Spring Boot

Quick Start

Version: 1.0

Date: 03.2021

www.ivoronline.com

ABOUT THIS BOOK

CONTENT

ABOUT THIS BOOK

This is first Book in the series

Spring Boot - Quick Start

Spring Boot - Accessories

Spring Boot - Security

Content

Intention of this Book is to quickly get you started using Spring Boot to develop Web Applications. Book covers basic functionalities like: Controllers, Endpoints, Entities, DTOs, Services, Repositories, etc.

Standalone Tutorials

The core of this book are standalone tutorials that explain different functionalities of Spring Boot.
Each tutorial contains minimum amount of code needed to explain specific functionality. Tutorials have minimum amount of encompassing text that explains related theory and different parts of the code. This approach allows students to grasp presented concepts in a very fast and efficient manner. Full code, which can also be downloaded from GitHub, prevents any time being wasted trying to make the code work. Simple examples allow for full understanding of the functionality without any unnecessary distractions.

Theoretical Background

Where needed tutorials are preceded by chapters focusing on theoretical background. This way reader can fully understand functionalities explained in the subsequent chapters. But such chapters are in minority and of secondary importance because the main focus is on practical applications.

Demo Applications

Book contains demo Applications that show how to combine functionalities covered in previous tutorials. More complex Application "Authors & Books" is broken into multiple steps showing how to add additional functionalities at each step.

WHY TUTORIALS?

"Things are only as complicated as they are badly explained."

Proper documentation is essential to avoid struggle and frustration when working with simple things that only seem complicated by not being properly documented and explained.

WHAT KIND OF TUTORIALS?

"Working example is worth thousand words."

Just like the picture is worth thousand words the same goes for the working example. Documentation in the form of working examples is proved to be the fastest and the most effective way of transferring knowledge. Sometimes an example is all you need to get the things done. And if there are some accompanying comments that explain what is going on even better. This approach is used in this book. This results in fast learning and the ability to apply tutorials when you need them in the spirit of Just In Time Support.

I wish you rapid learning!

1 Quick Start

Info

- Following tutorials show how to prepare development environment that will be used to create Spring Boot Application.
  - Install Java (Programming Language. Java can be installed as part of IntelliJ IDEA installation)
  - Install IntelliJ IDEA (free Community or paid Ultimate Edition with free 30 day trial)
- Then we show how to Create Spring Boot Project that will be used as a starting point for creating our applications
  - Using IntelliJ Community Edition (free)
  - Using IntelliJ Ultimate Edition (paid)

1.1 Install

Info

- Following tutorials show how to install applications which will be used to create Spring Boot Application
  - Java (Programming Language. Java can be installed as part of IntelliJ IDEA installation)
  - Install IntelliJ IDEA (IDE - Integrated Development Environment)

1.1.1 Java

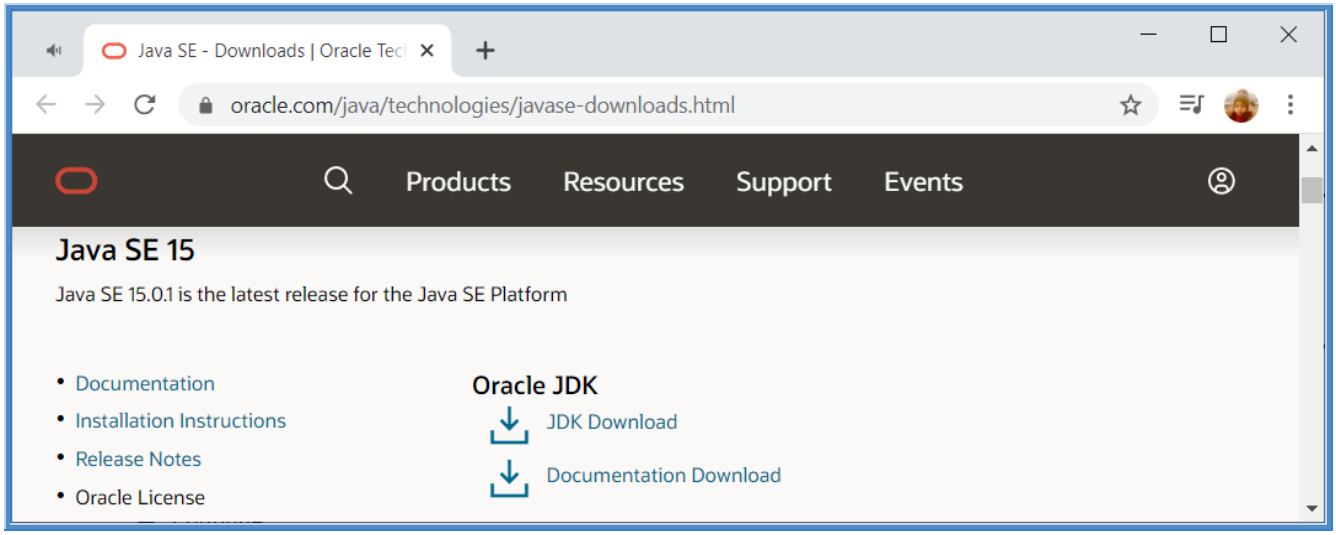
Info

- This tutorial shows how to install JAVA by using Windows installer program.
- Environment variables will be automatically set.
- You can **skip** this step since Java can also be installed as part of IntelliJ IDEA installation

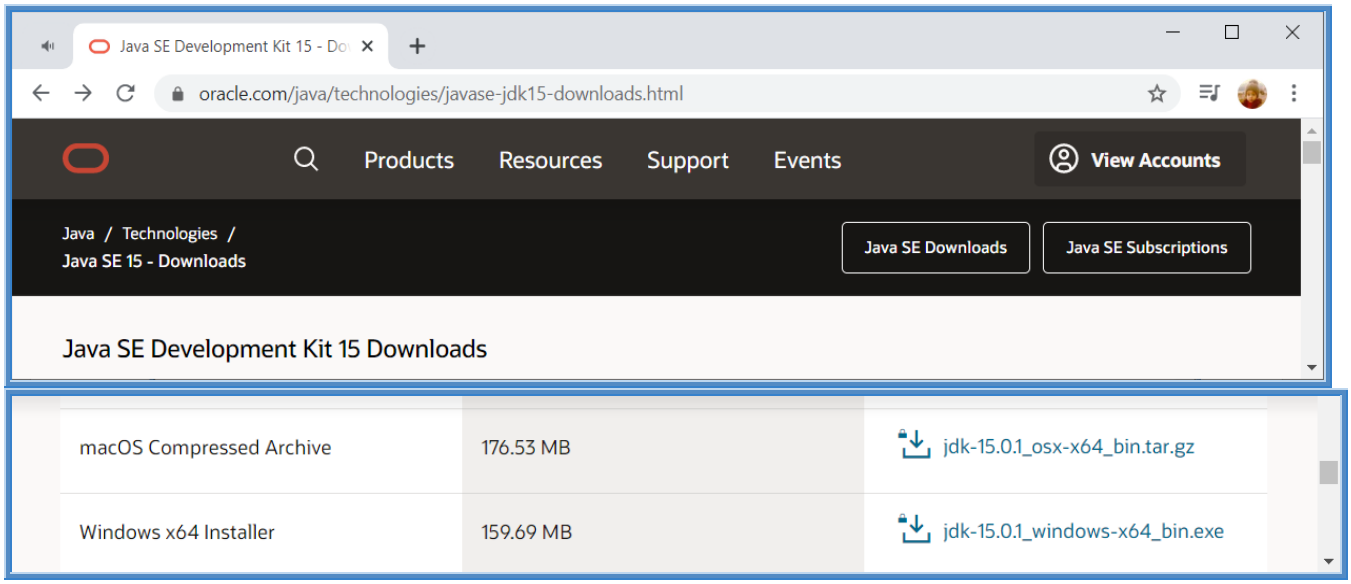
Download & Execute

- http://java.sun.com/javase/downloads/index.jsp
- Oracle JDK
- **jdk-15.0.1_windows-x64_bin.exe**
- I reviewed ...: CHECK
- Download jdk-15.0.1_windows-x64_bin.exe
- Save as: D:\Downloads\jdk-15.0.1_windows-x64_bin.exe
- Execute jdk-15.0.1_windows-x64_bin.exe

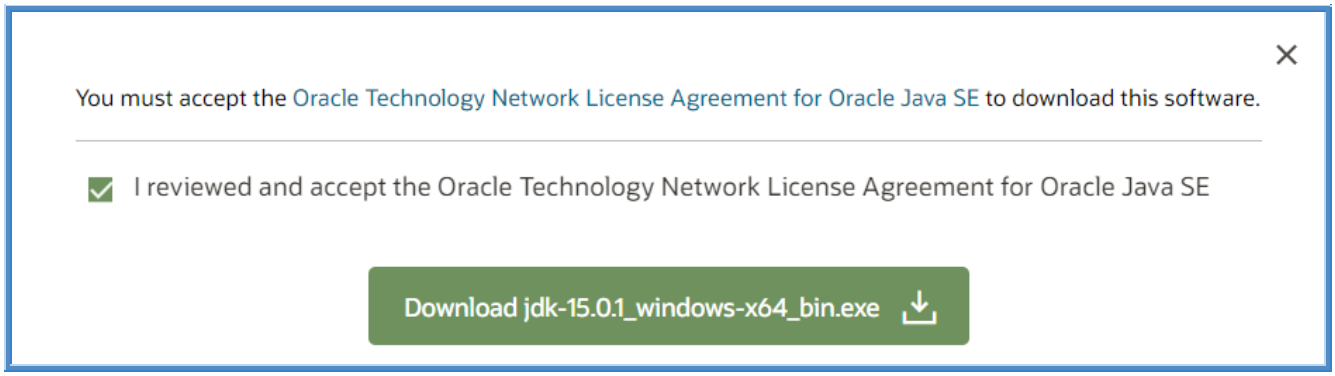
http://java.sun.com/javase/downloads/index.jsp - Oracle JDK



jdk-15.0.1_windows-x64_bin.exe



Download jdk-15.0.1_windows-x64_bin.exe



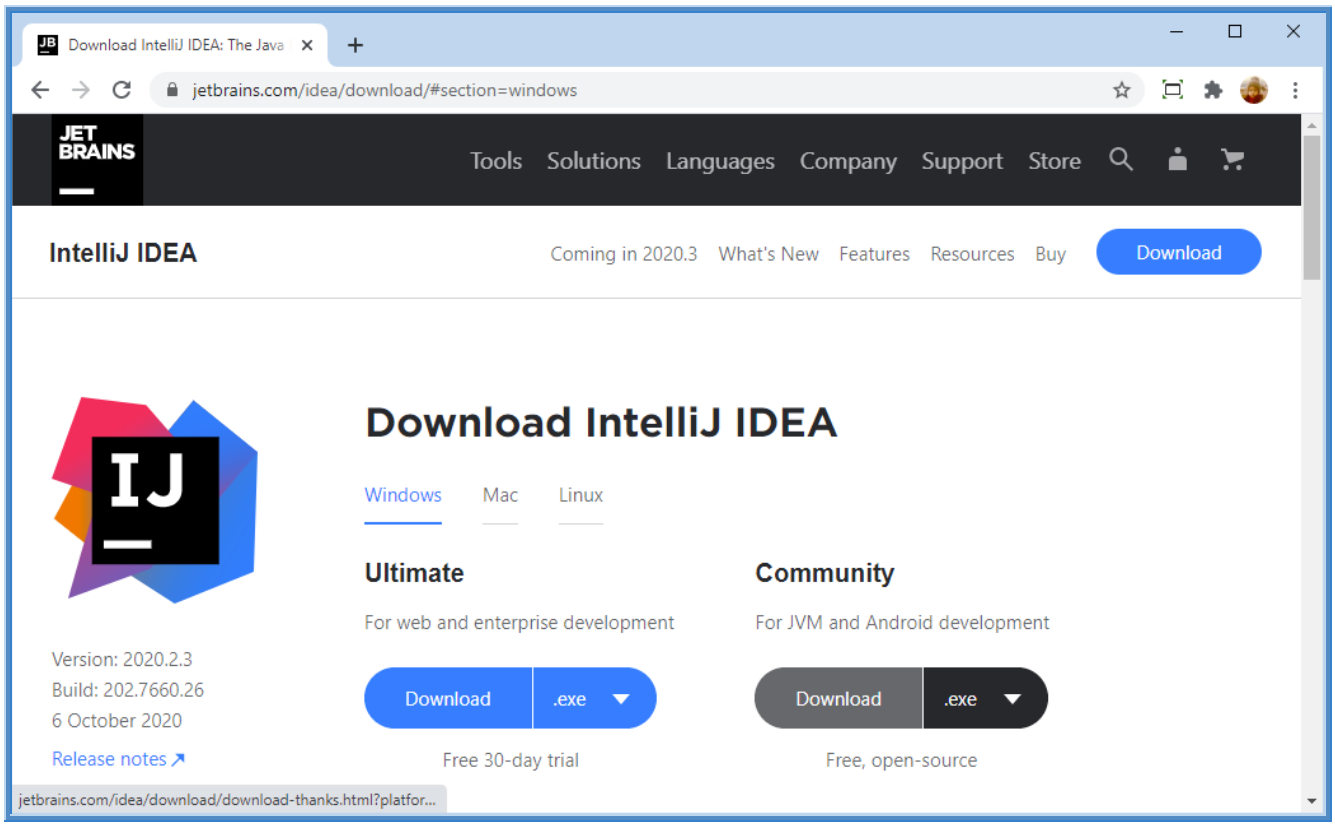
1.1.2 IntelliJ IDEA

Info

- This tutorial shows how to install IntelliJ IDE Integrated Development Environment (IDE)
  - Download IntelliJ IDEA (free Community or paid Ultimate Edition with free 30 day trial)
  - Install IntelliJ IDEA
  - Download Java JDK (if JDK is not already installed on the PC)
  - Customize (select dark or light interface)

Download IntelliJ IDEA

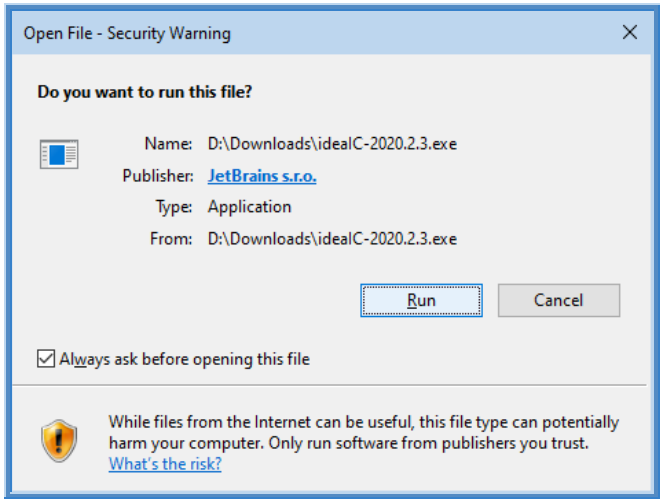
- https://www.jetbrains.com/idea
- Download
- Community
- Download
- Save as: D:\Downloads\ideaIC-2020.2.3.exe



Install IntelliJ IDEA

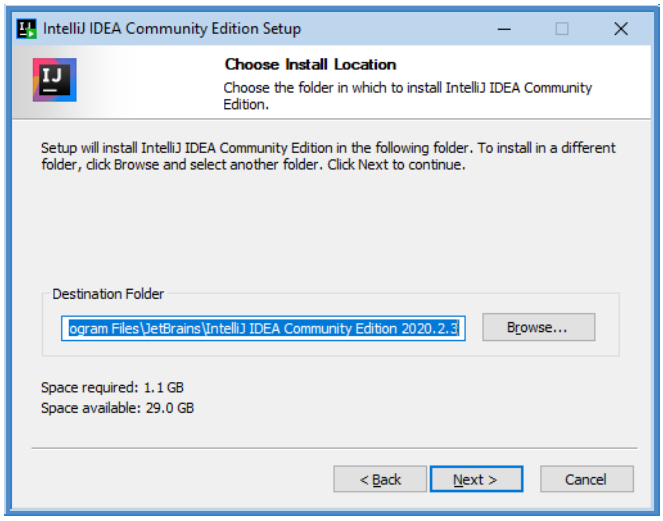
- Execute idealC-2020.2.3.exe
- Run
- 3*Next

Run

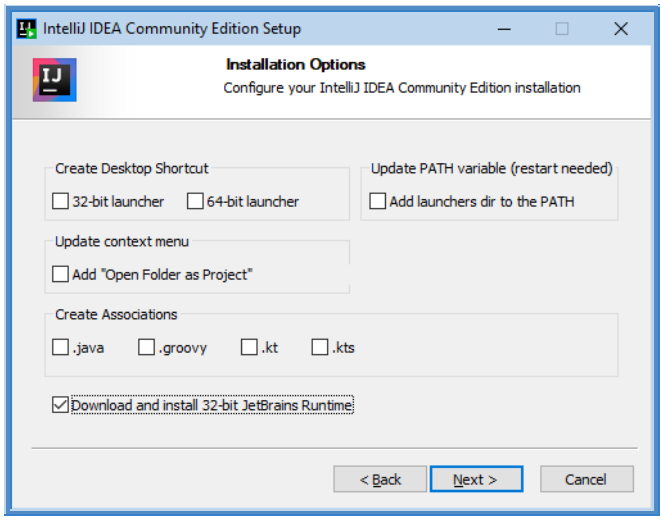


Next

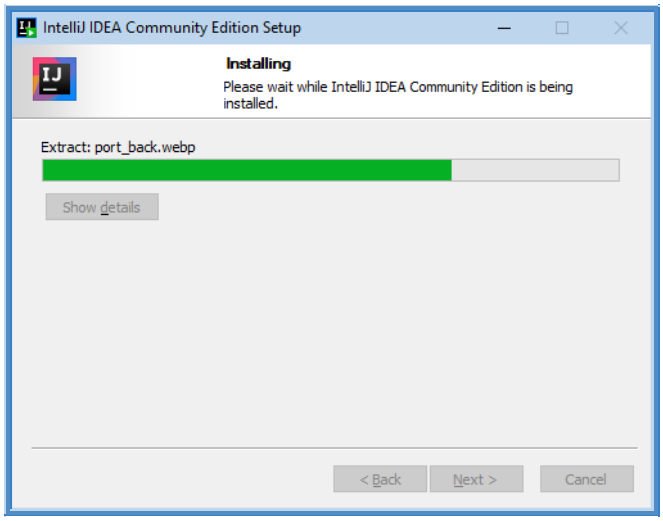
Next



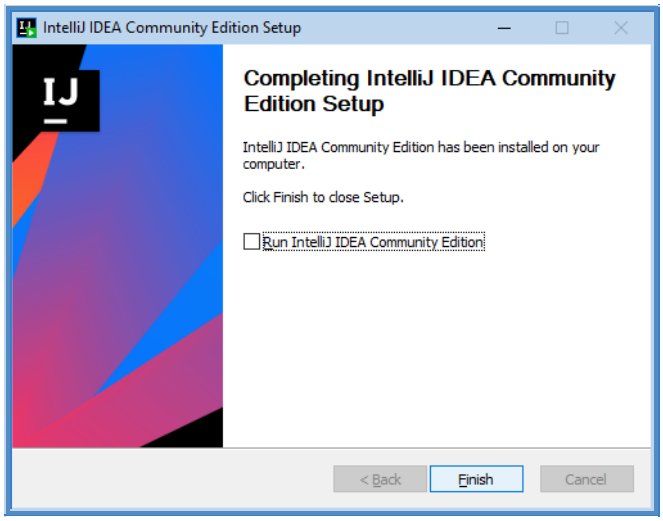
Next



Progress



Finish

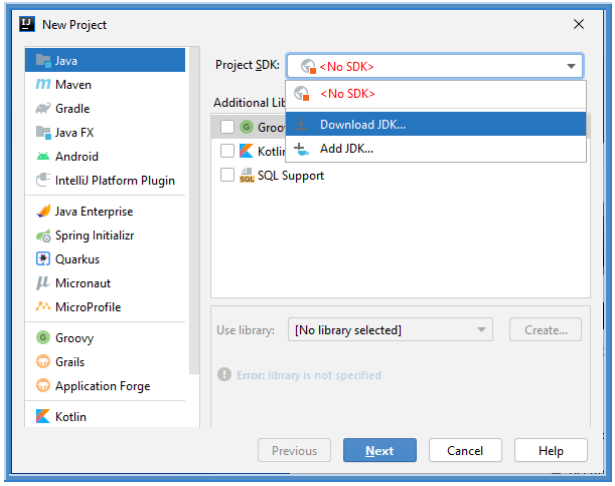


Download Java JDK

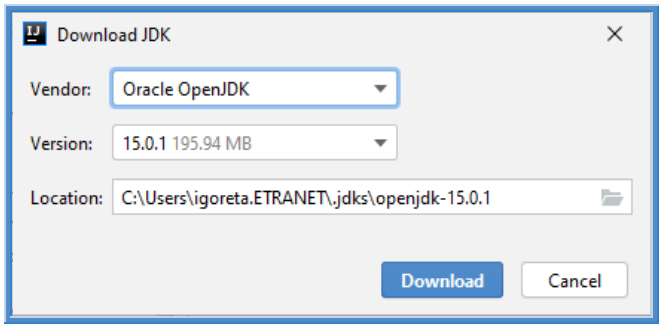
- Start IntelliJ IDEA
- New Project
- Java
- Download JDK

New Project

Download JDK



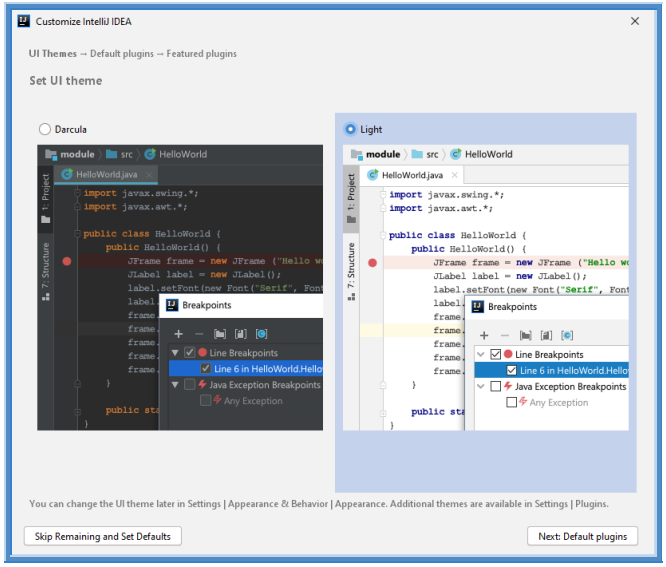
Download



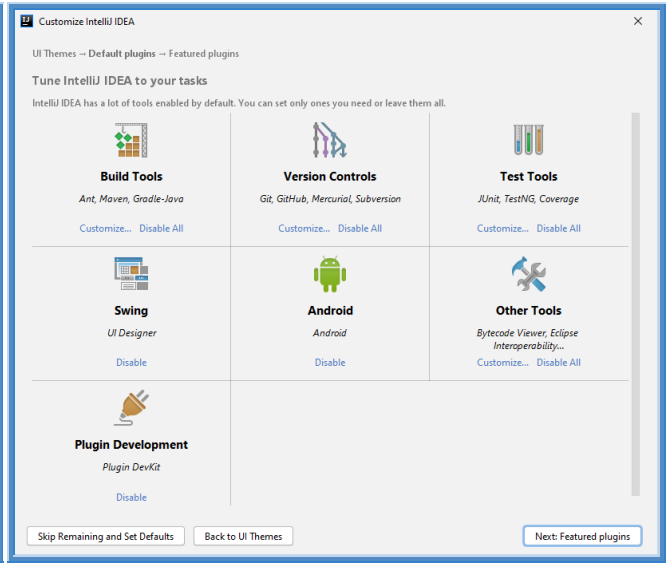
Customize

- Start IntelliJ IDEA
- Light
- 2*Next
- Start

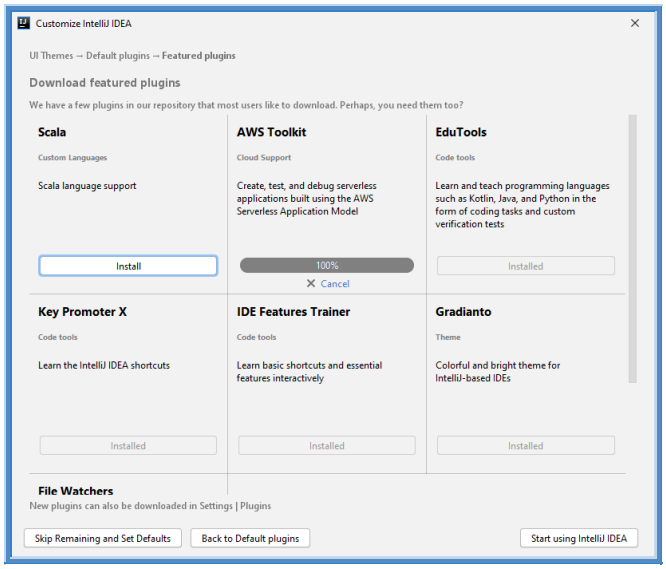
Light



Next



Start

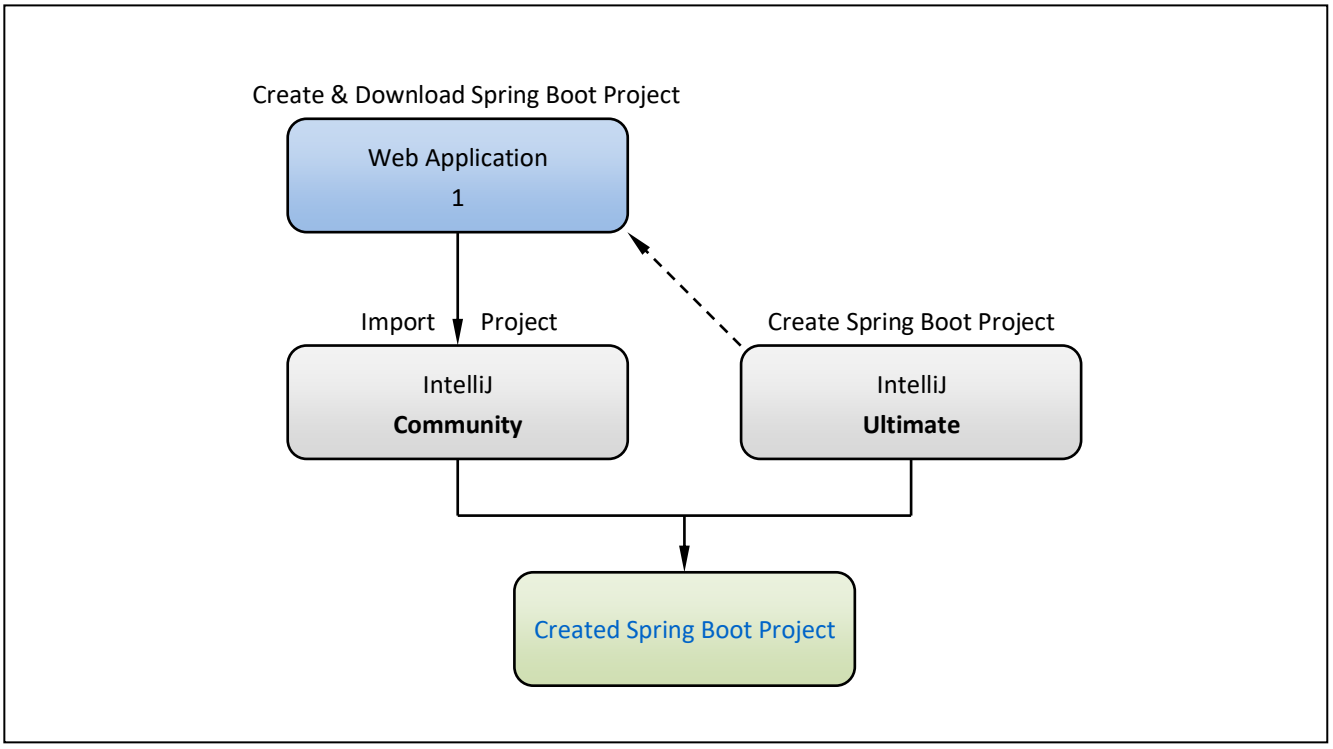


1.2 Create Project

Info

- Following tutorial show how to create Spring Boot Project using IntelliJ IDEA Integrated Development Environment (IDE).
- Depending on the edition of IntelliJ IDEA, Spring Boot Project can be created in two ways
  - Using IntelliJ Community Edition (use https://start.spring.io Web Application to create & download project)
  - Using IntelliJ Ultimate Edition (IntelliJ uses https://start.spring.io Web Application in the background)
- In both cases we end up with the same Spring Boot Project.

Create Spring Boot Project



1.2.1 Using IntelliJ - Community Edition (free)

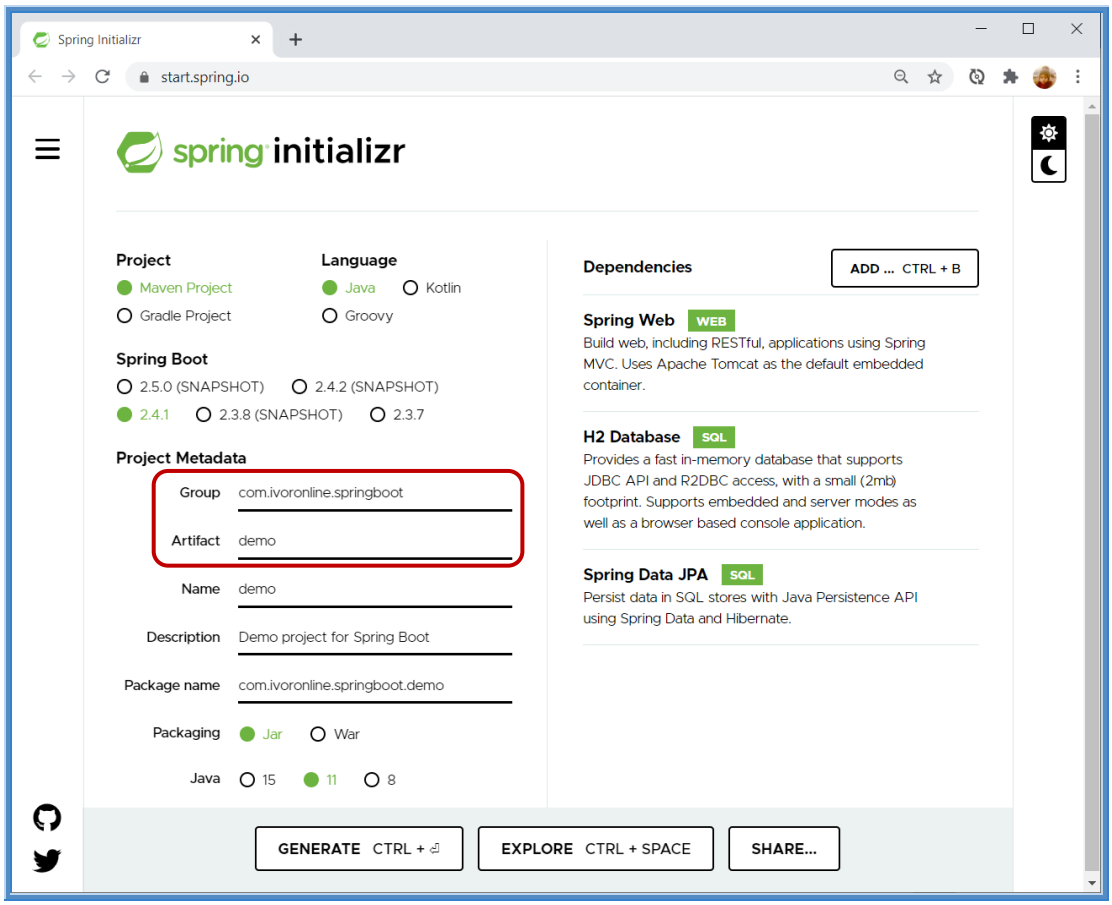
Info

- This tutorial shows how to Create Spring Boot Project using IntelliJ free **Community** edition
  - Create Spring Boot Project (use https://start.spring.io/ Web App that implements Spring Initializer)
  - Import Spring Boot Project into IntelliJ (Maven loads dependency JARs)
- Community edition doesn't support functionality to create Spring Project directly from within IntelliJ IDE.
  Therefore we use https://start.spring.io/ to select Spring Boot Starters (Maven dependencies).
  Downloaded Spring Project will not contain any Java Classes. Maven downloads them after importing Project to IntelliJ.

Create Spring Boot Project

- https://start.spring.io/
- Project: Maven Project
- Language: Java
- Spring Boot: 2.4.1
- Project Metadata
  - Group: com.ivoronline.springboot
  - Artifact: **demo** (Project Name)
  - Packaging: Jar
  - Java: 11
- Dependencies
  - Spring Web
  - Spring Data JPA
  - H2 Database
- Generate
- Save as: C:\Downloads\demo.zip
- Extract demo.zip

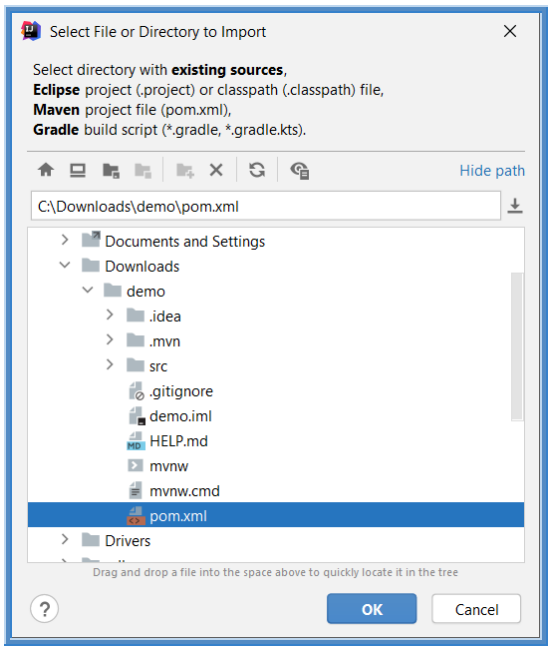
Generate Spring Project


Spring Initializr
start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
2.5.0 (SNAPSHOT)
2.4.2 (SNAPSHOT)
2.4.1
2.3.8 (SNAPSHOT)
2.3.7
Project Metadata
Group
com.ivoronline.springboot
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.ivoronline.springboot.demo
Packaging
Jar
War
Java
15
11
8
Dependencies
ADD ... CTRL + B
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
H2 Database
SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
Spring Data JPA
SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...

Import Spring Boot Project into IntelliJ

- Start IntelliJ
- File - New - **Project from existing sources**
- C:\Downloads\demo\pom.xml
- (wait for it to resolve Maven dependencies - download JARs with Java Classes)

Project from existing sources - pom.xml



1.2.2 Using IntelliJ - Ultimate Edition (paid)

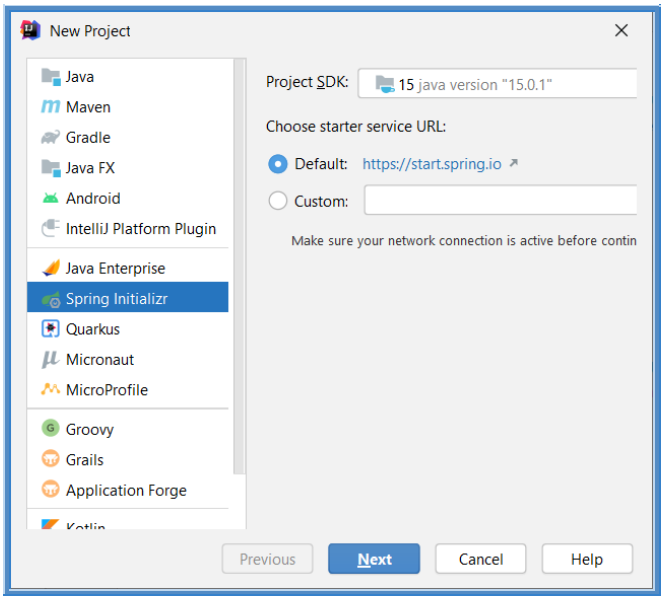
Info

- This tutorial shows how to Create Spring Boot Project using IntelliJ paid **Ultimate** Edition (free 30 day trial).
- In the background IntelliJ uses https://start.spring.io/ Web Application.
- That Web Application is also used in Create Spring Project - Using start.spring.io.

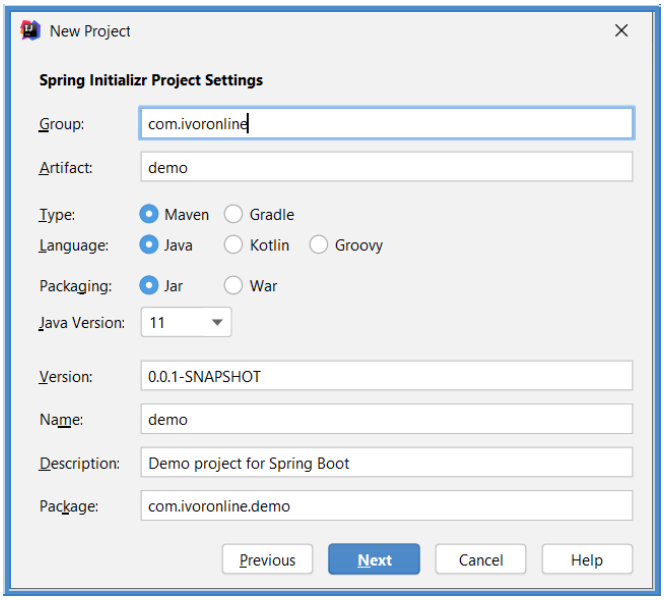
Create Spring Boot Project

- Start IntelliJ
- File - New - Project
- Spring Initializr - Next
- Project Settings
  - Group: com.ivoronline.springboot
  - Artifact: **demo** (Project Name)
  - Type: Maven
  - Language: Java
  - Packaging: Jar
  - Java Version: 11
  - Next
- Dependencies
  - Web: Spring Web
  - SQL: Spring Data JPA
  - SQL: H2 Database
  - Next
- Project name: demo
- Project location: C:\Projects\demo
- Finish
- (wait for it to resolve Maven dependencies - download JARs with Java Classes)

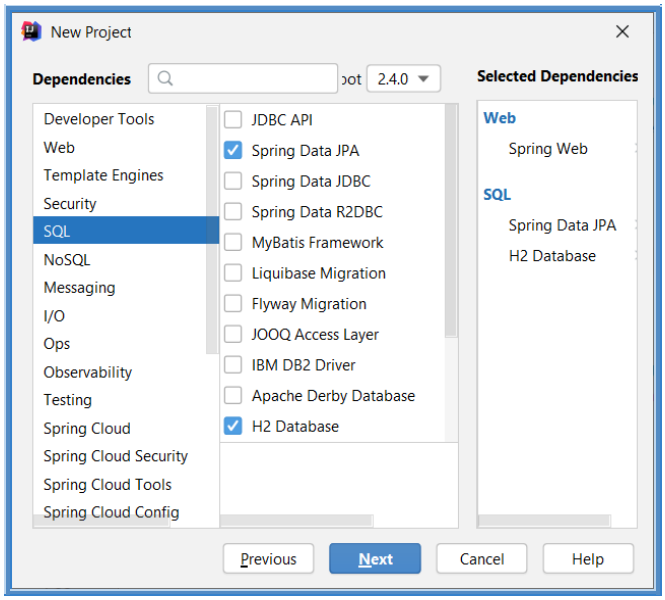
File - New - Project - Spring Initializr



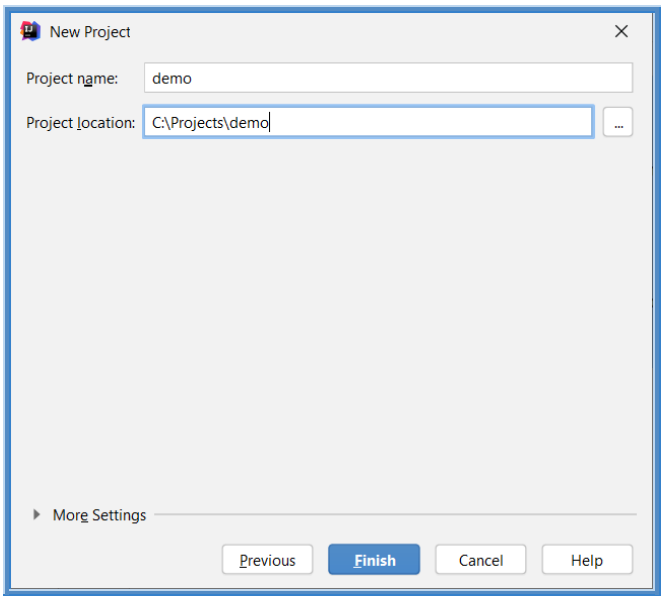
Project Settings



Dependencies



Project name



1.3 Application

Info

- Following tutorials show how to Run and Deploy Application.

1.3.1 Run

Info

- This tutorial shows how to run Spring Boot Application.
- We will and display a message in the Console just to make sure everything is working properly.

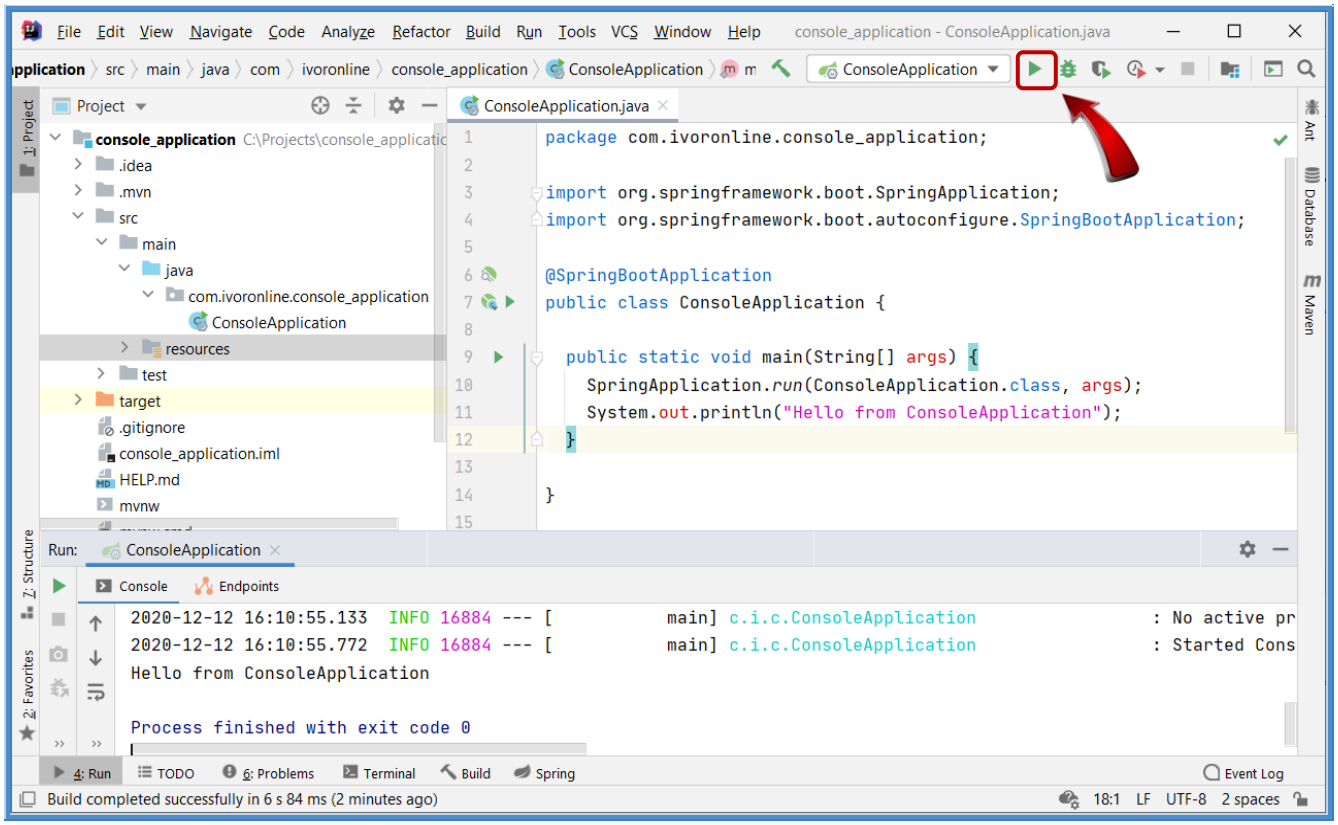
Run Application

- Create Spring Boot Project (no additional dependencies needed)
- Edit Java Class: TestSpringBootApplication.java (insert highlighted line)
- Run Application

ConsoleApplication.java

```
package com.ivoronline.console_application;

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

@SpringBootApplication
public class ConsoleApplication {

  public static void main(String[] args) {
    SpringApplication.run(ConsoleApplication.class, args);
    System.out.println("Hello from ConsoleApplication");
  }

}
```

Application



Console

```
Hello from ConsoleApplication
```

1.3.2 Deploy

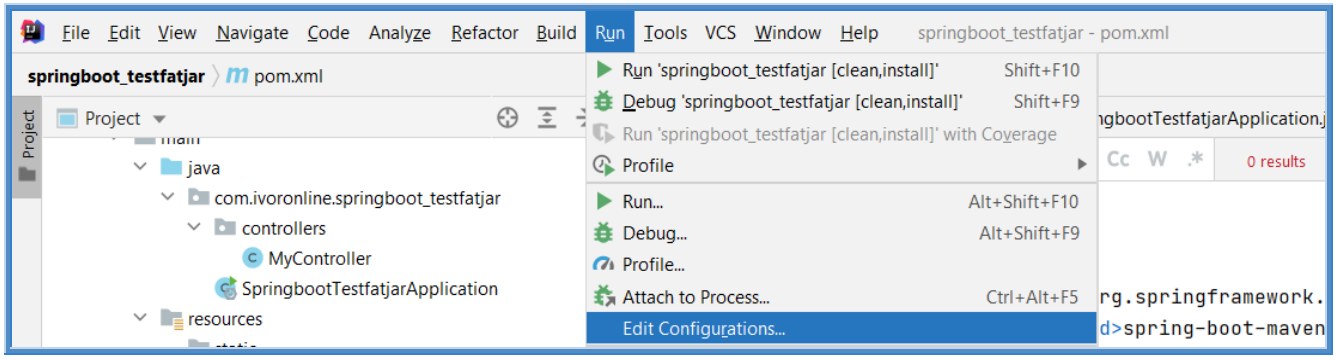
Info [R]

- This tutorial shows how to deploy Spring Boot Application by creating **Fat Jar**.
- Fat Jar will include everything needed for the application to run (including Tomcat Server).
- You can then simply copy Fat Jar (using FTP) to target Server (with Java installed) and start it with Java.
- To demonstrate how this works first create simple Spring Boot Application as shown in Return - Text and then
  - Maven Run Configuration - Create — clean install
  - Maven Run Configuration - Run — creates springboot_testfatjar-0.0.1-SNAPSHOT.jar
  - Run Fat Jar — java -jar myfatapp.jar
  - Results — http://localhost:8085/hello

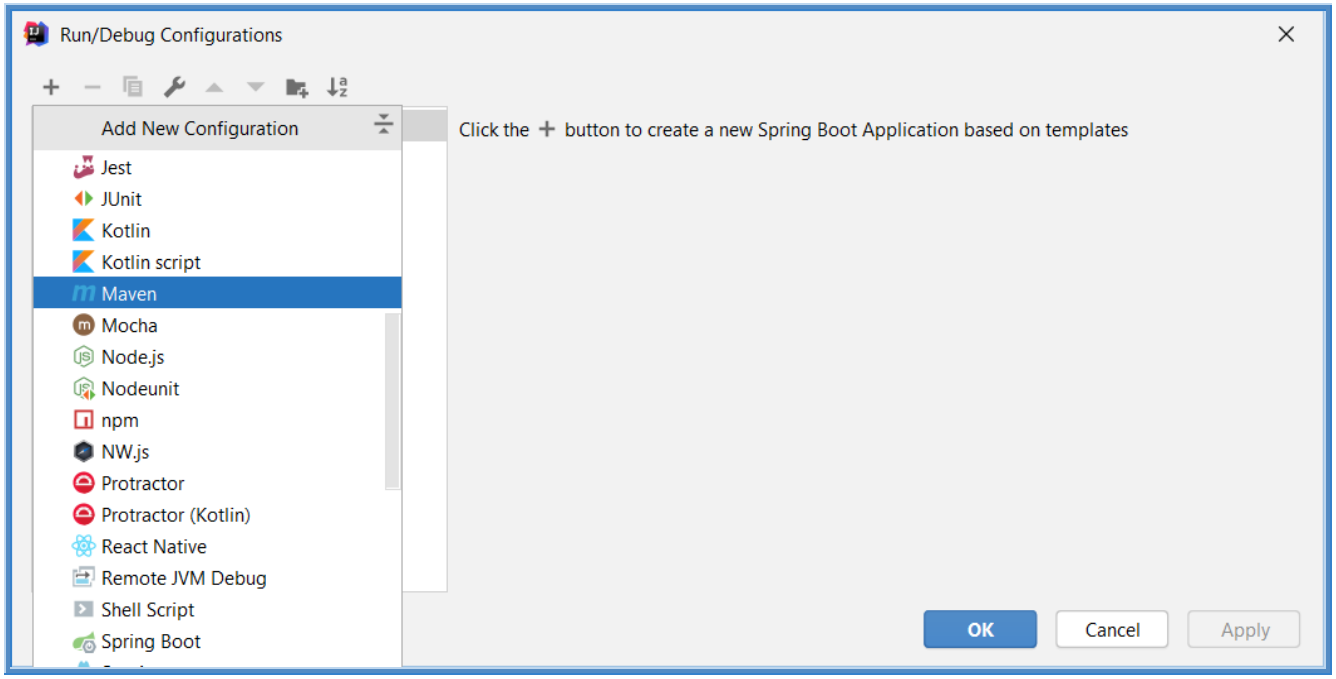
Maven Run Configuration - Create

- Run
- Edit Configurations
- Add New Configuration
- Maven
- Working Directory: C:/Projects/springboot_testfatjar
- Command line: clean install
- OK

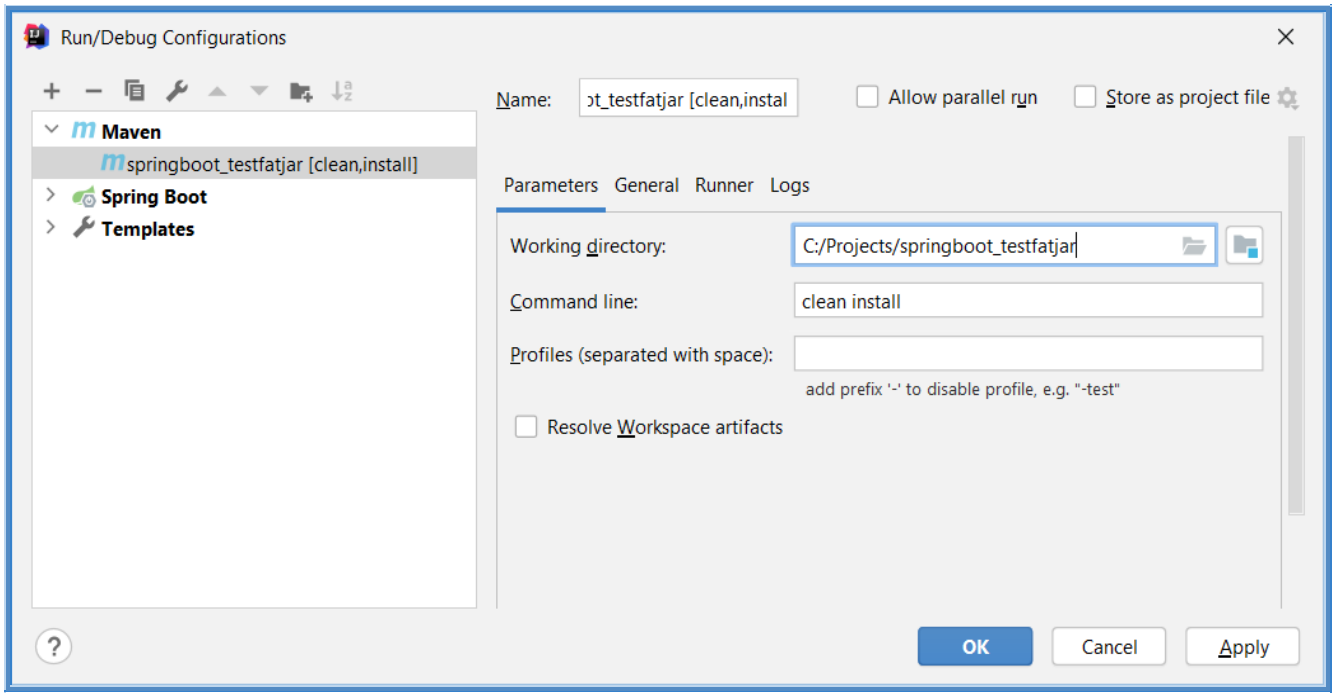
Run - Edit Configurations



Add New Configuration - Maven

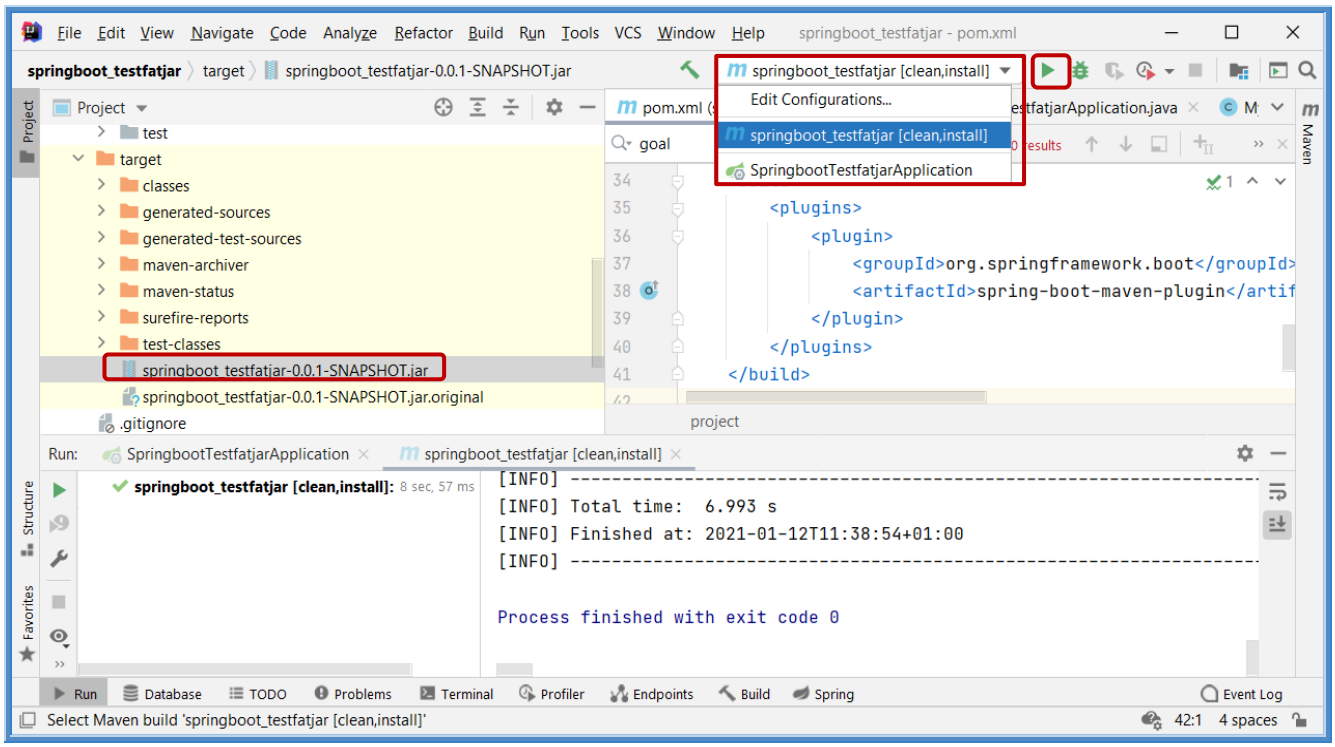


Command line: clean install



Maven Run Configuration - Run

- Select Maven Run Configuration: springboot_testfatjar [clean,install]
- Run (creates springboot_testfatjar-0.0.1-SNAPSHOT.jar)



Run Fat Jar

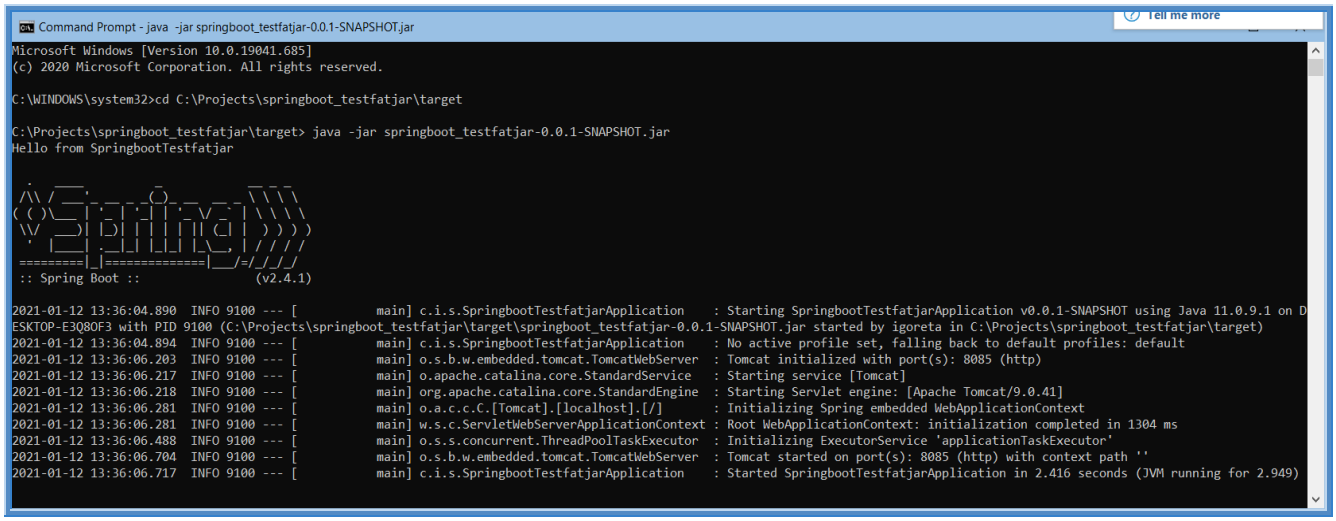
- Start Command Prompt
- (execute below commands)

Command Prompt

```
cd C:\Projects\springboot_testfatjar\target
java -jar springboot_testfatjar-0.0.1-SNAPSHOT.jar
```

Results

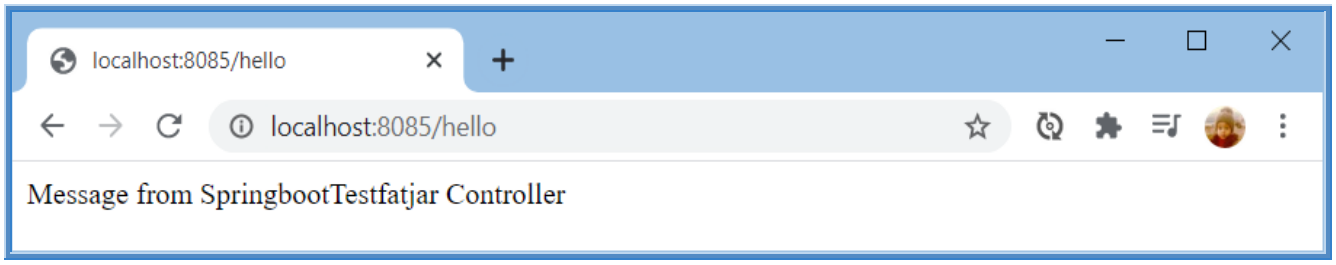
Command Prompt



Command Prompt Output

```
Hello from SpringbootTestfatjar
Tomcat initialized with port(s): 8085 (http)
```

http://localhost:8085/hello



2 Main Terms

Info

- Following tutorials explain main terms related to Spring Boot Development.
- They will be explained as stand-alone applications where each one is used to demonstrate specific functionality.
- Applications are designed to be as simple as possible to demonstrate functionality in question.
- That means that they might not comply to best coding practices because simplicity takes higher priority.
- How to combine all these functionalities in a proper way will be demonstrated in later chapter with Demo Applications.

2.1 Controller

Info

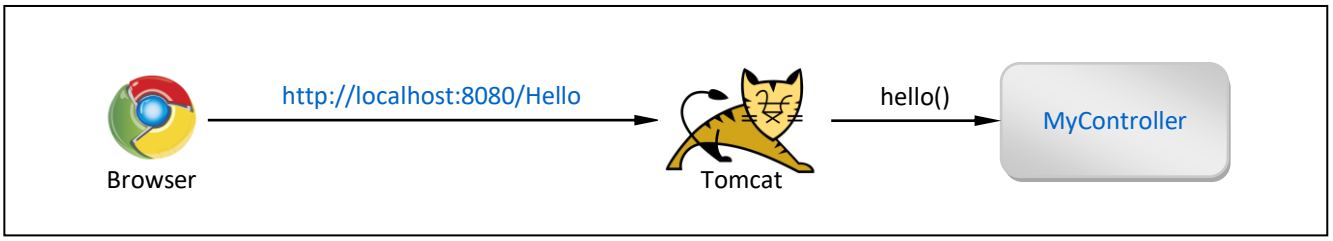
- Following tutorials explain different Controller Annotations.
- Bu using Controller Annotations you are telling Spring that Class contains Endpoints (Methods called for specific URLs).
- Spring needs to collect these Methods to properly configure Application Server (like Tomcat).
- Application Server needs to know for which URLs which Method needs to be called.

2.1.1 @Controller

Info

- @Controller Annotation tells Spring that Class is Controller. (it contains Endpoints)
- Endpoints in @Controller Class by default return a View. (HTML or Thymeleaf)
- If you want Endpoint to return actual data you can Annotated Endpoint with @ResponseBody. (as is done in this tutorial)

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @RequestMapping. Includes Tomcat HTTP Server.

Procedure

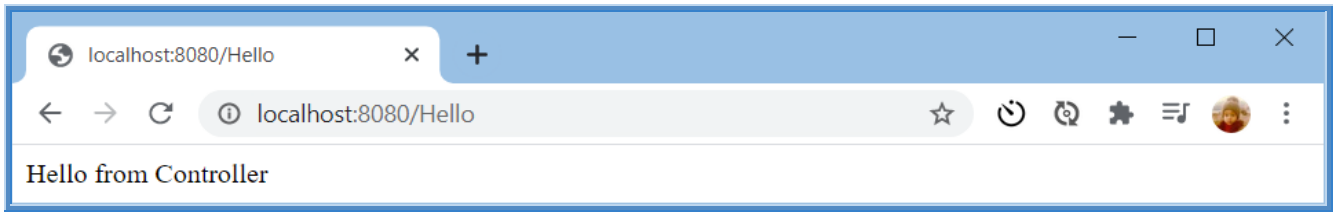
- Create Project: controller_returns_text (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyController.java

```
package com.ivoronline.controller_returns_text.controllers;

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

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello



pom.xml

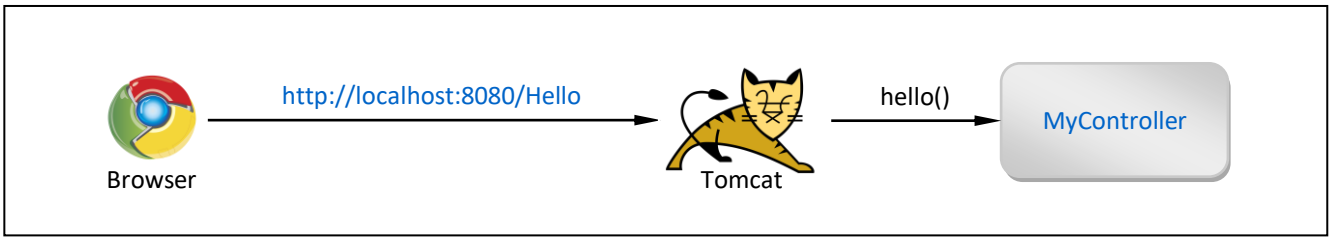
```
<dependencies>

