SOEN 387

WEB-BASED ENTERPRISE APPLICATIONS DESIGN

Tutorial 7

Web Presentation Patterns

Agenda

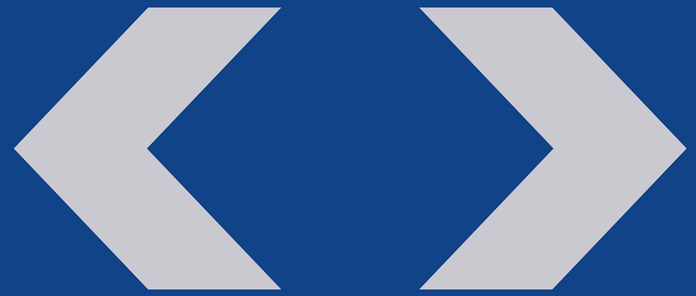
I. MVC Pattern

II. Front Controller Pattern

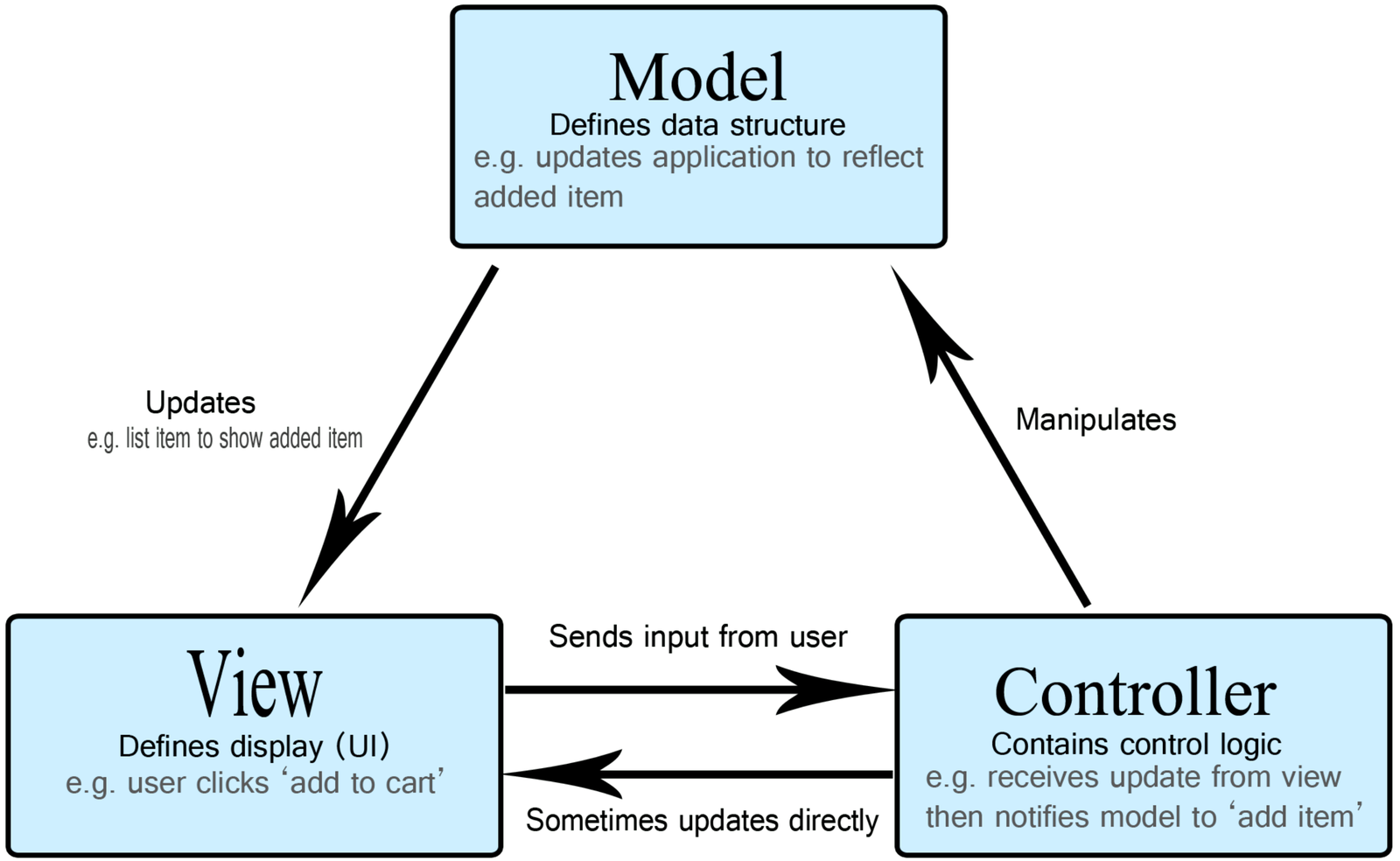
III. Page Controller Pattern

MVC Pattern

Model - View Controller



MVC Pattern


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

MVC Pattern

Step 1 : view.jsp

MVC Example

```
<%@ page contentType="text/html;charset=UTF-8"
language="java" %>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
    <title>MVC Example</title>
</head>
<body>
<form action="Servlet" method="POST">
    Email: <input type="text" name="email">
    <br />
    Password: <input type="text" name="password" />
    <input type="submit" value="Submit" />
</form>
</body>
</html>
```

MVC Pattern

Step 2 :
MVC_Servlet.java

```
public class mvc_Servlet extends javax.servlet.http.HttpServlet {
    protected void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws
javax.servlet.ServletException, IOException {
        String email=request.getParameter("email");
        String password=request.getParameter("password");
        javabean testobj = new javabean();
        testobj.setEmail(email);
        testobj.setPassword(password);
        request.setAttribute("demobean",testobj);
