



SPRING BOOT

ACCESSORIES

Version: 2.0

Date: 04.2021

www.ivoronline.com

1 DIFFERENT FUNCTIONALITIES.....	7
1.1 Validation.....	8
1.1.1 Validate - Request Parameters.....	11
1.1.2 Validate - DTO - From Request Parameters	14
1.1.3 Validate - DTO - From JSON.....	17
1.1.4 Validate - Custom Annotation	21
1.1.5 Catch Exception - Using @ExceptionHandler.....	25
1.1.6 Catch Exception - Using @ControllerAdvice	28
1.1.7 Throw Exception	31
1.2 WebClient.....	34
1.2.1 Server	35
1.2.2 Client - Mono	38
1.2.3 Client - Flux	42
1.3 Tasks.....	46
1.3.1 Startup.....	47
1.3.2 Scheduled	49
1.4 Custom Banner.....	51
2 FILTERS & INTERCEPTORS.....	53
2.1 Filters	54
2.1.1 Create Filter - Using Implements Filter.....	56
2.1.2 Create Filter - Using Extends OncePerRequestFilter	59
2.1.3 Create Filter Chain	62
2.1.4 Block Requests.....	65
2.1.5 HTTP Request Parameters - Get	69
2.1.6 HTTP Request Parameters - Add	73
2.1.7 HTTP Request Parameters - Edit.....	78
2.1.8 Apply Filter only for specific URL.....	83
2.2 Interceptors.....	86
2.2.1 Create Interceptor - Implements HandlerInterceptor.....	87
2.2.2 Create Interceptor Chain.....	90
2.2.3 HTTP Request/Response - Get Parameters.....	96
3 LOGGING	101
3.1 Slf4j	102
3.1.1 Using LoggerFactory.....	103
3.1.2 Using @Slf4j.....	105
3.1.3 Logback.....	107
3.1.4 Logback - Configure - XML.....	109
3.1.5 Logback - Configure - XML - DB	112
3.1.6 Logback - Configure - XML - DB - Custom	118
3.1.7 Logback - Configure - Properties	124
3.1.8 Log4j	127
3.1.9 Log4j - Configure - XML	130
3.1.10 Log4j - Configure - Properties.....	135
3.2 AOP Annotations.....	140
3.2.1 Log Start End.....	141
3.2.2 Log Execution Time	144

4 TESTING	147
4.1 JUnit 5	148
4.1.1 Create Test Class - Manually	149
4.1.2 Create Test Class - Automatically	152
4.1.3 Run Tests	155
4.1.4 Example	159
4.1.5 @SpringBootTest	162
4.1.6 @Test	165
4.1.7 @Disable	168
4.1.8 @BeforeAll, @AfterAll, @BeforeEach, @AfterEach	171
4.1.9 assert()	174
4.1.10 assertThrows()	177
4.2 Mockito	181
4.2.1 Mock - Object - @Mock	184
4.2.2 Mock - Object - @MockBean	189
4.2.3 Mock - Object - @Spy	194
4.2.4 Mock - Object - @Spybean	199
4.2.5 Mock - Method - when()	204
4.2.6 Mock - Method - given()	209
4.2.7 Mock - Method - doReturn()	214
4.2.8 @InjectMocks	219
4.2.9 verify()	224
4.2.10 verify() - ArgumentMatchers	229
4.2.11 verify() - ArgumentCaptor	234
4.3 MockMVC	239
4.3.1 @WebMvcTest	240
4.3.2 Test - Request - URL Mapping	243
4.3.3 Test - Request - Parameters	247
4.3.4 Test - Request - Path Variables	250
4.3.5 Test - Request - Body - JSON	253
4.3.6 Test - Response - Status	258
4.3.7 Test - Response - Body - Text	262
4.3.8 Test - Response - Body - JSON	265
5 DEMO APPLICATIONS	269
5.1 Currencies	270
5.1.1 Step 1 - Load Data	271
5.1.2 Step 2 - Controller	280
5.1.3 Step 3 - Validation	286
5.1.4 Step 4 - Security	291
5.1.5 Step 5 - Logging with Annotation into File	294
5.1.6 Step 6 - Logging with Filter into DB	301
5.1.7 Step 7 - Download Excel with DB Log	306
5.1.8 Step 8 - Unit Testing	311
6 APPENDIX	315
6.1 IntelliJ	316
6.1.1 Install	317
6.1.2 Create Project	321
6.1.3 Run Application	323
6.2 JUnit	324

6.2.1	Test Exception	325
6.3	Lombok.....	328
6.3.1	Lombok API - Use	329
6.3.2	Lombok Plugin - Install for IntelliJ.....	332
6.3.3	Lombok Plugin - Lombok.....	333
6.3.4	Lombok Plugin - Delombok	334
6.4	Mockito	336
6.4.1	@Mock - Into Property	337
6.4.2	@Mock - Into Setter.....	342
6.4.3	@Mock - Into Constructor	347
6.5	Swagger.....	352
6.5.1	GET	353
6.5.2	POST	358
6.5.3	Error - Unable to infer base URL.....	363

ABOUT THE BOOK

This is second 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 with non-core Spring Boot functionalities like:

Validation, Testing, Logging, Filters, Interceptors, 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 Different Functionalities

Info

- Following tutorials show how to use different Spring Boot functionalities.

1.1 Validation

Info

- Following tutorials show how to validate User Input by validating either
 - individual HTTP Request Parameters (@RequestParam @NotBlank String name)
 - DTO during Deserialization from
 - Request Parameters (@NotBlank public String name;)
 - JSON Body (@NotBlank public String name;)
- In both cases the same Annotations can be assigned either to
 - Endpoint's Input Parameter (@RequestParam @NotBlank String name)
 - DTO Property (@NotBlank public String name;)
 - additional Endpoint is used to catch validation Exceptions (and return errors to the User)
- Controller's `@ExceptionHandler` Methods only catch Exceptions thrown inside the same Controller.
Use `@ControllerAdvice` class as a central place to catch Exceptions from all Controllers.

Annotations

ANNOTATION	PARAMETERS
<code>@NotBlank</code>	(message = "String Name is mandatory")
<code>@NotNull</code>	(message = "Integer Age is mandatory")
<code>@Min(18)</code>	(value = 18, message = "Minimum value for Age is 18")
<code>@Max(100)</code>	(value = 100, message = "Maximum value for Age is 100")
<code>@Size(min=5, max=30)</code>	(min=5, max=30, message = "String must be between 5 and 30 characters")

Exceptions

VALIDATION TYPE	EXCEPTION	REFERENCE
<code>(@RequestParam @NotBlank String name)</code>	<code>MissingServletRequestParameterException</code>	<code>Request Parameters</code>
<code>(@Valid PersonDTO personDTO)</code>	<code>BindException</code>	<code>DTO - Parameters</code>
<code>(@Valid @RequestBody PersonDTO personDTO)</code>	<code>MethodArgumentNotValidException</code>	<code>DTO - JSON</code>

Validate & Throw Exceptions

Validate Request Parameters

MyController.java

```
@Validated
@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@RequestParam @NotBlank String name) {
        return "Hello " + name;
    }
}
```

Validate DTO (During Deserialization from Request Parameters)

MyController.java

```
@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@Valid PersonDTO personDTO) {
        return "Hello " + personDTO.name;
    }
}
```

PersonDTO.java

```
public class PersonDTO {

    //PROPERTIES
    @NotBlank public String name;

    //SETTERS (Only needed for Deserialization from Request Parameters. JSON uses Reflection instead.)
    public void setName(String name) { this.name = name; }

}
```

Validate DTO (During Deserialization from JSON)

MyController.java

```
@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/AddPerson")
    public String addPerson(@Valid @RequestBody PersonDTO personDTO) {
        return "Person added";
    }
}
```

PersonDTO.java

```
public class PersonDTO {
    //PROPERTIES
    @NotBlank public String name;
}
```

Catch Exceptions

- Controller's `@ExceptionHandler` Method only catches Exceptions thrown inside the same Controller.
- Use `@ControllerAdvice` class as a central place to catch Exceptions from all Controllers.

Exception Methods

(differ base on the type of Exception)

VALIDATION TYPE	EXCEPTION	REFERENCE
(<code>@RequestParam @NotBlank String name</code>)	<code>MissingServletRequestParameterException</code>	Request Parameters
(<code>@Valid PersonDTO personDTO</code>)	<code>BindException</code>	DTO - Parameters
(<code>@Valid @RequestBody PersonDTO personDTO</code>)	<code>MethodArgumentNotValidException</code>	DTO - JSON

MyController.java

(example for Request Parameters)

```
@Controller
public class MyController {

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public String handleExceptions(MissingServletRequestParameterException exception) {

        //GET EXCEPTION DETAILS
        String parameterType = exception.getParameterType(); //String
        String parameterName = exception.getParameterName(); //name
        String message      = exception.getMessage();          //Required String parameter 'name' is not present

        //RETURN MESSAGE
        return message;

    }
}
```

MyController.java

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public String handleIdentifiersNotMatchingException(MissingServletRequestParameterException exception) {

        //GET EXCEPTION DETAILS
        String parameterType = exception.getParameterType(); //String
        String parameterName = exception.getParameterName(); //name
        String message      = exception.getMessage();          //Required String parameter 'name' is not present

        //RETURN MESSAGE
        return message;

    }
}
```

1.1.1 Validate - Request Parameters

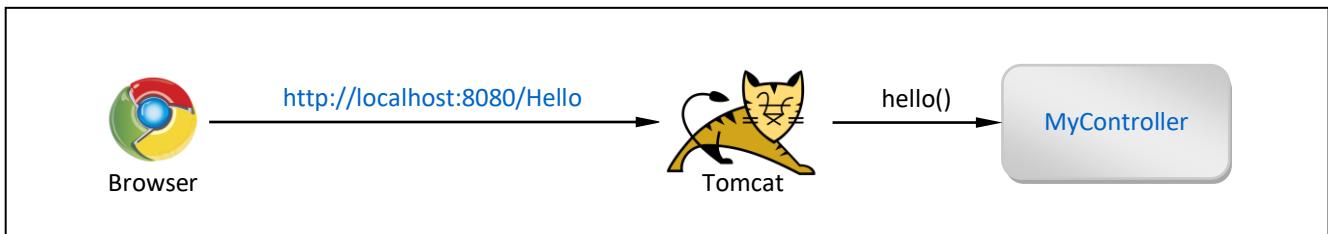
Info

[G] [R]

- This tutorial shows how to Validate Request Parameters when using `@RequestParam`.
If Parameter is missing or contains invalid value, `MissingServletRequestParameterException` is thrown.
We will create Endpoint that catches Exception and returns error back to the User.
- If Endpoint uses multiple `@RequestParam`, Exception is thrown only for the **first error**.
If you want all of the errors to be reported then you need to use DTO approach
 - DTO - From Request Parameters
 - DTO - From JSON

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- [Create Project:](#) `springboot_dto_json_object_constructor` (add Spring Boot Starters from the table)
- [Edit File:](#) `pom.xml` (add validation dependency)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside package controllers)

pom.xml

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

MyController.java

```
package com.ivoronline.springboot_validation_requestparameters.controllers;

import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.*;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Controller
@Validated
public class MyController {

    //=====
    // HELLO
    //=====

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(
        @RequestParam @NotBlank String name,
        @RequestParam @NotNull Integer age
    ) {
        return "Hello " + name;
    }

    //=====
    // HANDLE EXCEPTIONS (it only catches first exception)
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public String handleExceptions(MissingServletRequestParameterException exception) {

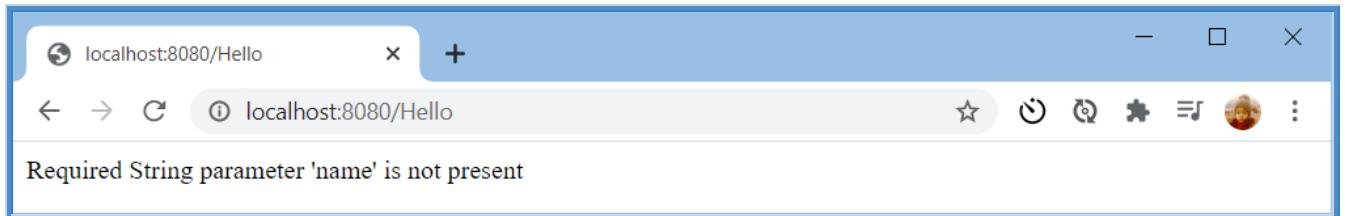
        //GET EXCEPTION DETAILS
        String parameterType = exception.getParameterType(); //String
        String parameterName = exception.getParameterName(); //name
        String message      = exception.getMessage();           //Required String parameter 'name' is not present

        //RETURN MESSAGE
        return message;
    }
}
```

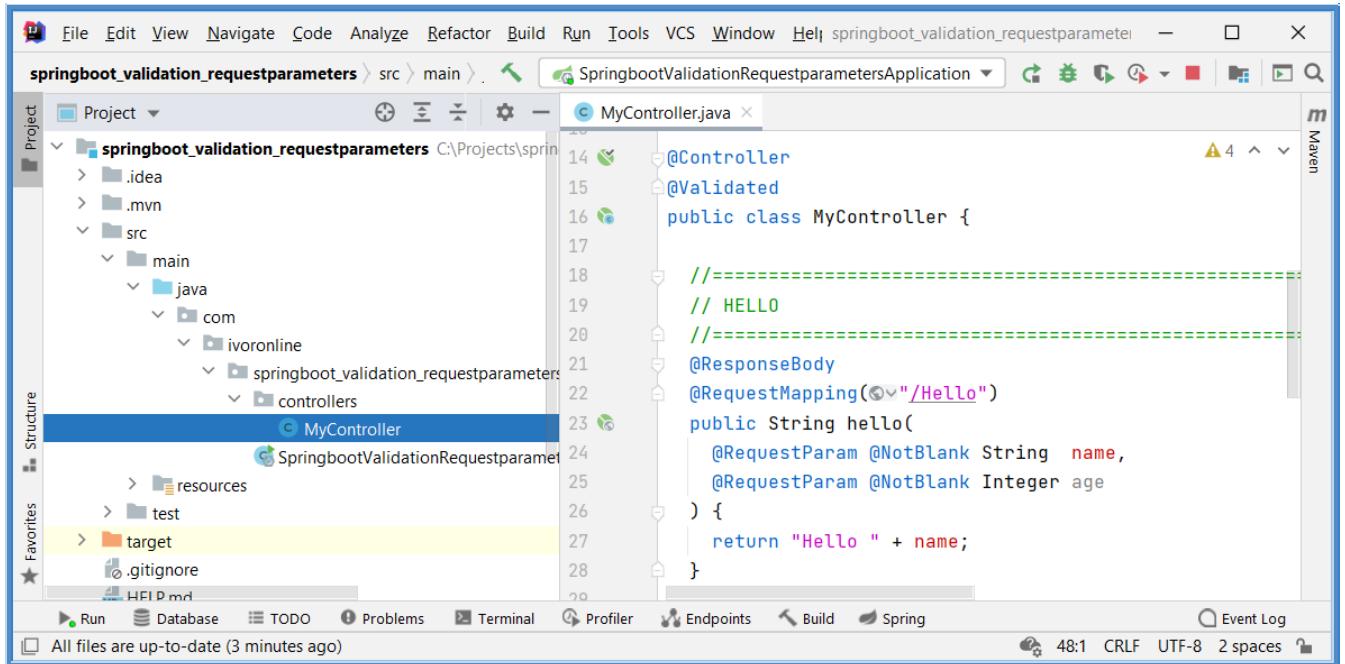
Results

<http://localhost:8080>Hello>

(reports only first error)



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-validation</artifactId>
</dependency>

</dependencies>
```

1.1.2 Validate - DTO - From Request Parameters

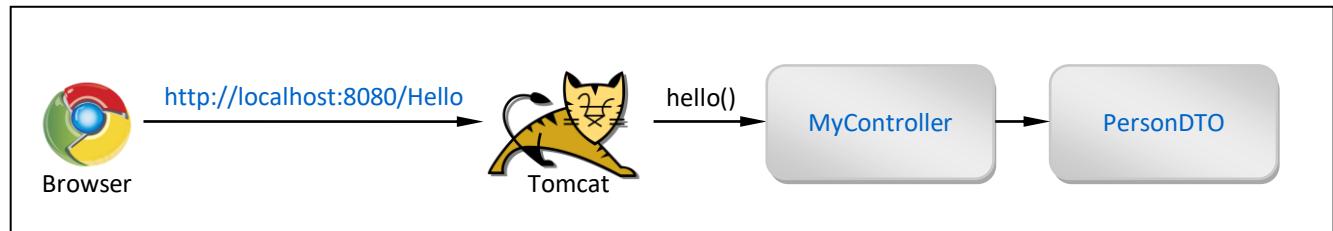
Info

[G]

- This tutorial shows how to perform Validation while Deserializing HTTP Request Parameters into DTO.
- If Parameter is missing or contains invalid value, `BindException` is thrown.
- You can create Endpoint that catches `BindException` and returns all validation errors back to the User.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_dto_json_object_constructor` (add Spring Boot Starters from the table)
- Edit File:** `pom.xml` (add validation dependency)
- Create Package:** controllers (inside main package)
 - Create Class:** `MyController.java` (inside package controllers)
- Create Package:** DTO (inside main package)
 - Create Class:** `PersonDTO.java` (inside package controllers)

pom.xml

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

PersonDTO.java

```
package com.ivoronline.springboot_deserialize_requestparameters.DTO;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

public class PersonDTO {

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

    @NotNull
    public Integer age;

    //SETTERS (used for deserialization)
    public void setName(String name) { this.name = name; }
    public void setAge (Integer age ) { this.age = age; }

}
```

MyController.java

```
package com.ivorononline.springboot_deserialize_requestparameters.controllers;

import com.ivorononline.springboot_deserialize_requestparameters.DTO.PersonDTO;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import javax.validation.Valid;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Controller
public class MyController {

    //=====
    // HELLO
    //=====

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@Valid PersonDTO personDTO) {
        return "Hello " + personDTO.name;
    }

    //=====
    // HANDLE EXCEPTIONS
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(BindException.class)
    public Map<String, String> handleExceptions(BindException exception) {

        //CREATE MAP FOR STORING ERRORS
        Map<String, String> errors = new HashMap<>();

        //ITERATE THROUGH ERRORS
        List<ObjectError> objectErrors = exception.getAllErrors();
        for(ObjectError objectError : objectErrors) {

            //GET ERROR
            FieldError fieldError = (FieldError) objectError;
            String fieldName = fieldError.getField();
            String errorMessage = fieldError.getDefaultMessage();

            //STORE ERROR
            errors.put(fieldName, errorMessage);

        }

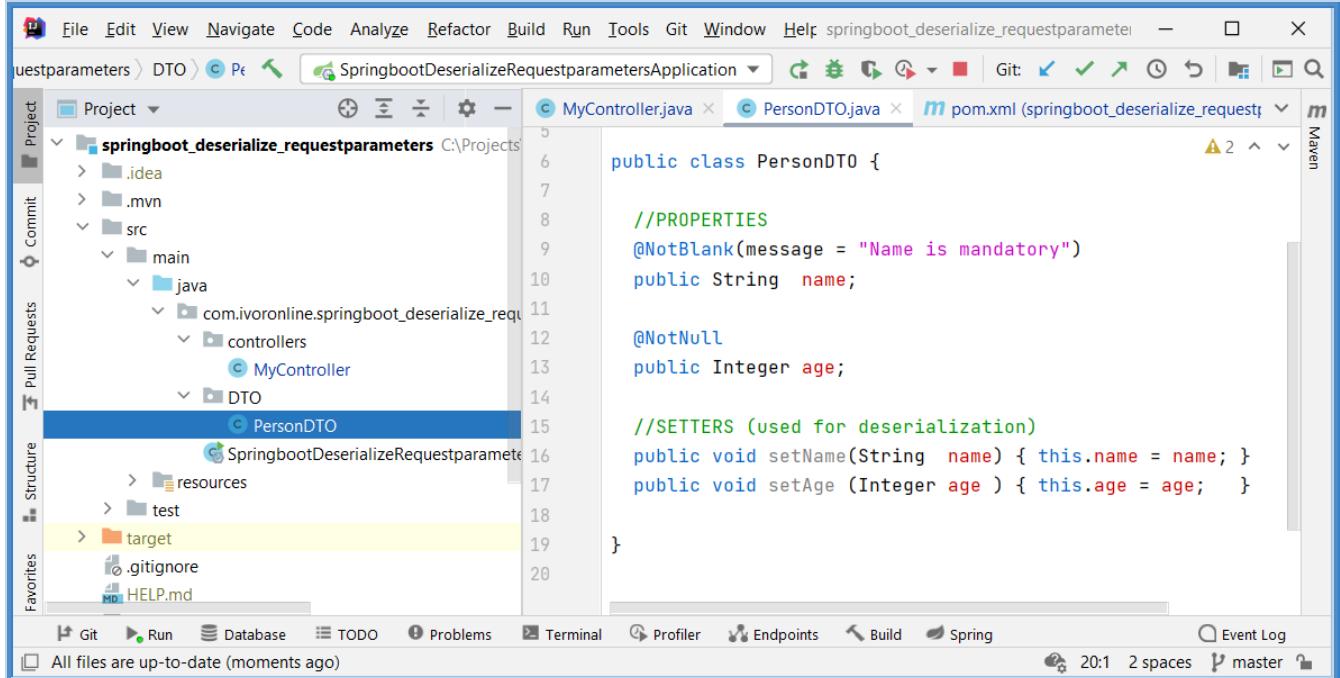
        //RETURN ERRORS
        return errors;
    }
}
```

Results

<http://localhost:8080>Hello>



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-validation</artifactId>  
    </dependency>  
  
</dependencies>
```

1.1.3 Validate - DTO - From JSON

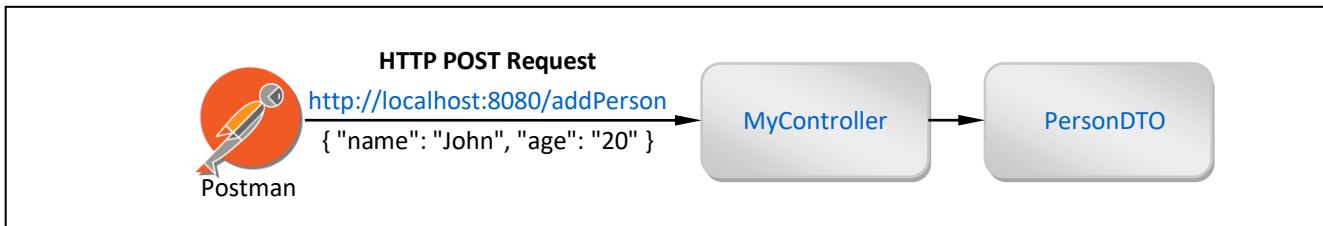
Info

[G] [R]

- This tutorial shows how to **Validate User Input** during Deserialization of **JSON into DTO**.
- Validation checks if JSON contains all parameters and if they have proper value.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project:** springboot_dto_json_validate (add Spring Boot Starters from the table)
- Edit File:** pom.xml (add validation dependency)
- Create Package:** DTO (inside main package)
 - Create Class:** PersonDTO.java (inside package DTO)
- Create Package:** controllers (inside main package)
 - Create Class:** MyController.java (inside package controllers)

pom.xml

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

PersonDTO.java

```
package com.ivoronline.springboot_json_validate.DTO;

import org.springframework.stereotype.Component;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Component
public class PersonDTO {

    @NotBlank(message = "Name is mandatory")                                // @NotBlank()
    @Size(min=5, max=30)                                                 // characters
    public String name;

    @NotNull                                         // must not be null
    @Min(value = 18, message = "Minimum value for Age is 18") // @Min(18)
    public Integer age;

}
```

MyController.java

```
package com.ivorononline.springboot_json_validate.controllers;

import com.ivorononline.springboot_json_validate.DTO.PersonDTO;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import javax.validation.Valid;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Controller
public class MyController {

    //=====
    // ADD PERSON
    //=====

    @ResponseBody
    @RequestMapping("/AddPerson")
    public String addPerson(@Valid @RequestBody PersonDTO personDTO) {
        return "Person added";
    }

    //=====
    // HANDLE EXCEPTIONS
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Map<String, String> handleExceptions(MethodArgumentNotValidException notValidExceptions) {

        //CREATE MAP FOR STORING ERRORS
        Map<String, String> errors = new HashMap<>();

        //ITERATE THROUGH ERRORS
        List<ObjectError> objectErrors = notValidExceptions.getBindingResult().getAllErrors();
        for(ObjectError objectError : objectErrors) {

            //GET ERROR
            FieldError fieldError = (FieldError) objectError;
            String fieldName = fieldError.getField();
            String errorMessage = fieldError.getDefaultMessage();

            //STORE ERROR
            errors.put(fieldName, errorMessage);
        }
    }

    //RETURN ERRORS
    return errors;
}
}
```

Results

- Start Postman

POST

```
http://localhost:8080/AddPerson
```

Headers

(add Key-Value)

```
Content-Type: application/json
```

Body

(option: raw)

```
{  
}
```

Postman

The screenshot shows the Postman application window. In the top navigation bar, 'File', 'Edit', 'View', 'Help', and 'Upgrade' are visible. Below the bar, there are tabs for 'Home', 'Workspaces', 'Reports', and 'Explore'. A search bar says 'Search Postman'. On the right side of the header, there are icons for cloud storage, a plus sign, a gear, a bell, and a refresh symbol.

In the main workspace, a 'DTO / New Request' section is open. It shows a 'GET' method selected, with the URL 'http://localhost:8080/AddPerson'. The 'Body' tab is active, showing a JSON object with two properties: 'name' and 'age'. The 'Status' field indicates a '400 Bad Request' response.

Below the request details, there are tabs for 'Body', 'Cookies', 'Headers (4)', and 'Test Results'. The 'Body' tab is selected, showing the JSON response in 'Pretty' format:

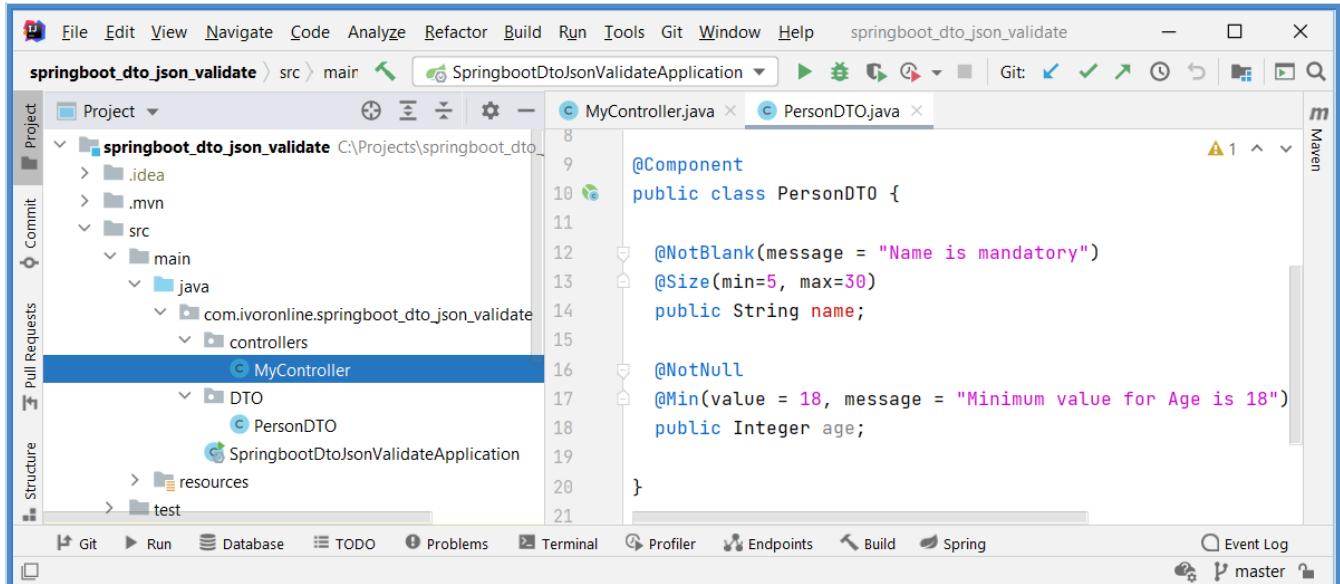
```
1 {  
2   "name": "Name is mandatory",  
3   "age": "must not be null"  
4 }
```

At the bottom of the interface, there are buttons for 'Find and Replace' and 'Console', along with a toolbar containing icons for 'Bootcamp', 'Runner', 'Trash', and other settings.

HTTP Response Body

```
{  
  "name": "Name is mandatory",  
  "age": "must not be null"  
}
```

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-validation</artifactId>
    </dependency>

</dependencies>
```

1.1.4 Validate - Custom Annotation

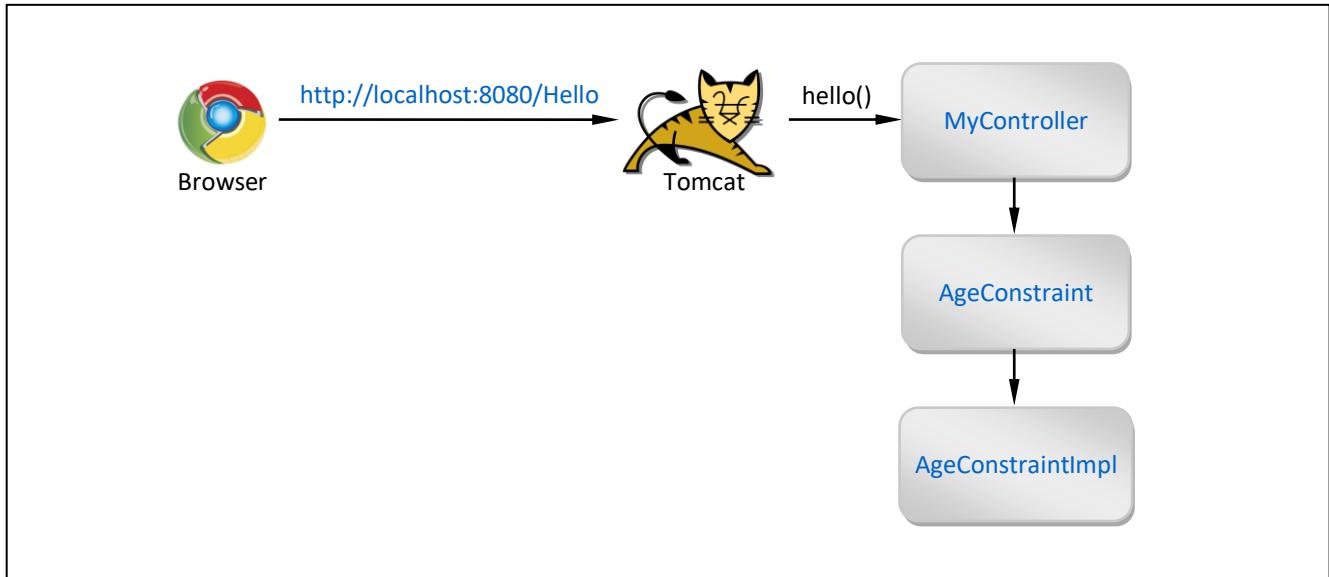
Info

[G] [R]