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

</dependencies>
```

2.1.2 @RestController

Info [G] [R]

- @RestController Annotation tells Spring that Class is Controller. (it contains Endpoints)
- Endpoints in @RestController Class can only return **data**. (they can't return Views like HTML or Thymeleaf)
- Using @RestController is the same as using @Controller in combination with @ResponseBody.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server

Procedure

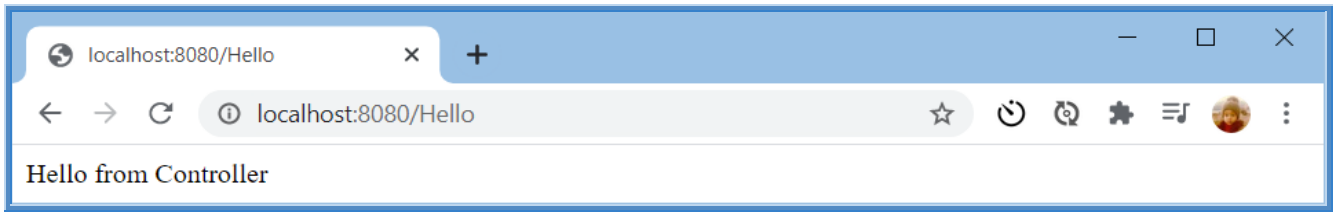
- Create Project: springboot_controller_annotation_restcontroller (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyController.java

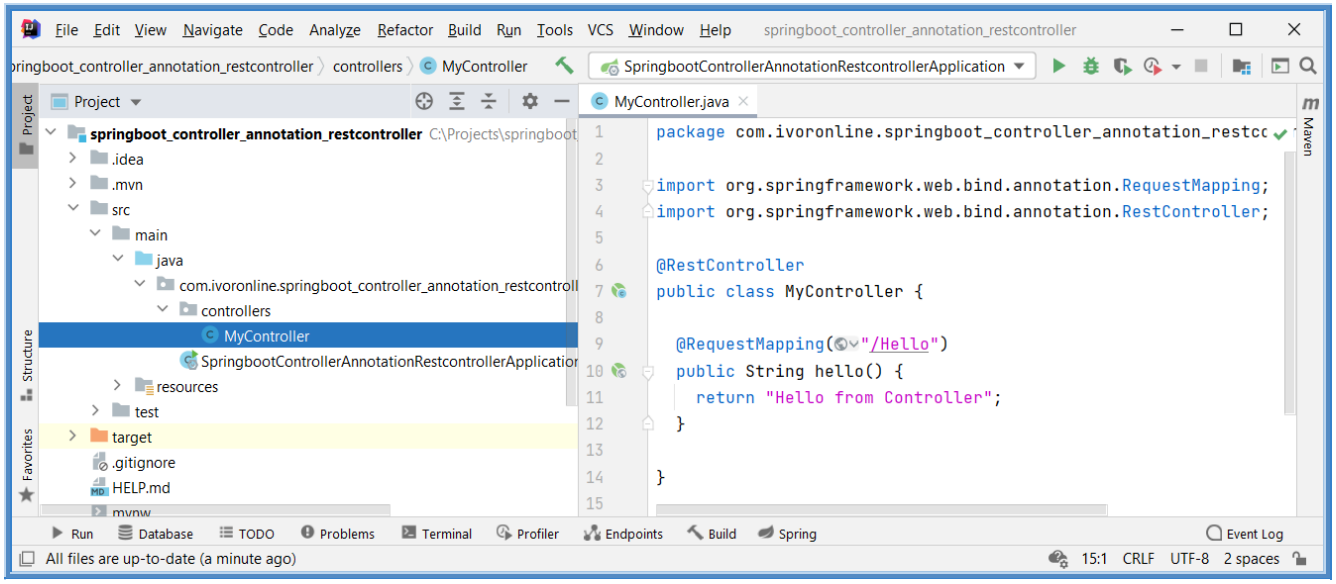
```
package com.ivoronline.springboot_controller_annotation_restcontroller.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello



Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

2.2 Endpoint

Info

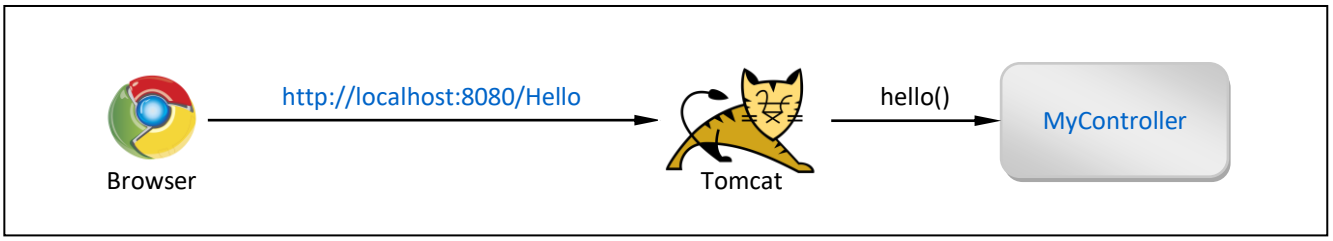
- Following tutorials explain how to work with Endpoints.
  Endpoint is Method called for specific URL.
  Spring collects Endpoints from Controllers to properly configure Application Server (like Tomcat).
  Application Server needs to know for which URL which Method needs to be called.
- Endpoint can return
  - **View** (HTML, JSP, Thymeleaf)
  - **Data** (Text, JSON)
- In the @RestController Endpoint can only return Data.
- In the @Controller Endpoint returns
  - View by default
  - Data if it is Annotated with @ResponseBody
- Endpoint Annotations define
  - for which URL Endpoint should be called ("/Hello")
  - for which type of HTTP Request Endpoint should be called (GET, POST, UPDATE, DELETE)

2.2.1 @RequestMapping

Info

- @RequestMapping tells Spring for which URL to call the Endpoint.
  Endpoint will be called for all types of HTTP Requests: GET, POST, UPDATE, DELETE.
- To specify that Endpoint should be called only for specific types of HTTP Requests you can combine Annotations: @GetMapping, @PostMapping.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @Controller, @ResponseBody, @RequestMapping, Tomcat Server

Procedure

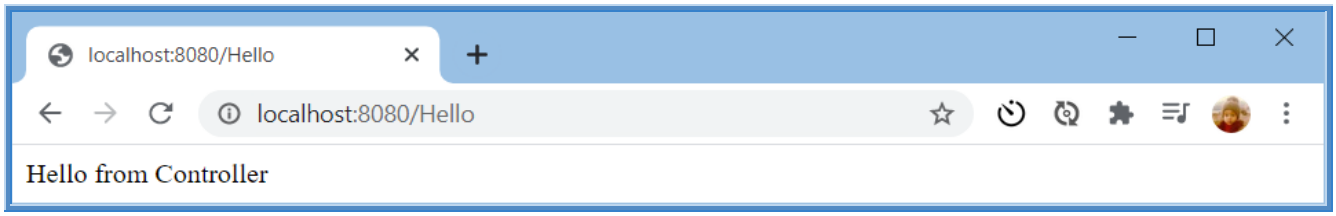
- Create Project: controller_returns_text (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyController.java

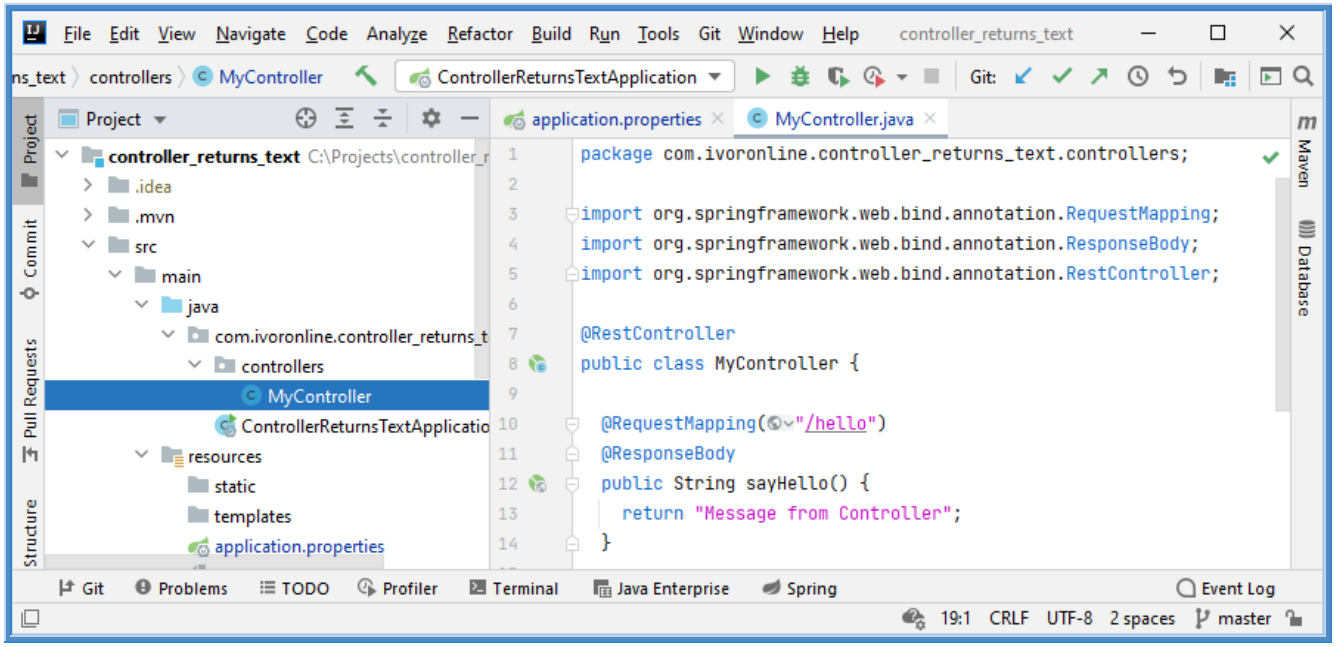
```
package com.ivoronline.controller_returns_text.controllers;

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

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello



Application Structure



pom.xml

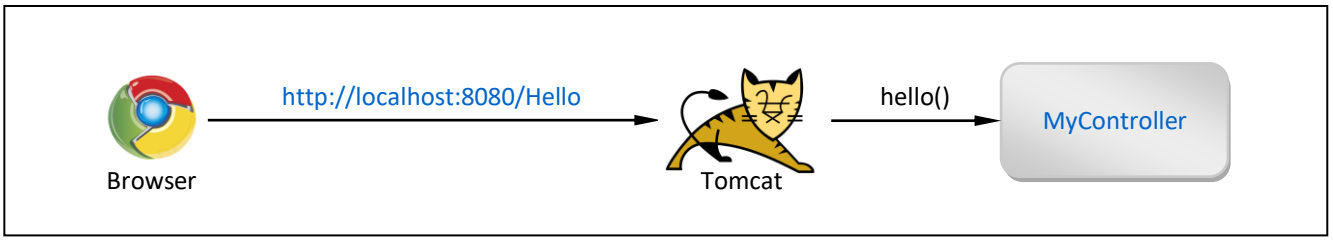
```
<dependencies>

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

</dependencies>
```

2.2.2 @GetMapping

Info [G]

- @GetMapping Annotation tells Spring for which URL to call this Endpoint.
  But Endpoint will be called only for HTTP **GET** Requests.
- You can combine it with other Annotations of this type.
  For instance if you also use @PostMapping on the same Endpoint, it will get called for both HTTP **GET** and **POST** Requests.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @Controller, @ResponseBody, @GetMapping, Tomcat Server

Procedure

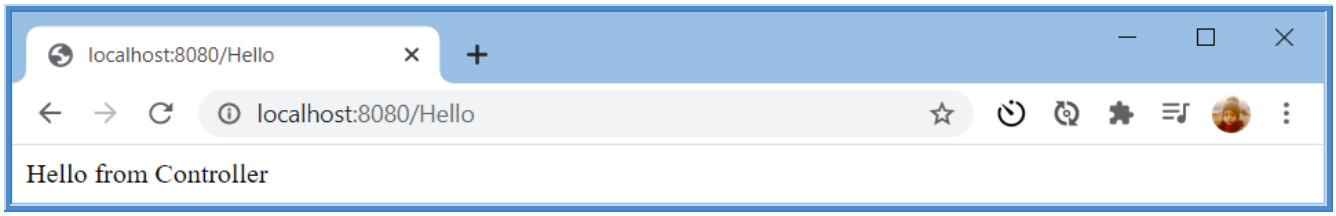
- Create Project: springboot_endpoint_annotation_getmapping (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_endpoint_annotation_getmapping.controllers;

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

@Controller
public class MyController {

  @ResponseBody
  @GetMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello



pom.xml

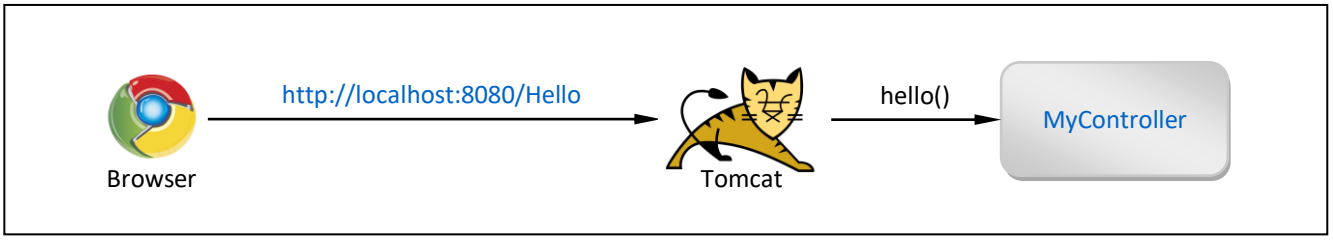
```
<dependencies>