        RequestDispatcher rd=request.getRequestDispatcher("success.jsp");
        rd.forward(request, response);
    }
    protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws
javax.servlet.ServletException, IOException {

    }
}
```

MVC Pattern

Step 3 :
Javabean.java

```
public class javabean implements Serializable{

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    private String email="null";
    private String password="null";
}
```

MVC Pattern

Step 4 :
success.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@page import="model.javabean"%>
<html>
<head>
    <title>Title</title>
</head>
<body>
<%
    javabean test =(javabean) request.getAttribute("demobean");
    out.print("Welcome, "+test.getEmail());
%>
</body>
</html>
```

MVC Pattern

Step 5 : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>controller.mvc_Servlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>/Servlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>view.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

MVC Pattern

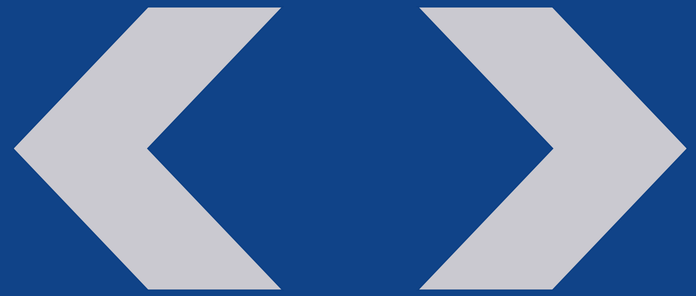
View.jsp

Success.jsp

Output

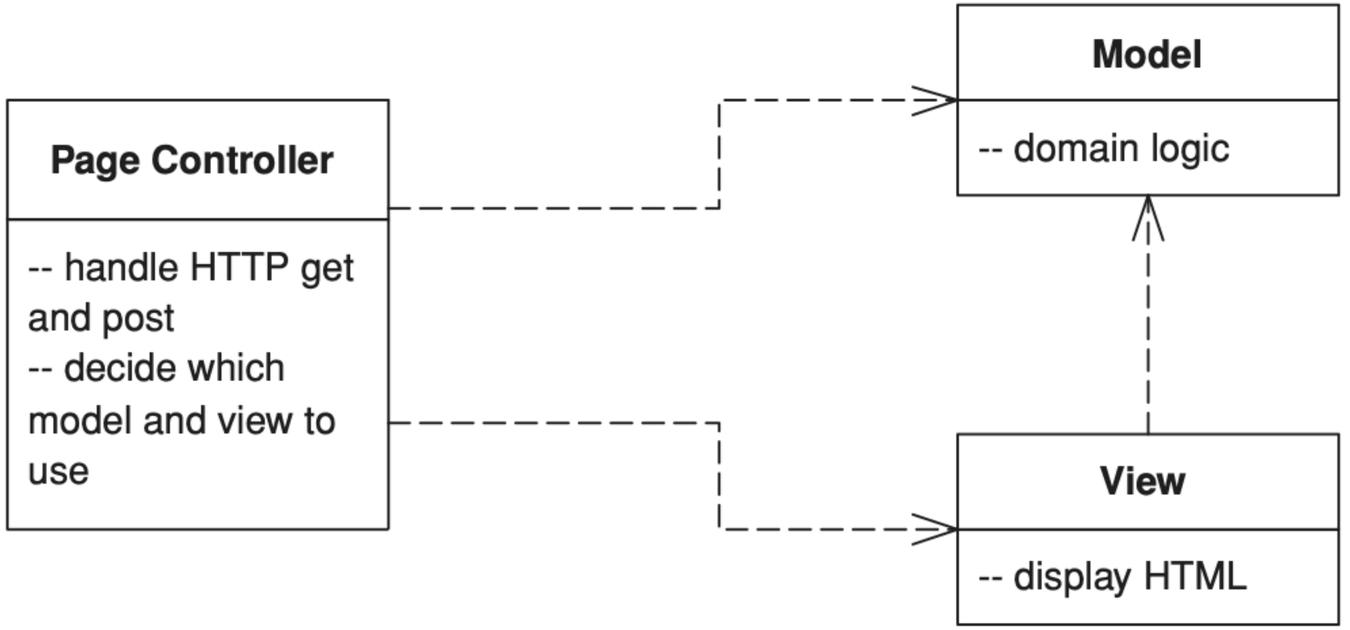
Page Controller Pattern

Page Controller



Page Controller

An object that handles a request for a specific page or action on a Web site.



Page Controller

Step 1 :
Album.java

Example:

```
public enum AlbumType { NOT_SPECIFIED, Regular, CLASSICAL }

public class Album {
    private int id;
    private String name;
    private AlbumType atype;

    public Album(int id, String name, AlbumType type) {
        this.id = id;
        this.name = name;
        this.atype = type;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
```

Page Controller

Step 1 :
Album.java

```
public void setId(int id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }
    public AlbumType getAlbumType() {
        return atype;
    }
    public void setAlbumtype(AlbumType atype) {
        atype = atype;
    }

    public static Album find(int id){
        switch(id) { // change this to read from a data source
            case 1: return new Album(1, "My first Album", AlbumType.NOT_SPECIFIED);
            case 2: return new Album(2, "Regular Album", AlbumType.Regular);
            case 3: return new Album(3, "Classical Album", AlbumType.CLASSICAL);
        }
        return null; // not found
```

Page Controller

Step 3 : AlbumController. java

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class AlbumController extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        Album album = Album.find(Integer.parseInt(request.getParameter("id")));
        if (album == null) {
            forward("/missingAlbumError.jsp", request, response);
            return;
        }
```

Page Controller

Step 3 : AlbumController. java

```
request.setAttribute("helper", album);
        if (album.getAlbumType() == AlbumType.CLASSICAL)
            forward("/classicalAlbum.jsp", request, response);
        else
            forward("/album.jsp", request, response);
    }

protected static void forward(String target,
                              HttpServletRequest request,
                              HttpServletResponse response)
            throws IOException, ServletException {
        RequestDispatcher dispatcher =
request.getServletContext().getRequestDispatcher(target);
        dispatcher.forward(request, response);
    }
}
```

Page Controller

Step 4 : index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<!DOCTYPE html>
<html>
<head>
    <title>Form input</title>
</head>
<body>
<form action="AlbumController" method="POST">
    Enter your id:
    <input type="text" name="id" />
    <input type="submit" value="Submit" />
</form>
</body>
</html>
```

Page Controller

Step 5 :
album.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>Regular Album</h1>
</body>
</html>
```

