FPT Education
FPT UNIVERSITY

Java

Spring MVC

Objectives

- Spring MVC Basics
- Spring MVC Framework
- Controller/Model/View
- Spring Interceptor
- Spring Validator

The MVC pattern

- MVC pattern breaks an application into three parts:
  - Model: The domain object model / service layer
  - View: Template code / markup
  - Controller: Presentation logic / action classes

- MVC defines interaction between components to promote separation of concerns and loose coupling
  - Each file has one responsibility
  - Enables division of labor between programmers and designers
  - Facilitates unit testing
  - Easier to understand, change and debug

Advantages of MVC Pattern

Separation of application logic and web design through the *MVC pattern*

- Integration with template languages
- Some MVC frameworks provide built-in components

- Other advantages include
  - Form validation
  - Error handling
  - Request parameter type conversion
  - Internationalization
  - IDE integration

Exploring Spring's Web MVC Framework

- The Spring Web MVC framework is built on generic Servlet known as a DispatcherServlet class (Front Controller).
- The DispatcherServlet class sends the request to the handlers with configurable handler mappings, theme resolution, locale and view resolution along with file uploading.
- The handleRequest(request, response) method is given by the default handler called Controller interface.
- The application controller implementation classes of the controller interface are as follows:
  - AbstractController
  - AbstractCommandController
  - SimpleFormController.

Spring MVC Features

- Powerful configuration of both framework and application classes.
- Separation of roles.
- Flexibility in choosing subclasses.
- Model transfer flexibility.
- No need of duplication of code.
- Specific validation and binding.
- Specific local and theme resolution.
- Facility of JSP form tag library.

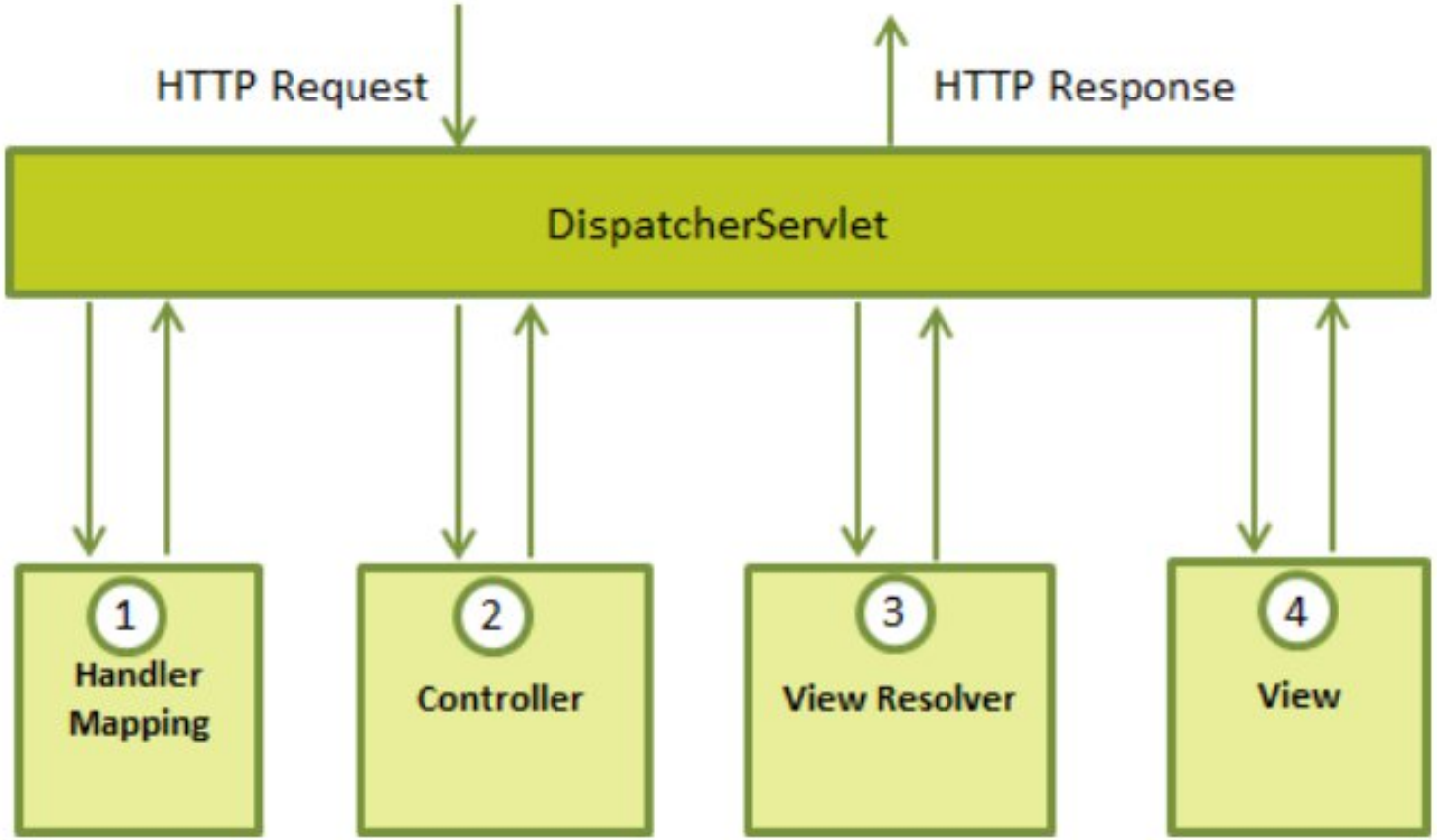
Spring Web MVC

- Spring Web MVC is the original web framework built on the Servlet API and has been included in the Spring Framework from the very beginning.
- The formal name, "Spring Web MVC," comes from the name of its source module (spring-webmvc), but it is more commonly known as "Spring MVC".

Spring Web MVC

- DispatcherServlet
- Filters
- Annotated Controllers
- Functional Endpoints
- URI Links
- Asynchronous Requests
- CORS
- Error Responses
- Web Security
- HTTP Caching
- View Technologies
- MVC Config
- HTTP/2

DispatcherServlet



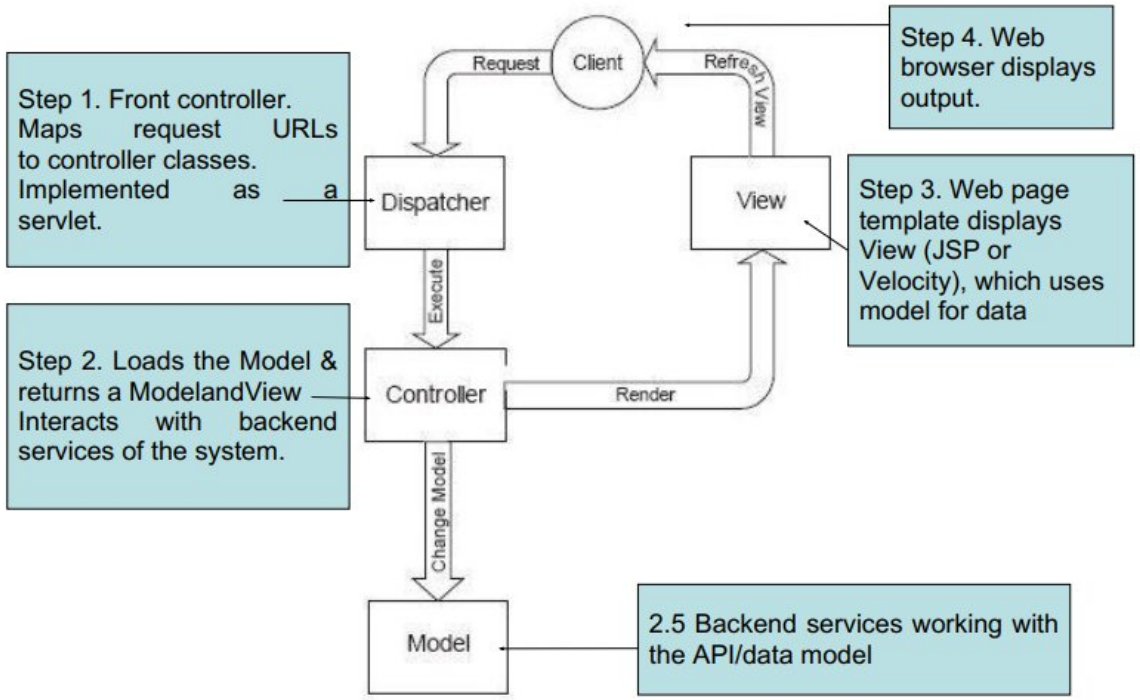
DispatcherServlet

Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet*

- After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
- The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
- The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
- Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

DispatcherServlet

- HandlerMapping, Controller, and ViewResolver are parts of *WebApplicationContext* which is an extension of the plain *ApplicationContext* with some extra features necessary for Web applications.



DispatcherServlet

- The DispatcherServlet class is important part of spring Web MVC framework.
- It is used for dispatching the request to application controllers.
- The DispatcherServlet class is configured in web.xml file of a web application.
- Map the request using Uniform Resource Locator (URL) mapping in the same *web.xml* file to handle any request

org.springframework.web.servlet

Class DispatcherServlet

java.lang.Object
- javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet
    - org.springframework.web.servlet.HttpServletBean
      - org.springframework.web.servlet.FrameworkServlet
        - org.springframework.web.servlet.DispatcherServlet

HadlerMapping/Controller/ViewResolver

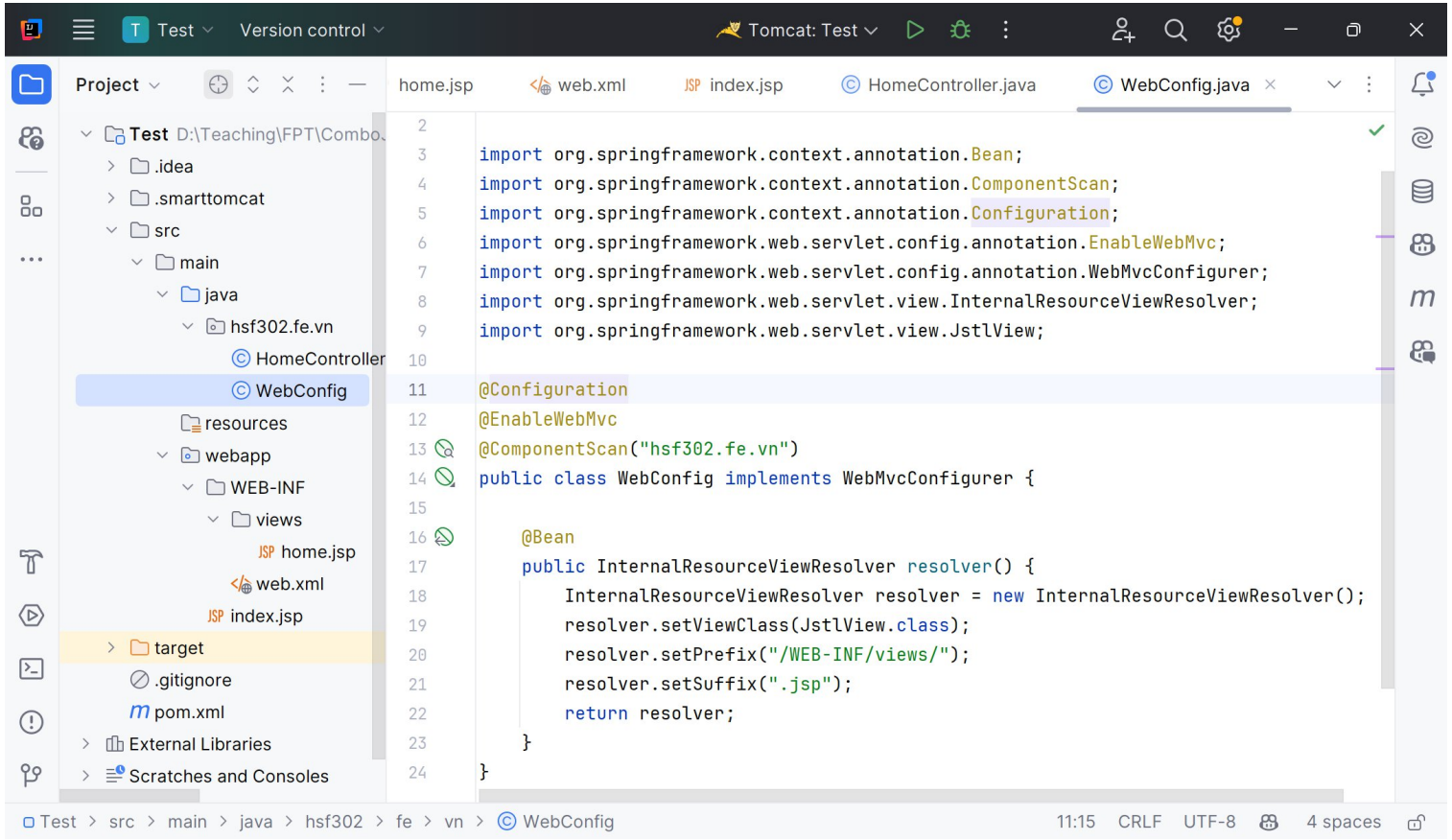
- Handler mapping: Manages the execution of controllers, provided they match the specified criteria.
- Controller: It handles the client's request.
- View resolver: Resolves view names to view used by the DispatcherServlet.
  - The mapping between the Logical name and the Physical View Location is taken care by the View Resolver object.
  - Spring comes with a set of Built-In Spring Resolvers.
  - We can write Custom View Resolvers by implementing the org.springframework.web.servlet.ViewResolver interface

View Resolver

- **InternalResourceViewResolver**
- **ContentNegotiatingViewResolver**
- BeanNameViewResolver
- FreeMarkerViewResolver
- JasperReportsViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- VelocityLayoutViewResolver
- VelocityViewResolver
- XmlViewResolver
- XsltViewResolver

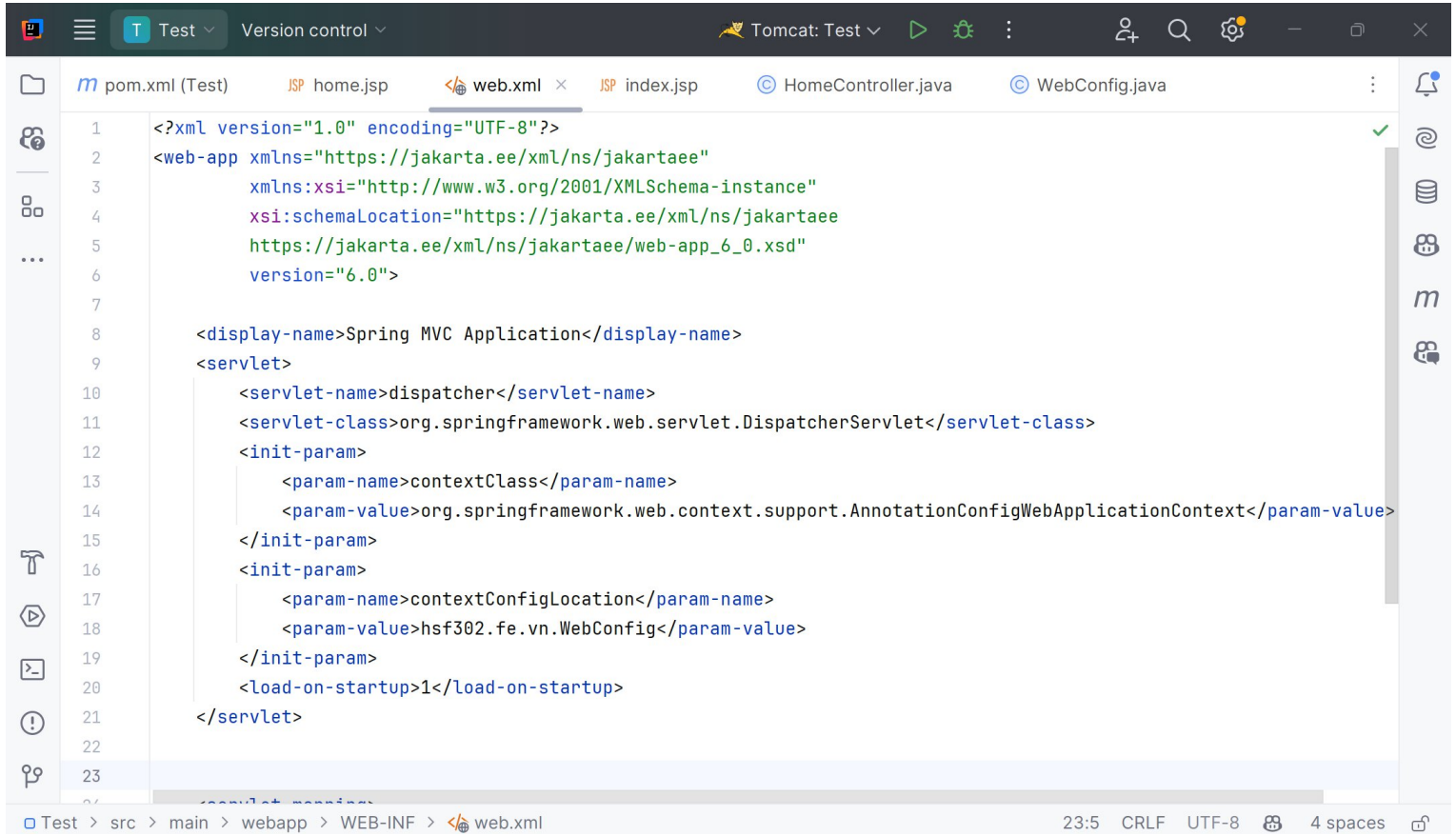
View Resolver

- InternalResourceViewResolver



Configure DispatcherServlet in web.xml

- Spring application will automatically be searched for and loaded by Spring for us.



Overriding *DispatcherServlet* defaults

Following 2 basic ones:

- *InternalResourceViewResolver (for jsp, css, images etc)*
- *ContentNegotiatingViewResolver (for ContentType response, useful for REST APIs)*

- If the Controller returns "**index**", InternalResourceViewResolver tries to find file as view **/WEB-INF/views/index.jsp**

Controllers

- The Controllers are central components of MVC
- Simply add **@Controller** annotation to a class
- Use **@RequestMapping** to map methods to url

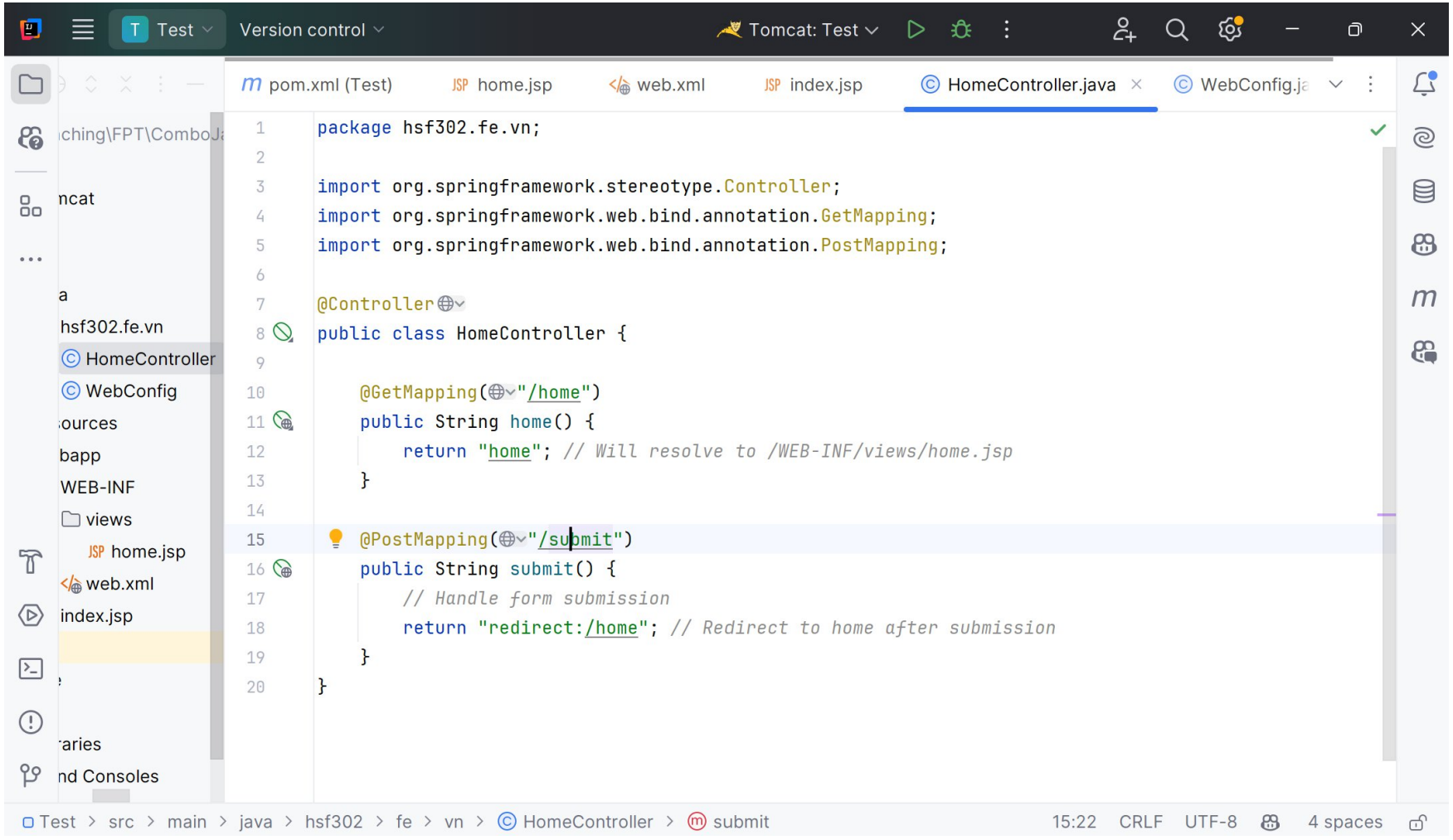
```
@Controller
public class BaseController {

        @RequestMapping(value="/")
        public String welcome(ModelMap model) {

                model.addAttribute("message", "Whaddap!!");

                //Spring uses InternalResourceViewResolver and return back index.jsp
                return "index";
        }
...
}
```

Controllers

Controller Annotation

- The @Controller annotation defines the class as a Spring MVC controller.
- The @RequestMapping annotation is used to map URLs like '/hello' onto an entire class or a particular handler method.
- @RequestMapping(method = RequestMethod.GET) is used to declare the printHello() method as the controller's default service method to handle HTTP GET request.

Example


Test
Version control
Tomcat: Test
pom.xml (Test)
home.jsp
web.xml
index.jsp
HomeController.java
WebConfig.j
hsf302.fe.vn
HomeController
WebConfig
WEB-INF
views
home.jsp
web.xml
index.jsp
1 package hsf302.fe.vn;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PostMapping;
6
7 @Controller
8 public class HomeController {
9
10 @GetMapping("/home")
11 public String home() {
12 return "home"; // Will resolve to /WEB-INF/views/home.jsp
13 }
14
15 @PostMapping("/submit")
16 public String submit() {
17 // Handle form submission
18 return "redirect:/home"; // Redirect to home after submission
19 }
20 }
Test > src > main > java > hsf302 > fe > vn > HomeController > submit
15:22 CRLF UTF-8 4 spaces

Controller arguments

- **@RequestParam, @PathVariable**
- **BindingResult** – to access the results of binding and validation (POJO – arbitrary java object that gets populated with request values **@Valid** – to enforce validation of the POJO)
- **Model, ModelMap, Map<String, ?>** access to the model object
- Raw **HttpServletRequest**, **HttpServletRespone**, **HttpSession**
- **@RequestHeader, @RequestBody**

More detailed Url Mapping

- @RequestMapping also accepts the following parameters:
  - method (GET/POST/PUT/DELETE...)
  - produces (mimeType)
  - consumes (mimeType)
  - params
  - headers

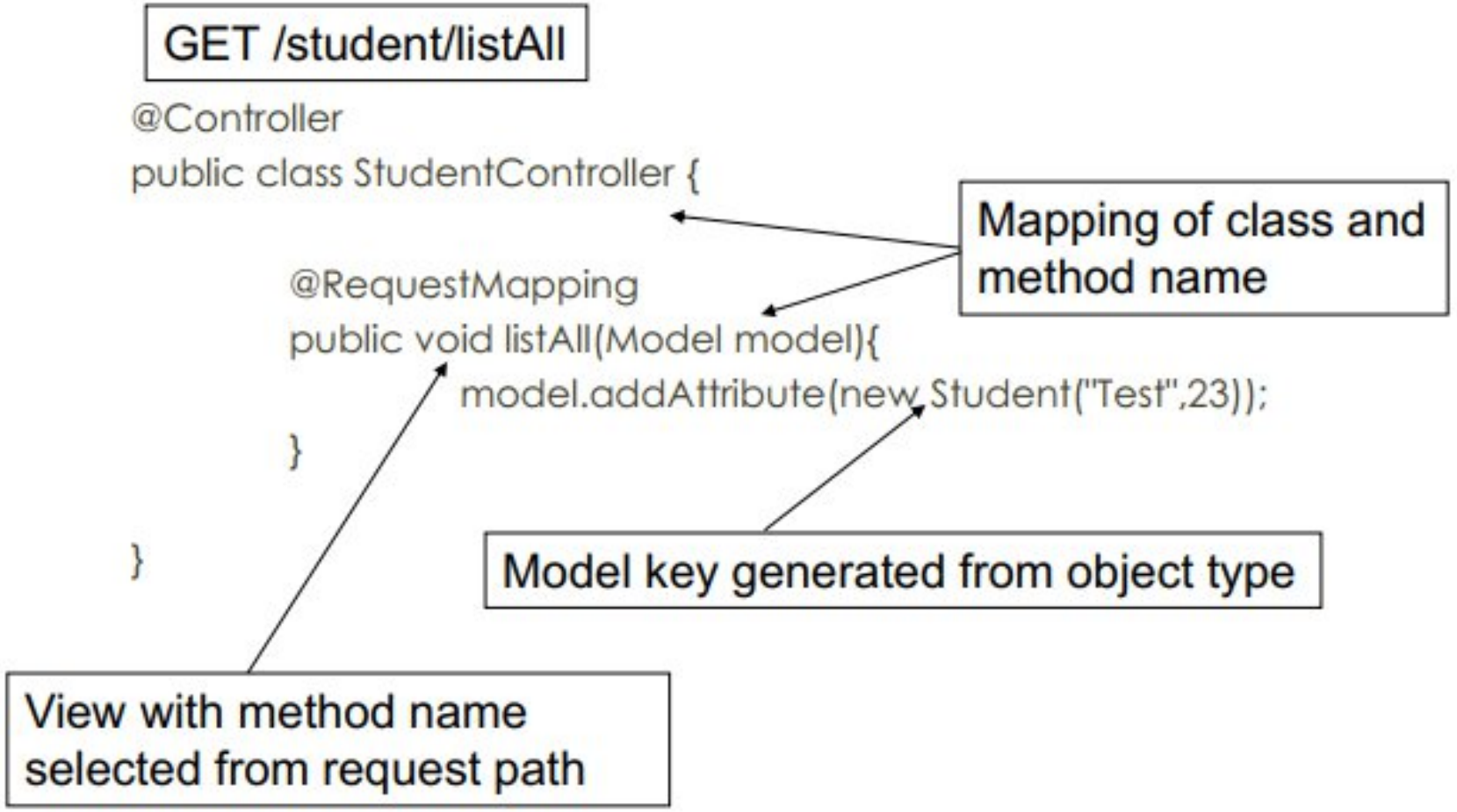
```
...
     @RequestMapping(value="/", method = RequestMethod.GET, produces = "text/html")
     public String welcome(ModelMap model) {

          model.addAttribute("message", "Whaddap!!");

          //Spring uses InternalResourceViewResolver and return back index.jsp
          return "index";
     }
...
}
```

Url Templates in Controllers

- **@PathVariable -** to map variables in URL paths
- path variables can also be Regular Expressions
- Can also do as follows /message/*/user/{name}
- Can also use comma-separated URL parameters (also called Matrix-Variables)
  - To do this, make setRemoveSemicolonContent=false for *RequestMappingHandlerMapping*

```
// GET = /message/lars;friends=bob,rob,andy
@RequestMapping(value="/message/{name}", method = RequestMethod.GET)
public String welcome( @PathVariable String name,  @MatrixVariable String[] friends, ModelMap model) {

      model.addAttribute("message", "Hello " + name + " from " + friends[0] + " & " + friends[1]);
      //Spring uses InternalResourceViewResolver and return back index.jsp
      return "index";
}
```

Convention over Configuration



Convention over Configuration

- Mapping By Convention

```
@Controller
public class StudentController {

    URL → /student/showList ,  View → showList.jsp
    @RequestMapping
    public void showList(Model model){}


    URL → /student/getStudent,  View → getStudent.jsp
    @RequestMapping
    public Student getStudent(){
        ...
        return stu;
    }
}
```

Model

- Controllers and view share a Java object referred as model, (‘M’ in MVC)
- A model can be of the type *Model* or can be a *Map* that can represent the model.
- The view uses this to display dynamic data that has been given by the controller

```
// Controller
@RequestMapping(value = "/{name}", method = RequestMethod.GET)
        public String welcome(@PathVariable String name, ModelMap model) {

        model.addAttribute("message", "Hello " + name);

        return "index";
}
```

- In View:

```
<html>
        body><h1>${message}</h1></body>
</html>
```

ModelAndView object in Spring MVC

- ModelAndView is an object that holds both the model and view. The handler returns the ModelAndView object and DispatcherServlet resolves the view using View Resolvers and View.

- The View is an object which contains view name in the form of the String and model is a map to add multiple objects.

```
ModelAndView model = new ModelAndView("employeeDetails");
model.addObject("employeeObj", new EmployeeBean(123));
model.addObject("msg", "Employee information.");
return model;
```

@ModelAttribute from Controller

- You can also use ***@ModelAttribute*** in controller to directly load URL value into the model
- A Model can represent objects that can be retrieved from database or files as well
- Model should not have logic, rather the controller should get the model and “transform” the model based on the request, while sending it to the View

View

- Spring MVC integrates with many view technologies:
  - JSP
  - Thymeleaf
  - Velocity
  - Freemarker
  - JasperReports
- Values sent to controller with POST or GET as usual
- Values made available to the view by the controller

Spring's form tag library

- Binding-aware JSP tags for handling form elements
- Integrated with Spring MVC to give the tags access to the model object and reference data
- Comes from spring-webmvc.jar
- Add the following to make the tags available:
- `<%@ taglib prefix="form"`
  `uri="http://www.springframework.org/tags/form" %>`

Spring's form tag library

Tag	Description
`form:form`	Generates the HTML `<form>` tag. It has the name attribute that specifies the command object that the inner tags should bind to.
`form:input`	Represents the HTML input text tag.
`form:password`	Represents the HTML input password tag.
`form:radiobutton`	Represents the HTML input radio button tag.
`form:checkbox`	Represents the HTML input checkbox tag.
`form:select`	Represents the HTML select list tag.
`form:options`	Represents the HTML options tag.
`form:errors`	Represents the HTML span tag. It also generates span tag from the error created as a result of validations.

The attributes: *path* & *modelAttribute/commandName*

- commandName/modelAttribute: name of a variable in the request scope or session scope that contaims the information about this form, it should be a been.
- path: name of a bean property that should be accessed in order to pass the information to from and to the controller.

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
<title>Spring MVC Form Handling</title>
</head>

<body>
  <h2>Employee</h2>
  <form:form method="POST" action="addEmployee" modelAttribute="employee">
    Id: <form:input path="id" />
    <br/>
    Name: <form:input path="name" />
    <input type="submit" value="Submit" />
  </form:form>
</body>
</html>
```

The *form* tag

- Renders a form tag and exposes the binding to inner tags
- You can specify any HTML attributes that are valid for an HTML form
- You can also tell it what the form backing object is (it uses 'command' by default).

```
@RequestMapping("/person/add")
public String addPerson(Model model) {
        Person person = new Person();
        model.addAttribute(person);
        return "addPerson";
}
```

```
<form:form method="get" commandName="person">
        <form:input path="name" />
</form:form>
```

```
<form method="get">
        <input type="text" name="name" />
</form>
```

The *input* tag

- Renders an HTML input tag with a type of ‘text’
- You can specify any HTML attributes valid for an HTML input
- You bind it to your model object by specifying the path relative to the backing object

```
@RequestMapping("/person/add")
public String addPerson(Model model) {
        Person person = new Person();
        person.setName(”Spencer”);
        model.addAttribute(person);
        return "addPerson";
}
```

```
<form:form method=”get” commandName=”person”>
        <form:input path=”name” />
</form:form>
```

```
<form method=”get”>
        <input type=”text” name=”name” value=”Spencer” />
</form>
```

The *checkbox* tag

- Renders an HTML input tag with a type of ‘checkbox’

```
public class Person {
        private boolean admin;
        private String[] languages;
}
```

```
<form:form commandName="person">
    <form:checkbox path="admin" />
    <form:checkbox path="languages" value="Java" />
    <form:checkbox path="languages" value="Scala" />
</form:form>
```

```
<form>
    <input type="checkbox" name="admin" value="true" />
    <input type="checkbox" name="languages" value="Java" />
    <input type="checkbox" name="languages" value="Scala" />
</form>
```

The *checkboxes* tag

- Similar to *checkbox* tag, but creates multiple checkboxes instead of one

```
public class Person {
        private boolean admin;
        private String[] favoriteLanguages;
        private List<String> allLanguages;
}
```

```
<form:form commandName="person">
    <form:checkbox path="admin" />
    <form:checkboxes path="favoriteLanguages" items="${allLanguages}" />
</form:form>
```

```
<form>
    <input type="checkbox" name="admin" value="true" />
    <input type="checkbox" name=" favoriteLanguages" value="Java" />
    <input type="checkbox" name=" favoriteLanguages" value="Scala" />
</form>
```

The *password* tag

- Renders an HTML input tag with a type of ‘password’

```
<form:form>
        <form:input path=”username” />
        <form:password path=”password” />
</form:form>
```

```
<form>
        <input type=”text” name=”username” />
        <input type=”password” name=”password” />
</form>
```

The *select* tag

- Renders an HTML select tag
- It can figure out whether multiple selections should be allowed
- You can bind options using this tag, as well as by nesting *option* and *options* tags

```
<form:select path="favoriteLanguage" items="${allLanguages}" />
```

```
<select name="favoriteLanguage">
        <option value="Java">Java</option>
        <option value="Scala">Scala</option>
</form>
```

The *option/options* tags

- Renders an HTML option (or multiple options)
- Nested within a select tag
- Renders ‘selected’ based on the value bound to the select tag

```
<form:select path=”favoriteLanguage”>
        <form:option value=”3” label=”.NET” />
        <form:options items=”${languages}” itemValue=”id” itemLabel=”name”
/>
</form:select>
```

```
<select name=”favoriteLanguage”>
        <option value=”.NET”>.NET</option>
        <option value=”Java”>Java</option>
        <option value=”Scala”>Scala</option>
</form>
```

The *errors* tag

- Renders an HTML span tag, containing errors for given fields
- You specify which fields to show errors for by specifying a path
  - path="name" – Would display errors for name field.
  - path="*" – Would display all errors

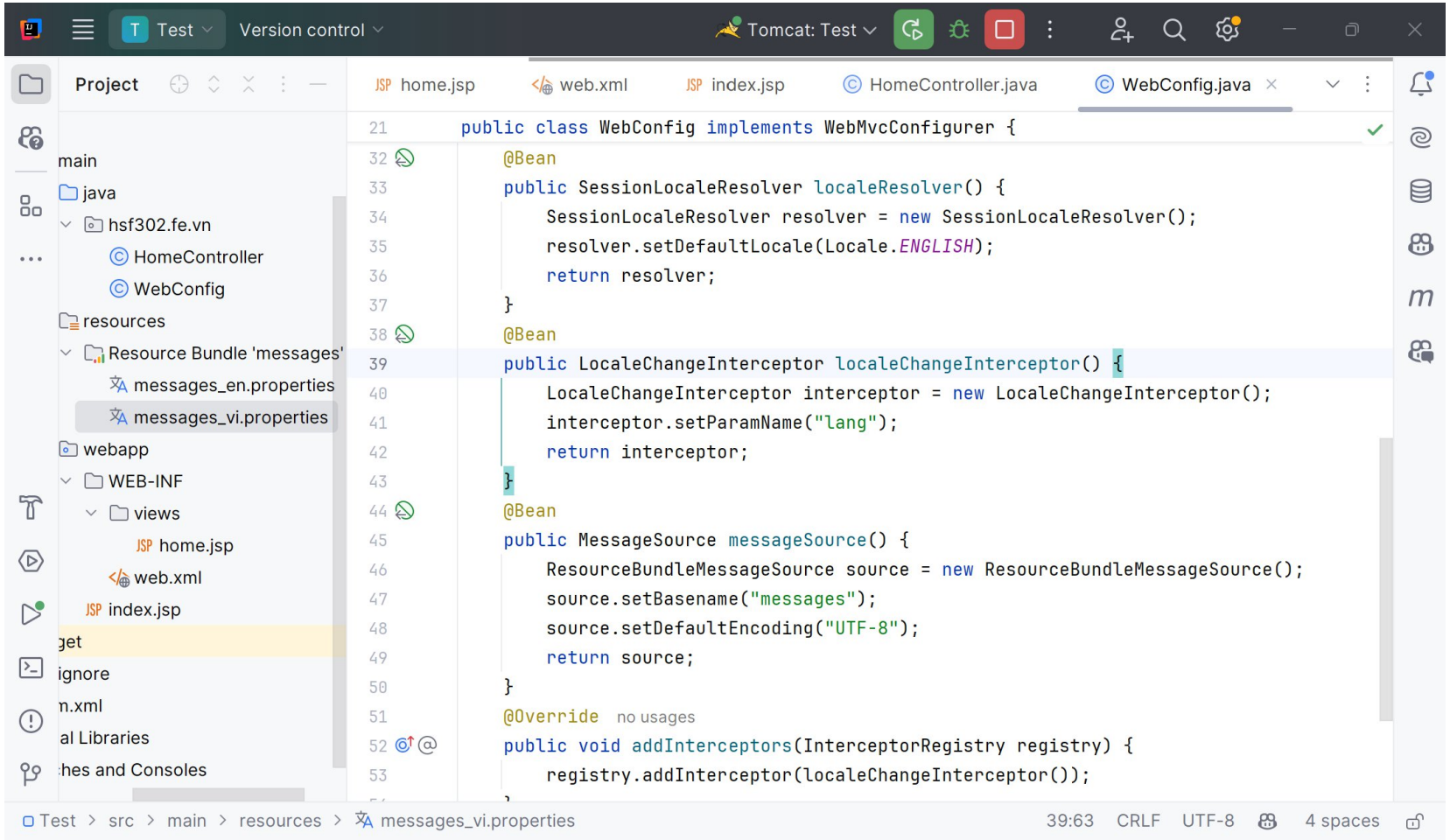
```
<form:errors path="*" cssClass="errorBox" />
<form:errors path="name" cssClass="specificErrorBox" />
```

```
<span name="*.errors" class="errorBox">Name is required.</span>
<span name="name.errors" class="specificErrorBox">Name is required.</span>
```

Spring Exception

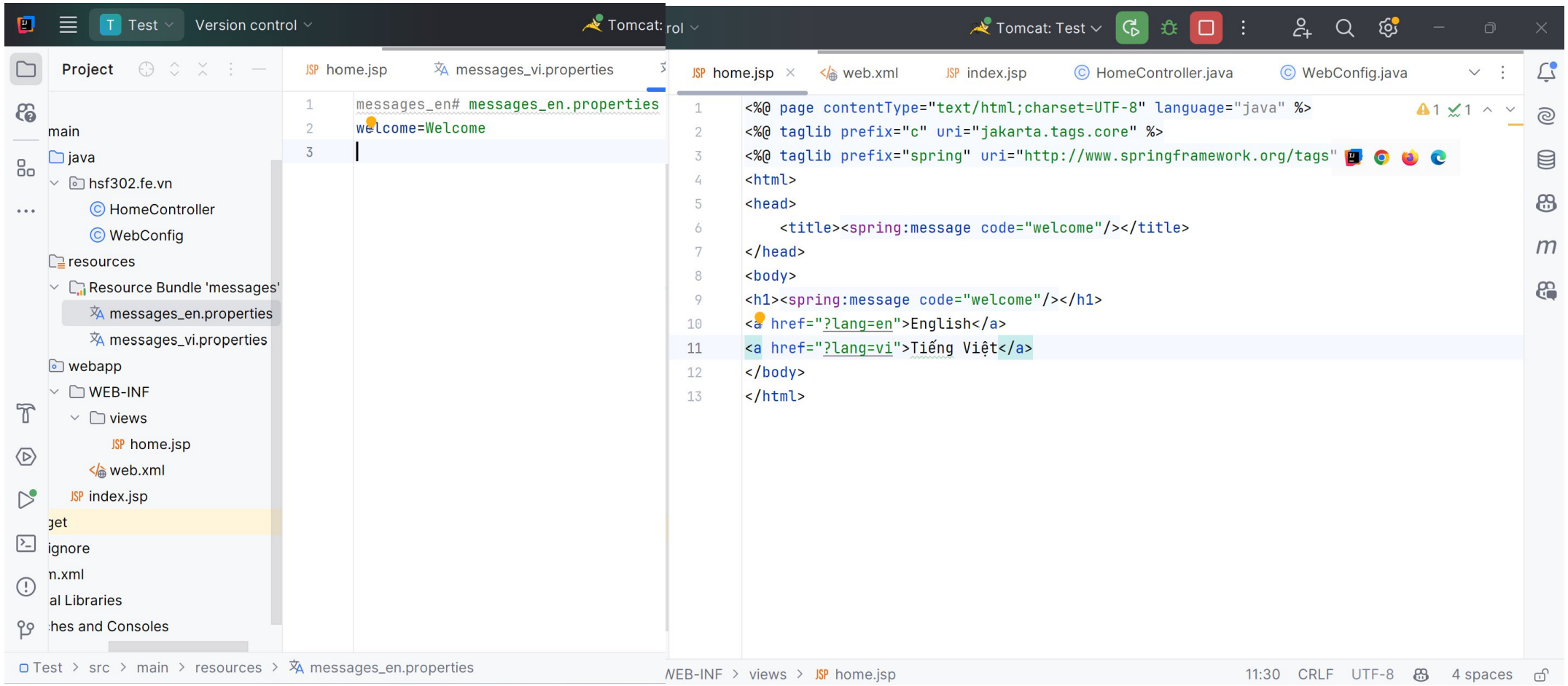
- @ExceptionHandler only handles exception getting raised from the controller where it is defined.
- It will not handle exceptions getting raised from other controllers. @ControllerAdvice annotation solves this problem.
- @ControllerAdvice annotation is used to define @ExceptionHandler,@InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods.

Locales




```
21  public class WebConfig implements WebMvcConfigurer {
32      @Bean
33      public SessionLocaleResolver localeResolver() {
34          SessionLocaleResolver resolver = new SessionLocaleResolver();
35          resolver.setDefaultLocale(Locale.ENGLISH);
36          return resolver;
37      }
38      @Bean
39      public LocaleChangeInterceptor localeChangeInterceptor() {
40          LocaleChangeInterceptor interceptor = new LocaleChangeInterceptor();
41          interceptor.setParamName("lang");
42          return interceptor;
43      }
44      @Bean
45      public MessageSource messageSource() {
46          ResourceBundleMessageSource source = new ResourceBundleMessageSource();
47          source.setBasename("messages");
48          source.setDefaultEncoding("UTF-8");
49          return source;
50      }
51      @Override  no usages
52      public void addInterceptors(InterceptorRegistry registry) {
53          registry.addInterceptor(localeChangeInterceptor());
```

Locales


Test
Version control
Tomcat: Test
Project
home.jsp
messages_vi.properties
web.xml
index.jsp
HomeController.java
WebConfig.java
messages_en# messages_en.properties
welcome=Welcome
main
java
hsf302.fe.vn
HomeController
WebConfig
resources
Resource Bundle 'messages'
messages_en.properties
messages_vi.properties
webapp
WEB-INF
views
home.jsp
web.xml
index.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"
<html>
<head>
<title><spring:message code="welcome"/></title>
</head>
<body>
<h1><spring:message code="welcome"/></h1>
English
Tiếng Việt
</body>
</html>
Test > src > main > resources > messages_en.properties
WEB-INF > views > home.jsp
11:30
CRLF
UTF-8
4 spaces

Spring Validation

- Spring provides a simplified set of APIs and supporting classes for validating domain objects.
- Spring features a Validator interface that you can use to validate objects. The Validator interface works using an Errors object so that while validating, validators can report validation failures to the Errors object.
- *Validation using Spring's Validator interface*

```
public interface Validator {

    /** Can this instances of the supplied clazz */
    boolean supports(Class<?> clazz);

    /**
    * Validate the supplied target object, which must be
    * @param target the object that is to be validated
    * @param errors contextual state about the validation process
    */
    void validate(Object target, Errors errors);
}
```

Spring Validation

- Standard Constraints

Annotation	Type	Description
`@Min(10)`	Number	must be higher or equal
`@Max(10)`	Number	must be lower or equal
`@AssertTrue`	Boolean	must be true, null is valid
`@AssertFalse`	Boolean	must be false, null is valid
`@NotNull`	any	must not be null
`@NotEmpty`	String / Collection's	must be not null or empty
`@NotBlank`	String	`@NotEmpty` and whitespaces ignored
`@Size(min,max)`	String / Collection's	must be between boundaries
`@Past`	Date / Calendar	must be in the past
`@Future`	Date / Calendar	must be in the future
`@Pattern`	String	must math the regular expression

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpt
import org.springframework.format.annotation.Dat
@Phone: defined custom validator.

```
    @Size(min=2, max=30)
private String name;

@NotEmpty @Email
private String email;

@NotNull @Min(18) @Max(100)
private Integer age;

@NotNull
private Gender gender;

@DateTimeFormat(pattern="MM/dd/yyyy")
@NotNull @Past
private Date birthday;

@Phone
private String phone;
```

@Autowired

- @Autowired on Setter Methods
- @Autowired on Properties
- @Autowired on Constructors
- @Autowired on Constructors

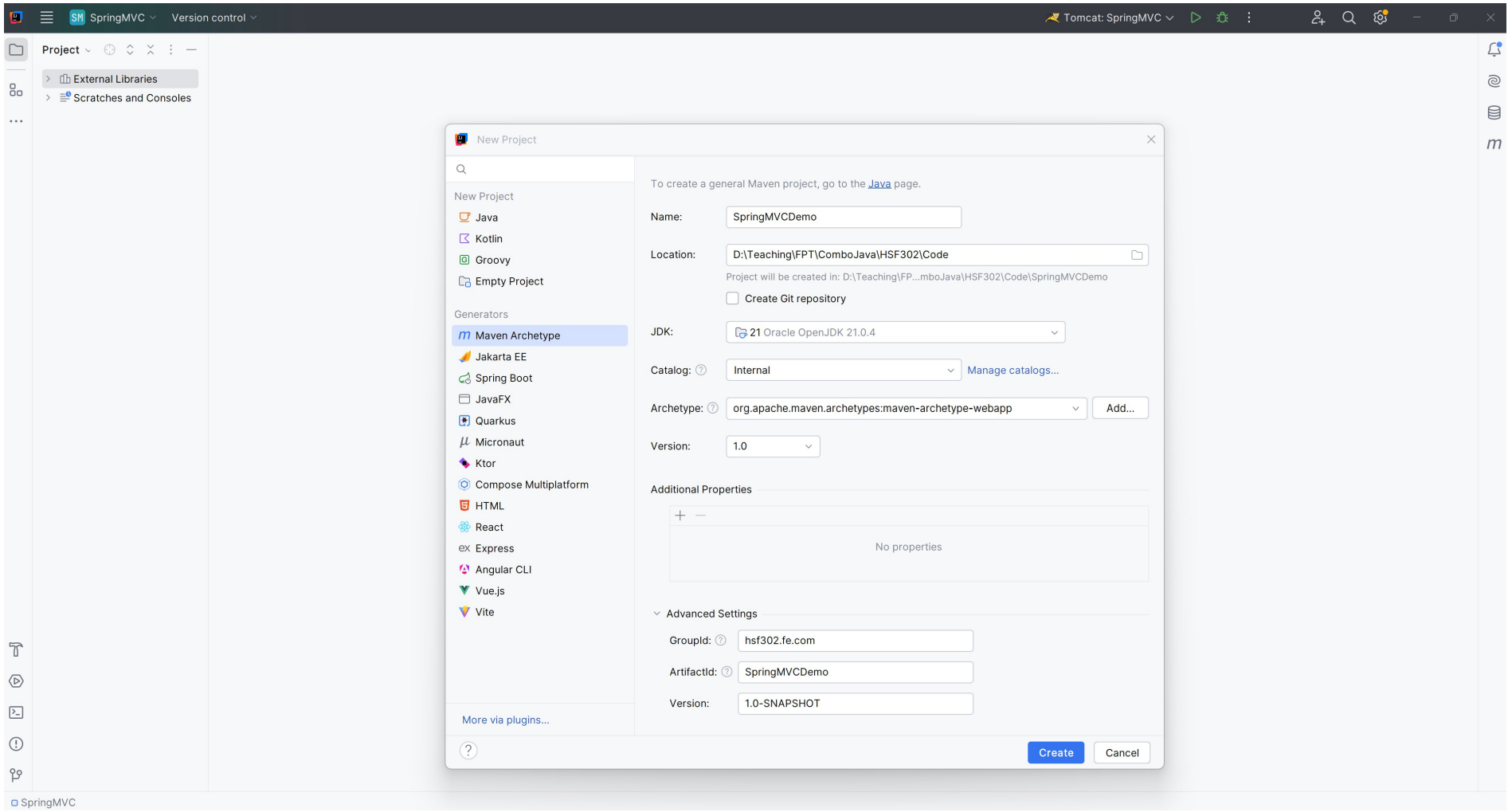
```java
public class MyController {

    private MyService myService;

    public MyController(MyService aService) { // constructor based injection
        this.myService = aService;
    }

    public void setMyService(MySerice aService) { // setter based injection
        this.myService = aService;
    }

    @Autowired
    public void setMyService(MyService aService) { // autowired by Spring
        this.myService = aService;
    }

    @RequestMapping("/blah")
    public String someAction()
    {
        // do something here
        myService.foo();

        return "someView";
    }
}
```

FPT Education
FPT UNIVERSITY

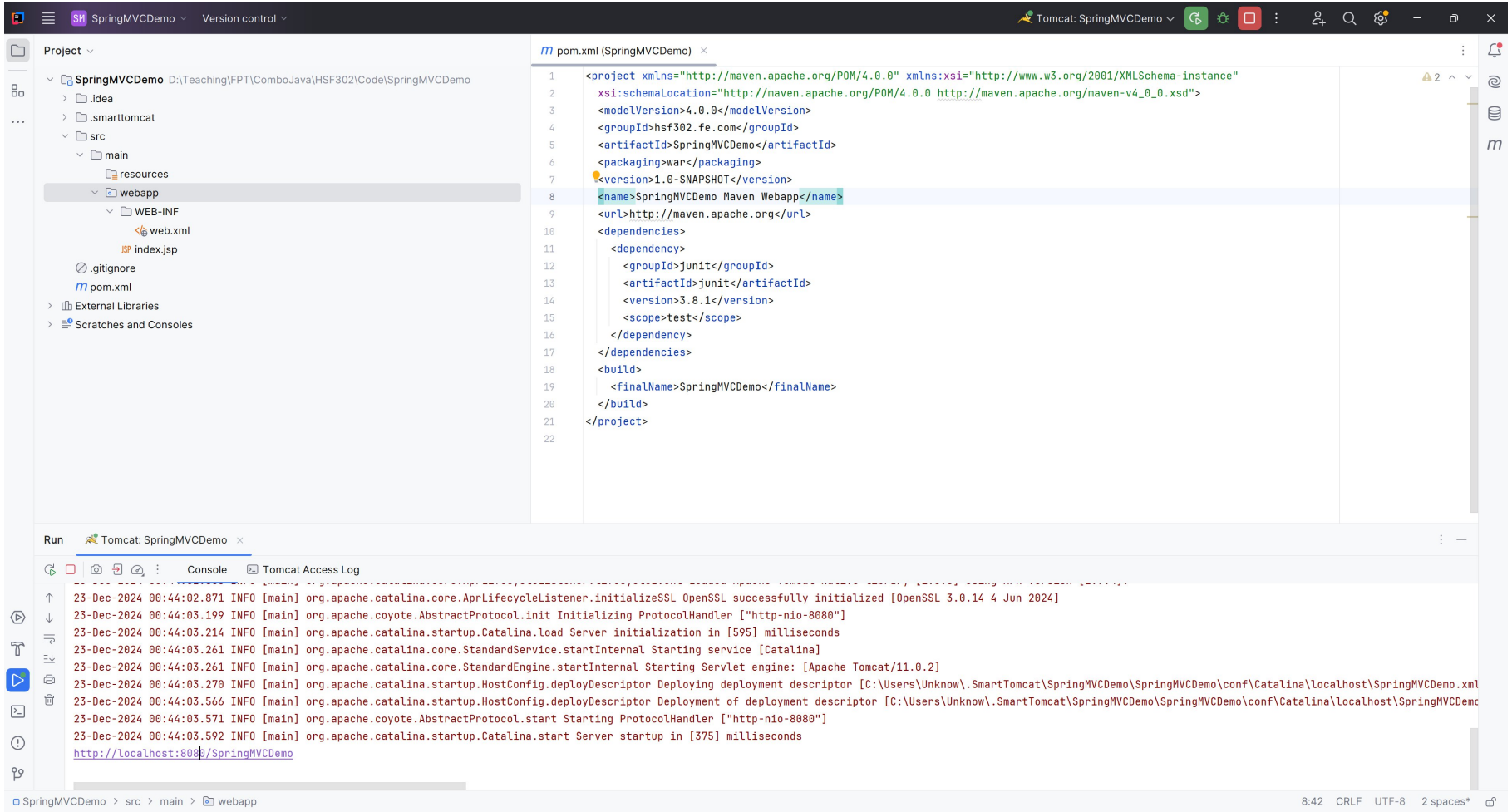
Java

Demo SpringMVC

Open IntelliJ, File | New | Maven Project



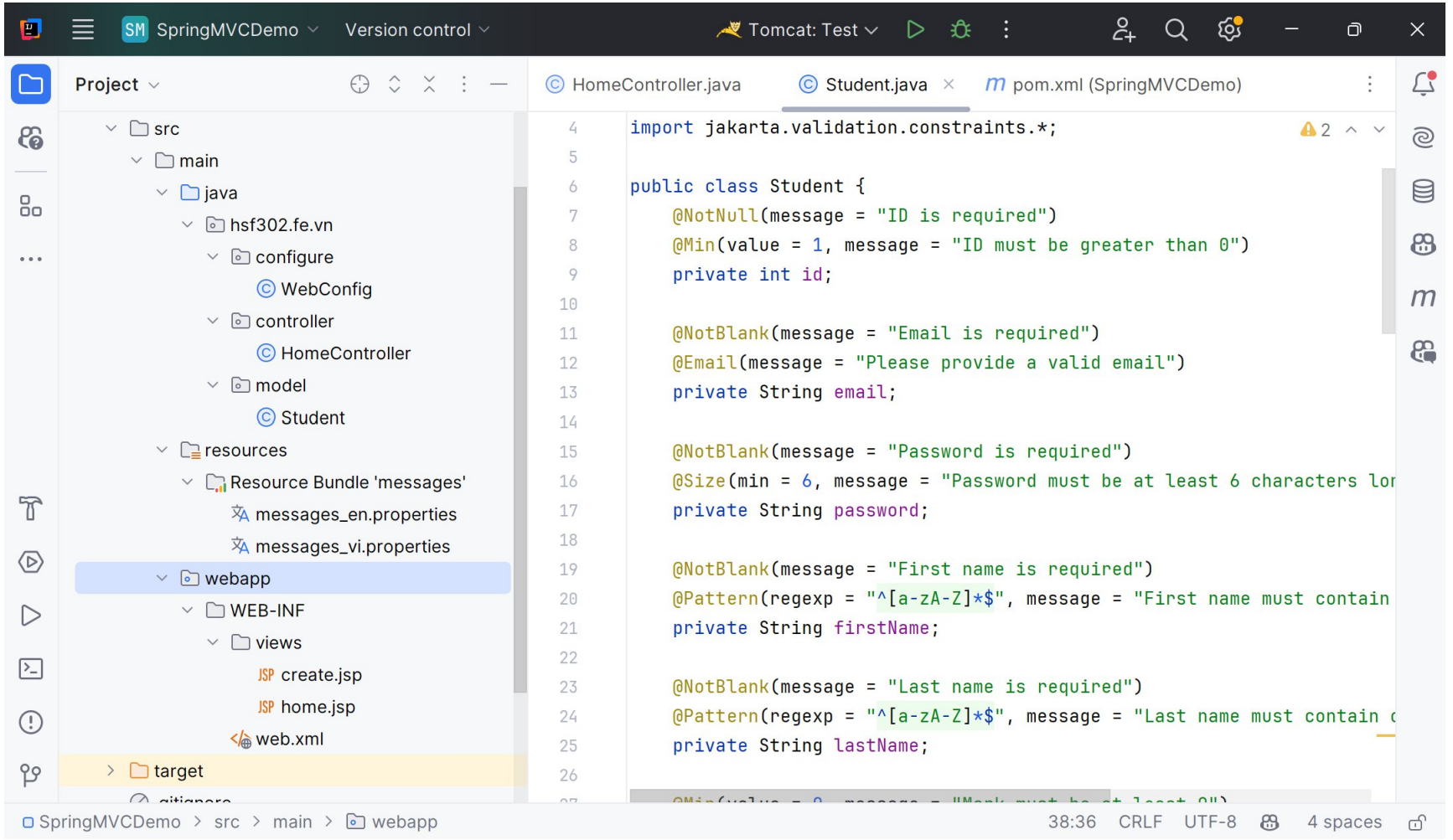
Run Project


SpringMVCDemo
Version control
Tomcat: SpringMVCDemo
Project
SpringMVCDemo D:\Teaching\FPT\ComboJava\HSF302\Code\SpringMVCDemo
.idea
.smarttomcat
src
main
resources
webapp
WEB-INF
web.xml
index.jsp
.gitignore
pom.xml
External Libraries
Scratches and Consoles
pom.xml (SpringMVCDemo)
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>hsf302.fe.com</groupId>
<artifactId>SpringMVCDemo</artifactId>
<packaging>war</packaging>
<version>1.0-SNAPSHOT</version>
<name>SpringMVCDemo Maven Webapp</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
<build>
<finalName>SpringMVCDemo</finalName>
</build>
</project>
Run
Tomcat: SpringMVCDemo
Console
Tomcat Access Log
23-Dec-2024 00:44:02.871 INFO [main] org.apache.catalina.core.AprLifecycleListener.initializeSSL OpenSSL successfully initialized [OpenSSL 3.0.14 4 Jun 2024]
23-Dec-2024 00:44:03.199 INFO [main] org.apache.coyote.AbstractProtocol.init Initializing ProtocolHandler ["http-nio-8080"]
23-Dec-2024 00:44:03.214 INFO [main] org.apache.catalina.startup.Catalina.load Server initialization in [595] milliseconds
23-Dec-2024 00:44:03.261 INFO [main] org.apache.catalina.core.StandardService.startInternal Starting service [Catalina]
23-Dec-2024 00:44:03.261 INFO [main] org.apache.catalina.core.StandardEngine.startInternal Starting Servlet engine: [Apache Tomcat/11.0.2]
23-Dec-2024 00:44:03.270 INFO [main] org.apache.catalina.startup.HostConfig.deployDescriptor Deploying deployment descriptor [C:\Users\Unknow\.SmartTomcat\SpringMVCDemo\SpringMVCDemo\conf\Catalina\localhost\SpringMVCDemo.xml
23-Dec-2024 00:44:03.566 INFO [main] org.apache.catalina.startup.HostConfig.deployDescriptor Deployment of deployment descriptor [C:\Users\Unknow\.SmartTomcat\SpringMVCDemo\SpringMVCDemo\conf\Catalina\localhost\SpringMVCDemo
23-Dec-2024 00:44:03.571 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
23-Dec-2024 00:44:03.592 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in [375] milliseconds
http://localhost:8080/SpringMVCDemo
SpringMVCDemo > src > main > webapp
8:42 CRLF UTF-8 2 spaces*

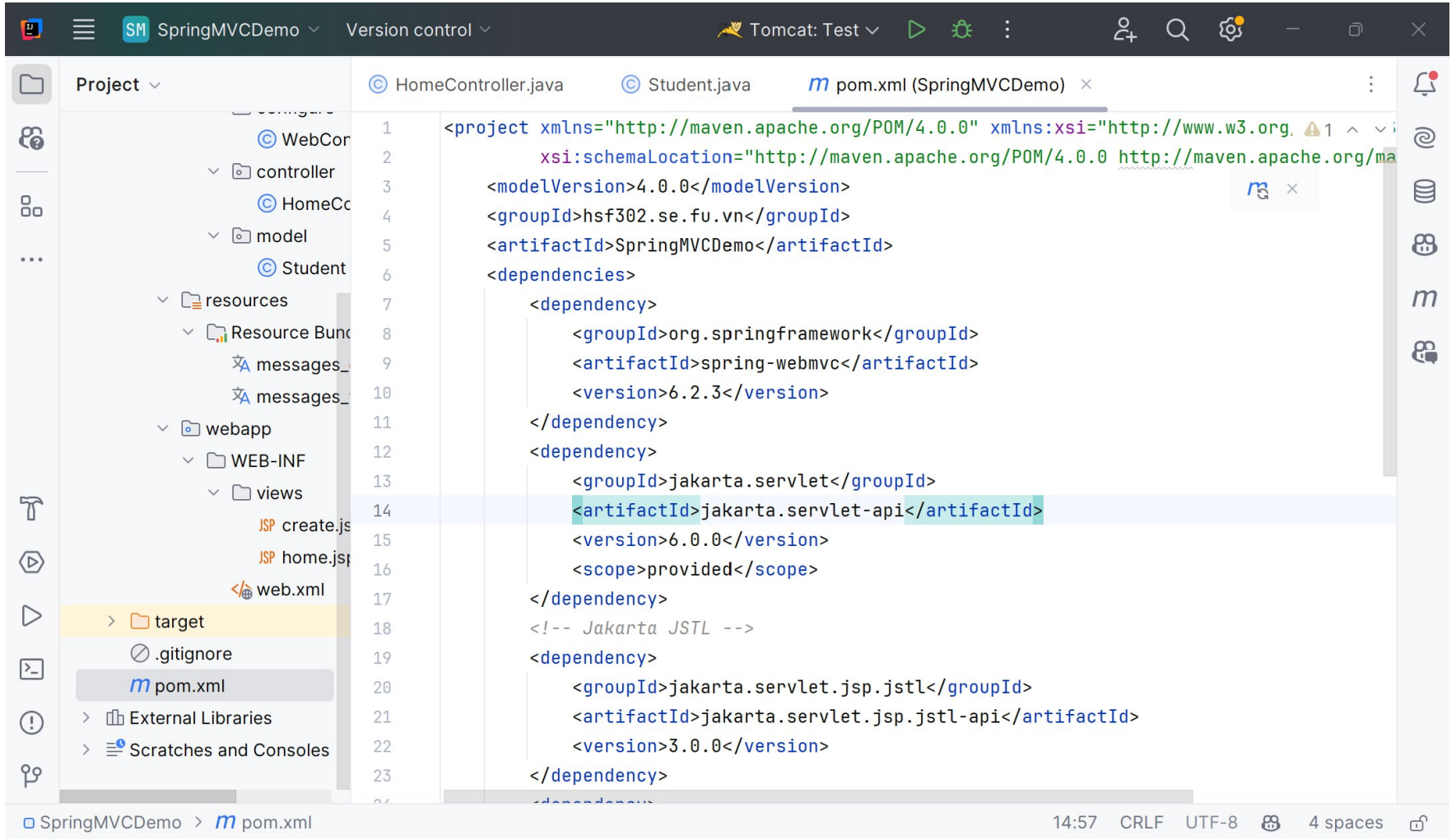
Result


localhost:8080/SpringMVCDem
localhost:8080/SpringMVCDemo/
Hello World!

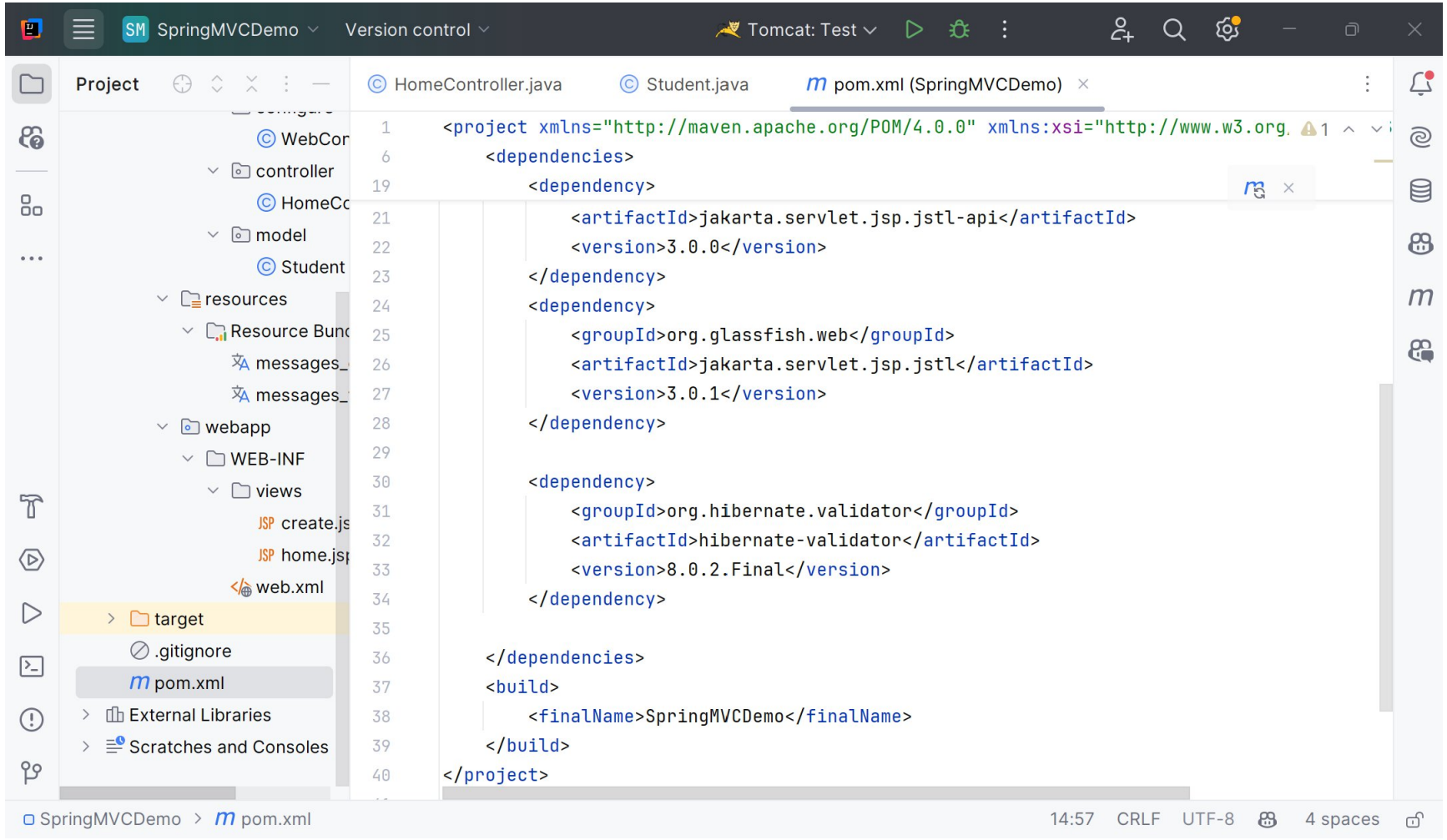
Create a structure of Maven Project



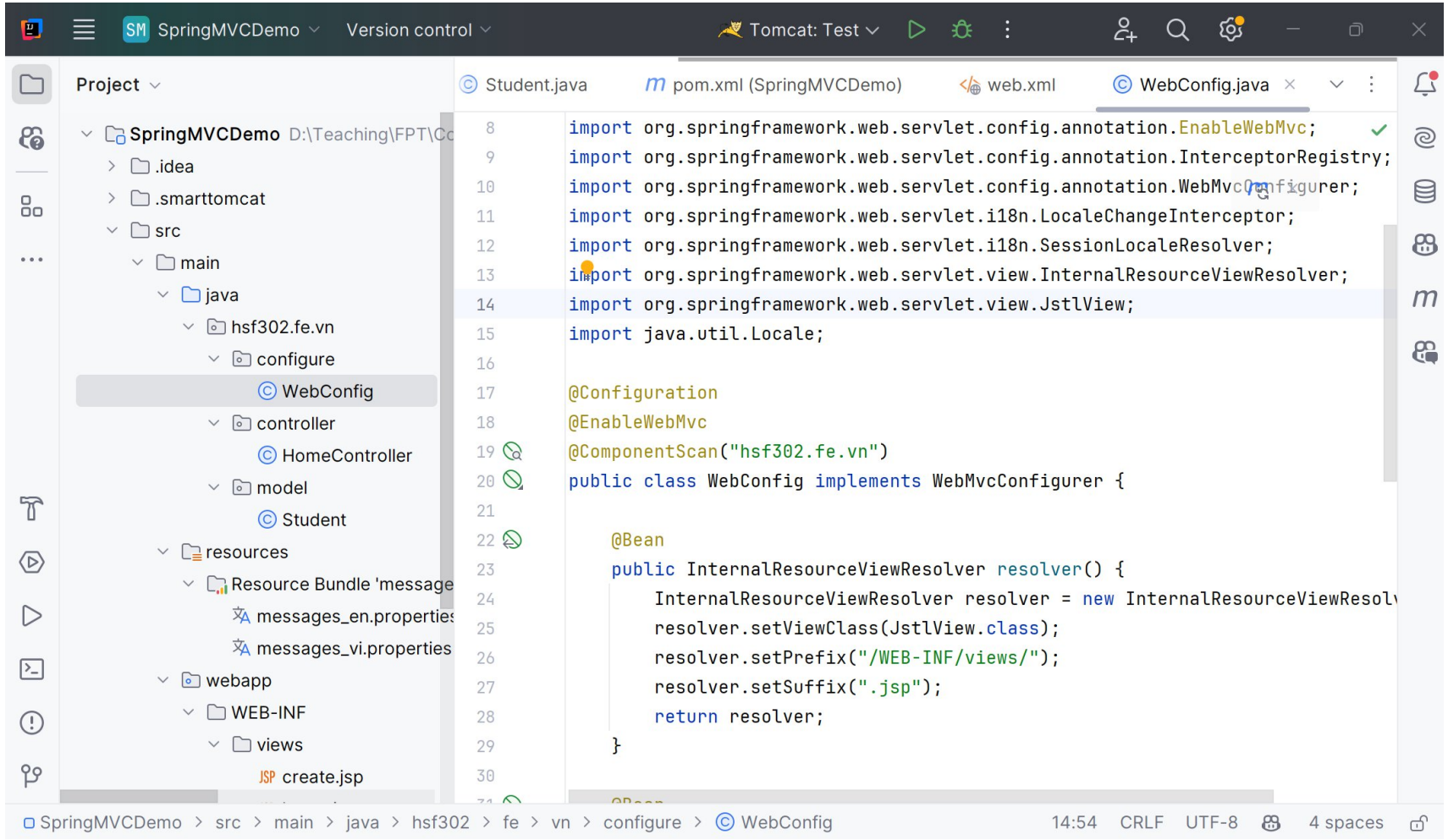
Edit pom.xml


SpringMVCDemo
Version control
Tomcat: Test
Project
HomeController.java
Student.java
pom.xml (SpringMVCDemo)
WebCor
controller
HomeCc
model
Student
resources
Resource Bunc
messages_
messages_
webapp
WEB-INF
views
create.js
home.jsp
web.xml
target
.gitignore
pom.xml
External Libraries
Scratches and Consoles
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/ma
<modelVersion>4.0.0</modelVersion>
<groupId>hsf302.se.fu.vn</groupId>
<artifactId>SpringMVCDemo</artifactId>
<dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>6.2.3</version>
</dependency>
<dependency>
<groupId>jakarta.servlet</groupId>
<artifactId>jakarta.servlet-api</artifactId>
<version>6.0.0</version>
<scope>provided</scope>
</dependency>
<!-- Jakarta JSTL -->
<dependency>
<groupId>jakarta.servlet.jsp.jstl</groupId>
<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
<version>3.0.0</version>
</dependency>
SpringMVCDemo
pom.xml
14:57
CRLF
UTF-8
4 spaces

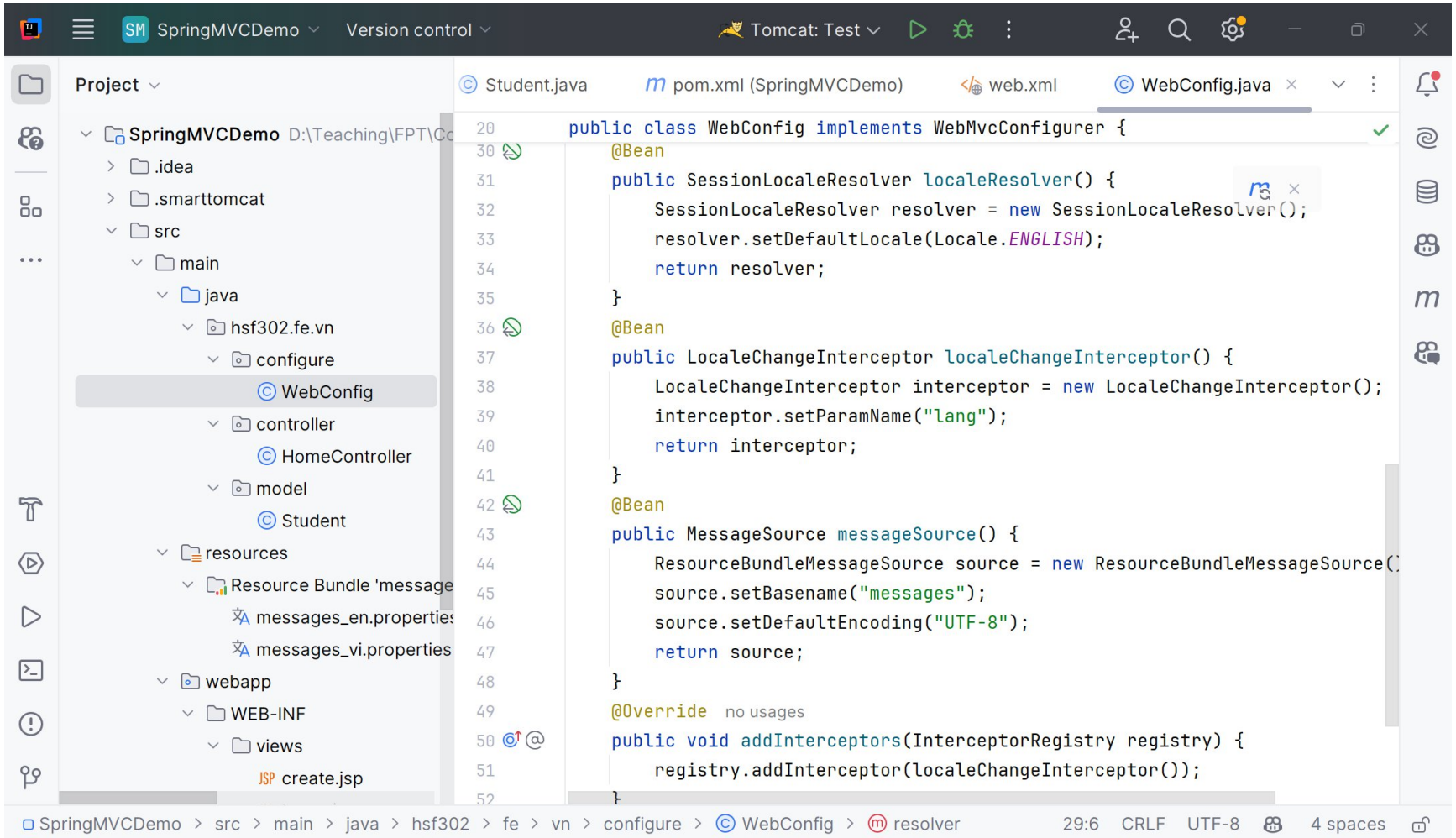
Edit pom.xml


SpringMVCDemo
Version control
Tomcat: Test
Project
HomeController.java
Student.java
pom.xml (SpringMVCDemo)
WebCor
controller
HomeCc
model
Student
resources
Resource Bunc
messages_
messages_
webapp
WEB-INF
views
create.js
home.jsp
web.xml
target
.gitignore
pom.xml
External Libraries
Scratches and Consoles
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
<dependencies>
<dependency>
<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
<version>3.0.0</version>
</dependency>
<dependency>
<groupId>org.glassfish.web</groupId>
<artifactId>jakarta.servlet.jsp.jstl</artifactId>
<version>3.0.1</version>
</dependency>
<dependency>
<groupId>org.hibernate.validator</groupId>
<artifactId>hibernate-validator</artifactId>
<version>8.0.2.Final</version>
</dependency>
</dependencies>
<build>
<finalName>SpringMVCDemo</finalName>
</build>
</project>
SpringMVCDemo
pom.xml
14:57
CRLF
UTF-8
4 spaces

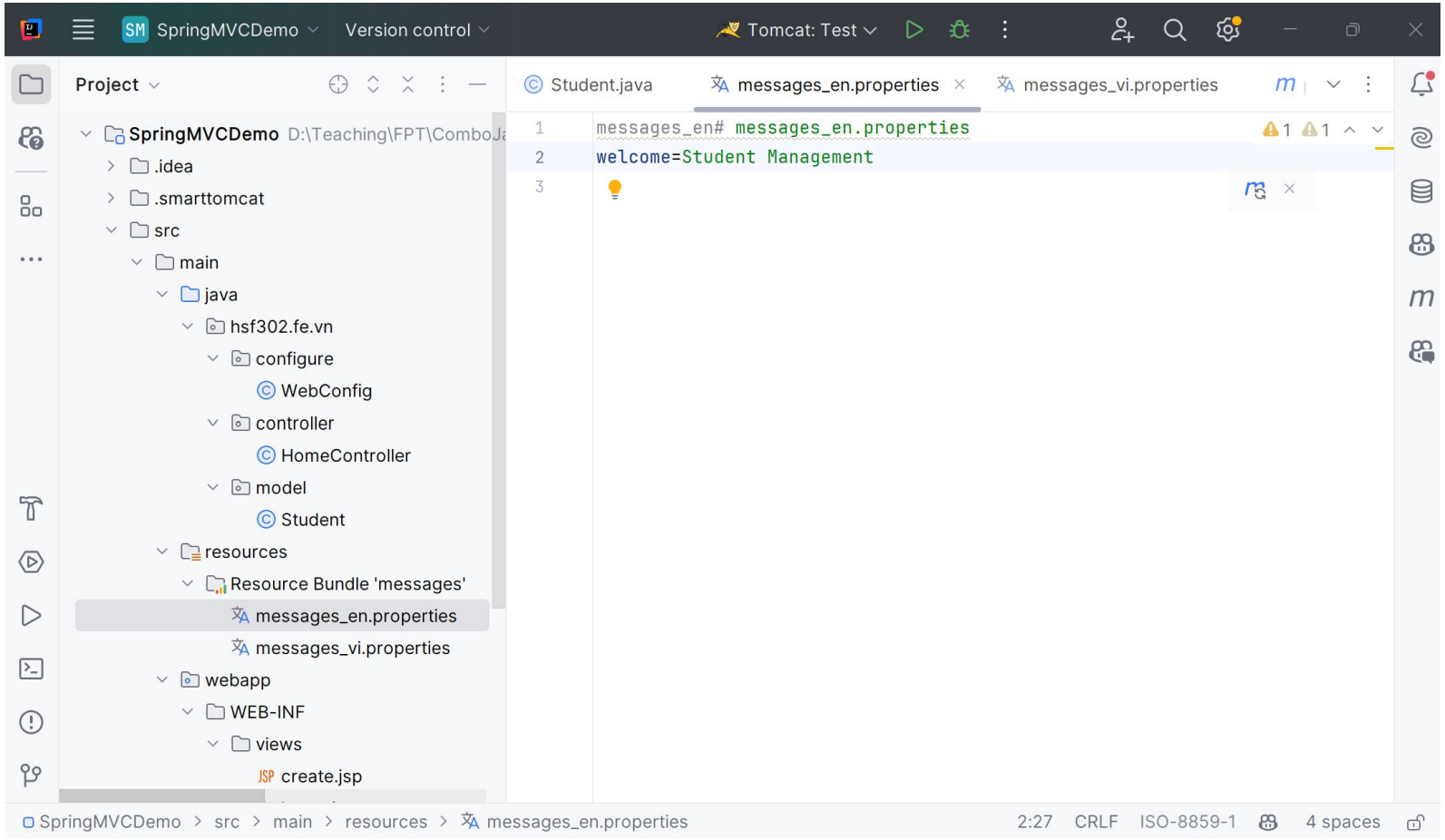
Create the WebConfig.java



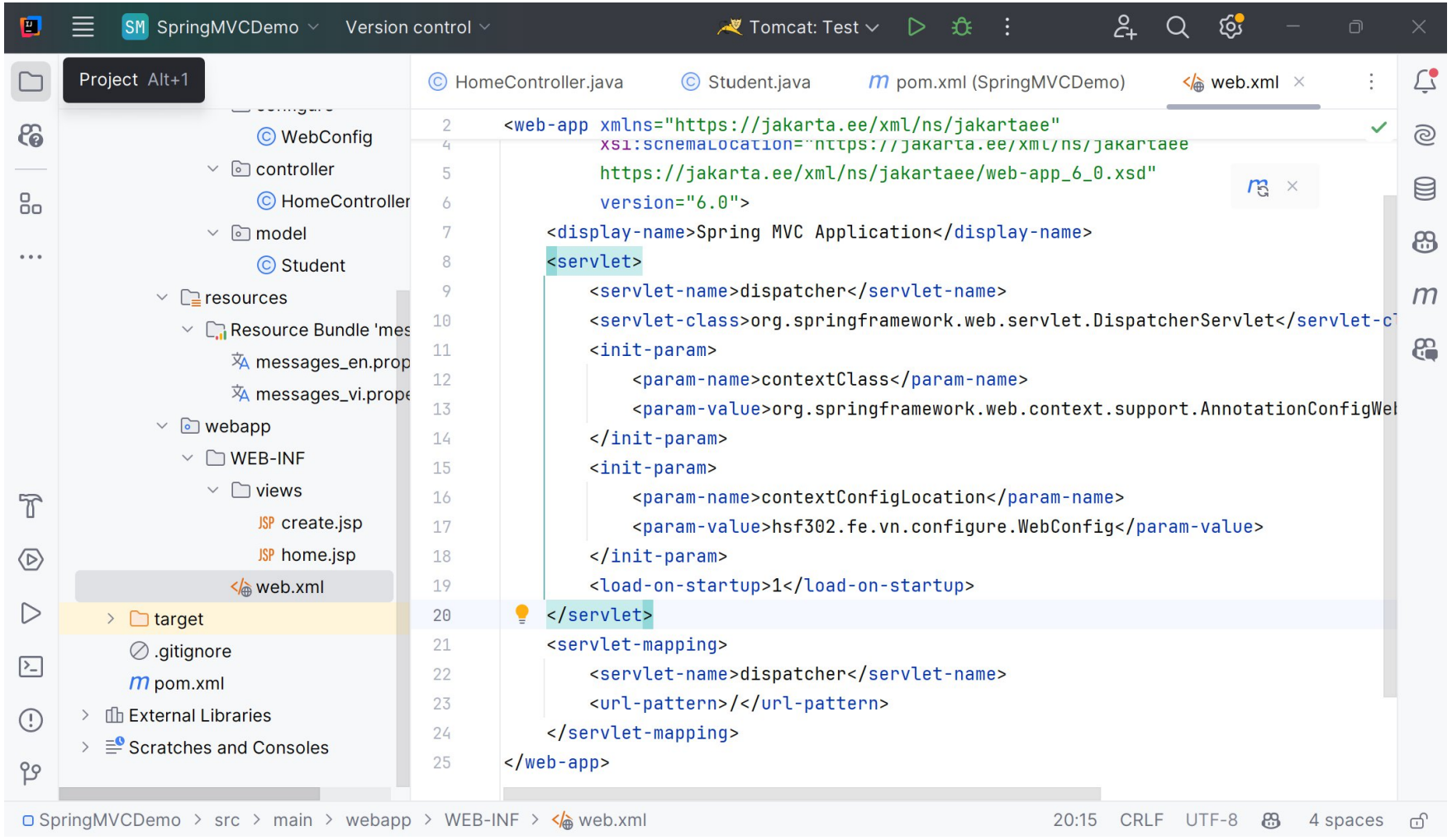
Edit the WebConfig.java



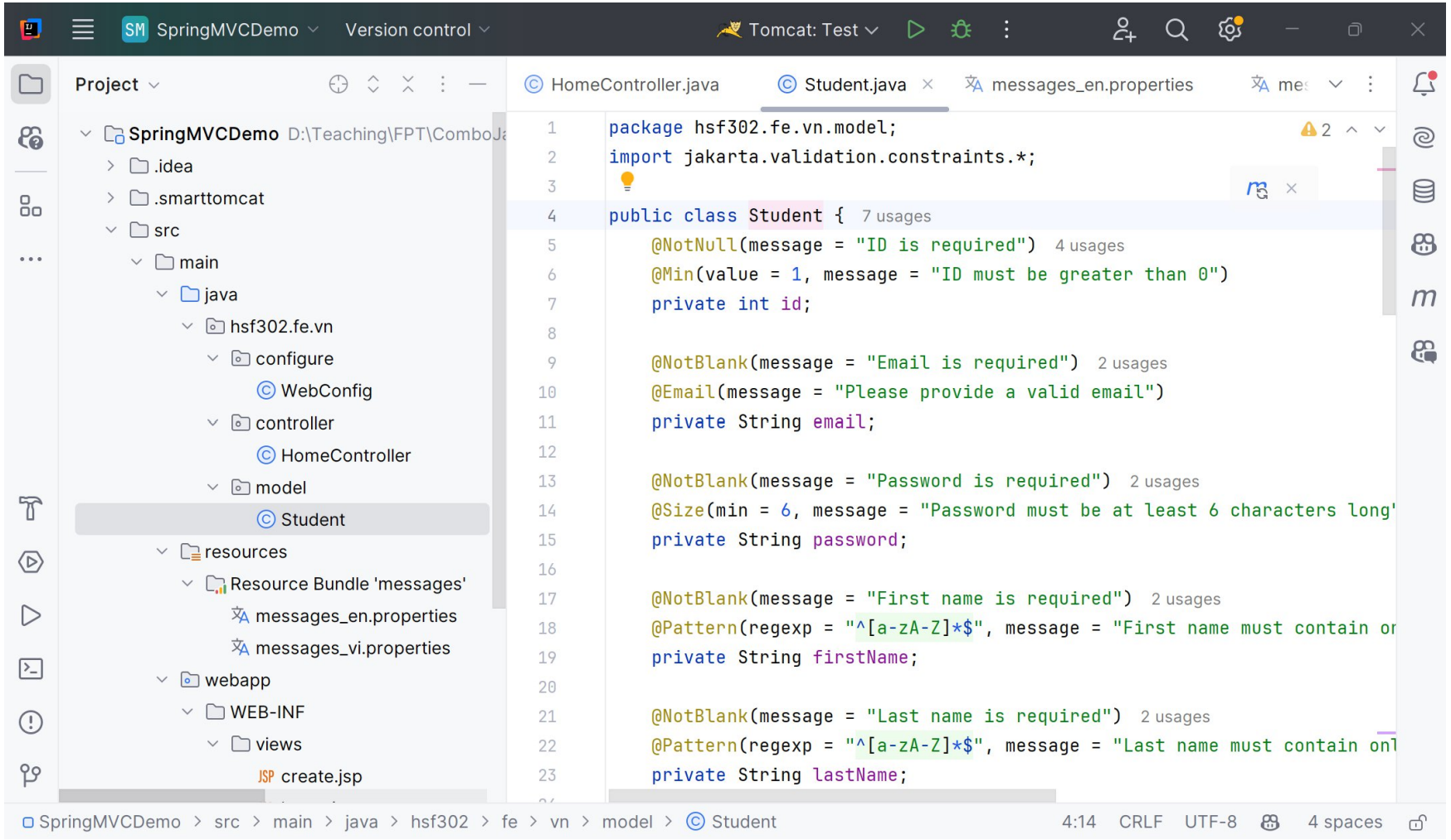
Create messages_en.properties and messages_vi.properties



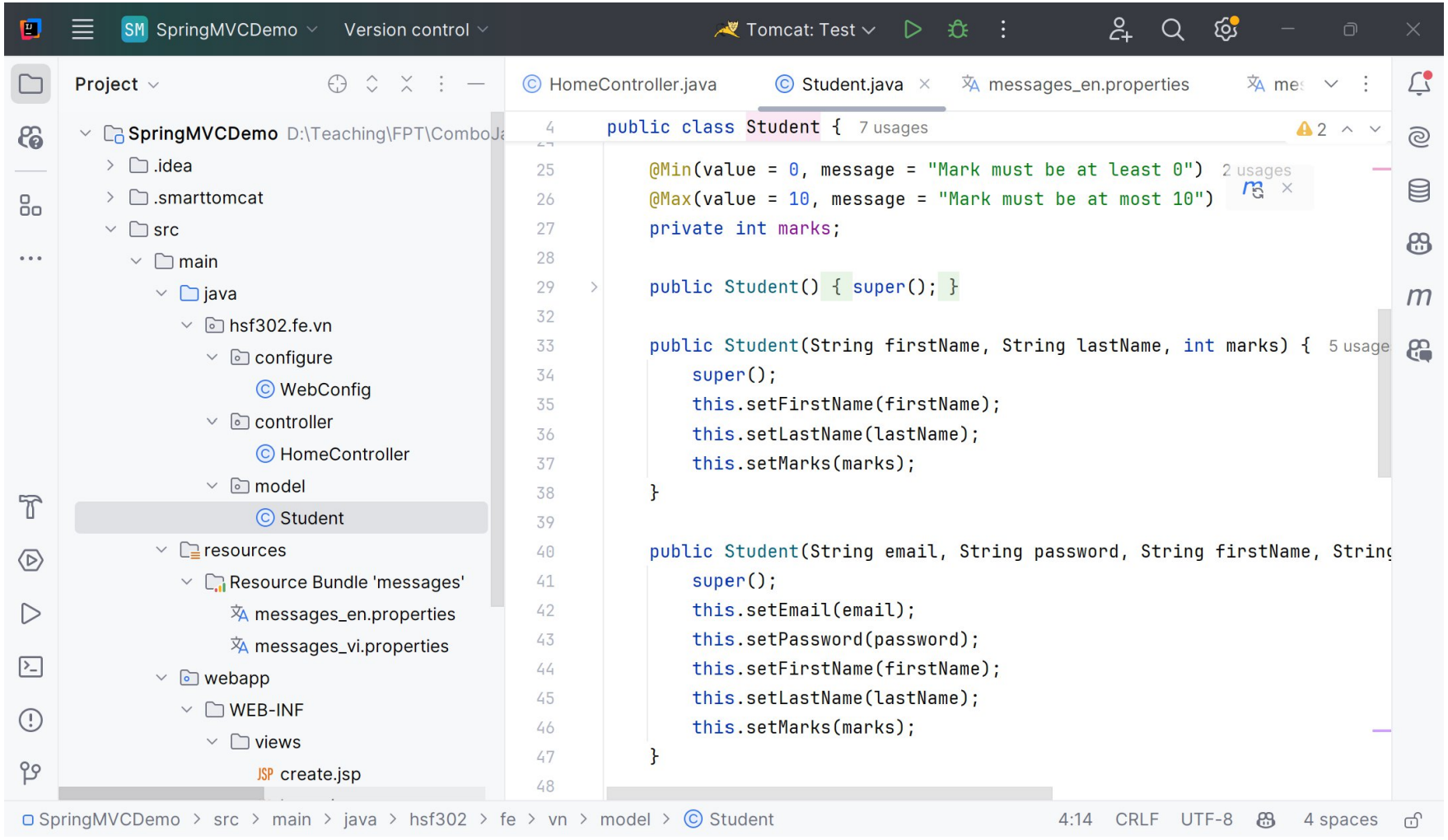
Edit web.xml


SpringMVCDemo
Version control
Tomcat: Test
Project Alt+1
HomeController.java
Student.java
pom.xml (SpringMVCDemo)
web.xml
WebConfig
controller
HomeController
model
Student
resources
Resource Bundle 'mes
messages_en.prop
messages_vi.prope
webapp
WEB-INF
views
create.jsp
home.jsp
web.xml
target
.gitignore
pom.xml
External Libraries
Scratches and Consoles
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
version="6.0">
<display-name>Spring MVC Application</display-name>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-c
<init-param>
<param-name>contextClass</param-name>
<param-value>org.springframework.web.context.support.AnnotationConfigWe
</init-param>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>hsf302.fe.vn.configure.WebConfig</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
SpringMVCDemo > src > main > webapp > WEB-INF > web.xml
20:15 CRLF UTF-8 4 spaces

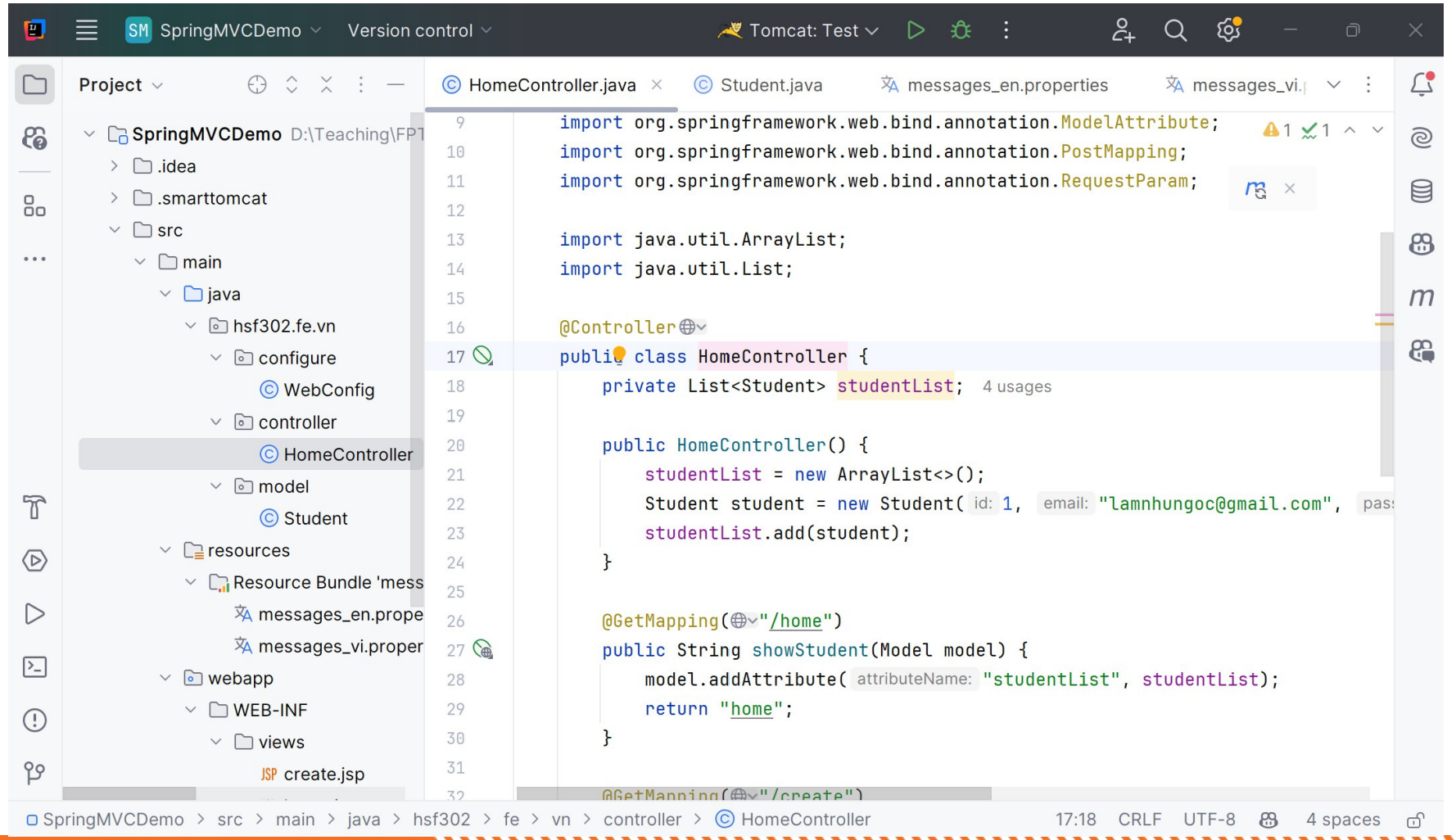
Create the Student.java



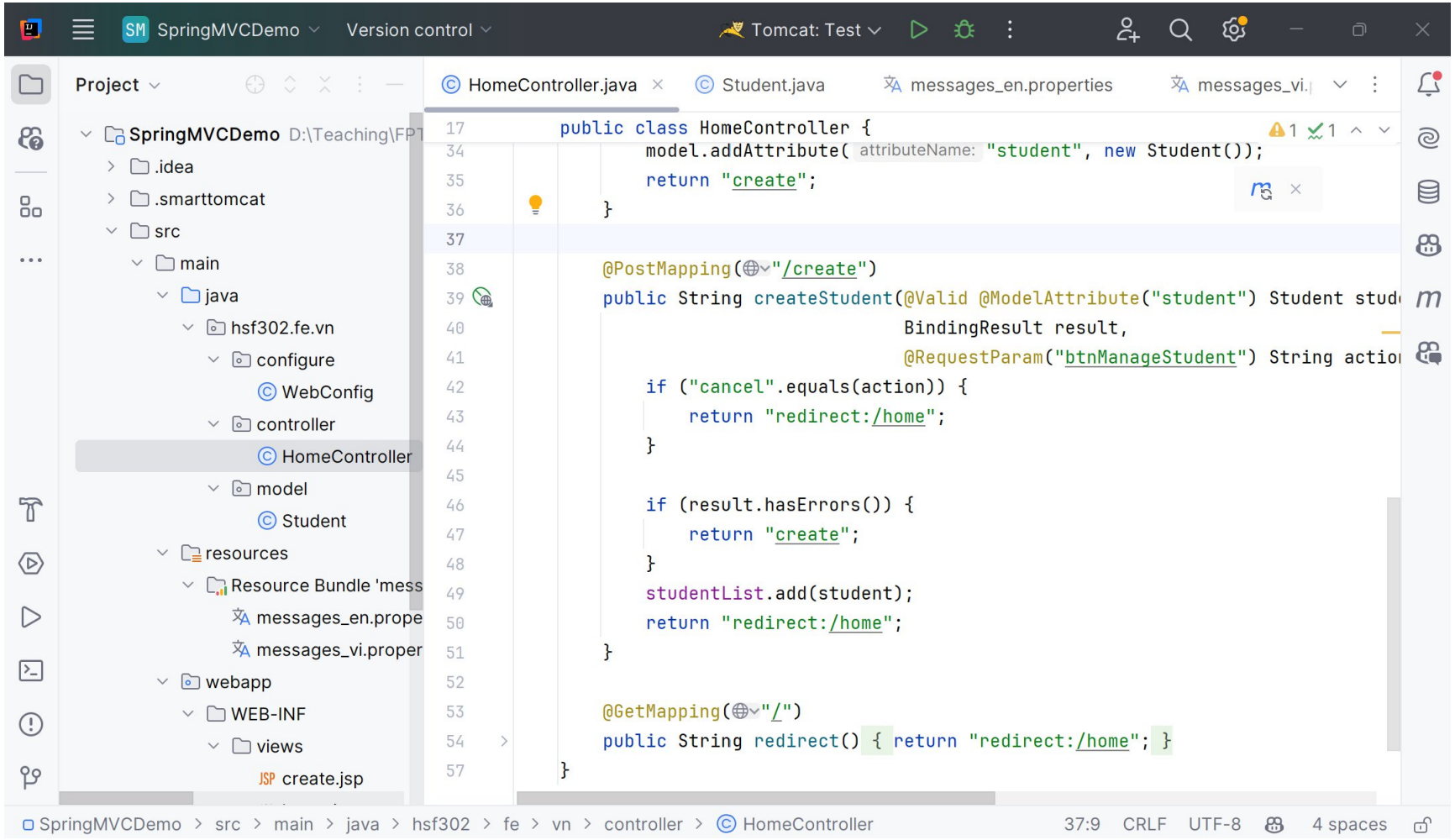
Edit the Student.java



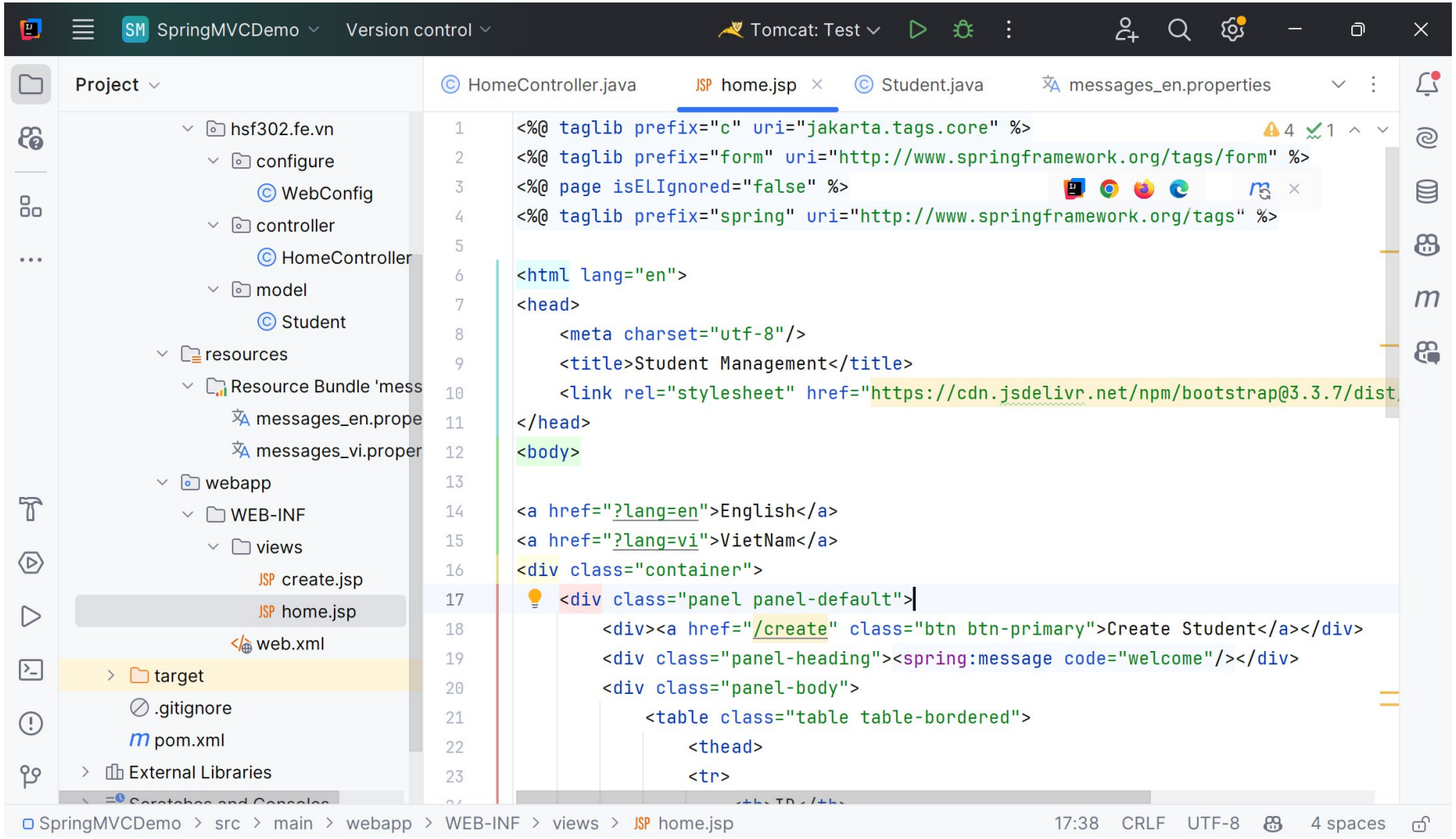
Create HomeController.java



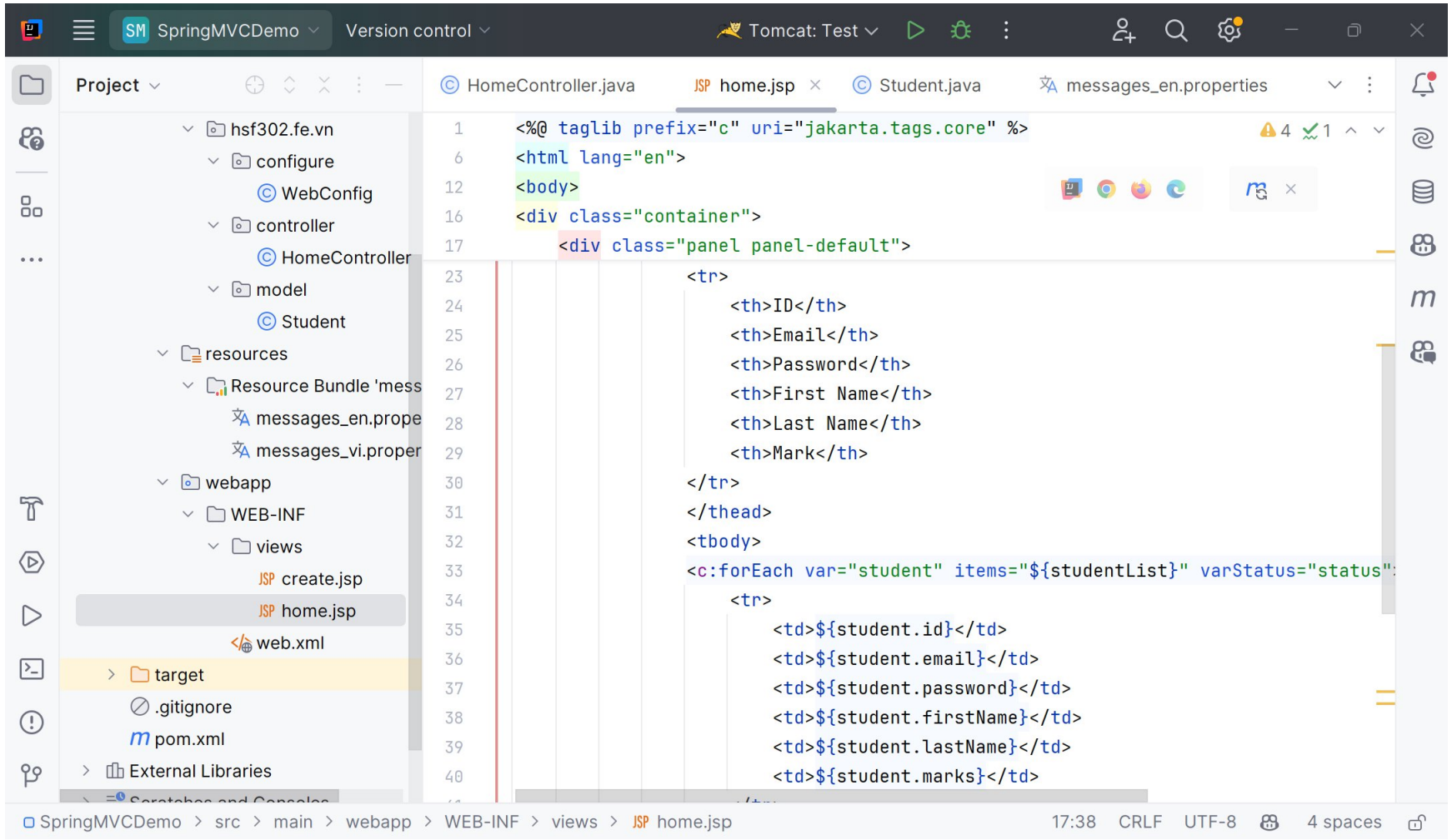
Edit HomeController.java



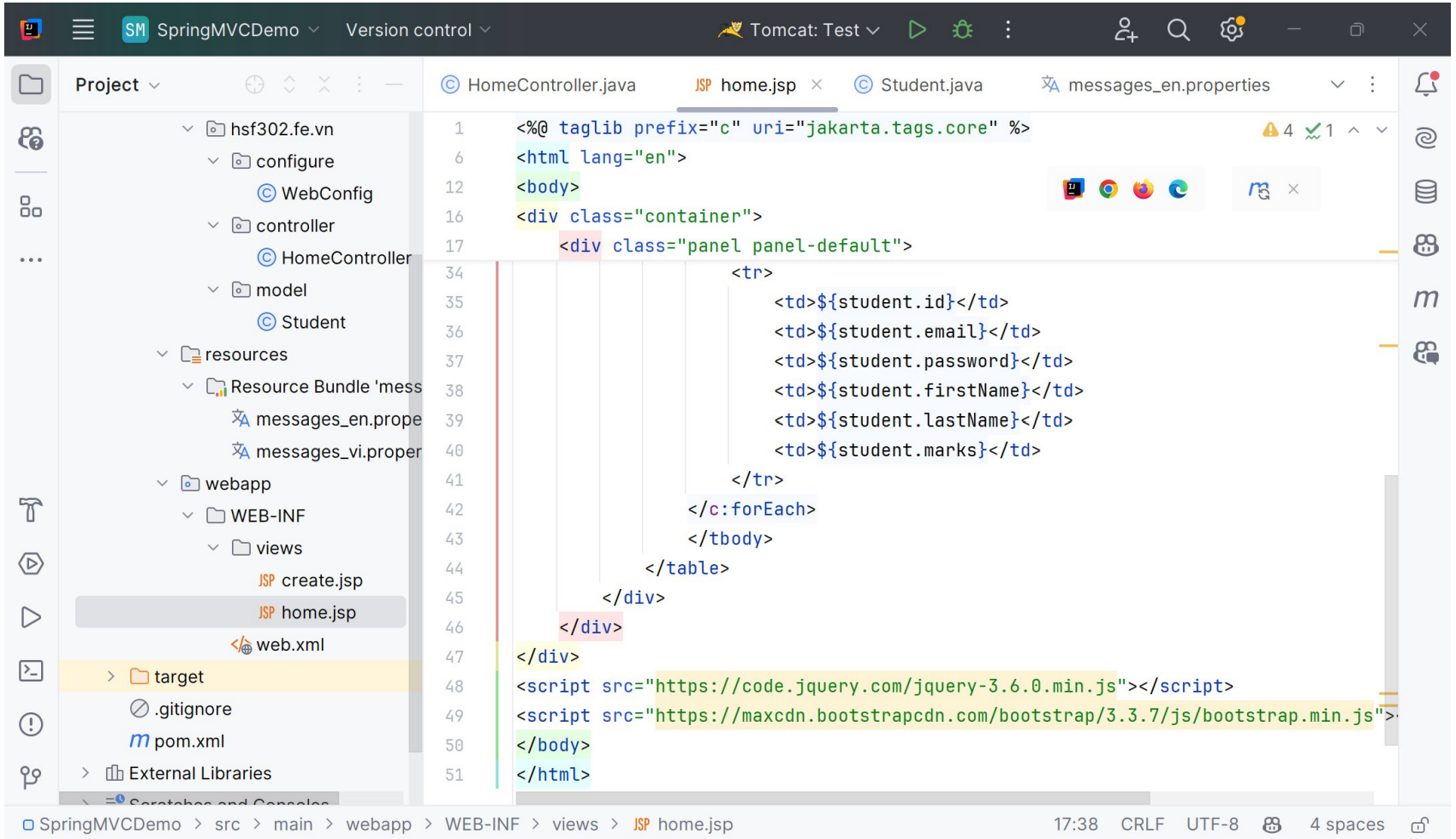
Create home.jsp



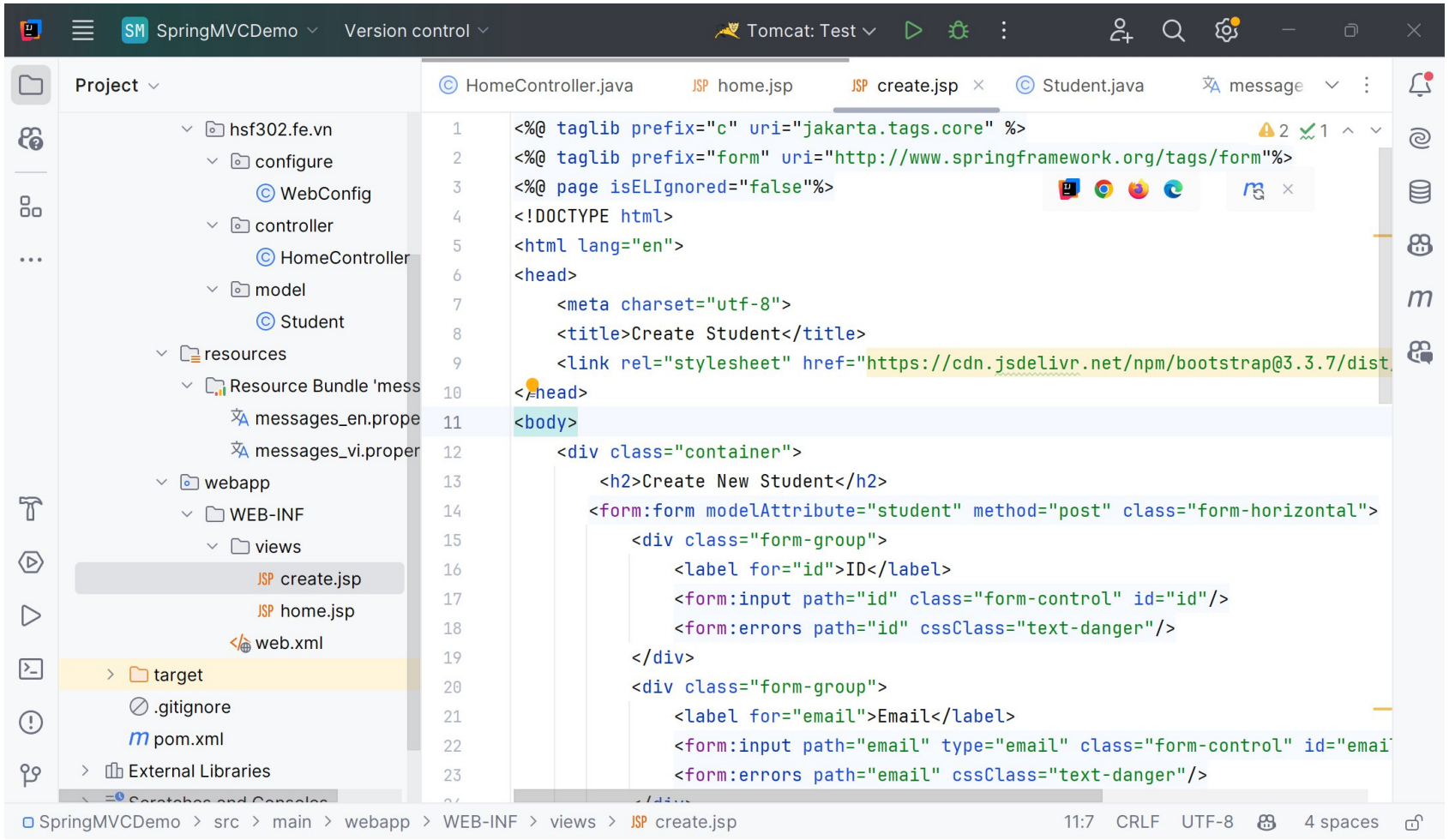
Edit home.jsp



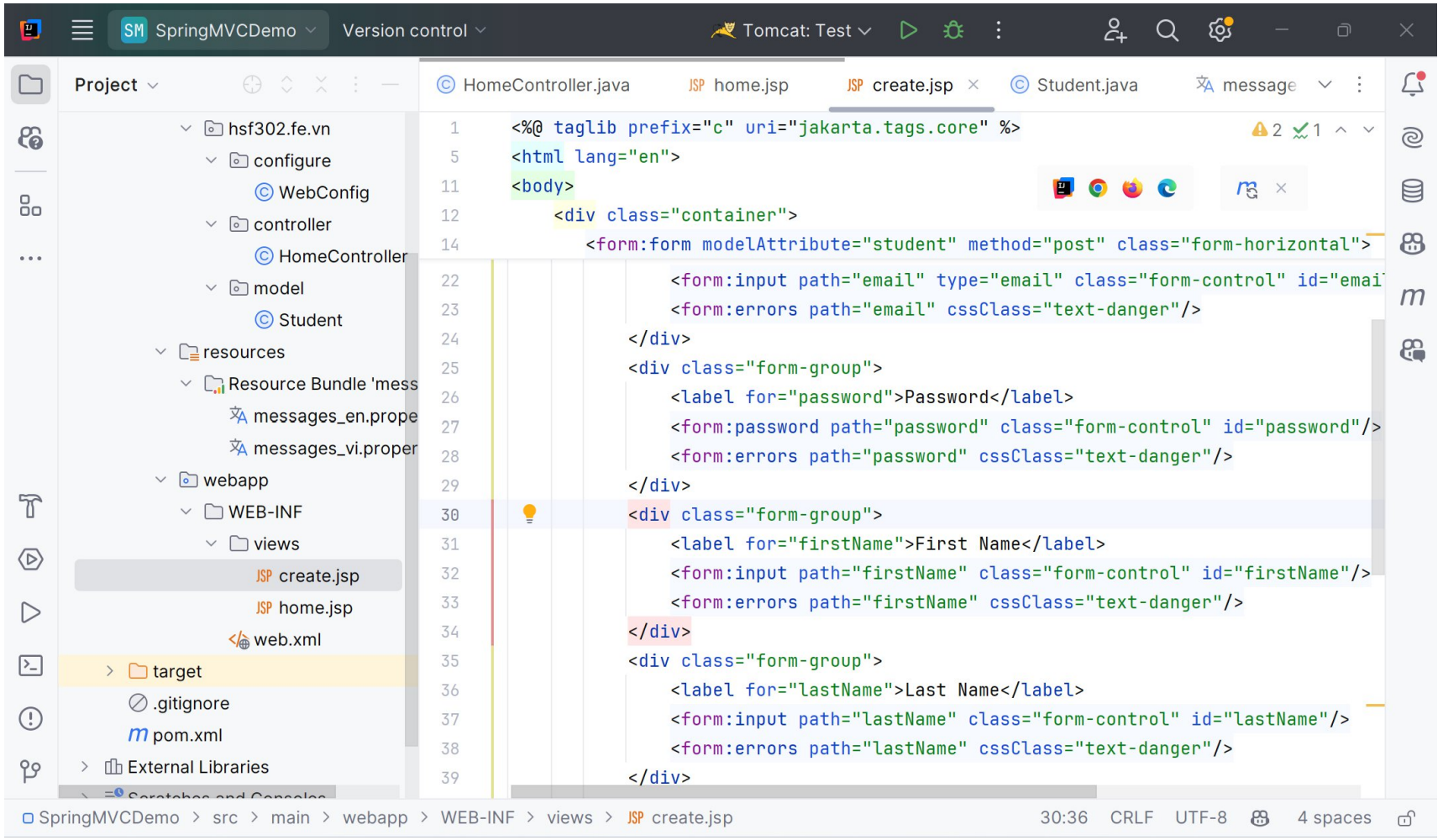
Edit home.jsp



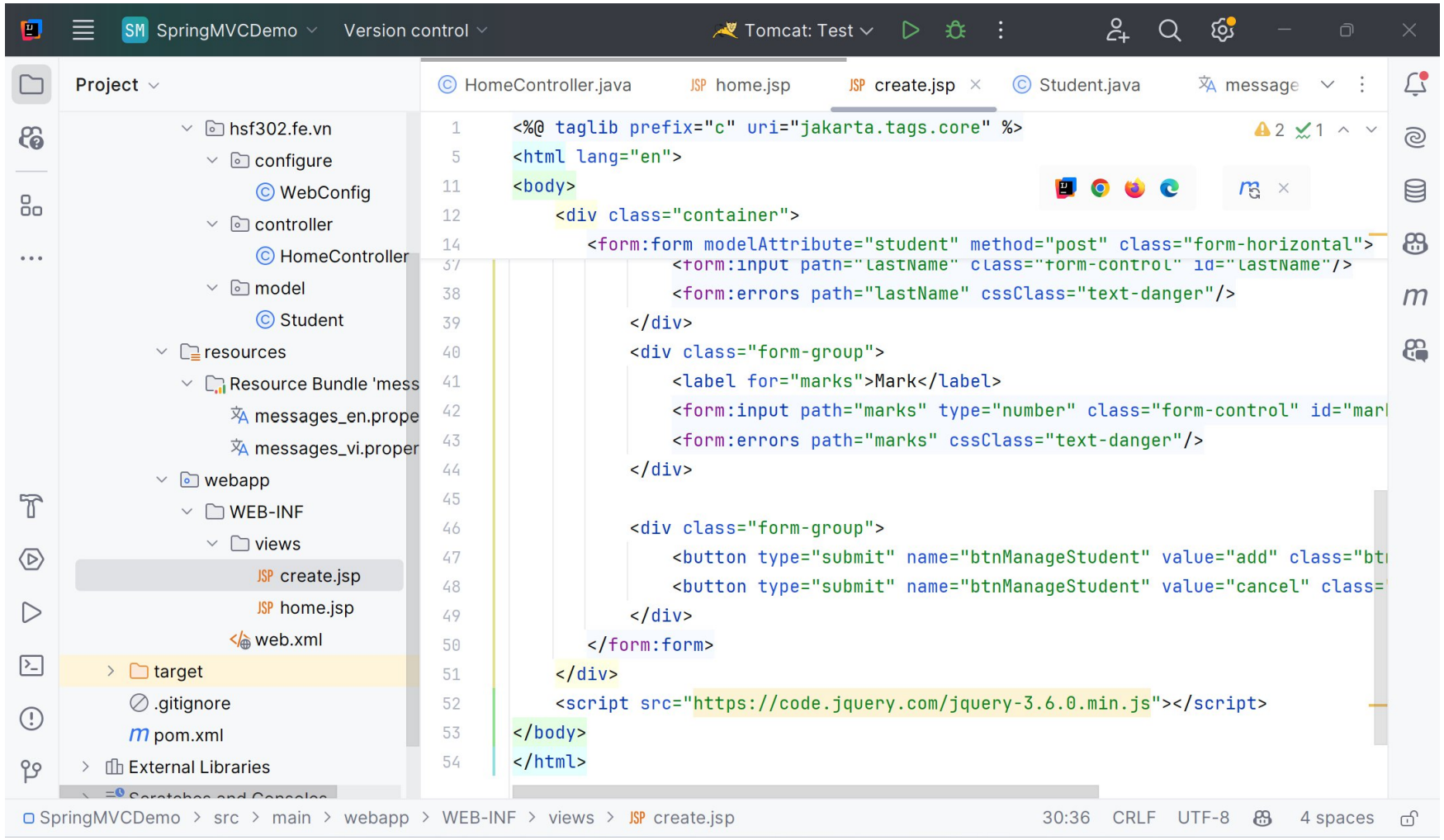
Create create.jsp



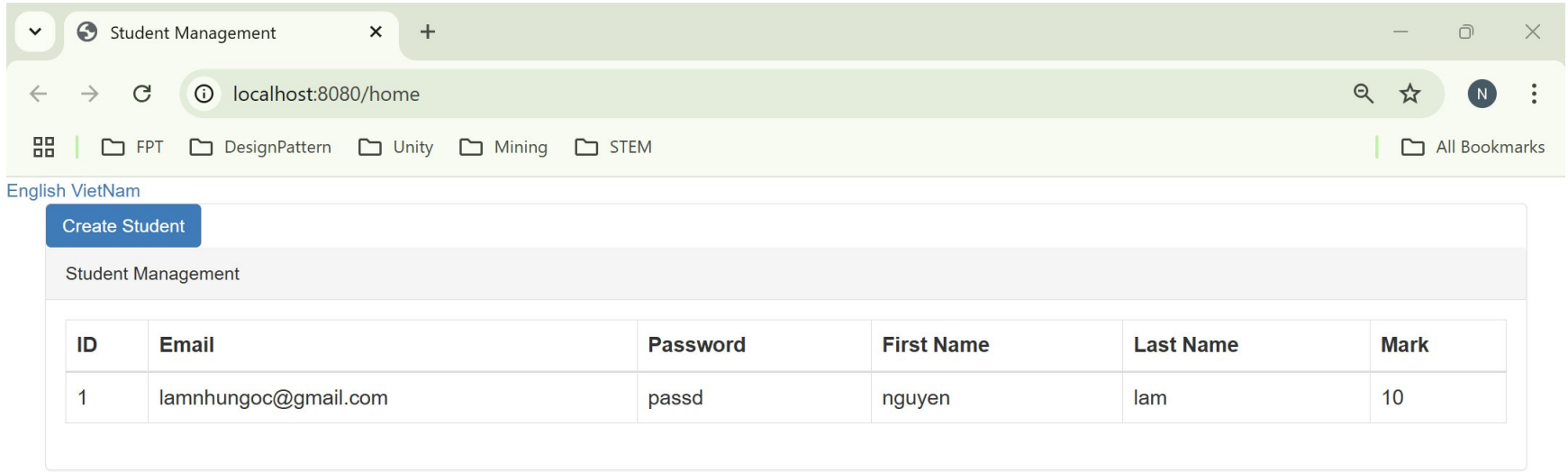
Edit create.jsp



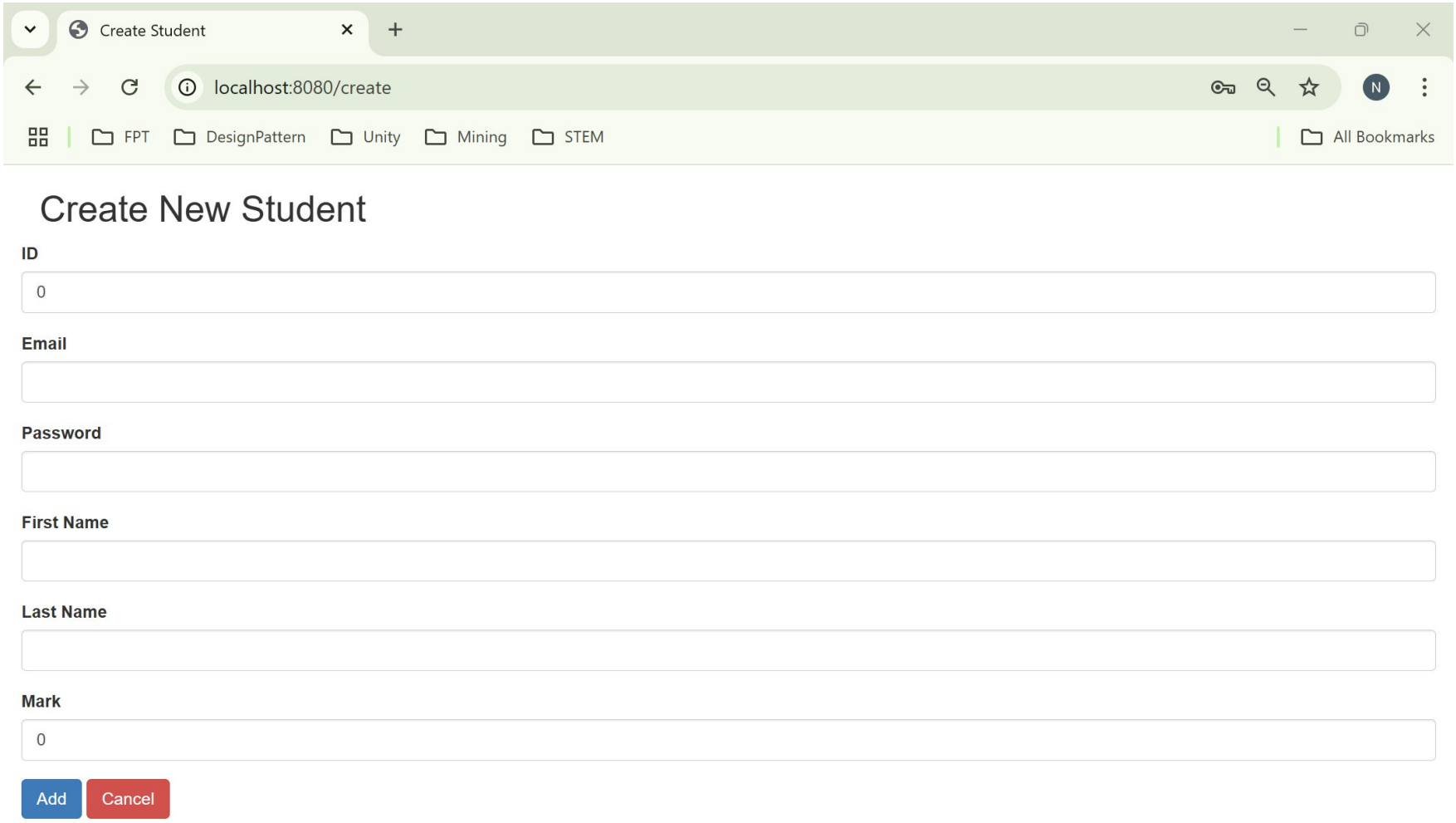
Edit create.jsp



Run Program


Student Management
localhost:8080/home
FPT
DesignPattern
Unity
Mining
STEM
All Bookmarks
English VietNam
Create Student
Student Management
ID
Email
Password
First Name
Last Name
Mark
1
lamnhungoc@gmail.com
passd
nguyen
lam
10

Run Program


Create Student
localhost:8080/create
FPT
DesignPattern
Unity
Mining
STEM
All Bookmarks
Create New Student
ID
0
Email
Password
First Name
Last Name
Mark
0
Add
Cancel

Summary

Concepts were introduced:

- Spring MVC Basics
- Spring MVC Framework
- Controller/Model/View
- Spring Interceptor
- Spring Validator