UNIVERSITÀ
DI TORINO

Introduction to Java Servlets

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

Spring Boot 3.0

- Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM).
- Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:
  - Autoconfiguration
  - An opinionated approach to configuration
  - The ability to create standalone applications
- These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup

https://www.ibm.com/topics/java-spring-boot

- it enables developers to create modular applications consisting of loosely coupled components that are ideal for microservices and distributed network applications
- it is idea to build REST servers using a route based approach similar to Express

- Spring Boot is a fantastic framework for building RESTful web services
- It simplifies the process of creating RESTful APIs by providing a range of features and tools
  - that streamline development

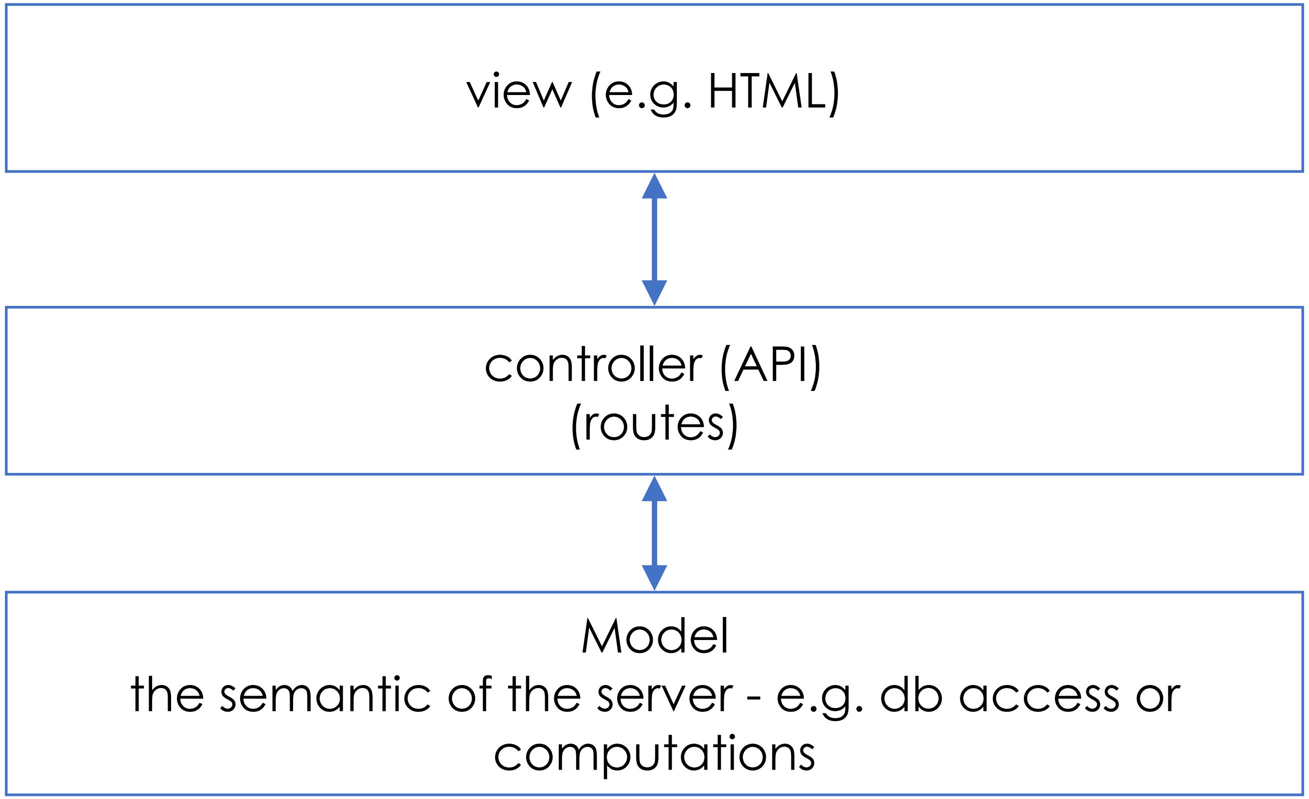
Dependency Management

- Spring Boot simplifies dependency management
  - by offering a wide array of pre-configured dependencies
  - You can include these dependencies in your project,
    - and Spring Boot will automatically configure them, reducing the need for extensive configuration.
    - For RESTful services, you can easily include dependencies for **Spring Web**,
      - which provides the necessary components for building RESTful endpoints

SpringBoot uses Java Annotations

- Java annotations are a form of metadata that can be added to Java source code.
  - They can be used to provide information about the program to the compiler,
    - to generate code at compile time, or to be processed at runtime.
- Java annotations are defined using the **@ symbol** followed by the name of the annotation.
  - Annotations can be applied to classes, methods, variables, parameters, and packages.
- Here are some examples of Java annotations:
  - **@Override**: This annotation is used to mark a method as an override of a method in a superclass or implemented interface.
  - **@Deprecated**: This annotation is used to mark a method or class as **obsolete.**
  - **@SuppressWarnings**: This annotation is used to suppress compiler warnings.
  - **@Autowired**: This annotation is used to tell Spring Boot to automatically inject a dependency into a field or **constructor.**
  - **@RestController**: This annotation is used to mark a class as a Spring Boot REST controller.
- Java annotations can be used for a variety of purposes, including:
  - Code generation:
    - Java annotations can be used to generate code at compile time. This can be useful for tasks such as generating boilerplate code or generating code from templates.
  - Runtime processing:
    - Some Java annotations can be processed at runtime. This can be useful for tasks such as validation, security, and logging.
  - Documentation:
  - Java annotations can be used to generate documentation for your code. This can be useful for improving the readability and maintainability of your code.

MVC Based



view (e.g. HTML)

controller (API)
(routes)

Model
the semantic of the server - e.g. db access or computations

Controller Classes

- Spring Boot naturally follows the MVC strategy
  - You create controller classes to define RESTful endpoints
  - These classes are annotated with @RestController to indicate that they will handle incoming HTTP requests
    - and return responses as JSON
  - You can also use annotations like **@GetMapping**, **@PostMapping**, @PutMapping, and @DeleteMapping
    - to map HTTP methods to specific controller methods
      - i.e. to create a get route you use

        ```
        @GetMapping('/')
        public Character getFirstCharacter(){
                return new Character("pip", "goofy", 1990);
                }
        ```

        - note: although we return a type, it will be returned as a JSON structure to the client

Request Mapping

- You can define request mappings in your controller methods using **@RequestMapping** or other HTTP method-specific annotations.
  - These mappings specify the URL path and HTTP method that a particular method should respond to.
    - For example, you can define a method that handles GET requests at **`/api/resource`**.

Data Transfer Objects (DTOs)

- To transfer data between your RESTful service and the client,
  - you can create Data Transfer Objects (DTOs)
  - These are simple Java classes that represent the data that you want to send or receive.
  - Spring Boot can automatically serialise these objects to JSON making it easy to work with data in your API

Response Handling

- Spring Boot simplifies the handling of responses.
- When a controller method returns an object,
  - Spring Boot will automatically convert it to the appropriate format (usually JSON)
    - and send it as an HTTP response.
  - You can use **`ResponseEntity`** to have more control over the response, including status codes and headers

Exception Handling

- Spring Boot provides mechanisms to handle exceptions gracefully
- You can use exception handling annotations like **`@ExceptionHandler`** to define how your API should respond to specific exceptions
  - This is particularly useful for returning meaningful error messages to clients.

Validation

- Input validation is essential in any web service.
- Spring Boot allows you to use validation annotations, such as **@Valid**, in your DTOs
  - to ensure that incoming data adheres to defined constraints.
  - If validation fails, appropriate error responses are automatically generated.

Security

- Spring Boot offers built-in security features for RESTful services.
- You can secure your APIs using **Spring Security**,
  - which allows you to implement authentication and authorisation, and control access to your endpoints

Documentation

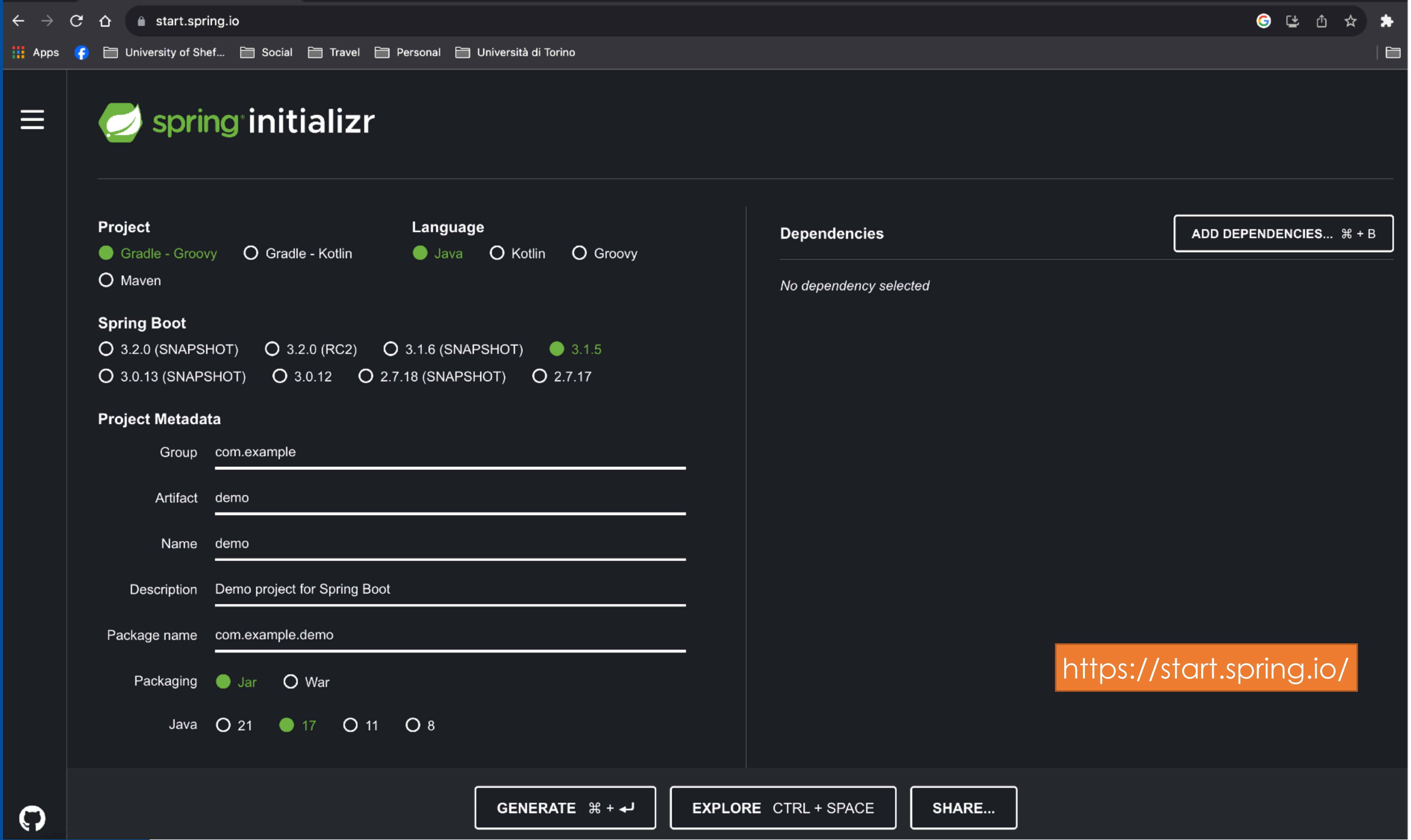
- Documentation is crucial for your RESTful API. Spring Boot integrates well with tools like **Swagger** to generate API documentation automatically
- This documentation can be accessed via a user-friendly UI,
  - making it easier for both developers and consumers to understand your API

Multi-Threaded

- By default, when you create a Spring Boot application, it uses the embedded Tomcat server.
- Tomcat is multi-threaded and capable of handling multiple concurrent requests.
- It uses a thread pool to process incoming requests, which means it can serve multiple clients simultaneously

An Example

© Prof. Fabio Cravegna, Università di Torino



Project

● Gradle - Groovy ○ Gradle - Kotlin
○ Maven

Language

● Java ○ Kotlin ○ Groovy

Spring Boot

○ 3.2.0 (SNAPSHOT) ○ 3.2.0 (RC2) ○ 3.1.6 (SNAPSHOT) ● 3.1.5
○ 3.0.13 (SNAPSHOT) ○ 3.0.12 ○ 2.7.18 (SNAPSHOT) ○ 2.7.17

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	● Jar ○ War
Java	○ 21 ● 17 ○ 11 ○ 8

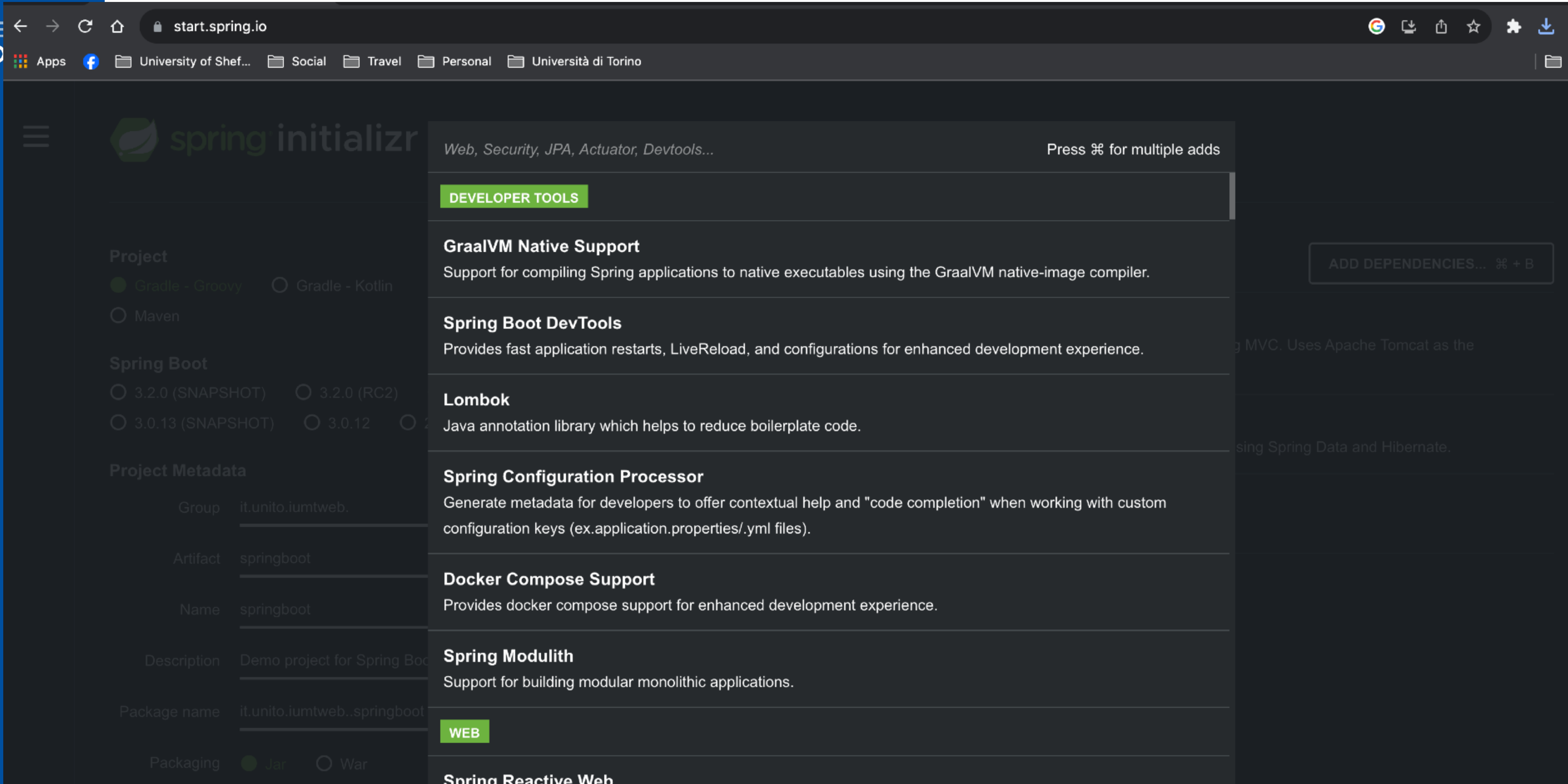
Dependencies

ADD DEPENDENCIES... ⌘ + B

No dependency selected

https://start.spring.io/

Adding Dependencies



Group: it.unito.iumtweb.

Artifact: springboot

Name: springboot

Description: Demo project for Spring Boo

Package name: it.unito.iumtweb..springboot

DEVELOPER TOOLS

GraalVM Native Support
Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).

Docker Compose Support
Provides docker compose support for enhanced development experience.

Spring Modulith
Support for building modular monolithic applications.

WEB

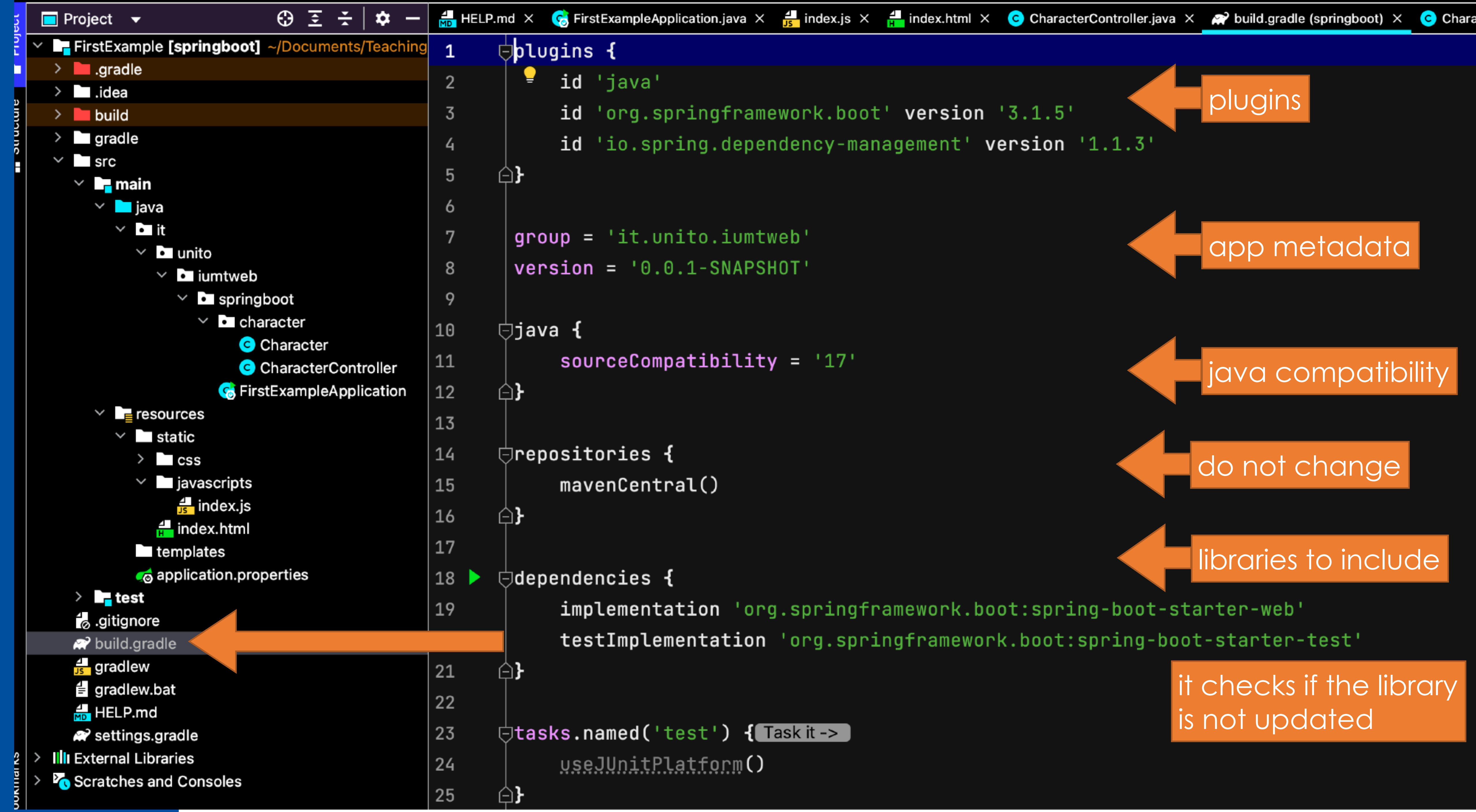
Spring Reactive Web

Gradle

- Gradle is an open-source build automation tool that is designed to automate the build process of software projects
- It is a versatile and powerful tool that helps you manage dependencies, compile source code, run tests, and package your application into a distributable format.
- Gradle is used in a wide range of software development projects, from small applications to large-scale enterprise systems.

