

Programowanie Aplikacyjne (PAP)

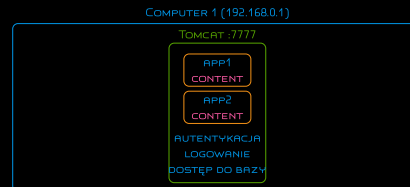
Web

Julian Myrcha
Institute of Computer Science
26.02.2025

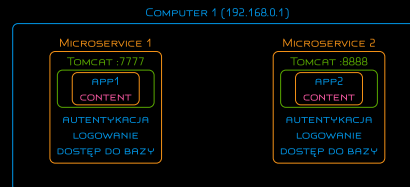


- W standardowym podejściu mamy monolityczne serwery hostujące wiele aplikacji

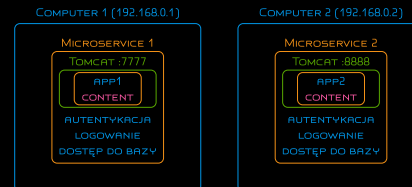
- podejście tradycyjne
- w jednym miejscu zarządzanie użytkownikami i konfiguracja środowiska



- W standardowym podejściu mamy monolityczne serwery hostujące wiele aplikacji
- Popularność zyskuje podejście mikroserwisów - oddzielnych programów realizujących niezależnie części funkcjonalności
- podział na mniejsze części ułatwia testowanie
- biblioteki i narzędzia ułatwiające konfigurację



- W standardowym podejściu mamy monolityczne serwery hostujące wiele aplikacji
- Popularność zyskuje podejście mikroservisów - oddzielnych programów realizujących niezależnie części funkcjonalności
- podział na mniejsze części ułatwia testowanie
- biblioteki i narzędzia ułatwiające konfigurację
- można łatwo tworzyć rozwiązania rozproszone



- przeglądarka analizuje to co dostała(zawartość) - i w zależności od tego inaczej wyświetla

plik html - wyświetla wynik analizy pliku

plik tekstowy - wyświetla zawartość pliku

```
1 jmy@bardex:$ telnet localhost 8080
2 Trying 127.0.0.1...
3 Connected to localhost.
4 Escape character is '^]'.
5 GET /FirstWebApp/first HTTP/1.1
6 HOST: localhost
7
8 HTTP/1.1 200 OK
9 Server: GlassFish Server Open Source Edition 4.1.1
10 X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.1.1 Java/Oracle Corporation/1.8)
11 Date: Fri, 21 Oct 2016 14:39:03 GMT
12 Content-Length: 23
13
14 Served at: /FirstWebAppConnection closed by foreign host.
15 jmy@bardex:$
```

- przeglądarka analizuje to co dostała(zawartość) - i w zależności od tego inaczej wyświetla

plik html - wyświetla wynik analizy pliku

plik tekstowy - wyświetla zawartość pliku

```
1 jmy@bequiet ~
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3 >
4 <HTML>
5 <HEAD>
6   <META http-equiv=Content-Type content="text/html; charset=iso-8859-2">
7   <link rel="stylesheet" type="text/css" href="/galera.css">
8   <TITLE>Serwer galera.ii.pw.edu.pl</TITLE>
9 </HEAD>
10
11 <BODY>
12 <CENTER>
13 ...
14 </HTML>
```

Przykład strony z formą html-ową:

```
1  <!DOCTYPE html>
2  <html>
3  <head><meta charset="UTF-8">
4      <title>EGUI</title>
5  </head>
6  <body>
7  <h2>Formy</h2>
8
9  <form action="first">
10     Imie:<br>
11     <input type="text" name="firstname" value="Jas">
12     <br>
13     Nazwisko:<br>
14     <input type="text" name="lastname" value="Fasola">
15     <br><br>
16     <input type="submit" name = 'btn' value="Wyslij">
17     <input type="submit" name = 'btn' value="Pomin">
18 </form>
19
20 <p>jak nacisniesz przycisk "Wyslij", dane zostana przeslane do strony "first".</p>
21 </body>
22 </html>
```

Do serwera idą dane:

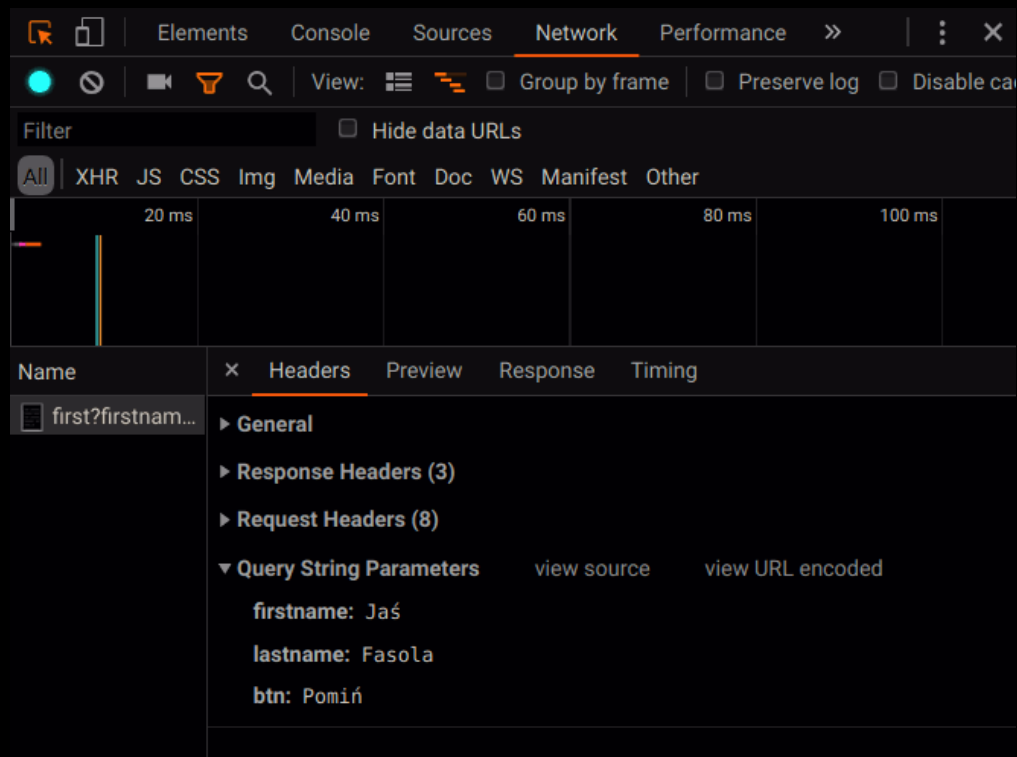
firstname -Jaś

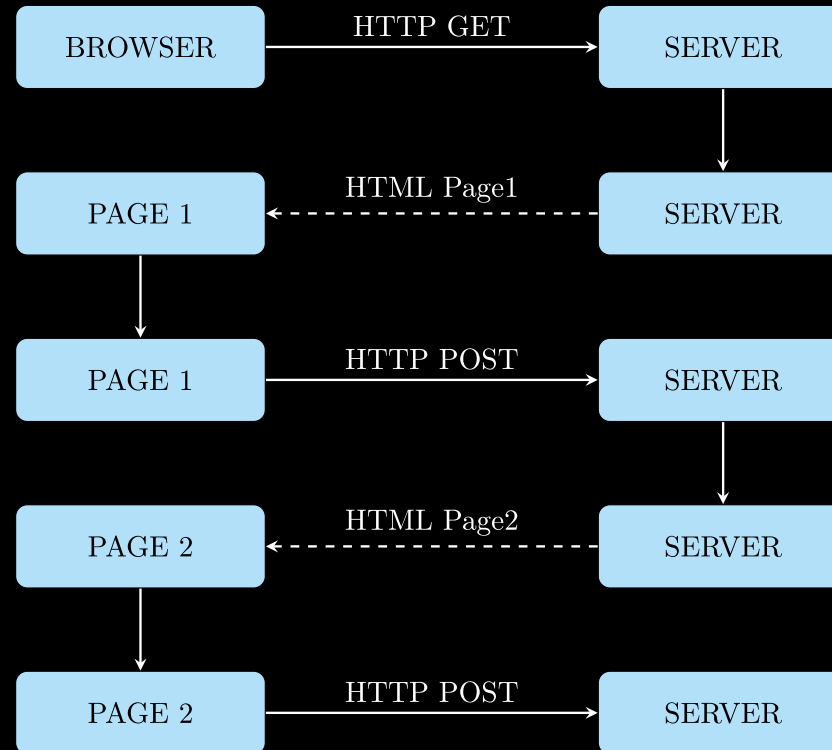
lastname -Fasola

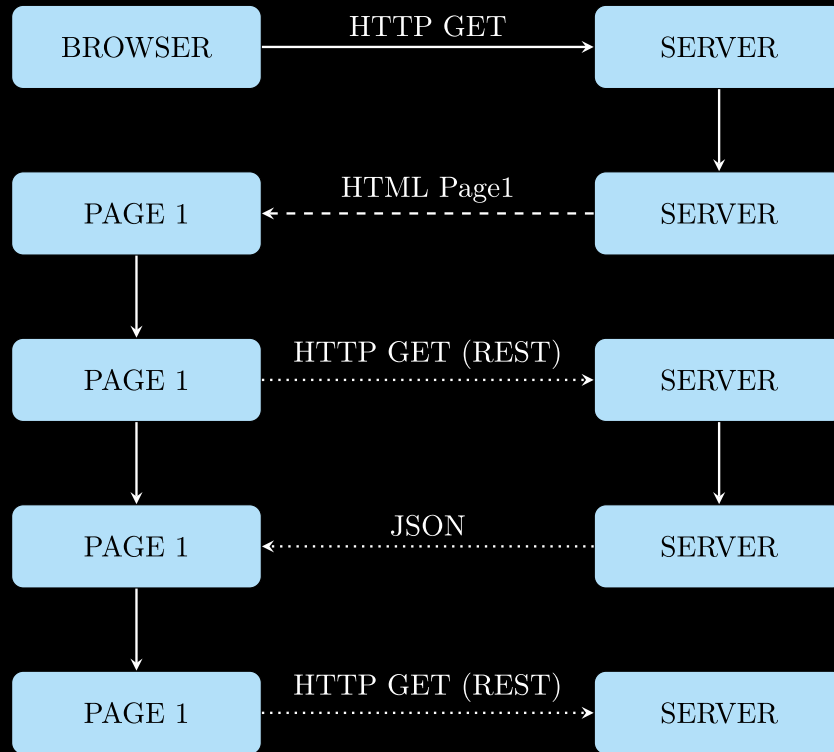
btn -Wyślij

Dla elementów typu submit w danych występuje tylko naciśnięty przycisk!

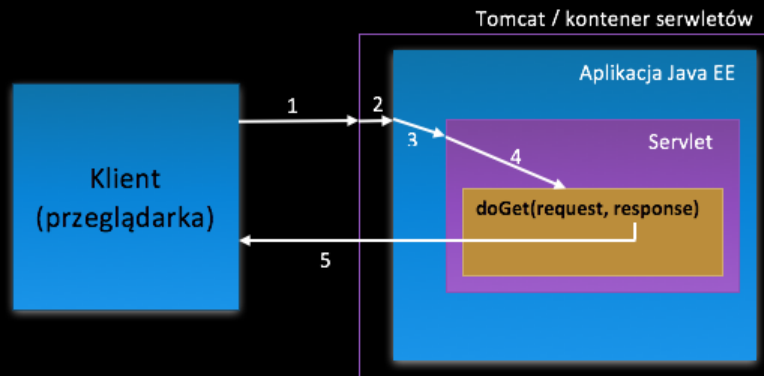
podglądanie parametrów: (Ctrl-Shift-C w chrome)



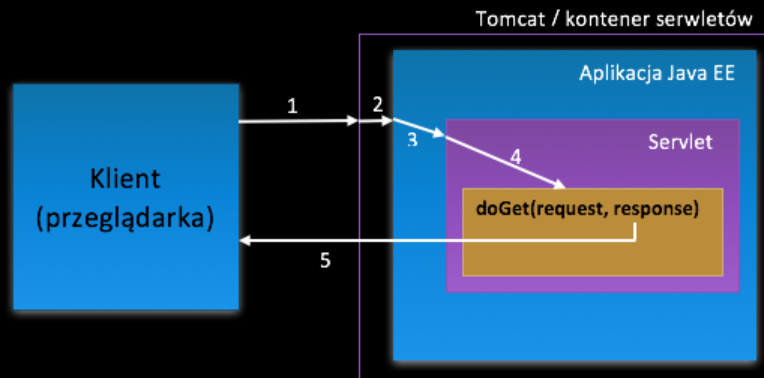




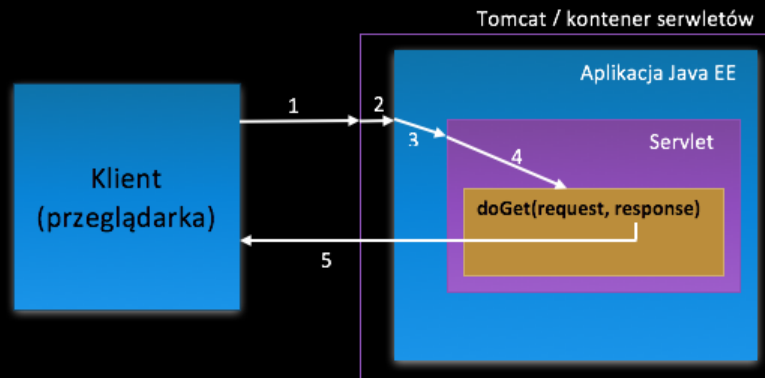
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- cykl życia serwletu:
 - utworzenie przez konstruktor
 - wywołanie metody `init()`
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - wywołanie metody `destroy()`



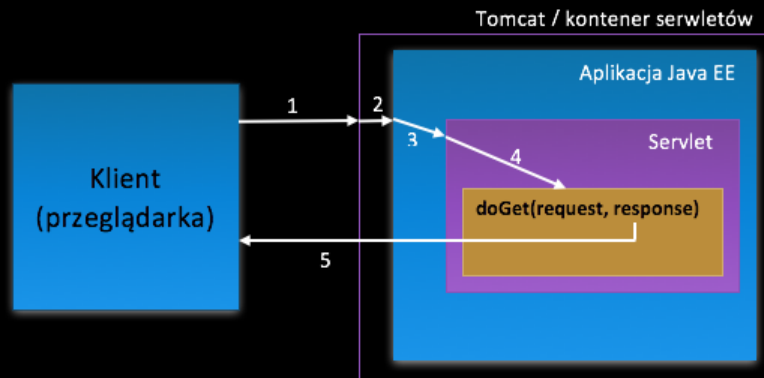
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- **cykl życia serwletu:**
 - utworzenie przez konstruktor
 - wywołanie metody `init()`
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - wywołanie metody `destroy()`



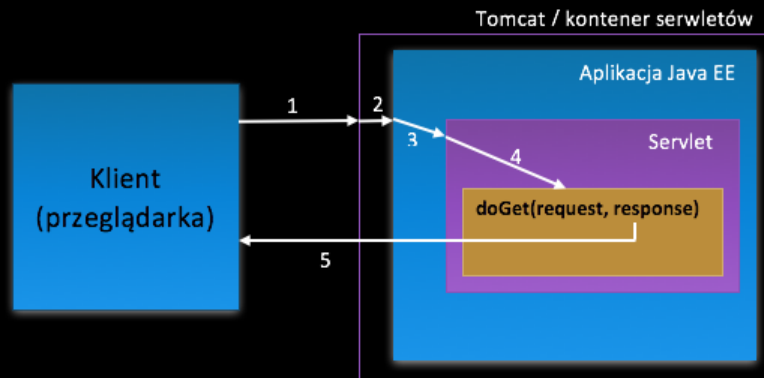
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- cykl życia serwletu:
 - **utworzenie przez konstruktor**
 - wywołanie metody `init()`
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - wywołanie metody `destroy()`



- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- cykl życia serwletu:
 - utworzenie przez konstruktor
 - **wywołanie metody `init()`**
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - wywołanie metody `destroy()`



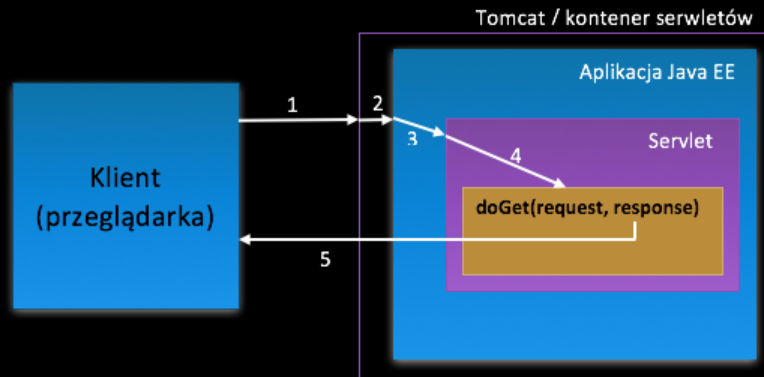
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- cykl życia serwletu:
 - utworzenie przez konstruktor
 - wywołanie metody `init()`
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - wywołanie metody `destroy()`



serwlet będzie współdzielony pomiędzy wszystkich użytkowników naszej aplikacji

- wielowątkowość

- serwlet za pomocą odpowiednich metod tworzy odpowiedź
 - `doGet()`, `doPost()`, `doTrace()`, `doHead()`, `doPut()`, `doOptions()`
- cykl życia serwletu:
 - utworzenie przez konstruktor
 - wywołanie metody `init()`
 - wielokrotne wywoływanie metod `doGet`, `doPost` itp.
 - **wywołanie metody `destroy()`**



- wybieramy `maven-archetype-webapp` w wersji 1.4

- wybieramy `maven-archetype-webapp` w wersji 1.4
`groupId` : pw.atj

- wybieramy `maven-archetype-webapp` w wersji 1.4
`groupId` : pw.atj
`artifactId` : webapp

- wybieramy `maven-archetype-webapp` w wersji 1.4
`groupId` : pw.atj
`artifactId` : webapp
`destination folder` : /home/user/atj

- wybieramy `maven-archetype-webapp` w wersji 1.4
`groupId` : pw.atj
`artifactId` : webapp
`destination folder` : /home/user/atj
`version` :1.0

- wybieramy `maven-archetype-webapp` w wersji 1.4
`groupId` : `pw.atj`
`artifactId` : `webapp`
`destination folder` : `/home/user/atj`
`version` : `1.0`
`Y` : `y`

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:

```
1  <properties>
2      ...
3      ↪ <jakartaee-api.version>9.1.0</jakartaee-api.version>
4  </properties>
5  <dependencies>
6      ...
7      <dependency>
8          <groupId>jakarta.platform</groupId>
9          <artifactId>jakarta.jakartaee-api</artifactId>
10         <version>${jakartaee-api.version}</version>
11         <scope>provided</scope>
12     </dependency>
13 </dependencies>
```

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:

```
1 package pw.atj;
2 import java.io.IOException;
3 import java.io.PrintWriter;
4 import jakarta.servlet.ServletException;
5 import jakarta.servlet.annotation.WebServlet;
6 import jakarta.servlet.http.HttpServlet;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9
10 @WebServlet("/helloWorld")
11 public class HelloWorld extends HttpServlet {
12     private static final long serialVersionUID = 1L;
13     public HelloWorld() {
14         super();
15     }
16     protected void doGet(HttpServletRequest request,
17         HttpServletResponse response)
18         throws ServletException, IOException {
19         String name = request.getParameter("name").trim();
20         response.setContentType("text/html");
21         PrintWriter out = response.getWriter();
22         out.print("<h2>Hello " + name + "</h2>");
23         out.close();
24     }
25     protected void doPost(HttpServletRequest request,
26         HttpServletResponse response)
27         throws ServletException, IOException {
28         // TODO Auto-generated method stub
29         doGet(request, response);
30     }
31 }
```

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloServlet.java`
- **dodajemy plik** `src/main/java/pw/atj/ApiServlet.java`

```
1 package pw.atj;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5 import jakarta.servlet.ServletException;
6 import jakarta.servlet.http.HttpServlet;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9
10 public class ApiServlet extends HttpServlet {
11
12     private static final long serialVersionUID = 1L;
13
14     public void doGet(HttpServletRequest request,
15                       HttpServletResponse response)
16         throws ServletException, IOException {
17
18         response.setContentType("application/json");
19         PrintWriter out = response.getWriter();
20         out.println("{ \"message\": \"Hello World!\" }");
21     }
22 }
```

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloServlet.java`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:

```
1 <web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3
4   ↪   xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
5   https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
6   version="5.0">
7
8   <display-name>Archetype Created Web
9   ↪   Application</display-name>
10
11   <servlet>
12     <servlet-name>ApiServlet</servlet-name>
13     <servlet-class>pw.atj.ApiServlet</servlet-class>
14   </servlet>
15   <servlet-mapping>
16     <servlet-name>ApiServlet</servlet-name>
17     <url-pattern>/api/hello</url-pattern>
18   </servlet-mapping>
19 </web-app>
```

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiHelloWorld.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:

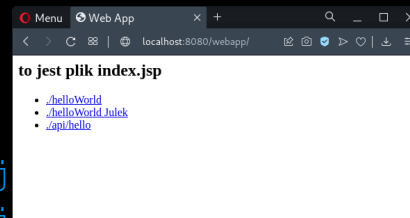
```
1 <head>
2   <title>Web App</title>
3 </head>
4
5 <body>
6   <h2>to jest plik index.jsp</h2>
7   <ul>
8     <li><a href='./helloWorld'> ./helloWorld</a></li>
9     <li><a href='./helloWorld?name=Julek'> ./helloWorld
10       ↪ Julek</a></li>
11     <li><a href='./api/hello'> ./api/hello</a></li>
12   </ul>
13 </body>
14 </html>
```

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy `src/main/webapp/WEB-INF/web.xml` plik
- modyfikujemy `src/main/webapp/index.jsp` plik
- `maven: package`

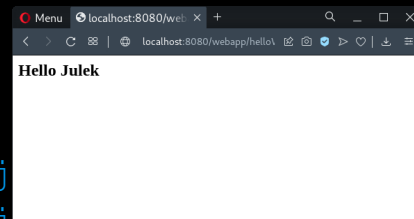
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy `src/main/webapp/WEB-INF/web.xml` plik
- modyfikujemy `src/main/webapp/index.jsp` plik
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`

- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- **dodajemy** **serwer** **Tomcat**
`/opt/apache/apache-tomcat-10.0.17`

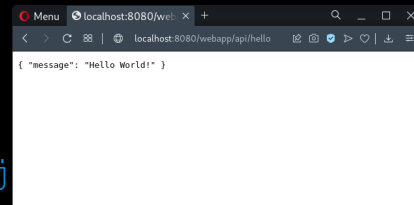
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`.
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- dodajemy serwer Tomcat
`/opt/apache/apache-tomcat-10.0.17`



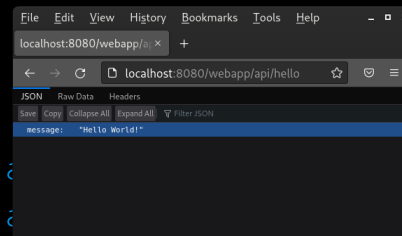
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`.
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- dodajemy serwer Tomcat
`/opt/apache/apache-tomcat-10.0.17`



- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- dodajemy serwer Tomcat
`/opt/apache/apache-tomcat-10.0.17`



- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`
- modyfikujemy `src/main/webapp/WEB-INF/web.xml` plik
- modyfikujemy `src/main/webapp/index.jsp` plik
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- dodajemy `server` Tomcat
`/opt/apache/apache-tomcat-10.0.17`



- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web.xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`
- `target:webapp: "Run on Tomcat Server"`
- dodajemy serwer Tomcat `/opt/apache/apache-tomcat-10.0.17`
- wyłączenie VS Code pozostawia włączony serwer Tomcat - trzeba go wyłączyć ręcznie

```
1 /opt/apache/apache-tomcat-10.0.17/bin/shutdown.sh
```

parametry czytane przez wszystkie serwlety

```
1 <context-param>
2   <param-name>myParam</param-name>
3   <param-value>the value</param-value>
4 </context-param>
```

Przykład odczytu:

```
1 ${initParam.myParam}
```

- Najlepiej za pomocą adnotacji
 - name nazwa servletu
 - value mapowanie servletu do ścieżki

```
1 package pl.edu.pw.atjtomcat;
2
3 import java.io.*;
4
5 import jakarta.servlet.http.*;
6 import jakarta.servlet.annotation.*;
7
8 @WebServlet(name = "helloServlet", value = "/hello-servlet",
9             initParams = {
10                 @WebInitParam(name = "message", value = "Hello World")
11             })
12 public class HelloServlet extends HttpServlet {
13     private String message;
14
15     public void init() {
16         message = getInitParameter("message");
17     }
18
19     public void doGet(HttpServletRequest request,
20                       HttpServletResponse response) throws IOException {
21
22         response.setContentType("text/html");
23         PrintWriter out = response.getWriter();
24         out.println("<html><body>");
25         out.println("<h1>" + message + "</h1>");
26         out.println("</body></html>");
27     }
28
29     public void destroy() {
30     }
31 }
```

- Można użyć wpisu w pliku web.config

```
1 <servlet>
2   <servlet-name>CalcServlet</servlet-name>
3   ↪ <servlet-class>pap.CalcServlet</servlet-class>
4   ↪ <init-param><param-name>saveDir</param-name>
5       ↪ <param-value>/home/jmy/Downloads</param-
6 </init-param>
7 <load-on-startup>1</load-on-startup>
8 </servlet>
```

- Najlepiej za pomocą adnotacji
 - `name` nazwa servletu
 - `value` mapowanie servletu do ścieżki

```
1 package pl.edu.pw.atjtomcat;
2
3 import java.io.*;
4
5 import jakarta.servlet.http.*;
6 import jakarta.servlet.annotation.*;
7
8 @WebServlet(name = "helloServlet", value = "/hello-servlet",
9             initParams = {
10                 @WebInitParam(name = "message", value = "Hello World")
11             })
12 public class HelloServlet extends HttpServlet {
13     private String message;
14
15     public void init() {
16         message = getInitParameter("message");
17     }
18
19     public void doGet(HttpServletRequest request,
20                       HttpServletResponse response) throws IOException {
21
22         response.setContentType("text/html");
23         PrintWriter out = response.getWriter();
24         out.println("<html><body>");
25         out.println("<h1>" + message + "</h1>");
26         out.println("</body></html>");
27     }
28
29     public void destroy() {
30     }
31 }
```

- Można użyć wpisu w pliku web.config

```
1 <servlet>
2   <servlet-name>CalcServlet</servlet-name>
3   ↪ <servlet-class>pap.CalcServlet</servlet-class>
4   ↪ <init-param><param-name>saveDir</param-name>
5       ↪ <param-value>/home/jmy/Downloads</param-
6   </init-param>
7   <load-on-startup>1</load-on-startup>
8 </servlet>
```

- Możemy przekazać parametry

```
1  @WebServlet(  
2      urlPatterns = "/imageUpload",  
3      initParams =  
4      {  
5          @WebInitParam(name = "saveDir", value = "D:/FileUpload"),  
6          @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")  
7      }  
8  )  
9  public class ImageUploadServlet extends HttpServlet {  
10  
11      public void doGet(HttpServletRequest request, HttpServletResponse response)  
12          throws IOException {  
13          String saveDir = getInitParameter("saveDir");  
14          String fileTypes = getInitParameter("allowedTypes");  
15  
16          PrintWriter writer = response.getWriter();  
17  
18          writer.println("saveDir = " + saveDir);  
19          writer.println("fileTypes = " + fileTypes);  
20      }  
21  }
```

- Filtry Servletowe to klasy java, które:
 - pozwalają przechwycić żądanie przed obsłużeniem przez servlet
 - pozwalają poprawić wynik przetwarzania
 - można je ustawić w łańcuch

```
1 // Import required java libraries
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.util.*;
6
7 // Implements Filter class
8 public class LogFilter implements Filter {
9     public void init(FilterConfig config) throws ServletException {
10
11         // Get init parameter
12         String testParam = config.getInitParameter("test-param");
13
14         //Print the init parameter
15         System.out.println("Test Param: " + testParam);
16     }
17
18     public void doFilter(ServletRequest request, ServletResponse response,
19         FilterChain chain) throws java.io.IOException, ServletException {
20
21         // Get the IP address of client machine.
22         String ipAddress = request.getRemoteAddr();
23
24         // Log the IP address and current timestamp.
25         System.out.println("IP " + ipAddress + ", Time " + new Date().toString());
26
27         // Pass request back down the filter chain
28         chain.doFilter(request, response);
29     }
30
31     public void destroy( ) {
```


- Wykorzystanie adnotacji:

```
1  @WebFilter("/admin")
2  public class MyFilter implements Filter {
3      @Override
4      public void init(FilterConfig filterConfig) throws ServletException {
5      }
6      @Override
7      public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
8          throws IOException, ServletException {
9          PrintWriter out = response.getWriter();
10
11          // This will print output on console
12          System.out.println(
13              "Before filter - Preprocessing before servlet");
14
15          // some authentication if required
16          chain.doFilter(request, response);
17
18          // This will print output on console
19          System.out.println(
20              "After servlet - Following code will execute after running the servlet - PostProcessing");
21      }
22      @Override
23      public void destroy() {
24      }
25  }
26 }
```

- Wykorzystanie adnotacji:

```
1 @WebFilter(  
2     urlPatterns = "/uploadFilter",  
3     initParams = @WebInitParam(name = "fileTypes", value = "doc;xls;zip;txt;jpg;png;gif")  
4 )  
5 public class UploadFilter implements Filter {  
6     // implements Filter's methods here...  
7 }
```

- Wykorzystanie adnotacji:
- Wykorzystanie pliku web.xml

```
1 public class LoginFilter implements Filter {
2     private static final long serialVersionUID = 1L;
3
4     public void init(FilterConfig filterConfig)
5         throws ServletException {
6
7     }
8
9     public void doFilter(ServletRequest request,
10        ServletResponse response, FilterChain chain)
11        throws IOException, ServletException {
12
13        response.setContentType("text/html");
14        PrintWriter out = response.getWriter();
15
16        //get parameters from request object.
17        String userName = request.getParameter("userName").trim();
18        String password = request.getParameter("password").trim();
19
20        //check for null and empty values.
21        if(userName == null || userName.equals("") ||
22            password == null || password.equals("")){
23            out.print("Please enter both username " +
24                "and password. <br/><br/>");
25            RequestDispatcher requestDispatcher =
26                request.getRequestDispatcher("/login.html");
27            requestDispatcher.include(request, response);
28        } //Check for valid username and password.
29        else if(userName.equals("jai") && password.equals("1234")){
30            chain.doFilter(request, response);
31        } else{
32            out.print("Wrong username or password. <br/><br/>");
33            RequestDispatcher requestDispatcher =
34                request.getRequestDispatcher("/login.html");
35            requestDispatcher.include(request, response);
36        }
37    }
```


- Wykorzystanie adnotacji:
- Wykorzystanie pliku web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://java.sun.com/xml/ns/javaee"
4 xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
7 id="WebApp_ID" version="2.5">
8
9   <servlet>
10     <servlet-name>WelcomeServlet</servlet-name>
11     <servlet-class>
12       com.w3spoint.business.WelcomeServlet
13     </servlet-class>
14   </servlet>
15
16   <servlet-mapping>
17     <servlet-name>WelcomeServlet</servlet-name>
18     <url-pattern>/WelcomeServlet</url-pattern>
19   </servlet-mapping>
20
21   <filter>
22     <filter-name>LoginFilter</filter-name>
23     <filter-class>
24       com.w3spoint.business.LoginFilter
25     </filter-class>
26   </filter>
27
28   <filter-mapping>
29     <filter-name>LoginFilter</filter-name>
30     <url-pattern>/WelcomeServlet</url-pattern>
31   </filter-mapping>
32
33   <welcome-file-list>
34     <welcome-file>login.html</welcome-file>
35   </welcome-file-list>
36
37 </web-app>
```




Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - plain old java object - wolność wyboru na czym to działa



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- **Modularne** - używamy tylko tej funkcjonalności której chcemy używać



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania
- **Spring Web Framework** - kolejna iteracja frameworka do aplikacji webowych



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania
- Spring Web Framework - kolejna iteracja frameworka do aplikacji webowych
- Centralizuje obsługę wyjątków zgłaszanych przez poszczególne technologie (JDBC, Hibernate, JDO)



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania
- Spring Web Framework - kolejna iteracja frameworka do aplikacji webowych
- Centralizuje obsługę wyjątków zgłaszanych przez poszczególne technologie (JDBC, Hibernate, JDO)
- Obsługa transakcji - niezależność od tego co pod spodem (czyli jedna baza albo JTA)



Najpopularniejszy framework do tworzenia aplikacji w Javie EE

- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania
- Spring Web Framework - kolejna iteracja frameworka do aplikacji webowych
- Centralizuje obsługę wyjątków zgłaszanych przez poszczególne technologie (JDBC, Hibernate, JDO)
- Obsługa transakcji - niezależność od tego co pod spodem (czyli jedna baza albo JTA)
- **SpringBoot** - konfigurator aplikacji w Springu

- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :



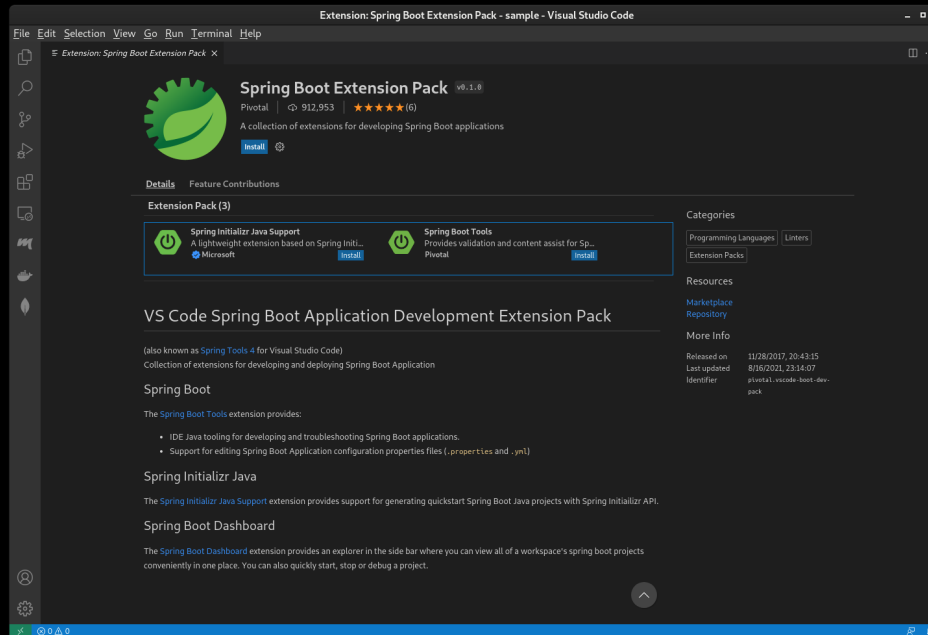
- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :
 - Java Development Kit (JDK)



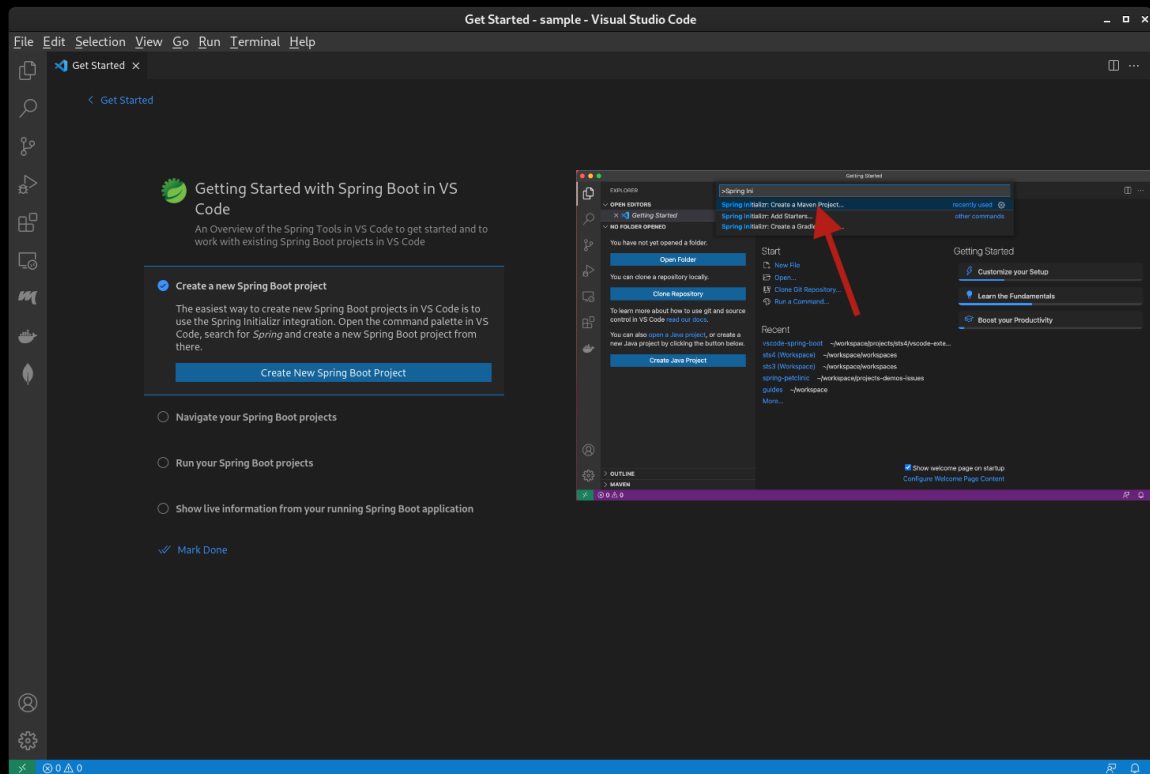
- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :
 - Java Development Kit (JDK)
 - **Extension Pack for Java**



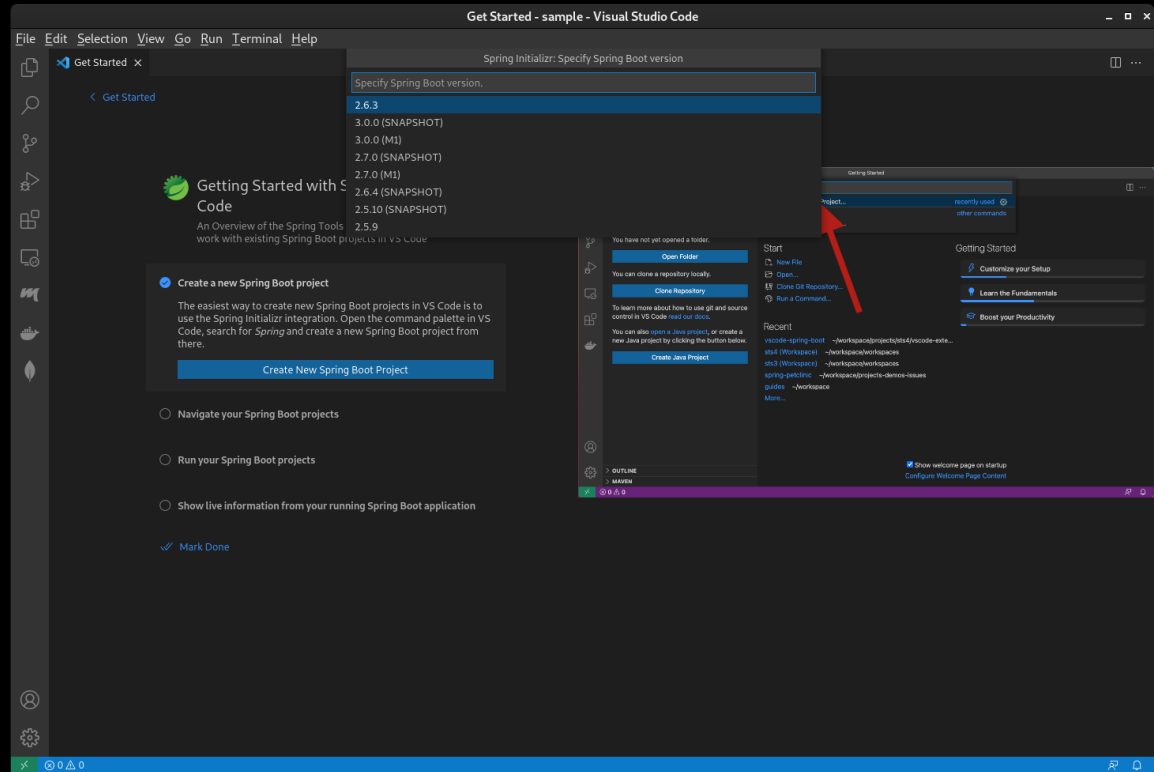
- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :
 - Java Development Kit (JDK)
 - Extension Pack for Java
 - **Spring Boot Extension Pack**



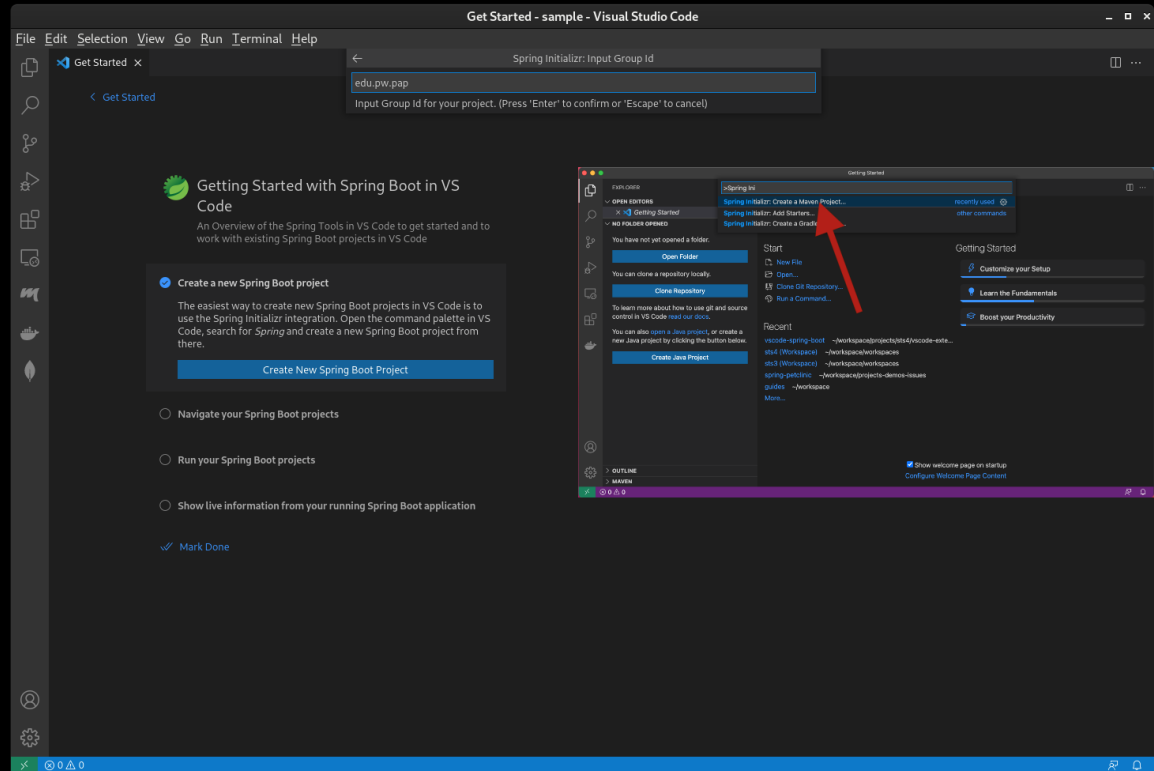
- wybranie czarodzieja



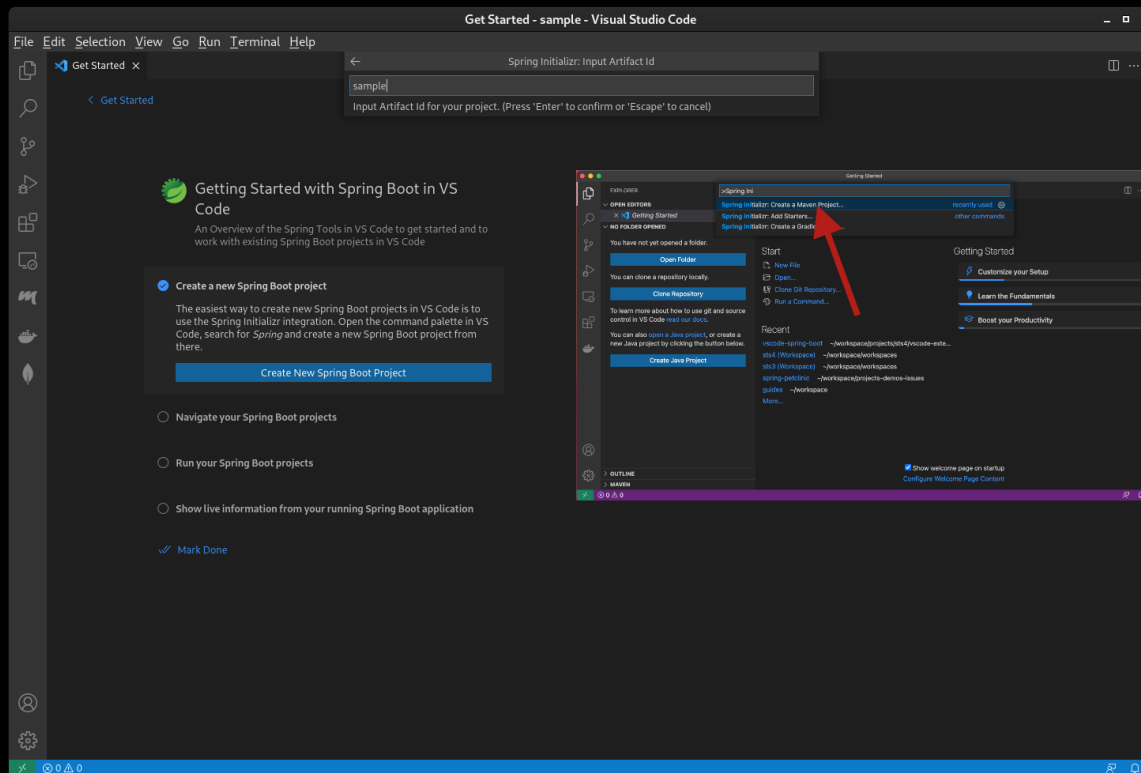
- wybranie czarodzieja
- wybranie wersji spring boota



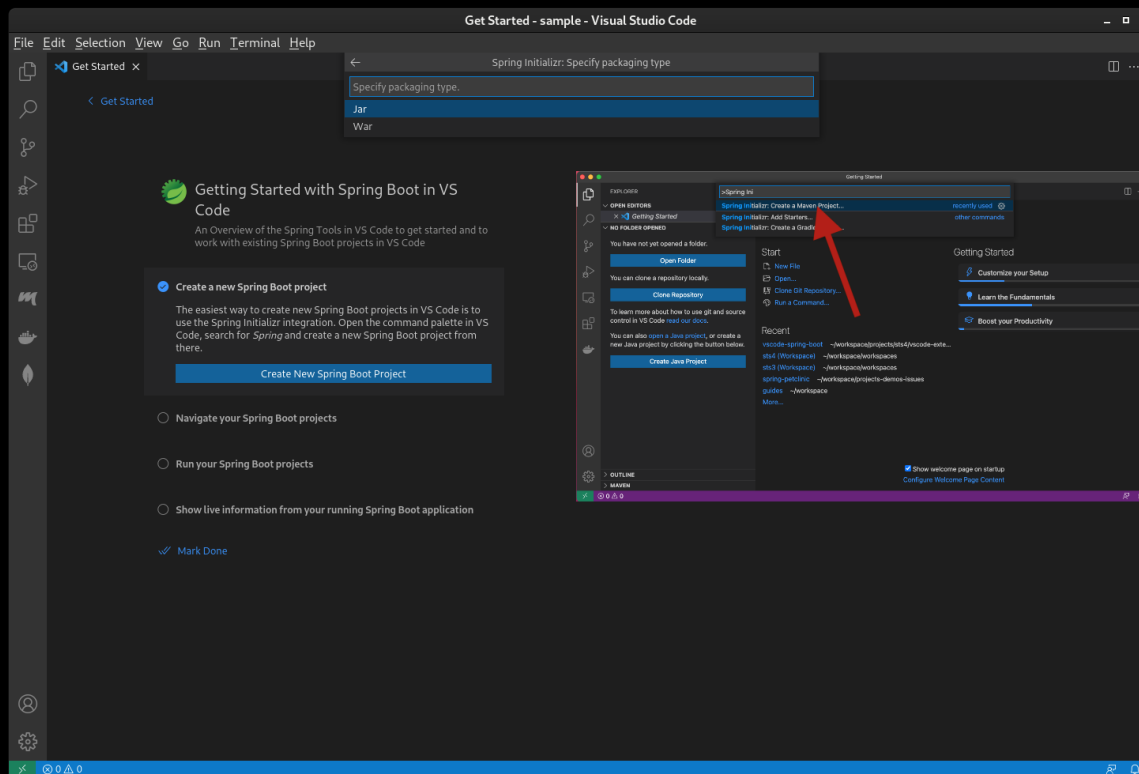
- wybranie czarodzieja
- wybranie wersji spring boota
- **maven group id**



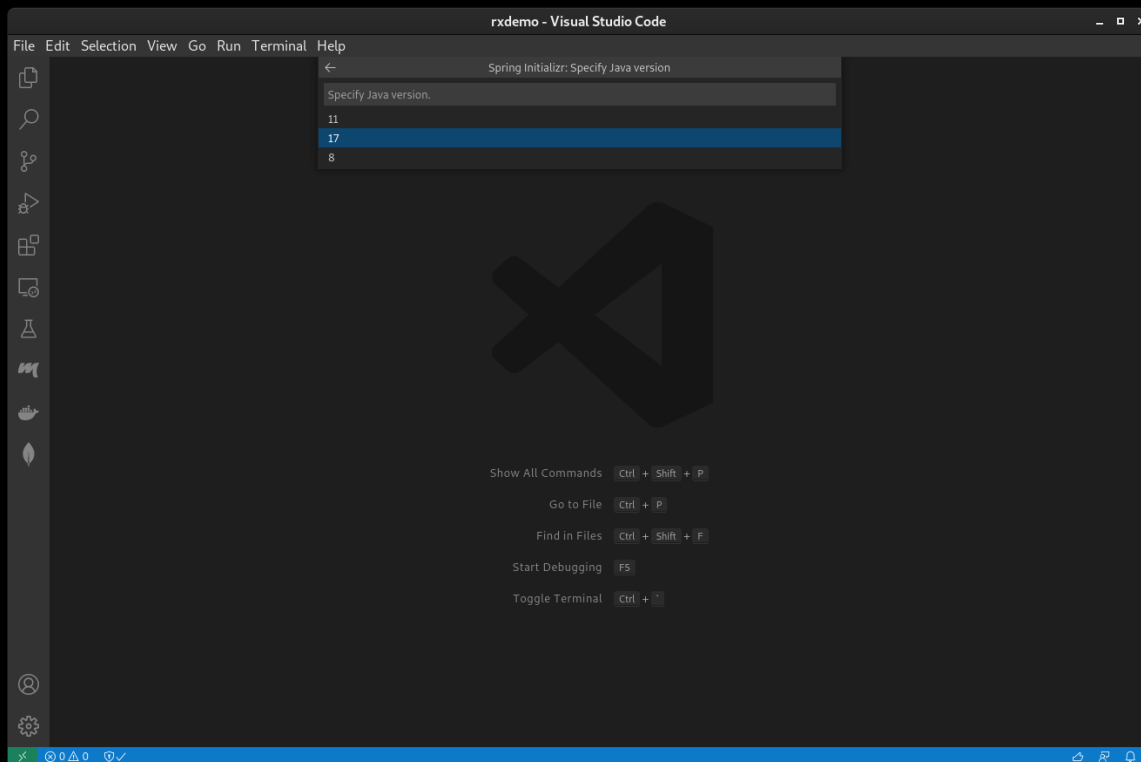
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- **maven artefact id**



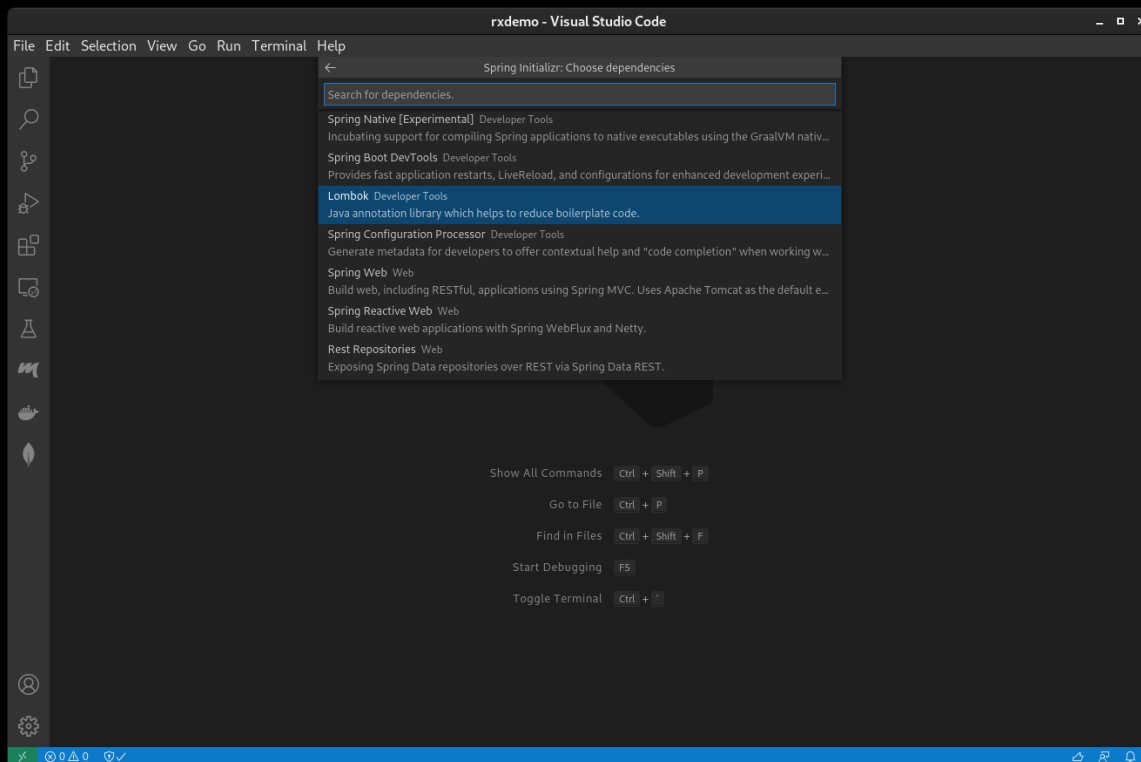
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



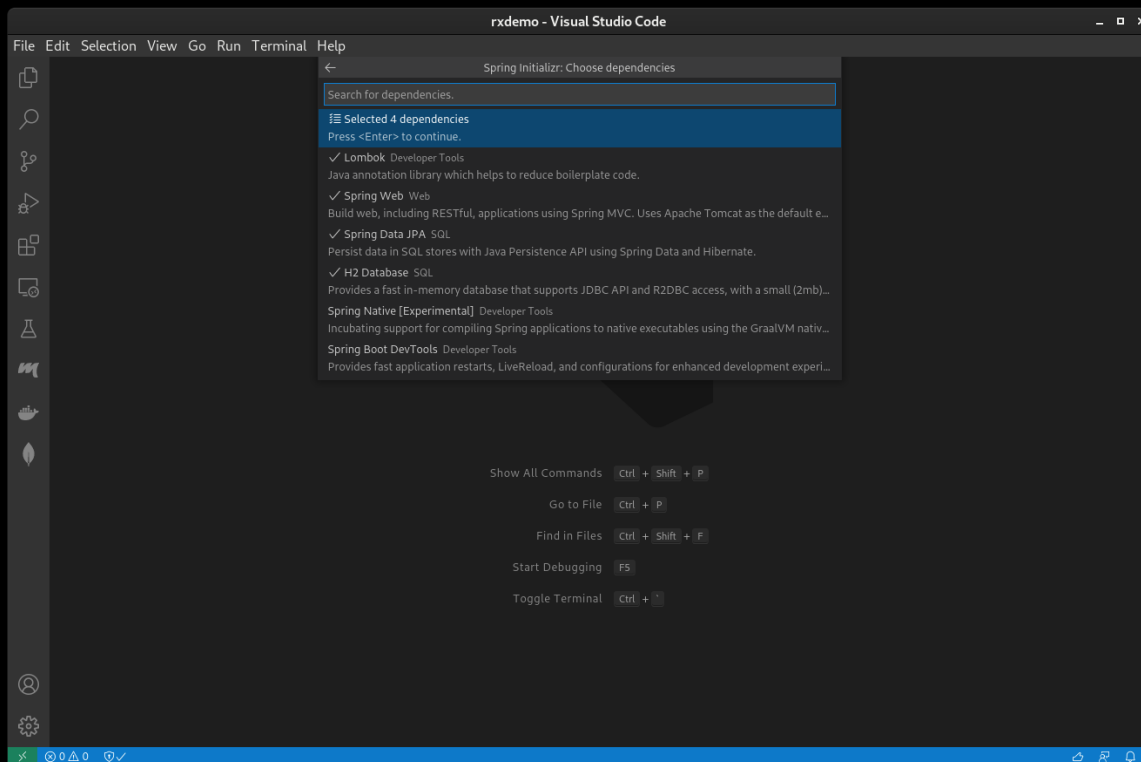
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



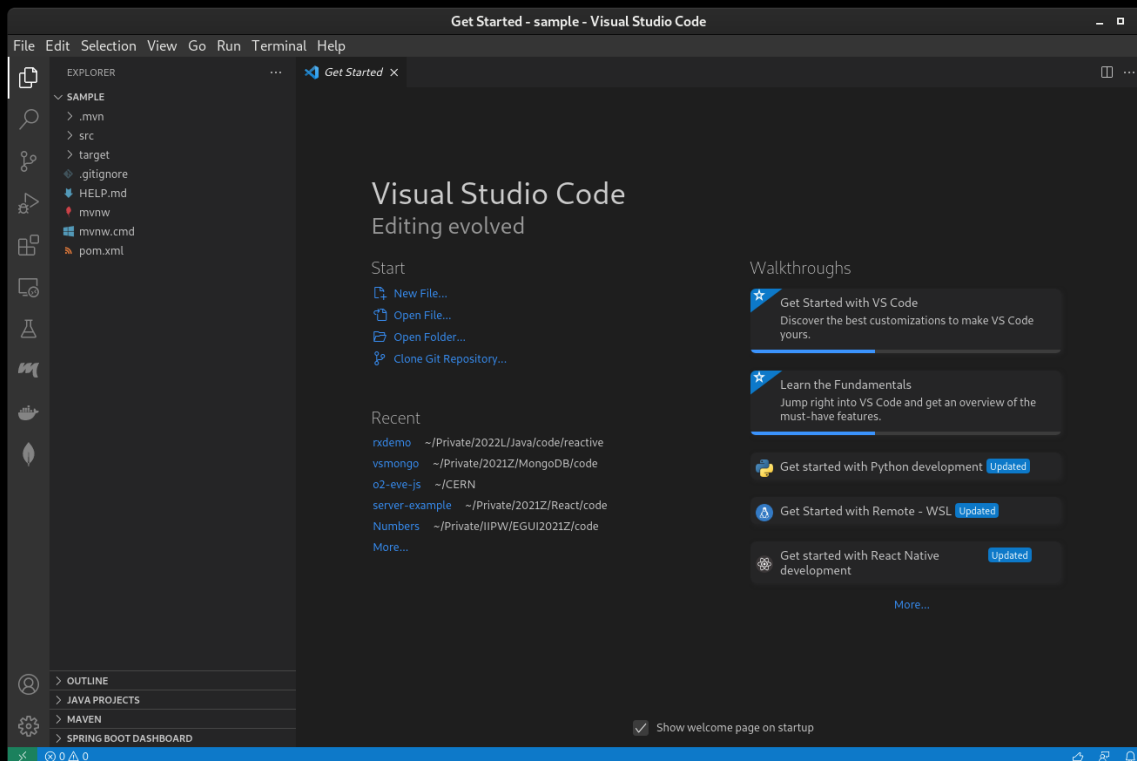
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



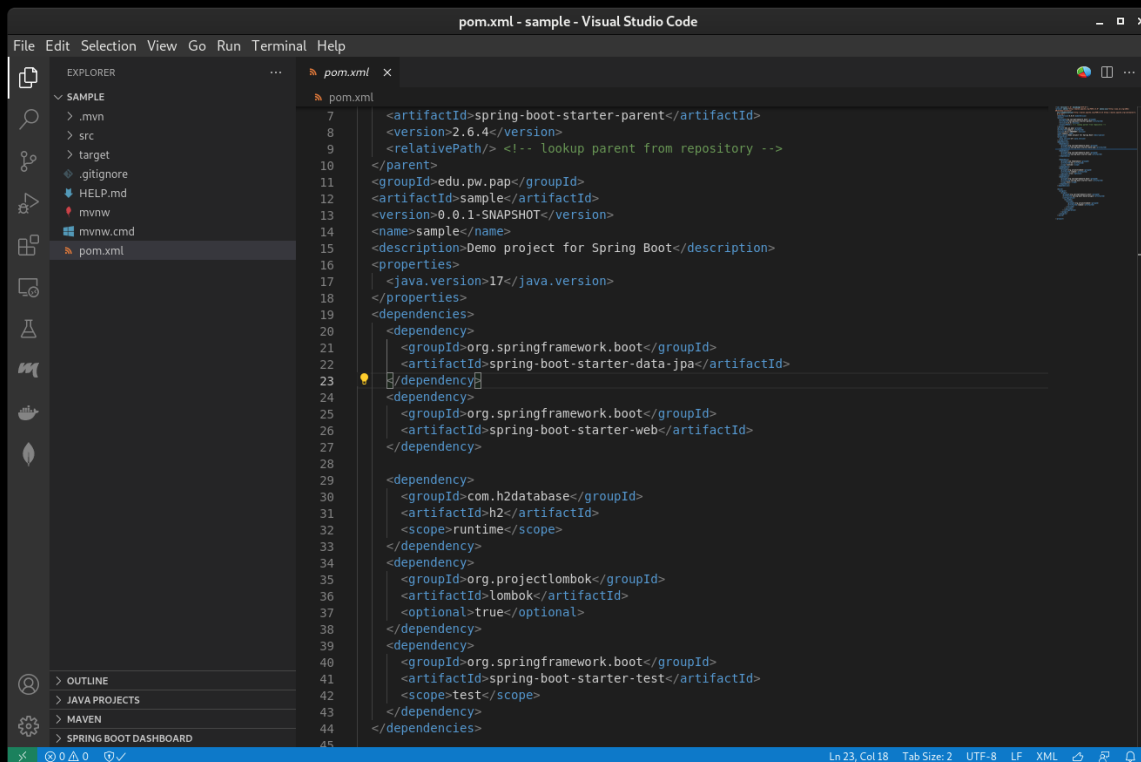
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



```
pom.xml - sample - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  SAMPLE
    > .mvn
    > src
    > target
    > .gitignore
    > HELP.md
    > mvnw
    > mvnw.cmd
    pom.xml
OUTLINE
  JAVA PROJECTS
  MAVEN
  SPRING BOOT DASHBOARD
pom.xml
  <?xml version='1.0' encoding='UTF-8'?>
  <!-- Parent POM -->
  <groupId>edu.pw.pap</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.6.4</version>
  <relativePath><!-- lookup parent from repository --></relativePath>
  </parent>
  <groupId>edu.pw.pap</groupId>
  <artifactId>sample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>sample</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

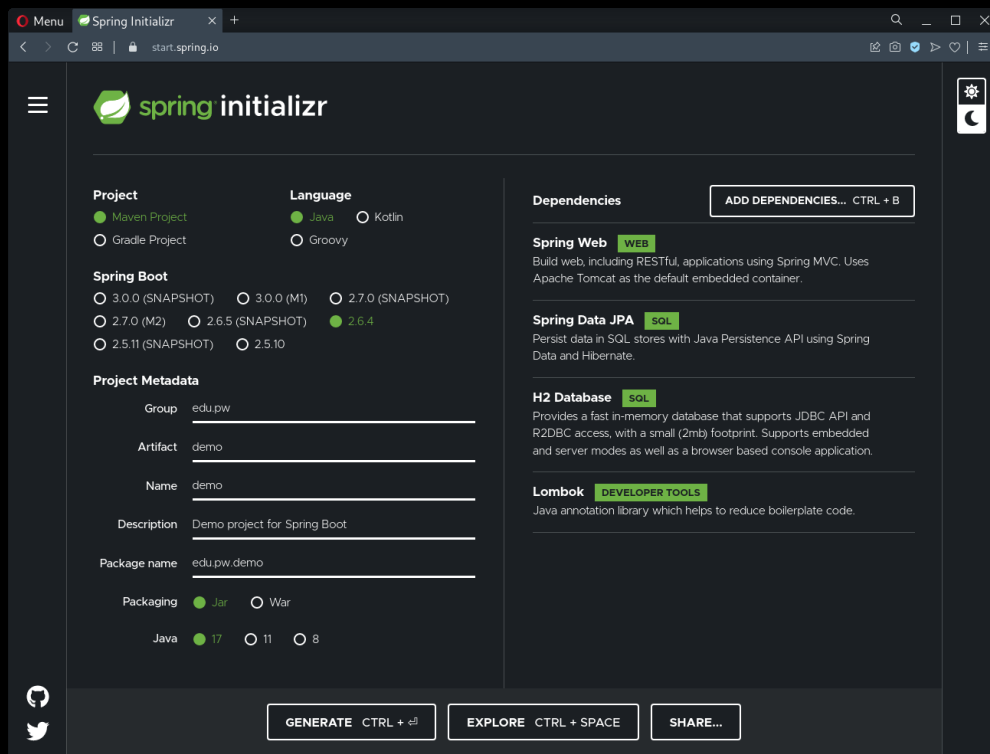
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację
- **pom.xml**

```
1 <dependencies>
2   <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-data-jpa</artifactId>
5   </dependency>
6   <dependency>
7     <groupId>org.springframework.boot</groupId>
8     <artifactId>spring-boot-starter-web</artifactId>
9   </dependency>
10
11   <dependency>
12     <groupId>com.h2database</groupId>
13     <artifactId>h2</artifactId>
14     <scope>runtime</scope>
15   </dependency>
16   <dependency>
17     <groupId>org.projectlombok</groupId>
18     <artifactId>lombok</artifactId>
19     <optional>true</optional>
20   </dependency>
21   <dependency>
22     <groupId>org.springframework.boot</groupId>
23     <artifactId>spring-boot-starter-test</artifactId>
24     <scope>test</scope>
25   </dependency>
26 </dependencies>
```

- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację
- pom.xml
- java

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class SampleApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SampleApplication.class, args);
11     }
12
13 }
```

- to samo można skonfigurować na stronie `start.spring.io`



The screenshot shows the Spring Initializr web application interface. The browser address bar displays `start.spring.io`. The page features a sidebar with a hamburger menu and social media icons (GitHub, Twitter). The main content area is divided into several sections:

- Project:** Includes radio buttons for `Maven Project` (selected) and `Gradle Project`.
- Language:** Includes radio buttons for `Java` (selected) and `Kotlin`, and `Groovy`.
- Spring Boot:** Includes radio buttons for various versions: `3.0.0 (SNAPSHOT)`, `3.0.0 (M1)`, `2.7.0 (SNAPSHOT)`, `2.7.0 (M2)`, `2.6.5 (SNAPSHOT)`, `2.6.4` (selected), and `2.5.11 (SNAPSHOT)`, `2.5.10`.
- Project Metadata:** Includes input fields for `Group` (edu.pw), `Artifact` (demo), `Name` (demo), `Description` (Demo project for Spring Boot), and `Package name` (edu.pw.demo).
- Packaging:** Includes radio buttons for `Jar` (selected) and `War`.
- Java:** Includes radio buttons for `17` (selected), `11`, and `8`.
- Dependencies:** Includes a button `ADD DEPENDENCIES... CTRL + B` and a list of selected dependencies:
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - H2 Database** (SQL): Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.

At the bottom, there are three buttons: `GENERATE CTRL + G`, `EXPLORE CTRL + SPACE`, and `SHARE...`.

- to samo można skonfigurować na stronie `start.spring.io`
- utworzony projekt pobieramy w postaci kompresowanego katalogu

The screenshot shows the Spring Initializr web application in a browser. The interface is dark-themed and organized into several sections:

- Project:** Includes radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for various versions: 3.0.0 (SNAPSHOT), 3.0.0 (M1), 2.7.0 (SNAPSHOT), 2.7.0 (M2), 2.6.5 (SNAPSHOT), **2.6.4** (selected), and 2.5.11 (SNAPSHOT), 2.5.10.
- Project Metadata:** A form with fields for:
 - Group:** `edu.pw`
 - Artifact:** `demo`
 - Name:** `demo`
 - Description:** `Demo project for Spring Boot`
 - Package name:** `edu.pw.demo`
 - Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
 - Java:** Includes radio buttons for **17** (selected), **11**, and **8**.
- Dependencies:** A section with a button **ADD DEPENDENCIES... CTRL + B** and a list of pre-selected dependencies:
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - H2 Database** (SQL): Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.

At the bottom, there are three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**. A notification in the top right corner indicates that `demo.zip` has been downloaded completely.

- tworzymy obiekt danych (Entity)

```
1 package edu.pw.pap.sample;
2
3 import lombok.Data;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.Id;
7
8 @Entity
9 @Data
10 public class Employee {
11     private @Id @GeneratedValue Long id;
12     private String name;
13     private String role;
14 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA to interfejsy z metodami obsługującymi tworzenie, odczytywanie, aktualizowanie i usuwanie rekordów

```
1 package edu.pw.pap.sample;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 public interface EmployeeRepository  
6     extends JpaRepository<Employee, Long> {  
7  
8 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
 - Niektóre repozytoria obsługują również stronicowanie danych i sortowanie.

```
1 package edu.pw.pap.sample;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 public interface EmployeeRepository  
6     extends JpaRepository<Employee, Long> {  
7  
8 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
 - Niektóre repozytoria obsługują również stronicowanie danych i sortowanie.
 - Spring Data realizuje implementacje w oparciu o konwencje występujące w nazewnictwie metod w interfejsie.

```
1 package edu.pw.pap.sample;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 public interface EmployeeRepository  
6     extends JpaRepository<Employee, Long> {  
7  
8 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
- LoadDatabase.java

```
1 package edu.pw.pap.sample;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13
14     @Bean
15     CommandLineRunner initDatabase(EmployeeRepository repository) {
16         return args -> {
17             log.info("Preloading " + repository.save(
18                 new Employee("Bilbo Baggins", "burglar")));
19             log.info("Preloading " + repository.save(
20                 new Employee("Frodo Baggins", "thief")));
21         };
22     }
23 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
- LoadDatabase.java
 - Spring Boot uruchomi WSZYSTKIE ziarna CommandLineRunner po załadowaniu kontekstu aplikacji.

```
1 package edu.pw.pap.sample;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13     @Bean
14     CommandLineRunner initDatabase(EmployeeRepository repository) {
15         return args -> {
16             log.info("Preloading " + repository.save(
17                 new Employee("Bilbo Baggins", "burglar")));
18             log.info("Preloading " + repository.save(
19                 new Employee("Frodo Baggins", "thief")));
20         };
21     }
22 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
- LoadDatabase.java
 - Spring Boot uruchomi WSZYSTKIE ziarna CommandLineRunner po załadowaniu kontekstu aplikacji.
 - CommandLineRunner dostanie referencję do utworzonego właśnie repozytorium

```
1 package edu.pw.pap.sample;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13     @Bean
14     CommandLineRunner initDatabase(EmployeeRepository repository) {
15         return args -> {
16             log.info("Preloading " + repository.save(
17                 new Employee("Bilbo Baggins", "burglar")));
18             log.info("Preloading " + repository.save(
19                 new Employee("Frodo Baggins", "thief")));
20         };
21     }
22 }
```

- tworzymy obiekt danych (Entity)
- Repozytoria Spring Data JPA
- LoadDatabase.java

```
1 2022-02-25 18:53:29.244 INFO 114052 --- [ main] edu....tion : Started SampleApplication in 1.964 seconds (JVM
   ↳ running for 2.332)
2 2022-02-25 18:53:29.273 INFO 114052 --- [ main] edu....base : Preloading Employee(id=1, name=Bilbo Baggins,
   ↳ role=burglar)
3 2022-02-25 18:53:29.275 INFO 114052 --- [ main] edu....base : Preloading Employee(id=2, name=Frodo Baggins,
   ↳ role=thief)
```

Spring MVC



EmployeeNotFoundException

```
1 package edu.pw.pap.sample;
2
3 class EmployeeNotFoundException extends RuntimeException {
4
5     EmployeeNotFoundException(Long id) {
6         super("Could not find employee " + id);
7     }
8 }
```

Spring MVC

- **EmployeeNotFoundException**
- **EmployeeNotFoundAdvice**

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.ControllerAdvice;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8
9 @ControllerAdvice
10 class EmployeeNotFoundAdvice {
11
12     @ResponseBody
13     @ExceptionHandler(EmployeeNotFoundException.class)
14     @ResponseStatus(HttpStatus.NOT_FOUND)
15     String employeeNotFoundHandler(EmployeeNotFoundException ex) {
16         return ex.getMessage();
17     }
18 }
19
```

Spring MVC

- EmployeeNotFoundException

- EmployeeNotFoundAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.ControllerAdvice;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8
9 @ControllerAdvice
10 class EmployeeNotFoundAdvice {
11
12     @ResponseBody
13     @ExceptionHandler(EmployeeNotFoundException.class)
14     @ResponseStatus(HttpStatus.NOT_FOUND)
15     String employeeNotFoundHandler(EmployeeNotFoundException ex) {
16         return ex.getMessage();
17     }
18 }
19
```

Spring MVC

- EmployeeNotFoundException

- EmployeeNotFoundExceptionAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

@ExceptionHandler : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.ControllerAdvice;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8
9 @ControllerAdvice
10 class EmployeeNotFoundExceptionAdvice {
11
12     @ResponseBody
13     @ExceptionHandler(EmployeeNotFoundException.class)
14     @ResponseStatus(HttpStatus.NOT_FOUND)
15     String employeeNotFoundHandler(EmployeeNotFoundException ex) {
16         return ex.getMessage();
17     }
18 }
19
```


- EmployeeNotFoundException

- EmployeeNotFoundAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

@ExceptionHandler : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

@ResponseStatus

: mówi, aby ustawić HttpStatus.NOT_FOUND.

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.ControllerAdvice;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8
9 @ControllerAdvice
10 class EmployeeNotFoundAdvice {
11
12     @ResponseBody
13     @ExceptionHandler(EmployeeNotFoundException.class)
14     @ResponseStatus(HttpStatus.NOT_FOUND)
15     String employeeNotFoundHandler(EmployeeNotFoundException ex) {
16         return ex.getMessage();
17     }
18 }
19
```


- EmployeeNotFoundException

- EmployeeNotFoundAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

@ExceptionHandler : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

@ResponseStatus : mówi, aby ustawić HttpStatus.NOT_FOUND.

```
1 package edu.pw.pap.sample;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.ControllerAdvice;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.ResponseStatus;
8
9 @ControllerAdvice
10 class EmployeeNotFoundAdvice {
11
12     @ResponseBody
13     @ExceptionHandler(EmployeeNotFoundException.class)
14     @ResponseStatus(HttpStatus.NOT_FOUND)
15     String employeeNotFoundHandler(EmployeeNotFoundException ex) {
16         return ex.getMessage();
17     }
18 }
19
```


- EmployeeNotFoundException

- EmployeeNotFoundExceptionAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

@ExceptionHandler : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

@ResponseStatus

: mówi, aby ustawić HttpStatus.NOT_FOUND.

```
1 package edu.pw.pap.sample;
2 import java.util.List;
3 import org.springframework.web.bind.annotation.DeleteMapping;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PathVariable;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.PutMapping;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RestController;
10 @RestController
11 class EmployeeController {
12
13     private final EmployeeRepository repository;
14     EmployeeController(EmployeeRepository repository) {
15         this.repository = repository;
16     }
17
18     // Aggregate root
19     // tag::get-aggregate-root[]
20     @GetMapping("/employees")
21     List<Employee> all() {
22         return repository.findAll();
23     }
24     // end::get-aggregate-root[]
25     ....
26 }
```


Spring MVC

- EmployeeNotFoundException

- EmployeeNotFoundExceptionAdvice

@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

@ExceptionHandler : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

@ResponseStatus

: mówi, aby ustawić HttpStatus.NOT_FOUND.

```
1 package edu.pw.pap.sample;
2 ....
3 @RestController
4 class EmployeeController {
5
6     private final EmployeeRepository repository;
7     ....
8     @PostMapping("/employees")
9     Employee newEmployee(@RequestBody Employee newEmployee) {
10         return repository.save(newEmployee);
11     }
12
13     // Single item
14
15     @GetMapping("/employees/{id}")
16     Employee one(@PathVariable Long id) {
17
18         return repository.findById(id)
19             .orElseThrow(() -> new EmployeeNotFoundException(id));
20     }
21
22 }
```

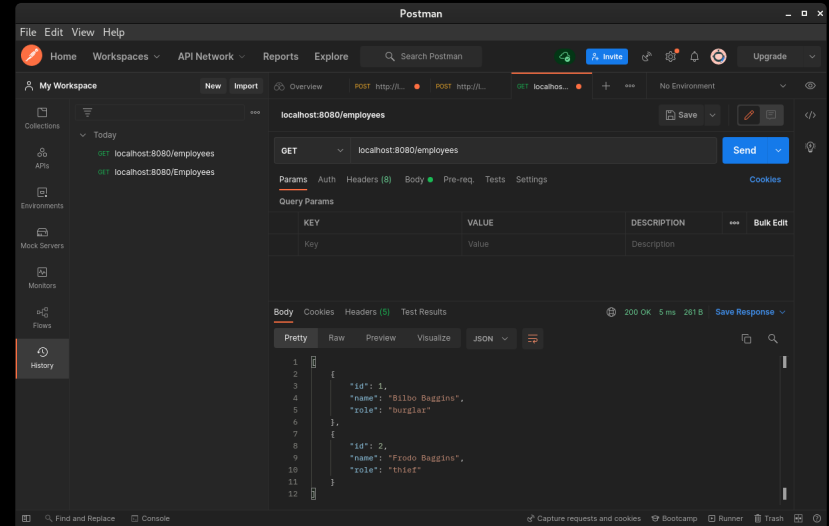

- EmployeeNotFoundException
 - EmployeeNotFoundExceptionAdvice
- @ResponseBody** : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi
- @ExceptionHandler** : konfiguruje poradę tak, aby odpowiadała tylko wtedy, gdy zostanie zgłoszony wyjątek EmployeeNotFoundException

@ResponseStatus

: mówi, aby ustawić HttpStatus.NOT_FOUND.

```
1 package edu.pw.pap.sample;
2 ....
3 @RestController
4 class EmployeeController {
5
6     private final EmployeeRepository repository;
7     ....
8     @PutMapping("/employees/{id}")
9     Employee replaceEmployee(@RequestBody Employee newEmployee,
10                             @PathVariable Long id) {
11         return repository.findById(id)
12             .map(employee -> {
13                 employee.setName(newEmployee.getName());
14                 employee.setRole(newEmployee.getRole());
15                 return repository.save(employee);
16             })
17             .orElseGet(() -> {
18                 newEmployee.setId(id);
19                 return repository.save(newEmployee);
20             });
21     }
22     @DeleteMapping("/employees/{id}")
23     void deleteEmployee(@PathVariable Long id) {
24         repository.deleteById(id);
25     }
26 }
```

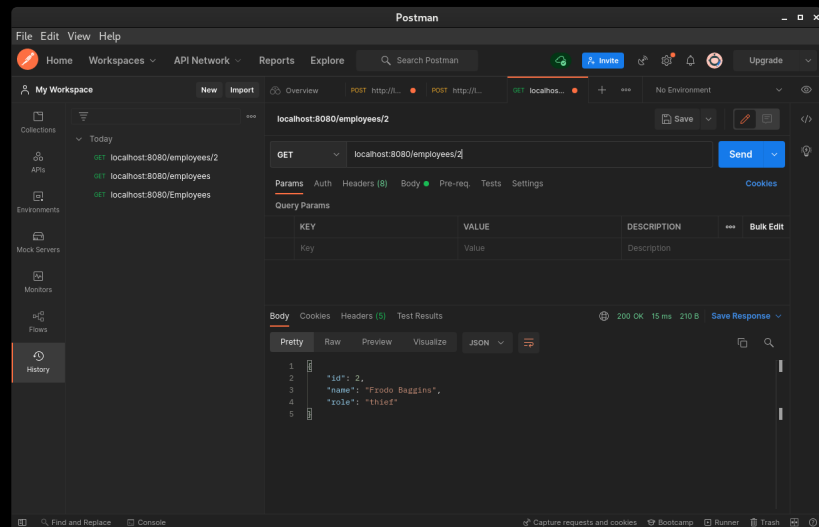
- odczyt wszystkich rekordów



- odczyt wszystkich rekordów

```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -v localhost:8080/employees
* Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /employees HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Fri, 25 Feb 2022 20:01:51 GMT
<
* Connection #0 to host localhost left intact
[{"id":1,"name":"Bilbo Baggins","role":"burglar"}, {"id":2,"name":"Frodo Baggins","role":"thief"}]jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

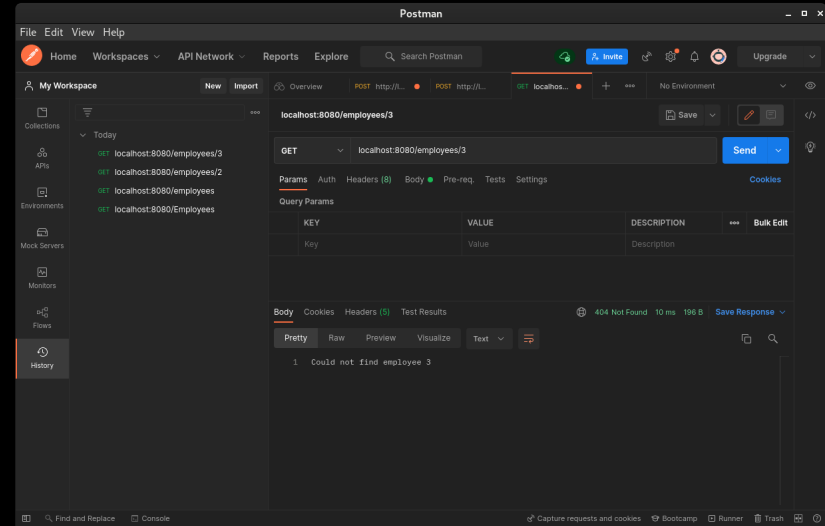
- odczyt wszystkich rekordów
- odczyt jednego rekordu



- odczyt wszystkich rekordów
- odczyt jednego rekordu

```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -v localhost:8080/employees/2
* Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /employees/2 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Fri, 25 Feb 2022 20:06:32 GMT
<
* Connection #0 to host localhost left intact
{"id":2,"name":"Frodo Baggins","role":"thief"}jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

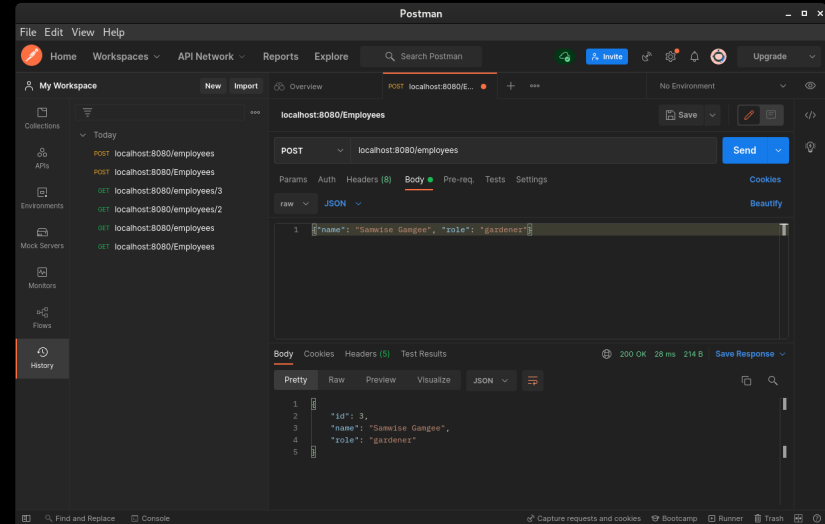
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu



- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu

```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -v localhost:8080/employees/4
* Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /employees/4 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404
< Content-Type: text/plain; charset=UTF-8
< Content-Length: 25
< Date: Fri, 25 Feb 2022 20:08:13 GMT
<
* Connection #0 to host localhost left intact
Could not find employee 4jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

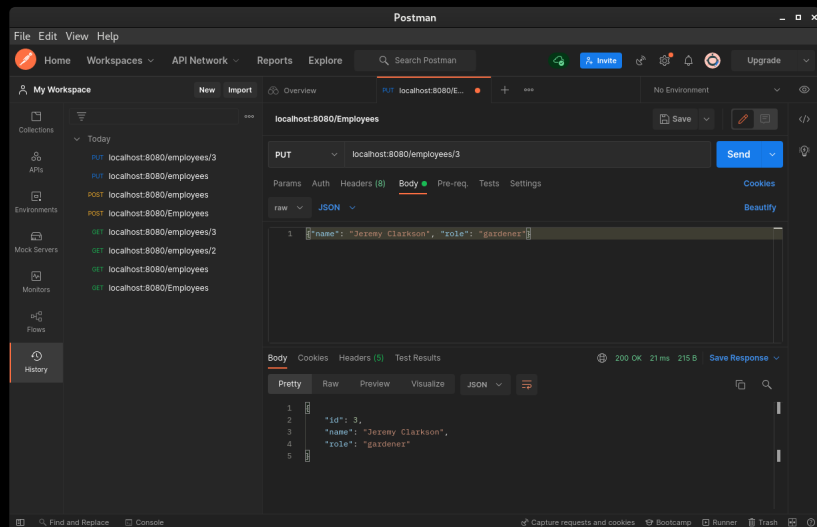
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- **dodanie rekordu**



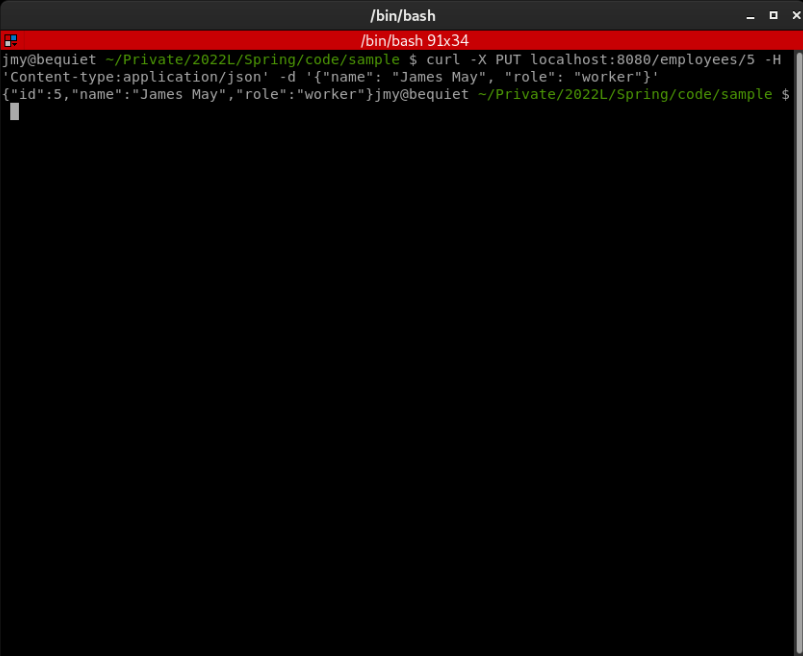
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu

```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -X POST localhost:8080/employees -H '
Content-type:application/json' -d '{"name": "Samwise Gamgee", "role": "gardener"}'
{"id":4,"name":"Samwise Gamgee","role":"gardener"}jmy@bequiet ~/Private/2022L/Spring/code/s
ample $
```

- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- **modyfikacja rekordu**

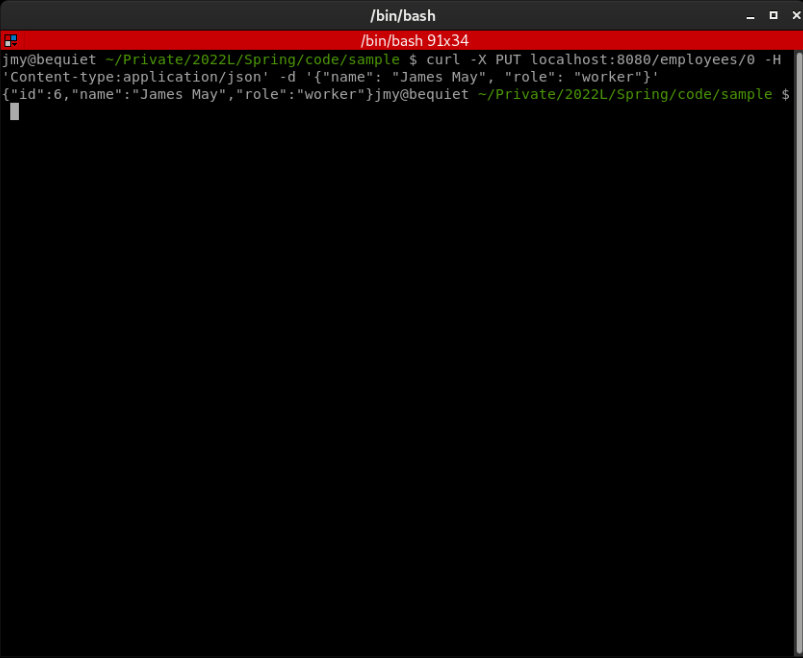


- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu

A terminal window with a dark background and light text. The title bar shows "/bin/bash" and window control buttons. The terminal content shows a user "jmy@bequiet" in a directory "~/Private/2022L/Spring/code/sample" running a curl command to perform a PUT request to "localhost:8080/employees/5". The command includes headers for "Content-type: application/json" and a JSON body with "name": "James May" and "role": "worker". The output shows the JSON body being sent. The prompt is "\$".

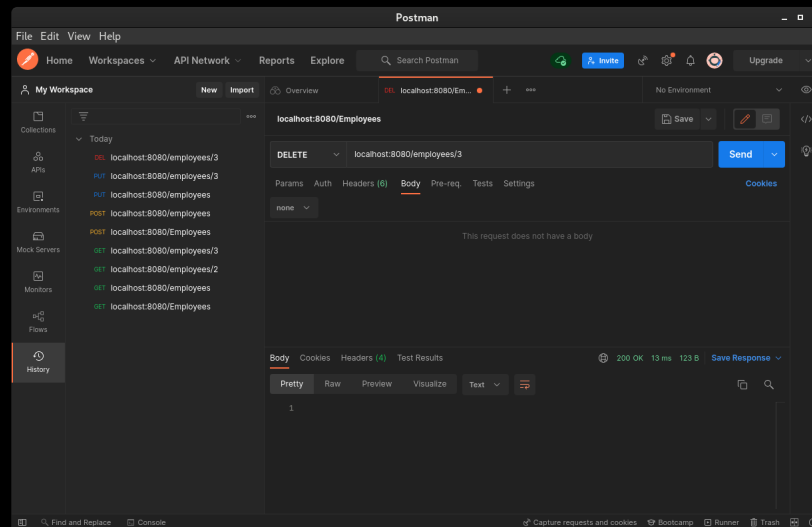
```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -X PUT localhost:8080/employees/5 -H
'Content-type:application/json' -d '{"name": "James May", "role": "worker"}'
{"id":5,"name":"James May","role":"worker"}jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu

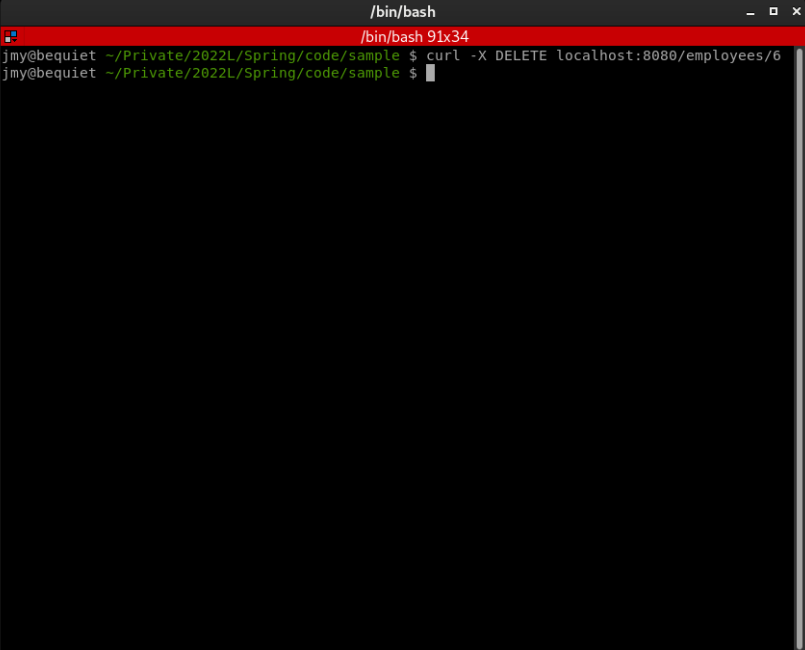
A terminal window titled "/bin/bash" with a red title bar. The prompt is "jmy@bequiet ~/Private/2022L/Spring/code/sample". The user enters a curl command: "curl -X PUT localhost:8080/employees/0 -H 'Content-type:application/json' -d '{\"name\": \"James May\", \"role\": \"worker\"}'". The command is executed, and the output is displayed: "{ \"id\": 6, \"name\": \"James May\", \"role\": \"worker\" }". The prompt returns to "jmy@bequiet ~/Private/2022L/Spring/code/sample \$".

```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -X PUT localhost:8080/employees/0 -H
'Content-type:application/json' -d '{"name": "James May", "role": "worker"}'
{"id":6,"name":"James May","role":"worker"}jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu
- **usunięcie rekordu**



- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu
- usunięcie rekordu



```
/bin/bash
/bin/bash 91x34
jmy@bequiet ~/Private/2022L/Spring/code/sample $ curl -X DELETE localhost:8080/employees/6
jmy@bequiet ~/Private/2022L/Spring/code/sample $
```

Moduły:

Core : Zapewnia podstawowe funkcje, takie jak DI (wstrzykiwanie zależności), internacjonalizacja, walidacja i AOP (programowanie zorientowane na aspekty)

Moduły:

Core : Zapewnia podstawowe funkcje, takie jak DI (wstrzykiwanie zależności), internacjonalizacja, walidacja i AOP (programowanie zorientowane na aspekty)

Data Access : Obsługuje dostęp do danych przez JTA (Java Transaction API), JPA (Java Persistence API) i JDBC (Java Database Connectivity)

Moduły:

Core : Zapewnia podstawowe funkcje, takie jak DI (wstrzykiwanie zależności), internacjonalizacja, walidacja i AOP (programowanie zorientowane na aspekty)

Data Access : Obsługuje dostęp do danych przez JTA (Java Transaction API), JPA (Java Persistence API) i JDBC (Java Database Connectivity)

Web : Obsługuje zarówno Servlet API (Spring MVC), jak i ostatnio Reactive API (Spring WebFlux), a dodatkowo obsługuje WebSockets, STOMP i WebClient

Moduły:

Core : Zapewnia podstawowe funkcje, takie jak DI (wstrzykiwanie zależności), internacjonalizacja, walidacja i AOP (programowanie zorientowane na aspekty)

Data Access : Obsługuje dostęp do danych przez JTA (Java Transaction API), JPA (Java Persistence API) i JDBC (Java Database Connectivity)

Web : Obsługuje zarówno Servlet API (Spring MVC), jak i ostatnio Reactive API (Spring WebFlux), a dodatkowo obsługuje WebSockets, STOMP i WebClient

Integration : Obsługuje integrację z Enterprise Java za pośrednictwem JMS (Java Message Service), JMX (Java Management Extension) i RMI (Remote Method Invocation)

Moduły:

Core : Zapewnia podstawowe funkcje, takie jak DI (wstrzykiwanie zależności), internacjonalizacja, walidacja i AOP (programowanie zorientowane na aspekty)

Data Access : Obsługuje dostęp do danych przez JTA (Java Transaction API), JPA (Java Persistence API) i JDBC (Java Database Connectivity)

Web : Obsługuje zarówno Servlet API (Spring MVC), jak i ostatnio Reactive API (Spring WebFlux), a dodatkowo obsługuje WebSockets, STOMP i WebClient

Integration : Obsługuje integrację z Enterprise Java za pośrednictwem JMS (Java Message Service), JMX (Java Management Extension) i RMI (Remote Method Invocation)

Testing : Szerokie wsparcie dla testów jednostkowych i integracyjnych poprzez Mock Objects, Test Fixtures, Context Management i Caching

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

- Ułatwia tworzenie samodzielnych aplikacji Spring z osadzonym Tomcatem lub podobnym kontenerem.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

Security : Zapewnia solidny mechanizm uwierzytelniania i autoryzacji dla projektów opartych na Spring.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- Przy minimalnym wsparciu deklaratywnym otrzymujemy ochronę przed typowymi atakami, takimi jak utrwalanie sesji, przechwytywanie kliknięć i fałszowanie żądań między witrynami.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- Przy minimalnym wsparciu deklaratywnym otrzymujemy ochronę przed typowymi atakami, takimi jak utrwalanie sesji, przechwytywanie kliknięć i fałszowanie żądań między witrynami.

Mobile : Zapewnia możliwości wykrywania urządzenia i odpowiedniego dostosowania aplikacji.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- Przy minimalnym wsparciu deklaratywnym otrzymujemy ochronę przed typowymi atakami, takimi jak utrwalanie sesji, przechwytywanie kliknięć i fałszowanie żądań między witrynami.

Mobile : Zapewnia możliwości wykrywania urządzenia i odpowiedniego dostosowania aplikacji.

- Ponadto obsługuje zarządzanie widokami zależne od urządzenia w celu zapewnienia optymalnego doświadczenia użytkownika, zarządzania preferencjami witryn i przełączania witryn.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- Przy minimalnym wsparciu deklaratywnym otrzymujemy ochronę przed typowymi atakami, takimi jak utrwalanie sesji, przechwytywanie kliknięć i fałszowanie żądań między witrynami.

Mobile : Zapewnia możliwości wykrywania urządzenia i odpowiedniego dostosowania aplikacji.

Batch : Zapewnia uproszczoną strukturę do tworzenia aplikacji wsadowych. Posiada intuicyjną obsługę planowania, restartowania, pomijania, zbierania metryk i rejestrowania.

Projekty:

Boot : Dostarcza nam zestaw rozszerzalnych szablonów do tworzenia projektów opartych na Spring.

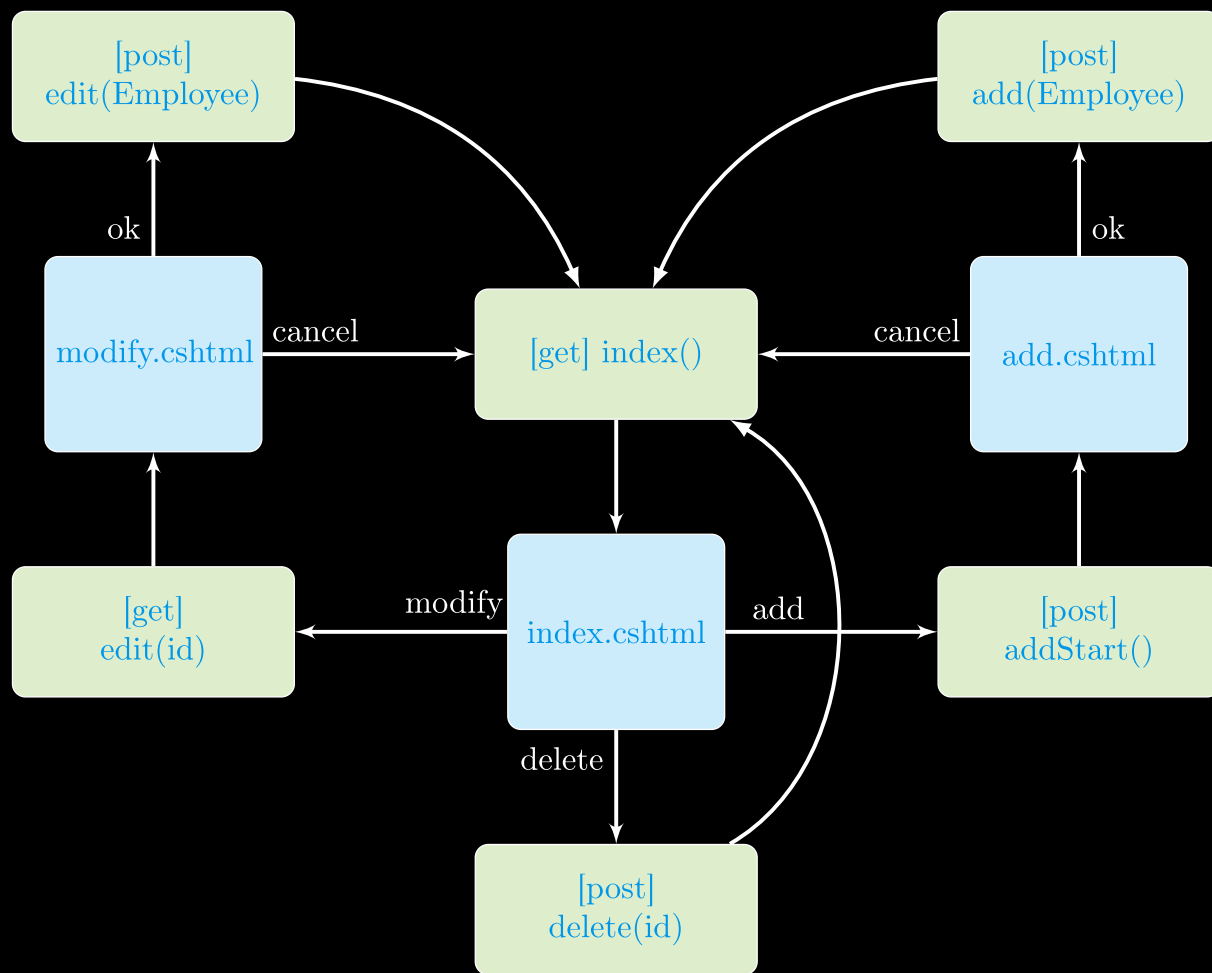
Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- Przy minimalnym wsparciu deklaratywnym otrzymujemy ochronę przed typowymi atakami, takimi jak utrwalanie sesji, przechwytywanie kliknięć i fałszowanie żądań między witrynami.

Mobile : Zapewnia możliwości wykrywania urządzenia i odpowiedniego dostosowania aplikacji.

Batch : Zapewnia uproszczoną strukturę do tworzenia aplikacji wsadowych. Posiada intuicyjną obsługę planowania, restartowania, pomijania, zbierania metryk i rejestrowania.

