

Spring Boot

v1.0.0

Spring Boot

Spring Boot

Spring Boot upraszcza tworzenie aplikacji z wykorzystaniem Springa i dostępnych w jego ramach projektów.

Wymaga bardzo niewielkiej lub zerowej konfiguracji, czyli eliminuje powtarzalne i nudne czynności.

Pozwala także w prosty sposób uruchomić utworzone aplikacje, używając polecenia **java -jar** i zarządzać zależnościami w projekcie.

Strona projektu: <https://projects.spring.io/spring-boot/>

Spring Boot

Tworząc aplikacje z wykorzystaniem **Springa** oraz jego komponentów, zauważamy, że w większości przypadków, z aplikacji na aplikację kopiujemy odpowiednie pliki konfiguracyjne, a kod w nich zawarty większości pozostaje taki sam.

Twórcy **Springa** również to zauważyli i zgodnie z oczekiwaniami developerów powstał projekt **Spring Boot**, którego założeniem jest ograniczenie konfiguracji do minimum.

Spring Boot

Spring Boot wykorzystuje mechanizm skanowania bibliotek projektu i na podstawie wykrycia określonych klas konfiguruje domyślne komponenty.

Oczywiście jeżeli dodamy własną konfigurację nadpisze ona tą określoną przez **Spring Boot**.

Jest to przykład podejścia **konwencja** nad **konfigurację**.

https://pl.wikipedia.org/wiki/Convention_Over_Configuration

Wymagania

Domyślnie **Spring Boot** wymaga Java 7 oraz Spring w wersji 4.3.10 lub wyższej.

Możliwe jest uruchomienie biblioteki w Javie 6 po przeprowadzeniu dodatkowej konfiguracji.

Polecana przez twórców Spring Boot wersją jest oczywiście Java 8.

Spring Boot zapewnia wsparcie dla Apache Maven w wersji 3.2 lub wyższej i Gradle 2.9 lub wyższej oraz wersji 3.

Instalacja i uruchomienie

Spring Boot może być używany jak zwykła biblioteka Javy.

Wystarczy dodać odpowiednie pliki **spring-boot-*.jar** do projektu.

Spring Boot nie wymaga żadnych dedykowanych narzędzi programistycznych.

Może być używany w dowolnym **IDE** czy też w edytorze tekstu, a programy napisane przy użyciu **Spring Boot** niczym się nie wyróżniają spośród innych programów napisanych w Javie.

Choć można używać **Spring Boot**, kopiując wprost odpowiednie biblioteki jar, wygodniejszym i jednocześnie zalecanym sposobem jest użycie **Apache Maven** lub **Gradle**.

Tworzenie projektu - Maven

Powszechną metodą tworzenia projektu jest bezpośrednio użycie **Mavena**.

Typowy plik **pom.xml** projektu dziedziczy po starterze **spring-boot-starter-parent**, dzięki czemu wpisując pozostałe zależności projektu, można pominąć sekcję **<version>**.

Wersje dodawanych bibliotek zależnych będą dobierane przez projekt **spring-boot-starter-parent** w taki sposób, aby były ze sobą kompatybilne.

Przykłady przedstawione w niniejszym dziale są oparte o wersję **1.5.9**.

Tworzenie projektu - Maven

Tworzymy projekt **Maven** za pomocą **IDE**.

Plik **pom.xml** uzupełniamy o następujące wpisy:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.9.RELEASE</version>
</parent>
```

Dziedziczenie po **spring-boot-starter-parent** powoduje, że projekt jest konfigurowany przez pewne parametry domyślne, m.in.:

- domyślna wersja kompilatora Java 1.6
- kodowanie znaków UTF-8
- mechanizm zarządzania zależnościami pozwalający na pomijanie sekcji `<version>` w dodawanych zależnościach.

Tworzenie projektu - Maven

Domyślną wersję kompilatora można zmienić poprzez dodanie do pom.xml właściwości **<java.version>**

