

# Spring in GCP

- [Instrukcja](#)
  - [Aplikacja](#)
    - [Wygenerowanie projektu](#)
    - [Uruchamianie projektu](#)
    - [Kontrola wersji](#)
  - [Zadania](#)
    - [Zadanie 1](#)
    - [Zadanie 2](#)
    - [Zadanie 3](#)
    - [Zadanie 4](#)
    - [Zadanie 5](#)
    - [Zadanie 6](#)
  - [Deployment w Google Cloud Platform](#)
    - [Setup środowiska](#)
    - [App Engine](#)
    - [Gradle + GCP](#)

# Instrukcja

## Aplikacja

### Wygenerowanie projektu

Aby wygenerować podstawową działającą aplikację SpringBootową, przejdź do <https://start.spring.io/> i wybierz:

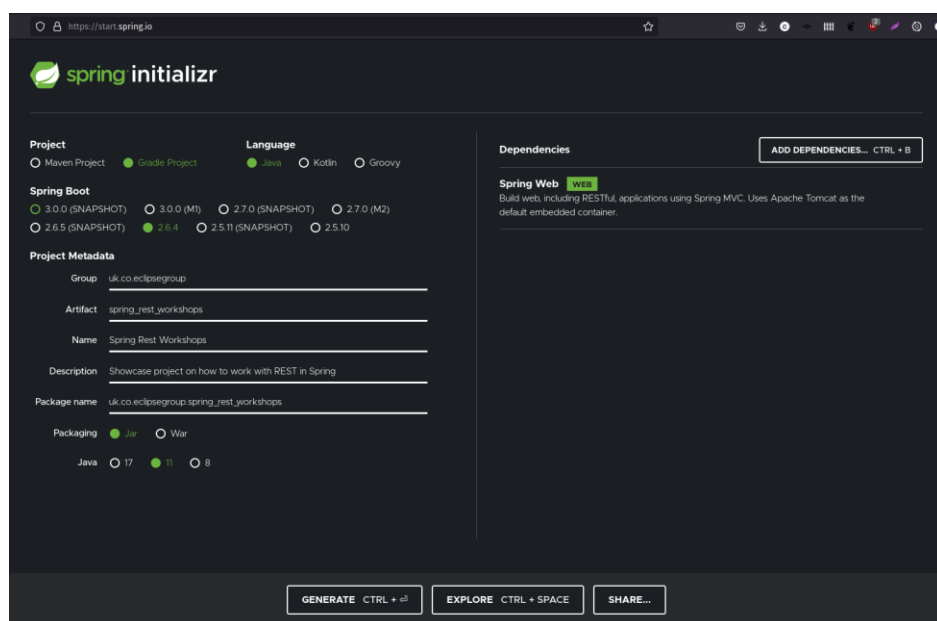
1. Gradle Project
2. Java
3. Zostaw domyślną wersję (2.6.4 w momencie pisania tej instrukcji)
4. Pakowanie do JAR
5. Wersja Javy 11

Jeśli posiadasz własną domenę internetową, wpisz ją w odwrotnej kolejności do **group**. Przykładowo, dla domeny <http://eclipsegroup.co.uk> będzie to **uk.co.eclipsegroup**. Będzie to początek bazowego pakietu w naszej aplikacji.

Artifact to nazwa aplikacji w obrębie danej organizacji. Będzie druga część bazowego pakietu naszej aplikacji. W tym wypadku będzie to **spring\_rest\_workshops**.

Podaj też nazwę oraz opis projektu.

Wygeneruj projekt.



The screenshot shows the Spring Initializr web application interface. The browser address bar displays <https://start.spring.io>. The page features the 'spring initializr' logo at the top left. Below the logo, there are several configuration sections:

- Project:** Radio buttons for 'Maven Project' and 'Gradle Project' (selected).
- Language:** Radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'.
- Spring Boot:** Radio buttons for versions: '3.0.0 (SNAPSHOT)', '3.0.0 (M1)', '2.7.0 (SNAPSHOT)', '2.7.0 (M2)', '2.6.5 (SNAPSHOT)', '2.6.4' (selected), '2.5.11 (SNAPSHOT)', and '2.5.10'.
- Project Metadata:** Text input fields for 'Group' (uk.co.eclipsegroup), 'Artifact' (spring\_rest\_workshops), 'Name' (Spring Rest Workshops), 'Description' (Showcase project on how to work with REST in Spring), and 'Package name' (uk.co.eclipsegroup.spring\_rest\_workshops).
- Packaging:** Radio buttons for 'Jar' (selected) and 'War'.
- Java:** Radio buttons for versions: '17', '11' (selected), and '8'.
- Dependencies:** A section with a search bar containing 'Spring Web' and a list of dependencies. A button 'ADD DEPENDENCIES... CTRL + B' is visible.

