Expense Manager Application

Problem Statement:

Problem Submissions Solutions Doubts

Expense Manager Application

Moderate • Score 240/240 Spring Hibernate(Intermediate)

Problem statement

Send feedback

Project Goal

The organization requires an Expense Manager Application to track employee income and expenses. Users can input their income and expenses, categorizing them into various types (e.g., Bonus, Salary, Consultancy for income, and Food, Education, Bills, Travel, and Miscellaneous for expenses). The application will be developed using Spring Boot, Hibernate, and MySQL, enabling straightforward expense tracking through CRUD operations.

▼ Features of the Application:

The application should allow to:

- Create users.
- Add the Income and Expenses of a User.
- View all users' income and expenses.
- Filter for Day, Month, and Year of IncomeDate.
- Filter for IncomeType and ExpenseType.

▶ Flowchart:

▼ Steps:

Use the following guidelines and hints to build the project.

1. We will be utilizing the Spring version **2.7.7** for this mini-project.
2. Create the following classes in the **entities** package with the required getters, setters and annotations as mentioned below:
   a. Create a **User** class that has the following attributes:
      - Integer id
      - String username
      - String nickname
      - String email
      - String address
      - List expenses (One To Many mapping)
      - List incomes (Many To Many mapping)

   b. Create an **Income** class that has the following attributes:
      - int id
      - double amount
      - LocalDate date
      - String description
      - Expense expense (One To One mapping)
      - List users (Many To Many mapping)
      - List incomeTypes (One To Many mapping)

   c. Create an **IncomeType** class that has the following attributes:
      - int id
      - String name
      - Income income (Many To One mapping)

   d. Create an **Expense** class that has the following attributes:
      - int id
      - double amount
      - LocalDate date
      - String description
      - User user (Many To One mapping)
      - Income income (One To One mapping)

Problem Submissions Solutions Doubts

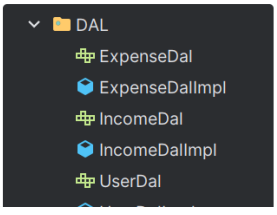
      - List expenseTypes (One To Many mapping)

   e. Create an **ExpenseType** class that has the following attributes:
      - int id
      - String name
      - Expense expense (Many To One mapping)

   The data should be saved in the database.

3. Possible Relationships between Entities:
   - User-Expense Class: One-to-many Relationship
   - User-Income Class: Many-to-many with the income
   - Income-Expense Class: One-to-one Relationship
   - Income-IncomeType Class: Many-to-one Relationship
   - Expense-ExpenseType Class: Many-to-one Relationship
4. Define relationships between entities using annotations.
5. Create the following DAL interfaces to handle database operations as shown below:

▼ Reference image



6. Create a service class for Expense, Income and User entities to handle business logic. Also, you are supposed to use **@Transactional** annotation for each service.

7. Create a controller class for the following entity in the controllers package to handle HTTP requests:

- Expense
- Income
- User

8. Test the application using tools such as Postman to ensure data is saved and retrieved correctly from the database.

End Points To Be Created:

1. User Endpoints:

- **GET "/users/allUsers"**: It retrieves the list of all users from the database.
- **GET "/users/{id}" (@PathVariable Integer id)**: It fetches a user by the given Id.
- **POST "/users/save" (@RequestBody User user)**: It saves a new user into the database.
- **POST "/users/checkUserExist" (@RequestBody User user)**: It checks if the given user exists in the database or not. If the given user is found it returns true otherwise false.
- **POST "/users/find" (@RequestBody User newUser)**: It fetches the given user if found else returns null.
- **GET "/users/filteredUserListByCalendar" (@RequestParam(value = "day", required = false) String day,@RequestParam(value = "month", required = false) String month,@RequestParam(value = "year", required = false) String year)**: It fetches the list of all user by the given filters i.e. by Day, Month, and Year of IncomeDate.
- **GET "/users/filteredUserListByType" (@RequestParam(value = "incomeType", required = false) String incomeType,@RequestParam(value = "expenseType", required = false) String expenseType)**: It fetches the list of all user by the given filter i.e. by IncomeType and ExpenseType.

2. Income Endpoints:

- **GET "/incomes/{incomeid}"**: It retrieves the income for the given id.
- **POST "/incomes/save/{userId}" (@PathVariable Integer userId, @RequestBody Income income)**: It registers a new income f the given userId.

3. Expense Endpoints:

- **POST "/expenses/save/{incomeId}" (@PathVariable Integer incomeId, @RequestBody Expense expense)**: It creates a new expense for a given incomeId.

Testing on Postman:

After successfully creating the application, you have to test your application.

1. Your application should test the following:

   - The user's registration should be successfully stored in the database.
   - On login, the user should be stored in the session.
   - Income should be successfully added, stored in the session, and linked correctly with the current user.
   - IncomeType should be successfully added and should be linked correctly with the Income.
   - Expenses should be successfully added and linked correctly with the current session income.
   - ExpenseType should be successfully added and should be linked correctly with Expense.

2. Your application should throw an error if:

   - The user is trying to register a new user with a username already in the database.
   - The user is trying to log in with a username that does not exist in the application.
   - The user is trying to add a date in the incorrect format.

3. Test all the exposed APIs of Backend on Postman, as follows:

Output: