Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

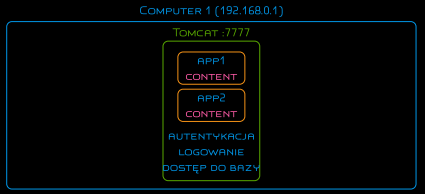
Web

Julian Myrcha
Institute of Computer Science
26.02.2025

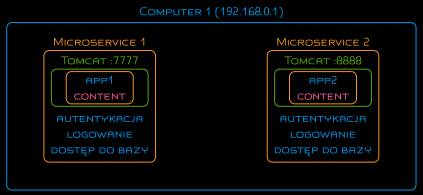
WUT

- W standardowym podejściu mamy monolityczne serwery hostujące wiele aplikacji

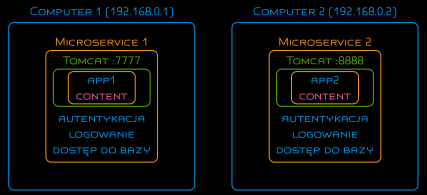
- podejście tradycyjne
- w jednym miejscu zarządzanie użytkownikami i konfiguracja środowiska



- 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 mikroserwisów - 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

```
jmy@bardex:$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /FirstWebApp/first HTTP/1.1
HOST: localhost

HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1.1  Java/Oracle Corporation/1.8)
Date: Fri, 21 Oct 2016 14:39:03 GMT
Content-Length: 23

Served at: /FirstWebAppConnection closed by foreign host.
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

```
jmy@bequiet ~
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
>
<HTML>
<HEAD>
  <META http-equiv=Content-Type content="text/html; charset=iso-8859-2">
  <link rel="stylesheet" type="text/css" href="/galera.css">
  <TITLE>Serwer galera.ii.pw.edu.pl</TITLE>
</HEAD>

<BODY>
<CENTER>
...
</HTML>
```

Przykład strony z formą html-ową:

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8">
        <title>EGUI</title>
</head>
<body>
<h2>Formy</h2>

<form action="first">
  Imie:<br>
  <input type="text" name="firstname" value="Jas">
  <br>
  Nazwisko:<br>
  <input type="text" name="lastname" value="Fasola">
  <br><br>
  <input type="submit" name = 'btn' value="Wyslij">
  <input type="submit" name = 'btn' value="Pomin">
</form>

<p>jak nacisniesz przycisk "Wyslij", dane zostana przeslane do strony "first".</p>
</body>
</html>
```

Do serwera idą dane:

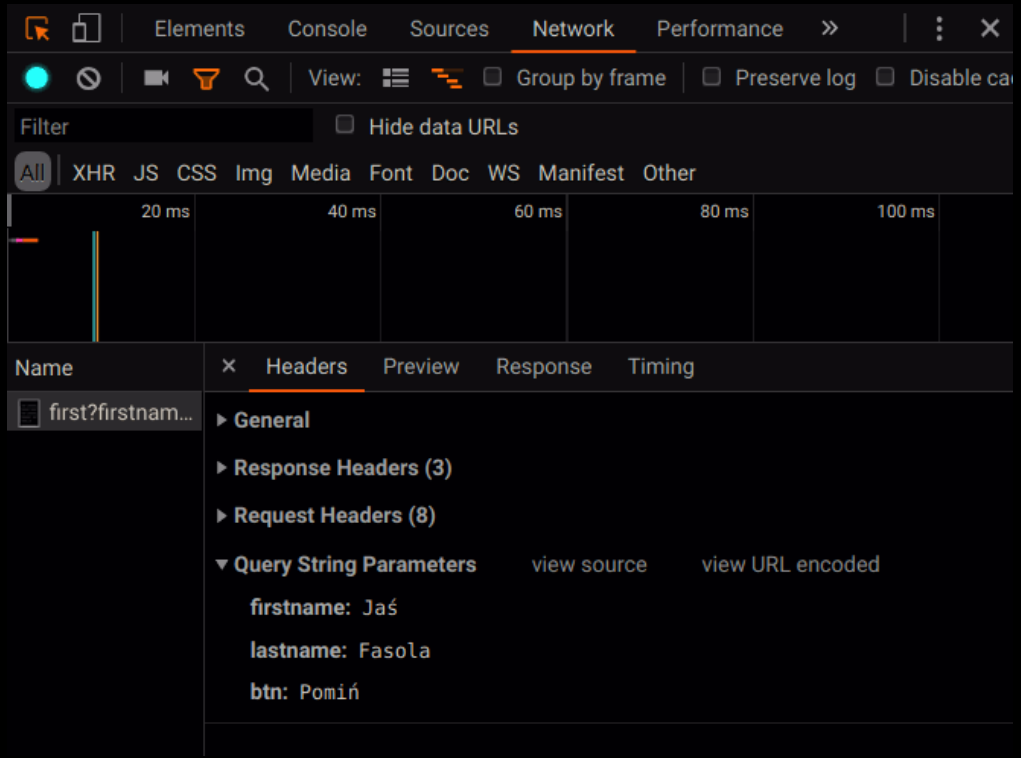
firstname -Jaś

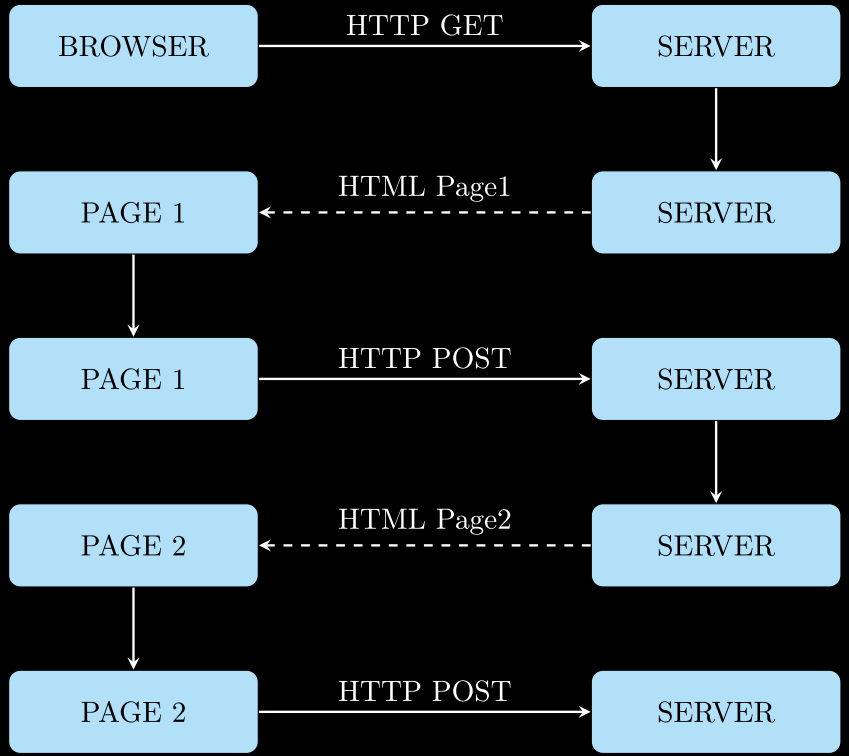
lastname -Fasola

btn -Wyślij

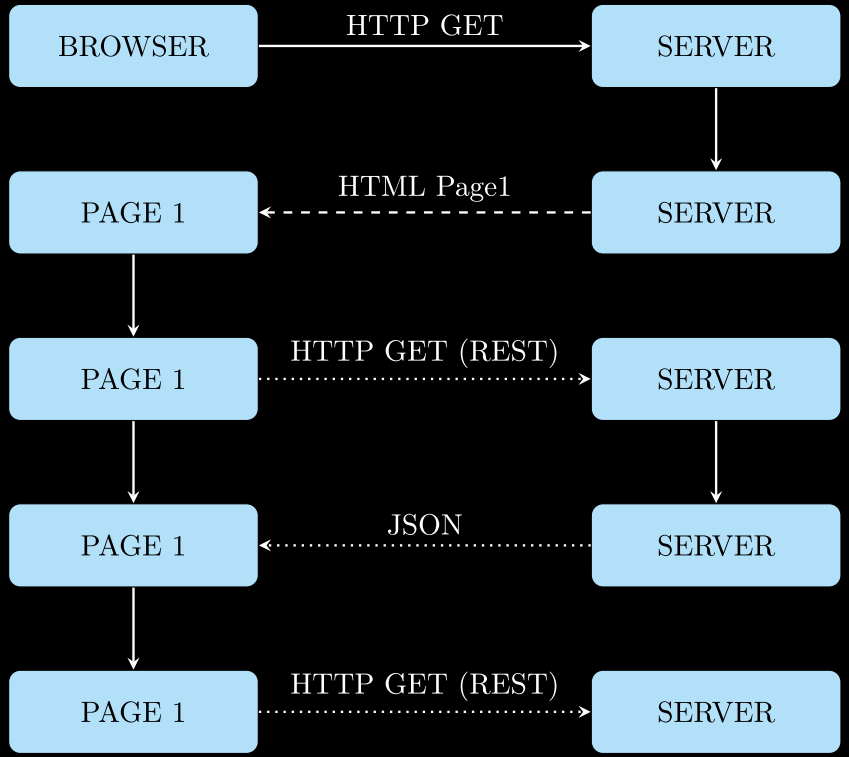
Dla elementów typu submit w danych występuje tylko naciśnięty przycisk!

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

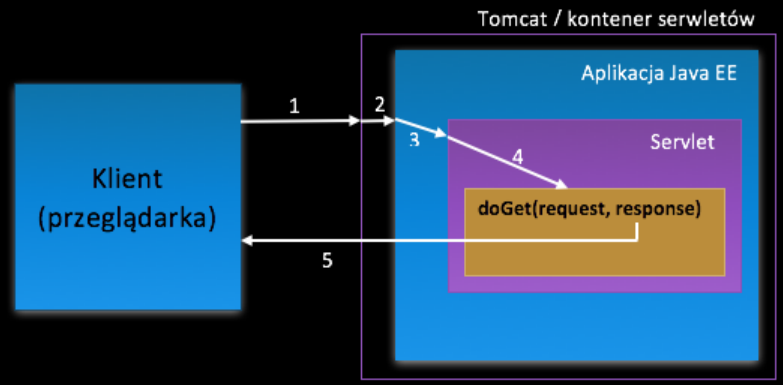



BROWSER
HTTP GET
SERVER
PAGE 1
HTML Page1
SERVER
PAGE 1
HTTP POST
SERVER
PAGE 2
HTML Page2
SERVER
PAGE 2
HTTP POST
SERVER

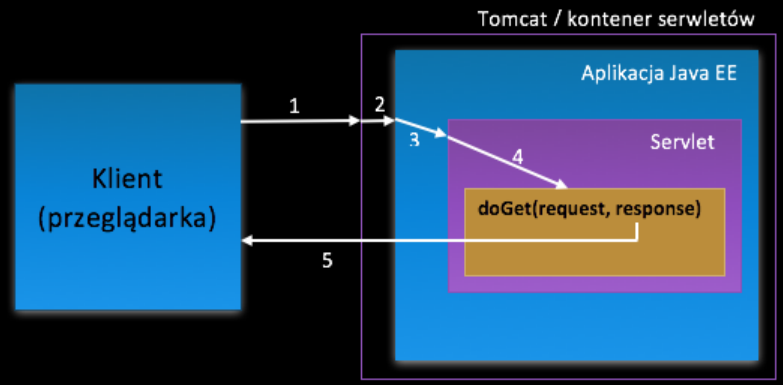
```
<!DOCTYPE html>
<html>
<body>

<h2>My First JavaScript</h2>

<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>

<p id="demo"></p>

</body>
</html>
```


BROWSER
HTTP GET
SERVER
PAGE 1
HTML Page1
SERVER
PAGE 1
HTTP GET (REST)
SERVER
PAGE 1
JSON
SERVER
PAGE 1
HTTP GET (REST)
SERVER

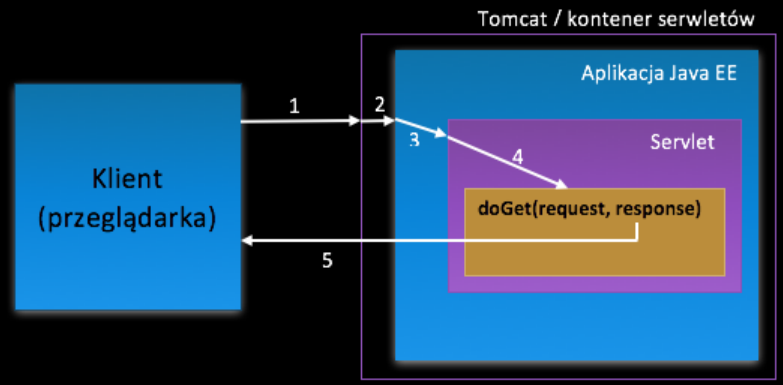
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
  - doGet(), doPost(), doTrace(), doHead(), doPut(), doOptions()
- cykl życia servletu:
  - utworzenie przez konstruktor
  - wywołanie metody init()
  - wielokrotne wywoływanie metod doGet, doPost itp.
  - wywołanie metody destroy()



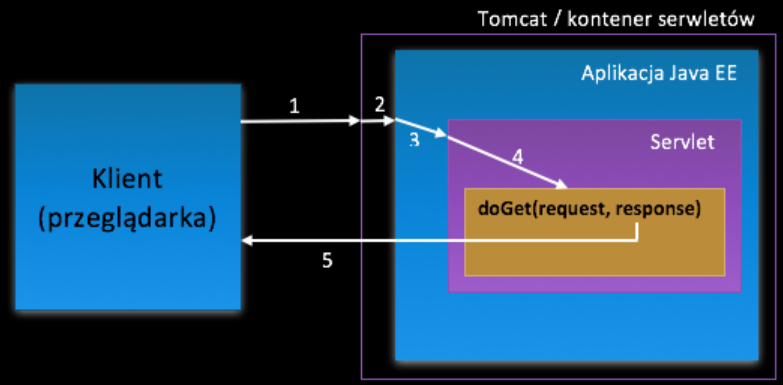
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
  - doGet(), doPost(), doTrace(), doHead(), doPut(), doOptions()
- cykl życia servletu:
  - utworzenie przez konstruktor
  - wywołanie metody init()
  - wielokrotne wywoływanie metod doGet, doPost itp.
  - wywołanie metody destroy()



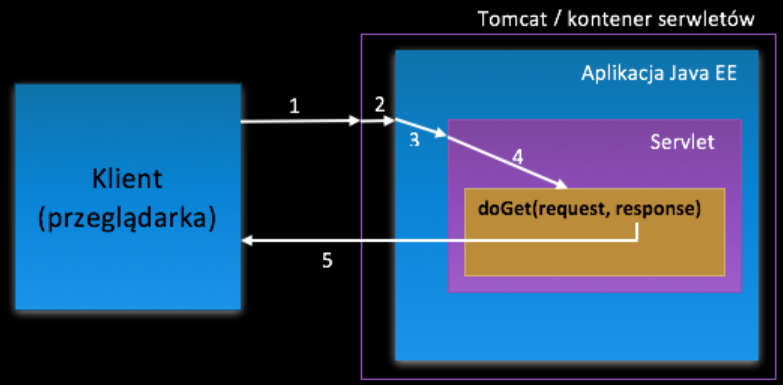
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
  - doGet(), doPost(), doTrace(), doHead(), doPut(), doOptions()
- cykl życia servletu:
  - utworzenie przez konstruktor
  - wywołanie metody init()
  - wielokrotne wywoływanie metod doGet, doPost itp.
  - wywołanie metody destroy()



