

73: Mapping Multiple paths inside Spring Web Application

```
8 @Controller
9 public class HomeController {
10
11     @RequestMapping(value = {"", "/", "home", "/login"})
12     public String displayHomePage()
13     {
14         return "home.html";
15     }
16 }
```

Multiple mapping for the same Web application:

We can Access the Web Application with any of the following url's:

<http://localhost:8080/login>

<http://localhost:8080/>

<http://localhost:8080>

<http://localhost:8080/home>

All the above will direct to same Application.

74: Introduction to Thymeleaf

until now we are always displaying a static message saying that Welcome to George School for all the users. Like if there are two different people trying to access your web application, we are always displaying the same static content, but ideally any production web applications, they should be able to display content dynamically based upon the user who is trying to access it.

Like in the scenarios, if I try to log in into an web application, I should be able to see the data which belongs to me only the same applies for the other person.

Thymeleaf, which is a modern server side java template engine. with the help of Thymeleaf, I can build some templates which are very similar to html files inside my web application.

while responding to my browser request, thymeleaf along with the spring will generate a html content based upon the template files that I have defined by populating the dynamic data, that I want to populate.

So this way Thymeleaf will bring some dynamic content display into our web application.

Other Famous Template Engines for displaying the content dynamically :

Jakarta Server Pages(JSP)

Jakarta Server Faces(JSF)

Apache Free Maker

Groovy

We are choosing Thymeleaf because we have greater integration with Spring MVC and Spring Security. Developers are building web applications by separating them into two components.

One is a frontend component, the other one is a backend component. So inside frontend, they will leverage certain javascript frameworks like Angular, React, which can help you to make a Rest Api invocation to the backend and to display the content dynamically. Thymeleaf because it gives a flexibility to them like a single developer can take care of both developing frontend and backend and one more advantage that we have is you don't have to deploy them into two different servers.

Like if you develop a web application with Angular and spring, you need to deploy your angular code separately and spring code separately.

Whereas with Thymeleaf and spring, if you develop a web application, you can deploy everything into a single server.

Thymeleaf and Spring, they have a very good integration and they offer certain template resolvers like `SpringResourceTemplateResolver`, which will help us to populate the content dynamically. Like I can define my template inside my spring project and this template will be discussing in few minutes. How this template looks like is, this looks like very familiar with the html.

We'll just bring some flavor of thymeleaf to display the content dynamically. Like you can see here on the left-hand side, I'm trying to display a table to the end user with the

help of my html code, because at the end of the day my browser can understand html, css and javascript. So it won't understand the thymeleaf. So it is the job of the `SpringResourceTemplateResolver` to convert this thymeleaf template into a html page after populating the data whenever it is trying to respond to a request that came from a browser. One more advantage with the templates is we don't have to write a lot of code.

INTRODUCTION TO THYMELEAF

easy
bytes

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments. This allows developers to build dynamic content inside the web applications.
- Thymeleaf has great integration with Spring especially with Spring MVC, Spring Security etc.
- The Thymeleaf + Spring integration packages offer a `SpringResourceTemplateResolver` implementation which uses all the Spring infrastructure for accessing and reading resources in applications, and which is the recommended implementation in Spring-enabled applications.

```
<table>
<thead>
<tr>
<th th:text="#{msgs.headers.name}">Name</th>
<th th:text="#{msgs.headers.price}">Price</th>
</tr>
</thead>
<tbody>
<tr th:each="prod: ${allProducts}">
<td th:text="${prod.name}">Oranges</td>
<td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
</tr>
</tbody>
</table>
```

Other famous template engines
supported by Spring:

1. Jakarta Server Pages (JSP)
2. Jakarta Server Faces (JSF)
3. Apache FreeMarker
4. Groovy

For more details, pls refer <https://www.thymeleaf.org/>

75: Building Dynamic Content Using Thymeleaf

Thymeleaf is a template not a static file.

So, we need to move that home.html to templates folder under resources section.