Key Features of Gradle

- Declarative Build Scripts:
  - Gradle uses Groovy or Kotlin as the scripting language for build configuration.
    - You define your build tasks in a declarative manner, specifying what you want to achieve, and Gradle takes care of the how.
- Dependency Management:
  - Gradle excels in dependency management. It can automatically download and manage project dependencies, making it easy to work with external libraries and frameworks.
- Plugin System:
  - Gradle's plugin system allows you to extend its functionality. You can use existing plugins or create custom ones to tailor your build process to your specific needs.
- Multi-Project Builds:
  - Gradle supports multi-project builds, where you can manage multiple subprojects within a single build. This is useful for organizing and building complex applications.
- Incremental Builds:
  - Gradle is efficient and can perform incremental builds. It only recompiles and retests the parts of the codebase that have changed since the last build, saving time and resources.
- IDE Integration:
  - Gradle integrates well with popular integrated development environments (IDEs) like IntelliJ IDEA and Eclipse, making it easy to work on your projects within your preferred IDE.



```
FirstExample [springboot] ~/Documents/Teaching
  .gradle
  .idea
  build
  gradle
  src
    main
      java
        it
          unito
            iumtweb
              springboot
                character
                  Character
                  CharacterController
                FirstExampleApplication
      resources
        static
          css
          javascripts
            index.js
          index.html
        templates
        application.properties
    test
  .gitignore
  build.gradle
  gradlew
  gradlew.bat
  HELP.md
  settings.gradle
External Libraries
Scratches and Consoles
```

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.1.5'
    id 'io.spring.dependency-management' version '1.1.3'
}

group = 'it.unito.iumtweb'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

tasks.named('test') { Task it ->
    useJUnitPlatform()
}
```

plugins

app metadata

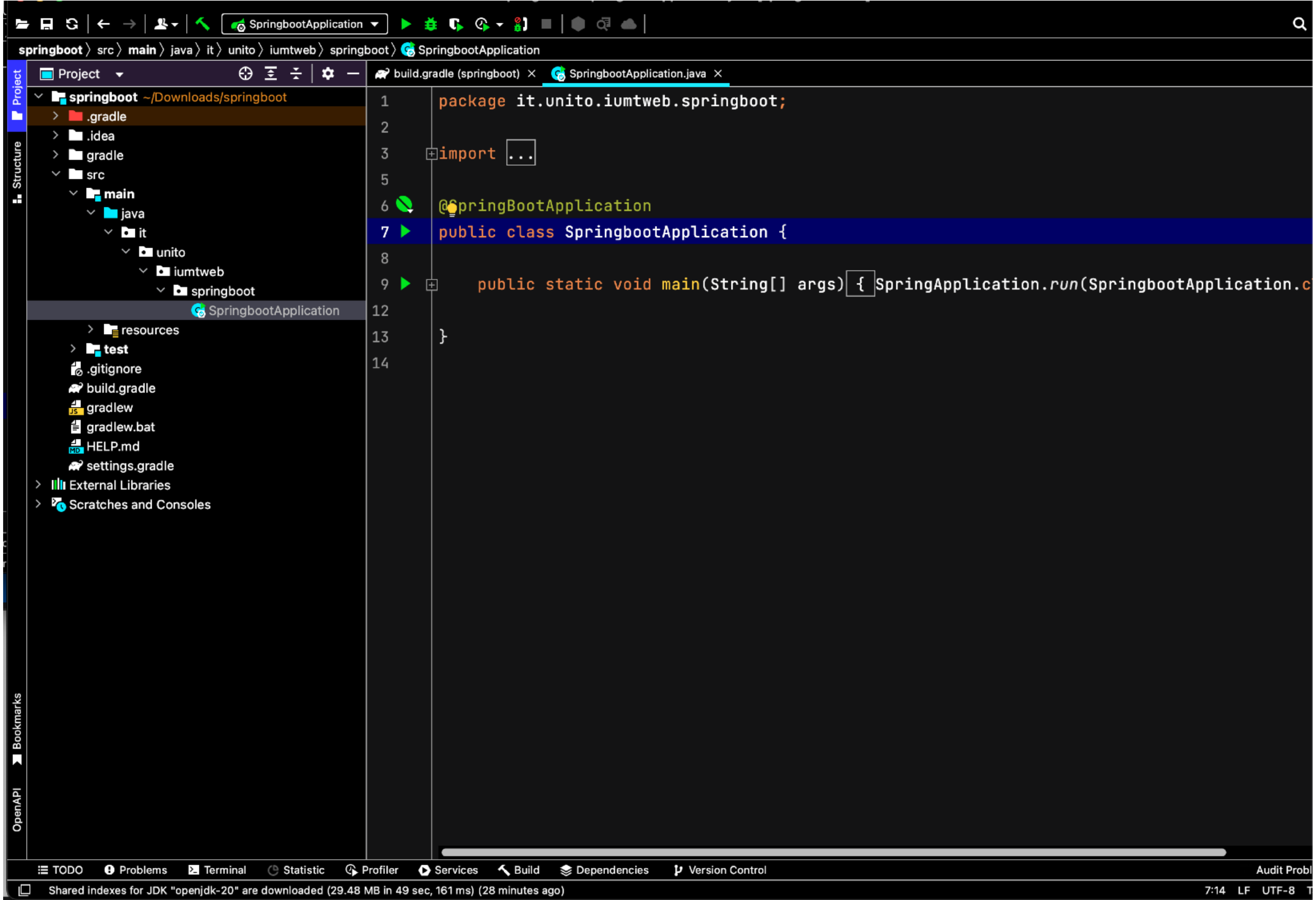
java compatibility

do not change

libraries to include

it checks if the library is not updated

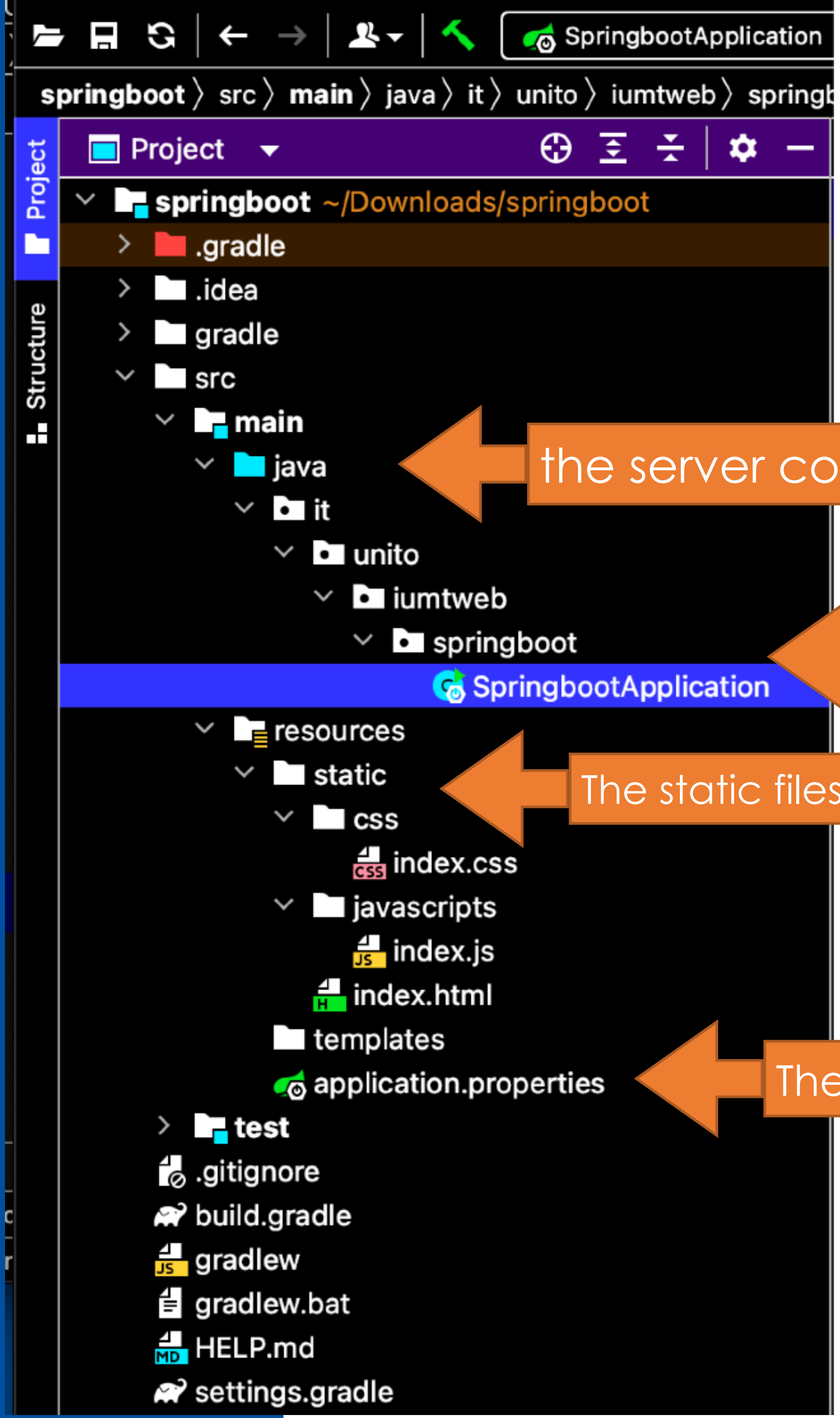
UNIVERSITÀ DI TORINO



```
package it.unito.iumtweb.springboot;

import ...