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

### Uruchamianie projektu

Rozpakuj ściągnięte archiwum i otwórz w IntelliJ plik **build.gradle** jako projekt.

Aby potwierdzić, że wszystko działa, uruchom aplikację:

- Z konsoli: `./gradlew bootRun`
- W IntelliJ uruchamiając metodę `main` w klasie **SpringRestWorkshopsApplication**, która jest w pakiecie **uk.co.eclipsesgroup.spring\_rest\_workshops**

W jednym i drugim przypadku, możemy uruchomić w przeglądarce <http://localhost:8080/>.

Zostaniemy powitani stroną błędu **404 Not Found**:



To znaczy, że serwer się uruchamia, więc jesteśmy gotowi do dalszej zabawy.

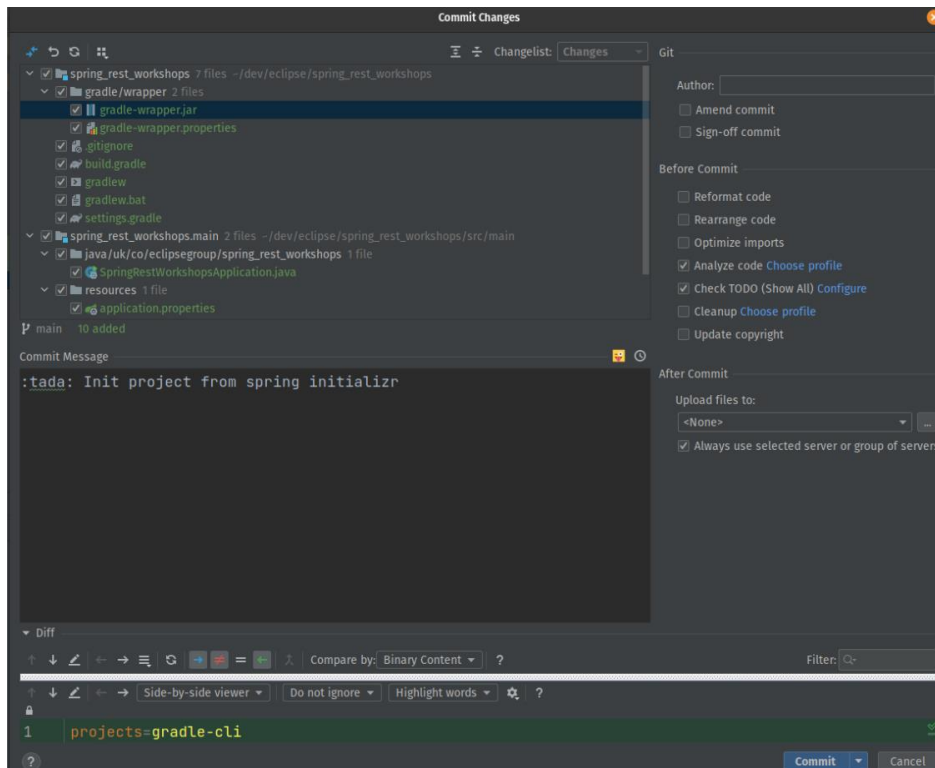
## Kontrola wersji

W konsoli uruchom

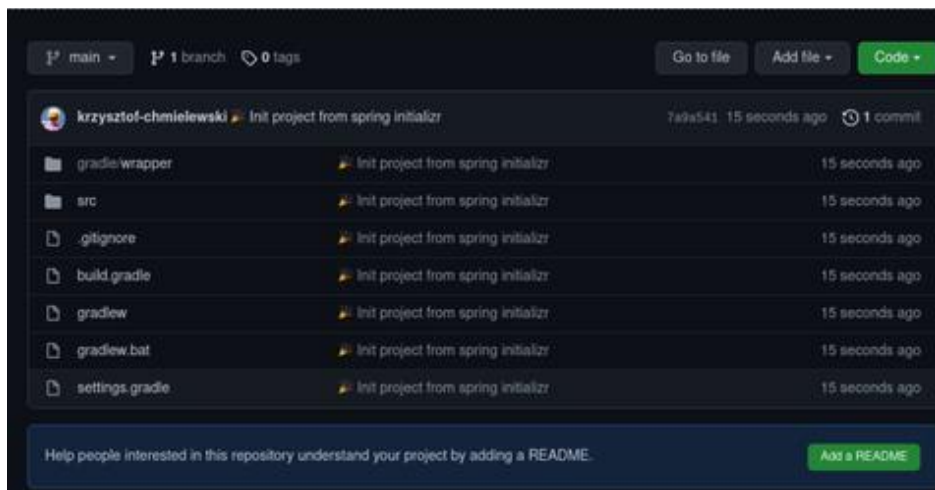
```
git init
```

