRepository	
Parent Class = JpaRepository, Sub Classes = None	
Responsibilities	**Collaborators**
- Retrieve expense records from the database. - Save, delete or edit expenses directly in the database. - Query expenses by user, category, or date.	- Expense - ExpenseController

Class Name: AddExpenseModal	
Responsibilities	**Collaborators**
- View component for adding a new expense. - Sends data to ExpenseController after the user adds an expense so that it can be saved.	- Dashboard - ExpenseController

Class Name: EditExpenseModal	
Responsibilities	**Collaborators**
- Show a form to edit existing expense	- ExpenseController - Dashboard
- Display all categories	
- Send update info through API once expense form is submitted	

Dashboard Page

Class Name: BarChartComponent	
Responsibilities	**Collaborators**
- Get user expense for the month	- ExpenseController - Dashboard
- Process and format data for display in a bar chart	
- Dynamically update for any edits in the expenses	

Class Name: Dashboard (from dashboard/page.js)	
Responsibilities	**Collaborators**
- The main dashboard UI page that imports and renders all of the components for expense tracking.	- AddExpenseModal, - BarChartComponent, - EditExpenseModal, - AddBudgetModal, - EditBudgetModal, - BudgetBar

Class Name: TotalAndCategory.js	
Responsibilities	**Collaborators**
- Calculate and display the total expenses for the selected month	- Dashboard - Category - ExpenseController
- Show a breakdown by category as to how much was spent in each category of the selected month	
- Show a list of the current month's individual expenses with description, amount, date, category and an option to either edit or delete the expense	
- Update the Dashboard whenever the selected month or the expense data changes	

Class Name: MonthSelector.js	
Responsibilities	**Collaborators**
- Provide a UI control like button for selecting the month	- Dashboard - ExpenseController - BudgetController
- Keep track of what month is selected	
- Keep the Dashboard updated whenever month changes so the data can be reloaded	

Class Name: PieChartComponent.js	
Responsibilities	**Collaborators**
- Take expense data which is grouped by category and show it as a piechart	- ExpenseController - Dashboard - Category
- Show the proportion of total spending in % that goes to each category for the selected month	
- Update the chart visually whenever a different month is selected or expenses change	

Budgets

Class Name: Budget	
Responsibilities	**Collaborators**
- An entity, structure, and model class that represents and stores information related to Budgets. - A “budget” table is created in the database based on this entity.	- BudgetController - Category
- Has getters and setters for all Budget entity fields such as budget_id, user_id, expense category, month, max amount, current amount spent.	

Class Name: BudgetController	
Responsibilities	**Collaborators**
- Handle all HTTP requests related to retrieving, adding, updating, and deleting monthly expense budgets.	- Budget - BudgetRepository - ExpenseRepository - Category
- Retrieve budgets filtered by user, month, and category from the database.	
- Calculate the current amount spent on related expenses when creating a budget.	
- Return list of expense categories that do not have a budget yet for a given month.	

Class Name: BudgetRepository	
Parent Class = JpaRepository, Sub Classes = None	
Responsibilities	**Collaborators**
- Retrieve monthly budgets from the database.	- Budget - BudgetController - BudgetController
- Save, delete or edit budgets directly in the database.	
- Query budgets by user, month, and category.	

Class Name: AddBudgetModal.js	
Responsibilities	**Collaborators**
- A view component for displaying a modal popup for adding a new budget.	- Dashboard - BudgetController
- Sends data (POST request) to BudgetController after the user adds a new budget so that it can be saved	

Class Name: EditBudgetModal.js	
Responsibilities	**Collaborators**
- A view component that displays a pop-up modal that allows the user to edit an existing budget. - Pre-fill the budget amount and category in the edit form, based on the selected budget. - Send the updated budget object to the BudgetController as a PUT request.	- Dashboard - BudgetController

Class Name: BudgetBar.js	
Responsibilities	**Collaborators**
- A view component that displays all of the budgets for the selected month, in the form of progress bars. - Shows the “amount spent” vs. the “budget limit” visually for each budget category. - Fetch budget data by sending a GET request to BudgetController - Budget Bars update dynamically when the selected month changes,when budgets are edited, or new expenses are added.	- Dashboard - BudgetController