- This tutorial shows how to Validate Request Parameters when using `@RequestParam`. If Parameter is missing or contains invalid value, `MissingServletRequestParameterException` is thrown. We will create Endpoint that catches Exception and returns error back to the User.
- If Endpoint uses multiple `@RequestParam`, Exception is thrown only for the **first error**. If you want all of the errors to be reported then you need to use DTO approach
 - DTO - From Request Parameters
 - DTO - From JSON

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: [springboot_dto_json_object_constructor](#) (add Spring Boot Starters from the table)
- Edit File: [pom.xml](#) (add validation dependency)
- Create Package: validators (inside main package)
 - Create Class: [AgeConstraintImpl.java](#) (inside package validators)
 - Create Class: [AgeConstraint.java](#) (inside package validators)
- Create Package: controllers (inside main package)
 - Create Class: [MyController.java](#) (inside package controllers)

pom.xml

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

AgeConstraintImpl.java

```
package com.ivoronline.springboot_validation_custom.validators;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class AgeConstraintImpl implements ConstraintValidator<AgeConstraint, Integer> {

    @Override
    public boolean isValid(Integer age, ConstraintValidatorContext ctxt) {
        return 18 < age && age < 100;
    }

}
```

AgeConstraint.java

```
package com.ivoronline.springboot_validation_custom.validators;

import javax.validation.Constraint;
import javax.validation.Payload;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = AgeConstraintImpl.class)
public @interface AgeConstraint {
    String message() default "Must be between 18 and 100";
    Class<?>[] groups () default {};
    Class<? extends Payload>[] payload() default {};
}
```

MyController.java

```
package com.ivorononline.springboot_validation_custom.controllers;

import com.ivorononline.springboot_validation_custom.validators.AgeConstraint;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import javax.validation.ConstraintViolation;
import javax.validation.ConstraintViolationException;
import javax.validation.Path;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

@Controller
@Validated
public class MyController {

    //=====
    // HELLO
    //=====

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@AgeConstraint @RequestParam Integer age) {
        return "John is " + age + " years old";
    }

    //=====
    // HANDLE EXCEPTIONS (it only catches first exception)
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(ConstraintViolationException.class)
    public Map<String, String> handleExceptions(ConstraintViolationException exception) {

        //CREATE MAP FOR STORING ERRORS
        Map<String, String> errors = new HashMap<>();

        //ITERATE THROUGH ERRORS
        Set<ConstraintViolation<?>> violations = exception.getConstraintViolations();
        for(ConstraintViolation<?> violation : violations) {

            //GET PROPERTY NAME
            String propertyName = null;
            for (Path.Node node : violation.getPropertyPath()) { propertyName = node.getName(); }

            //GET OTHER ERROR COMPONENTS
            String message = violation.getMessage();
            Integer invalidValue = (Integer) violation.getInvalidValue();

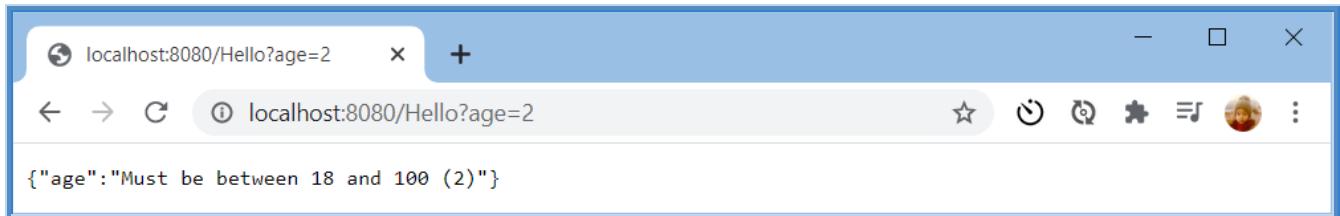
            //STORE ERROR
            errors.put(propertyName, message + " (" + invalidValue + ")");
        }
    }

    //RETURN MESSAGE
    return errors;
}

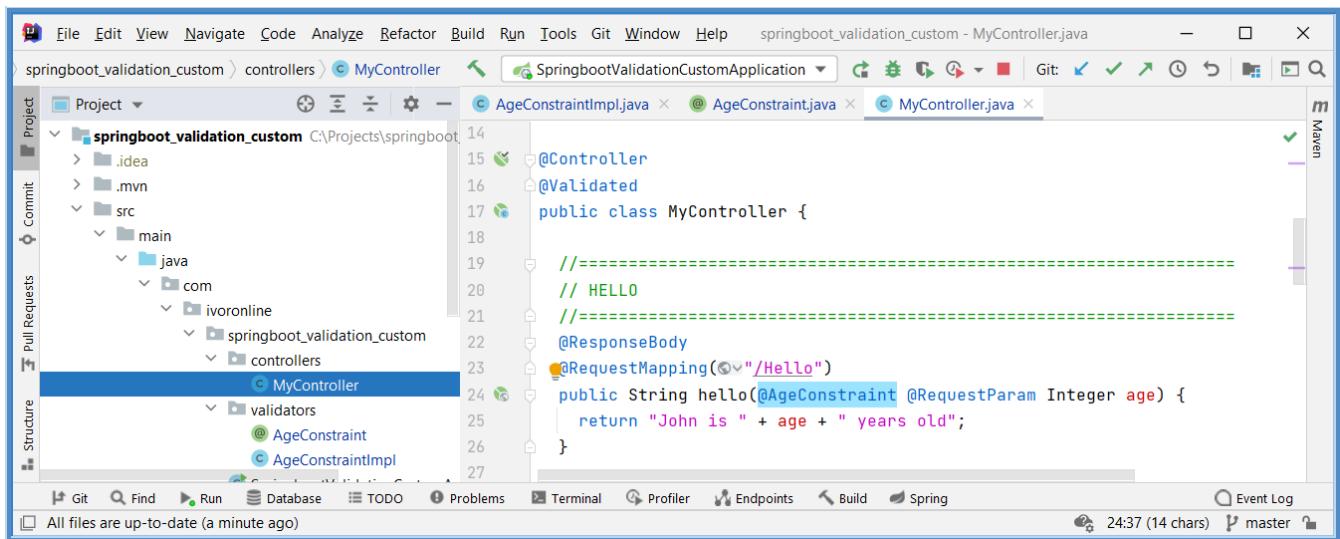
}
```

Results

<http://localhost:8080>Hello?age=2>



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-validation</artifactId>
    </dependency>

</dependencies>
```

1.1.5 Catch Exception - Using @ExceptionHandler

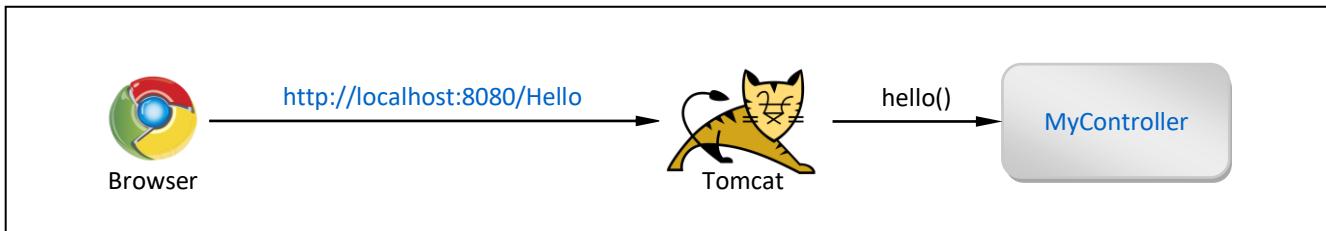
Info

[G]

- This tutorial shows how to catch Validation Exceptions by adding `@ExceptionHandler` Methods to the Controller.
- Such Methods can only catch Exceptions thrown inside the same Controller.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_validation_catchexception_eceptionhandler` (add Spring Boot Starters from the table)
- Edit File: `pom.xml` (add validation dependency)
- Create Package: `controllers` (inside main package)
- Create Class: `MyController.java` (inside package controllers)

pom.xml

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

MyController.java

```
package com.ivoronline.springboot_validation_catchexception_eceptionhandler.controllers;

import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.*;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Controller
@Validated
public class MyController {

    //=====
    // HELLO
    //=====

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(
        @RequestParam @NotBlank String name,
        @RequestParam @NotNull Integer age
    ) {
        return "Hello " + name;
    }

    //=====
    // HANDLE EXCEPTIONS (it only catches first exception)
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public String handleExceptions(MissingServletRequestParameterException exception) {

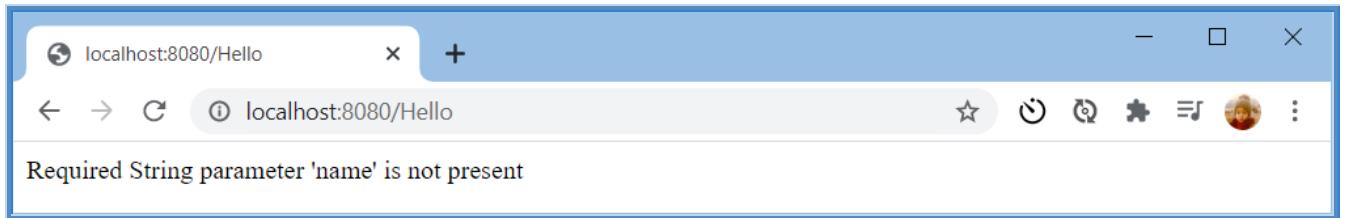
        //GET EXCEPTION DETAILS
        String parameterType = exception.getParameterType(); //String
        String parameterName = exception.getParameterName(); //name
        String message      = exception.getMessage();           //Required String parameter 'name' is not present

        //RETURN MESSAGE
        return message;
    }
}
```

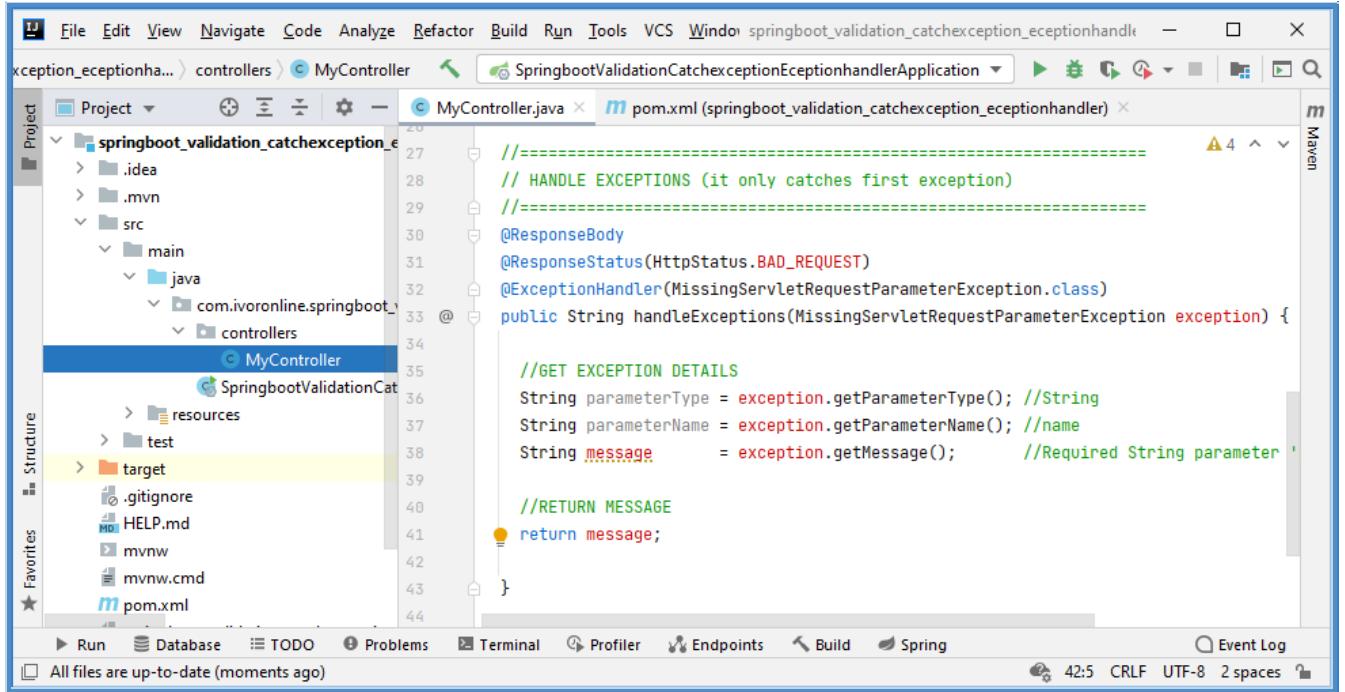
Results

<http://localhost:8080>Hello>

(reports only first error)



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-validation</artifactId>
    </dependency>

</dependencies>
```

1.1.6 Catch Exception - Using @ControllerAdvice

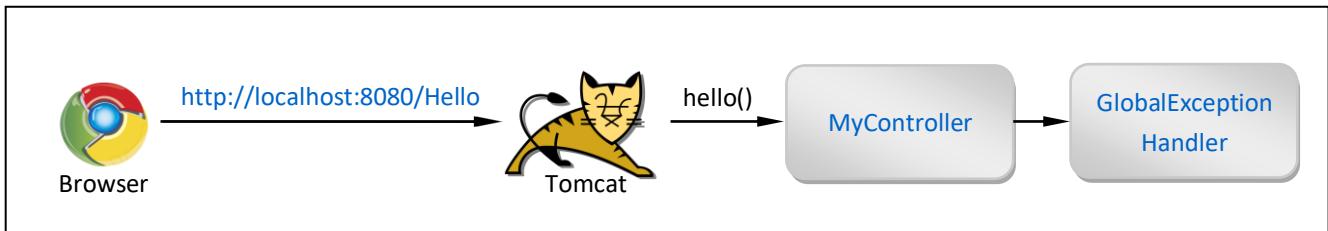
Info

[G]

- This tutorial shows how to catch Validation Exceptions by creating `@ControllerAdvice` Class.
- Such Class will catch Exceptions from all Controllers.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_validation_catchexception_eceptionhandler` (add Spring Boot Starters from the table)
- Edit File:** `pom.xml` (add validation dependency)
- Create Package:** controllers (inside main package)
 - Create Class:** `MyController.java` (inside package controllers)
 - Create Class:** `GlobalExceptionHandler.java` (inside package controllers)

pom.xml

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

MyController.java

```
package com.ivoronline.springboot_validation_catchexception_controlleradvice.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Controller
@Validated
public class MyController {

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(
        @RequestParam @NotBlank String name,
        @RequestParam @NotNull Integer age
    ) {
        return "Hello " + name;
    }
}
```

