1 MVC

In this chapter we introduce the student to MVC, a design pattern. A design pattern is a manner or approach followed to develop software. We will explain the concept and provide examples to illuminate it further.

1.1 What is MVC?

MVC is an acronym that stands for **M**odel **V**iew **C**ontroller. It is a software design pattern, that instructs us on how to develop software. MVC categorizes software components into three, namely:

- Model;
- View; and
- Controller.

1.1.1 Models

Models are components that are used to represent objects ("things") and process them. Models have a state. They are responsible for the business logic of the application. Examples of models are your simple **Java classes (POJOs), interfaces,** and **Enterprise Java Bens (EJBs)**.

1.1.2 Views

Views are web components that allow applications to interact with the user in an easy manner. They perform input/output operations by allowing a user to provide data to an application through a user interface, and also display outcome or processed data through the same user interface. Examples of views are **HTML** pages and **Java Server Pages (JSPs)**.

The difference between HTML pages and a JSPs is that an HTML pages work with static data while JSPs work with dynamic data. Static data is data that is not changing, it is known before hand. A classical example of static data is the sun, it never changes, it keeps on shinning. Dynamic data is data that is not constant, it changes frequently

depending on what is requested or required. A typical example of dynamic data is room temperature or time. These values are not static, they change constantly.

1.1.3 Controllers

Controllers are also web components that control the flow of data in an application. They receive data from views and thereafter decide what to do with the data. They can pass data to models for processing or simply pass the data to views for display. An example of a controller is a **Servlet**.

1.2 Example

In this section we are going to do an example. The purpose of the example is to demonstrate to the student how to design and implement a solution to a web problem. So, given a problem statement, we will do three things, namely:

- Discuss the flow of our solution.
- Identify MVC components emanating from the discussion.
- Implement the design in software.

Activity

Say we want to create a web application that will display a personalised greeting based on the user inputs. The application envisaged application should allow a user to enter their **name** and **gender**. Upon receiving the inputs, the application is expected to generate and display a personalised greeting message such as **"Hello Mr or Ms X. Welcome to the world of Web Applications Development.",** where X is a placeholder for the entered name, and Ms or Mr depends on the gender value entered.

To do

Design a solution to the problem. The solution must entail a discussion of the program flow, identification of MVC components which emanate from the discussion, and an implementation of the design in software.

Program flow

The program needs a home or landing page which will welcome users. This page will have static content. As a result an **HTML** page will be used for this purpose. The page will also have a link to another page that will allow a user to enter their **name** and **gender**. This too will be **HTML** page. The page will have a link to a controller which will determine the flow of the data.

The controller will be a **Servlet**. The servlet will call a **model** to perform the business logic on the provided data. The business logic to be performed is the generation of a personalised greeting message. The servlet will then forward the personalised greeting to a **JSP** for viewing. The JSP is the right component for this purpose because the generated data (greeting) is dynamic, it is not data that is known in advance, it is non-static data. The JSP will display the dynamic data. Lastly, the page will have a link to the home page.

MVC components

Consequently our design will consist of the following MVC components:

Views

- **index.html**: a home page.
- **personal_details.html**: a page that allows a user to enter their name and gender.
- **greeting.jsp**: page that displays the personalised greeting.

Controller

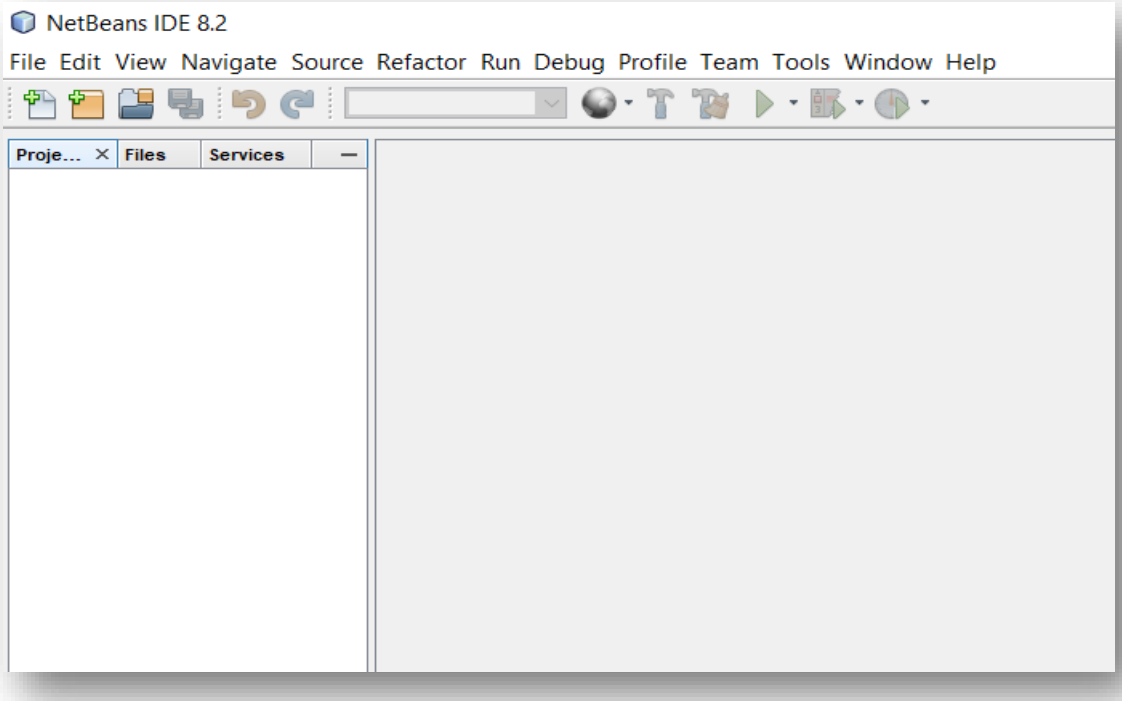
- **PersonalisedGreetingServlet.java**: controls the flow of greeting data.

Model

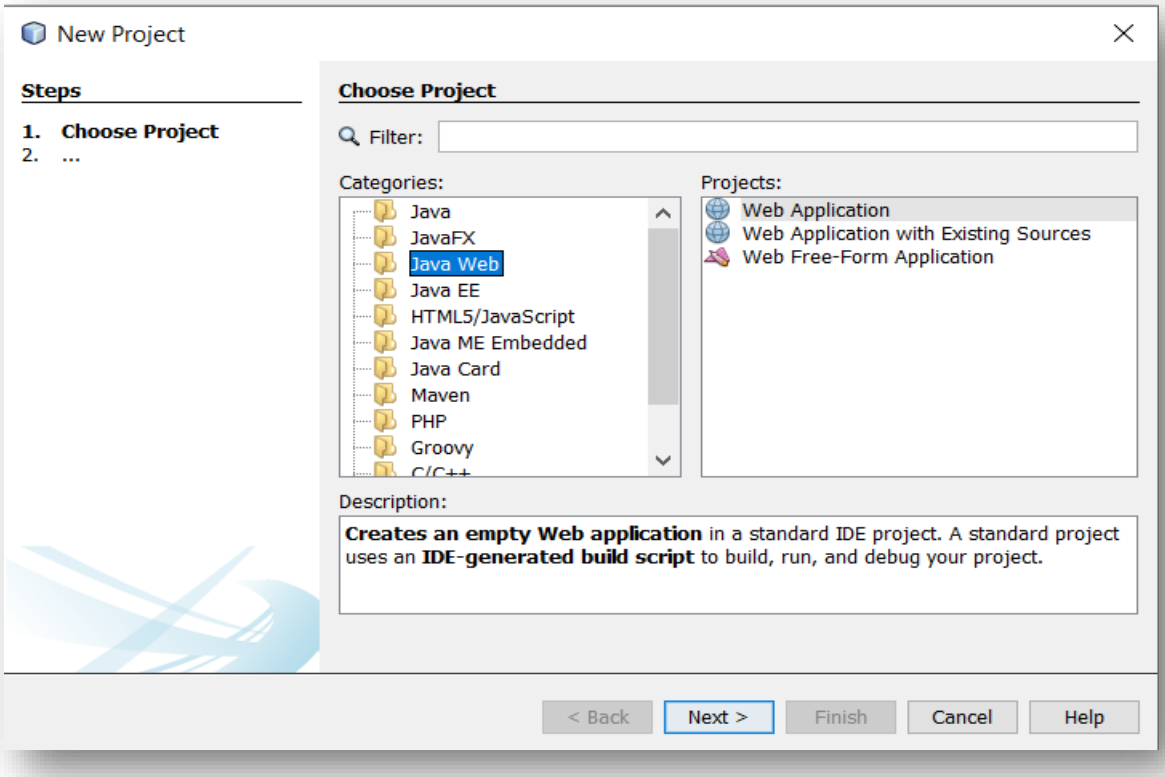
- **PersonalisedGreetingGenerator.java:** generates the personalised greeting message.