```
<properties>  
    <java.version>1.8</java.version>  
</properties>
```

Tworzenie projektu - Maven

Dodajemy również opcjonalny plugin **spring-boot-maven-plugin**, dzięki któremu będzie można wygenerować wykonywalny plik **.jar** z gotowym programem.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Tworzenie projektu - Maven - zależności

Startery w rozumieniu **Spring Boot** są to wygodne zależności, które można dołączyć do projektu.

Pojedynczy starter może zawierać zestaw kilku zależności, dzięki czemu programista może szybko dodać pewną funkcjonalność.

Dodajemy do projektu zależność odpowiedzialną za obsługę **Spring MVC**:

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

Startery

Przykłady popularnych starterów:

- **spring-boot-starter-data-jpa** – obsługa JPA
- **spring-boot-starter-freemarker** - system szablonów
- **spring-boot-starter-thymeleaf** – system szablonów
- **spring-boot-starter-web** – Spring MVC
- **spring-boot-starter-validation** – wsparcie dla walidacji
- **spring-boot-starter-test** – wsparcie dla testowania
- **spring-boot-devtools** – monitoruje aplikację i po zmianach automatycznie ją przeładowuje

Używanie Spring Boot

Spring Boot nie wymaga specjalnego układu kodu, zalecane jest jednak trzymanie się pewnych ogólnie przyjętych praktyk.

Po pierwsze klasy, które nie zawierają deklaracji pakietu (package), są traktowane, jakby znajdowały się w „**pakiecie domyślnym**”.

Powinniśmy unikać takich sytuacji. Może to powodować problemy podczas stosowania adnotacji Spring Boot takich jak **@ComponentScan** czy **@SpringBootApplication**.

Przyjęło się, żeby nazwy pakietów zaczynały się od odwróconej nazwy domeny, a następnie nazwa aplikacji, w naszym wypadku **pl.coderslab.projectname**.

Używanie Spring Boot

Zaleca się, aby główny plik aplikacji powinien umieszczać w pakiecie głównym projektu, ponad pozostałymi klasami.

Główna klasa ma często adnotację **@EnableAutoconfiguration**, która niejawnie ustawia bazową ścieżkę wyszukiwania dla poszczególnych elementów aplikacji (np. klas encji **@Entity**, gdy aplikacja korzysta z JPA).

Mechanizm autokonfiguracji próbuje automatycznie skonfigurować aplikację na podstawie bibliotek dołączonych do projektu.

Struktura projektu

Struktura projektu może wyglądać następująco:

```
pl
+- coderslab
  +- projectname
    +- Application.java
    |
    +- domain
    |   +- User.java
    |   +- UserAddress.java
    |
    +- service
    |   +- UserService.java
    |
    +- web
    |   +- UserController.java
```


Klasa Startowa

Uzupełniamy projekt o klasę startową dla naszej aplikacji:

```
package pl.coderslab;

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

@SpringBootApplication
public class Application {
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }
}
```

Adnotacja @SpringBootApplication

Adnotacja **@SpringBootApplication** grupuje kilka innych poznanych przez nas adnotacji m.in. :

- **@ComponentScan**,
- **@Configuration**,
- **@EnableAutoConfiguration** - informuje **Spring Boot** by dodawał ziarna konfiguracyjne na podstawie elementów naszej aplikacji.

Przykładowo wykrywa, że mamy zależności do **spring-webmvc** i na tej podstawie konfiguruje np. **DispatcherServlet** - oszczędzając nam pracy.

Testowa akcja

Dodajemy przykładowy kontroler oraz akcje:

```
package pl.coderslab.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HomeController {

    @RequestMapping("/")
    @ResponseBody
    public String home() {
        return "hello world !!!";
    }
}
```

Instalacja i uruchomienie

Tak przygotowaną aplikację możemy już uruchomić za pomocą **Mavena** przy pomocy konsolowej komendy:

```
mvn spring-boot:run
```

Możemy również przy pomocy **IDE** wybierając z menu kontekstowego **Run As** następnie **Spring Boot App** lub **Java Application**.

Wywołując polecenie **Mavena** :

mvn package - wygenerujemy w katalogu **target** naszej aplikacji plik wykonywalny **jar**.

Plik ten możemy uruchomić za pomocą polecenia: **java -jar nazwa_pliku.jar**

np.: **java -jar boot-maven-0.0.1-SNAPSHOT.jar**

Wynik naszej pracy możemy zobaczyć w przeglądarce po wejściu na adres:

<http://localhost:8080/>

Tworzenie projektu - Spring Initializr

Tworząc projekt z wykorzystaniem **Spring Boot** możemy również użyć serwisu internetowego **Spring Initializr**, który umożliwia określenie elementów, z jakich ma się składać aplikacja.

Na stronie projektu należy wpisać metadane tworzonego projektu jak np nazwa, grupa, artefakt oraz wybrać składniki (zależności i startery), które mają być dołączone do projektu, a wygenerowany projekt można pobrać w postaci archiwum zip.

Tak pobrany plik po rozpakowaniu, możemy następnie otworzyć przy pomocy dowolnego IDE.

Spring Initializr dostępny jest pod adresem: <http://start.spring.io/>

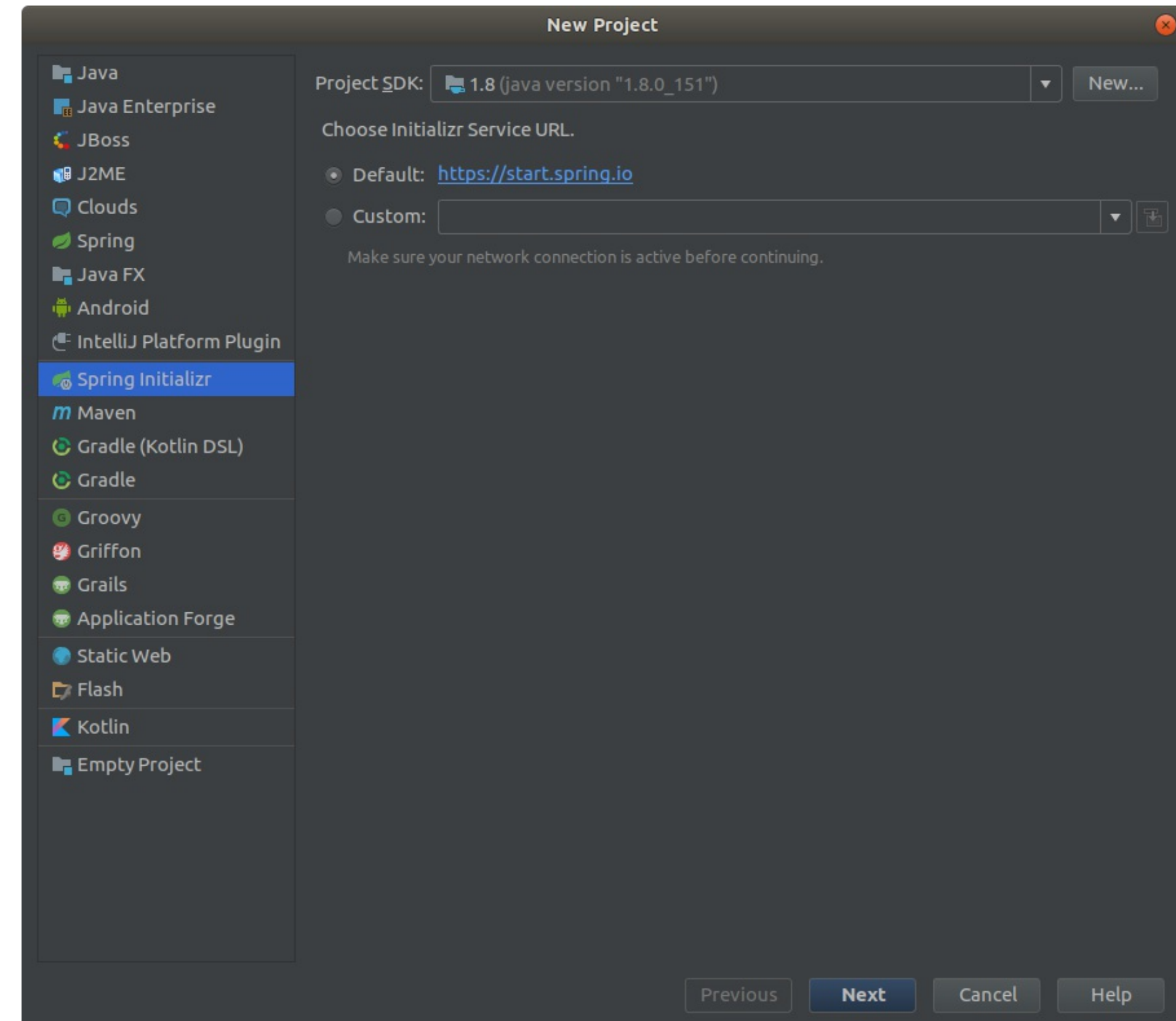
Spring Boot

Alternatywnym sposobem na utworzenie takiego projektu jest wykorzystanie wsparcia, jakie udostępnia nasze IDE – na przykładzie **IntelliJ**.