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

</dependencies>
```

2.2.3 Return - Data - Text

Info [G] [R]

- This tutorial shows how to use `@ResponseBody` to return Text from the Controller.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RequestMapping, Tomcat

Procedure

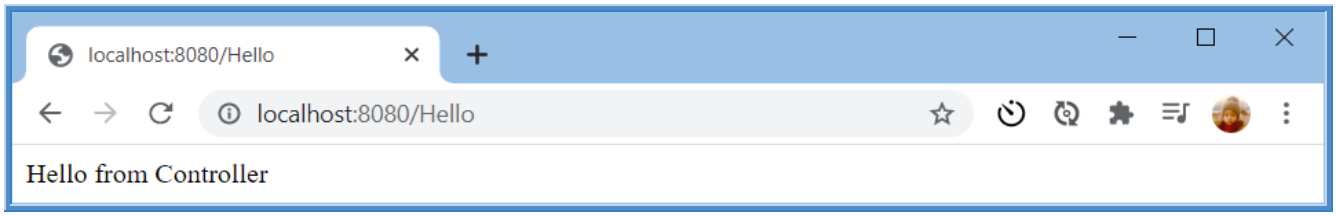
- Create Project: controller_returns_text (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyController.java

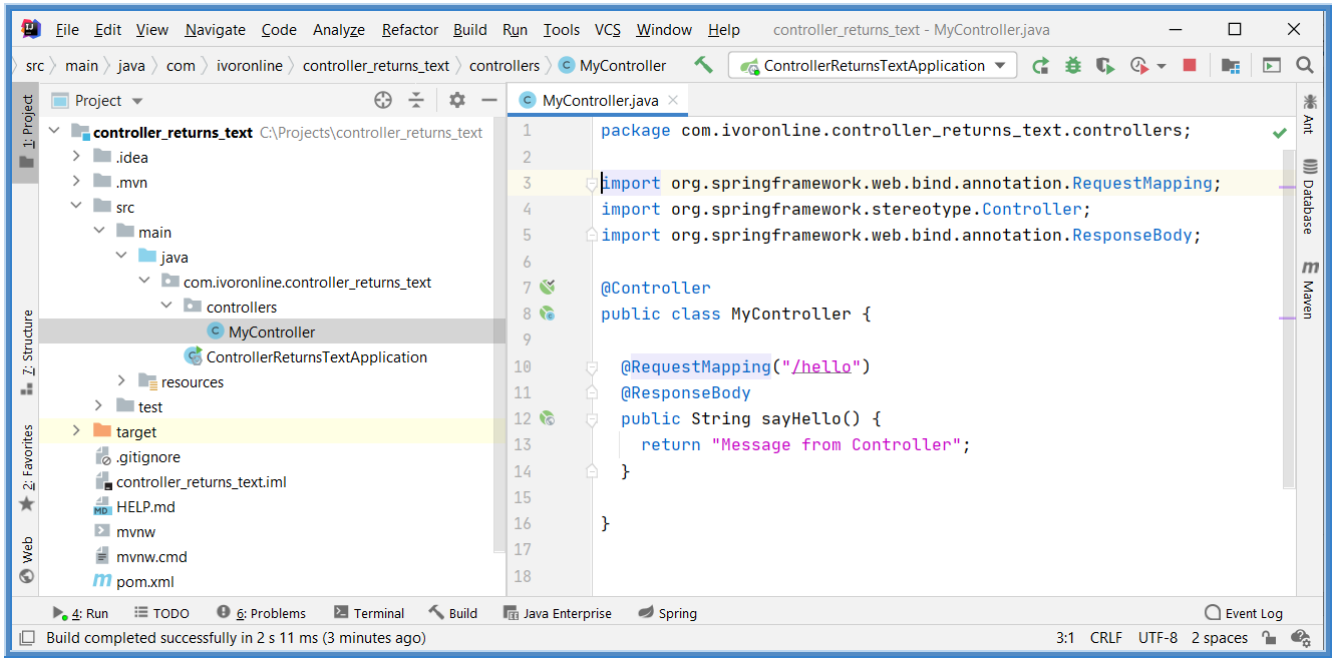
```
package com.ivoronline.controller_returns_text.controllers;

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

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello



Application Structure



pom.xml

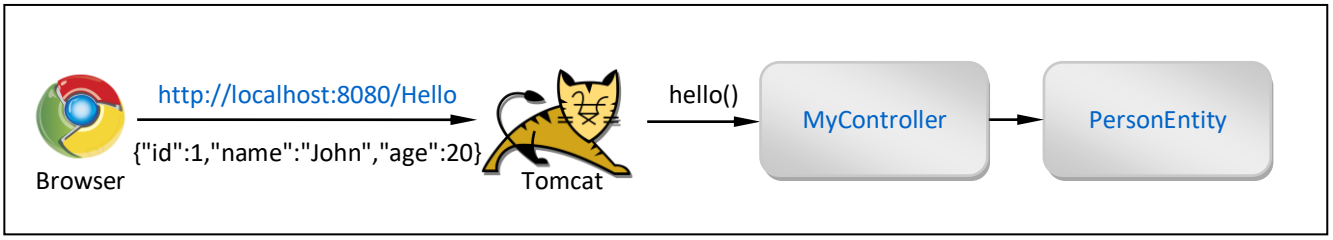
```
<dependencies>

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

</dependencies>
```

2.2.4 Return - Data - JSON

Info **[G] [R]**

- This tutorial shows how to use `@ResponseBody` to return instance of `PersonEntity` from the Controller as JSON String.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat HTTP Server.

Procedure

- Create Project: controller_returns_text_json (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
  - Create Interface: PersonEntity.java (inside package entities)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

PersonEntity.java

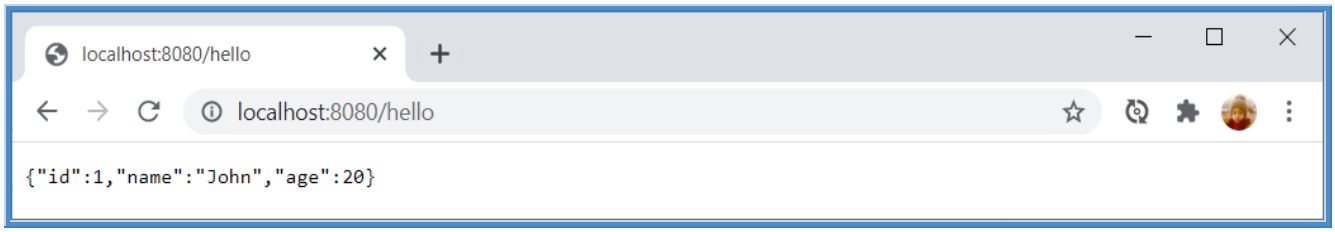
```
package com.ivoronline.springboot.controller_returns_text_json.entities;

public class PersonEntity {
  public Integer id;
  public String  name;
  public Integer age;
}
```

MyController.java

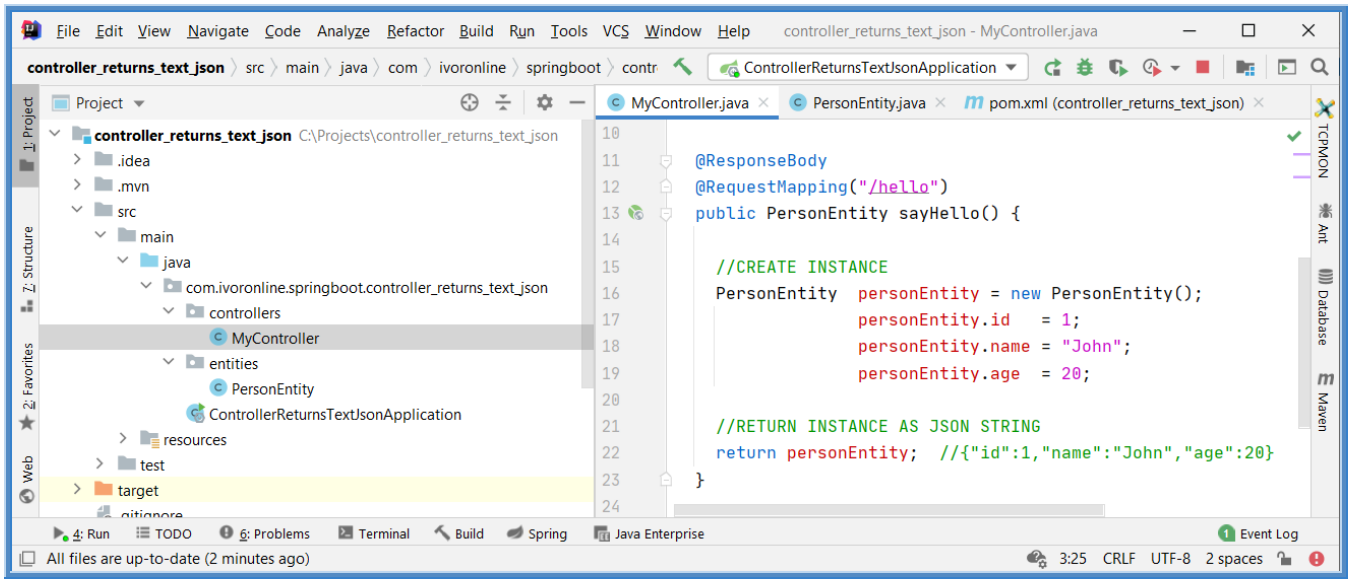
```
package com.ivoronline.springboot.controller_returns_text_json.controllers;

import com.ivoronline.springboot.controller_returns_text_json.entities.PersonEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/Hello")
  public PersonEntity hello() {

    //CREATE INSTANCE
    PersonEntity  personEntity      = new PersonEntity();
                  personEntity.id   = 1;
                  personEntity.name = "John";
                  personEntity.age  = 20;

    //RETURN INSTANCE AS JSON STRING
    return personEntity;  //{"id":1,"name":"John","age":20}
  }

}
```

Results

http://localhost:8080/Hello



Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

2.3 Entity

Info

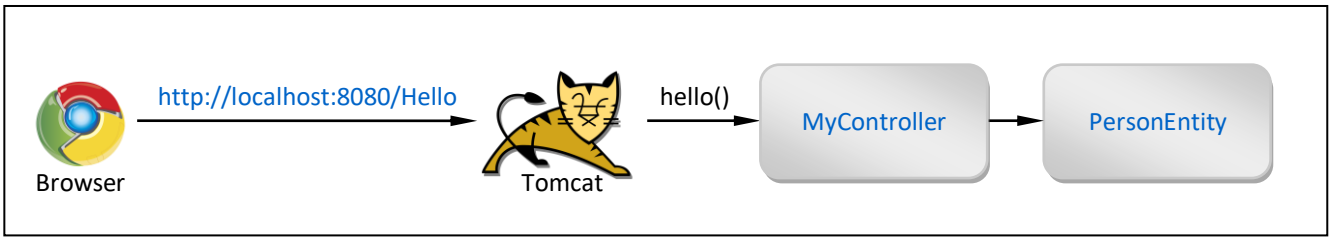
- Following tutorials show how to work with Entities.

2.3.1 Properties - Private - Getters & Constructor

Info **[G]**

- This tutorial shows how to create Entity that has **private** Properties but uses **Constructor** instead of **Setters**.
- You can combine Constructor and Setters where
  - **Constructor** is used to set **Required** or **Immutable** Properties
  - **Setters** are used to set **Optional** or **Mutable** Properties

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RequestMapping, Tomcat Server

Procedure

- Create Project: springboot_entity_constructor (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
  - Create Class: PersonEntity.java (inside package entities)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

PersonEntity.java

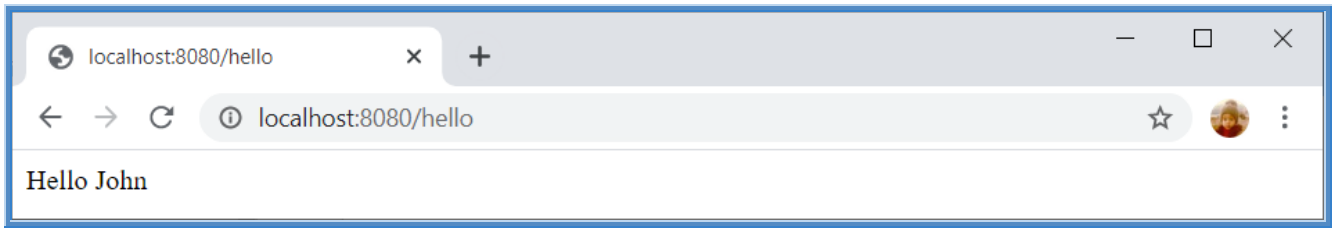
```
package com.ivoronline.springboot_entity_constructor.entities;

public class PersonEntity {

  //PROPERTIES
  private long    id;
  private String  name;
  private Integer age;

  //CONSTRUCTOR
  public PersonEntity(long Id, String name, Integer age) {
    this.id   = id;
    this.name = name;
    this.age  = age;
  }

  //GETTERS
  public long    getId  () { return id;   }
  public String  getName() { return name; }
  public Integer getAge () { return age;  }

}
```

MyController.java

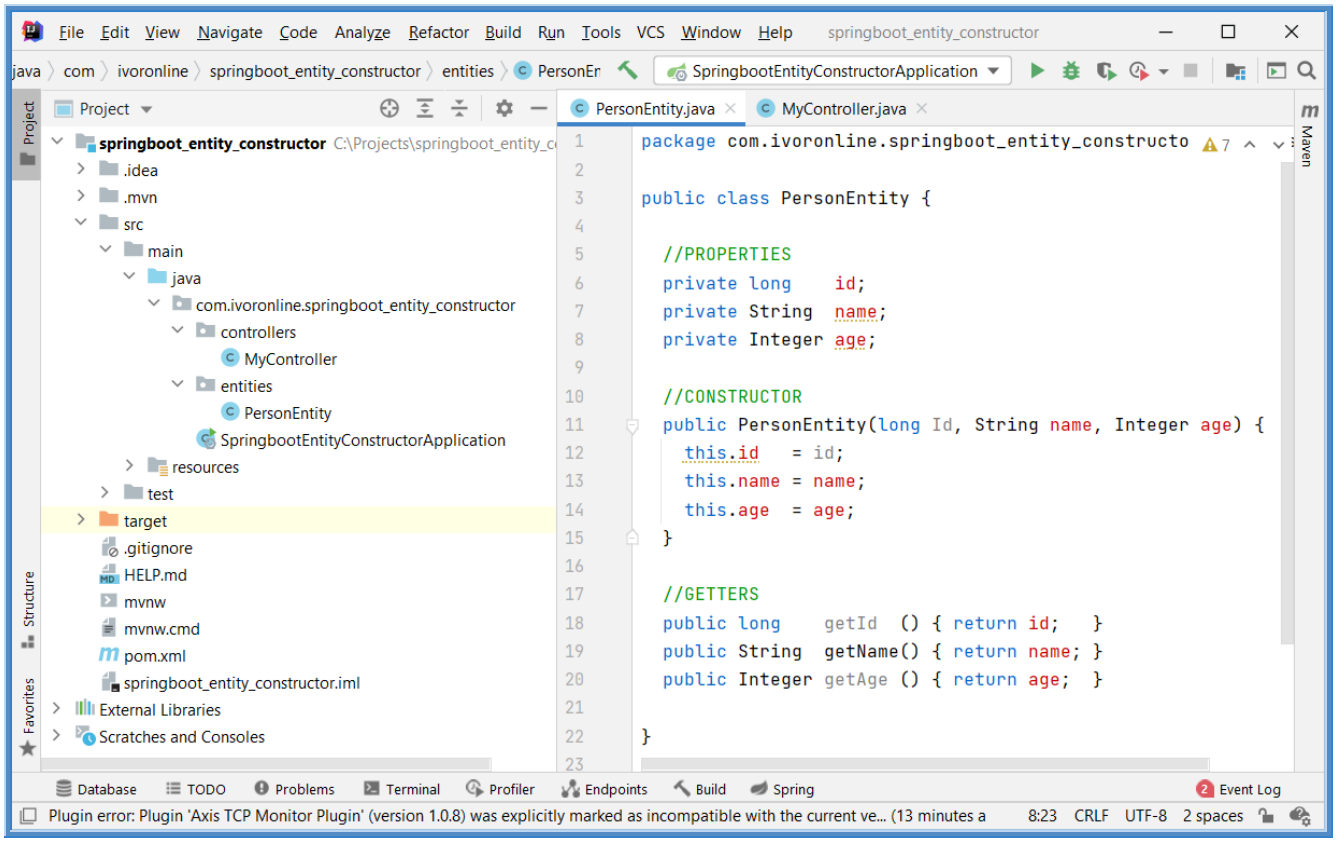
```
package com.ivoronline.springboot_entity_constructor.controllers;

import com.ivoronline.springboot_entity_constructor.entities.PersonEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    PersonEntity personEntity = new PersonEntity(1, "John", 20);
    return "Hello " + personEntity.getName();
  }

}
```