Implementation

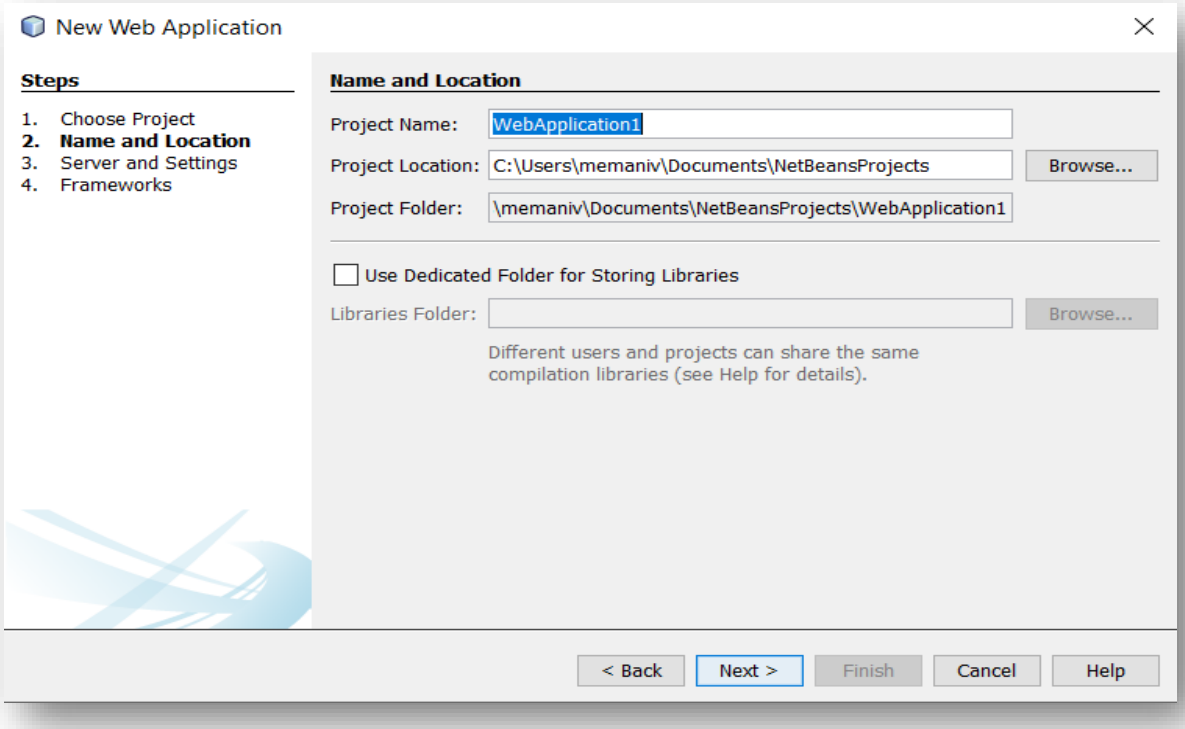
Launch NetBeans



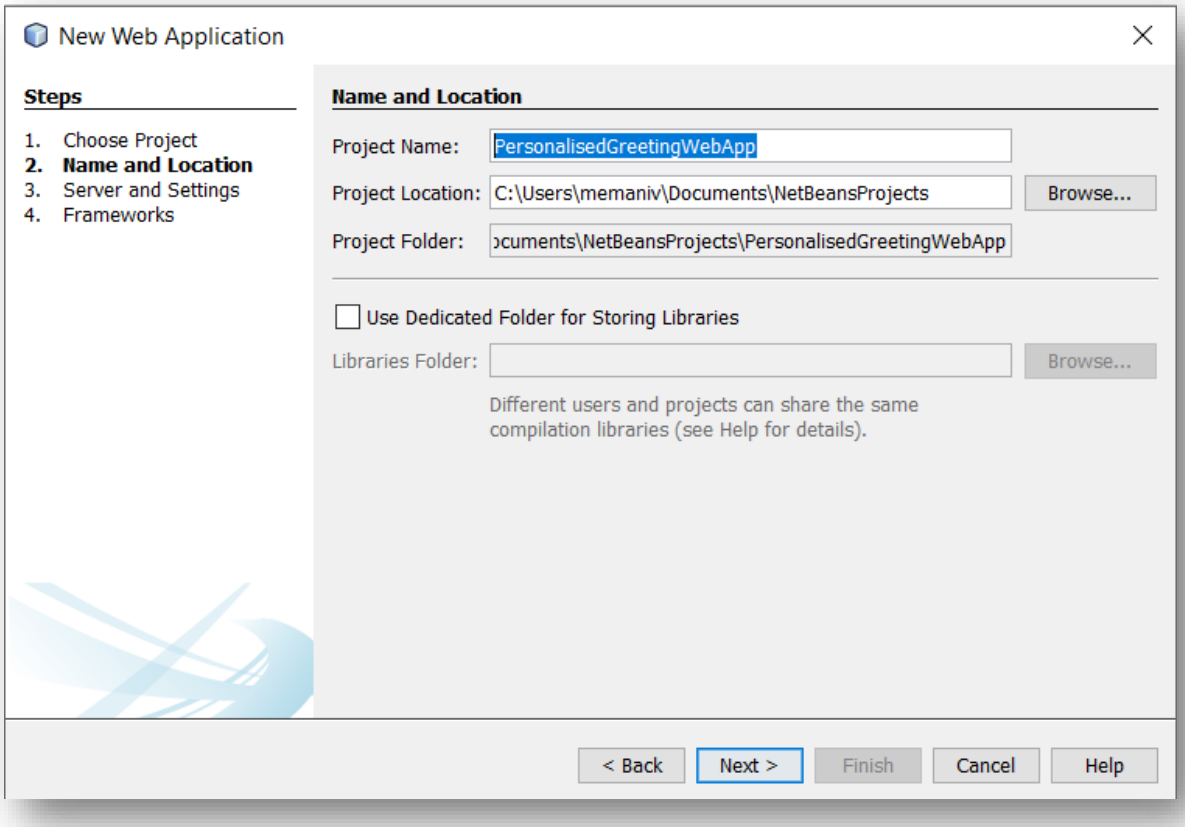
Select **File | New Project**



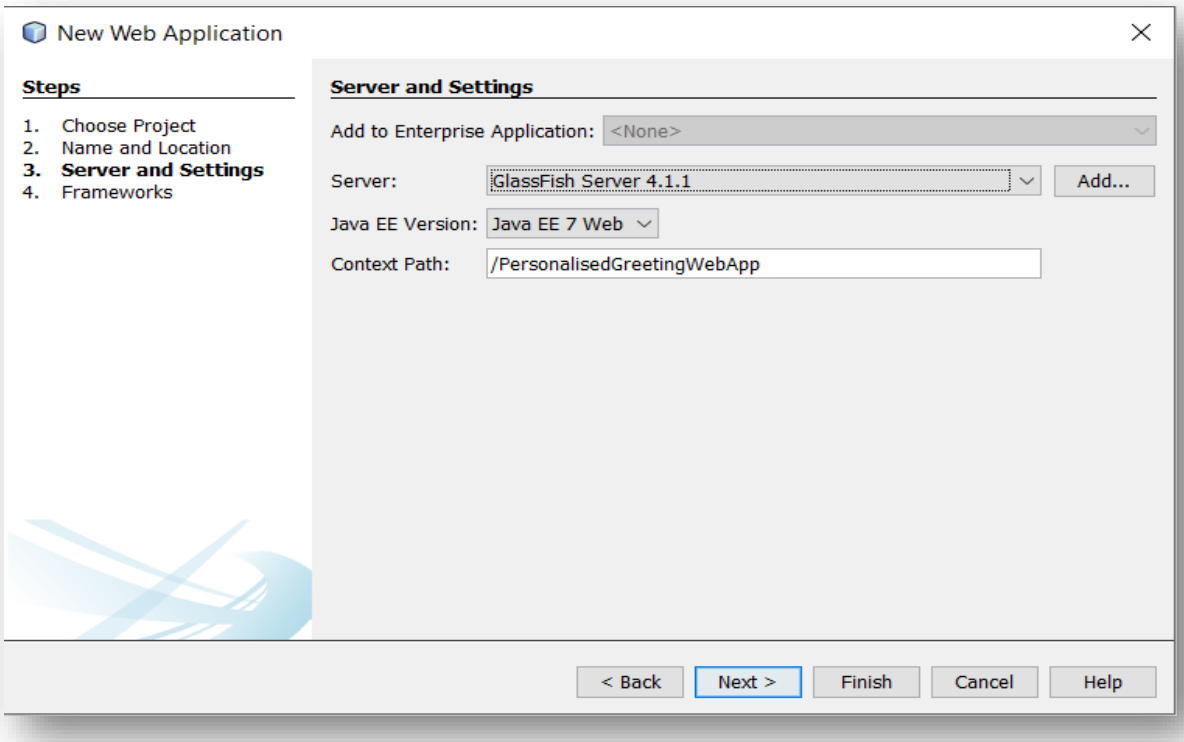
Click **Next**



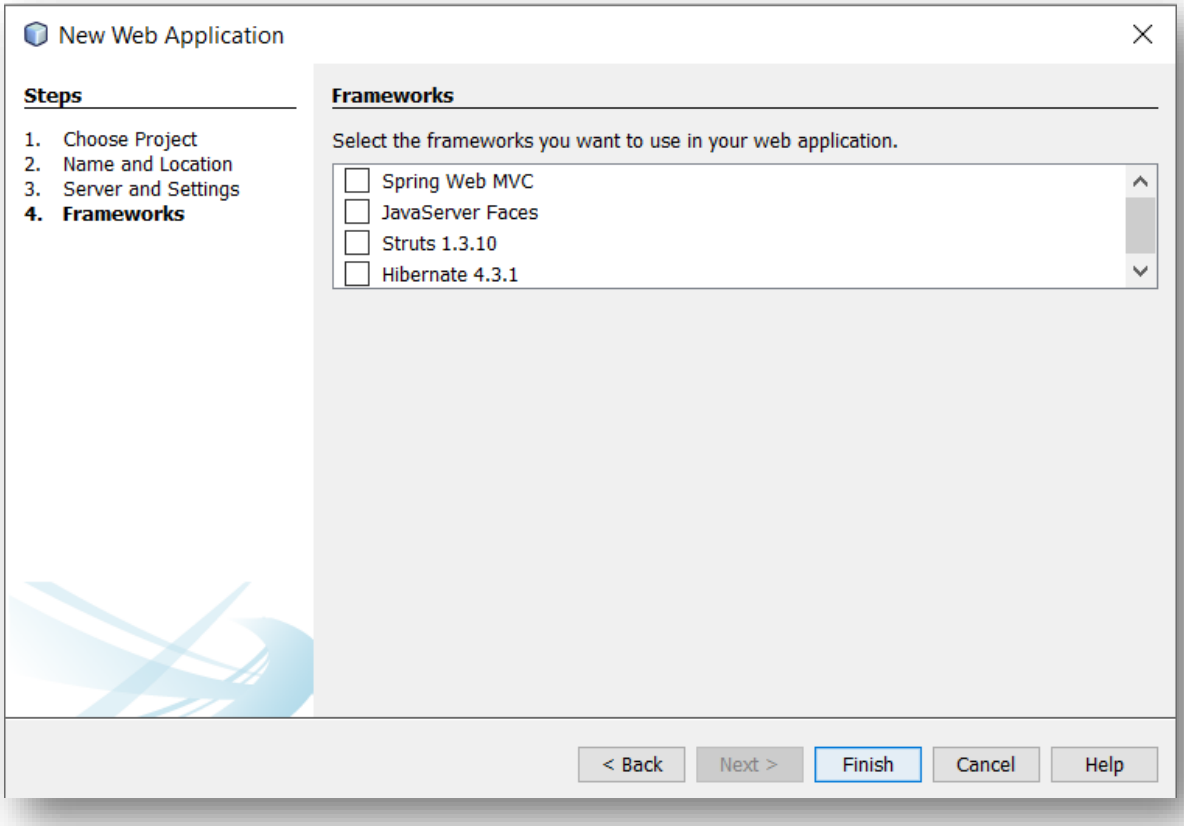
Name the project as **PersonalisedGreetingWebApp**



