MVC con Servlets y JSP

Raúl Estrada

Octubre 2020

1. Overview

In this quick article, we'll create a small web application that implements the Model View Controller (MVC) design pattern, using basic Servlets and JSPs.

We'll explore a little bit about how MVC works, and its key features before we move on to the implementation.

2. Introduction to MVC

Model-View-Controller (MVC) is a pattern used in software engineering to separate the application logic from the user interface. As the name implies, the MVC pattern has three layers.

The Model defines the business layer of the application, the Controller manages the flow of the application, and the View defines the presentation layer of the application.

Although the MVC pattern isn't specific to web applications, it fits very well in this type of applications. In a Java context, the Model consists of simple Java classes, the Controller consists of servlets and the View consists of JSP pages.

Here're some key features of the pattern:

- It separates the presentation layer from the business layer
- The Controller performs the action of invoking the Model and sending data to View
- The Model is not even aware that it is used by some web application or a desktop application

Let's have a look at each layer.

2.1. The Model Layer

This is the data layer which contains business logic of the system, and also represents the state of the application.

It's independent of the presentation layer, the controller fetches the data from the Model layer and sends it to the View layer.

2.2. The Controller Layer

Controller layer acts as an interface between View and Model. It receives requests from the View layer and processes them, including the necessary validations.

The requests are further sent to Model layer for data processing, and once they are processed, the data is sent back to the Controller and then displayed on the View.

2.3. The View Layer

This layer represents the output of the application, usually some form of UI. The presentation layer is used to display the Model data fetched by the Controller.

3. MVC With Servlets and JSP

To implement a web application based on MVC design pattern, we'll create the *Student* and *StudentService* classes – which will act as our Model layer.

S*tudentServlet* class will act as a Controller, and for the presentation layer, we'll create *student-record.jsp* page.

Now, let's write these layers one by one and start with *Student* class:

```
public class Student {
    private int id;
    private String firstName;
    private String lastName;

    // constructors, getters and setters goes here
}
```

Let's now write our *StudentService* which will process our business logic:

```
1  public class StudentService {
2
3      public Optional<Student> getStudent(int id) {
4          switch (id) {
5              case 1:
6                  return Optional.of(new Student(1, "John", "Doe"));
7              case 2:
8                  return Optional.of(new Student(2, "Jane", "Goodall"));
9              case 3:
10                 return Optional.of(new Student(3, "Max", "Born"));
11             default:
12                 return Optional.empty();
13         }
14     }
15 }
```

Now let's create our Controller class *StudentServlet*:

```
@WebServlet(
  name = "StudentServlet",
  urlPatterns = "/student-record")
public class StudentServlet extends HttpServlet {

    private StudentService studentService = new StudentService();

    private void processRequest(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

        String studentID = request.getParameter("id");
        if (studentID != null) {
            int id = Integer.parseInt(studentID);
            studentService.getStudent(id)
              .ifPresent(s -> request.setAttribute("studentRecord", s));
        }

        RequestDispatcher dispatcher = request.getRequestDispatcher(
          "/WEB-INF/jsp/student-record.jsp");
        dispatcher.forward(request, response);
    }
```

```
23
24      @Override
25      protected void doGet(
26        HttpServletRequest request, HttpServletResponse response)
27        throws ServletException, IOException {
28
29          processRequest(request, response);
30      }
31
32      @Override
33      protected void doPost(
34        HttpServletRequest request, HttpServletResponse response)
35        throws ServletException, IOException {
36
37          processRequest(request, response);
38      }
39  }
```

This servlet is the controller of our web application.

First, it reads a parameter *id* from the request. If the *id* is submitted, a *Student* object is fetched from the business layer.

Once it retrieves the necessary data from the Model, it puts this data in the request using the *setAttribute()* method.

Finally, the Controller *forwards* the request and response objects to a JSP, the view of the application.

Next, let's write our presentation layer *student-record.jsp*:

```
<html>
    <head>
        <title>Student Record</title>
    </head>
    <body>
    <%
        if (request.getAttribute("studentRecord") != null) {
            Student student = (Student) request.getAttribute("studentRecord");
    %>

    <h1>Student Record</h1>
    <div>ID: <%= student.getId()%></div>
    <div>First Name: <%= student.getFirstName()%></div>
    <div>Last Name: <%= student.getLastName()%></div>

    <%
        } else {
    %>

    <h1>No student record found.</h1>

    <% } %>
    </body>
</html>
```

And, of course, the JSP is the view of the application; it receives all the information it needs from the Controller, it doesn't need to interact with the business layer directly.

4. Conclusion

In this tutorial, we've learned about the MVC i.e. Model View Controller architecture, and we focused on how to implement a simple example.