Groups Dashboard & Management

Class Name: Group	
Responsibilities	**Collaborators**
- Shows a shared expense group - Store the data related to group such as id, name, owner	- GroupMember - User - GroupController

Class Name: GroupMember	
Responsibilities	**Collaborators**
- Show the link between a user and the groups they are in - Store members' data	- Group - User - GroupController

Class Name: GroupExpense	
Responsibilities	**Collaborators**
- Stores and manages the data related to a group expense. - - Has getters and setters for all Group Expense entity fields such as expense_id, user_id, paidby, category, total, shares, description, date.	- SharedExpense - User - GroupExpenseController

Class Name: SharedExpense	
Responsibilities	**Collaborators**
- Stores and manages the data related to a shared expense within members of a group. - Has getters and setters for all Group Expense entity fields such as expense_id, user_id, getExpense, getDebt, isSettled.	- GroupExpense - User - GroupExpenseController

Class Name: GroupExpenseController	
Responsibilities	**Collaborators**
- Finds all users in a group - Find the expenses in the group - Allows to add an expenses in a specific group - Tracks who paid for the item - Allows for updating the expense - Allows for deleting the expense	- GroupExpenseRepository - GroupRepository - UserRepository - SharedExpenseRepository - GroupMemberRepository

Class Name: GroupController	
Responsibilities	**Collaborators**
- Creates a new group - Invites users via email only if they are already a member of CashTag. - Syncs all users from login so that invited users can see the groups once they log in, not just the group owner - Returns all groups that belong to a user & returns all members within that group	- GroupExpenseRepository - UserRepository - GroupMemberRepository

Class Name: GroupRepository	
Parent class = JpaRepository	
Responsibilities	**Collaborators**
- Manages storage and retrieval of Group data - Use Spring Data JPA methods such as findByOwnerId()	- GroupController - Group

Class Name: GroupMemberRepository	
Parent class = JpaRepository	
Responsibilities	**Collaborators**
- Manages storage and retrieval of Group data - Use Spring Data JPA methods such as findGroupId() and findByUserId	- GroupMember - GroupController - GroupExpenseController

Class Name: GroupExpenseRepository	
Parent class = JpaRepository	
Responsibilities	**Collaborators**
- Manages storage and retrieval of Group and Expense data. - Use Spring Data JPA methods such as findGroupId()	- GroupExpense - GroupMemberRepository

Class Name: Group UI	
Responsibilities	**Collaborators**
- First thing users see, they can create a new group - It displays the groups the users already have	- GroupMember - Group

Class Name: Group ID UI (Details Page)	
Responsibilities	**Collaborators**
- When users are logged in they see the details of the group. The members of group, the table of expenses including how much each person pays - They're able to add/edit/delete an expense - They can invite more members and pay for an expense. Payment is only for current members, new members who were not part of a previous expense will not be charged.	- GroupController - SharedExpense - AddGroupExpenseModal - Category - GroupMember - EditExpenseModal

Class Name: SharedExpenseRepository	
Parent class = JpaRepository	
Responsibilities	**Collaborators**
- Manages storage and retrieval of User data within an a Group expense - Use Spring Data JPA methods such as findGroupId() and findByUserId	- GroupController - SharedExpense

Category

Class Name: Category	
Responsibilities	**Collaborators**
- An enum object that represents options that users can select when adding an expense.	

Class Name: CategoryController	
Responsibilities	**Collaborators**
- returns all values of the Category enum as a JSON list	- Category

Transaction History

Class Name: Transactions	
Responsibilities	**Collaborators**
- Displays a list of all expenses for the current user	- ExpenseController, - Expense - AddExpensedModal - EditExpenseModal - Category
- Allows the user to view details of each expense such as description, category, amount and date	
- Call the backend to fetch expenses and refresh the list if any changes made	

System Interaction

Our application runs on a Spring Boot backend environment, which interacts with a PostgreSQL database through Hibernate (JPA) for object-relational mapping. It also exposes a set of RESTful APIs through Spring Boot Controllers, that the frontend can access via HTTP requests, and this enables communication between the frontend and the backend services.