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



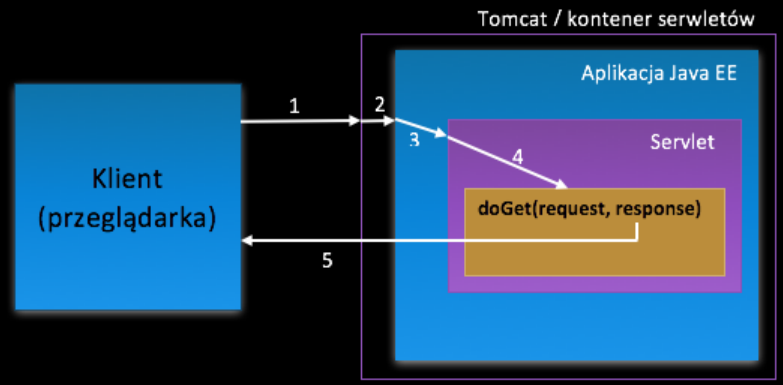
- serwlet za pomocą odpowiednich metod tworzy odpowiedź
  - doGet(), doPost(), doTrace(), doHead(), doPut(), doOptions()
- cykl życia servletu:
  - utworzenie przez konstruktor
  - wywołanie metody init()
  - wielokrotne wywoływanie metod doGet, doPost itp.
  - wywołanie metody destroy()



servlet będzie współdzielony pomiędzy wszystkich użytkowników naszej aplikacji

- wielowątkowość

- serwlet za pomocą odpowiednich metod tworzy odpowiedź
  - doGet(), doPost(), doTrace(), doHead(), doPut(), doOptions()
- cykl życia servletu:
  - 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
  artefactID : webapp

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

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

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

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

```
<properties>
   ...

   ↪  <jakartaee-api.version>9.1.0</jakartaee-api.version>
</properties>
<dependencies>
  ...
  <dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>${jakartaee-api.version}</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

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

```java
package pw.atj;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/helloWorld")
public class HelloWorld extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public HelloWorld() {
    super();
  }
  protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
    String name = request.getParameter("name").trim();
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<h2>Hello " + name + "</h2>");
    out.close();
  }
  protected void doPost(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
    // TODO Auto-generated method stub
    doGet(request, response);
  }
}
```

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

```java
package pw.atj;

import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class ApiServlet extends HttpServlet {

  private static final long serialVersionUID = 1L;

  public void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

      response.setContentType("application/json");
      PrintWriter out = response.getWriter();
      out.println("{ \"message\": \"Hello World!\" }");
  }
}
```

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

```xml
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪ xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
  https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
  version="5.0">

  <display-name>Archetype Created Web
  ↪ Application</display-name>
  <servlet>
    <servlet-name>ApiServlet</servlet-name>
    <servlet-class>pw.atj.ApiServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ApiServlet</servlet-name>
    <url-pattern>/api/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

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

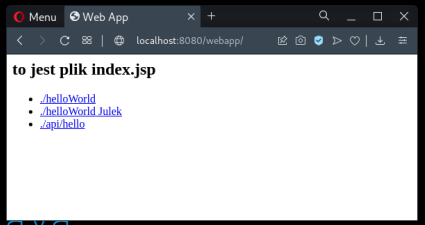
```
<head>
  <title>Web App</title>
</head>

<body>
  <h2>to jest plik index.jsp</h2>
  <ul>
    <li><a href='./helloWorld'> ./helloWorld</a></li>
    <li><a href='./helloWorld?name=Julek'> ./helloWorld
    ↪  Julek</a></li>
    <li><a href='./api/hello'> ./api/hello</a></li>
  </ul>
</body>
</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 plik `src/main/webapp/WEB-INF/web-xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: package`

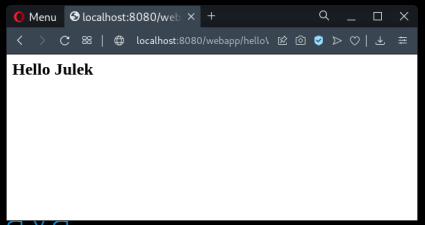
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.java`:
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web-xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: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

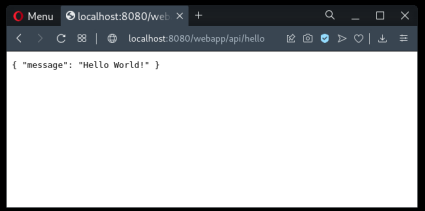
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.j`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web-xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: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



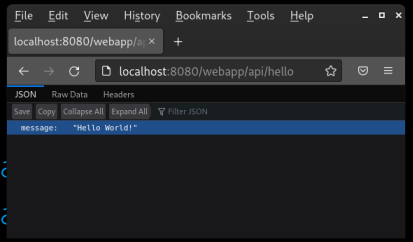
- wybieramy `maven-archetype-webapp` w wersji 1.4
- modyfikujemy plik `pom.xml`:
- dodajemy plik `src/main/java/pw/atj/HelloWorld.j`
- dodajemy plik `src/main/java/pw/atj/ApiServlet.java`:
- modyfikujemy plik `src/main/webapp/WEB-INF/web-xml`:
- modyfikujemy plik `src/main/webapp/index.jsp`:
- `maven: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

```
<context-param>
    <param-name>myParam</param-name>
    <param-value>the value</param-value>
</context-param>
```

Przykład odczytu:

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

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

```java
package pl.edu.pw.atjtomcat;

import java.io.*;

import jakarta.servlet.http.*;
import jakarta.servlet.annotation.*;

@WebServlet(name = "helloServlet", value = "/hello-servlet",
    initParams = {
      @WebInitParam(name = "message", value = "Hello World")
    })
public class HelloServlet extends HttpServlet {
  private String message;

  public void init() {
      message = getInitParameter("message");
  }

  public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h1>" + message + "</h1>");
    out.println("</body></html>");
  }

  public void destroy() {
  }
}
```

- Można użyć wpisu w pliku web.config

```
<servlet>
  <servlet-name>CalcServlet</servlet-name>
  <servlet-class>pap.CalcServlet</servlet-class>
  <init-param><param-name>saveDir</param-name>
    <param-value>/home/jmy/Downloads</param-
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

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

```java
package pl.edu.pw.atjtomcat;

import java.io.*;

import jakarta.servlet.http.*;
import jakarta.servlet.annotation.*;

@WebServlet(name = "helloServlet", value = "/hello-servlet",
    initParams = {
      @WebInitParam(name = "message", value = "Hello World")
    })
public class HelloServlet extends HttpServlet {
  private String message;

  public void init() {
      message = getInitParameter("message");
  }

  public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h1>" + message + "</h1>");
    out.println("</body></html>");
  }

  public void destroy() {
  }
}
```

- Można użyć wpisu w pliku web.config

```xml
<servlet>
  <servlet-name>CalcServlet</servlet-name>
  <servlet-class>pap.CalcServlet</servlet-class>
  <init-param><param-name>saveDir</param-name>
        <param-value>/home/jmy/Downloads</param-
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

- Możemy przekazać parametry

```java
@WebServlet(
        urlPatterns = "/imageUpload",
        initParams =
        {
            @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
            @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
        }
)
public class ImageUploadServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        String saveDir = getInitParameter("saveDir");
        String fileTypes = getInitParameter("allowedTypes");

        PrintWriter writer = response.getWriter();

        writer.println("saveDir = " + saveDir);
        writer.println("fileTypes = " + fileTypes);
    }
}
```

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

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Implements Filter class
public class LogFilter implements Filter  {
   public void  init(FilterConfig config) throws ServletException {

      // Get init parameter
      String testParam = config.getInitParameter("test-param");

      //Print the init parameter
      System.out.println("Test Param: " + testParam);
   }

   public void  doFilter(ServletRequest request, ServletResponse response,
      FilterChain chain) throws java.io.IOException, ServletException {

      // Get the IP address of client machine.
      String ipAddress = request.getRemoteAddr();

      // Log the IP address and current timestamp.
      System.out.println("IP "+ ipAddress + ", Time " + new Date().toString());

      // Pass request back down the filter chain
      chain.doFilter(request,response);
   }

   public void destroy( ) {
```

```
32          /* Called before the Filter instance is removed from service by the web container*/
33      }
34  }
```

- Wykorzystanie adnotacji:

```java
@WebFilter("/admin")
public class MyFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        PrintWriter out = response.getWriter();

        // This will print output on console
        System.out.println(
            "Before filter - Preprocessing before servlet");

        // some authentication if required
        chain.doFilter(request, response);

        // This will print output on console
        System.out.println(
            "After servlet - Following code will execute after running the servlet - PostProcessing");
    }
    @Override
    public void destroy() {

    }
}
```

- Wykorzystanie adnotacji:

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

- Wykorzystanie adnotacji:
- Wykorzystanie pliku web.xml

```
public class LoginFilter implements Filter {
       private static final long serialVersionUID = 1L;

       public void init(FilterConfig filterConfig)
               throws ServletException {

       }

     public void doFilter(ServletRequest request,
                     ServletResponse response, FilterChain chain)
                     throws IOException, ServletException {

       response.setContentType("text/html");
           PrintWriter out = response.getWriter();

           //get parameters from request object.
           String userName = request.getParameter("userName").trim();
           String password = request.getParameter("password").trim();

           //check for null and empty values.
           if(userName == null || userName.equals("") ||
                           password == null || password.equals("")){
                   out.print("Please enter both username " +
                                   "and password. <br/><br/>");
                   RequestDispatcher requestDispatcher =
                           request.getRequestDispatcher("/login.html");
                   requestDispatcher.include(request, response);
           }//Check for valid username and password.
           else if(userName.equals("jai") && password.equals("1234")){
                    chain.doFilter(request, response);
           }else{
                   out.print("Wrong username or password. <br/><br/>");
                   RequestDispatcher requestDispatcher =
                           request.getRequestDispatcher("/login.html");
                   requestDispatcher.include(request, response);
           }
```

```
37        }
38
39      public void destroy() {
40
41      }
42  }
```

- Wykorzystanie adnotacji:
- Wykorzystanie pliku web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">

  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>
            com.w3spoint.business.WelcomeServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/WelcomeServlet</url-pattern>
  </servlet-mapping>

  <filter>
          <filter-name>LoginFilter</filter-name>
          <filter-class>
                  com.w3spoint.business.LoginFilter
          </filter-class>
  </filter>

  <filter-mapping>
          <filter-name>LoginFilter</filter-name>
          <url-pattern>/WelcomeServlet</url-pattern>
  </filter-mapping>

   <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

</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

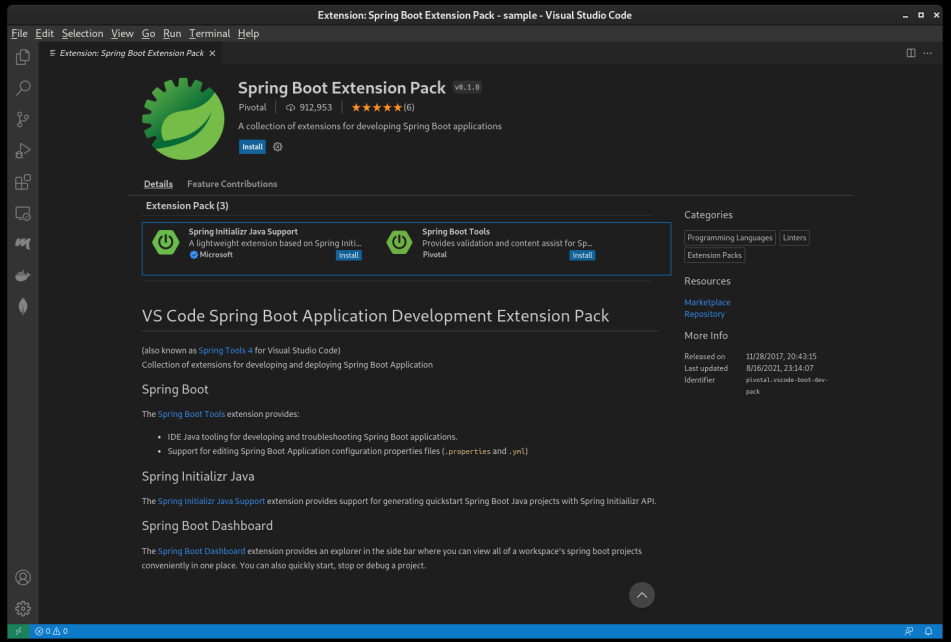
- Używamy POJO - `plain old java object` - wolność wyboru na czym to działa
- Modularne - używamy tylko tej funkcjonalności której chcemy używać
- Łatwe testowanie - wstrzykiwanie zależności ułatwia izolowanie modułów do testowania
- Spring Web Framework - kolejna iteracja frameworka do aplikacji webowych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

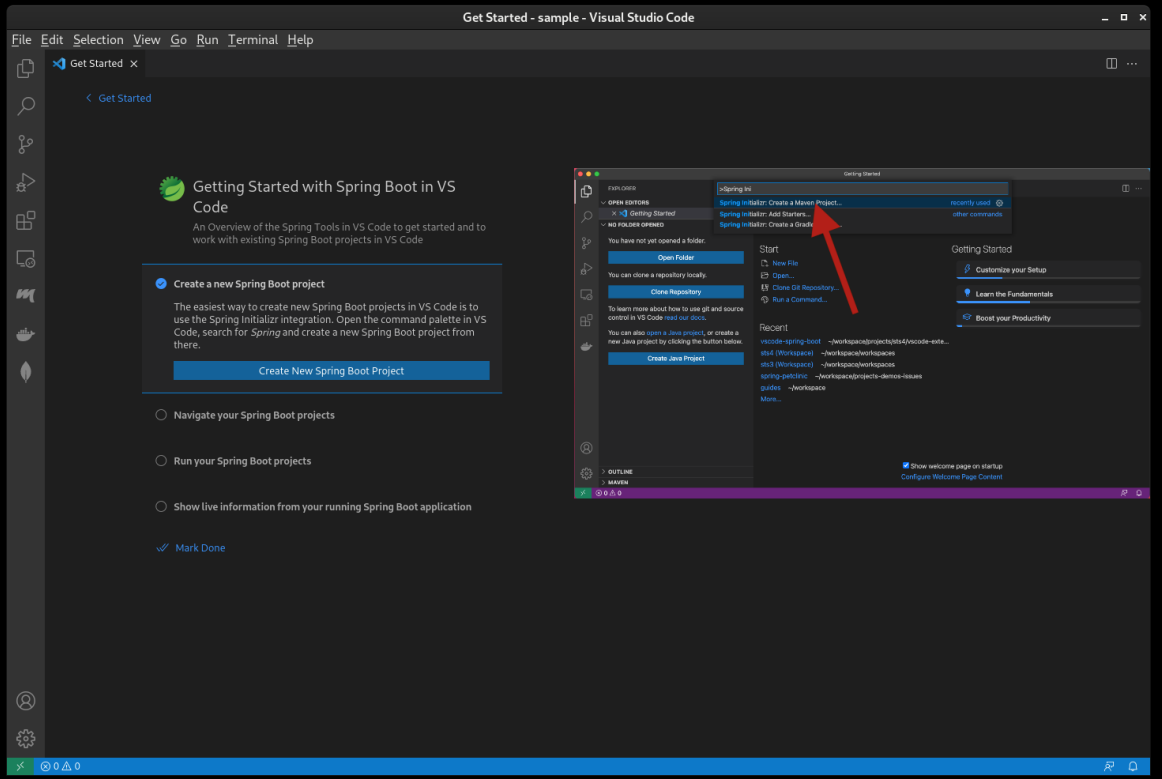
- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :
  - Java Development Kit (JDK)

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

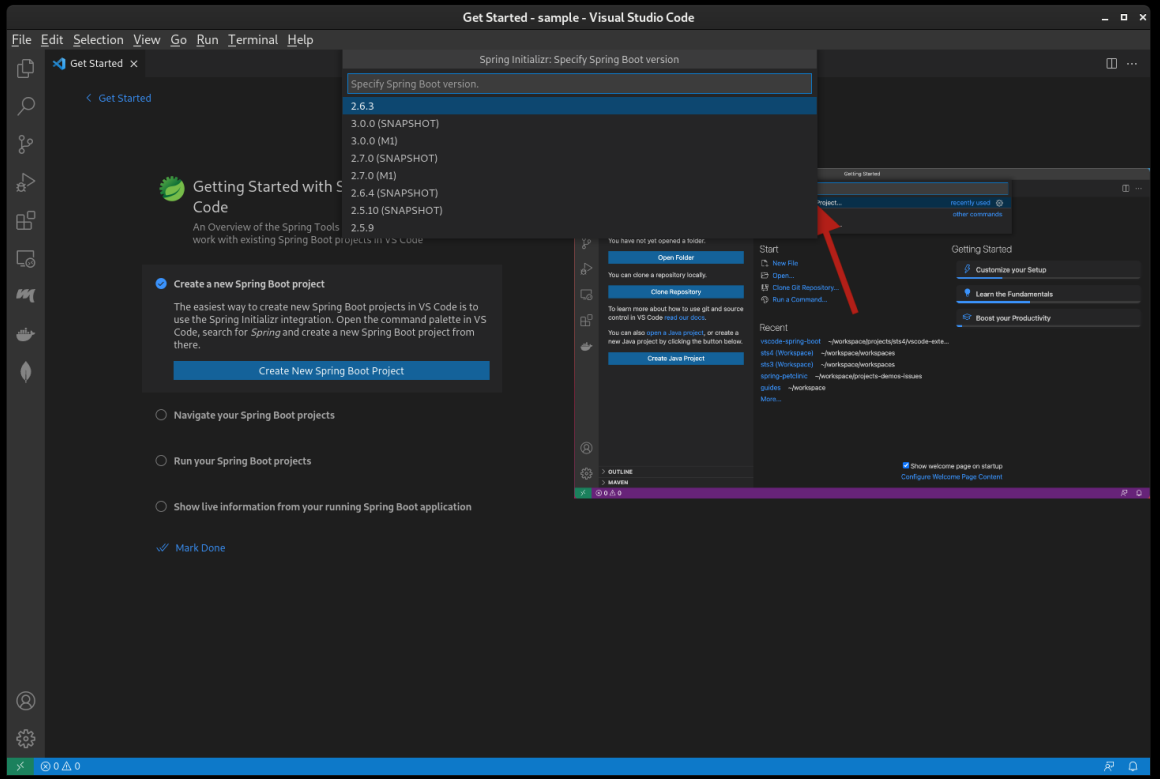
- Aby tworzyć aplikację Spring Boot w Visual Studio Code, trzeba zainstalować :
  - Java Development Kit (JDK)
  - Extension Pack for Java
  - Spring Boot Extension Pack



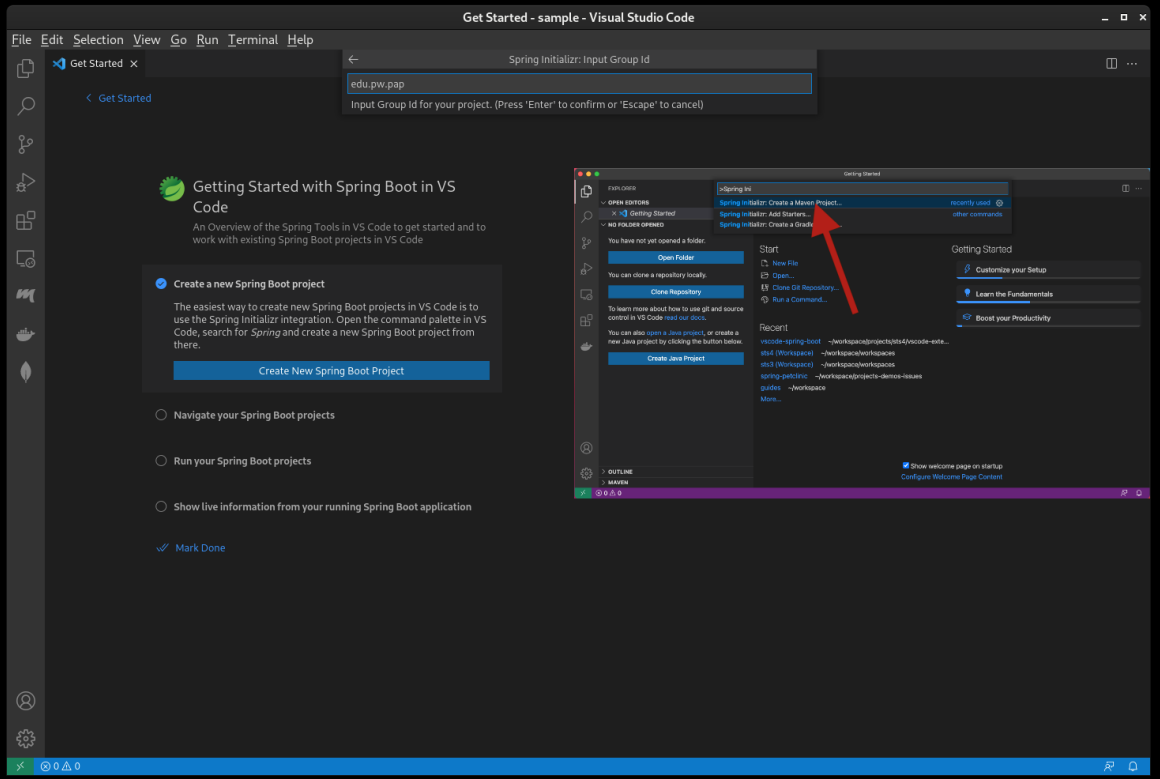
- wybranie czarodzieja



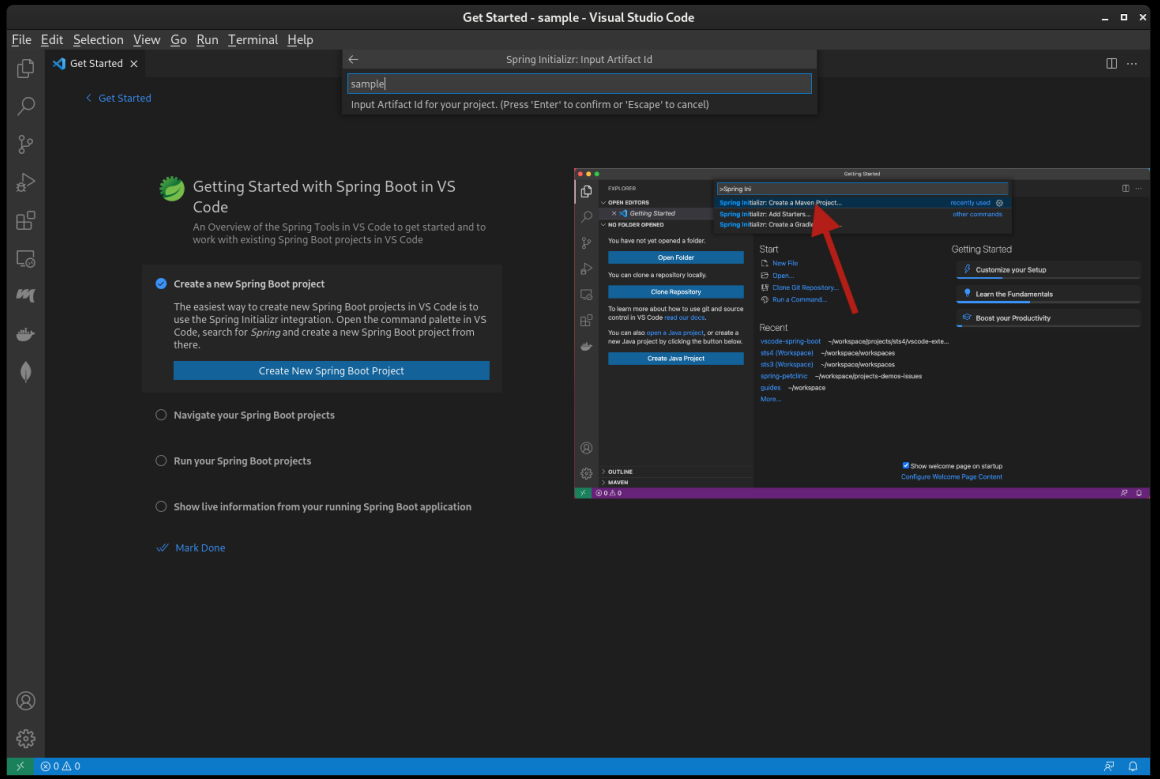
- wybranie czarodzieja
- wybranie wersji spring boota



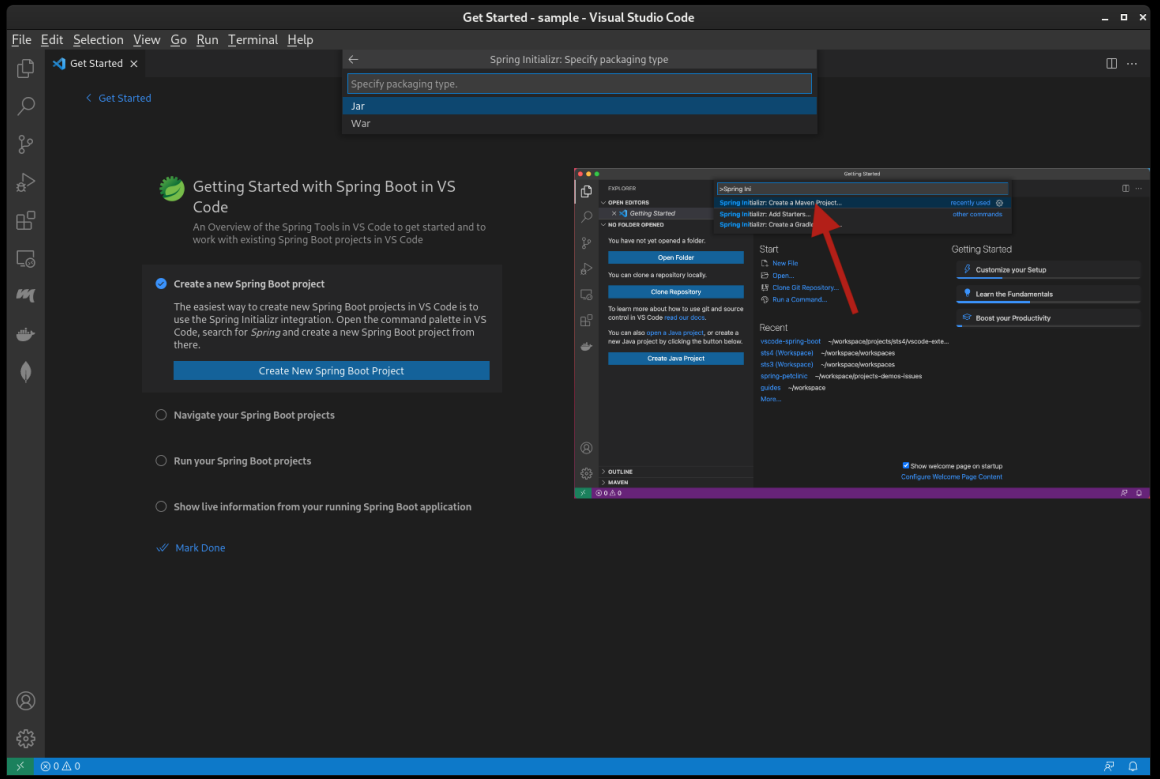
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id



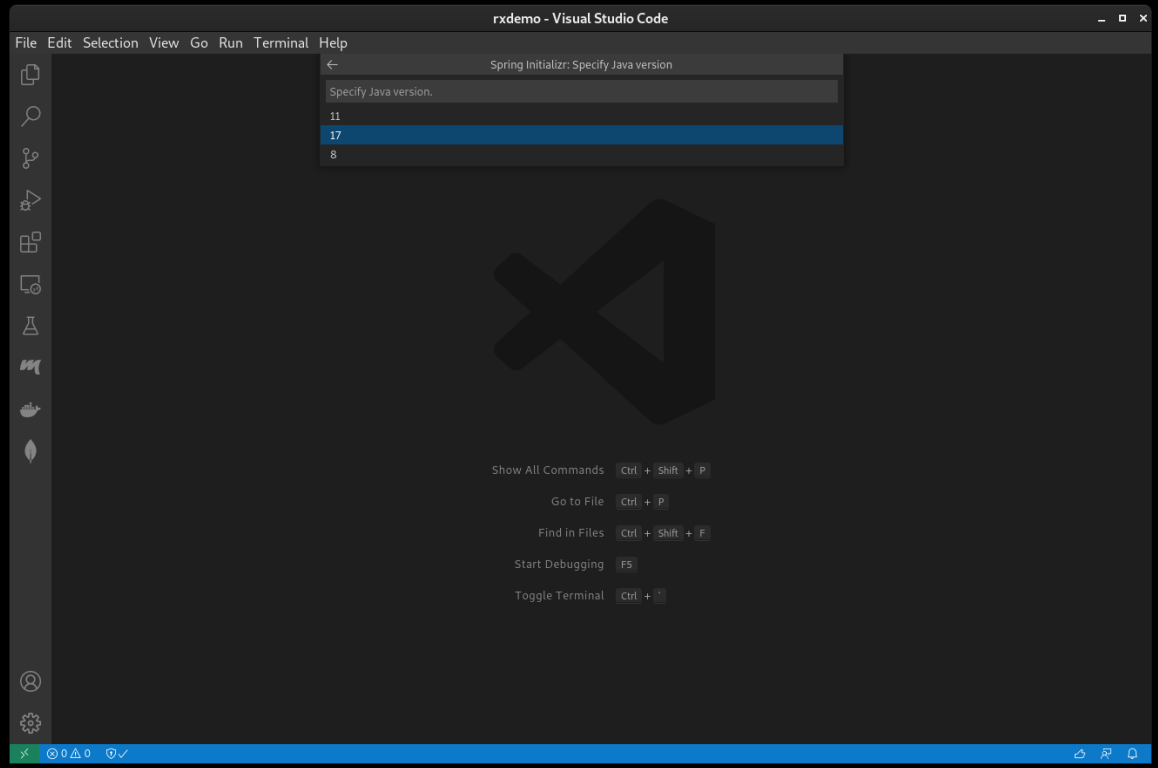
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id



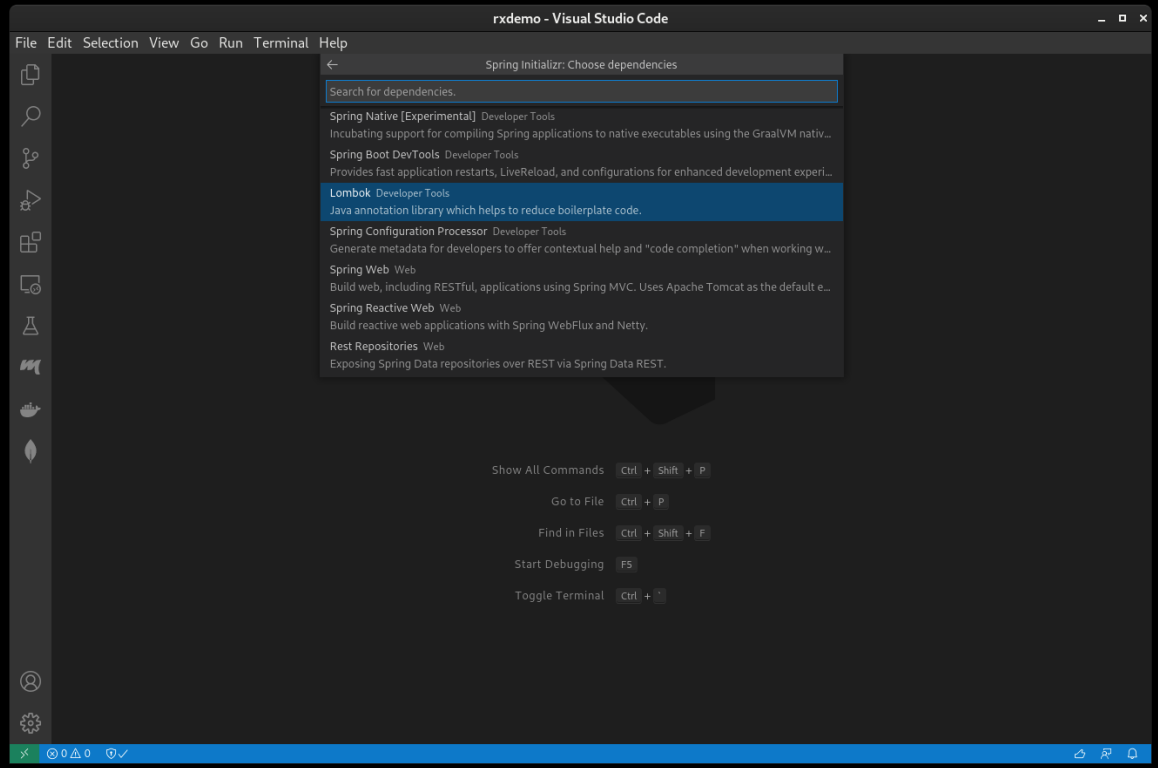
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



