### **Expense Reimbursement System: Detailed Project Overview**

The **Expense Reimbursement System** is a web application built using **Spring Boot** and **JPA** (Java Persistence API) for managing the submission, approval, and tracking of employee expenses within an organization. The system allows employees to submit expense reports, which can then be reviewed, approved, or rejected by managers.

### **1. Project Structure Overview**

The project is organized into multiple packages, each with a specific purpose, such as handling the business logic, database interactions, or web requests. The following packages are present in the project:

1. **model** - Contains Java classes that represent the data entities (tables in the database).
2. **repository** - Contains interfaces that interact with the database to perform CRUD operations.
3. **service** - Contains business logic related to managing roles, employees, expenses, and categories.
4. **controller** - Contains the REST API controllers that handle HTTP requests and return responses.

### **2. Detailed Explanation of Each Package and Class**

#### **2.1 model Package (Entities)**

This package contains Java classes that represent the database entities. These entities are mapped to the tables in the database, and JPA (Java Persistence API) handles the database interaction.

##### **Category.java**

* **Purpose**: Represents an **expense category** in the system, like "Medical Coverage", "Fuel Allowance", or "Education Allowances".
* **Attributes**:
  + id: The unique identifier for each category.
  + name: The name of the category.
  + status: A boolean indicating whether this category is **active** (1) or **inactive** (0).
* **Annotations**:
  + @Entity: Marks this class as an entity, meaning it will be mapped to a database table.
  + @Table(name = "categories"): Specifies the table name in the database.
  + @Id: Marks the field id as the primary key of the table.
  + @GeneratedValue(strategy = GenerationType.IDENTITY): Automatically generates unique values for id.

##### **Employee.java**

* **Purpose**: Represents an **employee** in the organization.
* **Attributes**:
  + id: Unique identifier for each employee.
  + name: Employee’s name.
  + email: Employee’s email address.
  + role: The employee’s **role** in the organization (linked to the Role entity).
* **Annotations**:
  + @ManyToOne: Specifies that each employee has one role, and many employees can have the same role (one-to-many relationship).
  + @JoinColumn(name = "role\_id"): Specifies the foreign key column (role\_id) linking Employee to Role.

##### **Expense.java**

* **Purpose**: Represents an **expense submission** made by an employee. This includes details like the amount, description, and the status of the expense (Pending, Approved, Rejected).
* **Attributes**:
  + id: Unique identifier for the expense.
  + employee: The employee who submitted the expense (linked to the Employee entity).
  + amount: The amount of money requested for reimbursement.
  + description: A brief description of the expense.
  + category: The category of the expense (linked to the Category entity).
  + status: The status of the expense (linked to the ExpenseStatus entity).
  + submitDate: The date and time the expense was submitted.
  + approvalDate: The date and time the expense was approved or rejected (if applicable).
* **Annotations**:
  + @ManyToOne: Specifies the relationships between the expense, employee, category, and expense status.
  + @JoinColumn(name = "employee\_id"): Defines the foreign key relationship between Expense and Employee.
  + @JoinColumn(name = "category\_id"): Defines the foreign key relationship between Expense and Category.
  + @JoinColumn(name = "status\_id"): Defines the foreign key relationship between Expense and ExpenseStatus.

##### **ExpenseStatus.java**

* **Purpose**: Represents the **status** of an expense (e.g., "Pending", "Approved", "Rejected").
* **Attributes**:
  + id: Unique identifier for the status.
  + name: The name of the status (e.g., "Pending", "Approved").
  + status: A boolean value indicating whether this status is **active**.
* **Annotations**:
  + @Entity: Marks the class as a JPA entity.
  + @Id: Marks the id field as the primary key.
  + @GeneratedValue: Automatically generates a unique value for id.

##### **Role.java**

* **Purpose**: Represents the **role** of an employee in the company (e.g., "Intern", "Software Engineer", "Manager").
* **Attributes**:
  + id: Unique identifier for the role.
  + name: The name of the role.
  + status: A boolean indicating whether this role is **active** or not.
* **Annotations**:
  + @Entity: Marks the class as a JPA entity.
  + @Id: Marks the id field as the primary key.
  + @GeneratedValue: Automatically generates a unique value for id.

#### **2.2 repository Package (Data Access Layer)**

This package contains JPA repositories that provide CRUD operations on the entities. These repositories extend JpaRepository, a Spring Data interface that automatically provides implementation for common database operations.

##### **CategoryRepository.java**

* **Purpose**: Provides methods to interact with the Category table in the database.
* **Methods**: Inherits basic methods like findAll(), save(), findById(), etc.

