# Deliverable 2

Mariana-Ionela Muntian

April 13, 2025

# Contents

# 1 Domain Model

## 1.1 Overview

The domain model defines the key business entities and the relationships between them. In this project, the main entities include:

- **Admin:** Represents system administrators with credentials for managing the application.
- **User:** End users who sign up and log in. Each user has attributes such as a username, password hash, and salary.
- **Budget:** Represents a financial budget linked to a unique card number. Each budget has an amount and is associated with users.
- **UserBudget:** A linking entity that associates Users with Budgets (modeling a many-to-one or many-to-many relationship as needed).
- **Category:** Represents product categories (e.g., Food, Travel) and groups similar Products.
- **Product:** Represents an expense or purchased item with attributes such as name, price, and date. Each Product is linked to a Category.
- **User-Product Association:** *(New)* Each **Product** is now associated with the **User** who purchased it. This link ensures that users have access only to the history of products they have purchased.
- **Data Transfer Objects (DTOs):** Such as `CategoryExpenseRequest` are used for transferring aggregate data (e.g., total expenses per category) between layers.

## 1.2 Conceptual Class Diagram

The conceptual class diagram below models the relationships among these entities.

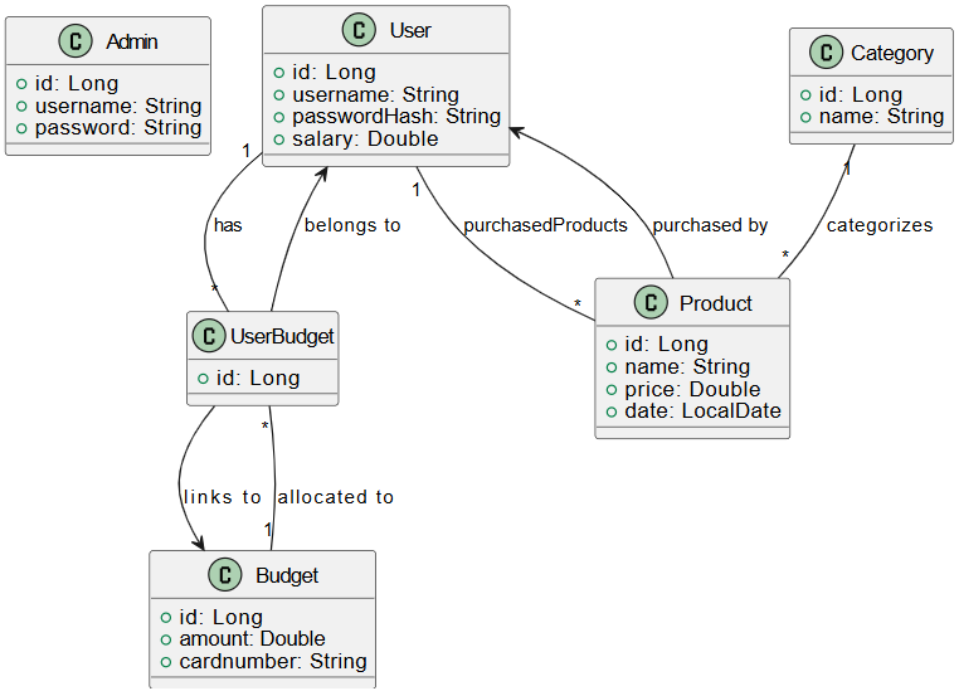


*Figure 1: Conceptual Class Diagram of the Domain Model*

**Notes:**

- The `UserBudget` class acts as a link between `User` and `Budget`.
- A `Category` aggregates multiple `Product` entities.
- `Product` entity includes a many-to-one association with `User`, and the `User` entity contains a collection of purchased products.
- DTOs facilitate data aggregation and transfer.

# 2 Architectural Design

## 2.1 Conceptual Architecture Overview

The system adopts a multi-tier (n-tier) architecture using the Model-View-Controller (MVC) pattern. The major layers are:

1. **Presentation Layer (Frontend):**
   - Implemented in Angular.
   - Provides user interfaces (login, dashboard, expense management, purchase history, charts, etc.).
   - Communicates with the backend via RESTful HTTP APIs.
2. **Application Layer (Backend):**
   - Built with Spring Boot (Java).
   - Consists of Controllers (REST endpoints), Services (business logic), and Repositories (data access using Spring Data JPA).
   - Uses DTOs for structured data exchange.
3. **Data Layer:**
   - Uses a relational database (e.g., MySQL or H2).
   - Persists the domain entities.