To use the thymeleaf we need to import the namespace of thymeleaf into my html page.

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

Thymeleaf tags starts with th.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>George School</title>
</head>
<body>
<h1 th:text="'Hello ,'+ ${username} + '!!! Welcome to George School'" ></h1>
</body>
</html>
```

Enclosed with in 'Single Quotations' --Static Content

@{username} -- For displaying the dynamic content received from Controller

Entire message is surrounded by "Double Quotes".

In the olden days we used to get the information from UI in the form of http servlet request and also we used to send the response in the form of http servlet response.

So inside those objects we can pass the parameters or variables that we want to be displayed on the UI.

Now, instead of doing all that manually by developer, my spring MVC framework has an interface called Model which take care of all accepting the parameters inside http request and also whatever data we populate inside model object, the same will be sent inside the response in the form of http servlet response.

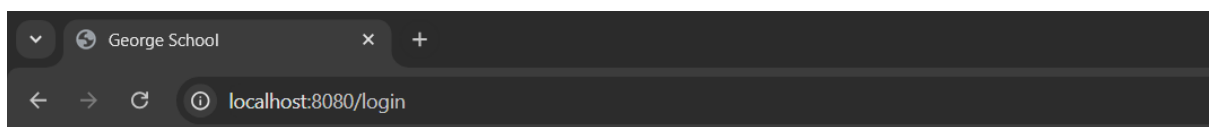
Model is like an interface where we can add the attribute and values associated to it .When the Front end requires it can take it from the model and viceversa.