by zainicjować projekt gitowy.

Następnie stwórzmy pierwszego commita, dodając wszystko, co podpowie nam IntelliJ:



Będziemy używać emotikon zgodnie z wytycznymi <https://gitmoji.dev/> , natomiast format wiadomości commitów będzie zgodny z wytycznymi z dokumentacji: <https://git.kernel.org/pub/scm/git/git.git/tree/Documentation/SubmittingPatches?id=HEAD#n133> . Tak to wygląda po wypushowaniu do GitHuba:



# Zadania

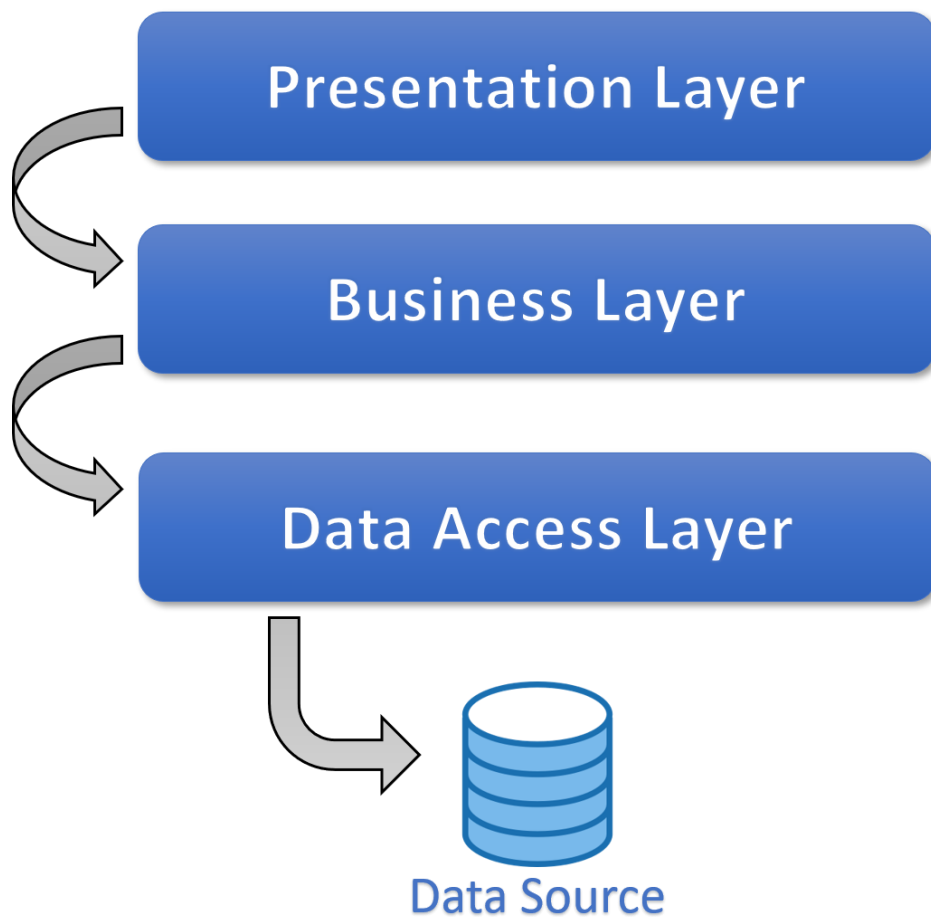
## Zadanie 1

Dodaj kontroler wraz z endpointem obsługującym metodę `GET`, który pozwoli na pobranie wszystkich wersji Javy oraz napisz testy z użyciem `MockMvc`. Nie zapomnij dodać zależności:

```
testImplementation 'org.assertj:assertj-core:3.22.0'
```

by korzystać z biblioteki `assertj`, która ułatwia pisanie asercji.