Step 6 :
classicalAlbum
.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>classic Album</h1>
</body>
</html>
```

Page Controller

Step 7 : missingAlbumError .jsp

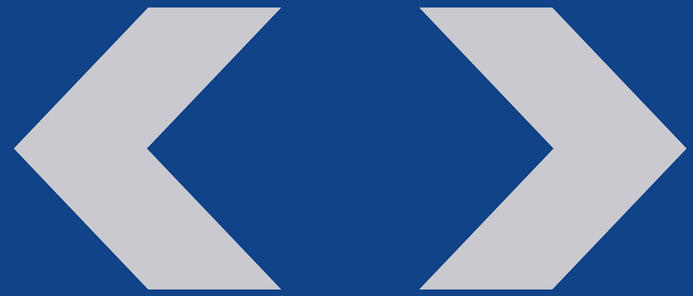
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>Missing album</h1>
</body>
</html>
```

Step 8 : Output

Index.jsp

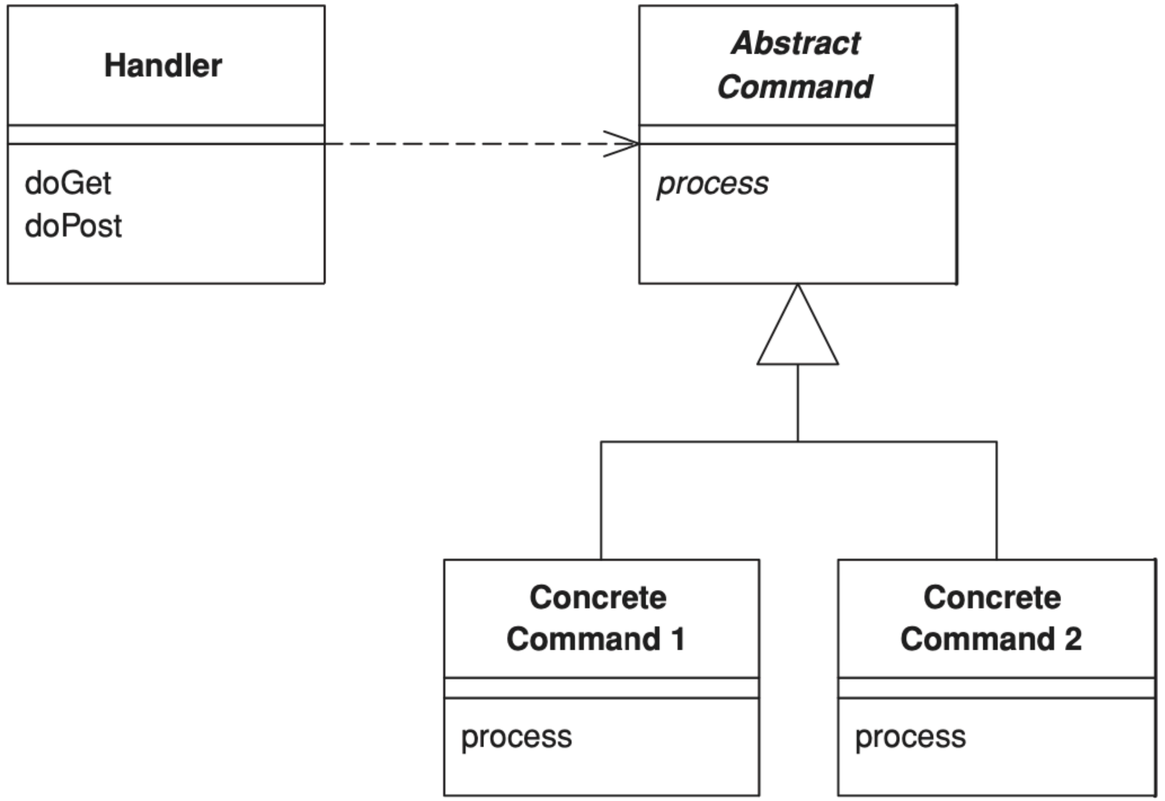
ID : 0 ID : 1 ID : 2

Front Controller Pattern



Front Controller

A controller that handles all requests for a Web site.



The *Front Controller Pattern* is mainly divided into two parts. A single dispatching controller and a hierarchy of commands.

Front Controller

Step 1. Maven Dependencies

The source code for this example can be found on this link.