## 2.2 Architectural Style and Rationale

- **Style:** Multi-tier (n-tier) Architecture using the MVC pattern.
- **Motivation:**
  - *Separation of Concerns:* Clear distinctions between the presentation, business logic, and data access layers improve maintainability.
  - *Modularity:* Each layer can be developed, tested, and scaled independently.
  - *Flexibility:* RESTful communication between the Angular frontend and Spring Boot backend facilitates future extensibility—such as adding alternative client applications.

## 2.3 High-Level Architecture Diagram

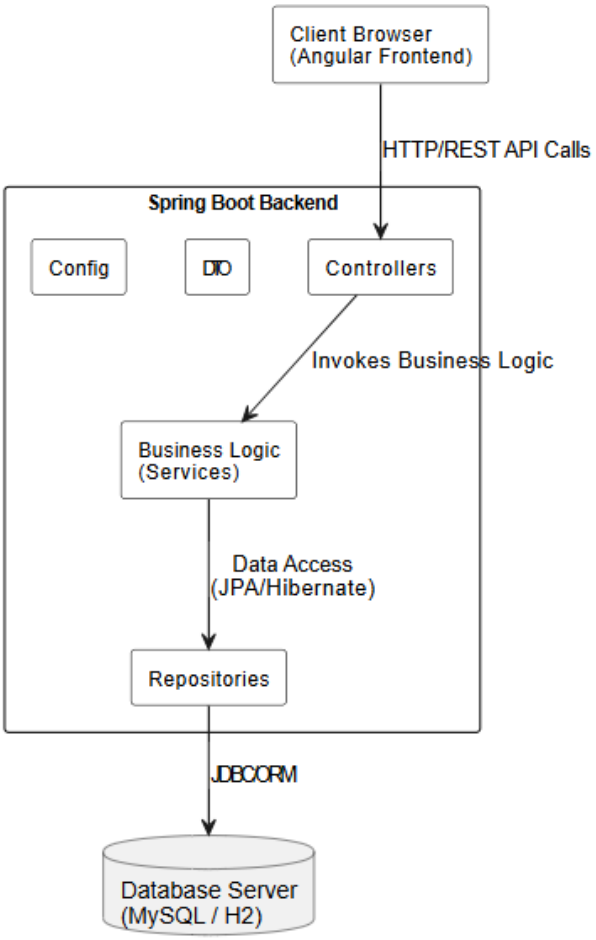


*Figure 2: High-Level Architectural Diagram*

# 3 Package Design

## 3.1 Backend Package Organization

The backend is organized into several packages that encapsulate related functionality. The primary packages include:

- **com.example.demo.Controllers:** Contains the REST controllers (e.g., `AdminController`, `BudgetController`, `ProductController`) that expose endpoints.
- **com.example.demo.BusinessLogic:** Contains service classes (e.g., `AdminService`, `BudgetService`, `ProductService`, `UserService`) to implement business rules.
- **com.example.demo.Model:** Contains the entity classes (e.g., `User`, `Budget`, `Product`, `Category`, `UserBudget`). *(Note: The `Product` class now includes an association with `User`.)*
- **com.example.demo.Repository:** Contains repository interfaces that extend `JpaRepository` for data persistence.
- **com.example.demo.DTO:** Contains Data Transfer Objects (e.g., `CategoryExpenseRequest`, `UserLoginRequest`).
- **com.example.demo.Config:** Contains configuration classes (e.g., security configurations).