GlobalExceptionHandler.java

```
package com.ivorononline.springboot_validation_catchexception_controlleradvice.controllers;

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

@ControllerAdvice
public class GlobalExceptionHandler {

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public String handleIdentifiersNotMatchingException(MissingServletRequestParameterException exception) {

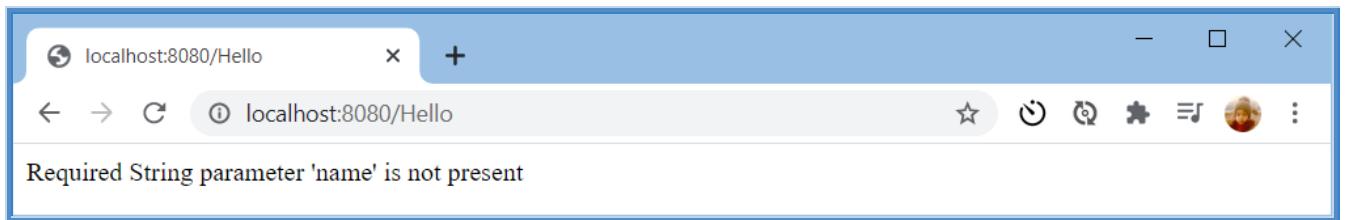
        //GET EXCEPTION DETAILS
        String parameterType = exception.getParameterType(); //String
        String parameterName = exception.getParameterName(); //name
        String message      = exception.getMessage();          //Required String parameter 'name' is not present

        //RETURN MESSAGE
        return message;
    }
}
```

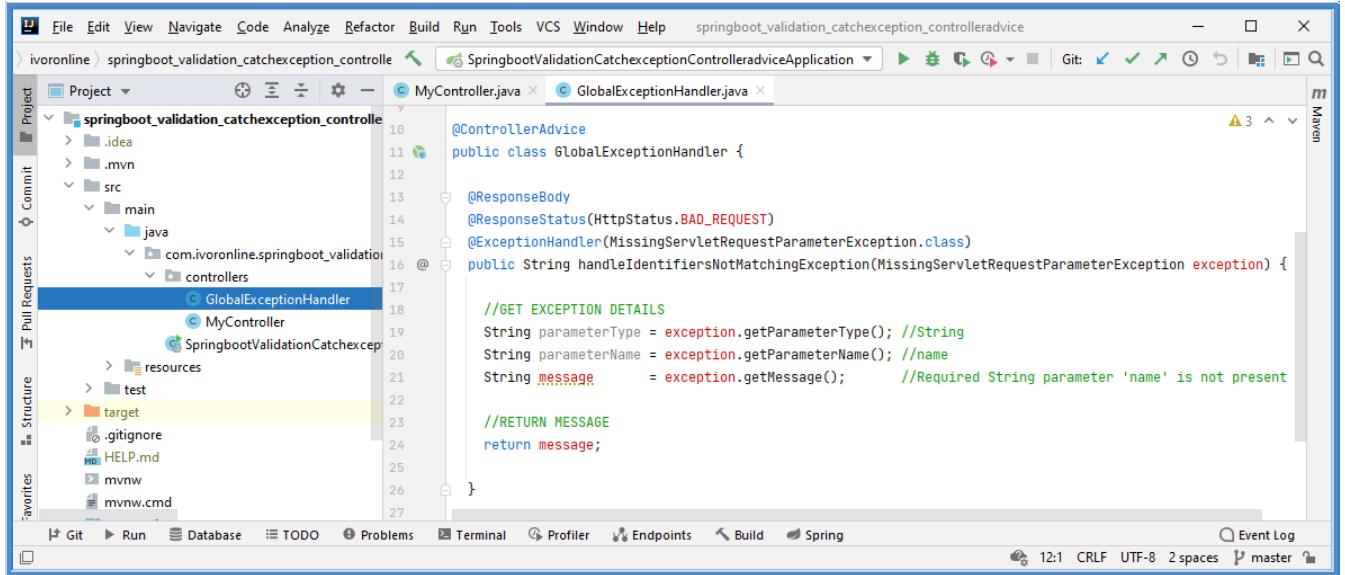
Results

<http://localhost:8080>Hello>

(reports only first error)



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-validation</artifactId>
</dependency>

</dependencies>
```

1.1.7 Throw Exception

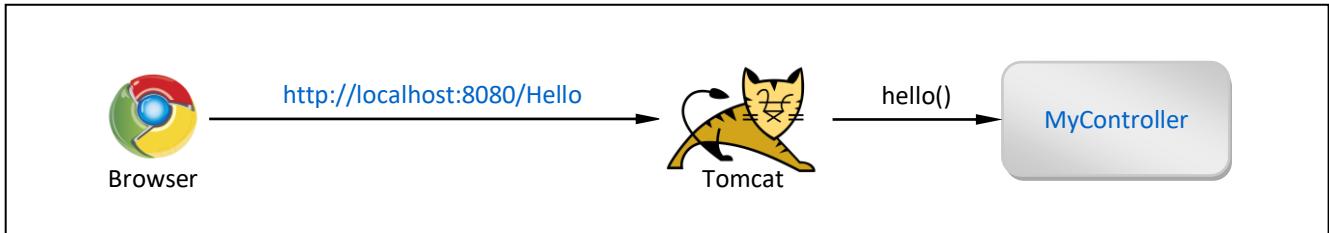
Info

[G]

- This tutorial shows how to catch Validation Exceptions by adding `@ExceptionHandler` Methods to the Controller.
- Such Methods can only catch Exceptions thrown inside the same Controller.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_validation_catchexception_eceptionhandler` (add Spring Boot Starters from the table)
- **Edit File:** `pom.xml` (add validation dependency)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside package controllers)

pom.xml

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

MyController.java

```
package com.ivoronline.springboot_validation_exception_throw.controllers;

import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

@Controller
@Validated
public class MyController {

    //=====
    // HELLO
    //=====

    @ResponseBody
    @RequestMapping("/Hello")
    public void hello() {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "There was some Exception");
    }

    //=====
    // HANDLE EXCEPTIONS
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(ResponseStatusException.class)
    public String handleExceptions(ResponseStatusException exception) {

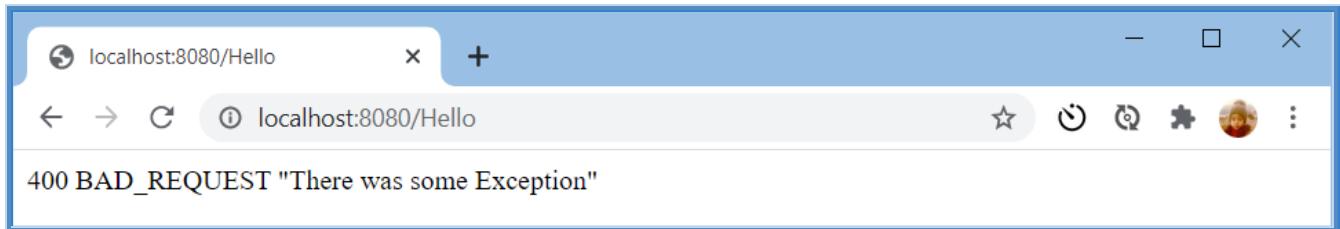
        //GET EXCEPTION DETAILS
        String message = exception.getMessage(); //400 BAD_REQUEST "There was some Exception"
        HttpStatus status = exception.getStatus(); //400 BAD_REQUEST
        String reason = exception.getReason(); //There was some Exception
        Integer statusValue = status.value(); //400
        String statusPhrase = status.getReasonPhrase(); //Bad Request

        //RETURN MESSAGE
        return message;
    }
}
```

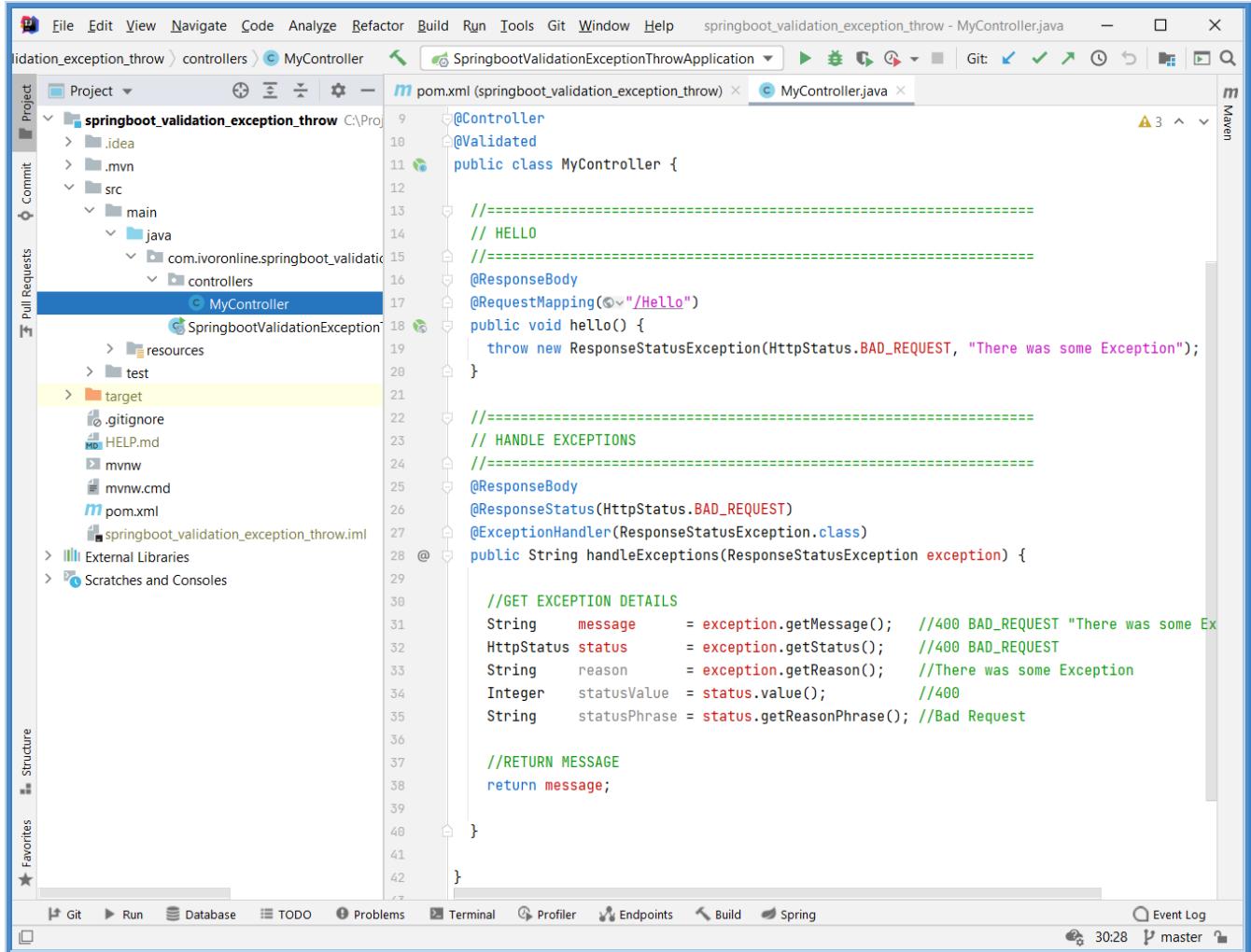
Results

<http://localhost:8080>Hello>

(reports only first error)



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-validation</artifactId>
    </dependency>

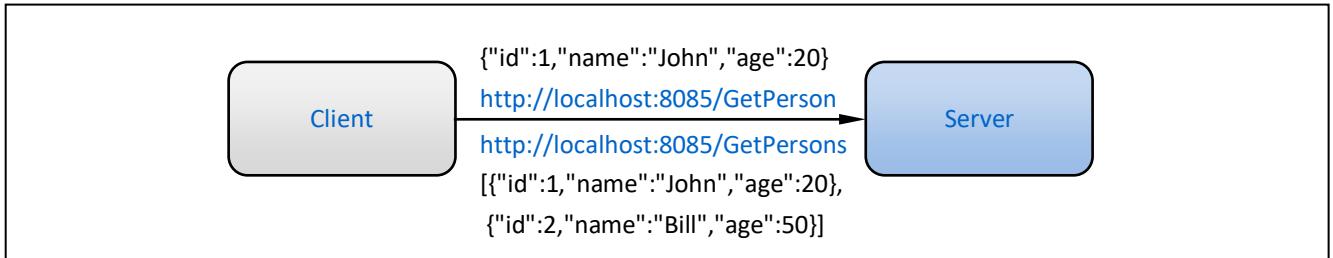
</dependencies>
```

1.2 WebClient

Info

- Following tutorials show how to use **WebClient** Class to call Remote Endpoints.
- To demonstrate this functionality we will create
 - Server** Spring Boot Application that returns Person(s) in JSON format
 - Client** Spring Boot Application that will call Endpoint(s) from the above Server to get Person(s)

Schema



1.2.1 Server

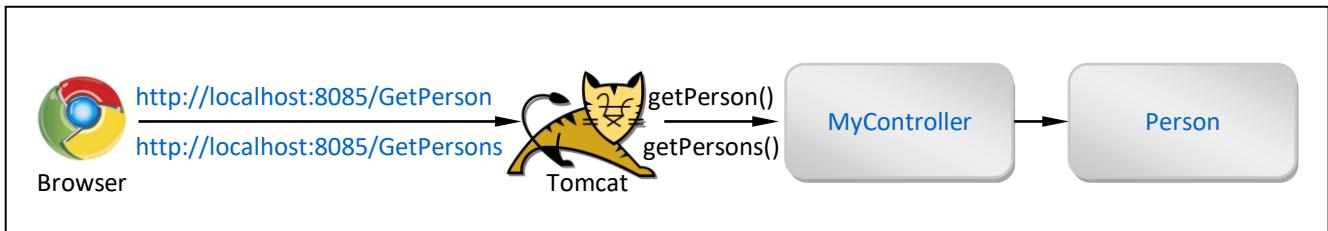
Info

[G]

- This tutorial shows how to create Spring Boot Application that returns Person(s) in JSON format.
- We will use it to demonstrate [WebClient](#) functionality (so that WebClient has something to call and work with).