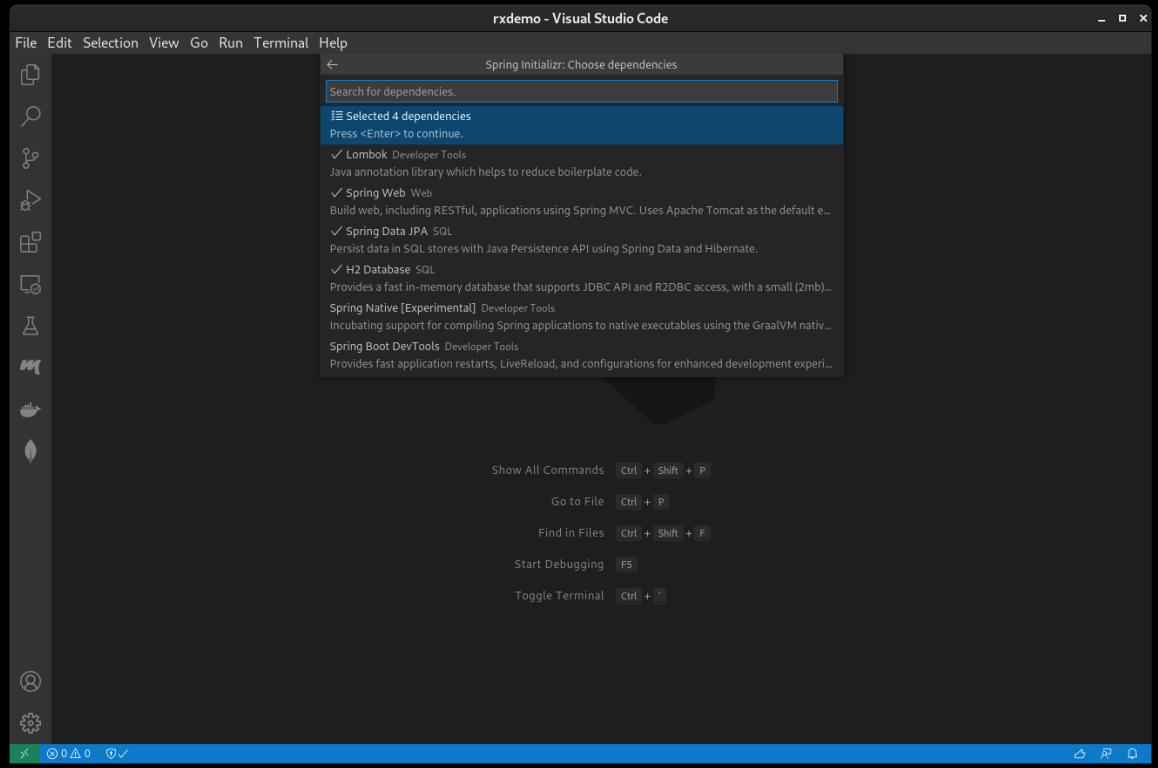
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



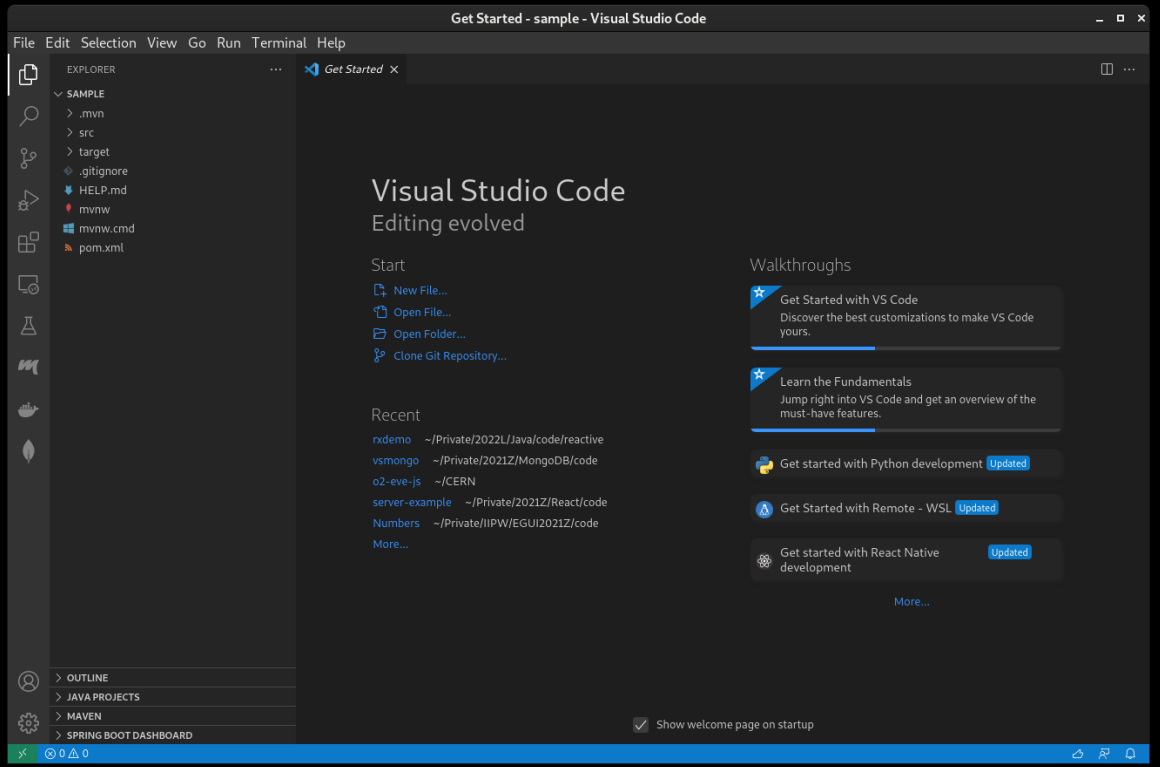
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



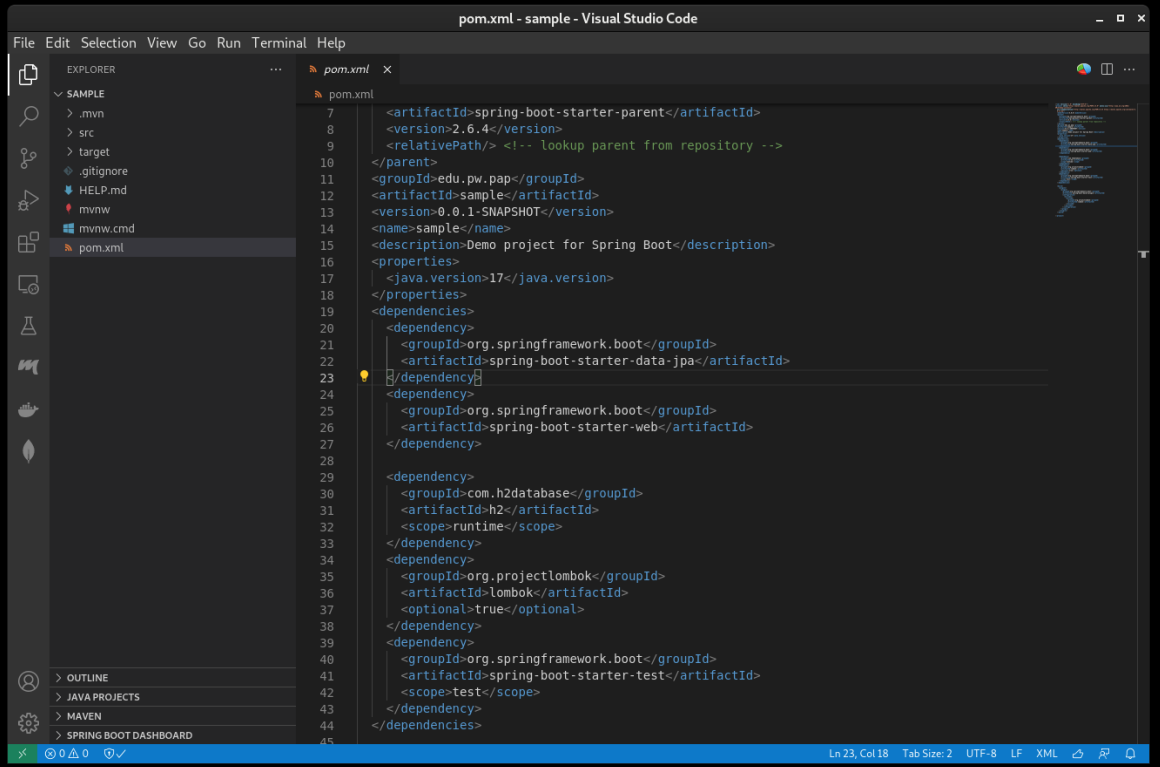
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację



- 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



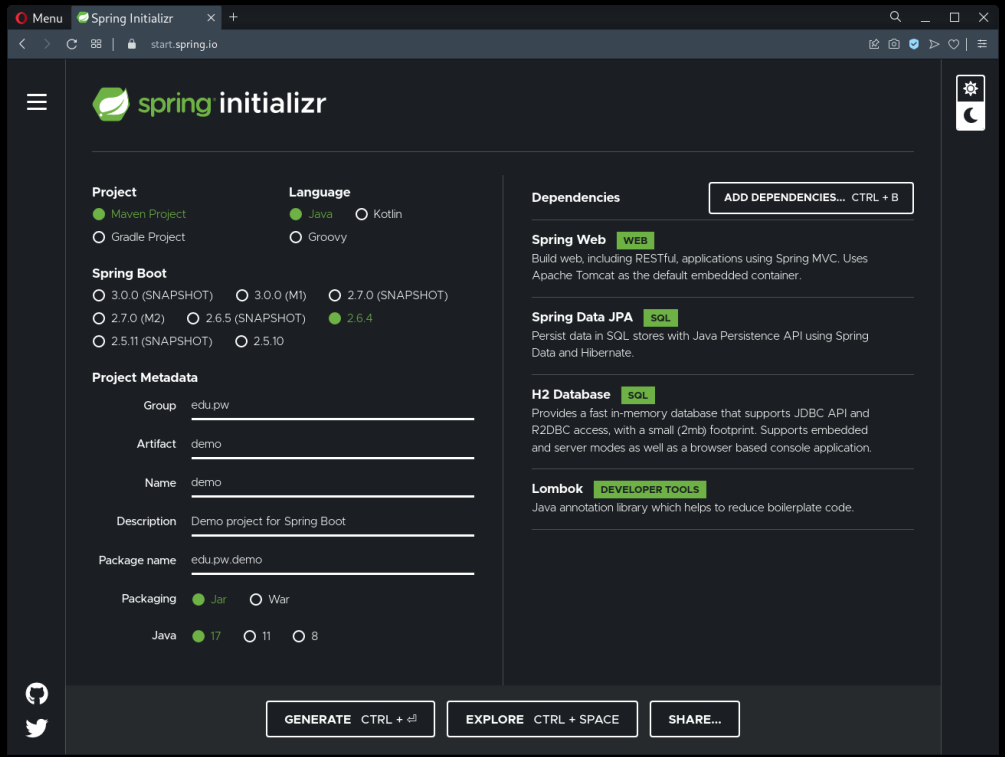
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację
- pom.xml

```xml
<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>
```

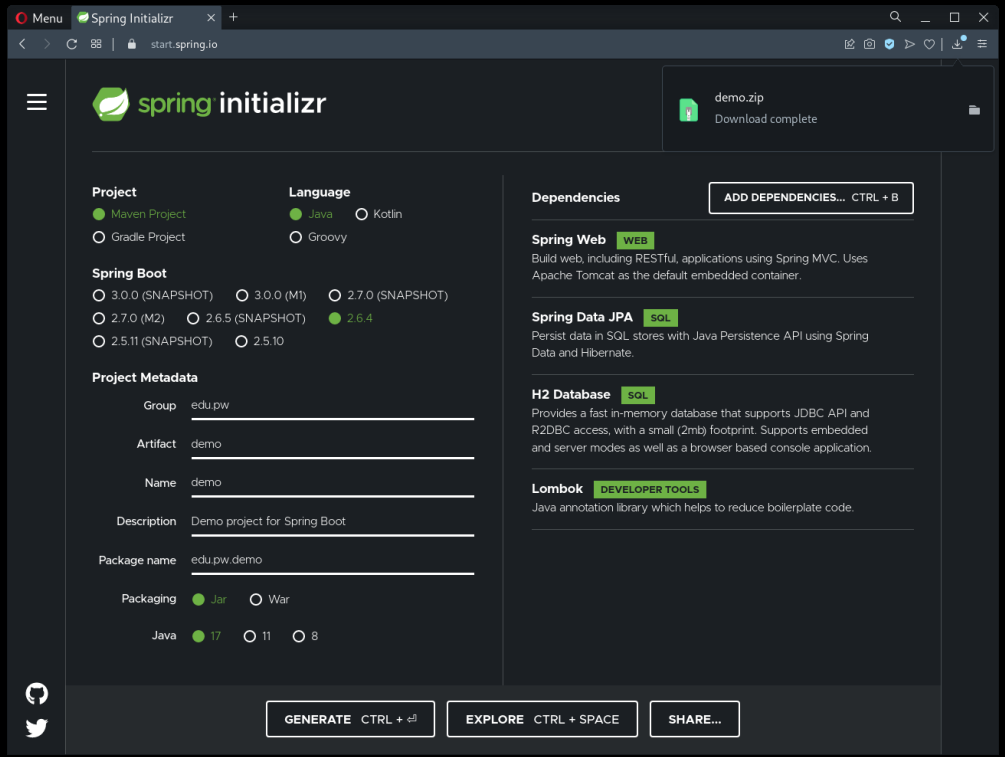
- wybranie czarodzieja
- wybranie wersji spring boota
- maven group id
- maven artefact id
- jak będziemy uruchamiali aplikację
- pom.xml
- java

```java
package edu.pw.pap.sample;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SampleApplication {

	public static void main(String[] args) {
		SpringApplication.run(SampleApplication.class, args);
	}

}
```

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



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



- tworzymy obiekt danych (Entity)

```java
package edu.pw.pap.sample;

import lombok.Data;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
@Data
public class Employee {
  private @Id @GeneratedValue Long id;
  private String name;
  private String role;
}
```

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

```java
package edu.pw.pap.sample;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository
                    extends JpaRepository<Employee, Long> {

}
```

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

```java
package edu.pw.pap.sample;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository
                    extends JpaRepository<Employee, Long> {

}
```

- 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

```java
package edu.pw.pap.sample;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository){
    return args -> {
      log.info("Preloading " + repository.save(
                  new Employee("Bilbo Baggins", "burglar")));
      log.info("Preloading " + repository.save(
                   new Employee("Frodo Baggins", "thief")));
    };
  }
}
```