## 3.2 Package Diagram

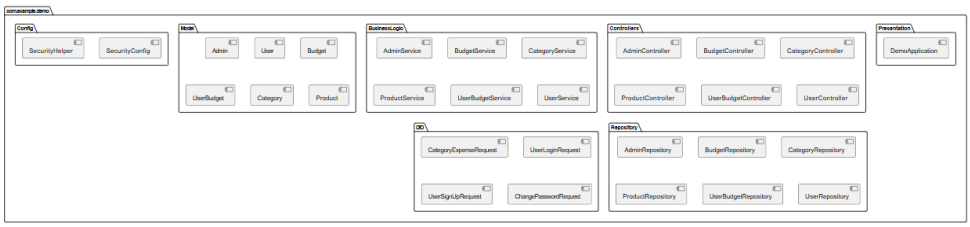


*Figure 3: Package Diagram*

# 4 Component and Deployment Diagrams

## 4.1 Component Diagram

The component diagram illustrates the major software components and their interactions:

- **Angular Frontend:**
  - Consists of UI Components (e.g., Login, Dashboard, Expense Management, Purchase History, Charts) and Services (HTTP calls to the backend).
- **Spring Boot Backend:**
  - Exposes RESTful APIs through Controllers.
  - Implements business logic in Services.
  - Interacts with the database using Repositories.
  - Provides a new endpoint to retrieve products purchased by a specific user.
- **Database Server:**
  - Persists data via an ORM (JPA/Hibernate).

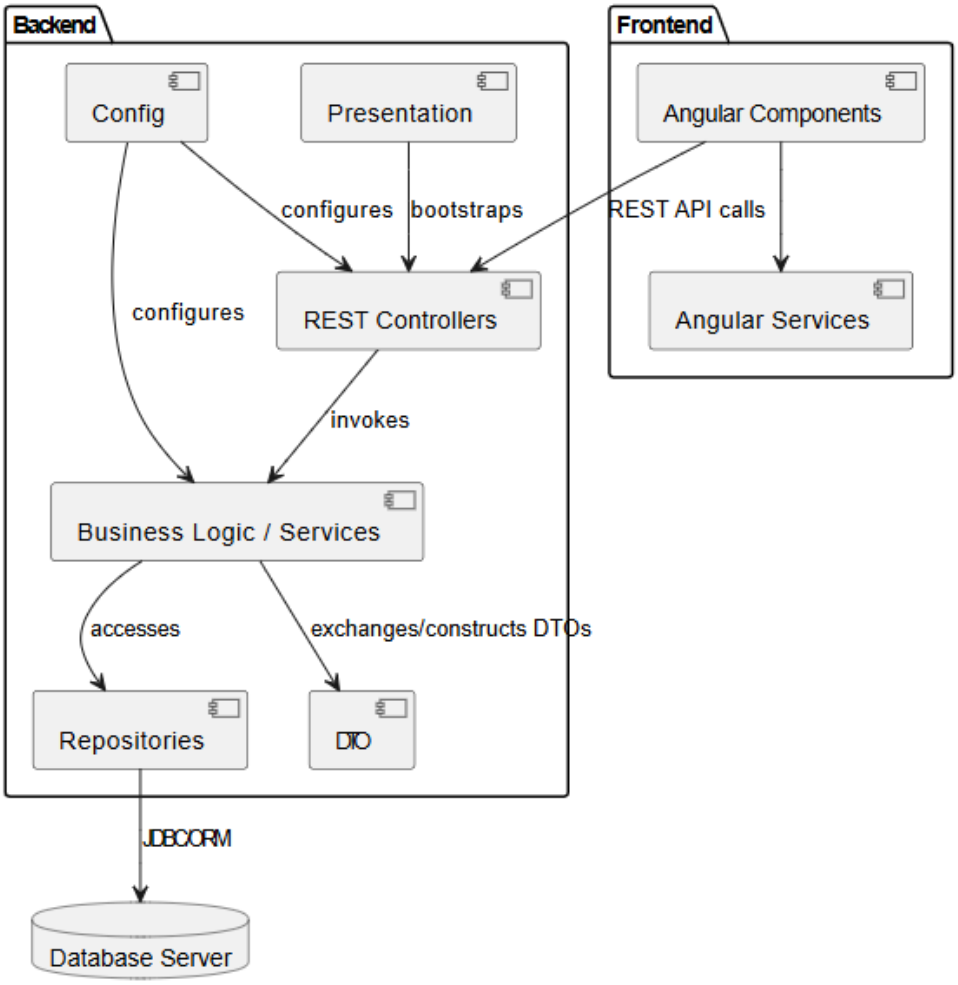


*Figure 4: Component Diagram*

## 4.2 Deployment Diagram

The deployment diagram represents the physical distribution of the components:

- **Client Browser:** Runs the Angular application.
- **Application Server:** Hosts the Spring Boot backend that provides RESTful APIs.
- **Database Server:** Stores domain data, accessed via JDBC/ORM. The `product` table now includes a `user_id` foreign key to link products to the purchasing user.