@SpringBootApplication
public class SpringbootApplication {

    public static void main(String[] args) { SpringApplication.run(SpringbootApplication.c

}

```

Shared indexes for JDK "openjdk-20" are downloaded (29.48 MB in 49 sec, 161 ms) (28 minutes ago)

Code Organisation



```
SpringbootApplication
springboot 〉 src 〉 main 〉 java 〉 it 〉 unito 〉 iumtweb 〉 springb

Project
springboot ~/Downloads/springboot
  .gradle
  .idea
  gradle
  src
    main
      java
        it
          unito
            iumtweb
              springboot
                SpringbootApplication
      resources
        static
          css
            index.css
          javascripts
            index.js
          index.html
        templates
        application.properties
    test
  .gitignore
  build.gradle
  gradlew
  gradlew.bat
  HELP.md
  settings.gradle
```

the server code

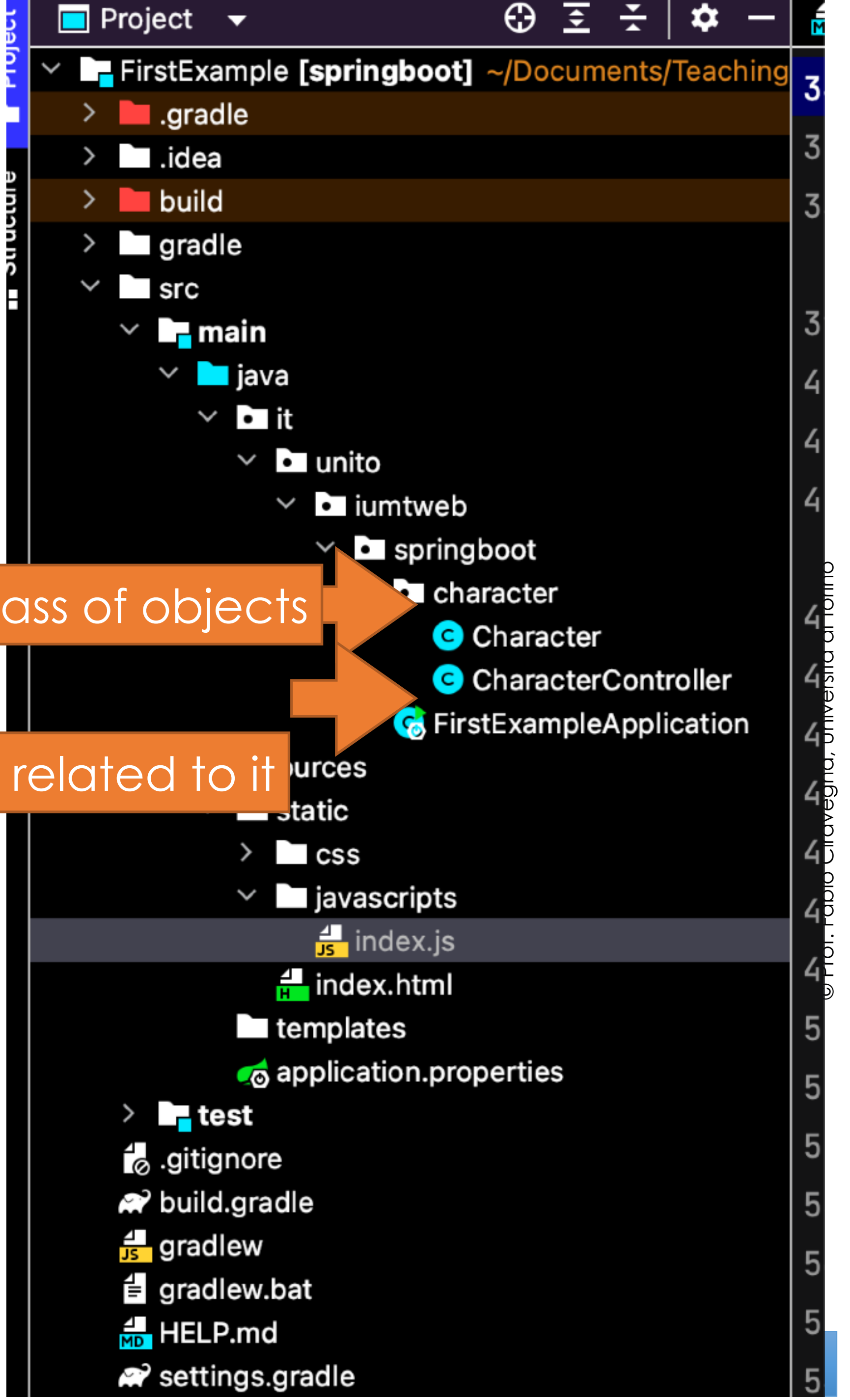
The RESTFUL server code

The static files (html, js, css, etc.) equivalent to the public folder in express

The server properties (e.g. port, where the database servers are, etc.)

Defining routes

- In order to define routes, it is necessary to define a controller.
- Typically we will have a controller connected to a specific logic
  - e.g. getting data from a database



