**CASE STUDY IMPLEMENTATION**

Finance Management System

BY,

POOJA SRI B

Mail ID: [poojasribk@gmail.com](mailto:poojasribk@gmail.com)

**DIRECTORY STRUCTURE:**

**A screenshot of a computer

Description automatically generated**

**Package: entity**

This package contains the entity classes used to represent the core data objects in the finance management system. These classes include:

* **Expense**: Represents an individual expense, including details such as the expense ID, user ID, amount, category ID, date, and description.
* **ExpenseCategory**: Represents a category for an expense, storing the category ID and the category name.
* **User**: Represents a user in the system, including user ID, username, password, and email.

Each class contains:

* A **default constructor** and a **parameterized constructor** to initialize instances with or without values.
* **Getters and setters** to access and modify the properties of the objects.

**Package: exception**

This package contains custom exception classes used to handle specific error conditions related to database and business logic operations:

* **DatabaseConnectionException**: Thrown when there is an issue with establishing a database connection, such as missing or incorrect database properties.
* **ExpenseNotFoundException**: Thrown when an attempt to retrieve an expense fails because the specified expense does not exist in the database.
* **InvalidEmailFormatException**: Thrown when an invalid email format is provided for user registration or update.
* **UserNotFoundException**: Thrown when an operation related to a user fails because the specified user cannot be found in the database.

**Package: util**

This package contains utility classes that provide common functionality for database operations:

* **DBConnUtil**: Provides methods to establish and close a connection to the database.
  + The getConnection method loads database properties, checks their validity, and establishes a connection to the database using JDBC.
  + The closeConnection method safely closes the provided database connection.
* **DBPropertyUtil**: Responsible for loading database properties from a configuration file (db.properties).
  + The loadProperties method reads the property file and returns a Properties object containing the database connection details.
  + The getConnectionString method constructs and returns the connection string by retrieving the URL, username, and password from the properties.

**Package: dao in test folder**

This package contains the Data Access Object (DAO) classes responsible for interacting with the database to perform CRUD operations.

* **FinanceRepositoryImplTest**: A JUnit test class that tests the functionality of the FinanceRepositoryImpl class, which is responsible for managing user and expense data in the database.
  + **BeforeClass**: Sets up the test environment before the test class runs. It initializes the financeRepository and establishes a database connection. The cleanDatabase method is called to remove any existing data, and the insertInitialData method adds sample data into the Users and ExpenseCategories tables.
  + **AfterClass**: Closes the database connection after the test class completes.
  + **Before**: Resets the database by cleaning and inserting initial data before each test method is executed.
  + **Test Methods**:
    - **testCreateUser**: Tests the createUser method by creating a new user and verifying that the user was added to the database.
    - **testCreateExpense**: Tests the createExpense method by creating a new expense and verifying that it was successfully inserted.
    - **testDeleteUser**: Tests the deleteUser method by deleting a user and verifying that the deletion was successful.
    - **testDeleteExpense**: Tests the deleteExpense method by deleting an expense after it has been created and verifying that the deletion was successful.
    - **testUpdateExpense**: Tests the updateExpense method by updating an existing expense and verifying that the update was successful.
    - **testGetAllExpenses**: Tests the getAllExpenses method by retrieving all expenses for a user and verifying that the correct number of expenses is returned.

Each test method follows a pattern of setting up necessary data, performing operations, and asserting that the expected outcomes occur, ensuring that the finance repository methods work as intended.

These classes and tests work together to provide a functional and testable system for managing users, expenses, and categories in the finance management application. The DAO classes interact with the database, while the exceptions handle potential errors, and utility classes simplify common tasks such as establishing a database connection.