Z menu górnego wybieramy:
File -> New -> Project...

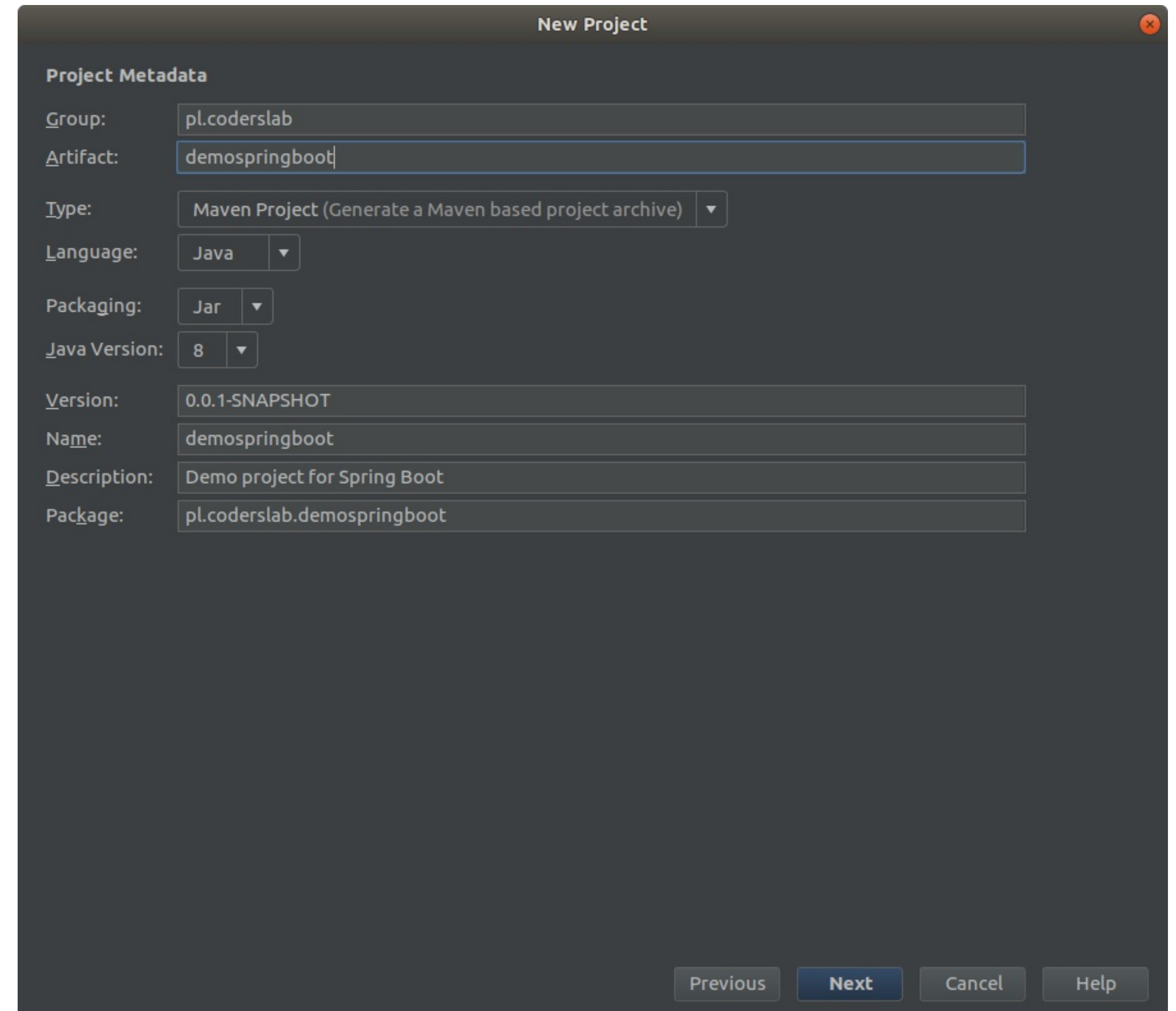
a następnie zaznaczamy opcję:
Spring Initializr.

Klikamy **Next**.



Spring Boot

Wypełniamy podstawowe dane – analogicznie jak w przypadku projektu **Maven**.



The screenshot shows the 'New Project' dialog box with the following fields and values:

Project Metadata	
Group:	pl.coderslab
Artifact:	demospringboot
Type:	Maven Project (Generate a Maven based project archive) ▼
Language:	Java ▼
Packaging:	Jar ▼
Java Version:	8 ▼
Version:	0.0.1-SNAPSHOT
Name:	demospringboot
Description:	Demo project for Spring Boot
Package:	pl.coderslab.demospringboot

At the bottom right, there are four buttons: 'Previous', 'Next' (highlighted), 'Cancel', and 'Help'.

Spring Boot

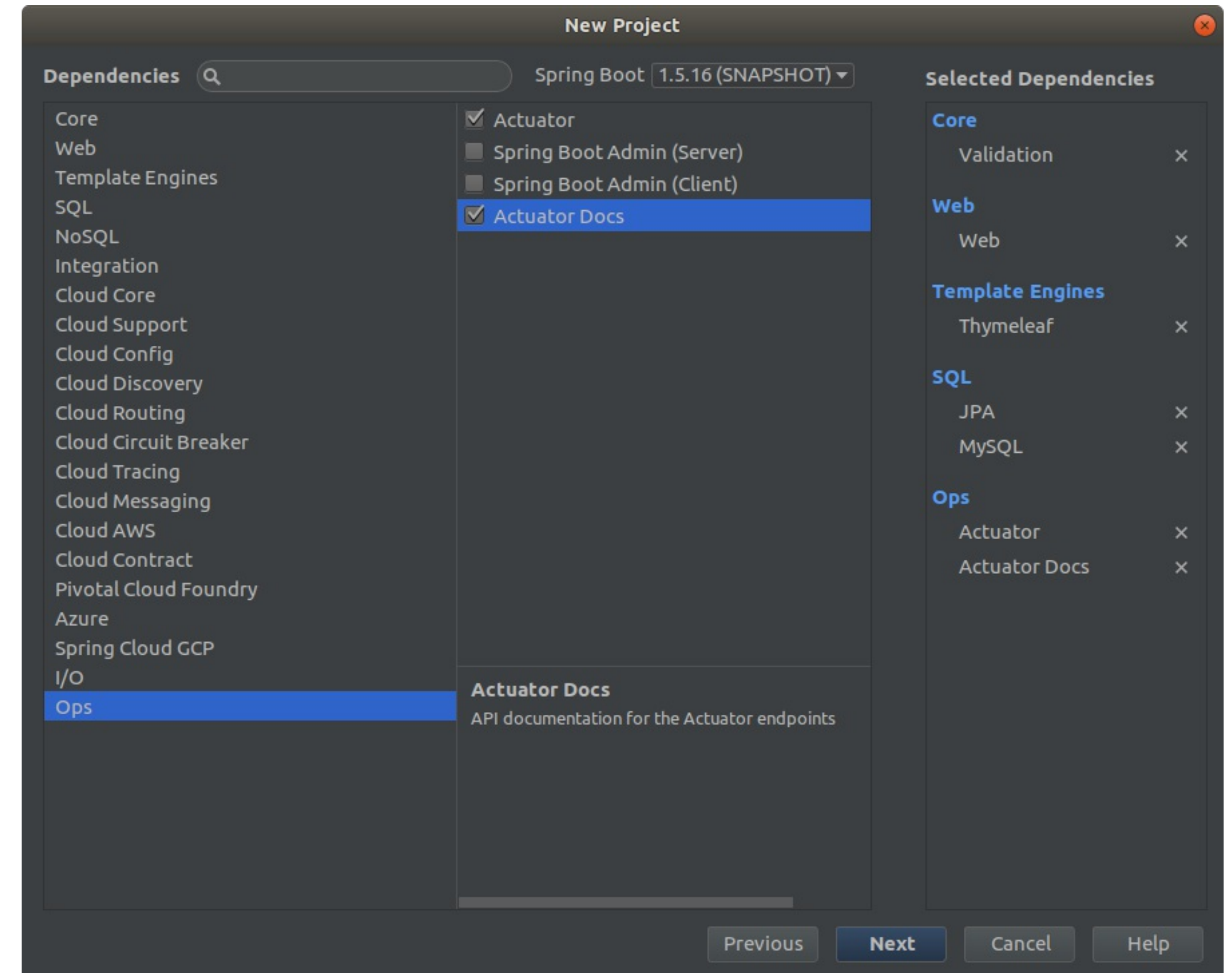
W następnym kroku określamy, jakie startery chcemy umieścić w naszej aplikacji.

W menu po lewej są moduły, które zawierają po kilka opcji. Lista wszystkich wybranych zależności znajduje się po prawej stronie.

Na niebiesko są wypisane moduły, a pod nimi startery, które trzeba zaznaczyć przy tworzeniu nowej aplikacji.

Wybierz je analogicznie w swoim projekcie.

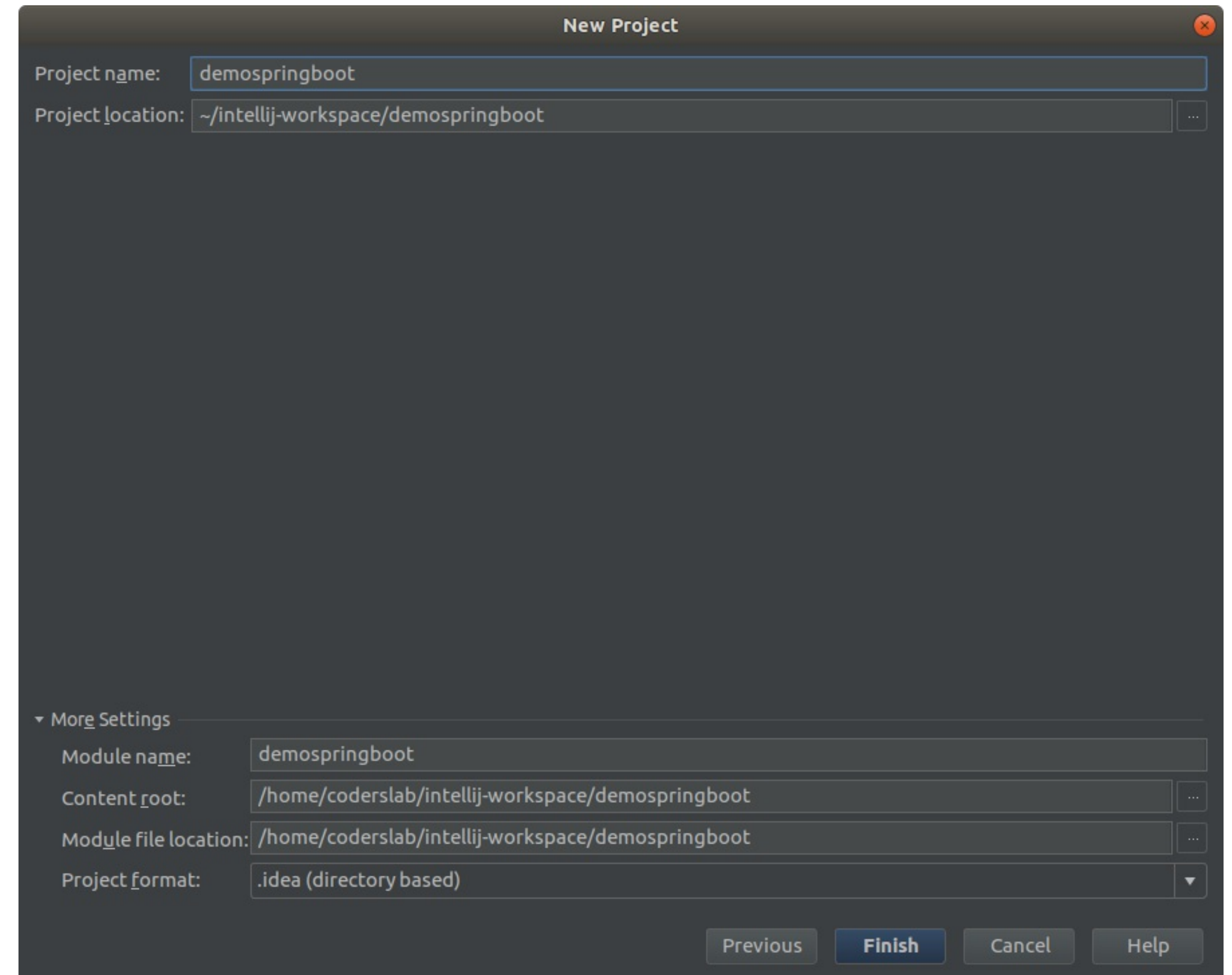
Wybierz także wersję **Spring Boota**, która zaczyna się od cyfry 1.