```
Project
FirstExample [springboot] ~/Documents/Teaching
  .gradle
  .idea
  build
  gradle
  src
    main
      java
        it
          unito
            iumtweb
              springboot
                character
                  Character
                  CharacterController
                FirstExampleApplication
      urces
        static
          css
          javascripts
            index.js
          index.html
        templates
        application.properties
    test
  .gitignore
  build.gradle
  gradlew
  gradlew.bat
  HELP.md
  settings.gradle
```

a class of objects

the controller related to it

© Prof. Fabio Ciravegna, Università di Torino

UNIVERSITÀ
DI TORINO

A Note on JSON in Java

© Prof. Fabio Cravegna, Università di Torino

GSON

A Google library for JSON in Java

- Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of
  - Download the gson library in order to use it (it is not in the standard java distribution)

http://code.google.com/p/google-gson/

nowadays it would be best to use Jakarta.Jackson but it is irrelevant here

x

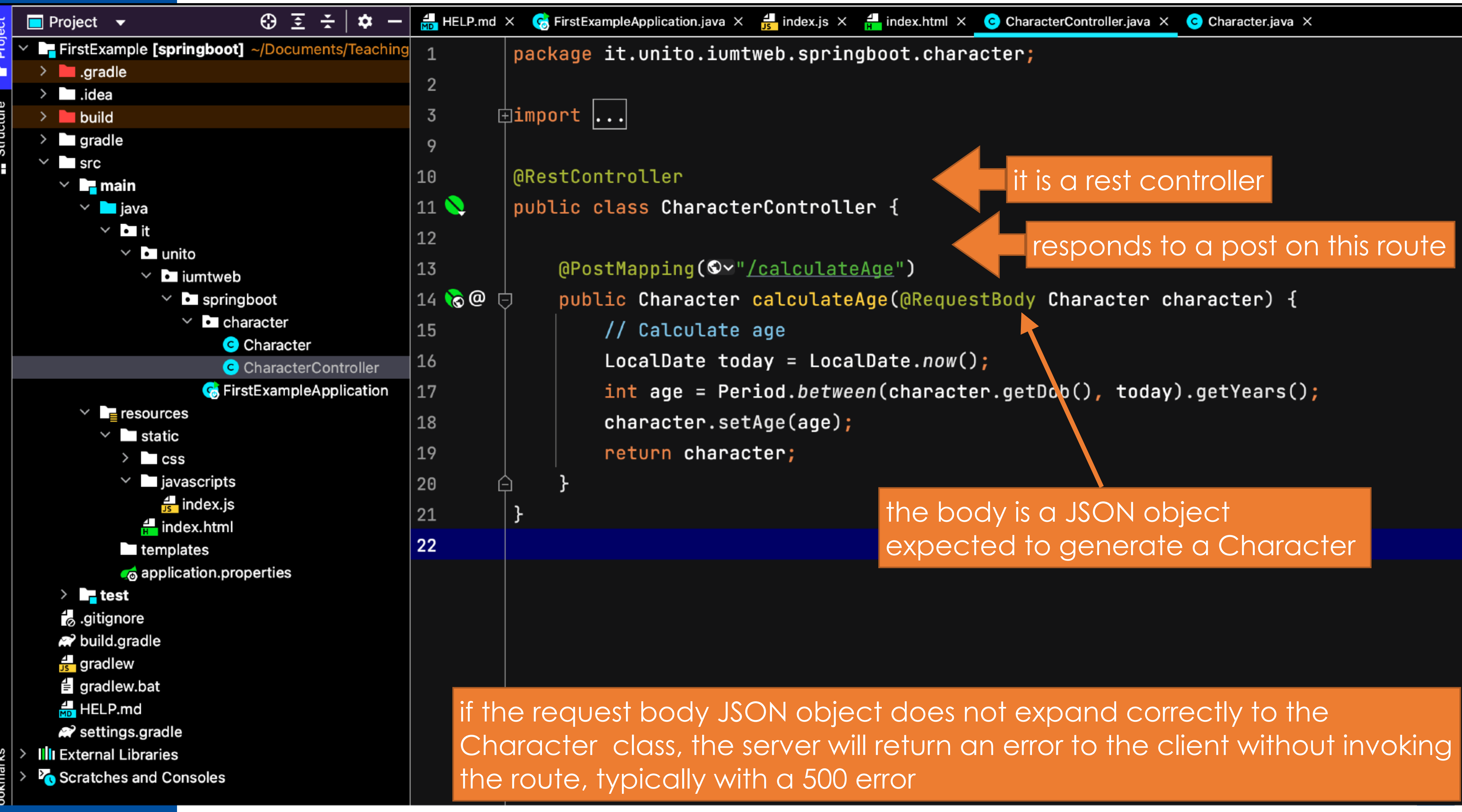
- Serialisation:

```
/* create Gson object */
Gson gson = new Gson();
/* create the object to serialise (any Java object)*/
class BagOfPrimitives {
  private int value1 = 1;
  private String value2 = "abc";
  private transient int value3 = 3;
  BagOfPrimitives() {
    // no-args constructor
  }
}
BagOfPrimitives obj = new BagOfPrimitives();
String json = gson.toJson(obj);
```

Deserialisation

```
BagOfPrimitives obj2 =
   gson.fromJson(jsonString,
                        BagOfPrimitives.class);
```

Expected Object class



```java
package it.unito.iumtweb.springboot.character;

import ...

@RestController
public class CharacterController {

    @PostMapping("/calculateAge")
    public Character calculateAge(@RequestBody Character character) {
        // Calculate age
        LocalDate today = LocalDate.now();
        int age = Period.between(character.getDob(), today).getYears();
        character.setAge(age);
        return character;
    }
}
```

it is a rest controller

responds to a post on this route

the body is a JSON object expected to generate a Character

if the request body JSON object does not expand correctly to the Character class, the server will return an error to the client without invoking the route, typically with a 500 error

but how is the controller connected to the application?

© Prof. Fabio Cravegna, Università di Torino

Spring's component scanning mechanism

- Spring's component scanning mechanism is a feature of the Spring Framework
  - that allows you to automatically detect and register Spring beans (components) in your application context
    - without explicitly defining them in the configuration
    - This mechanism simplifies the configuration of your Spring application by reducing the need for manual bean declarations.
- Here's an overview of how it works:
  - Annotations
    - Component scanning relies on annotations like **`@Component, @Service, @Repository,`** and **`@Controller`**
      - These annotations are used to mark Java classes as Spring-managed components.
  - Base Package:
    - You define a base package for component scanning. Spring scans all classes in this package and its sub-packages for classes annotated with these component annotations.
  - Auto-Registration
    - When Spring identifies a class with one of these annotations, it automatically registers it as a bean in the Spring application context.

UNIVERSITÀ DI TORINO

Commonly used component annotations

- @Component:
  - The most generic annotation, used to mark a class as a Spring-managed component (typically not used directly)
- @Service:
  - Used to indicate that a class is a service component, often used for business logic or service layers
    - typically part of the Model in MVC - generally providing the Business Logic)
- @Repository:
  - Used to mark a class as a repository component, typically for data access or persistence
    - again typically part of the Model in MVC - providing the Data Storage (e.g. database)
- @Controller:
  - Used to identify a class as a controller component in a Spring MVC web application
    - this is the Controller in MVC

To define the base dir of the component scan

