

System Design Document

Project CashTag

EECS 3311 - Sprint 1

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

Table of Contexts

CRC Cards.....3

System Interaction.....6

Architecture Overview.....7

Architecture Diagram.....7

System Decomposition8

CRC cards

Class name: User	
Responsibilities	Collaborators
- Store user information (id, firstname, lastname, email, createdAt)	- UserRepository
- Allow retrieval of all users	- UserController
- 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 data	- User
- Use Spring Data JPA methods such as findAll()	- UserController

Class Name: UserController	
Responsibilities	Collaborators
- Retrieves user data from UserRepo	- User

Class Name: UserController	
- Handles the HTTP requests to interact with user data	- UserRepository
- Map routes through api/users	

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

Class Name: Expense	
Responsibilities	Collaborators
- Stores and manages the data related to an expense.	- ExpenseRepository
- 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.	Expense, ExpenseController
- Save, delete or edit expenses directly in the database.	Expense
- Query expenses by user, category, or date.	ExpenseController

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

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

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

expense form is submitted	
---------------------------	--

Class Name: Dashboard (from dashboard/page.js)	
Responsibilities	Collaborators
<ul style="list-style-type: none"> - The main dashboard UI page that imports and renders all of the components for expense tracking. 	AddExpenseModal, BarChartComponent, EditExpenseModal

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:

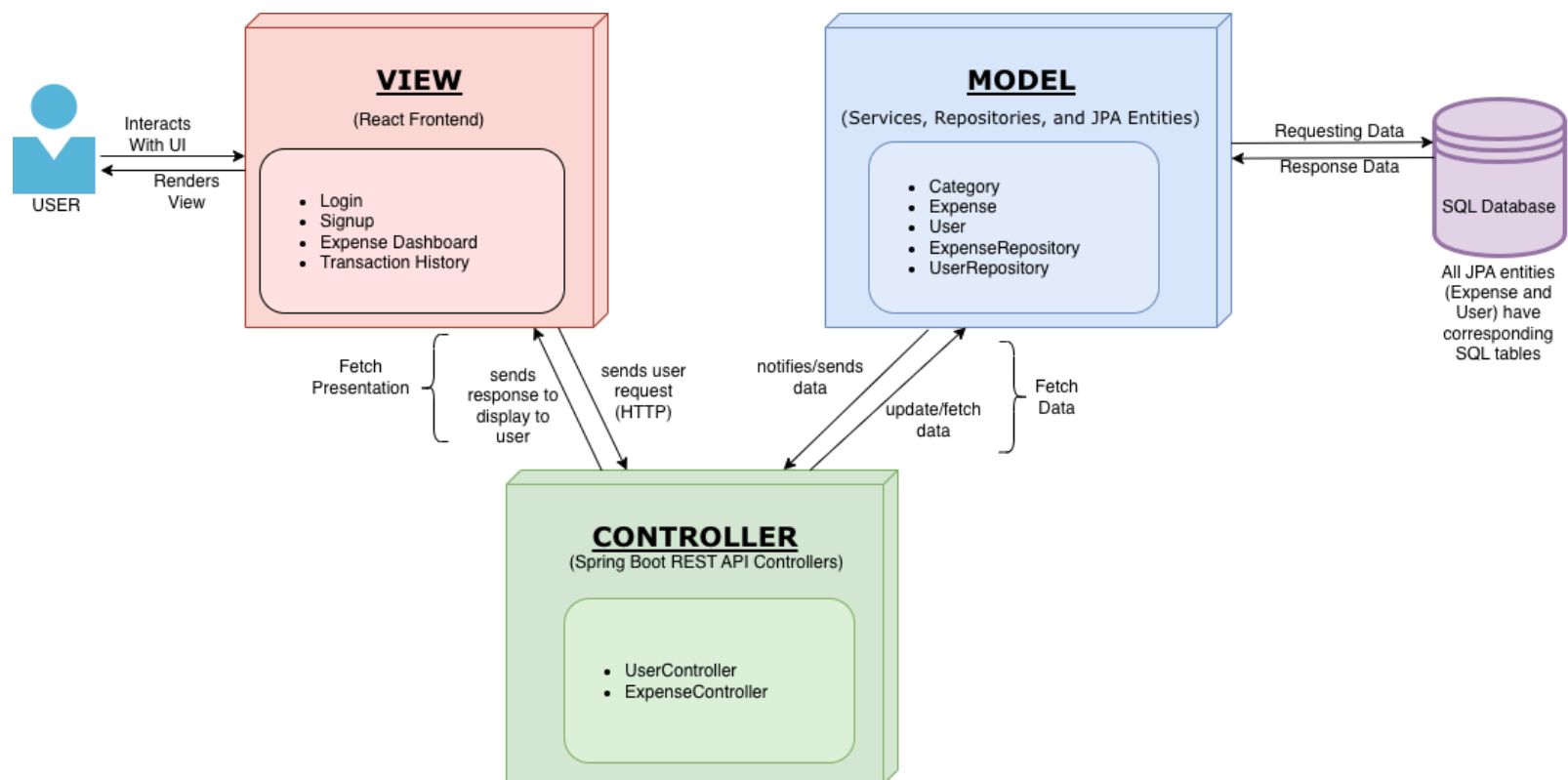
- **Operating System:** Any modern OS (Windows, macOS, or Linux) capable of running a Java Virtual Machine (JVM).
- **Database:** PostgreSQL server with proper configuration and connection credentials. For our project we have decided to go with a PostgreSQL database that is hosted through Supabase.
- **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. In the backend, we are using Springboot controllers to manage API requests, and we have models which represent core entities, business logic and communicate with the database. In the frontend, we have many views that the user can interact with.

The ExpenseController and UserController 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 expense and user information

Architecture Diagram:



System Decomposition:

Model-View-Controller (MVC) with Error Handling

MVC Component	Classes	Responsibilities	Error Handling
View	<i>Next.js pages:</i> <ul style="list-style-type: none"> - Login - SignUp, - Dashboard, - Add ExpenseForm - Edit ExpenseForm - ViewHistory 	<ul style="list-style-type: none"> - Display UI - get user inputs - validate input - Call controllers 	Invalid inputs show error messages to the user.
Controller	<i>Controllers:</i> <ul style="list-style-type: none"> - UserController, - ExpenseController, - CategoryController 	<ul style="list-style-type: none"> - Handle requests from the view, validate and manipulate data - Coordinate between view and model 	Invalid inputs or business rule violations result in HTTP 400 (Bad Request)
Model	<i>Repositories:</i> <ul style="list-style-type: none"> - UserRepository - ExpenseRepository <i>Database:</i> <ul style="list-style-type: none"> - PostgreSQL constraints handle all queries logged - pushes to the controller for developers to handle. 	<ul style="list-style-type: none"> - Manage data from User, Expense, Category etc. - Enforce database constraints - Handle all queries 	Database constraint violations or connection failures are pushed to the controller for developers to handle