Q1) Explain the architecture of your ---------- project and the role of Servlets/ JSP in it. (5 Marks)

In the **MVC (Model-View-Controller)** architecture for an **Expense Tracker** using **Servlets**, the application is divided into three distinct layers to ensure separation of concerns:

1. Model (Database - MySQL)

The **Model** represents the business logic and data layer. In this case:

- The **Model** interacts with the database (MySQL) to store, retrieve, and manipulate expense-related data.
- Tables in MySQL might include:
  - **expenses** (columns: id, date, category, amount, description)
  - **users** (if needed for user authentication)

The Model's role is to:

- Save new expense data (INSERT INTO expenses).
- Retrieve a list of expenses (SELECT * FROM expenses).

Example: A DAO (Data Access Object) class like ExpenseDAO manages database operations:

<u>ExpenseDAO.java</u>

```
package dao;

import model.Expense;

import utils.DBConnection;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

public class ExpenseDAO {

    public void addExpense(Expense expense) throws Exception {

        String sql = "INSERT INTO expenses (description, category, amount, date)
VALUES (?, ?, ?, ?)";
```

```
        try (Connection conn = DBConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, expense.getDescription());
            stmt.setString(2, expense.getCategory());
            stmt.setDouble(3, expense.getAmount());
            stmt.setDate(4, new java.sql.Date(expense.getDate().getTime()));
            stmt.executeUpdate();
        }
    }
    public List<Expense> getAllExpenses() throws Exception {
        List<Expense> expenses = new ArrayList<>();
        String sql = "SELECT * FROM expenses ORDER BY date DESC";
        try (Connection conn = DBConnection.getConnection();
             Statement stmt = conn.createStatement();
            ResultSet   rs   =   stmt.executeQuery(sql))
            { while (rs.next()) {
                Expense expense = new Expense();
                expense.setId(rs.getInt("id"));
                expense.setDescription(rs.getString("description"));
                expense.setCategory(rs.getString("category"));
                expense.setAmount(rs.getDouble("amount"));
                expense.setDate(rs.getDate("date"));
                expenses.add(expense);
            }
        }
        System.out.println("no results found");
        return expenses;
    }
}
```

2. View (JSP - Visible to User)

The **View** is responsible for presenting the user interface to the user. In this case:

- JSP files are used to display the interface for adding and viewing expenses.
- **AddExpense.jsp**:
  - A form for users to enter expense details (date, category, amount, description).
- **ViewExpense.jsp**:
  - A table that shows all expenses fetched from the database.

The role of the view:

- Display the data provided by the controller.
- Capture user input via forms.

3. Controller (Servlets)

The **Controller** handles user requests and orchestrates the flow between the Model and the View. In this case, there are two servlets:

1. **AddExpense Servlet**:
   - Handles the form submission from `AddExpense.jsp`.
   - Extracts the user input, calls the model (`ExpenseDAO`) to save the data in MySQL, and redirects to a success page or back to the form.
2. **ViewExpense Servlet**:
   - Fetches all expenses from the model (`ExpenseDAO`) and forwards the data to `ViewExpense.jsp` for display.

Role of Servlets:

- **Request Handling**: Receive HTTP requests from JSP pages.
- **Business Logic Coordination**: Call DAO methods to interact with the database.
- **Response Forwarding**: Forward the response to the appropriate JSP for display.

Flow of the Expense Tracker

1. **AddExpense Flow**:
   - User accesses `AddExpense.jsp` to add an expense.
   - The form is submitted to `AddExpenseServlet`.
   - The servlet:
     - Extracts form data (`date`, `category`, `amount`, `description`).
     - Calls the model (`ExpenseDAO.addExpense()`).
     - Redirects to a confirmation page or back to the form with a success message.
2. **ViewExpense Flow**:
   - User accesses `ViewExpenseServlet`.
   - The servlet:
     - Calls the model (`ExpenseDAO.getExpenses()`) to retrieve expense data.
     - Attaches the data to the request as an attribute.
     - Forwards the request to `ViewExpense.jsp`.
   - `ViewExpense.jsp` iterates over the list of expenses and displays them in a table.

Advantages of MVC in this Project

1. **Separation of Concerns**: Logic (Model), presentation (View), and control (Controller) are distinct.
2. **Scalability**: Easy to add features like user authentication or data filtering.
3. **Maintainability**: Changes in one layer (e.g., JSP design) don't impact others.

This setup ensures a clean and efficient design for the **Expense Tracker** application.

Q2) Describe the core modules and their functions in your project. (5 Marks)

Core Modules and Their Functions in the Expense Tracker Project

1. **User Interface Module (View - JSP)**
   - **Function**: Provides the interface for users to interact with the application.
     - **AddExpense.jsp**: Displays a form to input expense details such as date, category, amount, and description.
     - **ViewExpense.jsp**: Shows the list of expenses in a user-friendly tabular format.
   - **Purpose**: Captures user input and presents processed data visually.