```
@SpringBootApplication
@ComponentScan(basePackages = "com.example.myapp")
public class MyAppApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyAppApplication.class, args);
    }
}
```

By default it will get the main package (so most of the times you will not need to define this)

Annotations

- **@SpringBootApplication:**
  - Annotating the main application class with @SpringBootApplication indicates that it's the entry point of the Spring Boot application.
  - It combines several annotations, including @Configuration, @EnableAutoConfiguration, and @ComponentScan, providing a convenient way to configure and bootstrap the application.
- **@Controller:**
  - Used to mark a class as a Spring MVC controller in a web application.
  - Spring Boot will automatically discover and register @Controller components, making them accessible through web requests.

do not use. Use RestController

- **@RestController:**
  - A specialization of @Controller, @RestController is used to define RESTful web services.
  - It combines @Controller and @ResponseBody, indicating that the return values of methods should be serialized and sent as HTTP response data.
- **@Service:**
  - Annotating a class with @Service indicates that it's a service component in your application.
  - Typically used for business logic or service layers.
- **@Repository:**
  - Used to annotate data access classes or repositories.
  - Spring Boot treats classes annotated with @Repository as Spring beans and provides data access-related functionality.
- **@Component:**
  - The most generic annotation for marking a class as a Spring-managed component.
  - It can be used for any class you want to register as a bean.

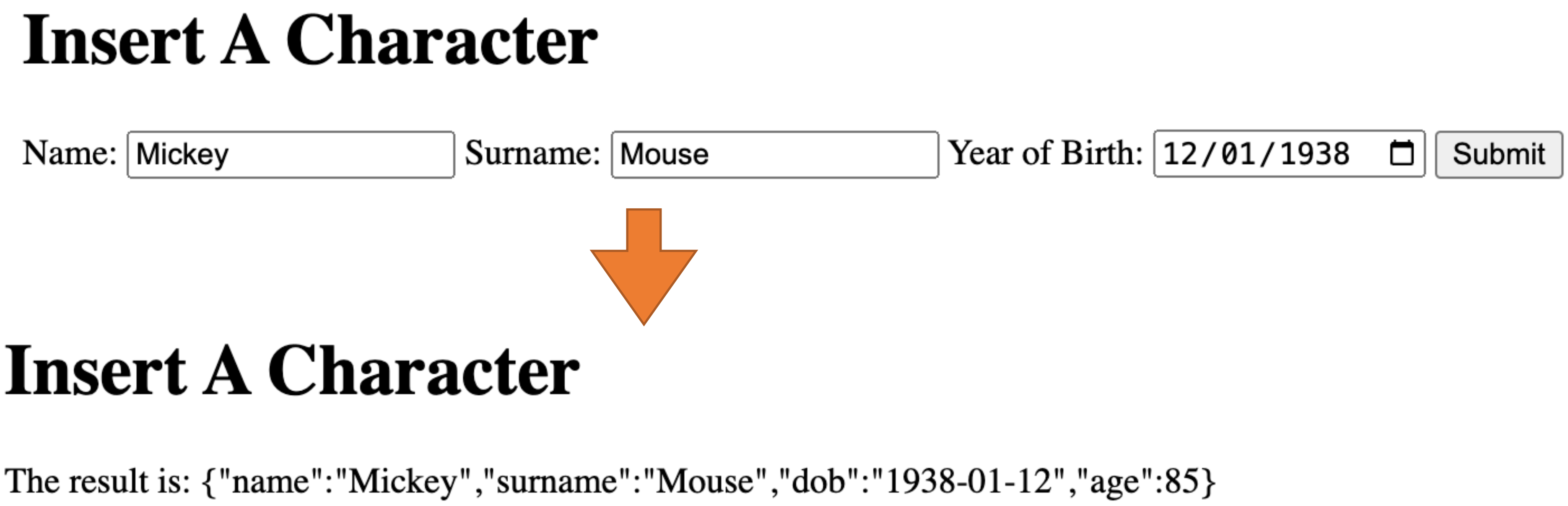
- **@Autowired:**
  - Used for automatic dependency injection.
  - When applied to fields, methods, or constructors, Spring Boot injects the required dependencies into these components.
- **@Transactional:**
  - Used to mark a method as transactional.
  - Spring Boot manages transactions for methods annotated with @Transactional, ensuring that they execute within a transactional context.
- **@EnableAutoConfiguration**:
  - Automatically configures Spring Boot based on the project's dependencies and settings.
  - You can use it to enable or disable specific auto-configurations.
- **@Entity:**
  - Used in JPA-based applications to mark a class as a JPA entity.
  - It indicates that the class represents a database table
- **@Value:**
  - Used to inject external properties or configuration values into Spring components.
  - You can use it to configure application properties with values from property files, environment variables, or command-line
- **@Configuration:**
  - Indicates that a class is a Spring configuration class.
  - It's used to define beans and configure the application context.

The Full Example in IntelliJ

© Prof. Fabio Cravegna, Università di Torino

We reuse the Disney Character example

- The server serves an HTML form asking for a character (name, surname, dob)
- The server returns the character object with added its age



Insert A Character

Name: Mickey Surname: Mouse Year of Birth: 12/01/1938

Insert A Character

The result is: {"name":"Mickey","surname":"Mouse","dob":"1938-01-12","age":85}

UNIVERSITÀ
DI TORINO

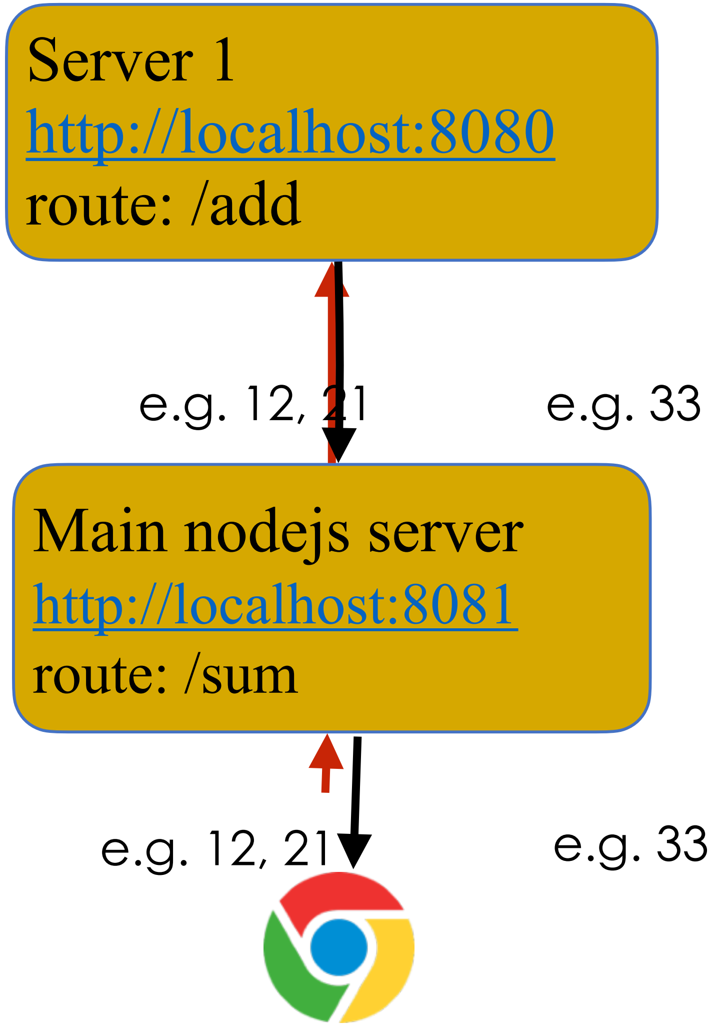
Questions?

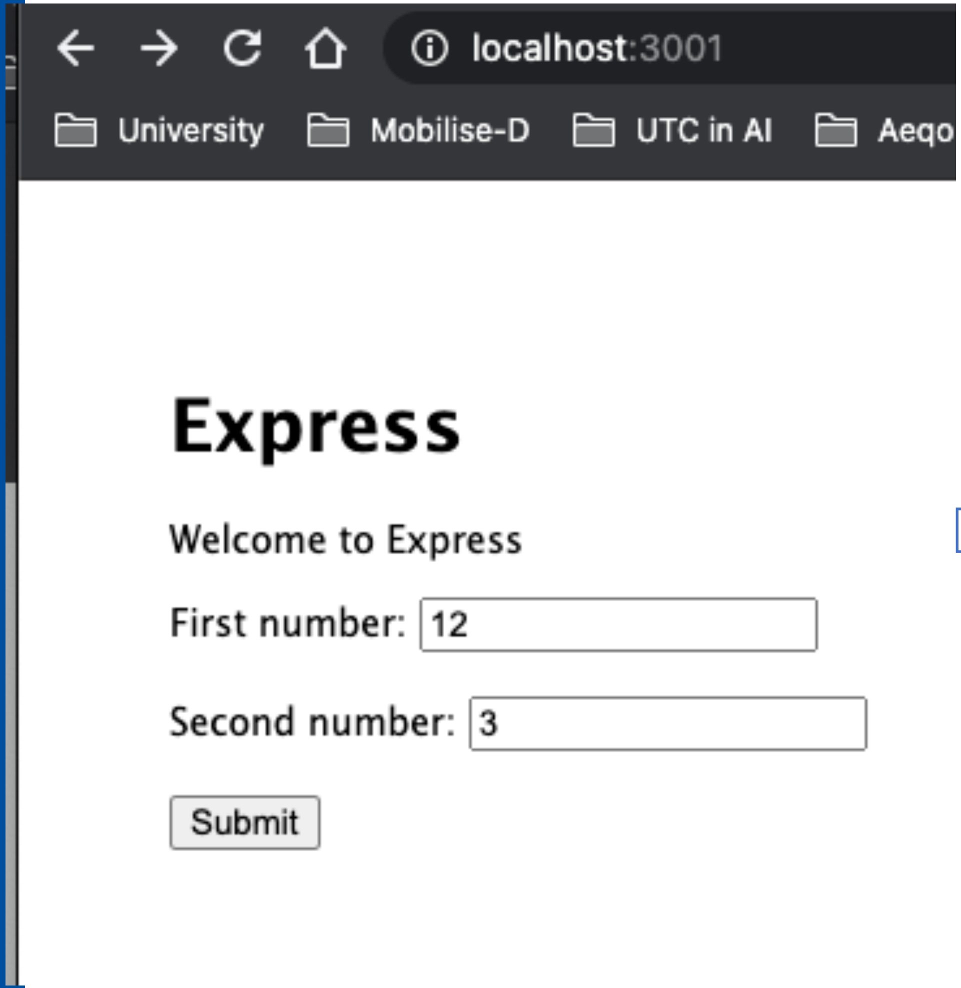
© Prof. Fabio Cravegna, Università di Torino

Connecting to another service

© Prof. Fabio Cravegna, Università di Torino

Connecting to another server



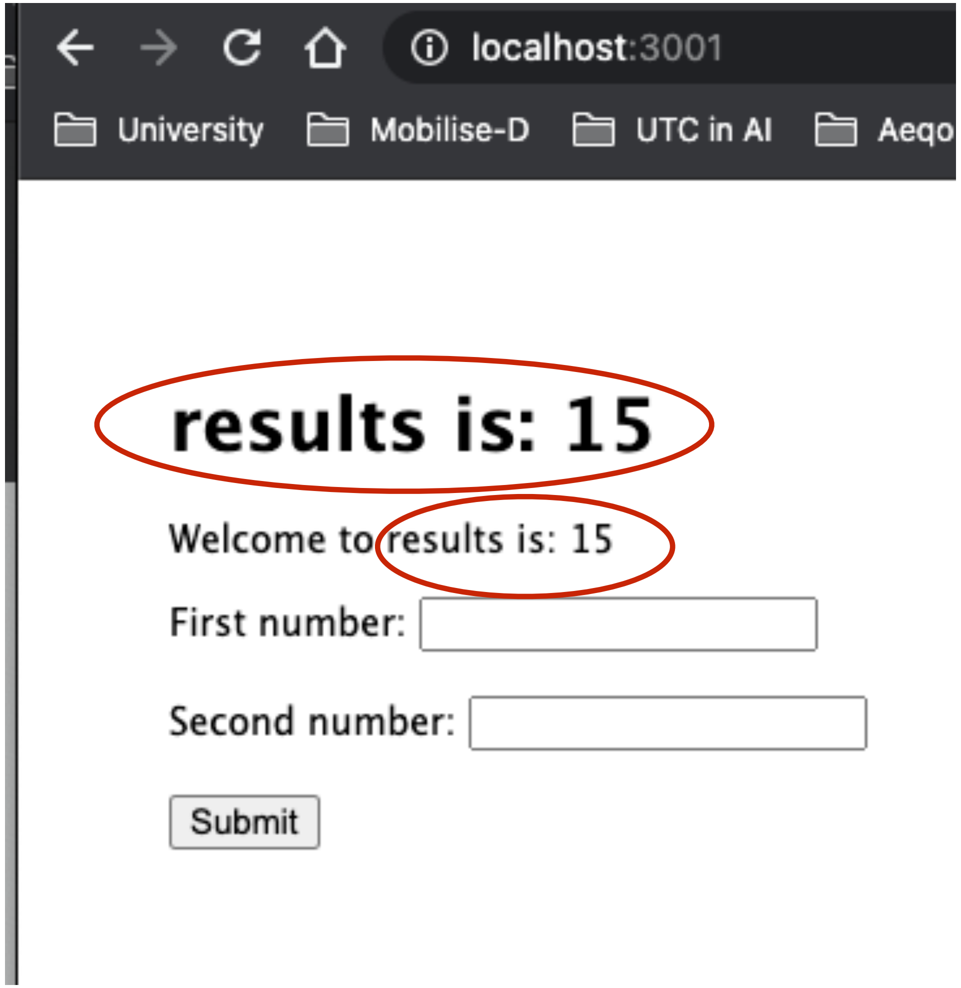


Express

Welcome to Express

First number: 12

Second number: 3



results is: 15

Welcome to results is: 15

First number:

Second number:

RestTemplate

- The RestTemplate class is a Spring Framework class that provides a simplified way to make HTTP requests.
- The RestTemplate class will automatically serialise the object received into JSON and send it in the HTTP request body.
- The RestTemplate class will also automatically deserialise the response from server 2 into an object. The object will contain the result of the operation by the remote server
- The RestTemplate class is a very powerful tool for making HTTP requests in Spring Boot applications.
- Here are some of the benefits of using the RestTemplate class:
  - It is easy to use, providing a simple, template method API over underlying HTTP client libraries.
  - It is thread-safe, so you can safely share it between multiple threads.
  - It supports a wide range of HTTP features, such as authentication, cookies, and caching.
  - It can be used to make requests to both HTTP and HTTPS endpoints.

UNIVERSITÀ
DI TORINO

Connecting to another server