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)
- Edit File: application.properties (specify port 8085 so that Client can run on 8080)
- Create Package: entities (inside main package)
 - Create Interface: Person.java (inside package entities)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)

application.properties

```
# PORT
server.port = 8085
```

Person.java

```
package com.ivoronline.springboot_webclient_server.entities;

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

MyController.java

```
package com.ivorononline.springboot_webclient_server.controllers;

import com.ivorononline.springboot_webclient_server.entities.Person;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.ArrayList;
import java.util.List;

@Controller
public class MyController {

    //=====
    // GET PERSON
    //=====

    @ResponseBody
    @RequestMapping("/GetPerson")
    public Person getPerson() {

        //CREATE PERSON
        Person person      = new Person();
        person.id      = 1;
        person.name   = "Susan";
        person.age     = 30;

        //RETURN PERSON
        return person;
    }

    //=====
    // GET PERSONS
    //=====

    @ResponseBody
    @RequestMapping("/GetPersons")
    public List<Person> getPersons() {

        //CREATE PERSON 1
        Person person1      = new Person();
        person1.id      = 2;
        person1.name   = "John";
        person1.age     = 40;

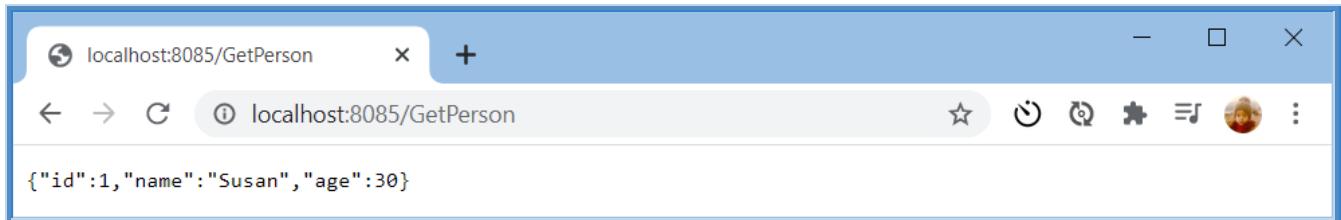
        //CREATE PERSON 2
        Person person2      = new Person();
        person2.id      = 3;
        person2.name   = "Bill";
        person2.age     = 50;

        //CREATE LIST
        List<Person> persons = new ArrayList();
        persons.add(person1);
        persons.add(person2);

        //RETURN PERSON
        return persons;
    }
}
```

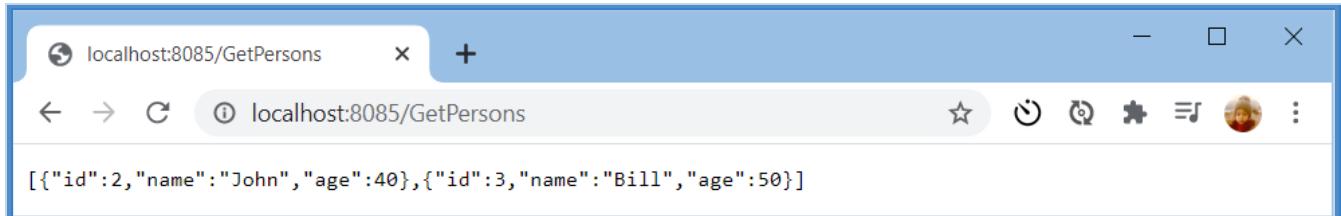
Results

<http://localhost:8085/GetPerson>



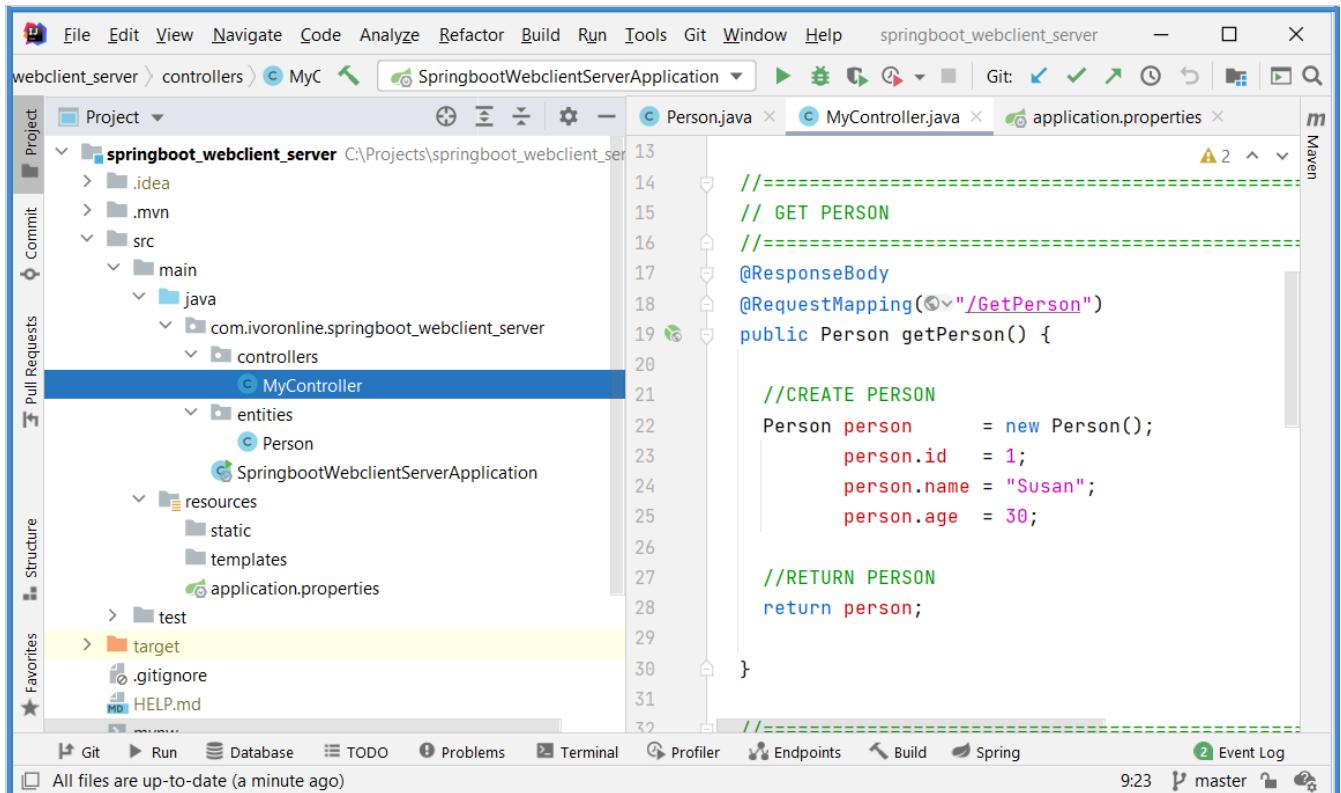
```
{"id":1,"name":"Susan","age":30}
```

<http://localhost:8085/GetPersons>



```
[{"id":2,"name":"John","age":40}, {"id":3,"name":"Bill","age":50}]
```

Application Structure



```
//=====
// GET PERSON
//=====

@RequestBody
@RequestMapping("/GetPerson")
public Person getPerson() {

    //CREATE PERSON
    Person person = new Person();
    person.id = 1;
    person.name = "Susan";
    person.age = 30;

    //RETURN PERSON
    return person;
}
```

pom.xml

```
<dependencies>

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

</dependencies>
```

1.2.2 Client - Mono

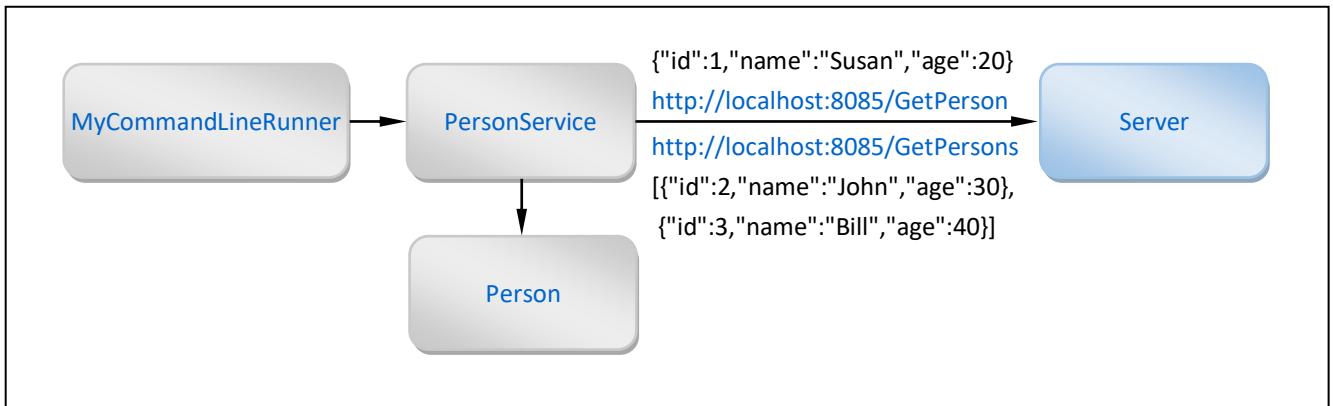
Info

[G] [R]

- This tutorial shows how to use [WebClient](#) Class to call Remote [Server](#) Endpoints that return Person(s).
- This tutorial uses [bodyToMono\(\)](#) to retrieve Data.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server
Web	Spring Reactive Web	Enables WebClient

Procedure

- **Create Project:** controller_returns_text_json (add Spring Boot Starters from the table)
- **Create Package:** entities (inside main package)
 - **Create Interface:** [Person.java](#) (inside package entities)
- **Create Package:** services (inside main package)
 - **Create Class:** [PersonService.java](#) (inside controllers package)
- **Create Package:** runners (inside main package)
 - **Create Class:** [MyCommandLineRunner.java](#) (inside runners package)

Person.java

```
package com.ivoronline.springboot_webclient_server.entities;

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

PersonService.java

```
package com.ivorononline.springboot_webclient.services;

import com.ivorononline.springboot_webclient.entities.Person;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;

import java.time.Duration;

public class PersonService {

    //=====
    // GET PERSON
    //=====

    public void getPerson() {

        //GET PERSON FROM SERVER
        Person person = WebClient.create("http://localhost:8085")
            .get()
            .uri("/GetPerson")
            .retrieve()
            .bodyToMono(Person.class)
            .block(Duration.ofSeconds(3));

        //DISPLAY PERSON
        System.out.println("getPerson ()");
        System.out.println(person.id + " " + person.name+ " " + person.age);

    }

    //=====
    // GET PERSONS
    //=====

    public void getPersons() {

        //GET PERSON FROM SERVER
        Person[] persons = WebClient.create("http://localhost:8085")
            .get()
            .uri("/GetPersons")
            .retrieve()
            .bodyToMono(Person[].class)
            .block(Duration.ofSeconds(3));

        //DISPLAY PERSONS
        System.out.println("getPersons()");
        for (Person person : persons) {
            System.out.println(person.id + " " + person.name+ " " + person.age);
        }

    }

}
```

MyCommandLineRunner.java

```
package com.ivorononline.springboot_webclient.runners;

import com.ivorononline.springboot_webclient.services.PersonService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyCommandLineRunner implements CommandLineRunner {

    @Autowired private PersonService personService;

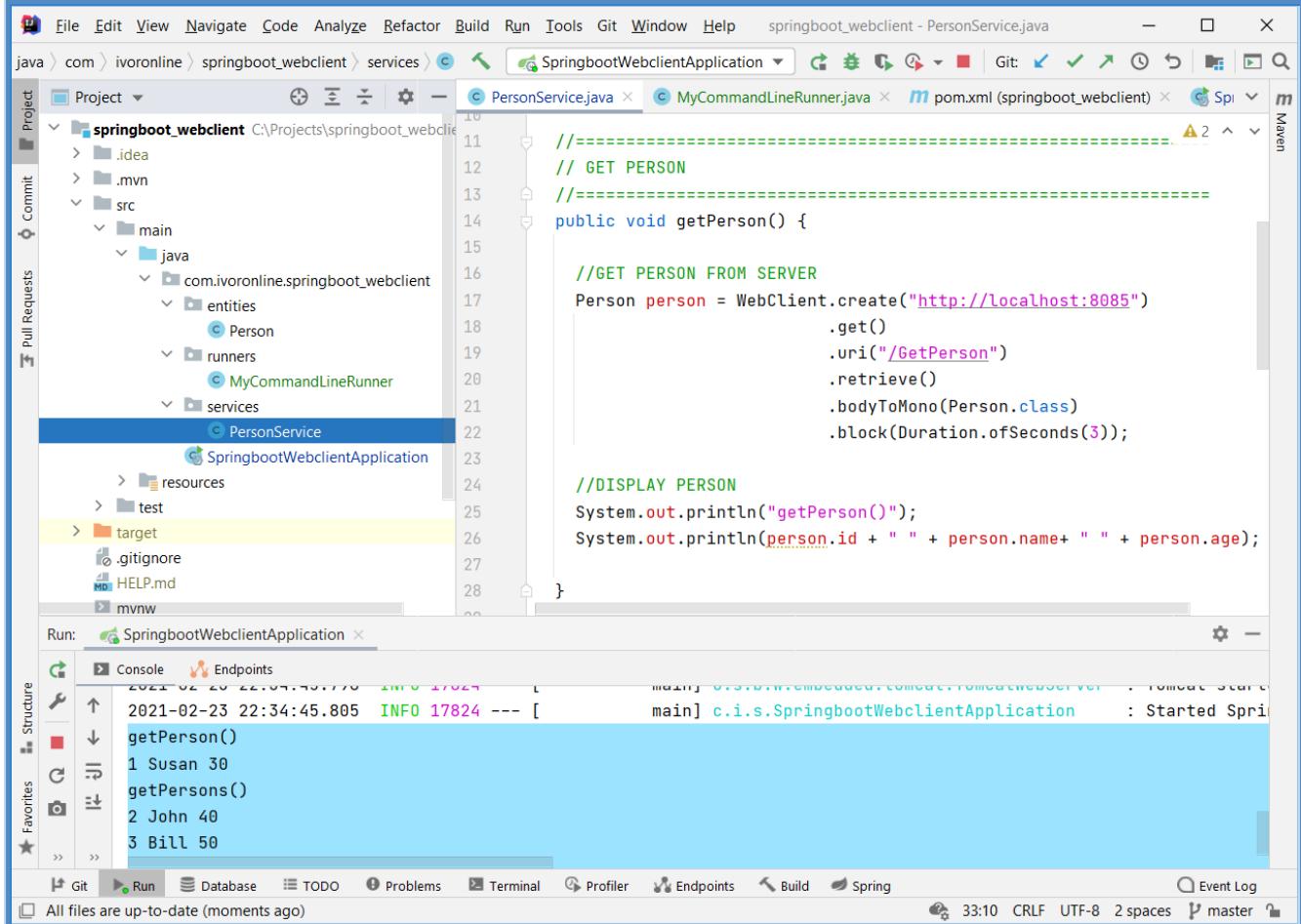
    @Override
    public void run(String... args) throws Exception {
        personService.getPerson();
        personService.getPersons();
    }
}
```

Results

Console

```
getPerson()
1 Susan 30
getPersons()
2 John 40
3 Bill 50
```

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-webflux</artifactId>
    </dependency>

</dependencies>
```

1.2.3 Client - Flux

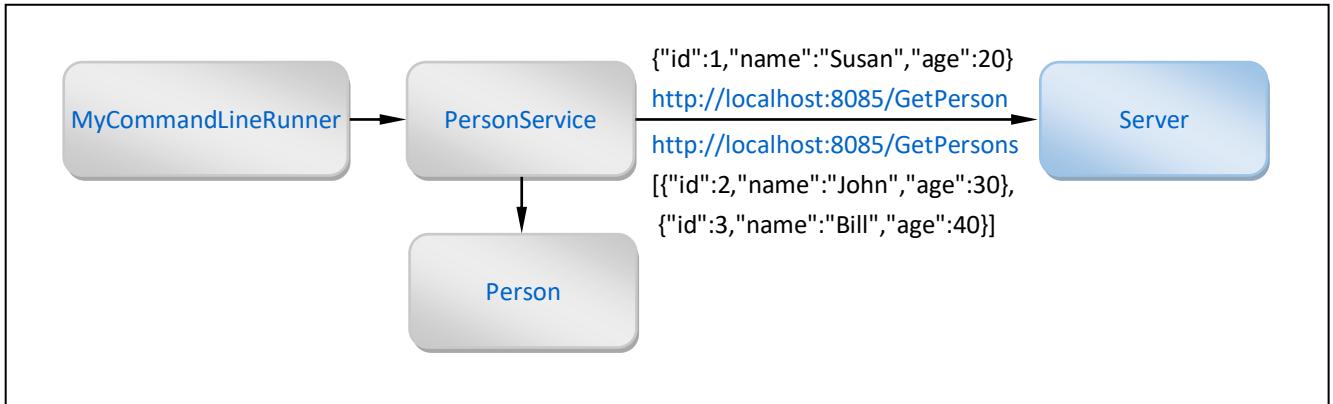
Info

[G] [R]

- This tutorial shows how to use [WebClient](#) Class to call Remote [Server](#) Endpoints that return Person(s).
- This tutorial uses [bodyToFlux\(\)](#) to retrieve Data in a [List<Person>](#).

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server
Web	Spring Reactive Web	Enables WebClient

Procedure

- **Create Project:** controller_returns_text_json (add Spring Boot Starters from the table)
- **Create Package:** entities (inside main package)
 - **Create Interface:** [Person.java](#) (inside package entities)
- **Create Package:** services (inside main package)
 - **Create Class:** [PersonService.java](#) (inside controllers package)
- **Create Package:** runners (inside main package)
 - **Create Class:** [MyCommandLineRunner.java](#) (inside runners package)

Person.java