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

```java
package edu.pw.pap.sample;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository){
    return args -> {
      log.info("Preloading " + repository.save(
                  new Employee("Bilbo Baggins", "burglar")));
      log.info("Preloading " + repository.save(
                   new Employee("Frodo Baggins", "thief")));
    };
  }
}
```

- 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

```java
package edu.pw.pap.sample;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository){
    return args -> {
      log.info("Preloading " + repository.save(
                  new Employee("Bilbo Baggins", "burglar")));
      log.info("Preloading " + repository.save(
                   new Employee("Frodo Baggins", "thief")));
    };
  }
}
```

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

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

Spring MVC

- EmployeeNotFoundExcepti

```
package edu.pw.pap.sample;

class EmployeeNotFoundException extends RuntimeException {

  EmployeeNotFoundException(Long id) {
    super("Could not find employee " + id);
  }
}
```

Spring MVC

- EmployeeNotFoundExcepti
- EmployeeNotFoundAdvice

```java
package edu.pw.pap.sample;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

  @ResponseBody
  @ExceptionHandler(EmployeeNotFoundException.class)
  @ResponseStatus(HttpStatus.NOT_FOUND)
  String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
  }
}

```

Spring MVC

- EmployeeNotFoundExcepti
- EmployeeNotFoundAdvice. **@ResponseBody** : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

```java
package edu.pw.pap.sample;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

  @ResponseBody
  @ExceptionHandler(EmployeeNotFoundException.class)
  @ResponseStatus(HttpStatus.NOT_FOUND)
  String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
  }
}

```

Spring MVC

- EmployeeNotFoundExcepti
- 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

```
package edu.pw.pap.sample;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

  @ResponseBody
  @ExceptionHandler(EmployeeNotFoundException.class)
  @ResponseStatus(HttpStatus.NOT_FOUND)
  String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
  }
}

```

Spring MVC

- EmployeeNotFoundExcepti
- 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,

```java
package edu.pw.pap.sample;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

  @ResponseBody
  @ExceptionHandler(EmployeeNotFoundException.class)
  @ResponseStatus(HttpStatus.NOT_FOUND)
  String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
  }
}

```

Spring MVC

- EmployeeNotFoundExcepti
- 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,

```
package edu.pw.pap.sample;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

  @ResponseBody
  @ExceptionHandler(EmployeeNotFoundException.class)
  @ResponseStatus(HttpStatus.NOT_FOUND)
  String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
  }
}

```

Spring MVC

- EmployeeNotFoundExcepti
- 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,

```
package edu.pw.pap.sample;
import java.util.List;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
@RestController
class EmployeeController {

  private final EmployeeRepository repository;
  EmployeeController(EmployeeRepository repository) {
    this.repository = repository;
  }

  // Aggregate root
  // tag::get-aggregate-root[]
  @GetMapping("/employees")
  List<Employee> all() {
    return repository.findAll();
  }
  // end::get-aggregate-root[]
  ....
}
```

Spring MVC

- EmployeeNotFoundExcepti
- 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,

```
package edu.pw.pap.sample;
....
@RestController
class EmployeeController {

  private final EmployeeRepository repository;
  ....
  @PostMapping("/employees")
  Employee newEmployee(@RequestBody Employee newEmployee) {
    return repository.save(newEmployee);
  }

  // Single item

  @GetMapping("/employees/{id}")
  Employee one(@PathVariable Long id) {

    return repository.findById(id)
        .orElseThrow(() -> new EmployeeNotFoundException(id));
  }

}
```

Spring MVC

- EmployeeNotFoundExcepti
- EmployeeNotFoundAdvice

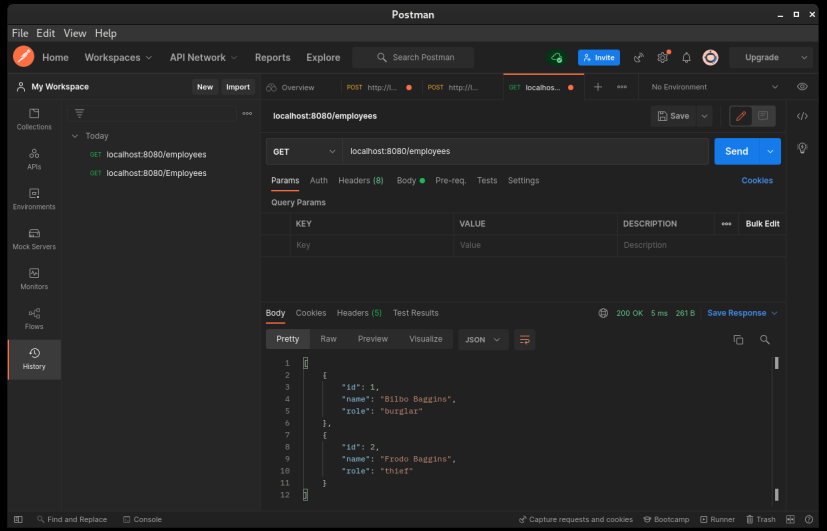
@ResponseBody : sygnalizuje, że ta rada jest przekazywana bezpośrednio do treści odpowiedzi

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

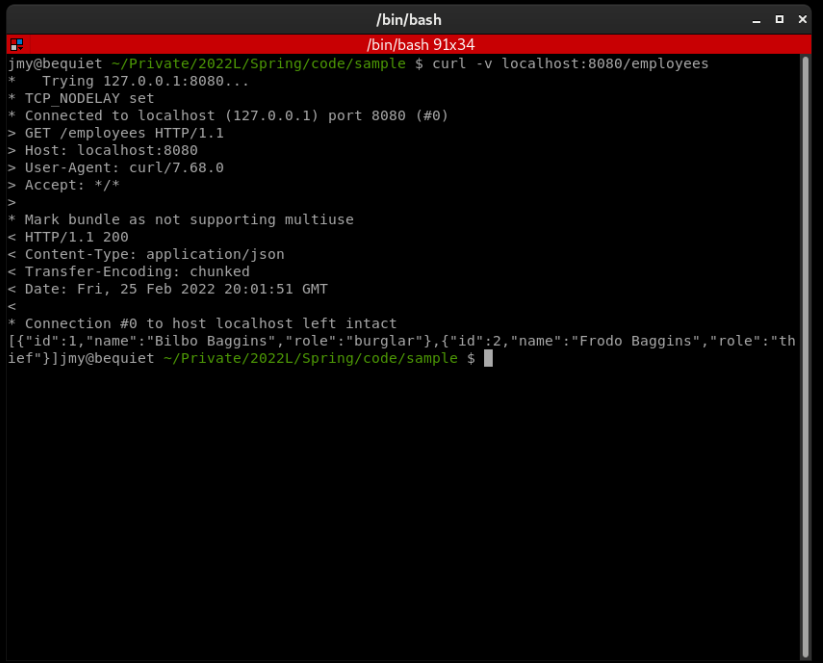
@ResponseStatus : mówi, aby ustawić HttpStatus.NOT_FOUND,

```
package edu.pw.pap.sample;
....
@RestController
class EmployeeController {

  private final EmployeeRepository repository;
  ....
  @PutMapping("/employees/{id}")
  Employee replaceEmployee(@RequestBody Employee newEmployee,
                           @PathVariable Long id) {
    return repository.findById(id)
        .map(employee -> {
          employee.setName(newEmployee.getName());
          employee.setRole(newEmployee.getRole());
          return repository.save(employee);
        })
        .orElseGet(() -> {
          newEmployee.setId(id);
          return repository.save(newEmployee);
        });
  }
  @DeleteMapping("/employees/{id}")
  void deleteEmployee(@PathVariable Long id) {
    repository.deleteById(id);
  }
}
```

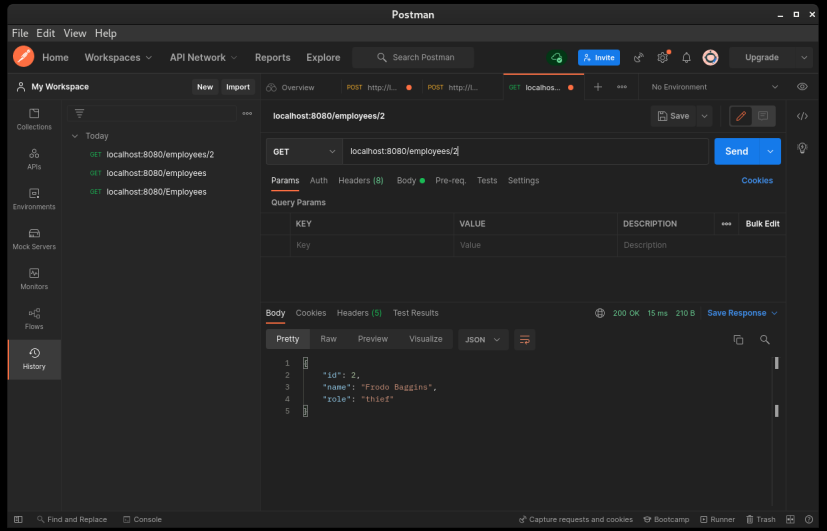
- odczyt wszystkich rekordów



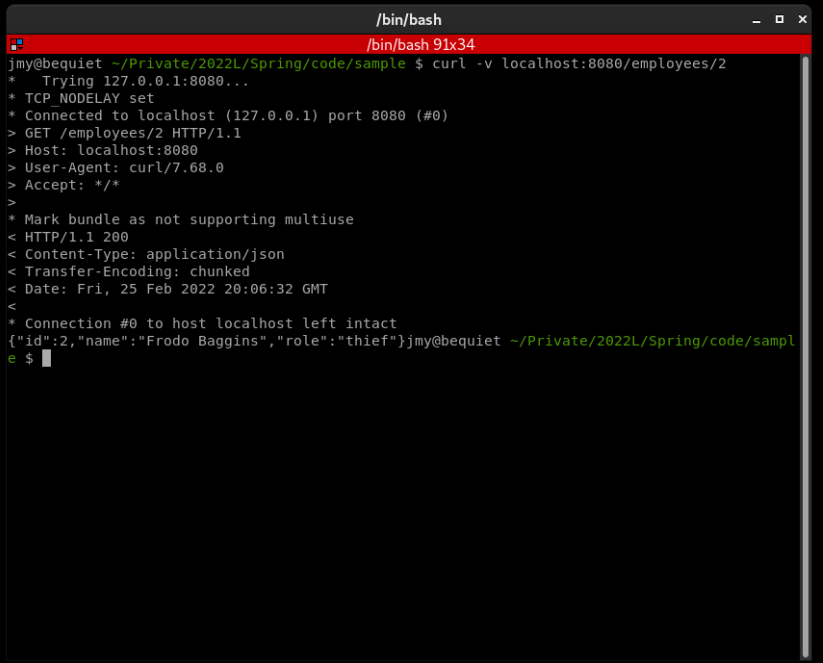
- odczyt wszystkich rekordów



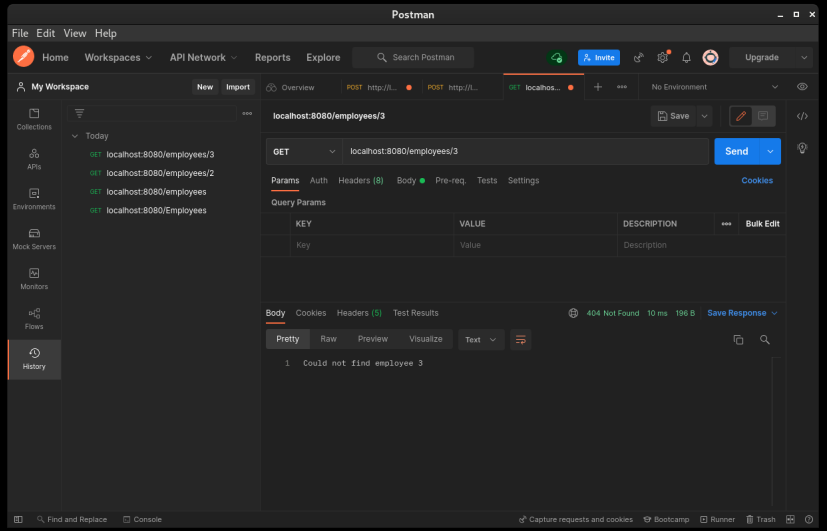
- odczyt wszystkich rekordów
- odczyt jednego rekordu



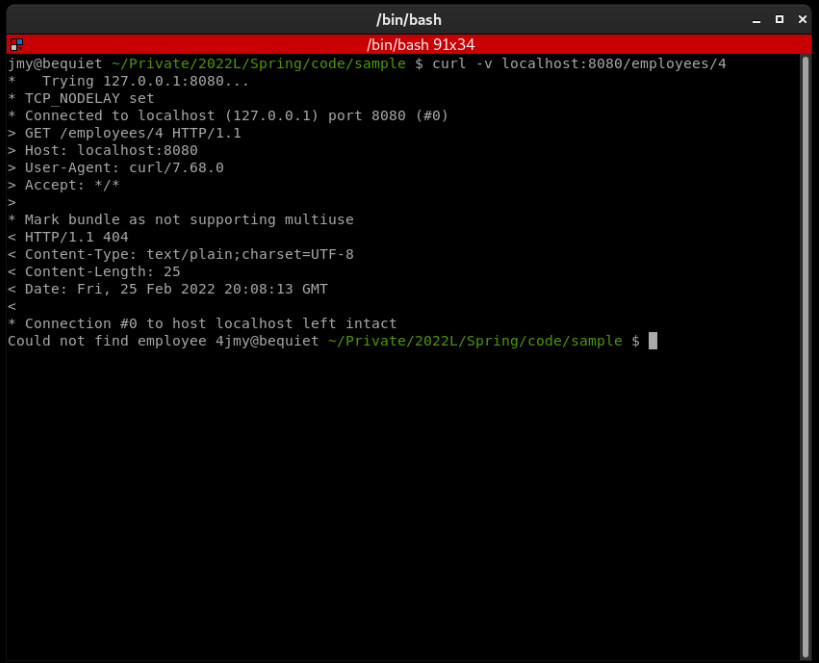
- odczyt wszystkich rekordów
- odczyt jednego rekordu



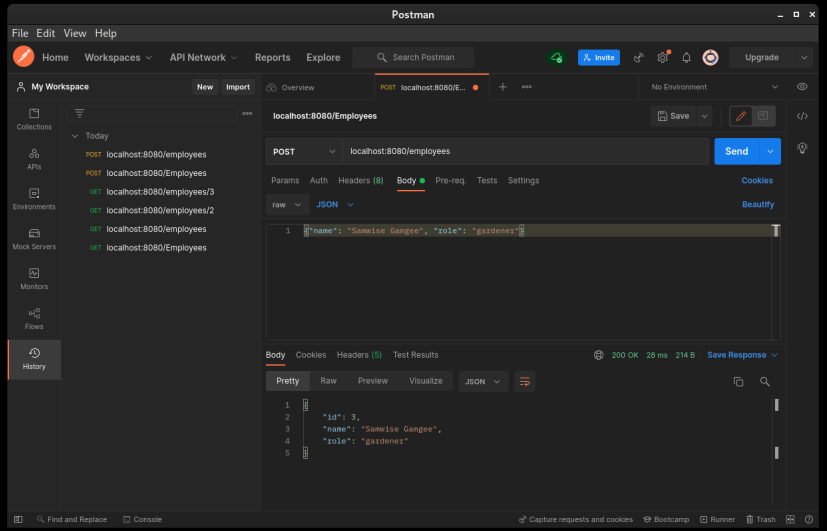
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu



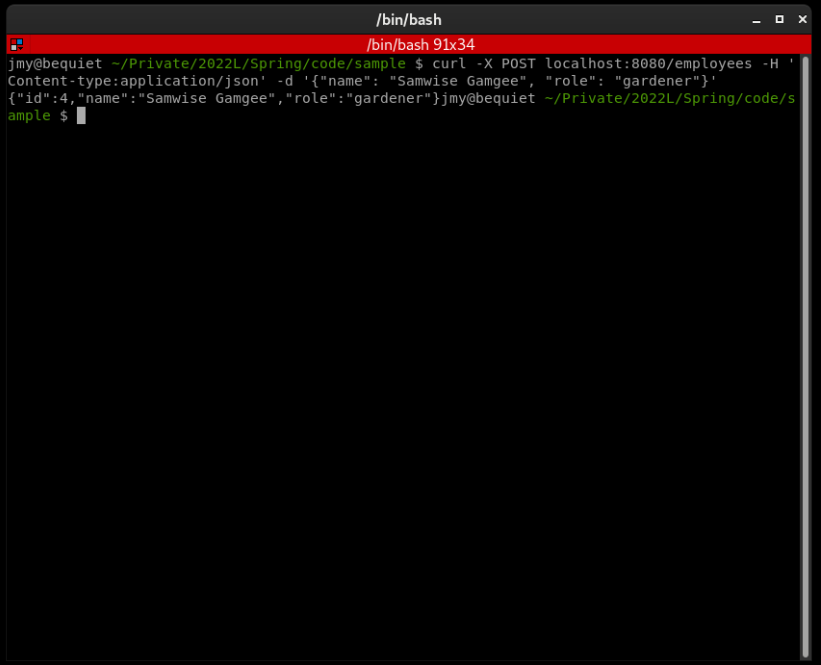
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu



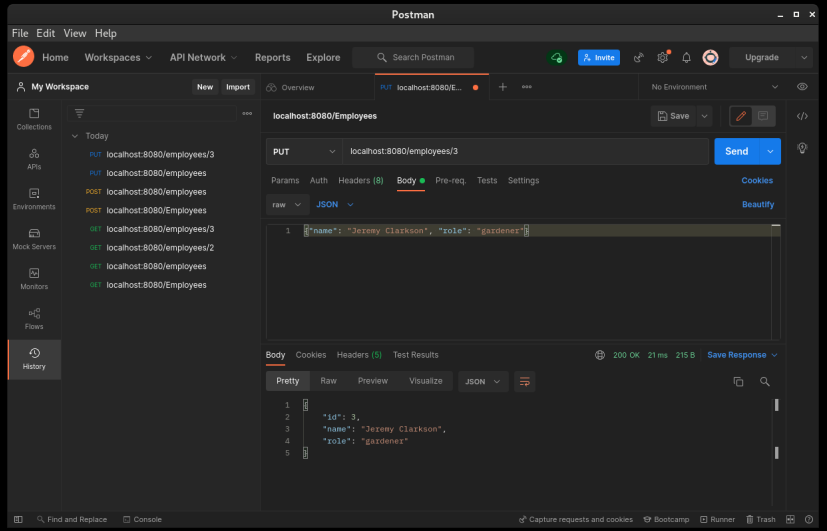
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu



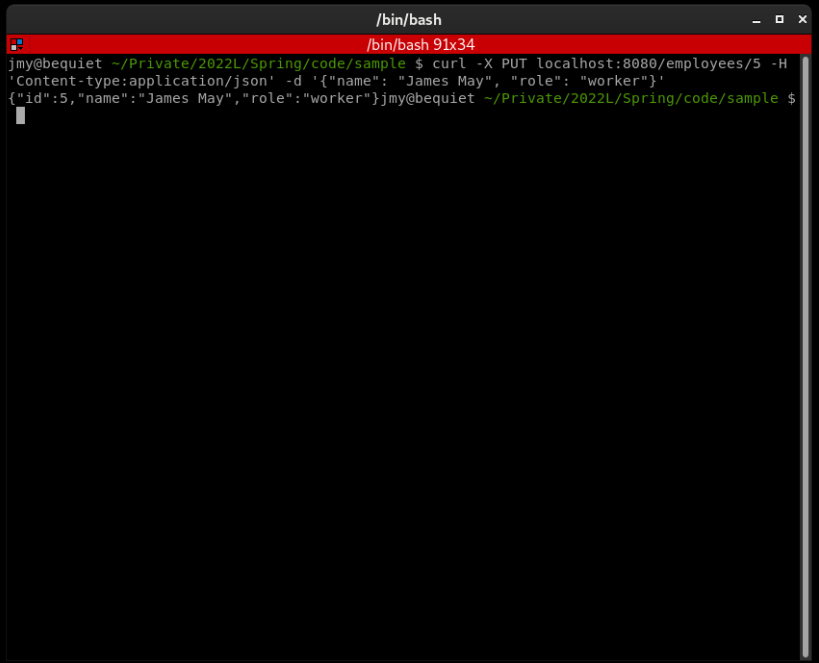
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu



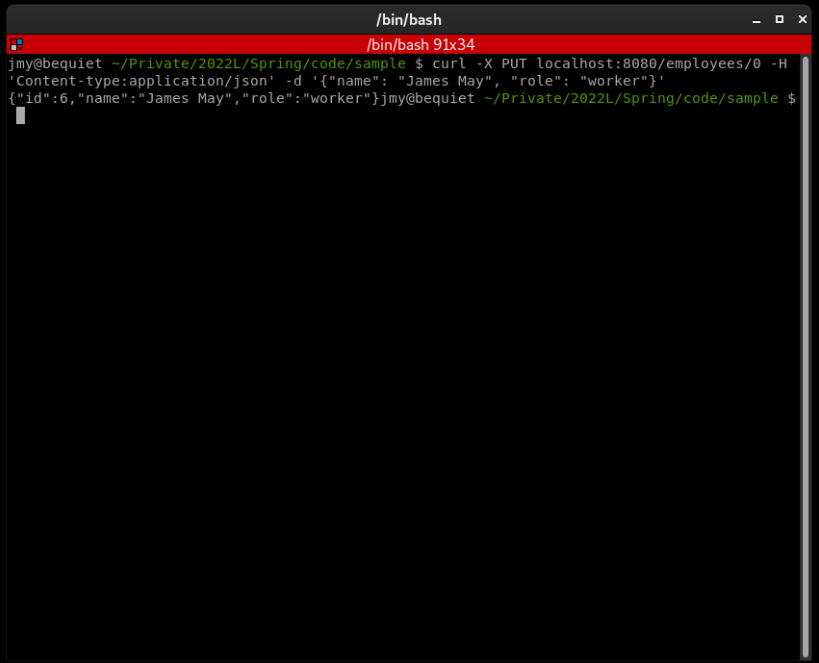
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu



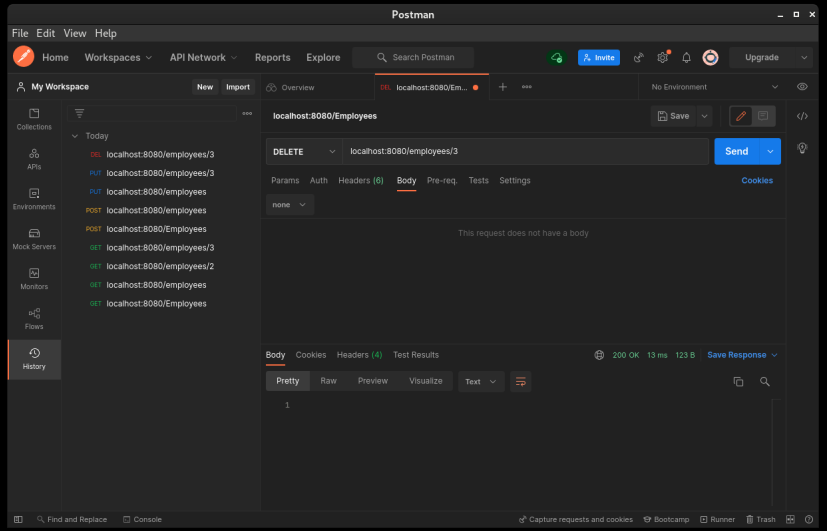
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu



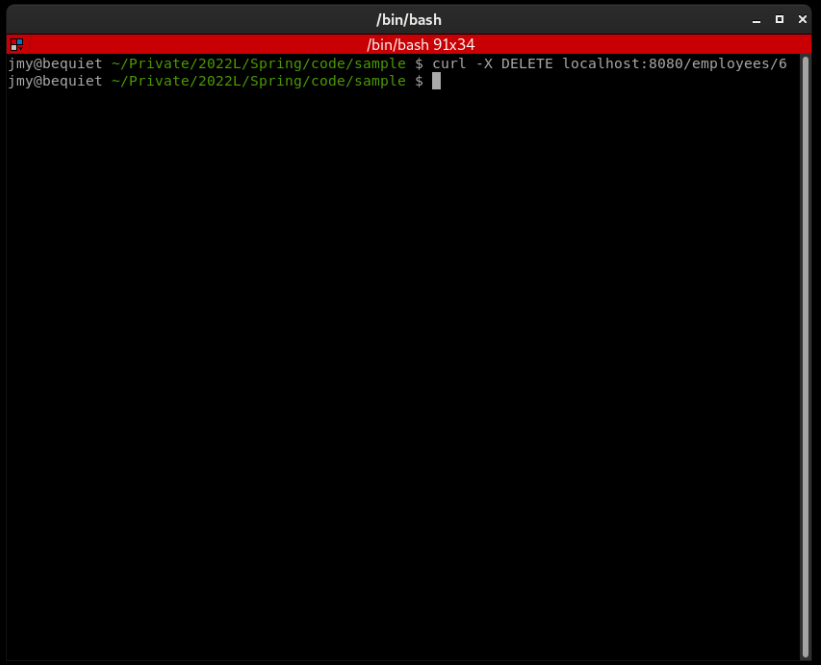
- odczyt wszystkich rekordów
- odczyt jednego rekordu
- odczyt nieistniejącego rekordu
- dodanie rekordu
- modyfikacja rekordu



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



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



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.

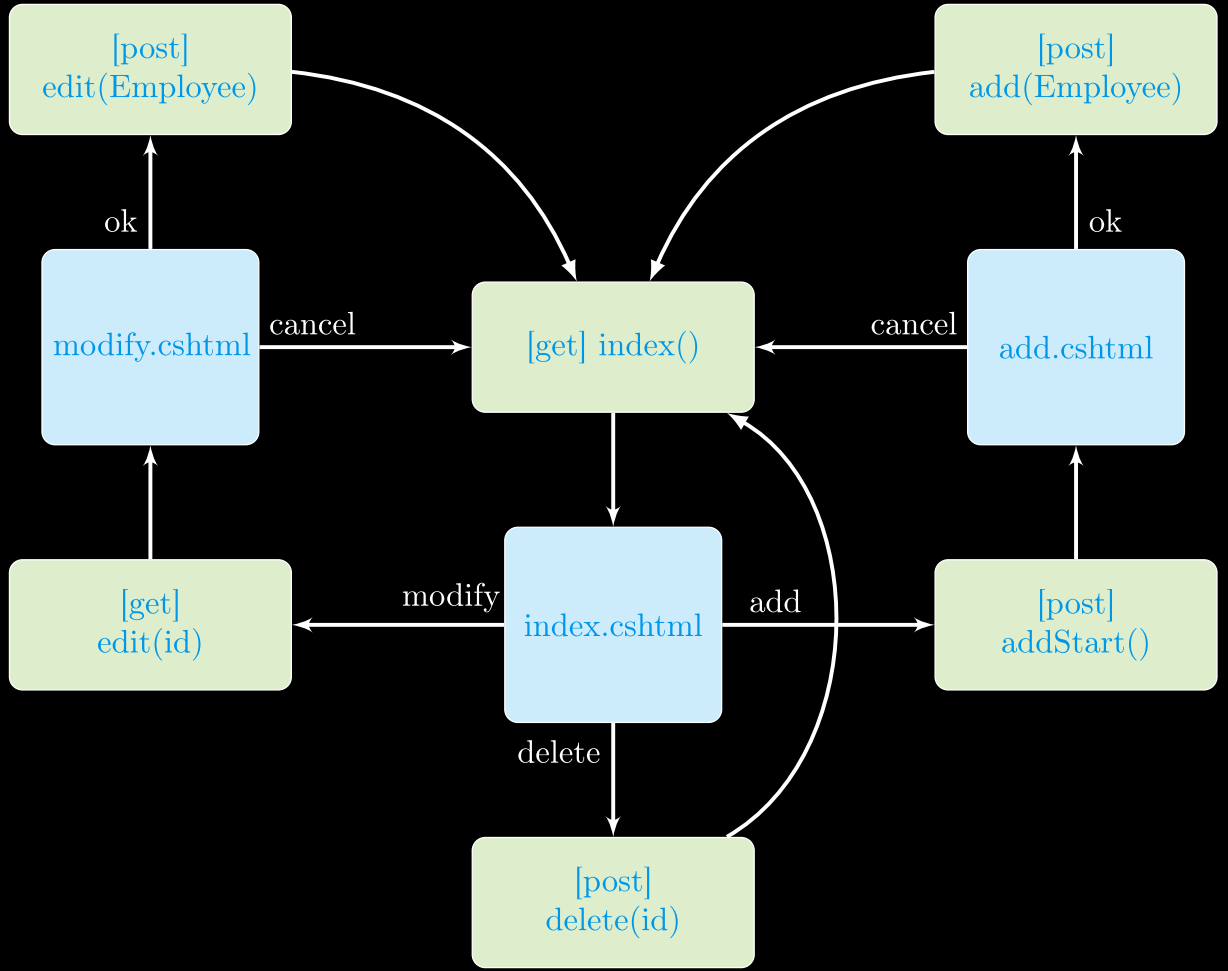
Cloud : Zapewnia obsługę łatwego opracowywania niektórych typowych wzorców systemów rozproszonych, takich jak wykrywanie usług, wyłącznik automatyczny i brama API.

- 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.

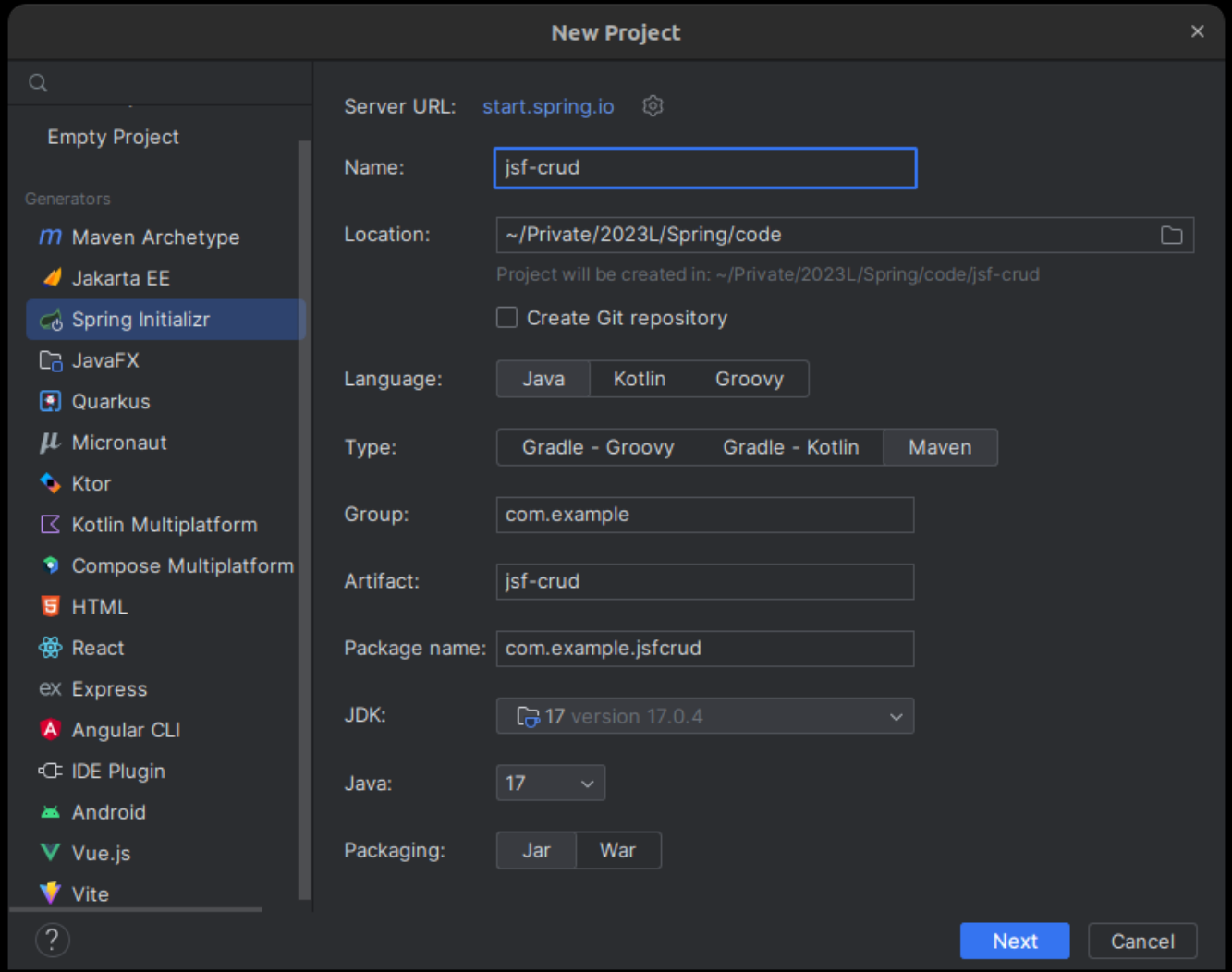
Batch : Zapewnia uproszczoną strukturę do tworzenia aplikacji wsadowych. Posiada intuicyjną obsługę planowania, restartowania, pomijania, zbierania metryk i rejestrowania.