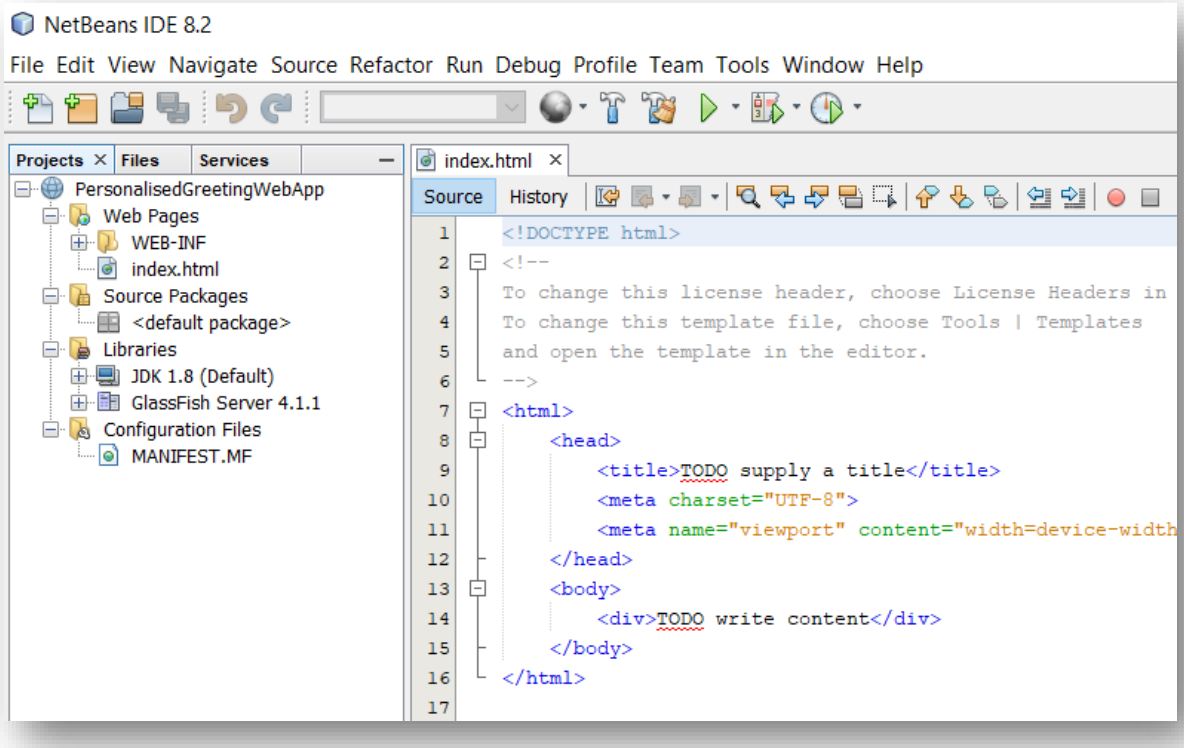
Click **Next**



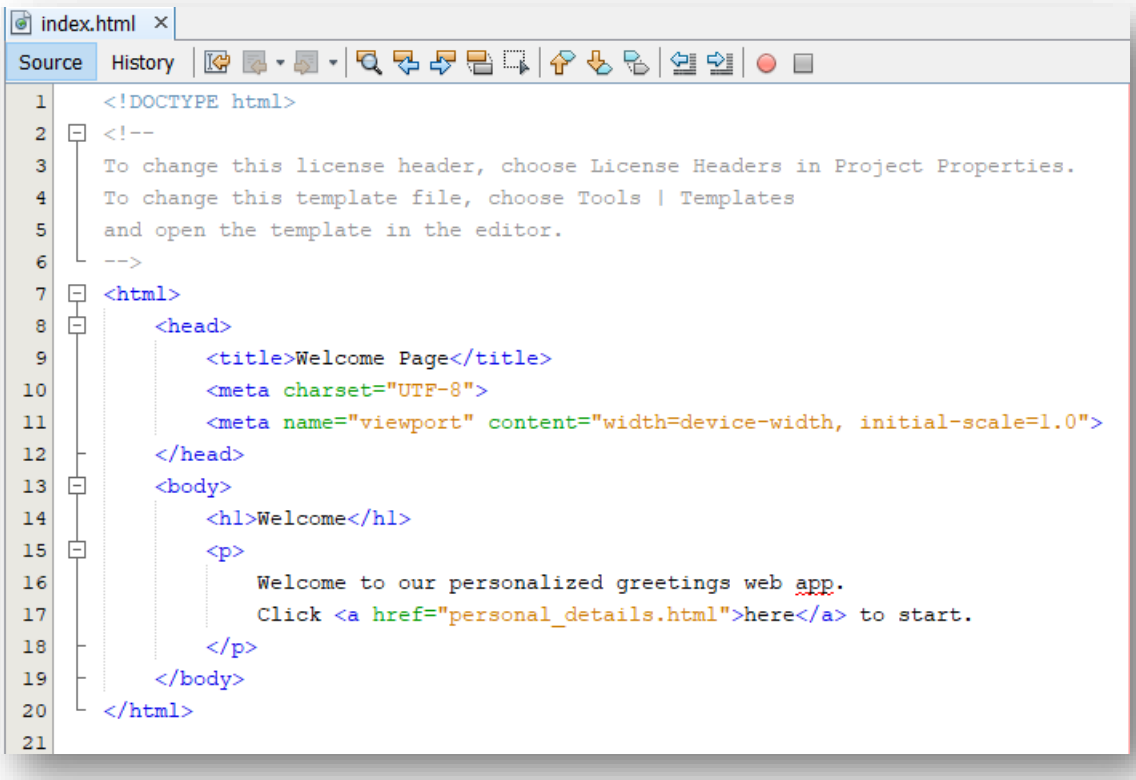
Click **Next**



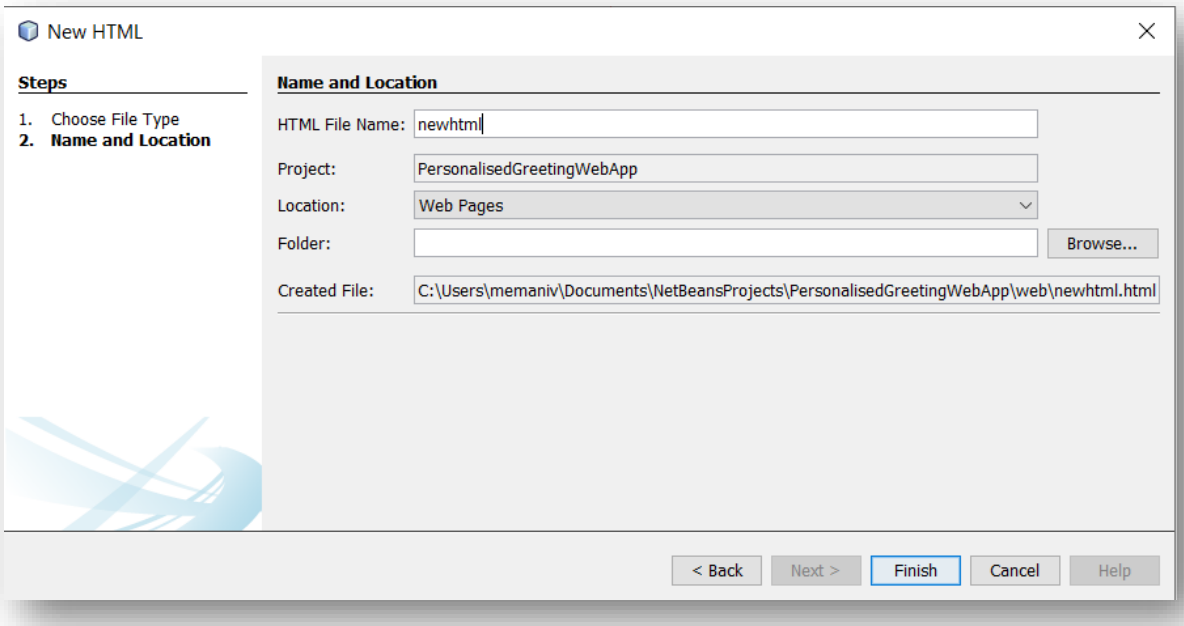
Click **Finish**



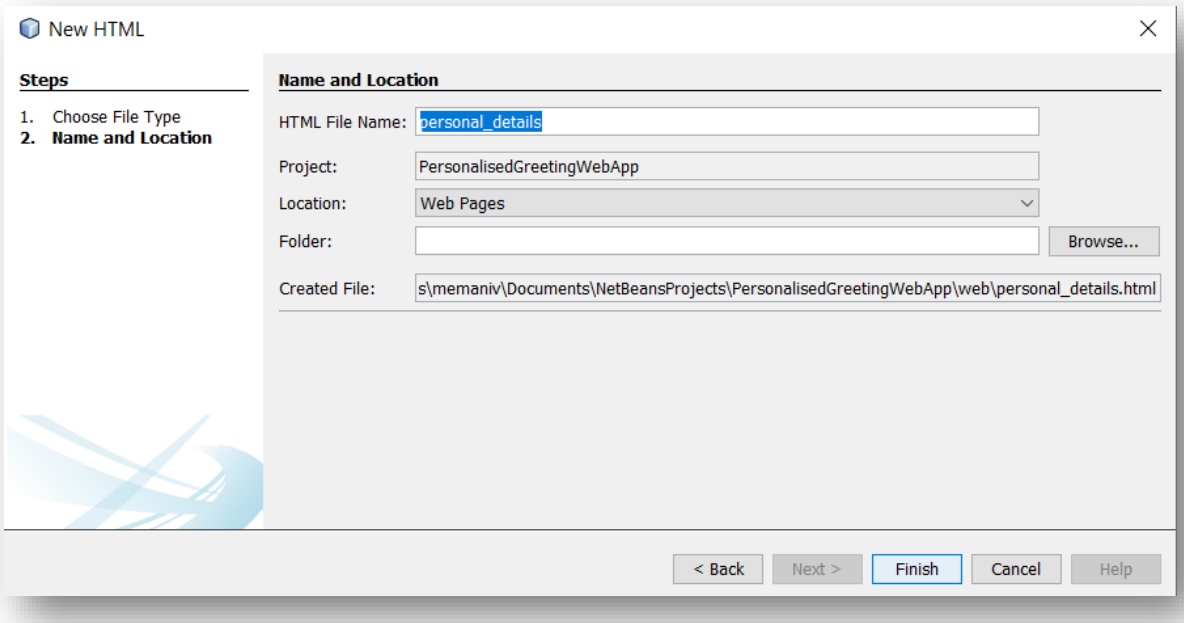
Modify the **index.html** file



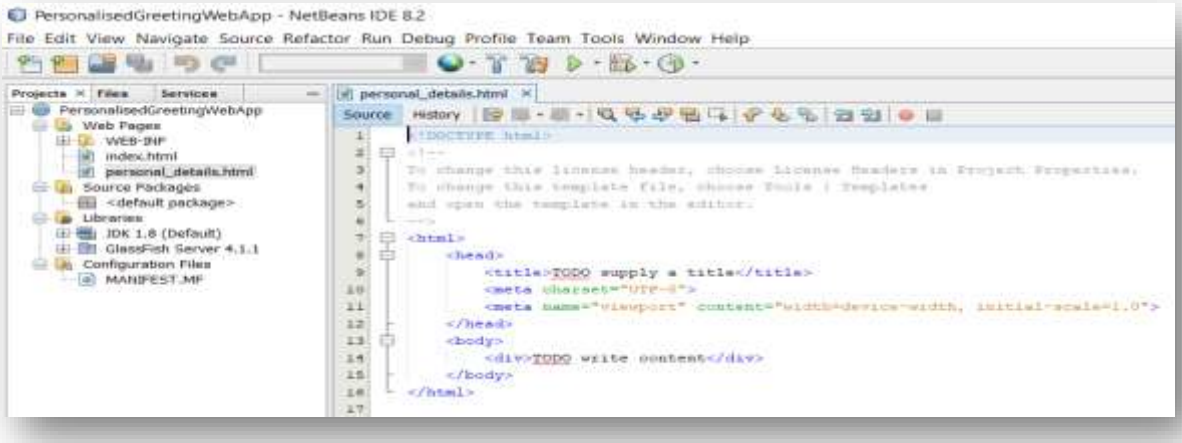
Right-click on the project select **New | HTML**



Name the file as **personal_details**



Click **Finish**



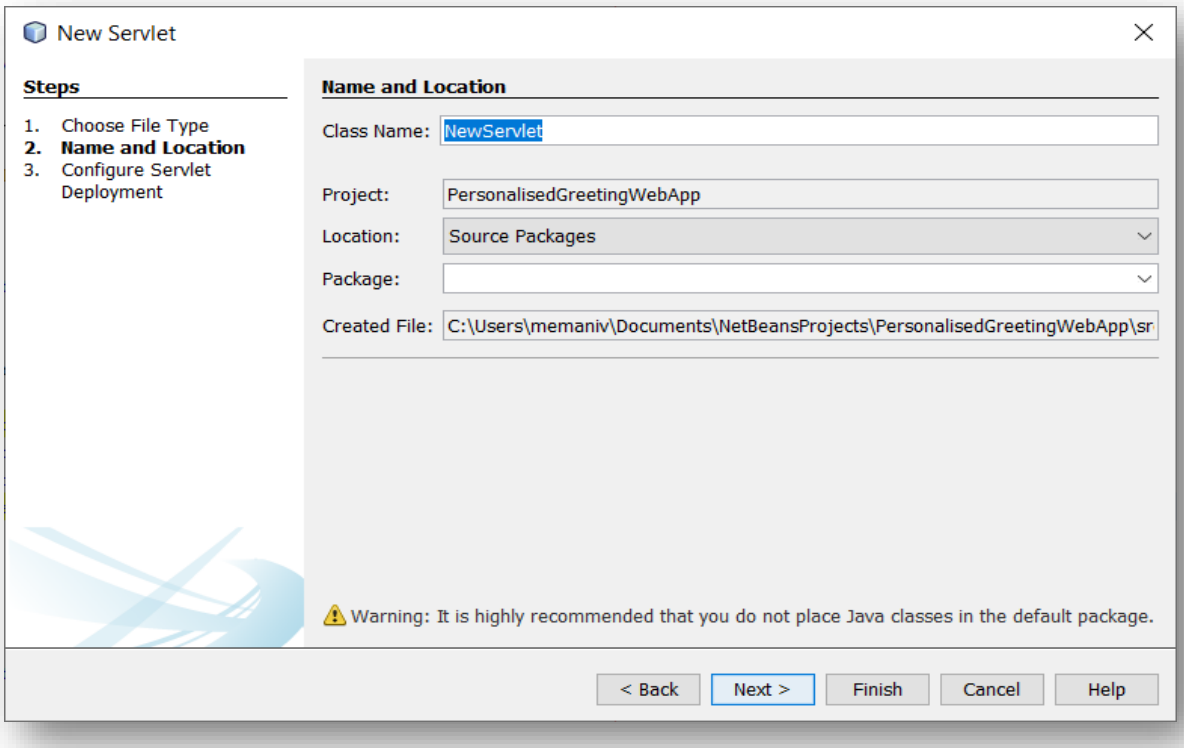
Modify **personal_details.html**

personal_details.html

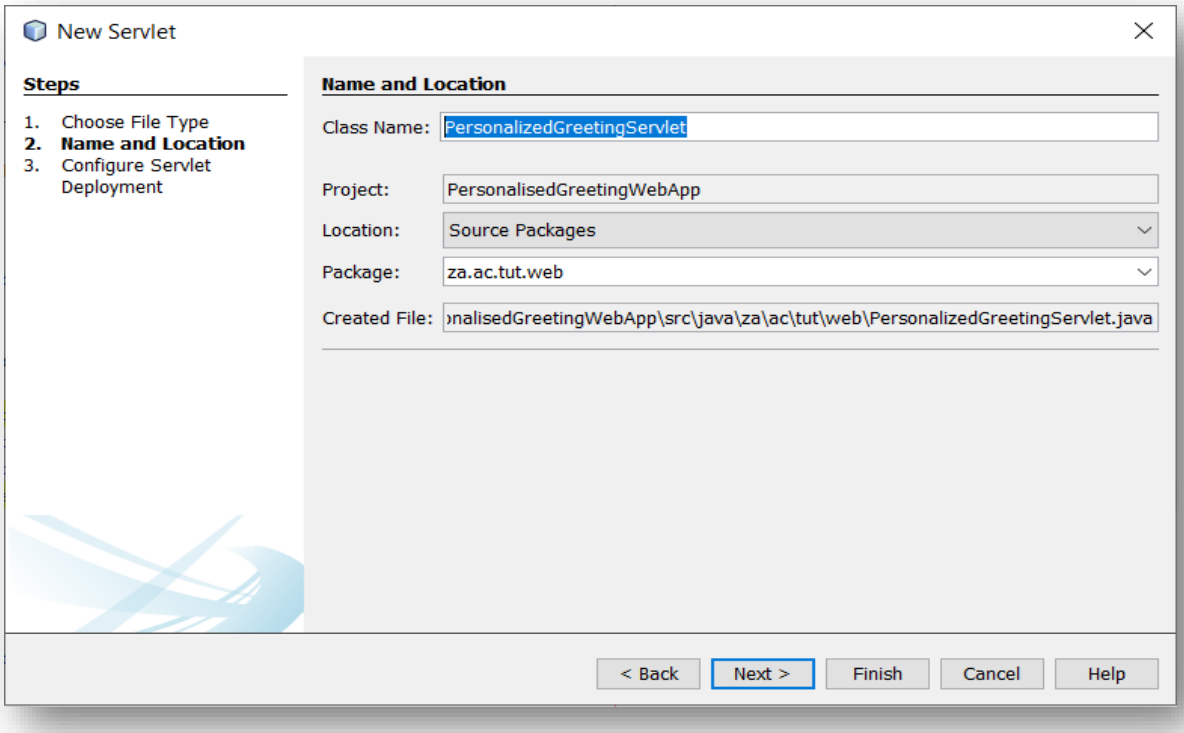
Source History

```
7   <html>
8       <head>
9           <title>Personal Details Page</title>
10          <meta charset="UTF-8">
11          <meta name="viewport" content="width=device-width, initial-scale=1.0">
12      </head>
13      <body>
14          <h1>Personal details</h1>
15          <p>
16              Please provide your personal details below:
17          </p>
18          <form action="PersonalizedGreetingServlet.do" method="POST">
19              <table>
20                  <tr>
21                      <td>Name: </td>
22                      <td><input type="text" name="name" required=""/></td>
23                  </tr>
24                  <tr>
25                      <td>Gender: </td>
26                      <td>
27                          <select name="gender">
28                              <option value="M">Male</option>
29                              <option value="F">Female</option>
30                          </select>
31                      </td>
32                  </tr>
33                  <tr>
34                      <td></td>
35                      <td>
36                          <input type="submit" value="GENERATE GREETING" />
37                      </td>
38                  </tr>
39              </table>
40          </form>
41      </body>
42  </html>
43
```

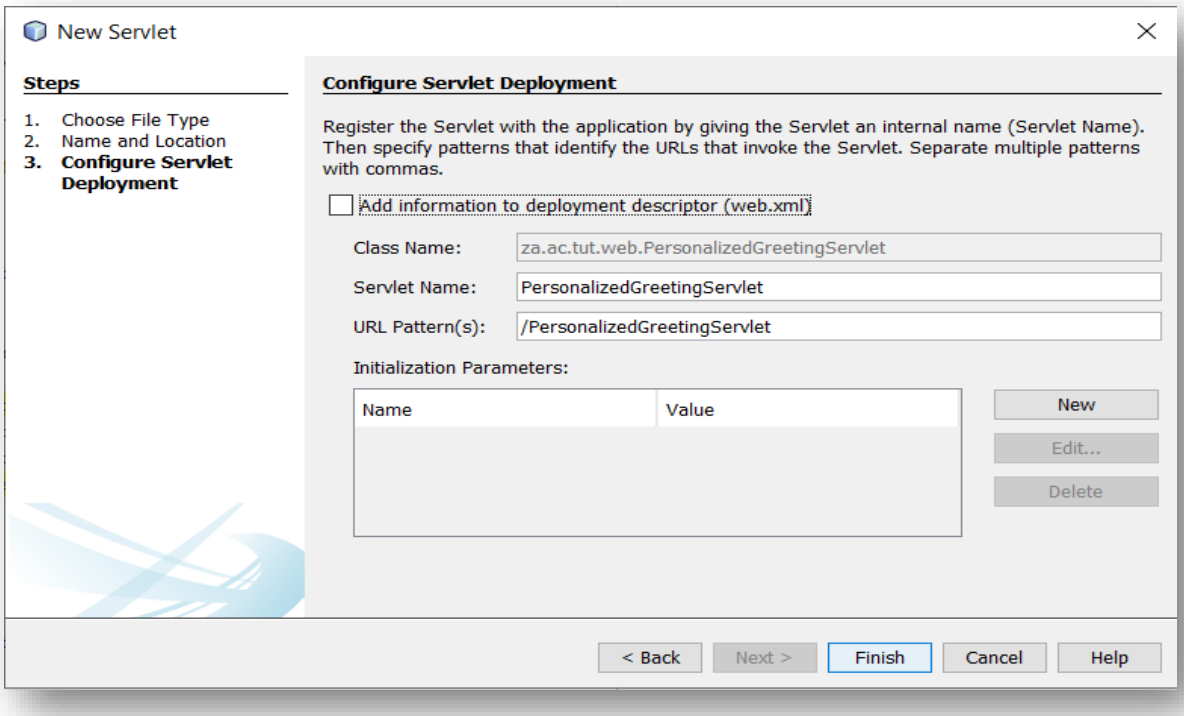
Right-click on the project and select **New | Servlet**



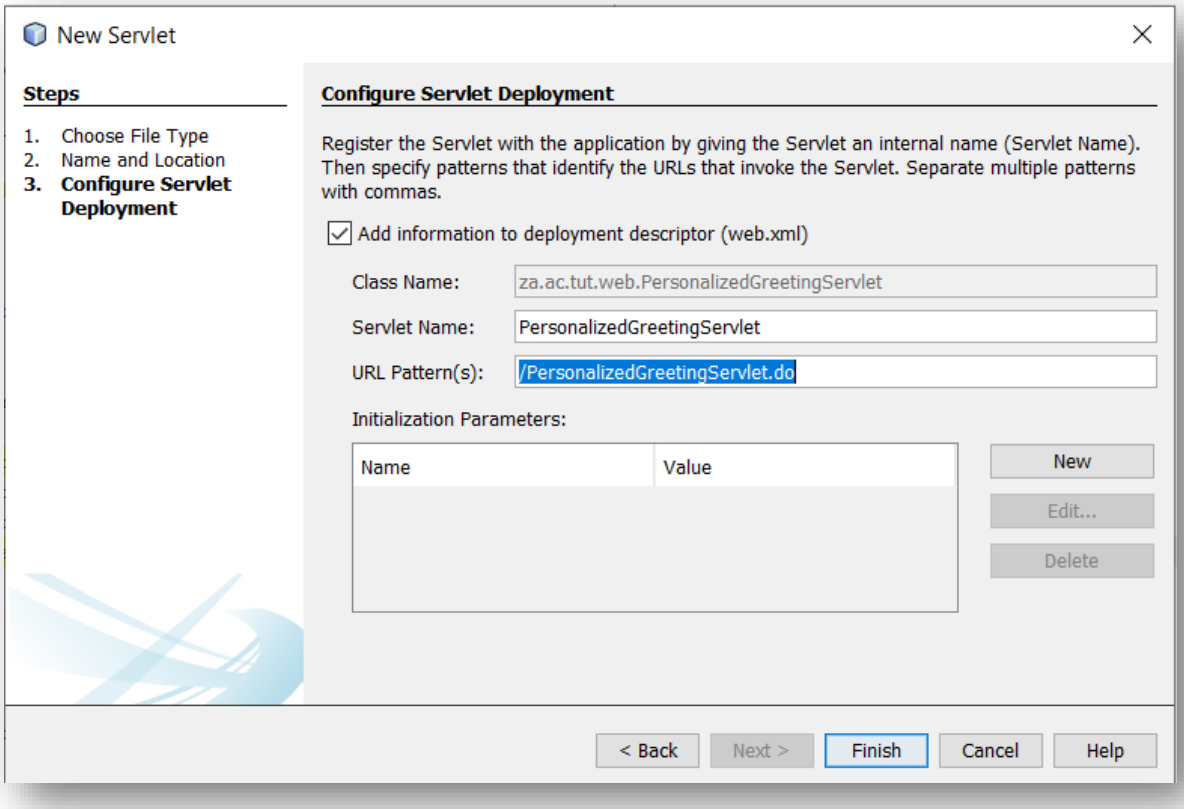
Name the servlet as **PersonalizedGreetingServlet** and package it as **za.ac.tut.web**



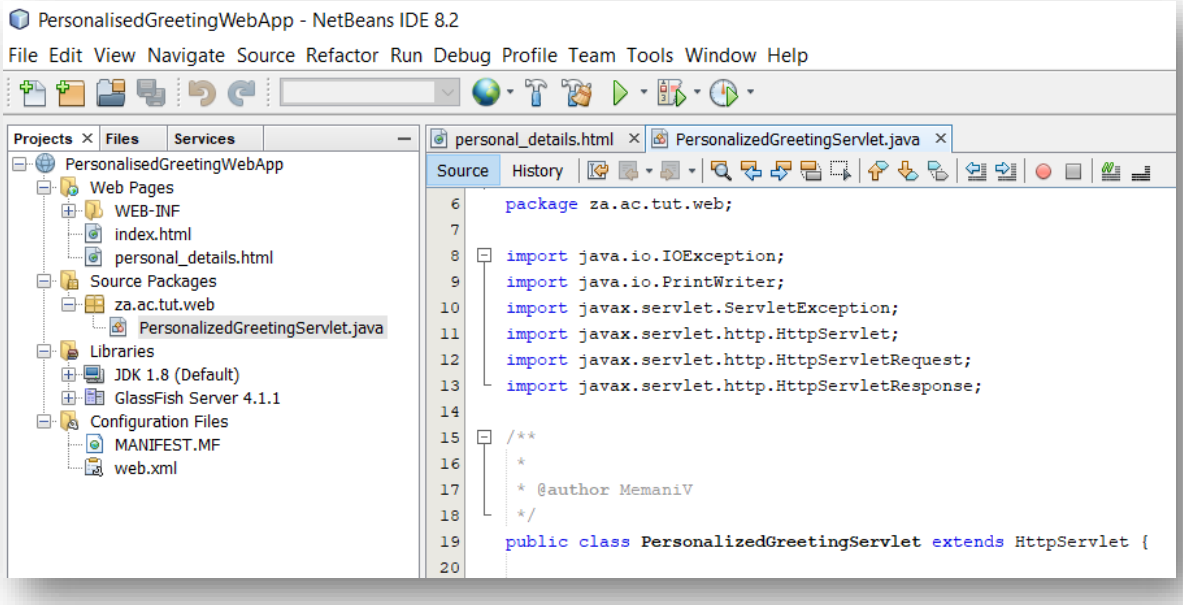
Click **Next**



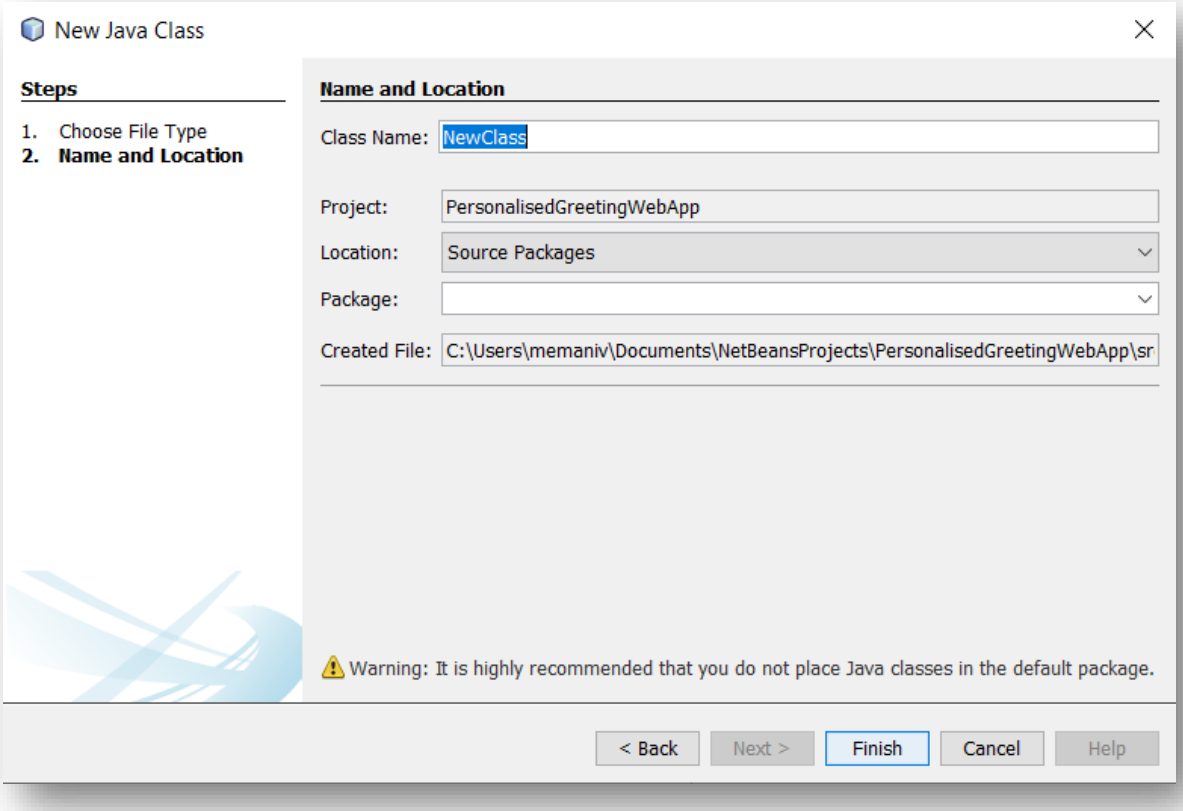
Select the checkbox and append **.do** to the URL pattern.



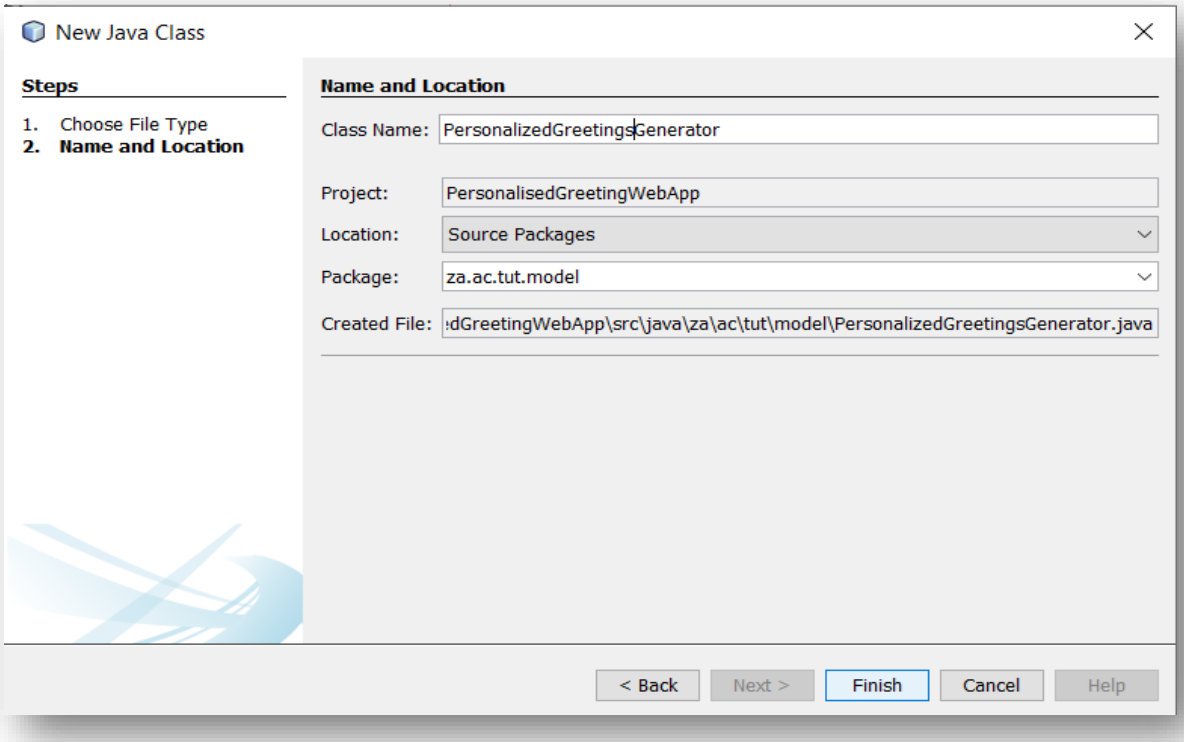
Click **Finish**



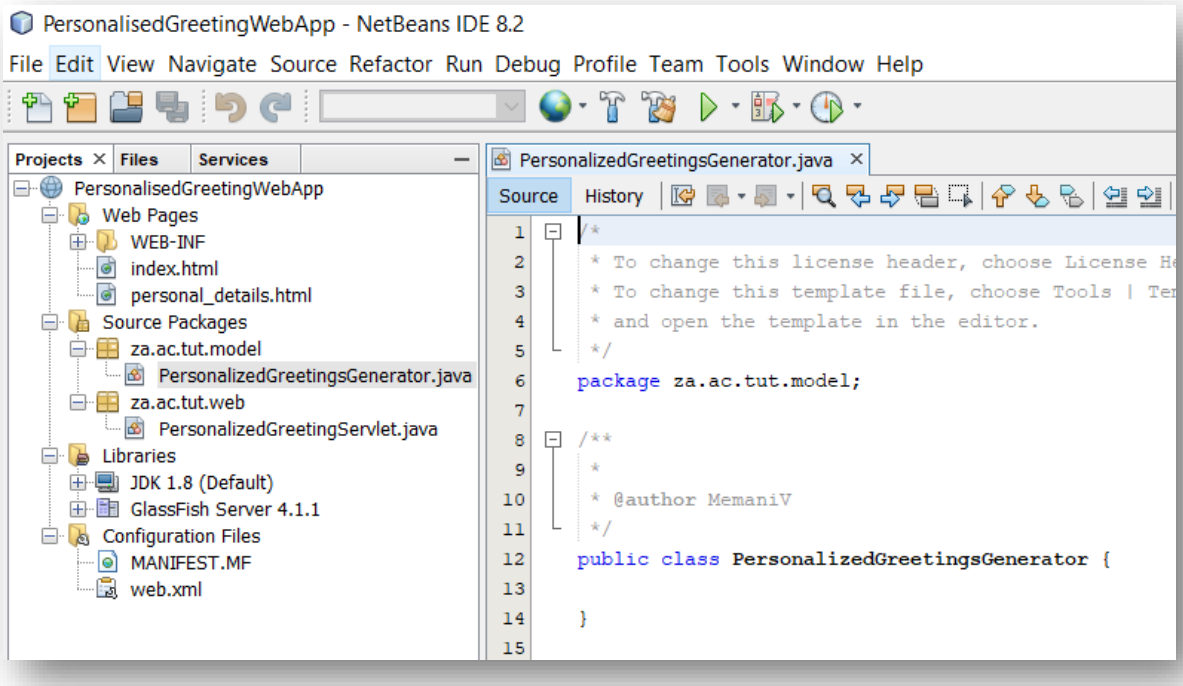
Right-click on the project and select **New | Java Class**



Name the class as **PersonalizedGreetingsGenerator** and package it as **za.ac.tut.model**



Click **Finish**



Modify the class

PersonalizedGreetingsGenerator.java

Source History

```java
package za.ac.tut.model;

/**
 *
 * @author MemaniV
 */
public class PersonalizedGreetingsGenerator {
    private String name;
    private char gender;

    public PersonalizedGreetingsGenerator(String name, char gender){
        this.name = name;
        this.gender = gender;
    }

    public String generateGreeting(){
        String title, greeting;

        title = determineTitle();
        greeting = "Hi " + title + " " + name + ". Welcome to the world of web development.";
        return greeting;
    }

    private String determineTitle() {
        String title = "Mr";

        if(gender == 'F' || gender == 'f'){
            title = "Ms";
        }

        return title;
    }
}
```

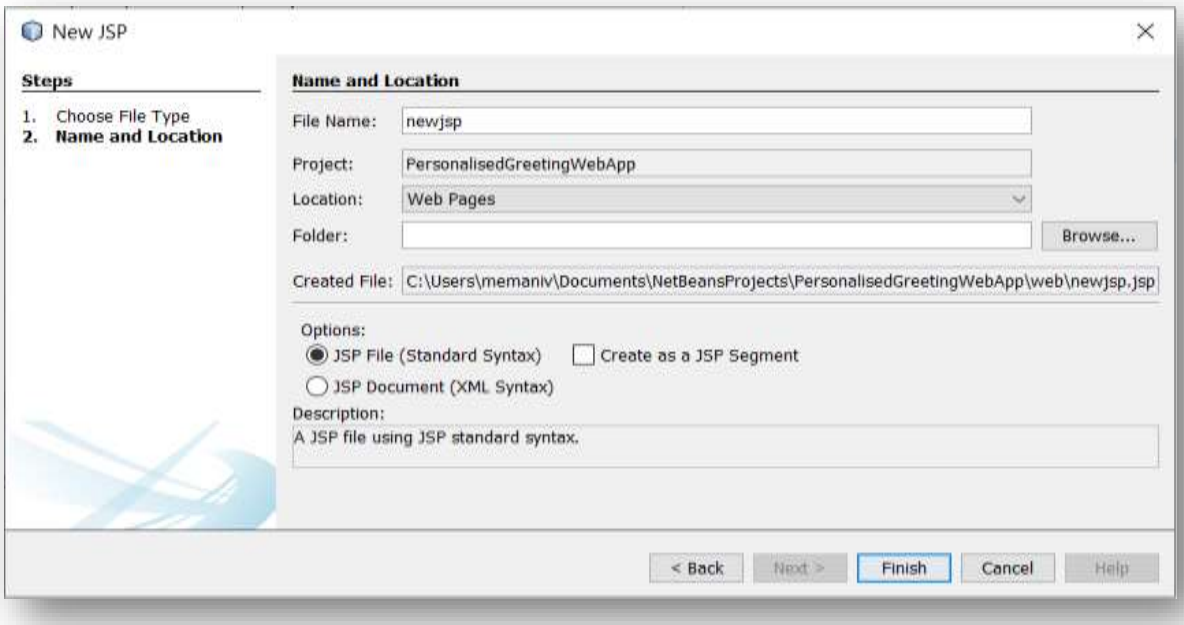
Modify the servlet

PersonalizedGreetingServlet.java

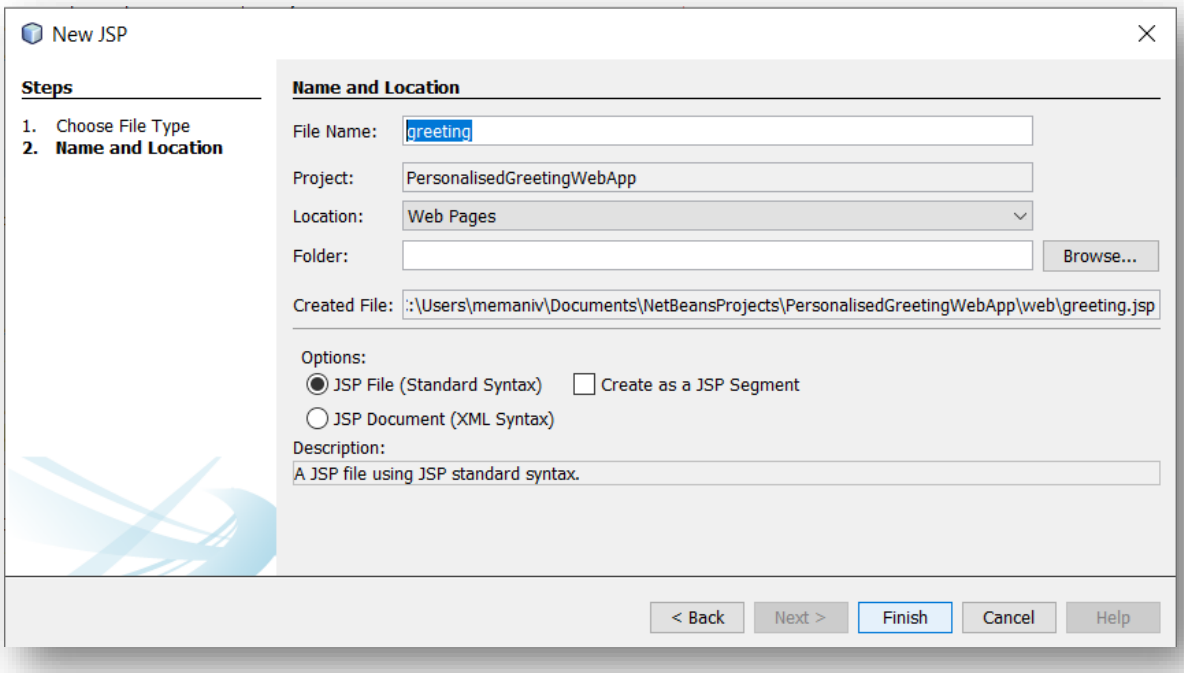
Source History

```java
6   package za.ac.tut.web;
7
8   import java.io.IOException;
9   import javax.servlet.RequestDispatcher;
10  import javax.servlet.ServletException;
11  import javax.servlet.http.HttpServlet;
12  import javax.servlet.http.HttpServletRequest;
13  import javax.servlet.http.HttpServletResponse;
14  import za.ac.tut.model.PersonalizedGreetingsGenerator;
15
16  /**
17   *
18   * @author MemaniV
19   */
20  public class PersonalizedGreetingServlet extends HttpServlet {
21      @Override
22      protected void doPost(HttpServletRequest request, HttpServletResponse response)
23              throws ServletException, IOException {
24          String name = request.getParameter("name");
25          char gender = request.getParameter("gender").charAt(0);
26
27          PersonalizedGreetingsGenerator pgg = new PersonalizedGreetingsGenerator(name, gender);
28          String greeting = pgg.generateGreeting();
29          request.setAttribute("greeting", greeting);
30
31          RequestDispatcher disp = request.getRequestDispatcher("greeting.jsp");
32          disp.forward(request, response);
33      }
34  }
35
```

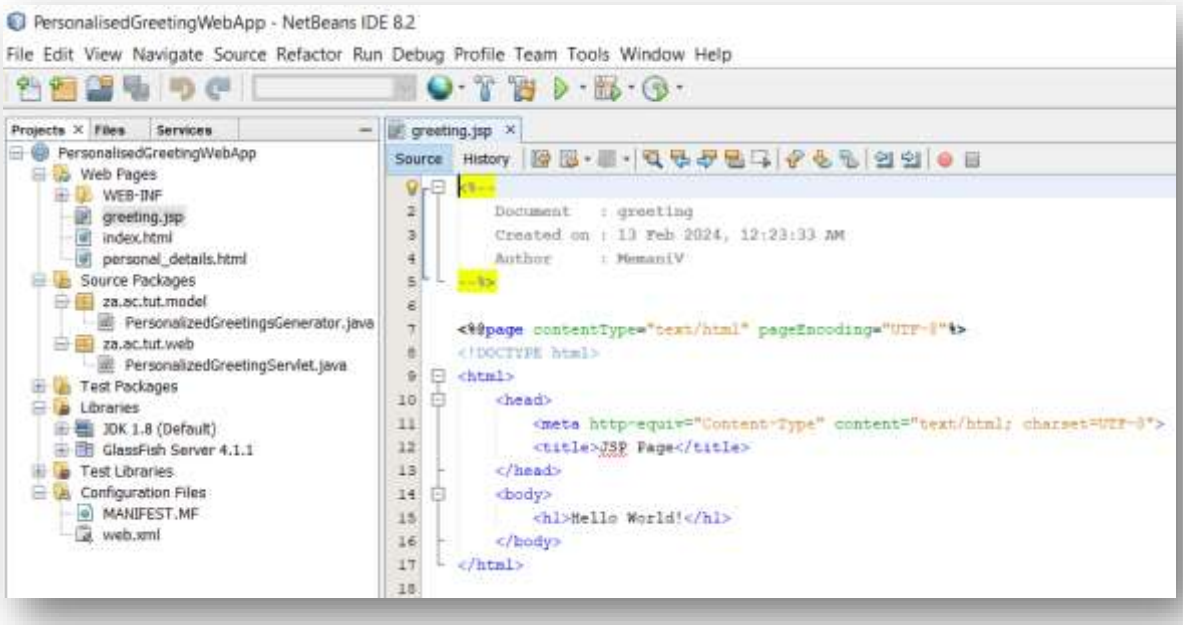
Right-click on the project and select **New | JSP**



Name the file as **greeting**



Click **Finish**



Modify the file

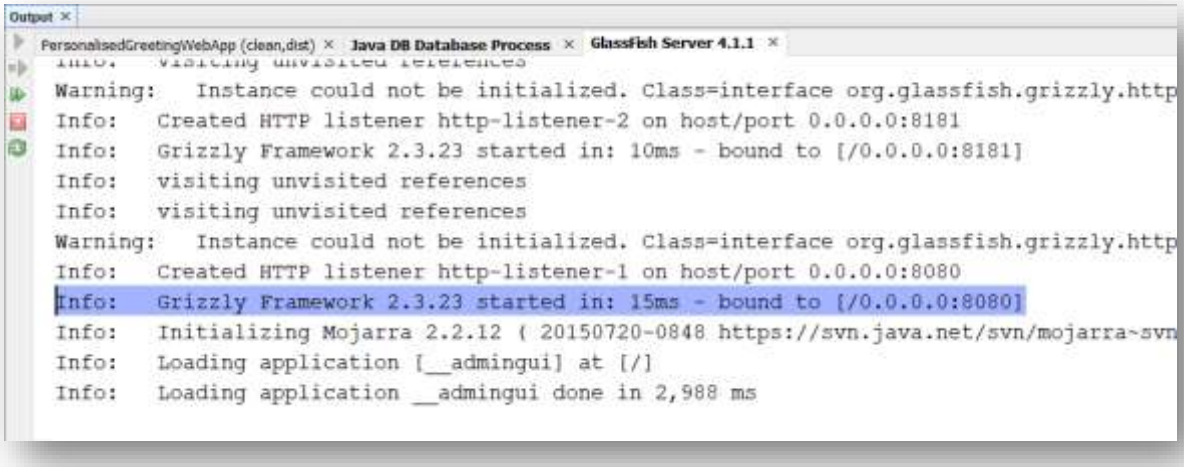
```jsp
<%--
    Document   : greeting
    Created on : 13 Feb 2024, 12:23:33 AM
    Author     : MemaniV
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Greetings Page</title>
    </head>
    <body>
        <hl>Greetings!</hl>
        <%
            String greeting = (String)request.getAttribute("greeting");
        %>
        <p>
            <%=greeting%>
        </p>
        <p>
            Click <a href="index.html">here</a> to go back to the main page.
        </p>
    </body>
</html>
```

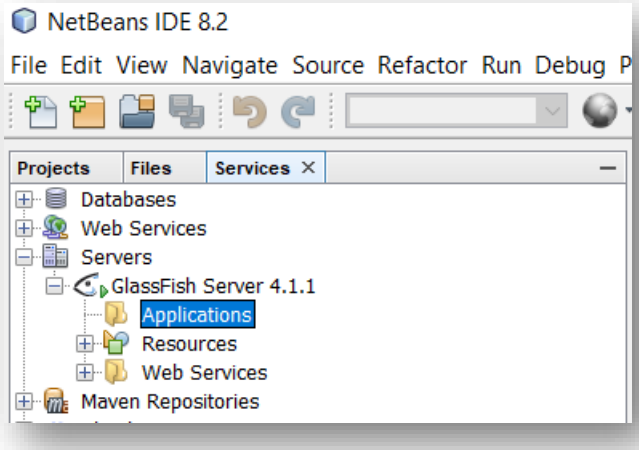
Right-click on the project and select **Clean and Build**.

```
Copying 4 files to C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\build\web
library-inclusion-in-archive:
library-inclusion-in-manifest:
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\build\empty
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\build\generated-sources\ap-source-output
Compiling 2 source files to C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\build\web\WEB-INF\classes
compile:
compile-jsps:
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\dist
Building jar: C:\Users\memaniv\Documents\NetBeansProjects\PersonalisedGreetingWebApp\dist\PersonalisedGreetingWebApp.war
do-dist:
dist:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Click on the **Services** tab, expand the **Servers**, right-click on **GlassFish**, and select **Start**.

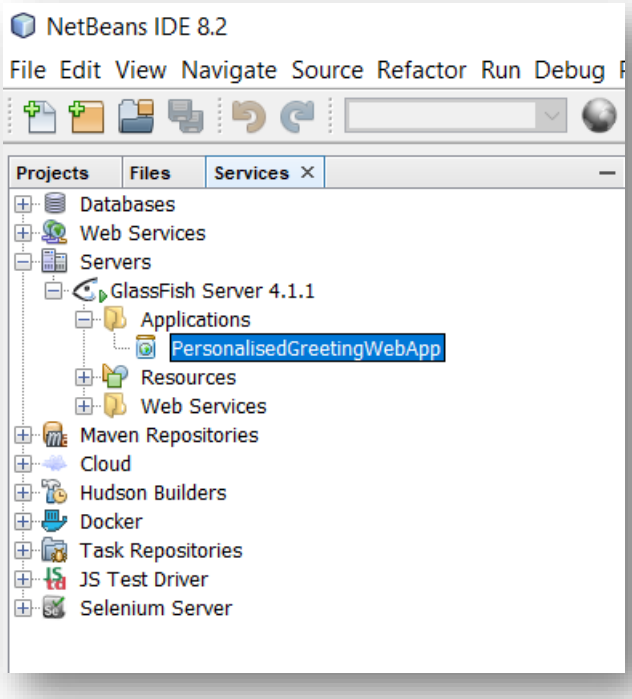


Expand **GlassFish** and the **Applications** folder.

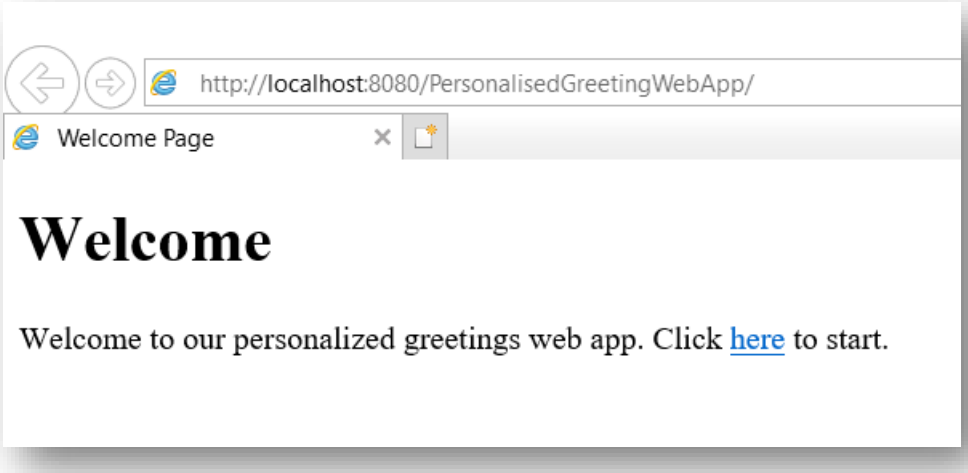


Go back to the **Projects** pane. Right-click on the project and select **Deploy**.

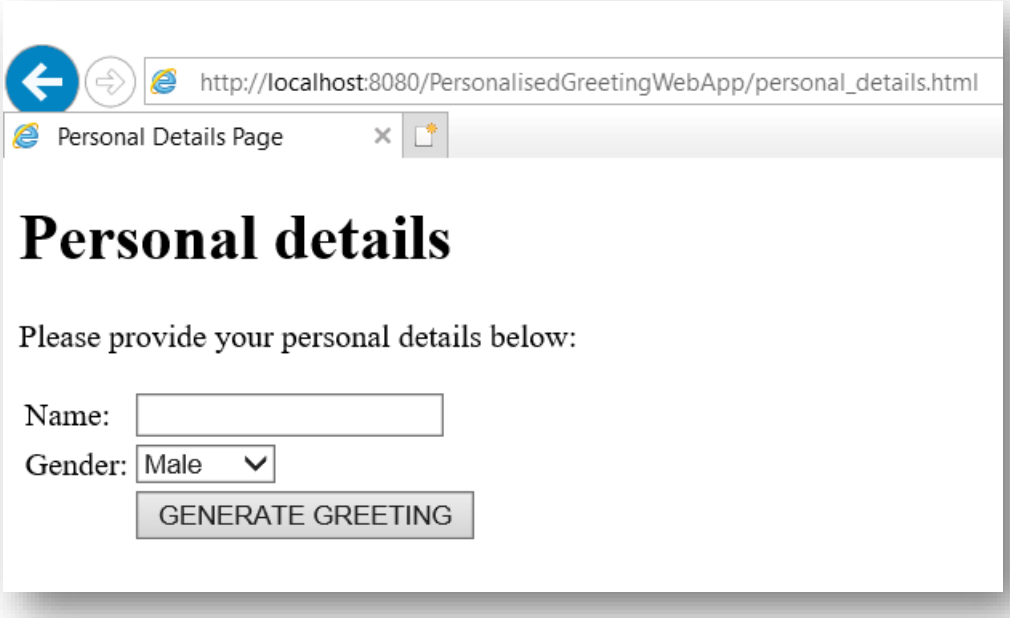
Go back to the **Services | Servers | Applications** panel



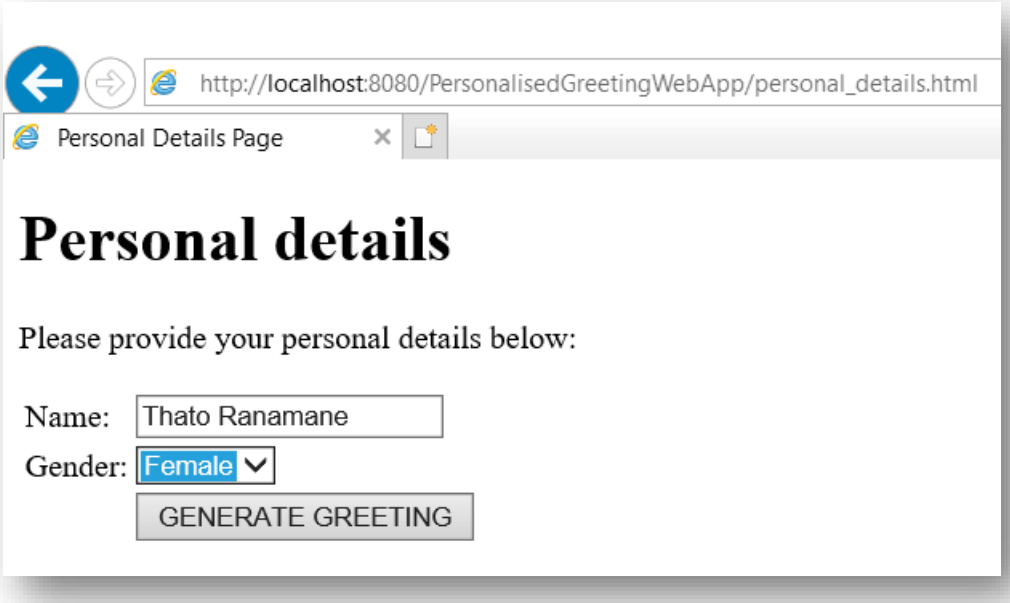
Right-click on the project and select **Open in Browser**



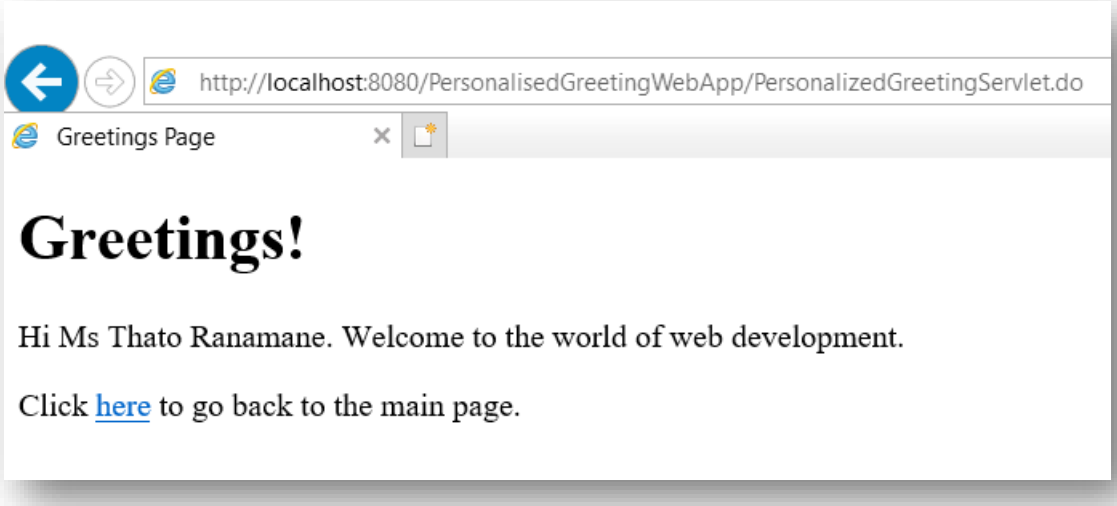
Click on the link



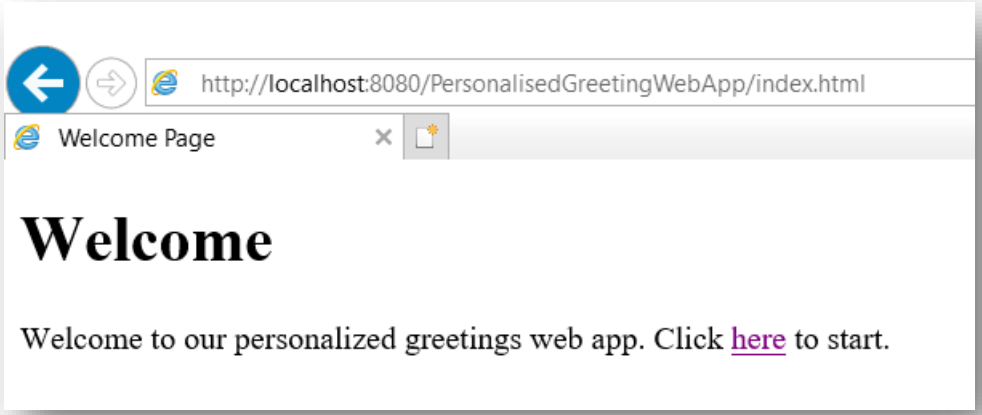
Complete form



Click on the button



Click on the link



1.3 Conclusion

In this chapter we managed to introduce the concept of MVC and demonstrated its application in a web development environment. We also referenced two JEE web components/standards, **JSPs** and **Servlets**. A detailed explanation of these two JEE specifications will follow in the next chapters.

Thank you for having taken time to go through the chapter. Please do the given DIYs. Enjoy the rest of the day and God bless you.

1.4 DIY

In the following activities you are expected to do the following to a given problem:

- Discuss the flow of your solution.
- Identify MVC components emanating from the discussion.
- Implement the design in software.

Activity 1

Create a web application that will determine and display the sum of two numbers given by a user.

Activity 2

Create a web application that will perform a rand to dollar conversion, and display the results. Assume that $1 is equal to R10.

Activity 3

Institution ABC needs a web application that can be used by prospective students determine whether they qualify for admission or not. The institution requires that students should have Mathematics and English as mandatory subjects, and obtain Admission Points Score (APS) not less than 20 based on the best six subjects done in grade 12. The institution uses the table below to calculate APS.

	Points per grade 12 achievement level
Languages	3
Science subjects	5

The specifications of the web applications are as follows:

- A prospective student must be allowed to enter the best six grade 12 subject results.
- The application must determine and display an outcome. The outcome must include APS and whether the prospective student qualifies for admission or not.

ABC approaches you to develop the web application for them.

Activity 4

Mujinga is an intern at a text analysis company called **TextSpectacles**. The company specializes in analysing text messages for customers. As an intern, Mujinga is given the responsibility of creating a web application that will help in the analysis of short text messages. Given a sentence, she is expected to determine, through the web application, the number of times each word in the sentence is occurring.

To make the process easier, it is assumed that the sentence under analysis will always contain letters and special characters, that is the commas (,) and periods (.). So, upon receiving a sentence, Mujinga is expected to remove the special characters from the text, get the individual words making up the sentence, change all the words to lowercases, and store them in a list.

The web application must then determine how many times each word in the list is occurring. The outcome of this determination must be displayed on the screen. As an established programmer at TextSpectacles, Mujinga approaches you with the request to help her accomplish the required task (unfortunately you can't say no).

Activity 5

Puleng has a younger brother, Sipho whom she wants to teach unit conversions. She's interested in two sets of conversions, namely **mass** and **length**.

Mass conversions

Below are the mass conversions of interest:

- Grams to kilograms
- Milligrams to grams
- Grams to milligrams

Length conversions

Below are the length conversions of interest:

- Millimeters to centimeters
- Centimeters to meters
- Meters to millimeters

Puleng approaches you to create such a web application for her, at a reasonable fee (It's nice being a programmer!!!).

Activity 6

Zinhle is a founder of a software development company called ***WeDoWebApps***. The company specializes in developing web applications for customers. Zinhle's company is currently working on a project that deals with message encryption. The customer wants a web application that will allow its employees to send plain text messages to the server and get back encrypted forms of the messages. The client explicitly ask for the usage of the Caeser Encryption Technique (CET).

Encryption is a data protection mechanism utilized to make readable data unreadable. The CET works as follows:

- All the letters in the plain text are first converted to lowercase letters.
- Each lowercase letter is replaced with a letter that is three spaces away from it in the alphabet. This means an **'a'** is replaced with a **'d'**, a **'b'** with an **'e'**, a **'c'** with an **'f'**, a **'w'** with a **'z'**, an **'x'** with an **'a'**, a **'y'** with a **'b'**, and a **'z'** with a **'c'**.

Example

Say you have the following message: **Hello Word!**

The message will first be converted to lowercases: **hello world!**

Subsequently each letter will be replaced by a letter that is three spaces away in the alphabet. So you will end up with the following unreadable message: **khoor zruog!**

The encrypted message (unreadable message) and the plain message must be stored in a local database to the server. The persisted or stored data must include a

Normally, the client side and the server side are located in different machines. But for the purposes of this exercise, both the client and the server will be on the same local machine.

You are an intern at WeDoWebApp. Ms Baloyi, a Senior Developer at ***WeDoWebApps*** gives the responsibilty of creating the required web application.