Spring Boot

Pojawi się okno końcowe, w którym klikamy opcję **Finish**.

Tworzy się nowy projekt już uzupełniony w potrzebne zależności i skonfigurowany odpowiednio do tworzenia aplikacji w Springu.



Spring Boot

Jako przykład skonfigurujemy aplikację zawierającą:

- **Web** - czyli wsparcie dla Spring MVC
- **Thymeleaf** - system szablonów
- **Validation** - wsparcie dla walidacji
- **JPA** - czyli wsparcie dla JPA
- **MySQL** - czyli wsparcie dla bazy danych
- **Actuator** - czyli narzędzie do monitorowania stanu aplikacji

Struktura wygenerowanej aplikacji

Struktura katalogów:

- **src/main/java** - tutaj umieścimy nasze pakiety oraz klasy Java, zawiera automatycznie utworzoną klasę startową o nazwie **Application**),
- **src/test/java** tutaj umieścimy testy naszej aplikacji,
- **src/main/resources** to miejsce na zasoby; podkatalog static ma zawierać pliki przetwarzane po stronie klienta (obrazki, JavaScript), a podkatalog templates szablony przetwarzane po stronie serwera,

Spring Boot - uruchomienie

Ponieważ zdefiniowaliśmy, że nasza aplikacja ma używać **MySQL**, przed uruchomieniem aplikacji **Spring Boot** wymaga od nas zdefiniowania danych dostępowych do bazy.

Dane konfiguracyjne aplikacji musimy umieścić w pliku **application.properties**.

Na konsoli otrzymamy komunikat o błędzie:

```
*****  
APPLICATION FAILED TO START  
*****
```

Description:

```
Cannot determine embedded database driver class for  
database type NONE
```

Ustawienia aplikacji

application.properties - jest to plik, który w naszej wygenerowanej aplikacji znajduje się w lokalizacji **src/main/resources** w pliku tym umieszczamy wpisy w postaci:

klucz = wartość

Przykładowe ustawienia danych dostępowych do bazy danych:

```
spring.jpa.hibernate.ddl-auto=create  
spring.datasource.url=jdbc:mysql://localhost:3306/db_example  
spring.datasource.username=springuser  
spring.datasource.password=ThePassword
```

Ustawienia aplikacji

Listę najbardziej popularnych ustawień znajdziemy pod adresem:
<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

Podpowiedzi udzieli nam również **IDE** tak jak w przypadku plików **java** przy użyciu kombinacji klawiszy **Ctrl+ spacja**.

Spring Boot - JSP

W przypadku gdy jako warstwy widoku chcemy używać plików **jsp** dodajemy do pliku **pom.xml** następujące zależności:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Jeżeli tworząc projekt za pomocą **IDE** lub **Initializera**, wybraliśmy opcję **Thymeleaf** należy odpowiedzialną za niego zależność usunąć z pliku.

Spring Boot - JSP

W pliku **application.properties** dodajemy wpisy odpowiedzialne za konfigurację **ViewResolver**:

```
spring.mvc.view.prefix=/WEB-INF/views/  
spring.mvc.view.suffix=.jsp
```

Jeżeli nie posiadamy w naszej aplikacji pliku **application.properties** należy go utworzyć w lokalizacji: **/src/main/resources**.

Dla powyższej konfiguracji pliki **.jsp** umieszczamy w **/src/main/webapp/WEB-INF/views**.

Dlaczego dopiero teraz ?

Wiele osób może sobie zadać pytanie, skoro mogę w łatwy sposób wygenerować podstawowy szablon aplikacji, w którym od samego początku można zacząć pisać kod realizujący logikę biznesową - dlaczego poznajemy go tak późno.

Bez znajomości ręcznej konfiguracji nie bylibyśmy w stanie stwierdzić, gdzie może leżeć błąd w przypadku jego wystąpienia oraz jakie komponenty mogą być odpowiedzialne za zmiany, które chcemy wprowadzić.

Spring Actuator

Spring Actuator

Actuator daje nam możliwość uzyskania w czasie rzeczywistym przydatnych informacji na temat działania naszej aplikacji.

Aby skorzystać z jego możliwości, dodajemy do pliku **pom.xml** następujący starter:

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

Możemy go również wskazać podczas tworzenia projektu **Spring Boot** przy użyciu **Initializer** lub możliwości naszego **IDE**.

Spring Actuator

Actuator udostępnia adresy (**endpointy**) wyświetlające określone informacje.

Endpointy możemy podzielić na publiczne i prywatne (dostępne tylko po zalogowaniu).

Co jest całkowicie zrozumiałe ze względu na bezpieczeństwo naszej aplikacji.

Endpointy publiczne:

- **/health** - zwraca proste informacje na temat statusu naszej aplikacji
- **/info** - wyświetla dowolne informacje o aplikacji.

Spring Actuator

Istnieje możliwość upublicznienia endpointów prywatnych - w tym celu musimy w pliku **application.properties** dodać następujący wpis:

```
management.security.enabled=false
```

W kolejnym module dowiemy się, w jaki sposób zabezpieczyć dostęp do naszej aplikacji przy pomocy **Spring Security** - pamiętajmy wtedy o zmianie tego ustawienia.

Nie należy udostępniać publicznie aplikacji z dostępem do wszystkich endpointów Actuatora.

Za pomocą ustawień w **application.properties** możemy również określić prefiks dla wszystkich endpointów, służy do tego wpis:

```
management.context-path=/appinfo
```

Endpoint /info

Adres ten zwraca dowolne informacje o naszej aplikacji, które przy pomocy odpowiednich wpisów dodajemy do pliku **application.properties**

Przykładowe wpisy:

```
info.app.description=Created with love  
info.app.java.source=@java.version@  
info.app.version = @version@  
info.app.name=Coderslab.pl Example Actuator App
```

Endpoint /info

Adres ten zwraca dowolne informacje o naszej aplikacji, które przy pomocy odpowiednich wpisów dodajemy do pliku **application.properties**

Przykładowe wpisy:

```
info.app.description=Created with love  
info.app.java.source=@java.version@  
info.app.version = @version@  
info.app.name=Coderslab.pl Example Actuator App
```

@java.version@ - jest to wartość pobierana z właściwości określonych w pliku **pom.xml** - tag **<java.version>** zawarty w tagu: **<properties>**.

Endpoint /info

Adres ten zwraca dowolne informacje o naszej aplikacji, które przy pomocy odpowiednich wpisów dodajemy do pliku **application.properties**

Przykładowe wpisy:

```
info.app.description=Created with love  
info.app.java.source=@java.version@  
info.app.version = @version@  
info.app.name=Coderslab.pl Example Actuator App
```

@java.version@ - jest to wartość pobierana z właściwości określonych w pliku **pom.xml** - tag **<java.version>** zawarty w tagu: **<properties>**.

@version@ - jest to wartość pobierana z właściwości określonych w pliku **pom.xml** - tag **<version>**.

Endpoint /info

W efekcie wywołania adresu: <http://localhost:8080/info> otrzymamy:

```
{
  app: {
    version: "0.0.1-SNAPSHOT",
    description: "Created with love",
    java: {
      source: "1.8.0_151"
    },
    name: "Coderslab.pl Example Actuator App"
  }
}
```

Endpoint /health

Endpoint **/health** zwraca informacje na temat statusu naszej aplikacji. W efekcie wywołania adresu: <http://localhost:8080/info> otrzymamy informacje zbliżone do poniższych:

```
{
  status: "UP",
  diskSpace: {
    status: "UP",
    total: 237607321600,
    free: 7443353600,
    threshold: 10485760
  },
  db: {
    status: "UP",
    database: "MySQL",
    hello: 1
  }
}
```

Spring Actuator Docs

Łatwym sposobem na poznanie wszystkich endpointów, jakie są dostępne, jest dodanie do projektu dodatkowego startera, który udostępni nam dokumentację pod domyślnym adresem:

<http://localhost:8080/docs/>

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

Możemy go również wskazać podczas tworzenia projektu **Spring Boot** przy użyciu **Initializer** lub możliwości naszego **IDE**.

Przydatne endpointy

Pozostałe endpointy przydatne do analizy stanu naszej aplikacji to:

- **/mappings** - udostępnia informacje o wszystkich adresach url, jakie udostępnia nasza aplikacja
- **/trace** - udostępnia informacje o ostatnich adresach url, jakie zostały wywołane wraz z nagłówkami - domyślnie dostępnych jest 100 ostatnich wywołań
- **/metrics** - udostępnia informacje o ostatnich adresach url, jakie zostały wywołane wraz z nagłówkami - domyślnie dostępnych jest 100 ostatnich wywołań
- **/beans** - wyświetla wszystkie beany naszej aplikacji

Listę wszystkich dostępnych endpointów znajdziemy pod adresem

<https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html#production-ready-endpoints>