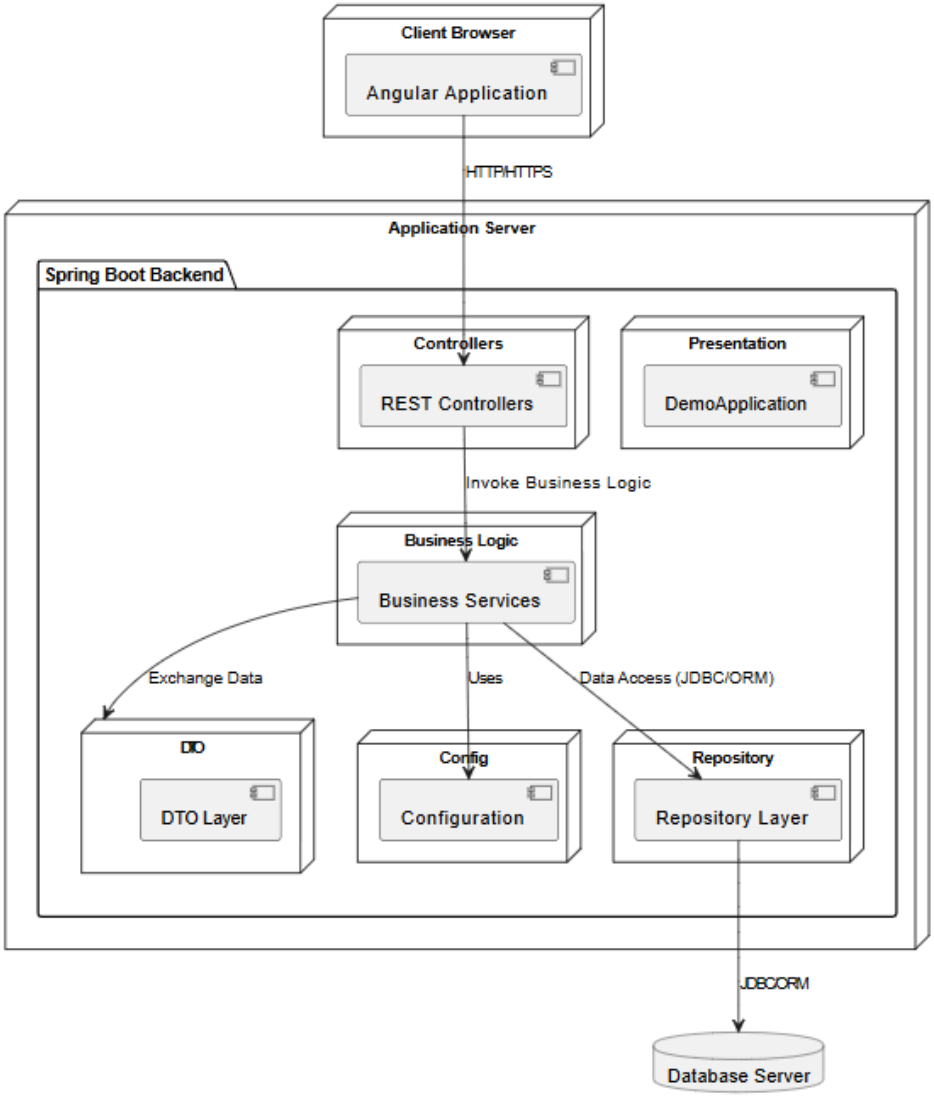


*Figure 5: Deployment Diagram*

# 5 Conclusion

This documentation deliverable provides a comprehensive overview of the system's design through:

- A detailed domain model (with an updated conceptual class diagram that includes the new User-Product association).
- An architectural design that leverages a multi-tier MVC pattern for clear separation of concerns.
- A package design that organizes backend functionalities into logical packages.
- Component and deployment diagrams showing the interaction and physical distribution of system components, including the new endpoint for a user's product purchase history.