2. **Controller Module (Servlets)**
   - **Function**: Manages the flow of data between the View and the Model.
     - **AddExpenseServlet**: Handles user input from the form, processes it, and sends it to the database via the model.
     - **ViewExpenseServlet**: Retrieves expense data from the database and forwards it to the view for display.
   - **Purpose**: Acts as an intermediary, ensuring smooth communication and business logic execution.

3. **Business Logic Module (Model - DAO Classes)**
   - **Function**: Handles all operations related to expense management, such as:
     - Adding new expenses to the database (addExpense()).
     - Retrieving a list of all expenses from the database (getExpenses()).
   - **Purpose**: Encapsulates database interaction and business rules, making the application modular and maintainable.

4. **Database Module (MySQL)**
   - **Function**: Stores and manages persistent data.
     - Maintains tables such as **expenses** (id, date, category, amount, description).
     - Supports queries for adding, retrieving, updating, and deleting expense records.
   - **Purpose**: Provides a reliable data storage and retrieval mechanism.

5. **Helper Utilities Module**
   - **Function**: Contains reusable helper classes or methods, such as:
     - **Database Connection Utility**: Ensures efficient and secure connections to the MySQL database using JDBC.
   - **Purpose**: Simplifies repetitive tasks and improves code reusability and readability.

Summary

Each module contributes to the overall functionality and maintainability of the Expense Tracker. The **View** enhances user experience, the **Controller** manages communication, the **Model** handles data, the **Database** stores information, and **Utilities** provide support for smooth operations.

Q3) Explain how you connect to the database in your project using JDBC. Include details about any tables used. (10 Marks)

Connecting to the Database in the Expense Tracker Project Using JDBC

The database connection is a crucial part of the Expense Tracker project, as it enables interaction between the application and the MySQL database for data storage and retrieval. Here's how it is done:

---

1. Setting Up the JDBC Connection

We use the JDBC (Java Database Connectivity) API to establish a connection between the Java application and the MySQL database.

Steps to Connect to the Database:

1. **Load the JDBC Driver**:
   Use the `Class.forName()` method to load the MySQL JDBC driver.

   java Copy code
   ```java
   Class.forName("com.mysql.cj.jdbc.Driver");
   ```

2. **Establish the Connection**:
   Use the `DriverManager.getConnection()` method to connect to the database.

   java Copy code
   ```java
   Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3
   ```

3. **Execute Queries**:
   Use `PreparedStatement` or `Statement` objects to execute SQL queries (e.g., INSERT, SELECT).

4. **Close the Connection**:
   Always close the `Connection`, `PreparedStatement`, and `ResultSet` objects to free up resources.

   java Copy code
   ```java
   connection.close();
   ```

2. Database Connection Utility Class

To streamline the process, a utility class is created for reusable database connections.

DatabaseConnection.java:

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/expense_track
    private static final String USER = "root";
    private static final String PASSWORD = "password";

    public static Connection getConnection() throws SQLException, ClassNotFoundE
        Class.forName("com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

3. Expense Tracker Tables

The following tables are used in the **expense_tracker** database:

expenses Table:

Stores details about individual expenses.

```sql
CREATE TABLE expenses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    date DATE NOT NULL,
    category VARCHAR(50) NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    description TEXT
);
```

Columns:

- `id`: Unique identifier for each expense (Primary Key).
- `date`: The date the expense occurred.
- `category`: The category of the expense (e.g., Food, Travel, Bills).
- `amount`: The amount of money spent.
- `description`: A brief description of the expense.

4. Using JDBC in the DAO Class

The DAO (Data Access Object) class handles database operations.

ExpenseDAO.java:
(Code is in pg no 1)

5. How it Works in the Servlets

The servlets interact with the DAO to handle user requests.

6. Advantages of JDBC in the Project

1. **Platform Independence**: JDBC works with multiple database systems (e.g., MySQL, PostgreSQL).
2. **Efficiency**: Using prepared statements enhances performance and security.
3. **Code Reusability**: The utility class (`DatabaseConnection`) ensures DRY (Don't Repeat Yourself) principles.
4. **Flexibility**: Any database-related changes can be managed by updating the DAO layer.

Q4) Write a servlet/JSP to DRIVING LOGIC. (30 Marks)

Servlet and JSP for Driving Logic in an Expense Tracker

In this scenario, the **driving logic** involves the core functionalities such as adding expenses and viewing them. Here's how the servlet and JSP code would work together to drive the logic:

Servlet Code (Driving Logic)

AddExpenseServlet.java

```
package servlet;

import dao.ExpenseDAO;
import model.Expense;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.text.SimpleDateFormat;
```

```
@WebServlet("/addExpense")
public class AddExpenseServlet extends HttpServlet
   { @Override
   protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
      try {
         String description = req.getParameter("description");
         String category = req.getParameter("category");
         double amount = Double.parseDouble(req.getParameter("amount"));
         String dateStr = req.getParameter("date");
         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
         java.util.Date date = sdf.parse(dateStr);

         Expense expense = new Expense();
         expense.setDescription(description);
         expense.setCategory(category);
         expense.setAmount(amount);
         expense.setDate(date);


         ExpenseDAO dao = new ExpenseDAO();
         dao.addExpense(expense);


         resp.sendRedirect("/expensestracker/viewExpenses");
      } catch (Exception e)
         { e.printStackTrace();
         resp.getWriter().println("Error: " + e.getMessage());
      }
   }
}
```