Spróbuj zrealizować to zadanie korzystając z architektury warstwowej (Kontroler korzysta z Serwisu → Serwis korzysta z repozytorium → Repozytorium posiada tylko dane)



## Zadanie 2

Dodaj endpoint, który pozwoli na dodawanie wersji Javy. Odpowiednia będzie metoda `Post`. Nie zapomnij o testach!

### Zadanie 3

Zmodyfikuj powyższy endpoint, by pozwolić na dodawanie wielu wersji Javy na raz. Niezbędne będzie dodanie do `application.properties`:

```
spring.jackson.deserialization.accept-single-value-as-array=true
```

Sprawi to, że Spring podczas próby deserializacji kodu w endpointzie, który przyjmuje kolekcję, zaakceptuje pojedynczy element i opakuje go w kolekcję, czyli:

```
{
  "name": "JDK 1.0",
  "version": 1.0
}
```

zostanie potraktowane jako:

```
[
  {
    "name": "JDK 1.0",
    "version": 1.0
  }
]
```

### Zadanie 4

Dodaj endpoint, który pozwoli na usuwanie wersji Javy np. na podstawie nazwy.

### Zadanie 5

Wyślij zapytanie GET do <https://quickchart.io/>, w którym będą przekazane wersje Javy w celu wygenerowania wykresu.

### Zadanie 6

Zmodyfikuj powyższy kod, by to zapytanie było żądaniem POST, tak jak na <https://quickchart.io/documentation/#post-endpoint> . Przetestuj samo budowanie zapytania.

# Deployment w Google Cloud Platform

## Setup środowiska

1. Załóż konto Google, może być te które już masz, ale pamiętaj że bezpieczniej jest zawsze założyć nowe niezależne konto tylko w tym celu.
  1. Pamiętaj o silnym hasle
  2. Pamiętaj o second factor:  
<https://support.google.com/accounts/answer/185839?hl=en&co=GENIE.Platform%3DAndroid>
  3. Przez pierwsze 3 miesiące możemy wykorzystać 300\$ za darmo żeby po eksplorować Google Cloud, potem część serwisów (jak App Engine) są darmowe w ograniczonym zakresie : <https://cloud.google.com/free>
2. Setup GCP
  1. Zainstaluj Google Cloud CLI <https://cloud.google.com/sdk/docs/install#linux>
  2. Zaloguj się: `gcloud auth login` więcej tutaj:  
<https://cloud.google.com/sdk/docs/initializing>

## App Engine

GCP App Engine: <https://cloud.google.com/appengine>

1. Wejdź do folderu z kodem i wywołaj `gcloud init`. Następnie wybierz opcje create new project i wpisz unikalną nazwe projektu ([więcej o projektach](#))
2. Upewnij się że billing jest włączony dla twojego projektu ([dokumentacja](#))
3. `gcloud app create` -> następnie wybierz europe-central-2 (warszawa) ([więcej o regionach](#))
4. Włącz api do budowania aplikacji w GCP : `gcloud services enable cloudbuild.googleapis.com`
5. Zbuduj jara za pomocą gradle: `./gradlew build`
6. Zdeplouij Jara : `gcloud app deploy build/libs/spring_rest_workshops-0.0.1-SNAPSHOT.jar`

## Gradle + GCP

Stwórz plik app.yaml

```
runtime: java11
entrypoint: 'java -Xmx128m -jar *.jar'
automatic_scaling:
  min_instances: 0
  max_instances: 1
instance_class: F2
```

Dodaj następujące linie na początek pliku settings.gradle (więcej o tym pluginie można znaleźć [tutaj](#))

```

pluginManagement {
    repositories {
        gradlePluginPortal()
        mavenCentral()
    }
    resolutionStrategy {
        eachPlugin {
            if (requested.id.id == "com.google.cloud.tools.appengine") {
                useModule("com.google.cloud.tools:appengine-gradle-
plugin:${requested.version}")
            }
        }
    }
}

```

Edytuj plik build.gradle

```

plugins {
    ...
    id 'com.google.cloud.tools.appengine' version '2.4.2'
}

...

appengine {
    stage {
        setAppEngineDirectory(".")
        artifact="${buildDir}/libs/${project.name}-${version}.jar"
    }
    deploy {
        projectId = "GCLOUD_CONFIG"
        version = "production"
    }
}

```

następnie można wywołać `./gradlew appengineShowConfiguration` aby zobaczyć, czy konfiguracja jest poprawną.

Aby zdeployować naszą aplikację używamy: `./gradlew appengineDeploy`