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

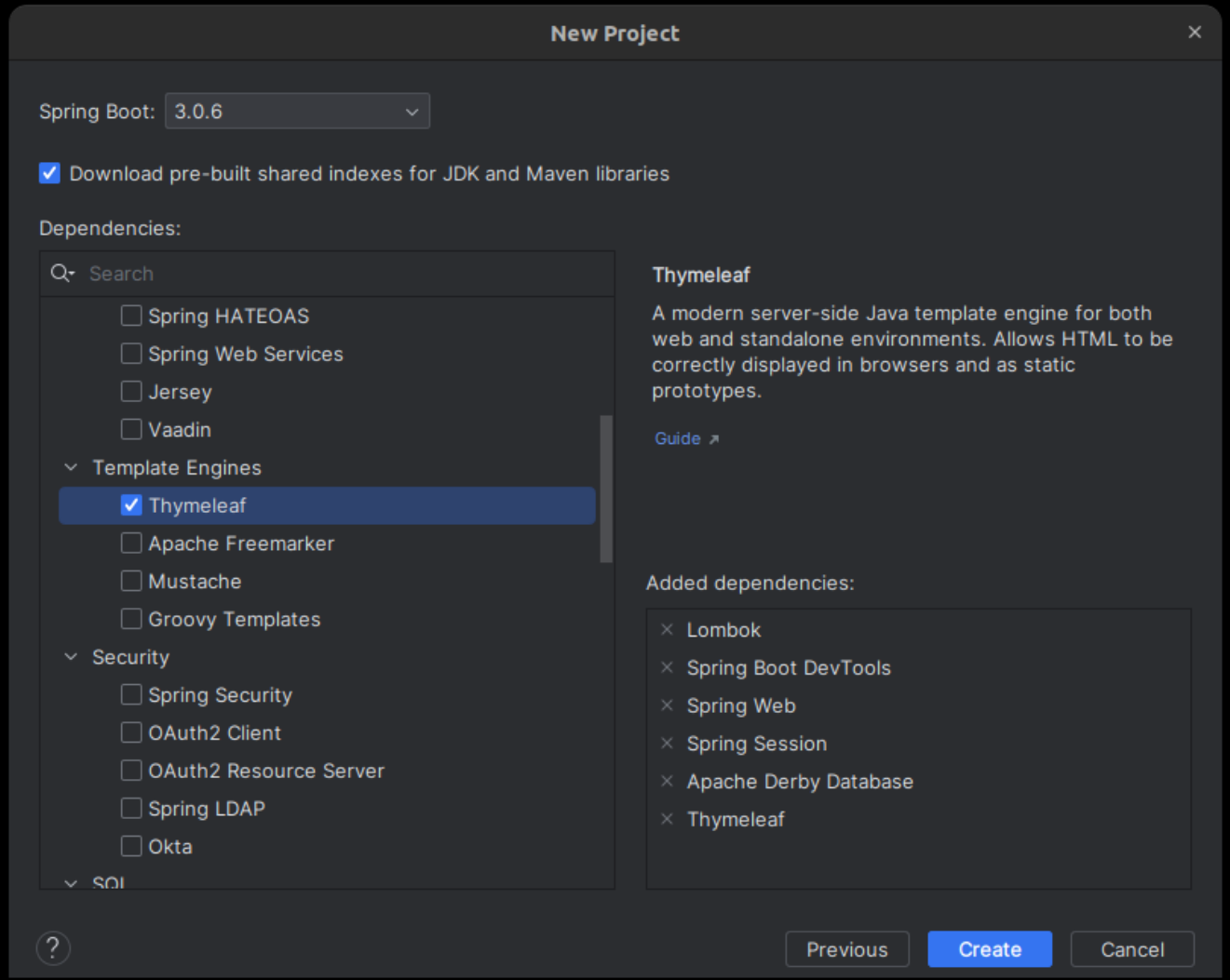

[post]
edit(Employee)
[post]
add(Employee)
ok
ok
modify.cshtml
cancel
[get] index()
cancel
add.cshtml
[get]
edit(id)
modify
index.cshtml
add
[post]
addStart()
delete
[post]
delete(id)

- 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:
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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.0.6</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.example</groupId>
	<artifactId>jsf-crud</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>jsf-crud</name>
	<description>jsf-crud</description>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-thymeleaf</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.session</groupId>
			<artifactId>spring-session-core</artifactId>
		</dependency>

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

```
                        <artifactId>derby</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>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                                <configuration>
                                        <excludes>
                                                <exclude>
                                                        <groupId>org.projectlombok</groupId>
                                                        <artifactId>lombok</artifactId>
                                                </exclude>
                                        </excludes>
                                </configuration>
                        </plugin>
                </plugins>
        </build>

</project>
```

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

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

```java
package com.example.jsfcrud.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Entity
@Data
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long EmployeeId;

  @NotBlank(message = "First Name should not be blank")
  private String firstName;

  @NotBlank(message = "Last Name should not be blank")
  private String lastName;

  private int salary;
}
```

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

```java
package com.example.jsfcrud.repositories;
import com.example.jsfcrud.entities.Employee;
import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface EmployeeRepository extends CrudRepository<Employee, Long> {
  List<Employee> findById(long id);
}

```

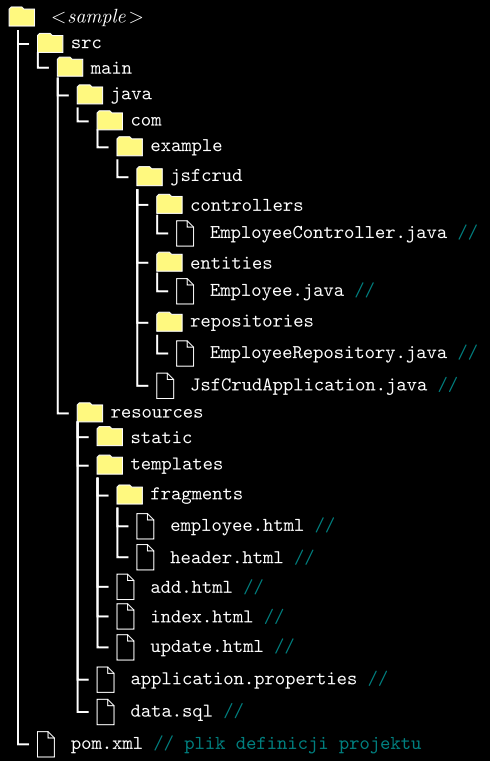
- Tworzymy katalogi `controllers` `entities` i `repositories`
- Tworzymy klasę `Employee`
- Dodajemy interfejs EmployeeRepository
  - interfejs `CrudRepository` z biblioteki `Spring Data JPA` załatwia operacje CRUD na danych
- EmployeeController

```java
package com.example.jsfcrud.repositories;
import com.example.jsfcrud.entities.Employee;
import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface EmployeeRepository extends CrudRepository<Employee, Long> {
  List<Employee> findById(long id);
}

```

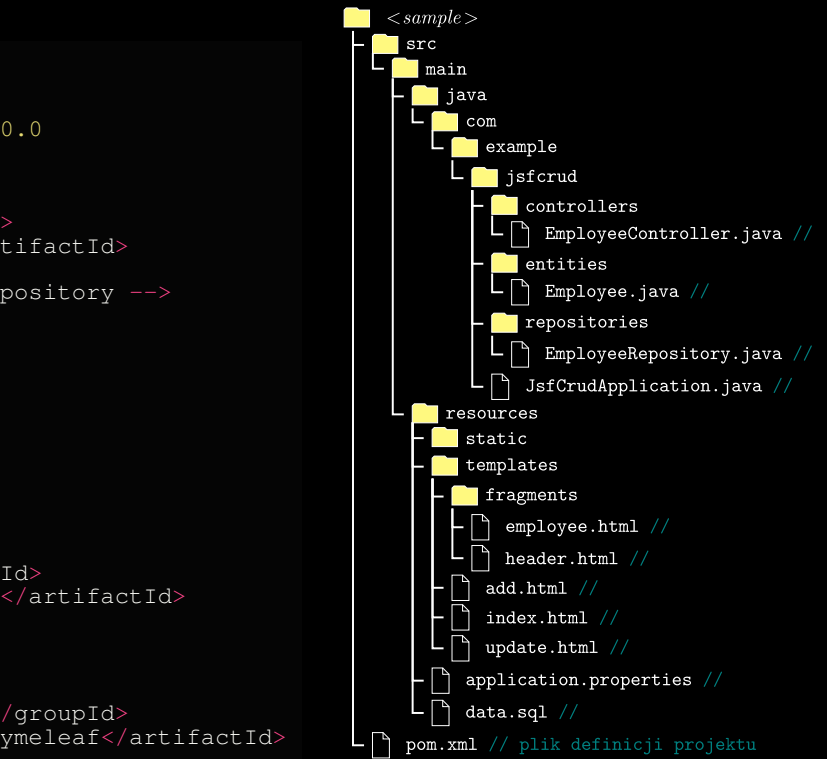
- Tworzymy katalogi `controllers` `entities` i `repositories`
- Tworzymy klasę `Employee`
- Dodajemy interfejs EmployeeRepository
  - interfejs `CrudRepository` z biblioteki `Spring Data JPA` załatwia operacje `CRUD` na danych
- EmployeeController

```java
package com.example.jsfcrud.controllers;
import com.example.jsfcrud.repositories.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class EmployeeController {
  private final EmployeeRepository repository;

  @Autowired
  public EmployeeController(EmployeeRepository repository) {
    this.repository = repository;
  }

  @GetMapping("/index")
  public String showEmployeeList(Model model) {

    model.addAttribute("employees", repository.findAll());
    return "index";
  }
}
```

```java
package com.example.jsfcrud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JsfCrudApplication {
        public static void main(String[] args) {

                SpringApplication.run(JsfCrudApplication.class, args);
        }
}
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         ↪  https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.0.6</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.example</groupId>
        <artifactId>jsf-crud</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>jsf-crud</name>
        <description>jsf-crud</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>jakarta.validation</groupId>
                        <artifactId>jakarta.validation-api</artifactId>
                        <version>3.0.2</version>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-thymeleaf</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-validation</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.session</groupId>
                        <artifactId>spring-session-core</artifactId>
```



```java
package com.example.jsfcrud.controllers;
import com.example.jsfcrud.repositories.EmployeeRepository;
import com.example.jsfcrud.entities.Employee;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class EmployeeController {
  private final EmployeeRepository repository;

  @Autowired
  public EmployeeController(EmployeeRepository repository) {
    this.repository = repository;
  }

  @GetMapping("/index")
  public String showEmployeeList(Model model) {

    model.addAttribute("employees", repository.findAll());
    return "index";
  }

  @GetMapping("/add")
  public String showNewEmployee(Employee employee) {
    return "add";
  }

  @PostMapping("/add")
  public String saveNewEmployee(@Valid Employee e, BindingResult result, Model
  ↪  model) {
    if (result.hasErrors()) {
      return "add";
    }
    repository.save(e);
    return "redirect:/index";
  }

  @GetMapping("/edit/{id}")
  public String showExistingEmployee(@PathVariable("id") long id, Model model)
```

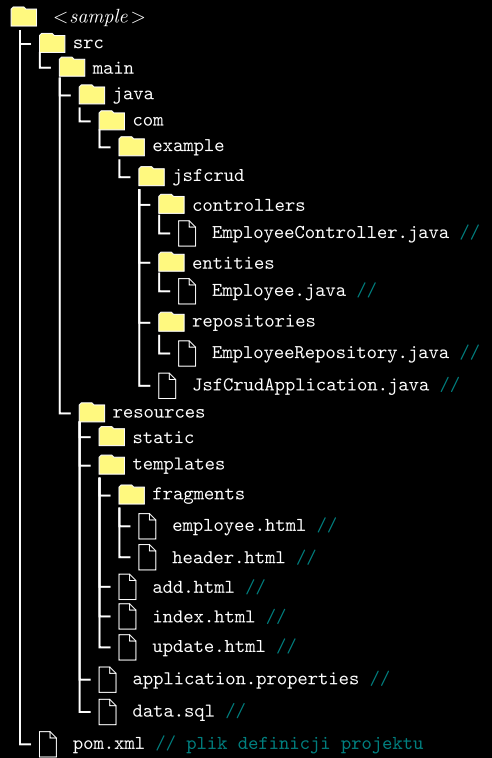
```
<sample>
├── src
│   └── main
│       ├── java
│       │   └── com
│       │       └── example
│       │           └── jsfcrud
│       │               ├── controllers
│       │               │   └── EmployeeController.java //
│       │               ├── entities
│       │               │   └── Employee.java //
│       │               ├── repositories
│       │               │   └── EmployeeRepository.java //
│       │               └── JsfCrudApplication.java //
│       └── resources
│           ├── static
│           ├── templates
│           │   ├── fragments
│           │   │   ├── employee.html //
│           │   │   └── header.html //
│           │   ├── add.html //
│           │   ├── index.html //
│           │   └── update.html //
│           ├── application.properties //
│           └── data.sql //
└── pom.xml // plik definicji projektu
```

```java
package com.example.jsfcrud.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Entity
//@Data
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long EmployeeId;

  @NotBlank(message = "First Name should not be blank")
  private String firstName;

  @NotBlank(message = "Last Name should not be blank")
  private String lastName;

  private int salary;


  public long getEmployeeId() {
    return EmployeeId;
  }

  public void setEmployeeId(long employeeId) {
    EmployeeId = employeeId;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }
```

```
<sample>
├── src
│   └── main
│       ├── java
│       │   └── com
│       │       └── example
│       │           └── jsfcrud
│       │               ├── controllers
│       │               │   └── EmployeeController.java //
│       │               ├── entities
│       │               │   └── Employee.java //
│       │               ├── repositories
│       │               │   └── EmployeeRepository.java //
│       │               └── JsfCrudApplication.java //
│       └── resources
│           ├── static
│           ├── templates
│           │   ├── fragments
│           │   │   ├── employee.html //
│           │   │   └── header.html //
│           │   ├── add.html //
│           │   ├── index.html //
│           │   └── update.html //
│           ├── application.properties //
│           └── data.sql //
└── pom.xml // plik definicji projektu
```

