Finance Management System

Coding Task 2

Service Provider Interface/Abstract class

Keep the interfaces and implementation classes in package dao

Define IFinaceRepository interface/abstract class with methods for adding/removing products to/from the cart and placing orders. The following methods will interact with database.

1. createUser()

parameter: User user

return type: boolean

2. createExpense()

parameter: Expense expense

return type: boolean

3. deleteUser()

parameter: userId

return type: boolean

4. deleteExpense(expenseId)

parameter: expenseId

return type: boolean

5. getAllExpenses(userId): list all expnses a user.

parameter: userId

return type: list of expenes

6. updateexpense(userId, Expense): should update order table and orderItems table.

parameter: userId, expense object

return type: boolean

In this task, we design the service layer of the Finance Management System by creating an interface IFinanceRepository in the dao package. This interface abstracts the core business operations related to users, expenses, and suggestions. It ensures a clean separation between data access and business logic, promoting modularity and flexibility.

Task 2.1 createUser()

Task 2.2 createExpense()

Task 2.3 deleteUser()

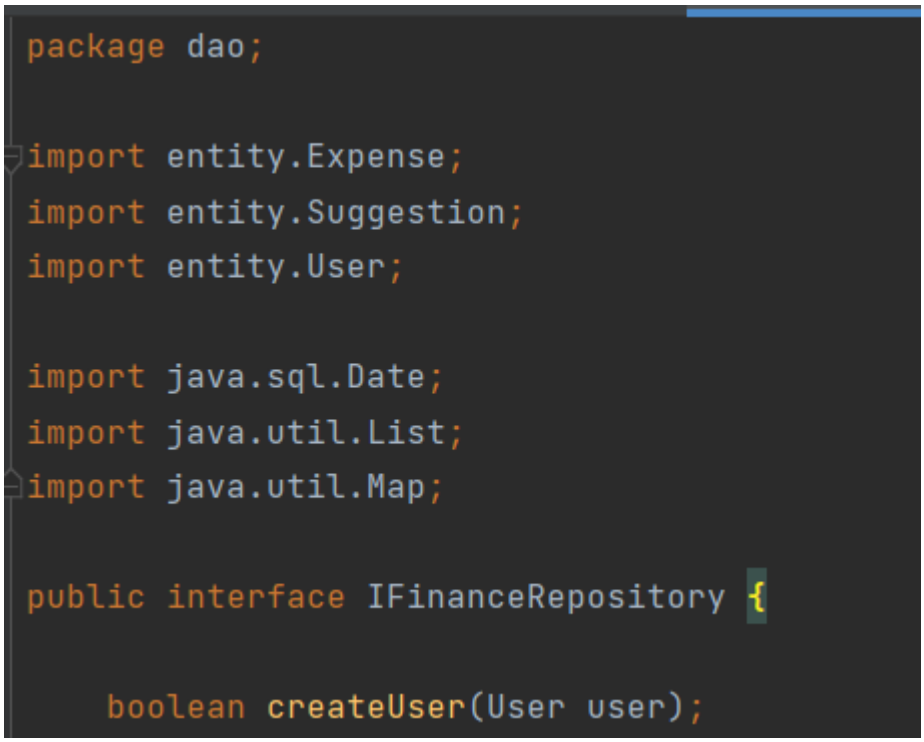
Task 2.4 deleteExpense(expenseId)

Task 2.5 getAllExpenses(userId)

Task 2.6 updateexpense(userId, Expense)

Each method in this interface represents a specific action that interacts with the underlying database, enabling easy implementation, testing, and maintenance.

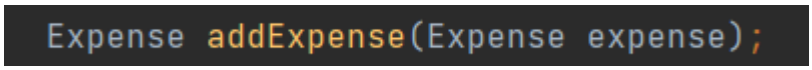
Task 2.1 createUser()



```
package dao;

import entity.Expense;
import entity.Suggestion;
import entity.User;

import java.sql.Date;
import java.util.List;
import java.util.Map;

public interface IFinanceRepository {

    boolean createUser(User user);
```

This method adds a new user to the system. It accepts a User object containing user details and returns a boolean indicating success or failure. This is essential for user registration.

Task 2.2 createExpense()



```
Expense addExpense(Expense expense);
```

This method inserts a new expense into the database. It accepts an Expense object, storing relevant information such as category, amount, and description for the logged-in user.

Task 2.3 deleteUser()

```
boolean deleteUserAndExpenses(int userId);
```

This method deletes a user from the system based on their unique user ID. It ensures the user's data, including their expenses, are removed appropriately (based on cascading or manual deletion).

Task 2.4 deleteExpense(expenseId)

```
boolean deleteExpense(int expenseId, int userId);
```

This method deletes a specific expense from the database using the provided expenseId. It helps users manage their finances by allowing them to remove old or incorrect records.

Task 2.5 getAllExpenses(userId)

```
List<Expense> getAllExpensesByUserId(int userId);
```

This method retrieves all expenses recorded by a particular user. It returns a list of Expense objects, allowing users to view and manage their full transaction history.

Task 2.6 updateexpense(userId, Expense)

```
boolean updateExpense(Expense expense);
```

This method updates the details of an existing expense for a given user. It accepts a userId and the modified Expense object, then applies the changes in the database. It's used when users want to edit recorded expenses.

Additional Implemented Methods

These additional implemented methods were included to enhance the system's functionality by providing advanced features like filtering, reporting, authentication, and user feedback, thereby improving user experience and data analysis capabilities.

```
User loginUser(String email, String password);
List<Expense> getExpensesByDateRange(int userId, Date fromDate, Date toDate);
List<Expense> getExpensesByCategory(int userId, int categoryId);
Expense getExpenseById(int expenseId, int userId);
Map<String, Double> getCategoryWiseReport(int userId, Date start, Date end);
Map<String, Double> getMonthlySummary(int userId, Date start, Date end);
boolean addSuggestion(Suggestion suggestion);
```

1. loginUser(String email, String password)

This method handles user authentication. It checks the database for matching credentials and returns a User object if login is successful, enabling session tracking and personalized experience.

2. getExpensesByDateRange(int userId, Date fromDate, Date toDate)

This method fetches a list of expenses for a user within a specific date range. It is particularly useful for generating custom reports or filtering recent spending behavior.

3. getExpensesByCategory(int userId, int categoryId)

This method retrieves expenses based on a selected category for a particular user. It's useful for analyzing spending habits and organizing transactions.

4. getExpenseById(int expenseId, int userId)

This method returns detailed information about a specific expense based on its ID and associated user ID. It supports detailed views and editing capabilities.

5. getCategoryWiseReport(int userId, Date start, Date end)

This method generates a report mapping each category to the total amount spent during the given date range. It provides insight into category-wise spending patterns.

6. getMonthlySummary(int userId, Date start, Date end)

This method returns a monthly breakdown of total expenses between the provided date range. It is essential for tracking trends and budgeting over time.

7. boolean addSuggestion(Suggestion suggestion)

This method allows users to submit suggestions or feedback. It accepts a Suggestion object and stores it in the system, promoting user engagement and system improvement.

Conclusion

The IFinanceRepository interface serves as a blueprint for interacting with the Finance Management System's core components. It encapsulates various functionalities like user registration, authentication, expense handling, reporting, and feedback submission. This abstraction not only ensures a consistent contract for implementing classes but also supports clean architecture principles such as separation of concerns and ease of maintenance. By covering all essential CRUD operations and adding meaningful analytics and feedback features, this interface significantly enhances the usability and robustness of the system.