```
@RestController
public class Server1Controller {

    @PostMapping("/sum")
     public String sum(@RequestBody NumberSumRecord record) {

        // Send the record to server 2
        String result = new RestTemplate().postForObject("http://localhost:8081/sum", record, String.class);

        // Return the result to the axios query
        return result;
    }
}

class NumberSumRecord {
    private int number1;
    private int number2;
    public NumberSumRecord(int number1, int number2) {      this.number1 = number1;    this.number2 = number2;
    public int getNumber1() {        return number1;     }
    public int getNumber2() {        return number2;     }
}
```

we are in the rest controller of server 1

we receive an object of type NumberSumRecord

we send it to server 2 on 8081

we postForObject, i.e. we want to receive an object back of type X (String in this case although it would be best to return a JSON object)

we return it to axios

normally you would use docker to create micro services but we will not cover that

UNIVERSITÀ DI TORINO

Note: Java Records

```
public class Person {

    private final String name;
    private final String address;

    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, address);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else if (!(obj instanceof Person)) {
            return false;
        } else {
            Person other = (Person) obj;
            return Objects.equals(name, other.name)
              && Objects.equals(address, other.address);
        }
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", address=" + address + "]";
    }

    // standard getters
}
```

note: they must all be final!

```
public record Person (
               String name,
               String address
               ) {}
```

note: fields in parentheses empty braces

Let’s see the code in IntelliJ

Questions?

© Prof. Fabio Cravegna, Università di Torino