The system depends on the following components in its operating environment:

- **Operating System:** Any modern OS (Windows, macOS, or Linux) capable of running a Java Virtual Machine (JVM).
- **Database:** PostgreSQL server with proper configuration and connection credentials. For our project we have decided to go with a PostgreSQL database that is hosted through Supabase.
- **Backend Framework:** Spring Boot with Hibernate for Object-Relational Mapping and REST API management.
- **Network:** Stable HTTP connection between frontend and backend, typically running on ports 8080 (backend) and 3000 (frontend).
- **Frontend Environment:** Modern web browser and Node.js environment for running and building the UI.

Architecture Overview:

Our system is following an MVC (model view controller) architecture.

Model Layer (Backend)

The model layer of the application defines core entities and the business logic of the system. These entities represent users, expenses, budgets, groups, and more, and they communicate directly with the PostgreSQL database through corresponding JPA repositories. These entities are represented as tables in the database, and the model layer helps enforce relationships between these entities.

Controller Layer (Backend)

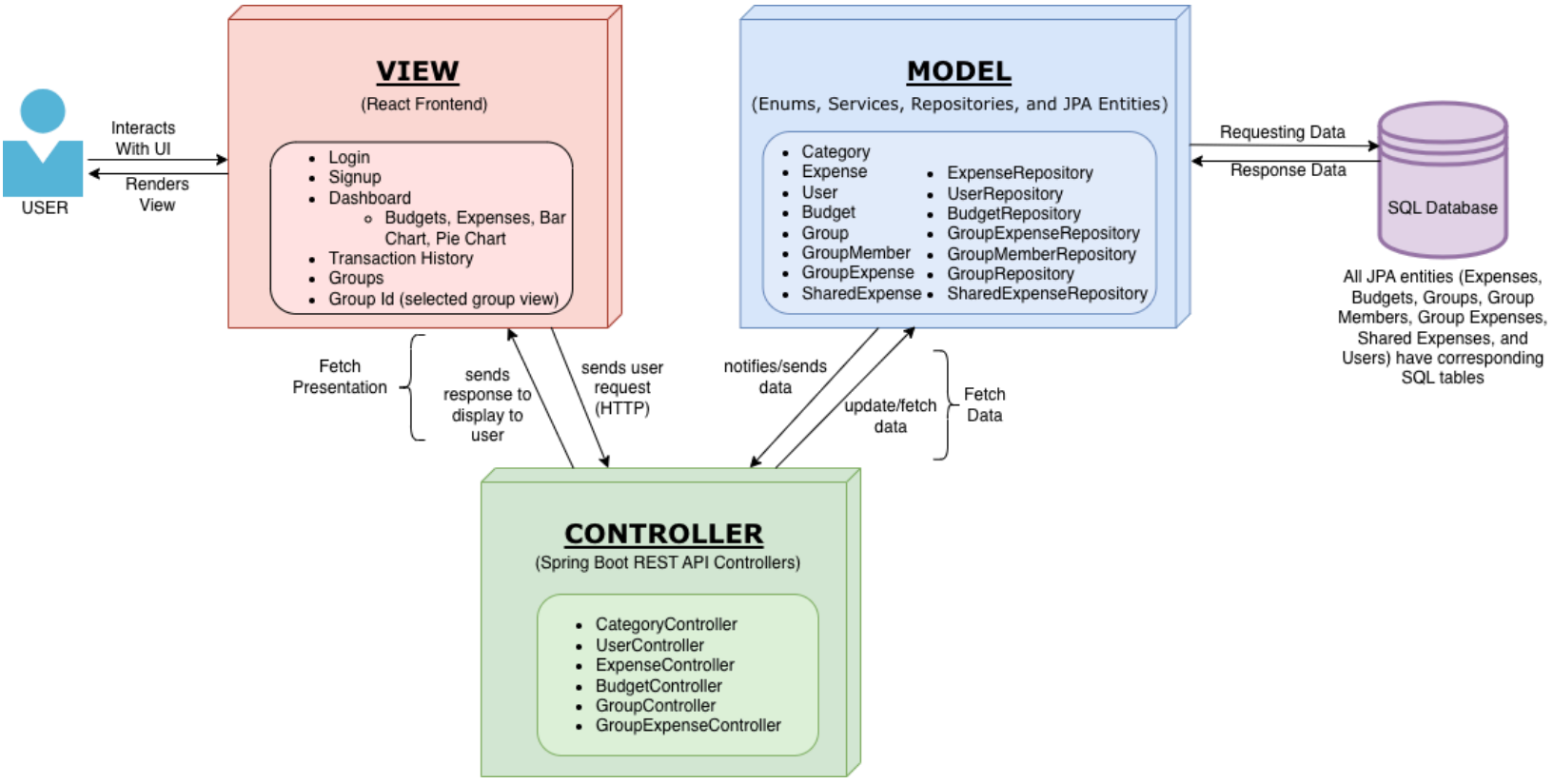
Controllers such as ExpenseController, BudgetController, UserController, etc. act as the middle layer between the views and the models, since they can handle incoming HTTP requests. They receive user requests from the views, process the data, interact with the models and repositories, and then are able to return the responses back to the views in JSON format. This

