MVC Architecture with JSP:

Sure! The Model-View-Controller (MVC) architecture is a design pattern used in software development to separate an application into three interconnected components: Model, View, and Controller. In the context of Java web development with JSP (JavaServer Pages), MVC helps to organize and manage the application's structure and logic. Let's break down the MVC architecture and provide two code examples to illustrate it.

1. Model:
The Model represents the application's data and business logic. It is responsible for managing and processing data. In a Java web application, the Model typically consists of Java classes that interact with the database or other data sources.

2. View:
The View represents the user interface and presentation layer. In a JSP-based application, the View is responsible for rendering HTML and displaying data to the user.

3. Controller:
The Controller acts as an intermediary between the Model and the View. It receives user input from the View, processes it, and interacts with the Model to update the data. In a Java web application, the Controller is often implemented as servlets.

Here are two code examples to demonstrate the Java MVC architecture with JSP:

Example 1: Model

Suppose you have a simple Model class for managing user data. This class interacts with a database to retrieve and update user information:

```java
public class UserModel {
    // Data access methods to interact with the database
    public User getUserById(int userId) {
        // Retrieve user from the database
        // ...
    }

    public void updateUser(User user) {
        // Update user data in the database
        // ...
    }
}
```

Example 2: View and Controller

In this example, we'll create a simple JSP page and a servlet as the Controller to display and update user information.

JSP (View): user.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
%>
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
    <h1>User Profile</h1>
    <p>Name: ${user.name}</p>
    <p>Email: ${user.email}</p>
    <form action="UpdateUserController" method="post">
        <input type="text" name="name" value="${user.name}" placeholder="New Name">
        <input type="text" name="email" value="${user.email}" placeholder="New Email">
        <input type="submit" value="Update">
    </form>
</body>
</html>
```

Servlet (Controller): UpdateUserController.java

```java
@WebServlet("/UpdateUserController")
public class UpdateUserController extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String newName = request.getParameter("name");
        String newEmail = request.getParameter("email");

        UserModel userModel = new UserModel();

        // Retrieve user from the database (assuming you have the user's ID)
        User user = userModel.getUserById(userId);

        // Update the user's data
```

```
        user.setName(newName);
        user.setEmail(newEmail);

        // Save the updated user back to the database
        userModel.updateUser(user);

        // Redirect back to the user.jsp page
        response.sendRedirect("user.jsp");
    }
}
```

In this example, the JSP page displays user information, and the servlet receives user input for updating the data. The Model (UserModel) is responsible for database interaction and data manipulation.

This separation of concerns in the MVC architecture helps maintain a clean and organized codebase, making it easier to manage and scale your Java web application.