The example has been taken from https://www.baeldung.com/java-front-controller-pattern

Example

First, we'll setup a new *Maven WAR* project with *javax.servlet-api* included:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.0-b01</version>
    <scope>provided</scope>
</dependency>
```

as well as *jetty-maven-plugin*:

```
<plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.4.0.M1</version>
    <configuration>
        <webApp>
            <contextPath>/front-controller</contextPath>
        </webApp>
    </configuration>
</plugin>
```

Step 2. Model

Front Controller

Next, we'll define a *Model* class and a model *Repository*.

```
public class Book {
    private String author;
    private String title;
    private Double price;
    // standard constructors, getters and setters
}
```

This will be the repository

```
public interface Bookshelf {
    default void init() {
        add(new Book("Wilson, Robert Anton & Shea, Robert",
          "Illuminati", 9.99));
        add(new Book("Fowler, Martin",
          "Patterns of Enterprise Application Architecture", 27.88));
    }
    Bookshelf getInstance();
    <E extends Book> boolean add(E book);
    Book findByTitle(String title);
}
```

Front Controller

Step 3. Front Controller

```
package com.baeldung.patterns.front.controller;

public class FrontControllerServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        FrontCommand command = getCommand(request);
        command.init(getServletContext(), request, response);
        command.process();
    }
    private FrontCommand getCommand(HttpServletRequest request) {
        try {
            Class type = Class.forName(String.format("com.baeldung.enterprise.patterns.front."
              + "controller.commands.%sCommand",
              request.getParameter("command")));
            return (FrontCommand) type.asSubclass(FrontCommand.class).newInstance();
        } catch (Exception e) {
            return new UnknownCommand();
        }
    }
}
```

Step 4. Front Command

Front Controller

Let's implement an abstract class called *FrontCommand*, which is holding the behavior common to all commands.

```
public abstract class FrontCommand {
    protected ServletContext context;
    protected HttpServletRequest request;
    protected HttpServletResponse response;

    public void init(ServletContext servletContext, HttpServletRequest servletRequest,
      HttpServletResponse servletResponse) {
        this.context = servletContext;
        this.request = servletRequest;
        this.response = servletResponse;
    }
    public abstract void process() throws ServletException, IOException;
    protected void forward(String target) throws ServletException, IOException {
        target = String.format("/WEB-INF/jsp/%s.jsp", target);
        RequestDispatcher dispatcher = context.getRequestDispatcher(target);
        dispatcher.forward(request, response);
    }
}
```

Front Controller

Step 5. Search Command

A concrete implementation of this abstract *FrontCommand* would be a *SearchCommand*.

```
public class SearchCommand extends FrontCommand {
    @Override
    public void process() throws ServletException, IOException {
        Book book = new BookshelfImpl().getInstance()
          .findByTitle(request.getParameter("title"));
        if (book != null) {
            request.setAttribute("book", book);
            forward("book-found");
        } else {
            forward("book-notfound");
        }
    }
}
```

Front Controller

Step 6. Unknown Command

we'll implement a second command, which is fired as fallback
in all cases, a command request is unknown to the Servlet:

```
public class UnknownCommand extends FrontCommand {
    @Override
    public void process() throws ServletException, IOException {
        forward("unknown");
    }
}
```

This view will be reachable at http://localhost:8080/front-controller/?command=Order&title=any-title or by completely leaving out the *URL* parameters.

Step 6. Web.xml

We now should be familiar with the *Front Controller Pattern* and its implementation as Servlet and command hierarchy.

Front Controller

With this *web.xml* we're able to run our web-application in any Servlet container:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
 http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
 version="3.1">
  <servlet>
    <servlet-name>front-controller</servlet-name>
    <servlet-class>
      com.baeldung.enterprise.patterns.front.controller.FrontControllerServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>front-controller</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

References

- https://www.guru99.com/jsp-mvc.html
- https://www.javaguides.net/2018/08/application-controller-design-pattern-in-java.html
- https://www.baeldung.com/java-front-controller-pattern
- https://www.baeldung.com/mvc-servlet-jsp
- https://github.com/light314/poeaa/tree/master/src/com/poeaa/distribution
- http://ce.sharif.edu/courses/97-98/2/ce418-1/resources/root/Books/Patterns%20of%20Enterprise%20Application%20Architecture%20-%20Martin%20Fowler.pdf

Thank You