```
8
9 @Controller
10 public class HomeController {
11
12     @RequestMapping(value= {"", "/", "home", "/login"})
13     public String displayHomePage(Model model)
14     {
15         model.addAttribute("username", "Chinmay Sai");
16         return "home";
17     }
18 }
19 |
```

Model interface will be added with username attribute and it's value.

With these thymeleaf is smart enough to read the variable username from the model object and generate a html code after populating the username value inside this and will send to the browser inside the response.

Example_26_Dynamic_Web_Page_Thymeleaf



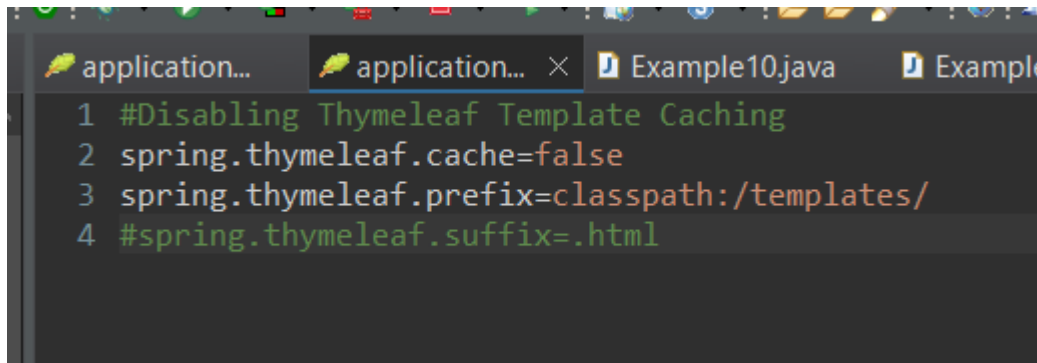
Hello , Chinmay Sai !!! Welcome to George School

76: Disabling Thymeleaf Template Caching

Thymeleaf by default enable the caching so very first time only it will do the compilation of your template and it will put inside a cache until you restart your server again, the same caching will be used.

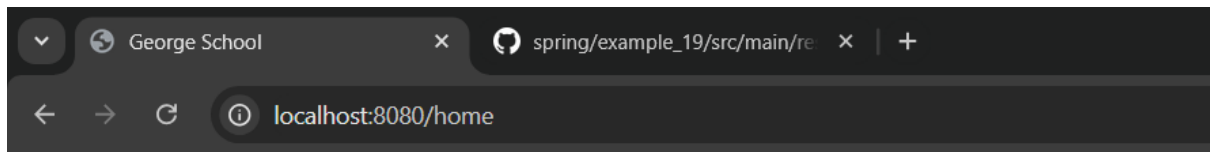
But there will be scenarios where we want changes to be reflected without restarting the server every time.

In that case we can set the caching to false.

A screenshot of an IDE window showing the 'application.properties' file. The file contains four lines of configuration: 1. #Disabling Thymeleaf Template Caching, 2. spring.thymeleaf.cache=false, 3. spring.thymeleaf.prefix=classpath:/templates/, and 4. #spring.thymeleaf.suffix=.html. The IDE has tabs for 'application...', 'application...' (closed), 'Example10.java', and 'Example...'.

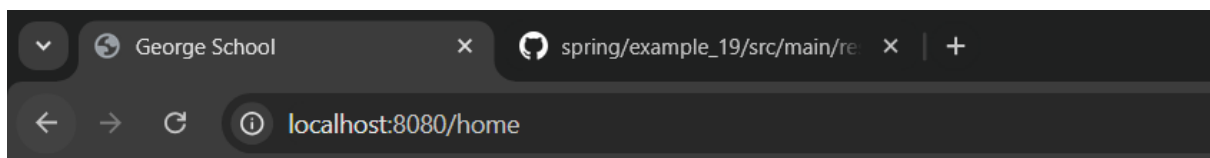
```
1 #Disabling Thymeleaf Template Caching
2 spring.thymeleaf.cache=false
3 spring.thymeleaf.prefix=classpath:/templates/
4 #spring.thymeleaf.suffix=.html
```

Once we set these in the application.properties and save the changes and reload the browser changes will be reflected.



Hello , Chinmay Sai !!! Welcome to George School

Post the changes in the html file without re-starting the server.

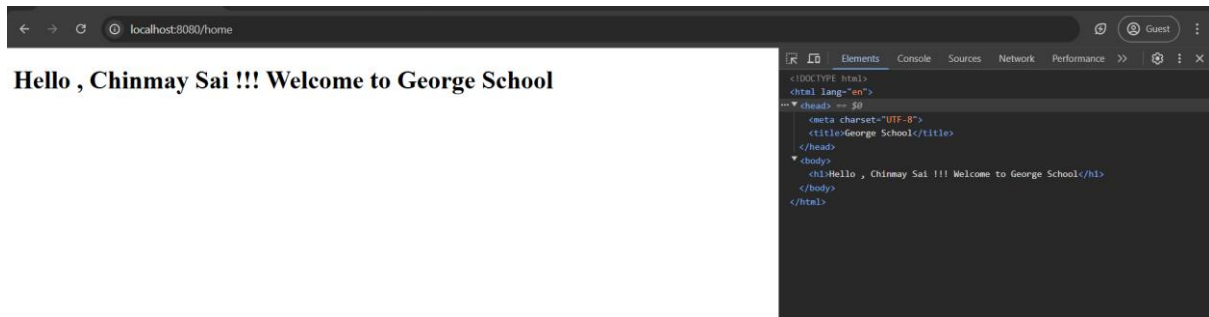


Hi , Chinmay Sai !!! Welcome to George School

Thymeleaf will convert into html file so that browser can understand.

We are disabling only thymeleaf cache.

So if we make any changes to the Java code .We must restart the server to view the changes.



Above is the html that is being returned to display.

77.Introduction to Spring Boot Dev Tools

Automatic restart:

Automatic restart means so whenever you make a change inside your java file or any property file, you just save it and you trigger a build.

So as soon as your build is completed, your dev tools will detect that and do a automatic restart of your server.

Live Reload:

Live reload means it will make you so lazy. Even browser refresh also, you don't have to do by yourself by the time your automatic restart is completed and you visit the browser, the page might have refreshed automatically with the help of live reload.

If there is a scenario where we are building web application with only spring and spring MVC without spring boot, then dev tools may not be an option for you.

SPRINGBOOT DEVTOOLS

- The Spring Boot DevTools provides features like **Automatic restart** & **LiveReload** that make the application development experience a little more pleasant for developers.
- It can be added into any of the SpringBoot project by adding the below maven dependency.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
</dependency>
```

DevTools maintains 2 class loaders, one with classes that doesn't change and other one with classes that change. When restart needed it only reload the second class loader which makes restarts faster as well.

DevTools includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. LiveReload related browser extensions are freely available for Chrome, Firefox

DevTools triggers a restart when ever a build is triggered through IDE or by maven commands etc.

DevTools disables the caching options by default during development.

Repackaged archives do not contain DevTools by default.

With dev tools, the Spring boot web application will maintain two class loaders, one with the classes that were written by the developer and other with the in-built classes (For each of dependency there might by many number of classes will be loaded into different class loader).

When an restart is needed it only reloads the classes/files that are written by developer sue to which it makes restart faster as well.

There will no modifications made to pre-defined classes

- The Spring Boot DevTools is pleasant for development.
- It can be added into

we don't need these dev tools in the production environment, we would need these only during the local development..
 For that only spring boot dev tools is smart enough to not put the libraries related to dev tools. whenever you are generating a jar or whenever you're generating a war or packaging your entire web application for higher environment deployment.
 So in those scenarios, by default, your packaged archives will not have any jars or dependencies related to dev tools. So as a developer we don't have to do that.
 Spring boot automatically will take care of that disabling spring boot dev tools for higher environments

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

DevTools maintains 2 class loaders, one with classes that doesn't change and other one with classes that change. When restart needed it only reloads the second class loader which makes restarts faster as well.

DevTools includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. LiveReload related browser extensions are freely available for Chrome, Firefox

DevTools triggers a restart whenever a build is triggered through IDE or by maven commands etc.

DevTools disables the caching options by default during development.

Repackaged archives do not contain DevTools by default.

Live Reload :

This browser refresh automatically won't work.

We have to install an extension of live reload inside your browser.

So whenever this extension is installed and enabled inside your browser, my spring boot web application can send a signal to refresh the browser automatically without manual intervention.

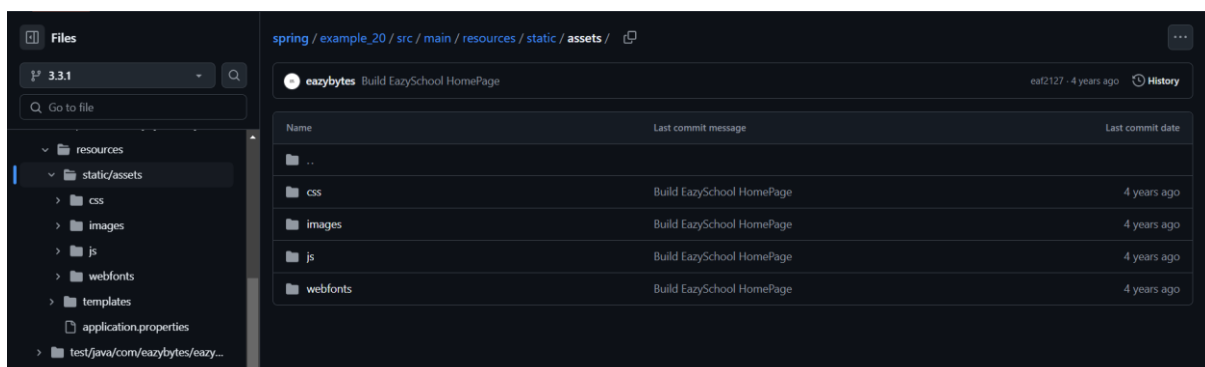
78: Implementation & Demo of Spring Dev Tools:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
```

DevTools is not going to help us in any coding related activities. Due to this we don't need this dependency during the compile time.
 Instead, we want this dependency to be available, once we start the application or once we start the server. That's why we need to mention this scope as runtime and optional as true.
 After making these changes, these DevTools is going to help us with automatic restarts whenever we make certain changes inside our web application.

79: Building Home Page of Eazy School Web Application

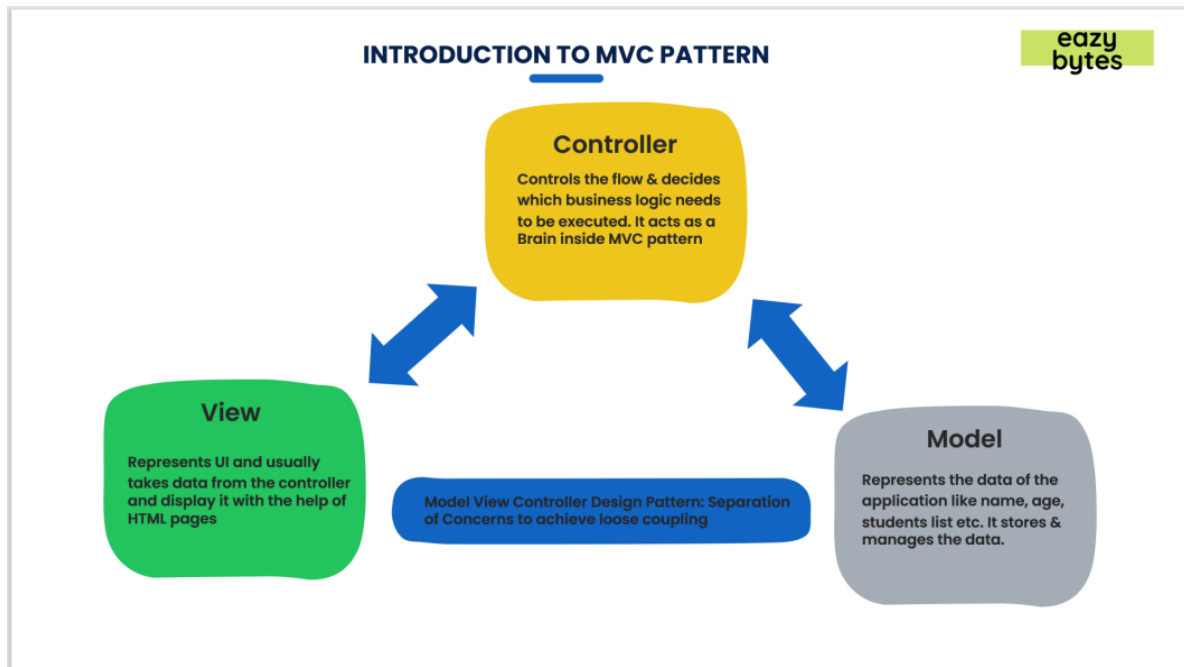
For Any static files Spring Boot will go and look in the static folder in this case we have placed all the required files in the assets folder.



From the screenshot we could see that all the static files were placed under the static folder.

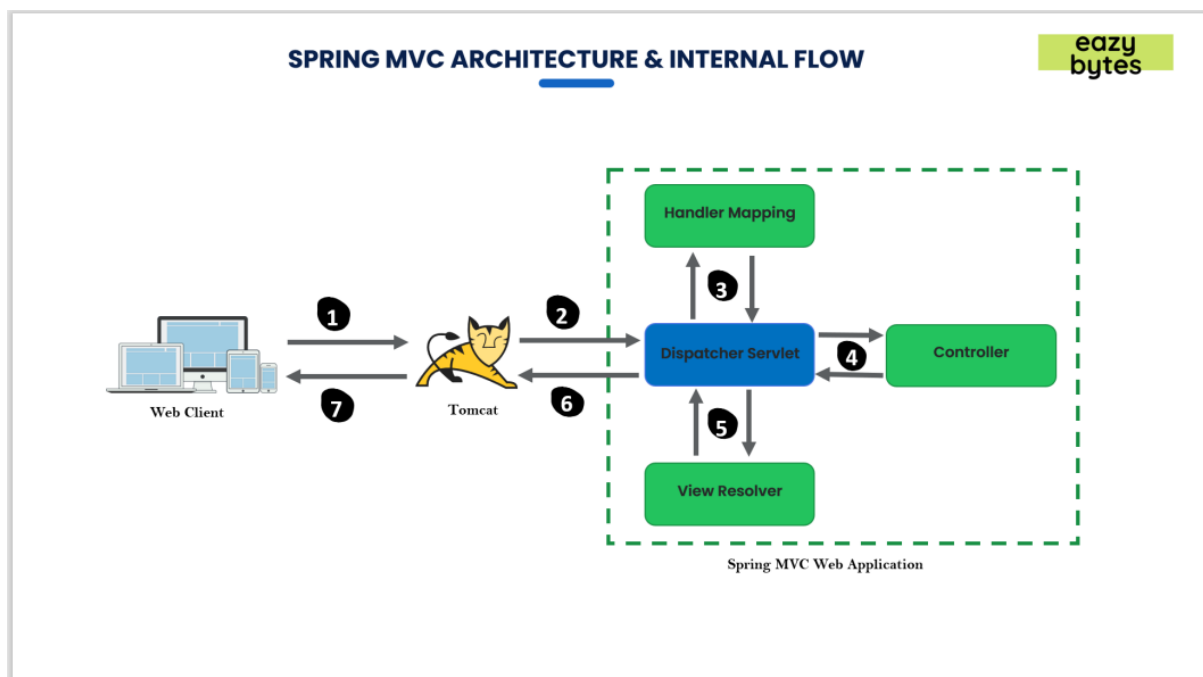
80.Check Eazy School Code.

81:Deep dive of Spring MVC internal Architecture:



In the early 2000, everyone used to develop a web application with the help of servlet, HTML and Java JDBC code, and they used to club everything wherever they want, like inside JSP pages. They used to write a lot of business logic, inside servlets they used to write a HTML code. So with that there is no proper segregation between the business logic, UI code and the data layer.

The primary purpose of the MVC pattern is to achieve loose coupling between the components that you have inside your web application, like view, backend code and persistent code.



SPRING MVC ARCHITECTURE & INTERNAL FLOW

easy bytes

- 1 Web Client makes HTTP request
- 2 Servlet Containers like Tomcat accepts the HTTP requests and handovers the Servlet Request to **Dispatcher Servlet** inside Spring Web App.
- 3 The Dispatcher Servlet will check with the **Handler Mapping** to identify the controller and method names to invoke based on the HTTP method, path etc.
- 4 The Dispatcher Servlet will invoke the corresponding controller & method. After execution, the controller will provide a view name and data that needs to be rendered in the view.
- 5 The Dispatcher Servlet with the help of a component called **View Resolver** finds the view and render it with the data provided by the controller.
- 6 The Servlet Container or Tomcat accepts the Servlet Response from the Dispatcher servlet and convert the same to HTTP response before returning to the client.
- 7 The browser or client intercepts the HTTP response and display the view, data etc.

Spring MVC Architecture and Internal Flow:

Once Tomcat receives that HTTP request, it will convert that request from HTTP request to the servlet request and hand over to the servlet inside Spring MVC, which is the Dispatcher Servlet

Dispatcher Servlet is also known as Front Controller.

Handler mapping again is one of the component inside Spring MVC framework, which will load all the configurations that we have done inside spring application and have it ready that way. Whenever Dispatcher Servlet is asking for certain information, it will provide with the controllers configurations that we have done.

Whenever my controller layer return a view name in this case home.html and if there is a scenario where I'm also sending some dynamic data with the help of model, it is the job of the view resolver to compile all the dynamic information to pass that all the dynamic information present inside my template files like home.html.

Once my ViewResolver done all the job of resolving the dynamic content, populating the values inside template and generating a proper HTML file, it will give the same information to the DispatcherServlet, and now my DispatcherServlet have a complete view details information received from the ViewResolver and the same information it will hand over to the Tomcat in the format of servlet response.

82: Separation of Header and Footer Using Thymeleaf replace tag:

Project Explorer

- ComponentExample (Spring_Code master)
 - Ex_12_AutoWire_Class_Fields (Spring_Code master)
 - Ex_13_AutoWire_Setter_method (Spring_Code master)
 - Ex_14_AutoWire_With_Constructor (Spring_Code master)
 - Ex_17_Singleton_Bean_scope (Spring_Code master)
 - Ex_18_Easy_Jazy_Instatiation (Spring_Code master)
 - Ex_19_Photo_Type_Scope (Spring_Code master)
 - Ex_20_Around_Advice_Example (Spring_Code master)
 - Ex_21_Before_Advice (Spring_Code master)
 - Ex_22_After_Throwing_Returning (Spring_Code master)
 - Ex_23_Annotation_Advice (Spring_Code master)
 - Ex_15_Auto_wiring_MultipleBeans (Spring_Code master)
 - Ex_16_Assignment (Spring_Code master)
 - Example_24 (boot) (Spring_Code master)
 - Example_25_Multiple_Mappings (boot) (Spring_Code master)
 - Example_26_Dynamic_Web_Page_Thymeleaf (boot) (Spring_Code master)
 - Example_27_Dev_Tools (boot) (devtools) (Spring_Code master)
 - Example_28_George_School_Home_Page (boot) (devtools) (Spring_Code master)
 - Example_29_Separation_of_Header_Footer (boot) (devtools)
- src/main/java
 - static
 - templates
 - footer.html
 - header.html
 - home.html
- application.properties
- src/test/java
- IRE System Library (Javase-21)
- Maven Dependencies
- src
 - main
 - test

application.html

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="utf-8">
5 </head>
6 <body>
7 <div th:replace="header">
8 </div>
9 <div th:replace="footer">
10 </div>
11 </body>
12 </html>

```

Including the Thymeleaf

Basically we have moved header and footer to different files so that we can have header and footer in all the files. Rather than writing the same code again. We can use the replace tag and get the header and footer content to the current webpage.

In this header element will be replaced with the content of header.html

Here we could see we have moved the header and footer tag to separate file and we will get them into current file using the thymeleaf replace tag

application.html

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="utf-8">
5 </head>
6 <body>
7 <div th:replace="header">
8 </div>
9 <div th:replace="footer">
10 </div>
11 </body>
12 </html>

```

George School

Here we gave "homePage" as path. So it will check the controller for the requestMapping and return the view corresponding to it.

When ever we click on George School it will going to HomeController and matches the Request Mapping and return the view corresponding to it.

We have set the Mapping to homePage and it will be going to HomeController RequestMapping and returns view corresponding to it(In this case home.html) will be view that will be returned by HomeController.

Now we will be adding the courses static webpage for courses(courses.html) to the templates . As this courses is part of header we will link that courses with corresponding controller and from controller it will be linked to static page

Consider a scenario where our controller will not have much business logic it will be going to display some static web Pages .In that we can use the below way rather than defining entire controller. We can create controller as well there would be no issue.

Do you know, we can register view controllers that create a direct mapping between the URL and the view name using the ViewControllerRegistry. This way, there's no need for any Controller between the two.

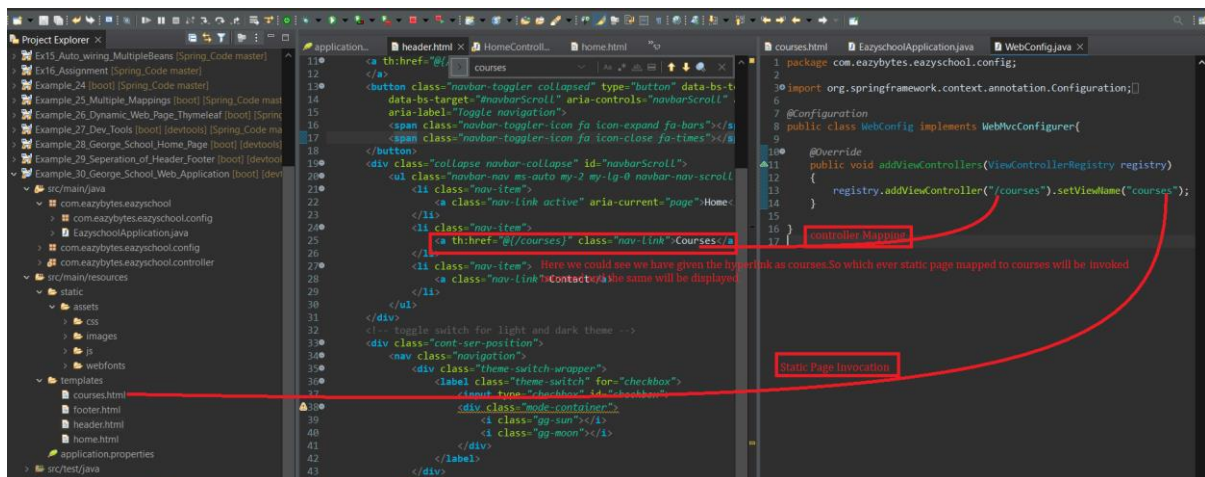
We can define an Configuration class

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController( urlPathOrPattern: "/courses").setViewName("courses");
        registry.addViewController( urlPathOrPattern: "/about").setViewName("about");
    }
}
```

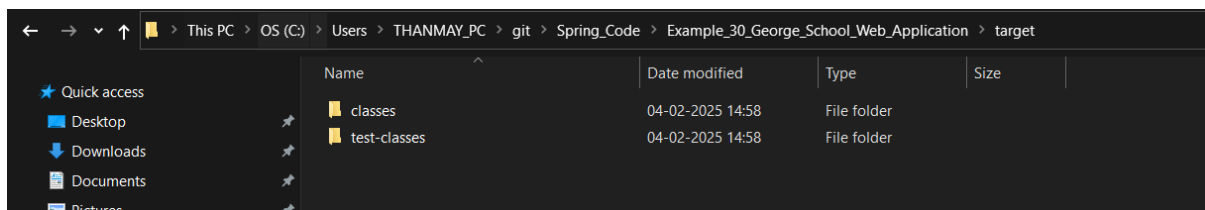
This will be re-directed to courses.html

This will be mapped to about.html static webpage



84: Resolving Build & cache issues inside Maven Projects:

Delete all the generated files under Target folder so that Build will happen again. Try incognito mode to avoid cache issues.



85: Building About Page of Eazy School Web Application

Ex_30

Ex_30

[illegible]

ModelAndView is a class available inside Spring MVC, which will help you to club both your model data and your view information.

Like if there is a scenario where you want to send some data to the UI along with the view name, then you can use the ModelAndView.

Model interface which will support sending the data and receiving the data between UI and backend code.

89:Submit Information from Contact Page Using POJO Object

The name present in the Front end and the POJO class should be matching other wise the mapping will not happen correctly.

Model class is similar to Form class

Taking the input from the POJO class and Saving details. --Check this commit for the Changes in Ex_30

90:Define actions for all the links in Home and Footer Page

On Click of Admission Now re-direct to contact Page:

```
<a th:href="@{/contact}" class="btn btn-style mt-lg-5 mt-4">Admission Now</a>
```

On Click of Apply Now re-direct to contact Page:

```
<a th:href="@{/contact}" class="btn btn-style mt-5">Apply Now</a>
```

On Click of Browse more courses re-direct to courses Page

```
<a th:href="@{/courses}" class="btn btn-style btn-style-secondary mt-sm-3">Browse more courses</a>
```

On Click of Join for free re-direct to contact Page:

```
</li>
<li><a th:href="@{/contact}" class="btn btn-style btn-style-2 mt-lg-0 mt-3">Join for free</a>
</li>
```

90.Building Holidays Page of Eazy School Web Application

Eazy Bytes Example_21