ViewExpensesServlet.java

```
package servlet;

import dao.ExpenseDAO;
import model.Expense;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet("/viewExpenses")
public class ViewExpensesServlet extends HttpServlet
    { @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try {
            ExpenseDAO dao = new ExpenseDAO();
            List<Expense> expenses = dao.getAllExpenses();
            System.out.println(expenses+"hi bro");
            req.setAttribute("expenses", expenses);
            req.getRequestDispatcher("expenses.jsp").forward(req,  resp);
```

```
        } catch (Exception e)
            { e.printStackTrace();
            resp.getWriter().println("Error: " + e.getMessage());
        }
    }
}
```

JSP Files (Driving Presentation Logic)

<u>index.jsp</u>

This JSP provides a form for users to add new expenses and shows a status message.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Expenses Tracker</title>
</head>

<body><!-- Header —>

<header>
<h1>Expense Tracker</h1>
</header>

<!-- Home Page Navigation -->
<div class="container">
<h2>Welcome to Expense Tracker</h2>
<div class="nav">
<a href="/expensestracker/viewExpenses">View Expenses</a>
<a href="#add-expense" onclick="showAddExpense()">Add Expense</a>

</div>
<!-- View Expenses Section (Initially hidden) -->
<div class="table-container" id="view-expenses">
<h2>All Expenses</h2>
<table>
<thead>
```

```
<tr>
<th>ID</th>
<th>Description</th>
<th>Category</th>
<th>Amount</th>
<th>Date</th></tr>
</thead>
<tbody>

<!-- Dynamically populate rows from database -->
<c:forEach var="expense" items="${expenses}">
<tr>
<td>${expense.id}</td>
<td>${expense.description}</td>
<td>${expense.category}</td>
<td>${expense.amount}</td>
<td>${expense.date}</td>
</tr>
</c:forEach>
</tbody>
</table>
</div>

<!-- Add Expense Form Section (Initially hidden) -->
<div class="form-container" id="add-expense">
<h2>Add Expense</h2>
<form action="/expensestracker/addExpense" method="POST">
<input type="text" name="description" placeholder="Description"
required>
<input type="text" name="category" placeholder="Category" required>
<input type="number" name="amount" placeholder="Amount"
required><input type="date" name="date" required>
<button type="submit">Add Expense</button>
</form>
</div>
</div>
<script>

// Function to show the "View Expenses" section
function showViewExpenses() {
document.getElementById('view-expenses').classList.add('active');
document.getElementById('add-expense').classList.remove('active');
}
// Function to show the "Add Expense" form section
```

```
function showAddExpense() {
document.getElementById('add-expense').classList.add('active');
document.getElementById('view-expenses').classList.remove('active');
}
</script>
</body>
</html>
```

Expenses.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>View Expenses</title>
</head>
<body>
<div class="container">
<h1>View Your Expenses</h1>
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Category</th>
<th>Amount</th>
<th>Date</th></tr>
</thead>
<tbody>
<c:forEach var="expense" items="${expenses}">
<tr>
<td>${expense.id}</td>
<td>${expense.description}</td>
<td>${expense.category}</td>
<td>${expense.amount}</td>
<td>${expense.date}</td>
</tr>
</c:forEach>
</tbody>
</table>
<a href="index.jsp">Add More Expenses</a>
```

```
</div>
</body>
</html>
```

How the Driving Logic Works

1. **Adding an Expense**:
    - The user fills out the form in **AddExpense.jsp**.
    - The form submits the data to **AddExpenseServlet**.
    - The servlet processes the request and saves the data in the database via ExpenseDAO.addExpense().
    - A success or error message is displayed to the user.
2. **Viewing Expenses**:
    - The user accesses **ViewExpenseServlet** by visiting a specific URL (e.g., /viewExpenses).
    - The servlet retrieves all expense records using ExpenseDAO.getExpenses().
    - The data is forwarded to **ViewExpense.jsp**, which displays it in a tabular format.

Advantages of This Approach

1. **Separation of Concerns**:
    - The servlets handle the driving logic, leaving the JSPs focused on presentation.
2. **Reusability**:
    - DAO methods like addExpense() and getExpenses() can be reused elsewhere.
3. **Scalability**:
    - Additional features like filtering expenses or generating reports can be easily added.

This implementation ensures that the driving logic is clean, modular, and aligns well with the MVC architecture.

DBConnection.java

```
package utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/expense_tracker";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() throws SQLException
        { try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException e) {
            throw new SQLException("JDBC Driver not found", e);
        }
    }
}
```

(Extra)

Web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
           file:///path/to/your/web-app_3_1.xsd"
        version="3.1">

  <servlet>
    <servlet-name>expenseTrackerServlet</servlet-name>
    <servlet-class>com.expensestracker.ExpenseTrackerServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>expenseTrackerServlet</servlet-name>
    <url-pattern>/trackExpenses</url-pattern>
  </servlet-mapping>
</web-app>
```