```
package com.ivoronline.springboot_webclient_server.entities;

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

PersonService.java

```
package com.ivorononline.springboot_webclient_flux.services;

import com.ivorononline.springboot_webclient_flux.entities.Person;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.client.WebClient;

import java.time.Duration;
import java.util.List;

@Component
public class PersonService {

    //=====
    // GET PERSON
    //=====

    public void getPerson() {

        //GET PERSON FROM SERVER
        Person person = WebClient.create("http://localhost:8085")
            .get()
            .uri("/GetPerson")
            .retrieve()
            .bodyToMono(Person.class)
            .block(Duration.ofSeconds(3));

        //DISPLAY PERSON
        System.out.println("getPerson()");
        System.out.println(person.id + " " + person.name+ " " + person.age);

    }

    //=====
    // GET PERSONS
    //=====

    public void getPersons() {

        //GET PERSON FROM SERVER
        List<Person> persons = WebClient.create("http://localhost:8085")
            .get()
            .uri("/GetPersons")
            .retrieve()
            .bodyToFlux(Person.class)
            .collectList()
            .block(Duration.ofSeconds(3));

        //DISPLAY PERSONS
        System.out.println("getPersons()");
        for (Person person : persons) {
            System.out.println(person.id + " " + person.name+ " " + person.age);
        }

    }

}
```

MyCommandLineRunner.java

```
package com.ivoronline.springboot_webclient.runners;

import com.ivoronline.springboot_webclient.services.PersonService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyCommandLineRunner implements CommandLineRunner {

    @Autowired private PersonService personService;

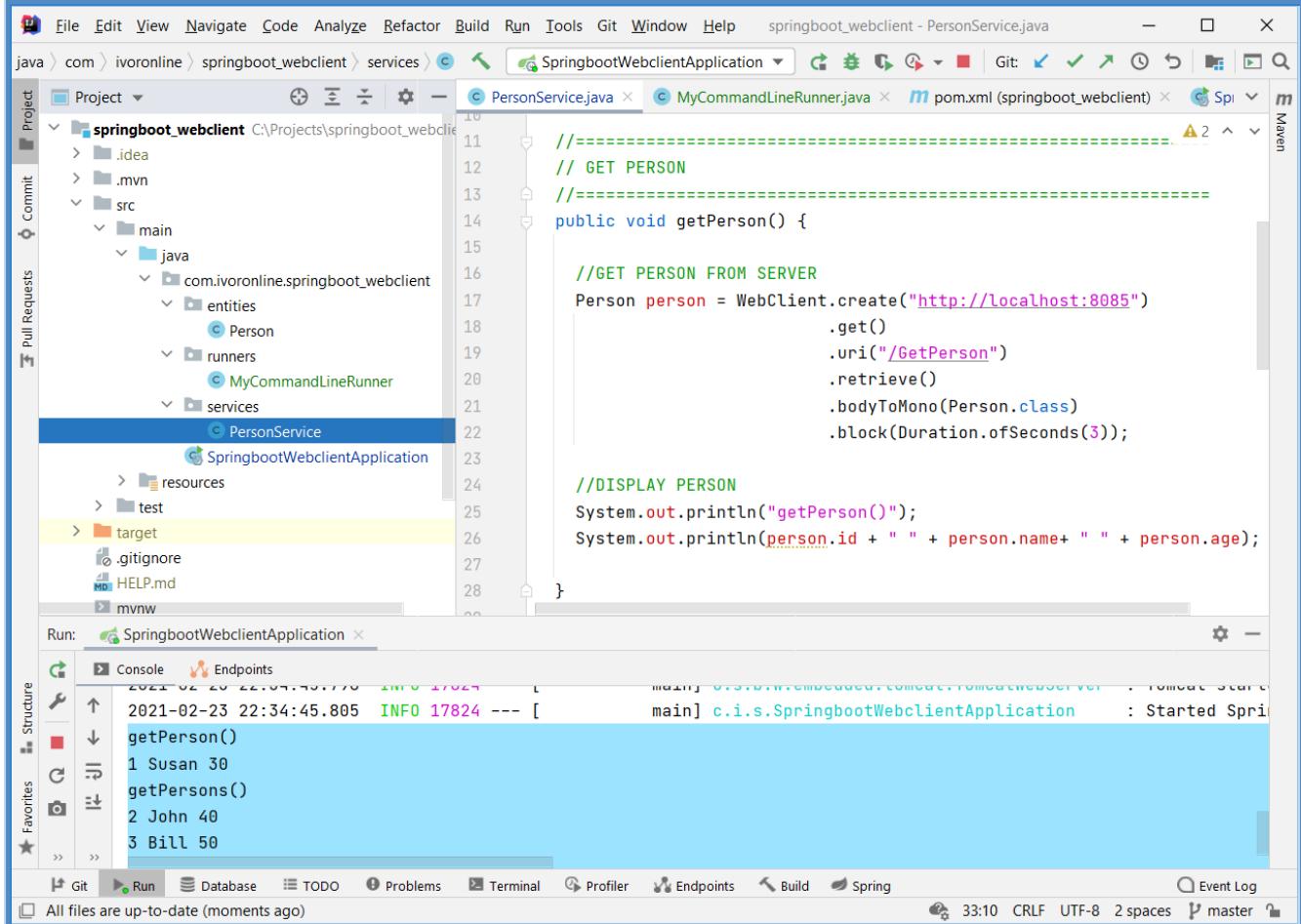
    @Override
    public void run(String... args) throws Exception {
        personService.getPerson();
        personService.getPersons();
    }
}
```

Results

Console

```
getPerson()
1 Susan 30
getPersons()
2 John 40
3 Bill 50
```

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-webflux</artifactId>
    </dependency>

</dependencies>
```

1.3 Tasks

Info

- Following tutorials show how to execute tasks on Startup or at certain intervals.

1.3.1 Startup

Info

[G]

- Following tutorials show how to create Class that implements `CommandLineRunner` to run Task at App startup.
- You simply need to `@Override` its `run()` Method with your custom code.
- You can use `@Order(1)` to specify order in which such Classes (and their `run()` Methods) should be executed.

Application Schema

[Results]



Procedure

- **Create Project:** `springboot_commandlinerunner` (Spring Boot Starters are not used)
- **Create Package:** `runners` (inside main package)
 - Create Class: `MyRunner1.java` (inside runners package)
 - Create Class: `MyRunner2.java` (inside runners package)

MyRunner1.java

```
package com.ivoronline.springboot_commandlinerunner.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component
@Order(1)
public class MyRunner1 implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello from MyRunner1");
    }

}
```

MyRunner2.java

```
package com.ivoronline.springboot_commandlinerunner.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component
@Order(2)
public class MyRunner2 implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello from MyRunner2");
    }

}
```

Results

Console

```
Hello from MyRunner1  
Hello from MyRunner2
```

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_commandlinerunner". The "src/main/java/com/ivoronline/springboot_commandlinerunner/runners" package contains two classes: "MyRunner1" and "MyRunner2".
- Code Editor:** The "MyRunner1.java" file is open, displaying its code:

```
7  @Order(1)  
8  @Component  
9  public class MyRunner1 implements CommandLineRunner {  
10  
11     @Override  
12     public void run(String... args) throws Exception {  
13         System.out.println("Hello from MyRunner1");  
14     }  
15 }  
16 }
```
- Run Tab:** The "SpringbootCommandlinerunnerApplication" configuration is selected.
- Console Tab:** Displays the application logs and output:

```
2021-02-27 12:12:05.578 INFO 25428 --- [           main] s.SpringbootCommandlinerunnerApplication : Starting SpringBootApplication  
2021-02-27 12:12:05.580 INFO 25428 --- [           main] s.SpringbootCommandlinerunnerApplication : No active profile has been configured, falling back to default profiles: default  
2021-02-27 12:12:05.974 INFO 25428 --- [           main] s.SpringbootCommandlinerunnerApplication : Started SpringBootApplication in 0.01 seconds (process info)  
Hello from MyRunner1  
Hello from MyRunner2
```
- Bottom Status Bar:** Shows "Build completed successfully in 3 sec, 546 ms (7 minutes ago)".

1.3.2 Scheduled

Info

[G] [R]

- This tutorial shows how to use **@Scheduled** Annotation to **periodically run methods** (in the future).
- To schedule a Method you just need to use **@Scheduled(fixedDelay=5000)** Annotation.
To enable **@Scheduled** Annotation you need to add **@EnableScheduling** to some **@Configuration** Class.
- Method's **Class** must be declared as Spring Component for Spring to detect it.
In this example this is achieved also by **@Configuration** (so **@Configuration** has dual role in this example).

@Scheduled Parameters

(milliseconds)

```
@Scheduled(fixedRate = 5000)          //Measured from the START of previous task
@scheduled(fixedDelay = 5000)          //Measured from the END   of previous task
@scheduled(fixedDelay = 5000, initialDelay = 1000) //Delay before first execution
@scheduled(cron="*/5 * * * * MON-FRI") //Execute on weekdays
```

Application Schema

[Results]

A rounded rectangular button with a light gray background and a thin black border. The text "MyTasks" is centered inside in a blue font.

Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	To keep the Application running

Procedure

- Create Project:** springboot_taskscheduler.tasks (add Spring Boot Starters from the table)
- Create Package:** tasks (inside main package)
 - Create Class:** **MyTasks.java** (inside runners package)

MyTasks.java

```
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;

@Configuration
@EnableScheduling
public class MyTasks {

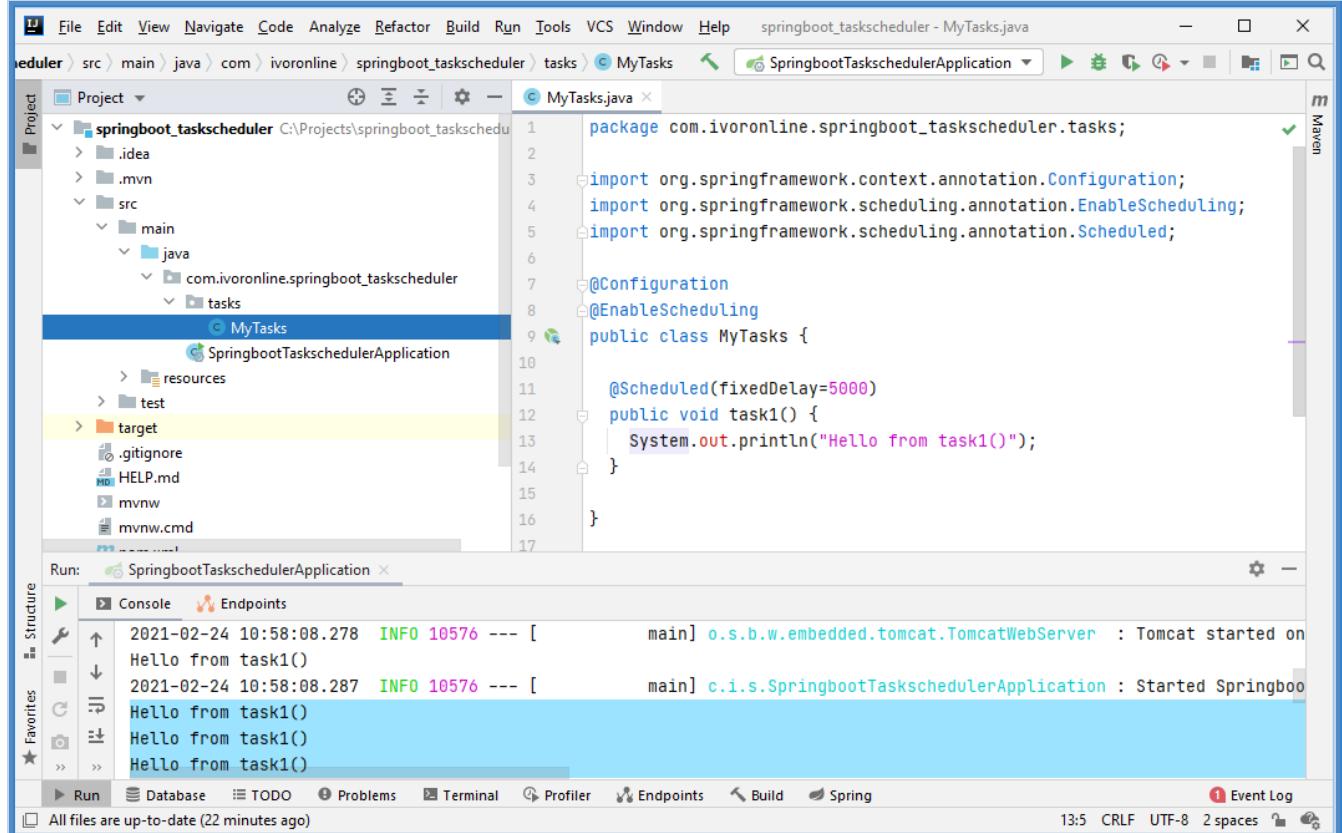
    @Scheduled(fixedDelay=5000)
    public void task1() {
        System.out.println("Hello from task1()");
    }
}
```

Results

Console

```
Hello from task1()  
Hello from task1()
```

Application Structure



pom.xml

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

1.4 Custom Banner

Info

[G] [R]

- This tutorial shows how to create Custom Banner that is shown in the Console when Application starts.
You can use different Web Sites to generate such banner from entered text (tutorial uses patorjk.com).
- By default file with banner should be located in `src/main/resources/banner.txt`.
If default location and name are not used, custom parameter can be specified inside `application.properties`.

`application.properties`

(when using custom path and name)