##### **EmployeeRepository.java**

* **Purpose**: Provides methods to interact with the Employee table in the database.
* **Methods**: Inherits basic methods for interacting with employee data.

##### **ExpenseRepository.java**

* **Purpose**: Provides methods to interact with the Expense table in the database.
* **Custom Methods**:
  + findByStatus(ExpenseStatus status): Finds all expenses by their status (e.g., Pending, Approved, Rejected).
  + findByEmployeeAndSubmitDateBetween(Employee employee, LocalDateTime startDate, LocalDateTime endDate): Finds all expenses for a particular employee within a date range.

##### **ExpenseStatusRepository.java**

* **Purpose**: Provides methods to interact with the ExpenseStatus table in the database.
* **Methods**: Basic CRUD operations for expense statuses.

##### **RoleRepository.java**

* **Purpose**: Provides methods to interact with the Role table in the database.
* **Methods**: Basic CRUD operations for employee roles.

#### **2.3 service Package (Business Logic)**

This package contains the **business logic** for managing employees, roles, expenses, and categories. It processes data received from the repositories and prepares it for the controller to return as a response.

##### **ExpenseService.java**

* **Purpose**: Contains business logic for managing expenses, employees, roles, and categories.
* **Methods**:
  + getAllRoles(), getAllEmployees(), getAllCategories(), getAllExpenseStatuses(): Fetches all roles, employees, categories, and expense statuses from the database.
  + getPendingExpenses(): Fetches all expenses that are currently **pending**.
  + addExpense(Expense expense): Validates and adds a new expense to the system, setting its status to "Pending".
    - Validates the employee and category.
    - Sets the expense's status and submission date.
  + updateExpenseStatus(int expenseId, int statusId): Updates the status of an existing expense (either "Approved" or "Rejected").
    - Validates the expense's existence and status ID.
    - Sets the **approval date** if the status is "Approved" or "Rejected".
  + getExpensesByEmployeeAndDateRange(int employeeId, LocalDate startDate, LocalDate endDate): Fetches expenses for a specific employee within a date range.
  + getExpensesByStatus(String statusName): Fetches expenses based on their status (e.g., "Approved", "Rejected").

#### **2.4 controller Package (API Endpoints)**

This package contains the **controller classes** that handle incoming HTTP requests, invoke the appropriate service methods, and return HTTP responses. These controllers expose RESTful APIs for the client (e.g., frontend or external system) to interact with the backend.

##### **ExpenseController.java**

* **Purpose**: Exposes HTTP endpoints for managing roles, employees, categories, and expenses.
* **Endpoints**:
  + GET /api/roles: Fetches all roles.
  + GET /api/employees: Fetches all employees.
  + GET /api/categories: Fetches all categories.
  + GET /api/expense-statuses: Fetches all expense statuses.
  + GET /api/expenses: Fetches all expenses.
  + GET /api/expenses/pending: Fetches only **pending** expenses.
  + POST /api/expenses: Adds a new expense by calling the addExpense() method in ExpenseService.
  + PATCH /api/expenses/{expenseId}/status: Updates the status of an expense (e.g., "Approved" or "Rejected").
  + GET /api/expenses/employee/{employeeId}: Retrieves expenses for an employee within a date range.
  + GET /api/expenses/history: Retrieves expenses based on status (e.g., "Approved", "Rejected").

### **3. Database Schema**

The database schema is used to store the information about employees, expenses, roles, categories, and status. Here’s a summary of each table:

* **role**: Contains different roles that employees can have (e.g., "Intern", "Software Engineer").
* **employee**: Contains employee details (name, email) and references a role.
* **categories**: Contains expense categories (e.g., "Fuel", "Medical").
* **expense\_status**: Contains the different statuses an expense can have (e.g., "Pending", "Approved").
* **expense**: Stores expense submissions, including the amount, description, employee, category, and status.

### **4. System Flow**

1. **Employee Role Management**:
   1. Employees are assigned roles (e.g., Intern, Software Engineer, Manager).
2. **Expense Submission**:
   1. Employees submit expenses by providing details such as the amount, category, and description.
   2. The submitted expense has an initial status of "Pending".
3. **Expense Review and Approval**:
   1. Managers can approve or reject expenses by updating the expense's status.
   2. Approved expenses will have an approval date set in the database.
4. **Expense Querying**:
   1. Employees and managers can view expenses filtered by status, date range, or employee.

### **5. Conclusion**

The **Expense Reimbursement System** is a well-structured application built using **Spring Boot** and **JPA**, following best practices in separating concerns into different layers (model, repository, service, controller). This system allows employees to submit expenses, and managers to review and approve/reject them based on the submitted details.