- **Dodatkowo obsługuje skalowanie w górę dla zadań o dużej liczbie zadań poprzez optymalizację i partycjonowanie.**



- budujemy projekt w oparciu o `spring-boot-starter-parent`
 - spring initializer
 - dodanie zależności

- budujemy projekt w oparciu o `spring-boot-starter-parent`
 - `spring initializer`
 - dodanie zależności

New Project



Empty Project

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Kotlin Multiplatform

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE Plugin

Android

Vue.js

Vite

Server URL: start.spring.io

Name: jsf-crud

Location: ~/Private/2023L/Spring/code

Project will be created in: ~/Private/2023L/Spring/code/jsf-crud

☐ Create Git repository

Language:

Java

Kotlin

Groovy

Type:

Gradle - Groovy

Gradle - Kotlin

Maven

Group:

com.example

Artifact:

jsf-crud

Package name:

com.example.jsfcrud

JDK:

17 version 17.0.4

Java:

17

Packaging:

Jar

War

Next

Cancel

- budujemy projekt w oparciu o `spring-boot-starter-parent`
 - spring initializer
 - dodanie zależności

New Project



Spring Boot: 3.0.6

☒ Download pre-built shared indexes for JDK and Maven libraries

Dependencies:

Q Search

- ☐ Spring HATEOAS
- ☐ Spring Web Services
- ☐ Jersey
- ☐ Vaadin

Template Engines

- ☒ Thymeleaf
- ☐ Apache Freemarker
- ☐ Mustache
- ☐ Groovy Templates

Security

- ☐ Spring Security
- ☐ OAuth2 Client
- ☐ OAuth2 Resource Server
- ☐ Spring LDAP
- ☐ Okta

SQL

Thymeleaf

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

[Guide](#)

Added dependencies:

- × Lombok
- × Spring Boot DevTools
- × Spring Web
- × Spring Session
- × Apache Derby Database
- × Thymeleaf



Previous

Create

Cancel

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>3.0.6</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>jsf-crud</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>jsf-crud</name>
15    <description>jsf-crud</description>
16    <properties>
17        <java.version>17</java.version>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-thymeleaf</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>org.springframework.boot</groupId>
26            <artifactId>spring-boot-starter-web</artifactId>
27        </dependency>
28        <dependency>
29            <groupId>org.springframework.session</groupId>
30            <artifactId>spring-session-core</artifactId>
31        </dependency>
32
33        <dependency>
34            <groupId>org.springframework.boot</groupId>
35            <artifactId>spring-boot-devtools</artifactId>
36            <scope>runtime</scope>
37            <optional>true</optional>
38        </dependency>
39        <dependency>
40            <groupId>org.apache.derby</groupId>
```

```
41         <artifactId>derby</artifactId>
42         <scope>runtime</scope>
43     </dependency>
44     <dependency>
45         <groupId>org.projectlombok</groupId>
46         <artifactId>lombok</artifactId>
47         <optional>true</optional>
48     </dependency>
49     <dependency>
50         <groupId>org.springframework.boot</groupId>
51         <artifactId>spring-boot-starter-test</artifactId>
52         <scope>test</scope>
53     </dependency>
54 </dependencies>
55
56 <build>
57     <plugins>
58         <plugin>
59             <groupId>org.springframework.boot</groupId>
60             <artifactId>spring-boot-maven-plugin</artifactId>
61             <configuration>
62                 <excludes>
63                     <exclude>
64                         <groupId>org.projectlombok</groupId>
65                         <artifactId>lombok</artifactId>
66                     </exclude>
67                 </excludes>
68             </configuration>
69         </plugin>
70     </plugins>
71 </build>
72
73 </project>
```

- Tworzymy katalogi controllers entities i repositories
- Tworzymy klasę Employee
- Dodajemy interfejs EmployeeRepository
 - interfejs CrudRepository z biblioteki Spring Data JPA ułatwia operacje CRUD na danych
- EmployeeController

- Tworzymy katalogi controllers entities repositories
- Tworzymy klasę Employee
- Dodajemy interfejs EmployeeRepository
 - interfejs CrudRepository z biblioteki Spring Data JPA ułatwia operacje CRUD na danych
- EmployeeController

```
1 package com.example.jsfcrud.entities;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 import jakarta.validation.constraints.NotBlank;
9 import lombok.Data;
10
11 @Entity
12 @Data
13 public class Employee {
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     private long EmployeeId;
17
18     @NotBlank(message = "First Name should not be blank")
19     private String firstName;
20
21     @NotBlank(message = "Last Name should not be blank")
22     private String lastName;
23
24     private int salary;
25 }
```

- Tworzymy katalogi `controllers`, `entities`, `repositories`
- Tworzymy klasę `Employee`
- Dodajemy interfejs `EmployeeRepository`
 - interfejs `CrudRepository` z biblioteki Spring Data JPA ułatwia operacje CRUD na danych
- `EmployeeController`

```
1 package com.example.jsfcrud.repositories;
2 import com.example.jsfcrud.entities.Employee;
3 import org.springframework.data.repository.CrudRepository;
4
5 import java.util.List;
6
7 public interface EmployeeRepository extends CrudRepository<Employee, Long> {
8     List<Employee> findById(long id);
9 }
10
```

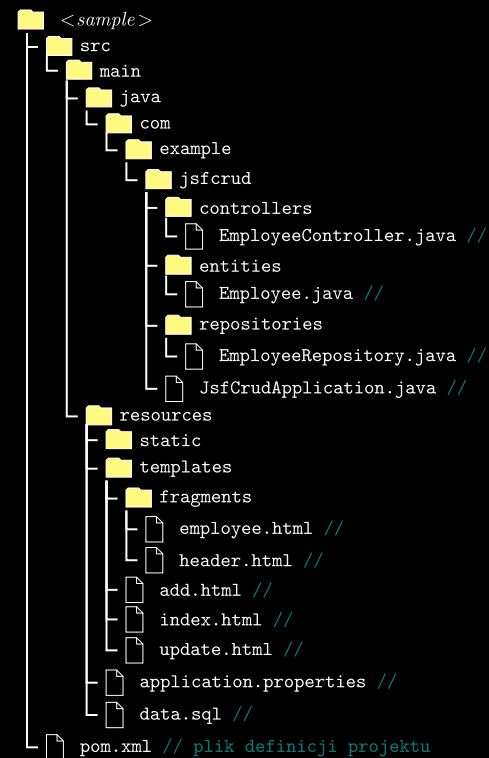
- Tworzymy katalogi `controllers`, `entities` i `repositories`
- Tworzymy klasę `Employee`
- Dodajemy interfejs `EmployeeRepository`
 - interfejs `CrudRepository` z biblioteki `Spring Data JPA` ułatwia operacje `CRUD` na danych
- `EmployeeController`

```
1 package com.example.jsfcrud.repositories;
2 import com.example.jsfcrud.entities.Employee;
3 import org.springframework.data.repository.CrudRepository;
4
5 import java.util.List;
6
7 public interface EmployeeRepository extends CrudRepository<Employee, Long> {
8     List<Employee> findById(long id);
9 }
10
```

- Tworzymy katalogi controllers entities repositories
- Tworzymy klasę Employee
- Dodajemy interfejs EmployeeRepository
 - interfejs CrudRepository z biblioteki Spring Data JPA ułatwia operacje CRUD na danych
- EmployeeController

```
1 package com.example.jsfcrud.controllers;
2 import com.example.jsfcrud.repositories.EmployeeRepository;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7
8 @Controller
9 public class EmployeeController {
10     private final EmployeeRepository repository;
11
12     @Autowired
13     public EmployeeController(EmployeeRepository repository) {
14         this.repository = repository;
15     }
16
17     @GetMapping("/index")
18     public String showEmployeeList(Model model) {
19
20         model.addAttribute("employees", repository.findAll());
21         return "index";
22     }
23 }
```

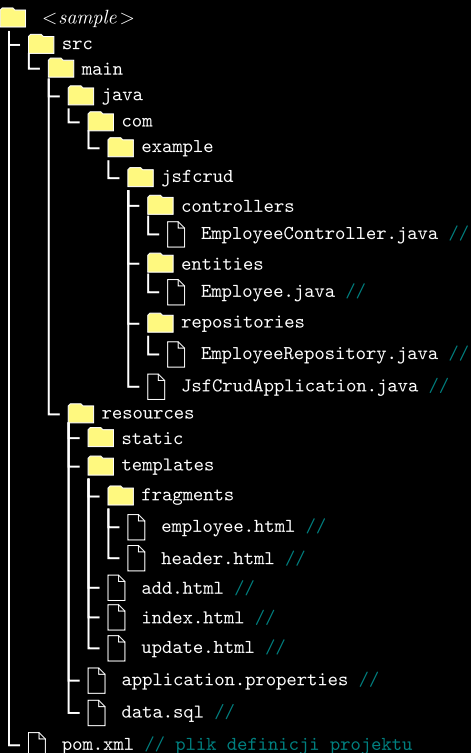
```
1 package com.example.jsfcrud;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class JsfcCrudApplication {  
8     public static void main(String[] args) {  
9  
10         SpringApplication.run(JsfcCrudApplication.class, args);  
11     }  
12 }
```




```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
   ↪ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   ↪ https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>3.0.6</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>jsf-crud</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>jsf-crud</name>
15    <description>jsf-crud</description>
16    <properties>
17        <java.version>17</java.version>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>jakarta.validation</groupId>
22            <artifactId>jakarta.validation-api</artifactId>
23            <version>3.0.2</version>
24        </dependency>
25
26        <dependency>
27            <groupId>org.springframework.boot</groupId>
28            <artifactId>spring-boot-starter-thymeleaf</artifactId>
29        </dependency>
30        <dependency>
31            <groupId>org.springframework.boot</groupId>
32            <artifactId>spring-boot-starter-web</artifactId>
33        </dependency>
34
35        <dependency>
36            <groupId>org.springframework.boot</groupId>
37            <artifactId>spring-boot-starter-validation</artifactId>
38        </dependency>
39
40        <dependency>
41            <groupId>org.springframework.session</groupId>
42            <artifactId>spring-session-core</artifactId>
43        </dependency>

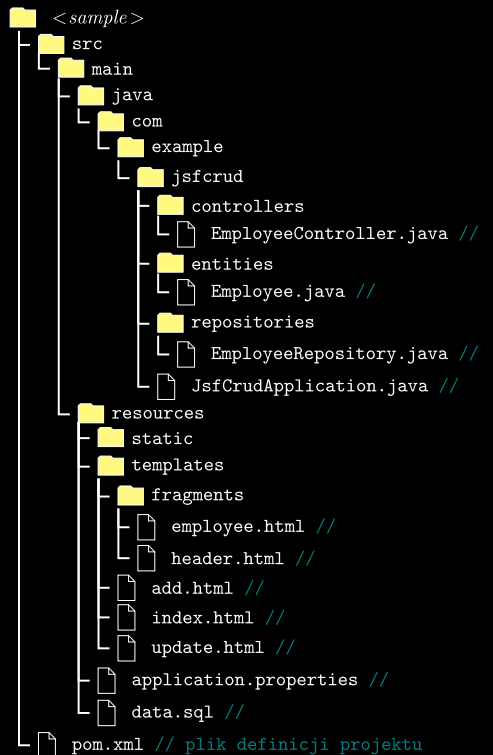
```




```

1 package com.example.jsfcrud.controllers;
2 import com.example.jsfcrud.repositories.EmployeeRepository;
3 import com.example.jsfcrud.entities.Employee;
4 import jakarta.validation.Valid;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12
13 @Controller
14 public class EmployeeController {
15     private final EmployeeRepository repository;
16
17     @Autowired
18     public EmployeeController(EmployeeRepository repository) {
19         this.repository = repository;
20     }
21
22     @GetMapping("/index")
23     public String showEmployeeList(Model model) {
24
25         model.addAttribute("employees", repository.findAll());
26         return "index";
27     }
28
29     @GetMapping("/add")
30     public String showNewEmployee(Employee employee) {
31         return "add";
32     }
33
34     @PostMapping("/add")
35     public String saveNewEmployee(@Valid Employee e, BindingResult result, Model
        ↪ model) {
36         if (result.hasErrors()) {
37             return "add";
38         }
39         repository.save(e);
40         return "redirect:/index";
41     }
42
43     @GetMapping("/edit/{id}")
44     public String showExistingEmployee(@PathVariable("id") long id, Model model)

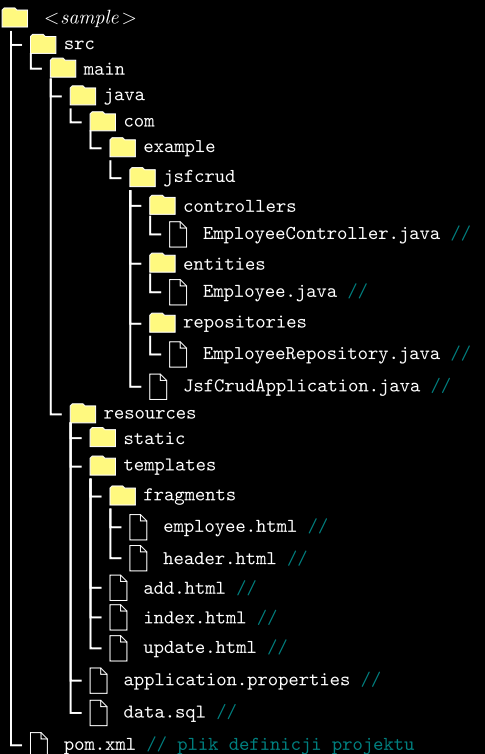
```




```

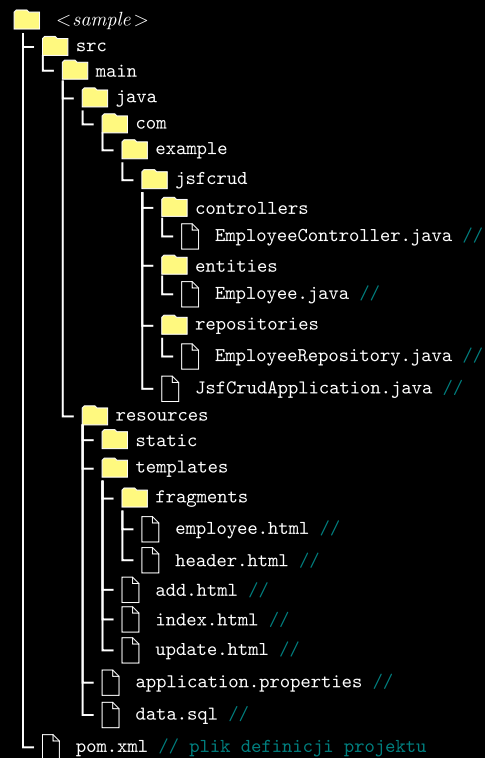
1 package com.example.jsfcrud.entities;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 import jakarta.validation.constraints.NotBlank;
9 import lombok.Data;
10
11 @Entity
12 //@Data
13 public class Employee {
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     private long EmployeeId;
17
18     @NotBlank(message = "First Name should not be blank")
19     private String firstName;
20
21     @NotBlank(message = "Last Name should not be blank")
22     private String lastName;
23
24     private int salary;
25
26
27     public long getEmployeeId() {
28         return EmployeeId;
29     }
30
31     public void setEmployeeId(long employeeId) {
32         EmployeeId = employeeId;
33     }
34
35     public String getFirstName() {
36         return firstName;
37     }
38
39     public void setFirstName(String firstName) {
40         this.firstName = firstName;
41     }
42
43     public String getLastName() {
44         return lastName;
45     }
46
47 }

```



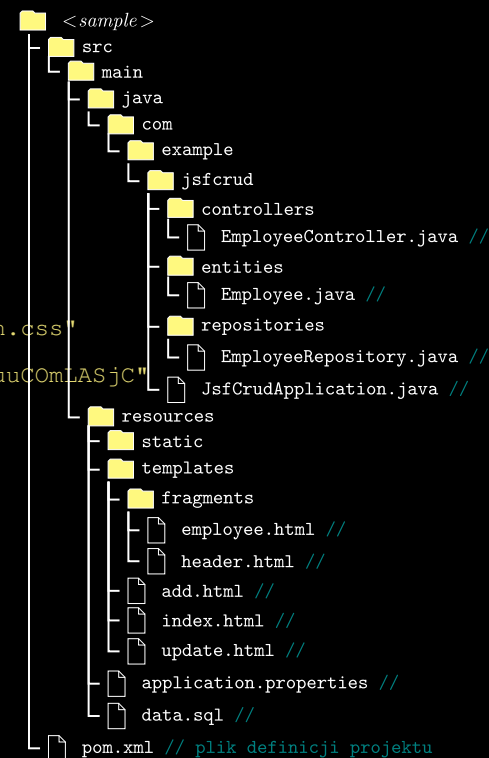
crud/src/main/resources/templates/fragments/employee.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <span th:fragment="employee">
9     <div class="row mb-3">
10       <label for="firstName" class="col-form-label">Name</label>
11       <input type="text" th:field="*{firstName}" class="form-control"
12         ↪ id="firstName" placeholder="Name">
13       <span th:if="${#fields.hasErrors('firstName')}}"
14         ↪ th:errors="*{firstName}" class="text-danger"></span>
15     </div>
16     <div class="row mb-3">
17       <label for="lastName" class="col-form-label">lastName</label>
18       <input type="text" th:field="*{lastName}" class="form-control"
19         ↪ id="lastName" placeholder="Name">
20       <span th:if="${#fields.hasErrors('lastName')}}" th:errors="*{lastName}"
21         ↪ class="text-danger"></span>
22     </div>
23     <div class="row mb-3">
24       <label for="salary" class="col-form-label">Salary</label>
25       <input type="text" th:field="*{salary}" class="form-control"
26         ↪ id="salary" placeholder="Email">
27       <span th:if="${#fields.hasErrors('salary')}}" th:errors="*{salary}"
28         ↪ class="text-danger"></span>
29     </div>
30   </span>
31 </body>
32 </html>
```