```
# CUSTOM PATH AND NAME
spring.banner.image.location = classpath:/src/main/resources/mybanner.txt
```

Design Banner

- <http://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20>
- Type Custom Text: IVORONLINE.COM
- Test All
- Select & Copy (below to font you want)

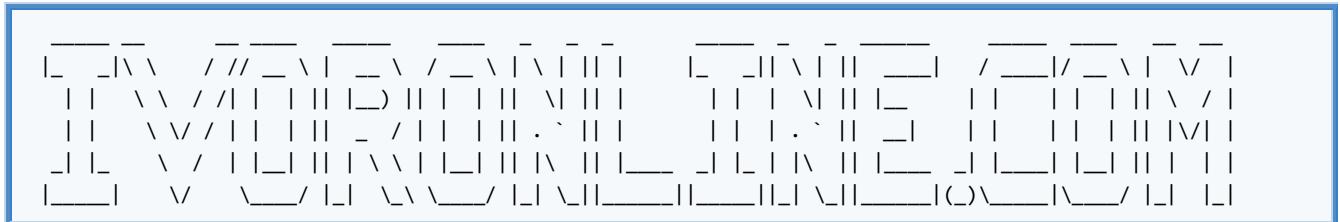
Design Banner

The screenshot shows the 'Text to ASCII Art Generator (TAAC)' application window. At the top, the URL is <http://patorjk.com/software/taag/#p=testall&f=Graffiti&t=IVORONLINE.COM>. The main interface has a title bar 'Main Controls - *FIGlet and AOL Macro Fonts Supported*'. It includes dropdowns for 'Font' (set to 'Graffiti'), 'Character Width' (Default), and 'Character Height' (Default). A text input field contains 'IVORONLINE.COM'. Below the input is a preview area showing the ASCII art representation of the text. To the right of the preview is a sidebar titled 'Other Stuff From patorjk.com That You Might Like:' with links to Typing Speed Test, Keyboard Layout Analyzer, Text Color Fader, Snake Game, My Photography Site, and Main Page. At the bottom of the preview area, there's a section for 'Font Name: Big Money-sw' with 'Use Font' and 'Select & Copy' buttons. Below this is another preview area for 'Font Name: Big' with similar controls. At the very bottom of the window is a footer with a 'Share Link' button and the 'patorjk.com' logo.

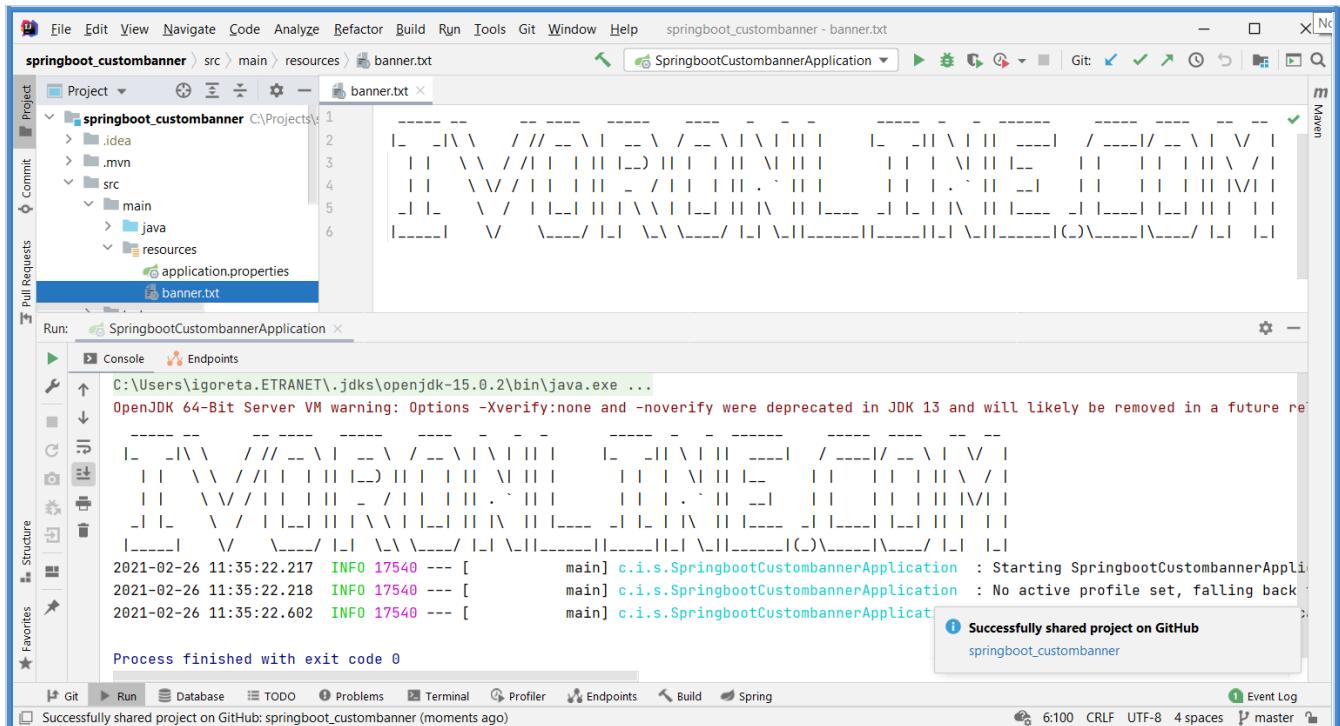
Add Banner

- [Create Project:](#) `springboot_cuatombanner` (there are no Spring Boot Starters)
- Create file: `src/main/resources/banner.txt`
- (Paste below Banner)
- Run

`banner.txt`



Results



2 Filters & Interceptors

Info

- Following tutorials show how to use Filters & Interceptors.

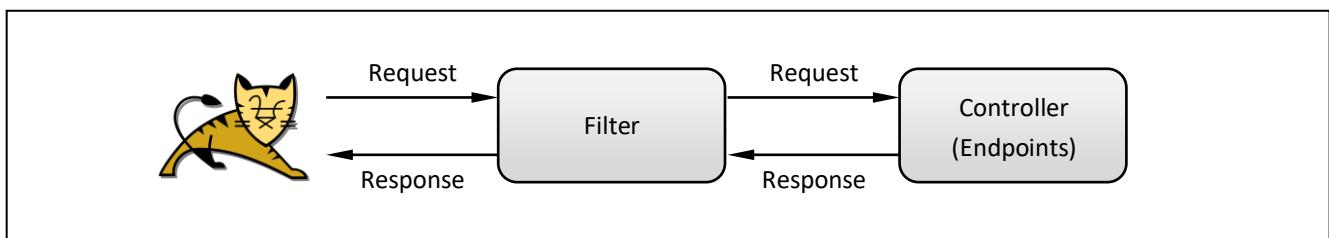
2.1 Filters

Info

[R]

- **Filter** is **Java Class** that is used to **intercept HTTP Requests and Responses**.
- Filters perform additional operations before sending
 - HTTP Request to the Controller (it can modify HTTP Request before it gets to Controller)
 - HTTP Response from the Controller (it can modify HTTP Response before it is returned to the User)
- Every Filter is **called twice** (because HTTP Requests and Responses pass through the same Filters)
 - first during HTTP Request
 - then during HTTP Response
- Your custom **Filter Class** needs
 - to **implement Filter Interface** (so that Spring would know how to use it)
 - to **@Override doFilter() Method** (contains actual useful code performed by the Filter)
 - **@Component Annotation** (for Spring to detect it, create Filter Object, add it to Filter Chain)
- Filters are used for
 - **Security** (create Authentication Object from JWT Authorities)
 - **Logging** (log HTTP Requests/Responses: User, Endpoint, HTTP Response)
 - **Error Handling** (give feedback to User if HTTP Request has invalid Parameters/Format)

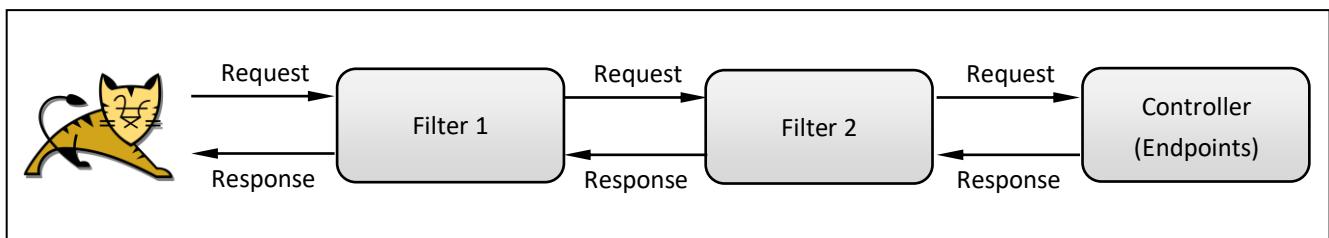
Filter is called twice



Filter Chain

- **Filter Chain** is ordered set of Filters that are
 - called in **sequence** before sending HTTP Request to the Controller
 - called in **reverse order** before sending HTTP Response from the Controller
- Filter Chain allows you to organize complex Filter Logic into multiple Filter Classes (for easier maintenance).

Filters are called in reverse order during HTTP Response



`chain.doFilter(request, response);`

- Call to `chain.doFilter(request, response)` Method is used to separate code that is execute during Request or Response.
- To return before reaching Controller don't call `chain.doFilter(request, response)` Method.

Request and Response Code

- Call to `chain.doFilter(request, response)` represents border between code that is execute during Request or Response
 - Code before `chain.doFilter(request, response)` is executed during HTTP Request
 - Code after `chain.doFilter(request, response)` is executed during HTTP Response
- Method `chain.doFilter()` calls
 - `dofilter()` Method of the next filter in chain
 - Endpoint if there are no more filters in chain
- When end of `dofilter()` Method is reached code returns to the next line after the call to `chain.doFilter()` in previous Filter
 - causing subsequent code to be executed during Http Response (code after call to `chain.doFilter()`)
 - causing filters to be executed in reverse order during Http Response (but only code after call to `chain.doFilter()`)

MyFilter.java

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {  
    System.out.println("MyFilter: Code for HTTP Request"); //CODE FOR HTTP REQUEST  
    chain.doFilter(request, response); //DIVIDES CODE FOR HTTP REQUEST AND RESPONSE  
    System.out.println("MyFilter: Code for HTTP Response"); //CODE FOR HTTP RESPONSE  
}
```

Return before reaching Controller

- If you don't call `chain.doFilter(request, response)` then Filter returns to previous Filter after its call to `chain.doFilter()`. This way Filter can stop propagation of HTTP Request to subsequent Filters or Controller and initiate HTTP Response.
- For instance Filter can analyze HTTP Request Parameters and if they are not valid
 - Filter can prevent Request from reaching Controller (or subsequent Filters down the Filter chain)
 - return HTTP Response instructing User which Parameters were wrong

2.1.1 Create Filter - Using Implements Filter

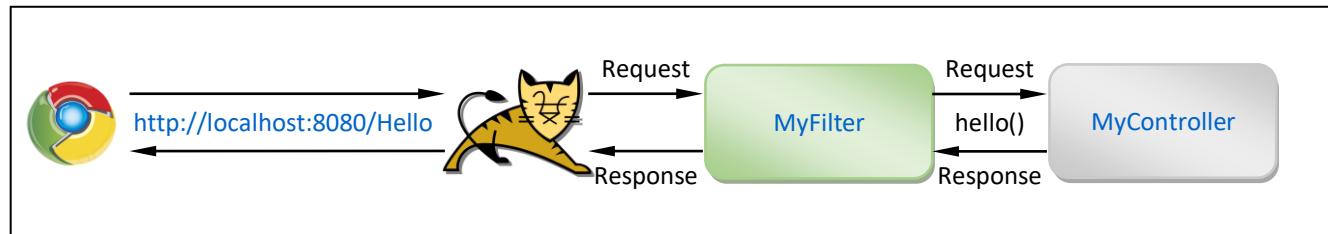
Info

[G]

- This tutorial shows how to create Filter Class by using [implements Filter](#).
- Code before `chain.doFilter(request, response)` is executed during HTTP Request.
- Code after `chain.doFilter(request, response)` is executed during HTTP Response.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- Create Package:** controllers (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)
- Create Package:** filters (inside main package)
 - Create Class:** `MyFilter.java` (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_filter_create.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() {
        System.out.println("Controller: Code from Controller");
        return "Hello from Controller";
    }
}
```

MyFilter.java

```
package com.ivoronline.springboot_filter_create.filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import org.springframework.stereotype.Component;

@Component
public class MyFilter implements Filter {

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

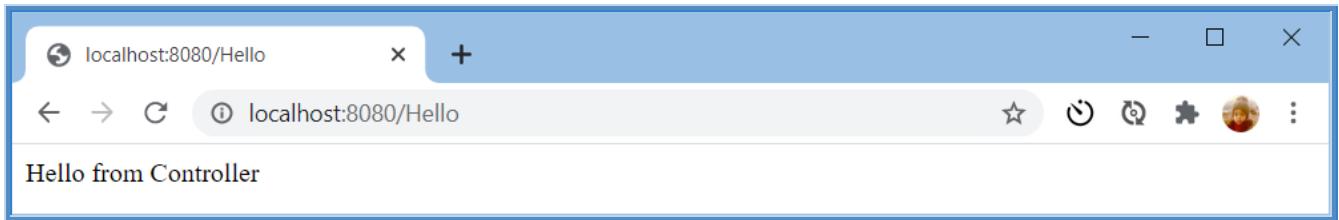
        //THIS CODE IS EXECUTED DURING HTTP REQUEST
        System.out.println("MyFilter : Code for HTTP Request");

        //CALLS
        // - doFilter() METHOD OF THE NEXT FILTER IN CHAIN
        // - Endpoint IF THERE ARE NO MORE FILTERS IN CHAIN
        //RETURNS FROM THIS METHOD DURING HTTP RESPONSE
        // - CAUSING SUBSEQUENT CODE TO BE EXECUTED DURING HTTP RESPONSE
        // - CAUSING FILTERS TO BE EXECUTED IN REVERSE ORDER DURING HTTP RESPONSE (BUT ONLY BELOW CODE)
        chain.doFilter(request, response); //DIVIDES HTTP REQUEST AND RESPONSE CODE

        //THIS CODE IS EXECUTED DURING HTTP RESPONSE
        System.out.println("MyFilter : Code for HTTP Response");
    }
}
```

Results

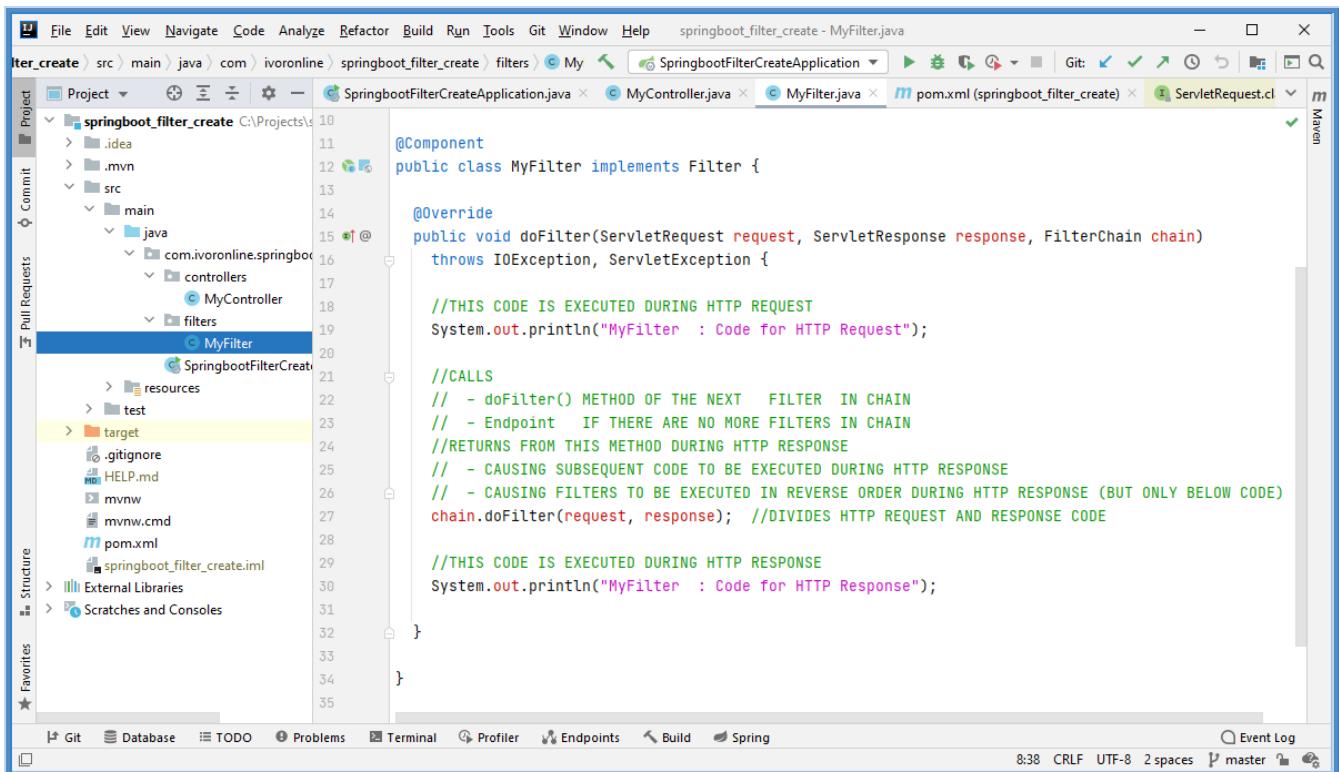
<http://localhost:8080>Hello>



Console