allows the views to dynamically update and display the latest information in regards to the backend entities.

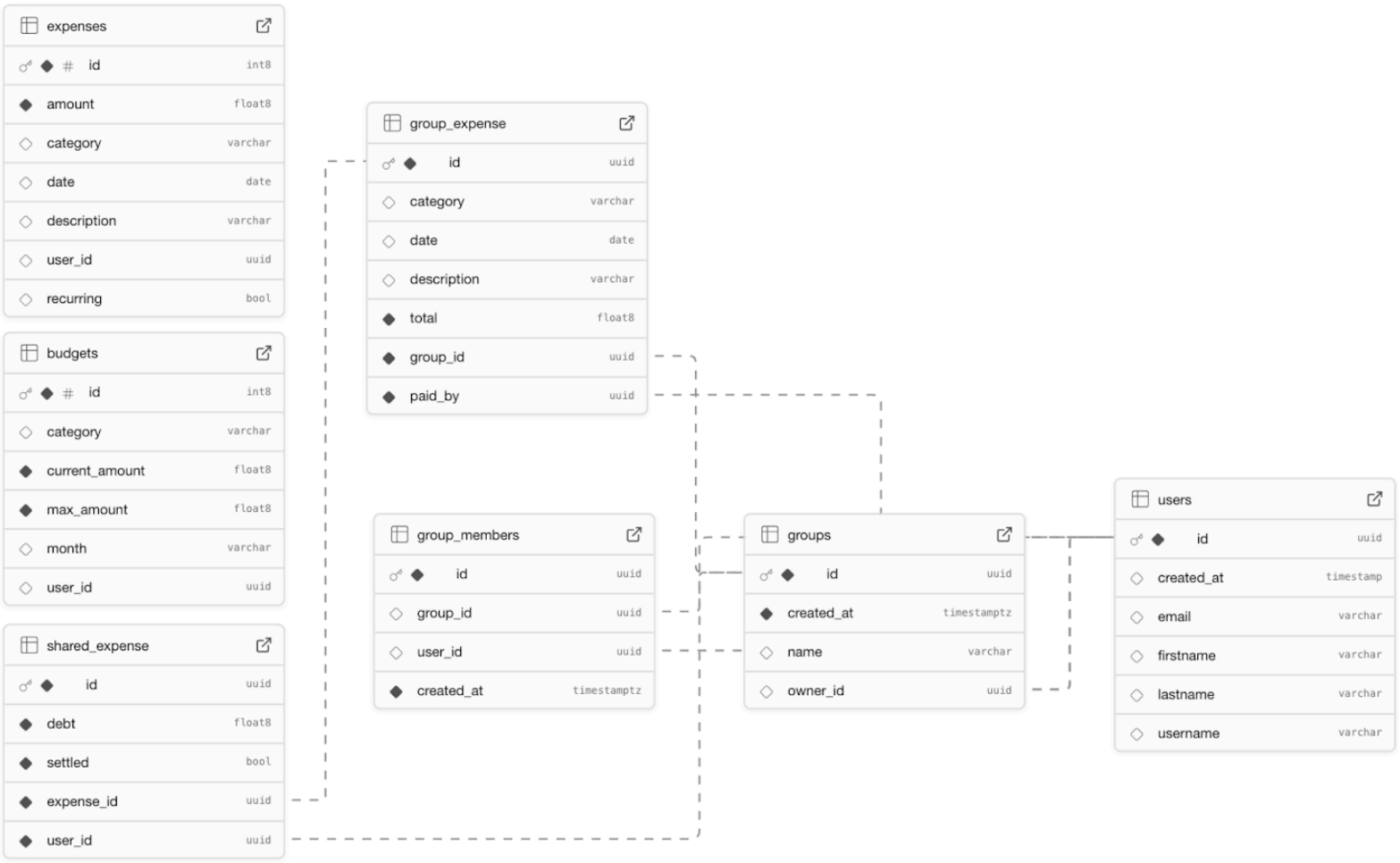
View Layer (Frontend)