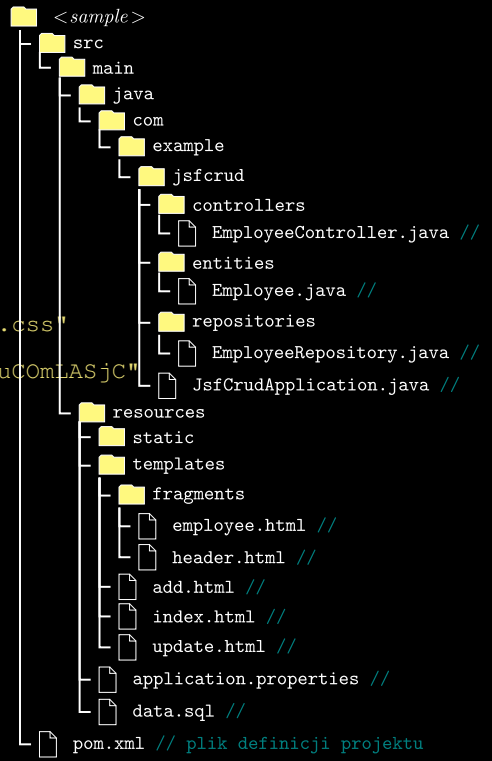
crud/src/main/resources/templates/fragments/employee.html

```
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"
           ↪  id="firstName" placeholder="Name">
12         <span th:if="${#fields.hasErrors('firstName')}"
           ↪  th:errors="*{firstName}" class="text-danger"></span>
13     </div>
14     <div class="row mb-3">
15         <label for="lastName" class="col-form-label">lastName</label>
16         <input type="text" th:field="*{lastName}" class="form-control"
           ↪  id="lastName" placeholder="Name">
17         <span th:if="${#fields.hasErrors('lastName')}" th:errors="*{lastName}"
           ↪  class="text-danger"></span>
18     </div>
19     <div class="row mb-3">
20         <label for="salary" class="col-form-label">Salary</label>
21         <input type="text" th:field="*{salary}" class="form-control"
           ↪  id="salary" placeholder="Email">
22         <span th:if="${#fields.hasErrors('salary')}" th:errors="*{salary}"
           ↪  class="text-danger"></span>
23     </div>
24 </span>
25 </body>
26 </html>
```



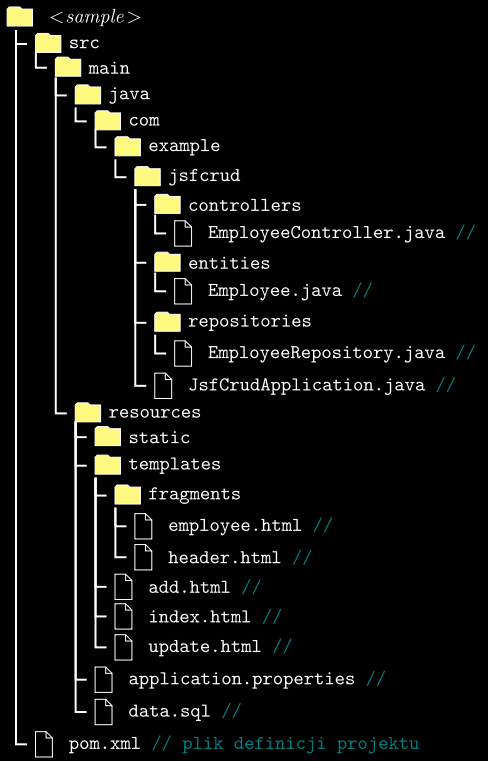
crud/src/main/resources/templates/fragments/header.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
<head th:fragment="header">
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link
     ↪  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
    rel="stylesheet"
     ↪  integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
    crossorigin="anonymous">
    <title th:text="${title}"></title>
</head>
<body>
</body>
</html>
```



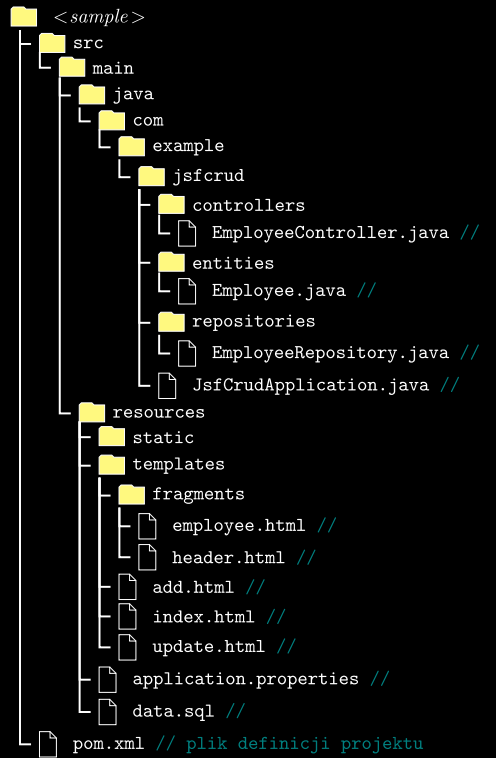
crud/src/main/resources/templates/add.html

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

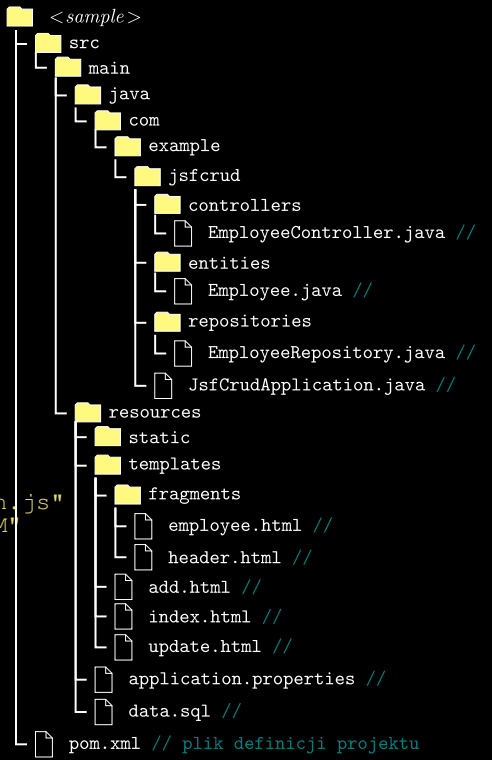


crud/src/main/resources/templates/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
<head th:insert="~{fragments/header :: header}" th:remove="tag"></head>
<body>
<div th:switch="${employees}" class="container my-5">
    <h2 th:case="null">No employees yet!</h2>
    <div th:case="*">
        <h2 class="my-5">Employees</h2>
        <table class="table table-striped table-responsive-md">
            <thead>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Edit</th>
                <th>Delete</th>
            </tr>
            </thead>
            <tbody>
            <tr th:each="e : ${employees}">
                <td th:text="${e.firstName}"></td>
                <td th:text="${e.lastName}"></td>
                <td><a th:href="@{/edit/{id}(id=${e.employeeId})}" class="btn
                ↪  btn-primary">Edit</a></td>
                <td><a th:href="@{/delete/{id}(id=${e.employeeId})}"
                ↪  class="btn btn-primary">Delete</a></td>
            </tr>
            </tbody>
        </table>
    </div>
    <p class="my-5"><a href="/add" class="btn btn-primary">Add
    ↪  Employee</a></p>
</div>
</body>
</html>
```



```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="utf-8">
<head th:insert="~{fragments/header :: header}" th:remove="tag"></head>
<body>
<div class="container">
    <h2 class="mb-5"><span th:text="${title}" th:remove="tag">Page Title</span></h2>

    <form action="#" th:action="@{/update/{id}(id=${employee.employeeId})}" th:object="${employee}" method="post">
        <span th:insert="~{fragments/employee :: employee}" th:remove="tag"></span>
        <button type="submit" class="btn btn-primary">Update</button>
        <a href="/index" class="btn btn-danger">Cancel</a>
    </form>

</div>
<!-- Option 1: Bootstrap Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>

</body>
</html>
```



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

```xml
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <version>9.0.44</version>
      <scope>provided</scope>
</dependency>
```

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

```xml
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <version>9.0.44</version>
</dependency>
```

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

```xml
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</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 dostarczanymi przez środowisko uruchomieniowe Tomcat, musimy ustawić dwie zależności o podanym zakresie:

```xml
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <version>9.0.44</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <version>2.4.4</version>
    <scope>provided</scope>
</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 dostarczanymi przez środowisko uruchomieniowe Tomcat, musimy ustawić dwie zależności o podanym zakresie:
- Musimy poinformować Spring, gdzie znajduje się pliki JSP

w pliku application.properties

```
spring.mvc.view.prefix: /WEB-INF/jsp/
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 dostarczanymi 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)

```java
@SpringBootApplication(scanBasePackages = "atj")
public class SpringBootJspApplication extends
    ↪ SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder
        ↪ configure(SpringApplicationBuilder builder) {
        return
            ↪ builder.sources(SpringBootJspApplication.class);
    }

    public static void main(String[] args) {

        ↪ SpringApplication.run(SpringBootJspApplication.class)
    }
}
```

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

- Załóżmy, 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      // gettery, settery, konstruktory i toString
6  }
7
8  public interface PersonService {
9      Collection<Person> getPersons();
10     Person addPerson(Person person);
11 }
```

- Załóżmy, 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/.

```java
@Controller
@RequestMapping("/person")
public class PersonController {

    private final PersonService service;

    public PersonController(PersonService service) {
        this.service = service;
    }

    @GetMapping("/viewPersons")
    public String viewPersons(Model model) {
        model.addAttribute("persons",
        ↪  service.getPersons());
        return "view-persons";
    }
}
```

- Załóżmy, 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/.

```
<%@ page contentType="text/html;charset=UTF-8"
↪  language="java" %>
<%@taglib prefix="c"
↪  uri="http://java.sun.com/jsp/jstl/core"%>
<html>
    <head>
        <title>View People</title>
        <link href="<c:url value="/css/common.css"/>"
        ↪  rel="stylesheet" type="text/css">
    </head>
    <body>
        <table>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>First Name</th>
                    <th>LastName</th>
                    <th>LastName</th>
                </tr>
            </thead>
            <tbody>
                <c:forEach items="${personss}"
                ↪  var="person">
                    <tr>
                        <td>${person.id}</td>
                        <td>${person.firstName}</td>
                        <td>${person.lastName}</td>
                    </tr>
                </c:forEach>
            </tbody>
        </table>
    </body>
</html>
```

- Załóżmy, 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
       ↪  addPerson(@ModelAttribute("person")
13       Person person, RedirectAttributes
         ↪  redirectAttributes) {
14       final RedirectView redirectView = new
         ↪  RedirectView("/person/addPerson", true);
15       Person savedPerson =
         ↪  personService.addPerson(person);
16
         ↪  redirectAttributes.addFlashAttribute("savedPerson",
         ↪  savedPerson);
17
         ↪  redirectAttributes.addFlashAttribute("addPersonSuccess"
         ↪  true);
18       return redirectView;
19     }
20 }
```

- Załóżmy, 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

add-person.jsp

```
<%@ taglib prefix="c"
↪  uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form"
↪  uri="http://www.springframework.org/tags/form" %>
<%@ page contentType="text/html;charset=UTF-8"
↪  language="java" %>
<html>
    <head>
        <title>Add Person</title>
    </head>
    <body>
        <c:if test="${addPersonSuccess}">
            <div>Successfully added Person with ID:
            ↪  ${savedPerson.id}</div>
        </c:if>

        <c:url var="add_person_url"
        ↪  value="/person/addPerson"/>
        <form:form action="${add_person_url}"
        ↪  method="post" modelAttribute="person">
            <form:label path="Id">ID: </form:label>
            <form:input type="text" path="id"/>
            <form:label path="name">First Name:
            ↪  </form:label>
            <form:input type="text" path="firstName"/>
            <form:label path="name">Last Name:
            ↪  </form:label>
            <form:input type="text" path="lastName"/>
            <input type="submit" value="submit"/>
        </form:form>
    </body>
</html>
```

- Załóżmy, 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 }
```

- Załóżmy, 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

```java
@Service
public class personServiceImpl implements personService
 ↪ {

    private final personRepository personRepository;

    // constructors, other override methods

    @Override
    public person addperson(person person) {
        final Optional<personData> existingperson =
         ↪ personRepository.findById(person.getId());
        if (existingperson.isPresent()) {
            throw new DuplicatepersonException(person);
        }

        final personData savedperson =
         ↪ personRepository.add(convertperson(person));
        return convertpersonData(savedperson);
    }

    // conversion logic
}
```

- Załóżmy, 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 =
       ↪  DuplicatepersonException.class)
5      public ModelAndView
       ↪  duplicatepersonException(DuplicatepersonException
       ↪  e) {
6          final ModelAndView modelAndView = new
           ↪  ModelAndView();
7          modelAndView.addObject("ref",
           ↪  e.getperson().getId());
8          modelAndView.addObject("object",
           ↪  e.getperson());
9          modelAndView.addObject("message", "Cannot add
           ↪  an already existing person");
10         modelAndView.setViewName("error-person");
11         return modelAndView;
12     }
13 }
```

- Załóżmy, 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

error.person.jsp

```
1
```

- The Domain Layer

```java
package sbthyme;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;

@Entity
@Data
@AllArgsConstructor
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    //@NotBlank(message = "Name is mandatory")
    private String name;

    //@NotBlank(message = "Email is mandatory")
    private String email;
}
```

- 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
  ↪  CrudRepository<Employee, Long> {
8 }
```

- The Domain Layer
- @0+:Load Database

```
package sbthyme;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository) {
    return args -> {
      log.info("Preloading " + repository.save(
        new Employee(1, "Jeremy", "Clarkson")));
      log.info("Preloading " + repository.save(
        new Employee(3, "James", "May")));
      log.info("Preloading " + repository.save(
        new Employee(7, "Richard", "Hammond")));
    };
  }
}
```

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

```
package sbthyme;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository) {
    return args -> {
      log.info("Preloading " + repository.save(
        new Employee(1, "Jeremy", "Clarkson")));
      log.info("Preloading " + repository.save(
        new Employee(3, "James", "May")));
      log.info("Preloading " + repository.save(
        new Employee(7, "Richard", "Hammond")));
    };
  }
}
```

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

```
package sbthyme;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class LoadDatabase {
  private static final Logger log
                = LoggerFactory.getLogger(LoadDatabase.class);
  @Bean
  CommandLineRunner initDatabase(EmployeeRepository repository) {
    return args -> {
      log.info("Preloading " + repository.save(
        new Employee(1, "Jeremy", "Clarkson")));
      log.info("Preloading " + repository.save(
        new Employee(3, "James", "May")));
      log.info("Preloading " + repository.save(
        new Employee(7, "Richard", "Hammond")));
    };
  }
}
```

- The Domain Layer
- @0+:Controller Layer

```
package sbthyme;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class EmployeeController {
    private final EmployeeRepository repository;
    EmployeeController(EmployeeRepository repository) {
      this.repository = repository;
    }

    @GetMapping("/add")
    public String showAddEmployeeForm(Employee employee) {
        return "add-employee";
    }

    @PostMapping("/add")
    public String addUser(/*@Valid*/ Employee employee,
                          BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "add-user";
        }
        repository.save(employee);
        return "redirect:/index";
    }
}
```

- The Domain Layer

```java
@GetMapping("/add")
public String showAddEmployeeForm(Employee employee) {
    return "add-employee";
}

@PostMapping("/add")
public String addUser(/*@Valid*/ Employee employee,
                      BindingResult result, Model model) {
    if (result.hasErrors()) {
        return "add-user";
    }
    repository.save(employee);
    return "redirect:/index";
}
@GetMapping("/index")
public String showEmployeeList(Model model) {
    model.addAttribute("employees", repository.findAll());
    return "index";
}
```