Results

http://localhost:8080/Hello



Application Structure



pom.xml

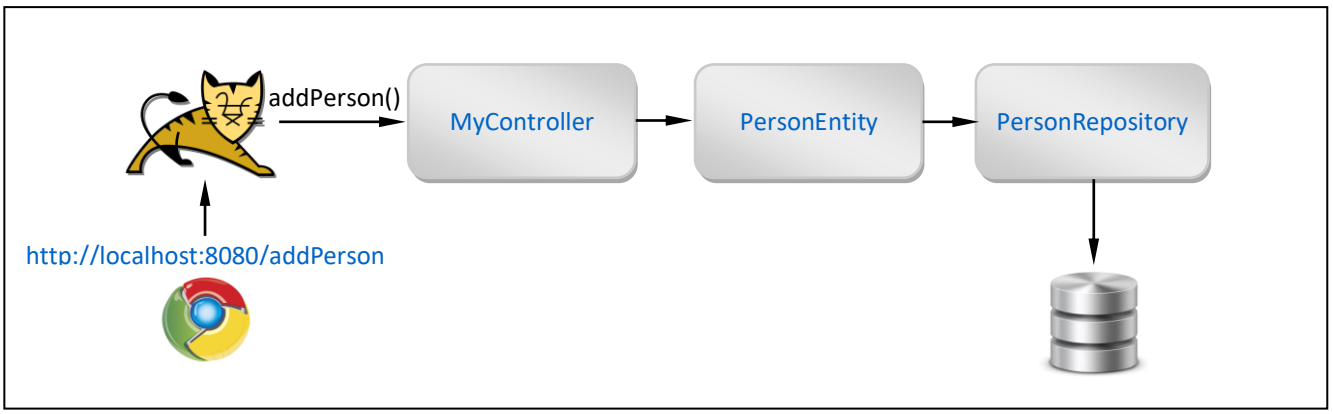
```
<dependencies>

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

</dependencies>
```

2.3.2 @Entity

Info

- In this tutorial we will use JPA's **@Entity** Annotation to indicate that instances of this Class should be stored in DB.
  Without this Hibernate will not be able to pick up this Class and do its magic with it by automatically creating Tables, Columns and then taking care od persistent its instances in selected DB.
  Classes annotated with @Entity must have Property that is declared as Primary Key by using @Id Annotation.
  As reminder: JPA API is interface that defines @Entity and Hibernate API is actual implementation that stores Entity in DB.
- For each @Entity Class Hibernate creates DB Table
  - that has the same name as the name of the Class
  - that has Columns with the same names as Class Properties
- When you include JPA Spring Boot Starter you also need to select some DB Spring Boot Starter: H2, MySQL, PostgreSQL.
  Otherwise Spring Boot will not be able to initialize JPA and your Application would not run.
  This tutorial includes H2 in-memory DB for which no additional installation and configuration is needed.
- JPA Annotations can only be used on SQL DBs (H2, MySQL, Oracle) and can't be used on NoSQL DBs (MongoDB).

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server
SQL	Spring Data JPA	Enables @Entity and @Id
SQL	H2 Database	Enables in-memory H2 DB

Procedure

- Create Project: entity_entity (add Spring Boot Starters from the table)
- Edit: application.properties (specify H2 DB name & enable H2 Web Console)
- Create Package: entities (inside main package)
  - Create Class: PersonEntity.java (inside package entities)
- Create Package: repositories (inside main package)
  - Create Interface: PersonRepository.java (inside package repositories)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

application.properties

```
# H2 DATABASE
spring.datasource.url     = jdbc:h2:mem:testdb
spring.h2.console.enabled = true
```

PersonEntity.java

```
package com.ivoronline.entity_entity.entities;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class PersonEntity {

  @Id
  public Integer id;
  public String  name;
  public Integer age;

}
```

PersonRepository.java

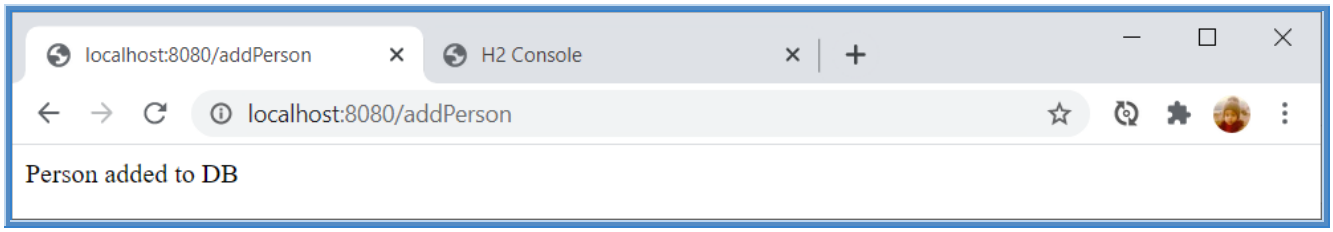
```
package com.ivoronline.entity_entity.repositories;

import com.ivoronline.entity_entity.entities.PersonEntity;
import org.springframework.data.repository.CrudRepository;

public interface PersonRepository extends CrudRepository<PersonEntity, Integer> { }
```

MyController.java

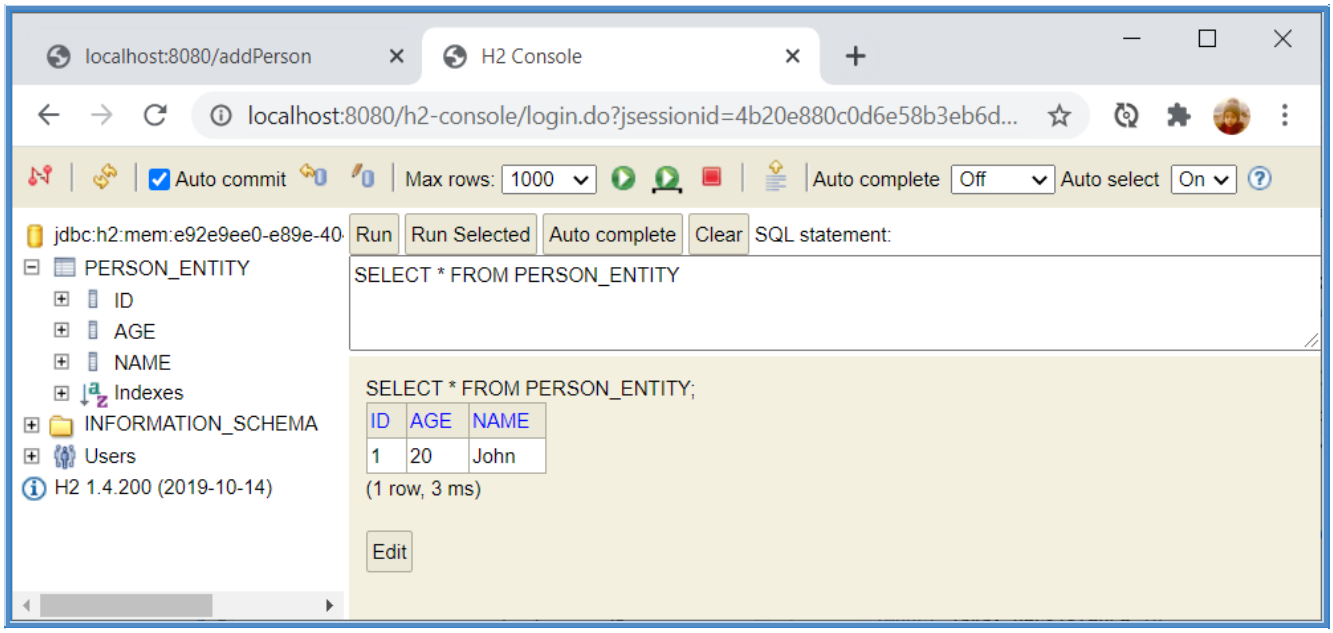
```java
package com.ivoronline.entity_entity.controllers;

import com.ivoronline.entity_entity.entities.PersonEntity;
import com.ivoronline.entity_entity.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class MyController {

  @Autowired
  PersonRepository personRepository;

  @ResponseBody
  @RequestMapping("/addPerson")
  public String addPerson() {

    //CREATE ENTITY OBJECT
    PersonEntity personEntity      = new PersonEntity();
                 personEntity.id   = 1;
                 personEntity.name = "John";
                 personEntity.age  = 20;

    //STORE ENTITY OBJECT INTO DB
    personRepository.save(personEntity);

    //RETURN SOMETHING TO BROWSER
    return personEntity.name + " was stored into DB";

  }

}
```

Results

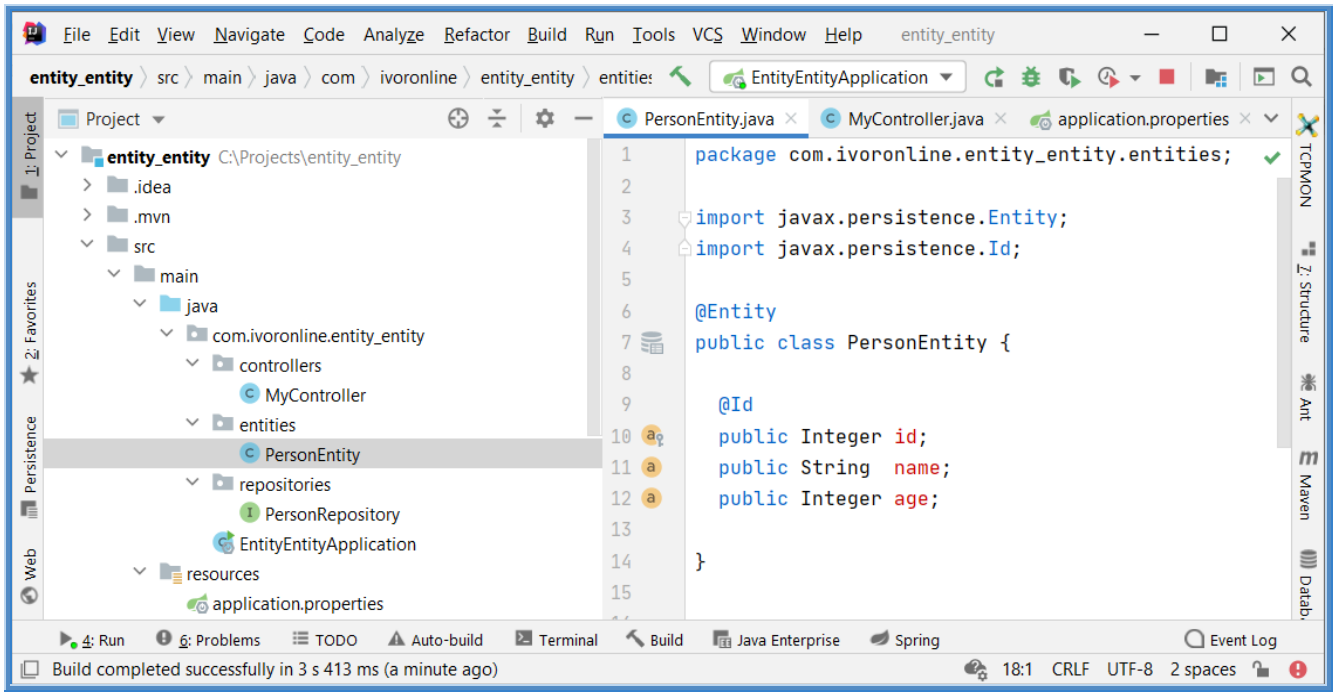
http://localhost:8080/addPerson



http://localhost:8080/h2-console - Connect - PERSON_ENTITY - Run *(H2 Console)*



Application Structure



pom.xml

```
<dependencies>

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

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

  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>

</dependencies>
```