The frontend is built using Next.js and consists of many interactive views that users can engage with directly. These views collect user input, display relevant data, and dynamically update the data, based on responses returned by the backend controllers. Through API (controller) requests, the frontend can access backend models and also render real-time changes in user, expenses, budgets, and group information.

Architecture Diagram:



Database Schemas:



System Decomposition: Model-View-Controller (MVC) with Error Handling

MVC Component	Classes	Responsibilities	Error Handling
View	*Next.js pages*: - Login - SignUp, - Dashboard, - Add ExpenseForm - Edit ExpenseForm - BudgetBar - Transactions - PieChartComponent - TotalAndCategory - Groups	- Display UI views - Get user inputs - Validate user input - Call backend controllers to fetch data and to save user responses.	Invalid inputs show error messages to the user. Any missing required fields, will prompt the user to enter some input. (Ex: A category and amount is required to add a new expense). For Login, if an incorrect password is provided, the user will be re-prompted until the correct password is entered.
Controller	*Controllers*: - UserController, - ExpenseController, - CategoryController - BudgetController - GroupController - GroupExpenseController	- Handle HTTP requests from the view, validates, and manipulates data. - Coordinate between the view and the model classes.	Invalid inputs or business rule violations result in HTTP 400 (Bad Request)

Model	*Entities:* - User - Expense - Group - GroupMember - GroupExpense - SharedExpense - Budget - Category *Repositories:* - UserRepository - ExpenseRepository - BudgetRepository - GroupRepository - GroupMemberRepository - GroupExpenseRepository - SharedExpenseRepository *Database*: - PostgreSQL constraints handle all queries logged - pushes to the controller for developers to handle.	- Manage data from User, Expense, Category etc. - Enforce database constraints - Handle all queries	Database constraint violations or connection failures are pushed to the controllers for developers to handle.

System Decomposition: Major Subsystems

1. Authentication
   - Handles user sign up and login.
   - Ensures that only logged in users have access to the other pages within the app.
   - Components**:** Supabase Auth, Login page, Sign-up page, User Controller.

2. User Management
   - Manages all existing users, user profile information, and user account retrieval.
   - Components**:** UserController, UserRepository, User.

3. Expense Management
   - Handles adding, editing, deleting, listing and searching through expenses.
   - Includes monthly expense totals, expenses by category, and corresponding charts.
   - Components**:** ExpenseController, ExpenseRepository, Expense, Dashboard, Transactions.

4. Budget Management
   - Manages optional monthly budgets that are set per category.
   - Tracks total spent vs. budget limit and renders progress bars.
   - Components**:** BudgetController, BudgetRepository, Budget, BudgetBar, EditBudgetModal, AddBudgetModal, Dashboard.

5. Group & Shared Expense Subsystem
   - Handles the creation of groups, group member invitations, shared expense tracking, and pay settlements.
   - Components**:** GroupController, GroupExpenseController, GroupRepository, GroupMemberRepository, SharedExpenseRepository.