crud/src/main/resources/templates/fragments/header.html

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
3 <head th:fragment="header">
4     <!-- Required meta tags -->
5     <meta charset="utf-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <!-- Bootstrap CSS -->
8     <link
9         ↪ href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
10         ↪ rel="stylesheet"
11         ↪ integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWFSpd3yD65VohhpuuCOMLASjC"
12         ↪ crossorigin="anonymous">
13     <title th:text="${title}"></title>
14 </head>
15 <body>
16 </body>
17 </html>
```



crud/src/main/resources/templates/add.html

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head th:insert="~{fragments/header :: header}" th:remove="tag"></head>
4 <body>
5 <div class="container my-5">
6   <h2 class="mb-5">New Employee</h2>
7   <form action="#" th:action="@{/add}" th:object="${employee}"
8     ↪ method="post">
9     <span th:insert="~{fragments/employee :: employee}"
10       ↪ th:remove="tag"></span>
11     <button type="submit" class="btn btn-primary">Add</button>
12     <a href="/index" class="btn btn-danger">Cancel</a>
13   </form>
14 </div>
15 </body>
16 </html>
```



crud/src/main/resources/templates/index.html

```

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
3 <head th:insert="~{fragments/header :: header}" th:remove="tag"></head>
4 <body>
5 <div th:switch="${employees}" class="container my-5">
6   <h2 th:case="null">No employees yet!</h2>
7   <div th:case="*">
8     <h2 class="my-5">Employees</h2>
9     <table class="table table-striped table-responsive-md">
10       <thead>
11         <tr>
12           <th>First Name</th>
13           <th>Last Name</th>
14           <th>Edit</th>
15           <th>Delete</th>
16         </tr>
17       </thead>
18       <tbody>
19         <tr th:each="e : ${employees}">
20           <td th:text="${e.firstName}"></td>
21           <td th:text="${e.lastName}"></td>
22           <td><a th:href="@{/edit/{id}}(id=${e.employeeId})" class="btn
23             ↳ btn-primary">Edit</a></td>
24           <td><a th:href="@{/delete/{id}}(id=${e.employeeId})"
25             ↳ class="btn btn-primary">Delete</a></td>
26         </tr>
27       </tbody>
28     </table>
29   </div>
30   <p class="my-5"><a href="/add" class="btn btn-primary">Add
31     ↳ Employee</a></p>
32 </div>
33 </body>
34 </html>

```



```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
3 <head th:insert=~{fragments/header :: header} th:remove="tag"></head>
4 <body>
5 <div class="container">
6   <h2 class="mb-5"><span th:text="${title}" th:remove="tag">Page
7     ↳ Title</span></h2>
8   <form action="#" th:action="@{/update/{id}(id=${employee.employeeId})}"
9     ↳ th:object="${employee}" method="post">
10     <span th:insert=~{fragments/employee :: employee}"
11       ↳ th:remove="tag"></span>
12     <button type="submit" class="btn btn-primary">Update</button>
13     <a href="/index" class="btn btn-danger">Cancel</a>
14   </form>
15 </div>
16 <!-- Option 1: Bootstrap Bundle with Popper -->
17 <script
18   ↳ src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
19   integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
20   crossorigin="anonymous"></script>
21 </body>
22 </html>
```



- Uruchomione jako samodzielna aplikacja (z domyślnym osadzonym kontenerem serwletów Tomcat)

```
1 <dependency>
2   <groupId>org.apache.tomcat.embed</groupId>
3   <artifactId>tomcat-embed-jasper</artifactId>
4   <version>9.0.44</version>
5   <scope>provided</scope>
6 </dependency>
```

- Uruchomione jako samodzielna aplikacja (z domyślnym osadzonym kontenerem serwetów Tomcat)
- aby umożliwić naszej aplikacji kompilację i renderowanie stron JSP

```
1 <dependency>
2   <groupId>org.apache.tomcat.embed</groupId>
3   <artifactId>tomcat-embed-jasper</artifactId>
4   <version>9.0.44</version>
5 </dependency>
```

- Uruchomione jako samodzielna aplikacja (z domyślnym osadzonym kontenerem serwletów Tomcat)
- aby umożliwić naszej aplikacji kompilację i renderowanie stron JSP
- **potrzebujemy biblioteki jstl**

```
1 <dependency>
2   <groupId>javax.servlet</groupId>
3   <artifactId>jstl</artifactId>
4   <version>1.2</version>
5 </dependency>
```

- Uruchomione jako samodzielna aplikacja (z domyślnym osadzonym kontenerem serwletów Tomcat)
- aby umożliwić naszej aplikacji kompilację i renderowanie stron JSP
- potrzebujemy biblioteki jstl
- aby zależności dostarczane przez naszą aplikację nie kolidowały z tymi dostarczonymi przez środowisko uruchomieniowe Tomcat, musimy ustawić dwie zależności o podanym zakresie:

```
1 <dependency>
2   <groupId>org.apache.tomcat.embed</groupId>
3   <artifactId>tomcat-embed-jasper</artifactId>
4   <version>9.0.44</version>
5   <scope>provided</scope>
6 </dependency>
7
8 <dependency>
9   <groupId>org.springframework.boot</groupId>
10  <artifactId>spring-boot-starter-tomcat</artifactId>
11  <version>2.4.4</version>
12  <scope>provided</scope>
13 </dependency>
```

- Uruchomione jako samodzielna aplikacja w pliku `application.properties` (z domyślnym osadzonym kontenerem serwetów Tomcat)
- aby umożliwić naszej aplikacji kompilację i renderowanie stron JSP
- potrzebujemy biblioteki `jstl`
- aby zależności dostarczane przez naszą aplikację nie kolidowały z tymi dostarczonymi przez środowisko uruchomieniowe Tomcat, musimy ustawić dwie zależności o podanym zakresie:
- **Musimy poinformować Spring, gdzie znajduje się pliki JSP**

```
1 spring.mvc.view.prefix: /WEB-INF/jsp/  
2 spring.mvc.view.suffix: .jsp
```

- Uruchomione jako samodzielna aplikacja (z domyślnym osadzonym kontenerem serwletów Tomcat)
- aby umożliwić naszej aplikacji kompilację i renderowanie stron JSP
- potrzebujemy biblioteki jstl
- aby zależności dostarczane przez naszą aplikację nie kolidowały z tymi dostarczonymi przez środowisko uruchomieniowe Tomcat, musimy ustawić dwie zależności o podanym zakresie:
- Musimy poinformować Spring, gdzie znajduje się pliki JSP
- jeśli uruchamiamy w kontenerze internetowym, musimy rozszerzyć `SpringBootServletInitializer` (czyli teraz nie jest potrzeby)

```
1 @SpringBootApplication(scanBasePackages = "atj")
2 public class SpringBootJspApplication extends
   ↳ SpringBootServletInitializer {
3
4     @Override
5     protected SpringApplicationBuilder
6       ↳ configure(SpringApplicationBuilder builder) {
7         return
8           ↳ builder.sources(SpringBootJspApplication.class);
9     }
10
11     public static void main(String[] args) {
12         ↳ SpringApplication.run(SpringBootJspApplication.class);
13     }
14 }
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:

```
1 public class Person {
2     private Integer id;
3     private String firstName;
4     private String lastName;
5     // getter, setter, konstruktor i toString
6 }
7
8 public interface PersonService {
9     Collection<Person> getPersons();
10    Person addPerson(Person person);
11 }
```

- Założmy, że mamy usługę `PersonService`, która pomaga nam wyszukiwać wszystkie obiekty `Person`:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- `PersonController` zwróci szablon widoku o nazwie `view-persons` - potrzebujemy `view-persons.jsp` w katalogu `/WEB-INF/jsp/`.

```
1 @Controller
2 @RequestMapping("/person")
3 public class PersonController {
4
5     private final PersonService service;
6
7     public PersonController(PersonService service) {
8         this.service = service;
9     }
10
11     @GetMapping("/viewPersons")
12     public String viewPersons(Model model) {
13         model.addAttribute("persons",
14             ↪ service.getPersons());
15         return "view-persons";
16     }
17 }
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.

```
1 <%@ page contentType="text/html; charset=UTF-8"
  ↳ language="java" %>
2 <%@taglib prefix="c"
  ↳ uri="http://java.sun.com/jsp/jstl/core"%>
3 <html>
4   <head>
5     <title>View People</title>
6     <link href="<c:url value="/css/common.css"/>"
      ↳ rel="stylesheet" type="text/css">
7   </head>
8   <body>
9     <table>
10      <thead>
11        <tr>
12          <th>ID</th>
13          <th>First Name</th>
14          <th>Last Name</th>
15          <th>Last Name</th>
16        </tr>
17      </thead>
18      <tbody>
19        <c:forEach items="${personss}"
      ↳ var="person">
20          <tr>
21            <td>${person.id}</td>
22            <td>${person.firstName}</td>
23            <td>${person.lastName}</td>
24          </tr>
25        </c:forEach>
26      </tbody>
27    </table>
28  </body>
29 </html>
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy

```
1 public class PersonController {
2
3     // (.juz istniejacy kod
4
5     @GetMapping("/addPerson")
6     public String addPersonView(Model model) {
7         model.addAttribute("person", new Person());
8         return "add-person";
9     }
10
11     @PostMapping("/addPerson")
12     public RedirectView
13     ↪ addPerson(@ModelAttribute("person")
14     ↪ Person person, RedirectAttributes
15     ↪ redirectAttributes) {
16         final RedirectView redirectView = new
17         ↪ RedirectView("/person/addPerson", true);
18         Person savedPerson =
19         ↪ personService.addPerson(person);
20
21         ↪ redirectAttributes.addFlashAttribute("savedPerson",
22         ↪ savedPerson);
23
24         ↪ redirectAttributes.addFlashAttribute("addPersonSuccess",
25         ↪ true);
26         return redirectView;
27     }
28 }
```

- Założmy, że mamy usługę PersonService, add-person.jsp która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy

```
1 <%@ taglib prefix="c"
   ↳ uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@ taglib prefix="form"
   ↳ uri="http://www.springframework.org/tags/form" %>
3 <%@ page contentType="text/html; charset=UTF-8"
   ↳ language="java" %>
4 <html>
5   <head>
6     <title>Add Person</title>
7   </head>
8   <body>
9     <c:if test="${addPersonSuccess}">
10       <div>Successfully added Person with ID:
11         ↳ ${savedPerson.id}</div>
12     </c:if>
13
14     <c:url var="add_person_url"
15       ↳ value="/person/addPerson"/>
16     <form:form action="${add_person_url}"
17       ↳ method="post" modelAttribute="person">
18       <form:label path="Id">ID: </form:label>
19       <form:input type="text" path="id"/>
20       <form:label path="name">First Name:
21         ↳ </form:label>
22       <form:input type="text" path="firstName"/>
23       <form:label path="name">Last Name:
24         ↳ </form:label>
25       <form:input type="text" path="lastName"/>
26       <input type="submit" value="submit"/>
27     </form:form>
28   </body>
29 </html>
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy
- **Obsługa Błędów**

```
1 public class DuplicatePersonException extends
   ↳ RuntimeException {
2     private final Person person;
3
4     public DuplicatePersonException(Person person) {
5         this.person = person;
6     }
7
8     // getter methods
9 }
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy
- Obsługa Błędów

```
1 @Service
2 public class personServiceImpl implements personService
3     ↪ {
4     private final personRepository personRepository;
5
6     // constructors, other override methods
7
8     @Override
9     public person addperson(person person) {
10         final Optional<personData> existingperson =
11             ↪ personRepository.findById(person.getId());
12         if (existingperson.isPresent()) {
13             throw new DuplicatepersonException(person);
14         }
15
16         final personData savedperson =
17             ↪ personRepository.add(convertperson(person));
18         return convertpersonData(savedperson);
19     }
20
21     // conversion logic
22 }
```

- Założmy, że mamy usługę PersonService, która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy
- Obsługa Błędów

```
1 @ControllerAdvice
2 public class LibraryControllerAdvice {
3
4     @ExceptionHandler(value =
5         ↳ DuplicatepersonException.class)
6     public ModelAndView
7         ↳ duplicatepersonException(DuplicatepersonException
8         ↳ e) {
9         final ModelAndView modelAndView = new
10             ↳ ModelAndView();
11         modelAndView.addObject("ref",
12             ↳ e.getperson().getId());
13         modelAndView.addObject("object",
14             ↳ e.getperson());
15         modelAndView.addObject("message", "Cannot add
16             ↳ an already existing person");
17         modelAndView.setViewName("error-person");
18         return modelAndView;
19     }
20 }
```

- Założmy, że mamy usługę PersonService, error.person.jsp która pomaga nam wyszukiwać wszystkie obiekty Person:
- Możemy napisać kontroler Spring MVC, aby udostępnić to jako stronę internetową:
- PersonController zwróci szablon widoku o nazwie view-persons - potrzebujemy view-persons.jsp w katalogu /WEB-INF/jsp/.
- Obsługa formularzy
- Obsługa Błędów

1

- The Domain Layer

```
1 package sbthyme;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7 import lombok.AllArgsConstructor;
8 import lombok.Data;
9
10 @Entity
11 @Data
12 @AllArgsConstructor
13 public class Employee {
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     private long id;
17
18     //@NotBlank(message = "Name is mandatory")
19     private String name;
20
21     //@NotBlank(message = "Email is mandatory")
22     private String email;
23 }
```

- The Domain Layer
- @0+:The Repository Layer

```
1 package sbthyme;
2
3 import org.springframework.data.repository.CrudRepository;
4 import org.springframework.stereotype.Repository;
5
6 @Repository
7 public interface EmployeeRepository extends
8     ↳ CrudRepository<Employee, Long> {
9 }
```

- The Domain Layer
- @0+:Load Database

```
1 package sbthyme;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13
14     @Bean
15     CommandLineRunner initDatabase(EmployeeRepository repository) {
16         return args -> {
17             log.info("Preloading " + repository.save(
18                 new Employee(1, "Jeremy", "Clarkson")));
19             log.info("Preloading " + repository.save(
20                 new Employee(3, "James", "May")));
21             log.info("Preloading " + repository.save(
22                 new Employee(7, "Richard", "Hammond")));
23         };
24     }
25 }
```

- The Domain Layer

- @0+:Spring Boot uruchomi WSZYSTKIE ziarna CommandLineRunner po załadowaniu kontekstu aplikacji.

```
1 package sbthyme;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13
14     @Bean
15     CommandLineRunner initDatabase(EmployeeRepository repository) {
16         return args -> {
17             log.info("Preloading " + repository.save(
18                 new Employee(1, "Jeremy", "Clarkson")));
19             log.info("Preloading " + repository.save(
20                 new Employee(3, "James", "May")));
21             log.info("Preloading " + repository.save(
22                 new Employee(7, "Richard", "Hammond")));
23         };
24     }
25 }
```

- The Domain Layer
 - @0+:CommandLineRunner
dostanie referencję do utworzonego właśnie repozytorium

```
1 package sbthyme;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 public class LoadDatabase {
11     private static final Logger log
12         = LoggerFactory.getLogger(LoadDatabase.class);
13
14     @Bean
15     CommandLineRunner initDatabase(EmployeeRepository repository) {
16         return args -> {
17             log.info("Preloading " + repository.save(
18                 new Employee(1, "Jeremy", "Clarkson")));
19             log.info("Preloading " + repository.save(
20                 new Employee(3, "James", "May")));
21             log.info("Preloading " + repository.save(
22                 new Employee(7, "Richard", "Hammond")));
23         };
24     }
25 }
```

- The Domain Layer
- @0+:Controller Layer

```
1 package sbthyme;
2 import org.springframework.stereotype.Controller;
3 import org.springframework.ui.Model;
4 import org.springframework.validation.BindingResult;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PostMapping;
7
8 @Controller
9 public class EmployeeController {
10     private final EmployeeRepository repository;
11     EmployeeController(EmployeeRepository repository) {
12         this.repository = repository;
13     }
14
15     @GetMapping("/add")
16     public String showAddEmployeeForm(Employee employee) {
17         return "add-employee";
18     }
19
20     @PostMapping("/add")
21     public String addUser(/*@Valid*/ Employee employee,
22                          BindingResult result, Model model) {
23         if (result.hasErrors()) {
24             return "add-user";
25         }
26         repository.save(employee);
27         return "redirect:/index";
28     }
29 }
```

- The Domain Layer

```
1 @GetMapping("/add")
2 public String showAddEmployeeForm(Employee employee) {
3     return "add-employee";
4 }
5
6 @PostMapping("/add")
7 public String addUser(/*@Valid*/ Employee employee,
8                      BindingResult result, Model model) {
9     if (result.hasErrors()) {
10         return "add-user";
11     }
12     repository.save(employee);
13     return "redirect:/index";
14 }
15 @GetMapping("/index")
16 public String showEmployeeList(Model model) {
17     model.addAttribute("employees", repository.findAll());
18     return "index";
19 }
```