```
MyFilter : Code for HTTP Request
Controller: Code from Controller
MyFilter : Code for HTTP Response
```

Application Structure



pom.xml

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

2.1.2 Create Filter - Using Extends OncePerRequestFilter

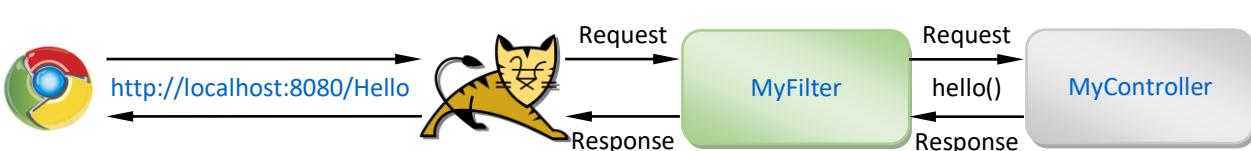
Info

[G]

- This tutorial shows how to create Filter Class by using `extends OncePerRequestFilter`.
- This gives you access to `HttpServletRequest` and `HttpServletResponse`.
They have additional Properties compared to `ServletRequest` and `ServletResponse` from Create Filter - Implements Filter.
- Code before `chain.doFilter(request, response)` is executed during HTTP Request.
Code after `chain.doFilter(request, response)` is executed during HTTP Response.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- **Create Package:** controllers (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Package:** filters (inside main package)
 - **Create Class:** `MyFilter.java` (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_filter_extends_onceperrequestfilter.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() {
        System.out.println("Controller: Code from Controller");
        return "Hello from Controller";
    }
}
```

MyFilter.java

```
package com.ivorononline.springboot_filter_extends_onceperrequestfilter.filters;

import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class MyFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws IOException, ServletException {

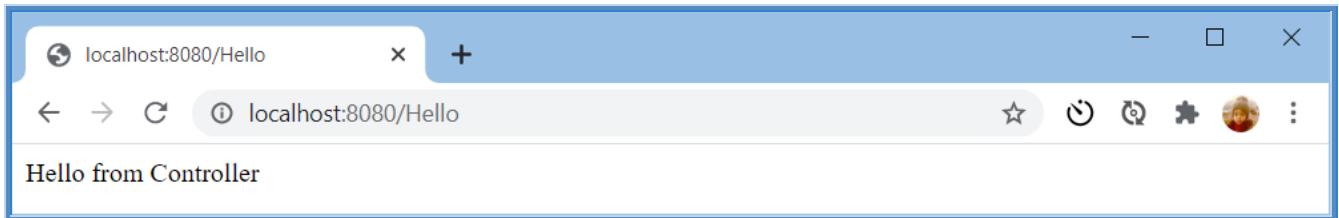
        //THIS CODE IS EXECUTED DURING HTTP REQUEST
        System.out.println("MyFilter : Code for HTTP Request");

        //CALLS
        // - doFilter() METHOD OF THE NEXT FILTER IN CHAIN
        // - Endpoint IF THERE ARE NO MORE FILTERS IN CHAIN
        //RETURNS FROM THIS METHOD DURING HTTP RESPONSE
        // - CAUSING SUBSEQUENT CODE TO BE EXECUTED DURING HTTP RESPONSE
        // - CAUSING FILTERS TO BE EXECUTED IN REVERSE ORDER DURING HTTP RESPONSE (BUT ONLY BELOW CODE)
        chain.doFilter(request, response); //DIVIDES HTTP REQUEST AND RESPONSE CODE

        //THIS CODE IS EXECUTED DURING HTTP RESPONSE
        System.out.println("MyFilter : Code for HTTP Response");
    }
}
```

Results

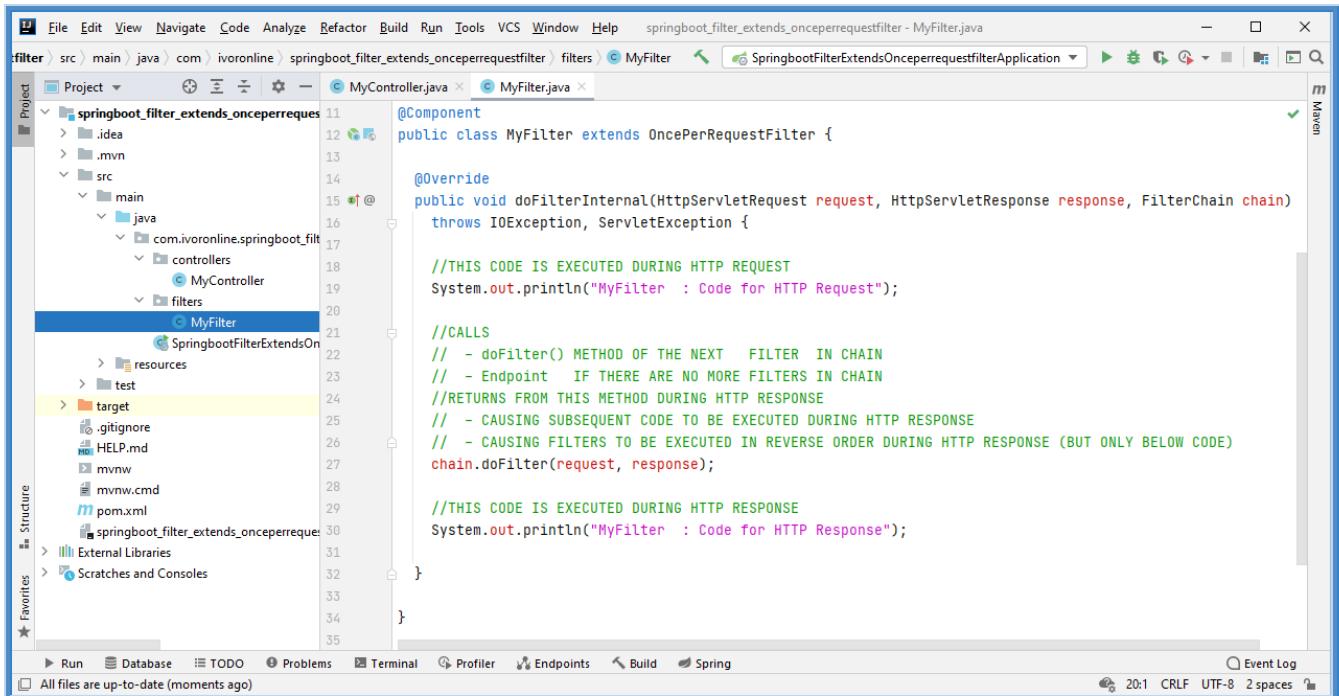
<http://localhost:8080>Hello>



Console

```
MyFilter : Code for HTTP Request
Controller: Code from Controller
MyFilter : Code for HTTP Response
```

Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

2.1.3 Create Filter Chain

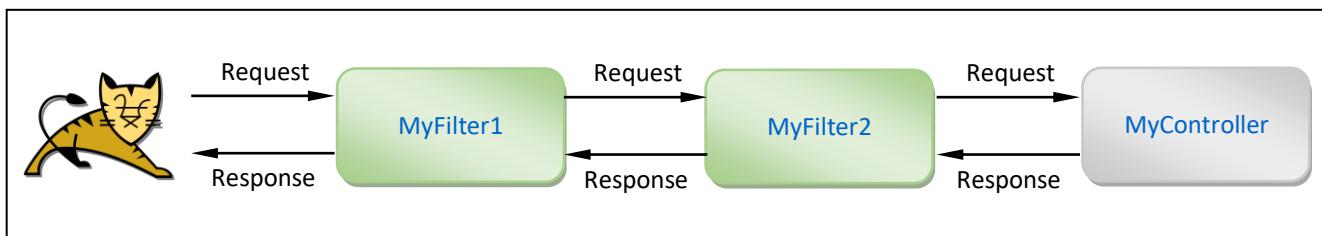
Info

[G]

- This tutorial shows how to create Filter Chain by creating two Filter Classes.
- We will use **@Order(1)** to specify in which order Filters should be executed.
They will be executed in that order during **HTTP Request**.
They will be executed in reverse order during **HTTP Response**.
- Code before `chain.doFilter(request, response)` is executed during **HTTP Request**.
Code after `chain.doFilter(request, response)` is executed during **HTTP Response**.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- **Create Package:** controllers (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Package:** filters (inside main package)
 - **Create Class:** `MyFilter1.java` (inside controllers package)
 - **Create Class:** `MyFilter2.java` (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_filter_create.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() {
        System.out.println("Controller: Code from Controller");
        return "Hello from Controller";
    }
}
```

MyFilter1.java

```
package com.ivoronline.springboot_filter_chain.filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import org.springframework.stereotype.Component;

@Component
public class MyFilter1 implements Filter {

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

        //THIS CODE IS EXECUTED DURING HTTP REQUEST
        System.out.println("MyFilter1 : Code for HTTP Request");

        //DIVIDES HTTP REQUEST AND RESPONSE CODE
        chain.doFilter(request, response);

        //THIS CODE IS EXECUTED DURING HTTP RESPONSE
        System.out.println("MyFilter1 : Code for HTTP Response");
    }
}
```

MyFilter2.java

```
package com.ivoronline.springboot_filter_chain.filters;

import org.springframework.stereotype.Component;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.io.IOException;

@Component
public class MyFilter2 implements Filter {

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

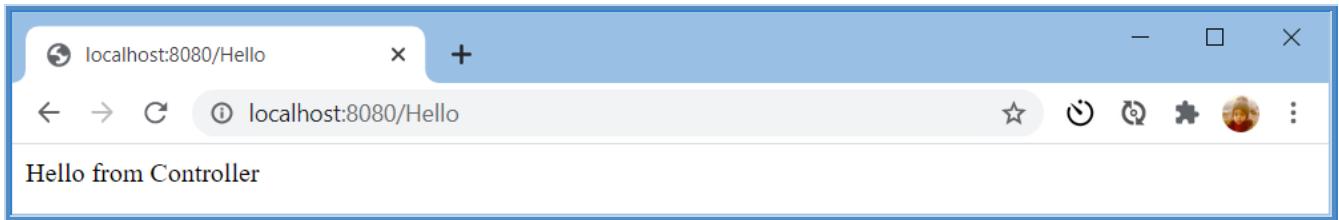
        //THIS CODE IS EXECUTED DURING HTTP REQUEST
        System.out.println("MyFilter2 : Code for HTTP Request");

        //DIVIDES HTTP REQUEST AND RESPONSE CODE
        chain.doFilter(request, response);

        //THIS CODE IS EXECUTED DURING HTTP RESPONSE
        System.out.println("MyFilter2 : Code for HTTP Response");
    }
}
```

Results

<http://localhost:8080>Hello>



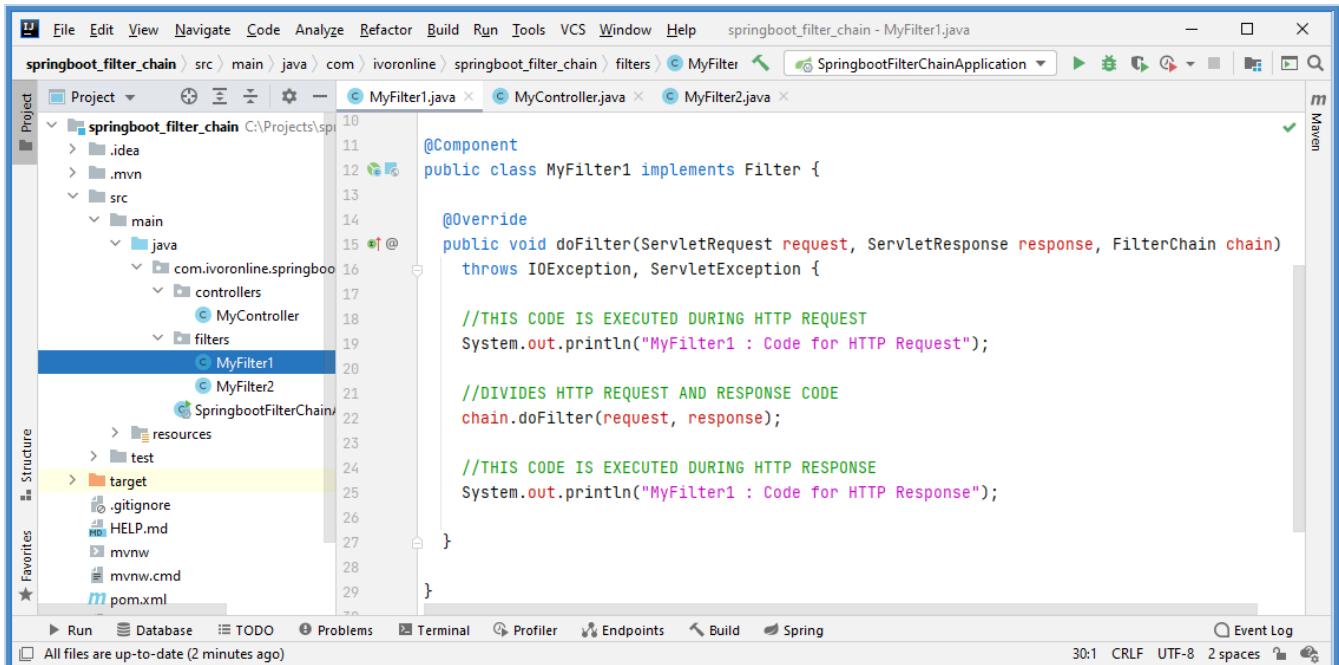
Console

```
MyFilter1 : Code for HTTP Request  
MyFilter2 : Code for HTTP Request
```

```
Controller: Code from Controller
```

```
MyFilter2 : Code for HTTP Response  
MyFilter1 : Code for HTTP Response
```

Application Structure



pom.xml

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

2.1.4 Block Requests

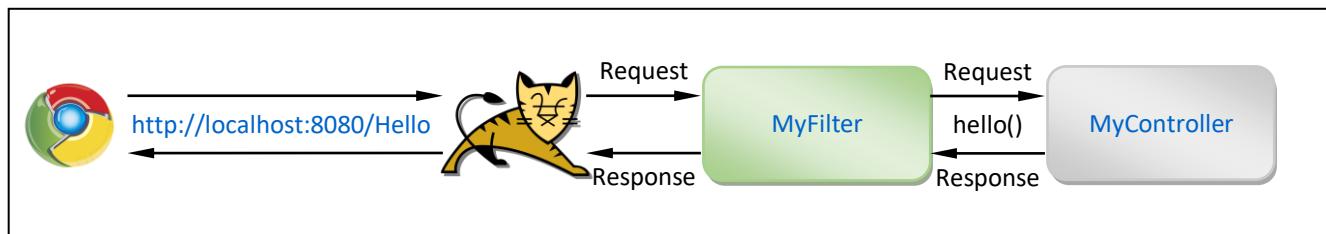
Info

[G]

- This tutorial shows how to use Filter to block HTTP Requests if they have wrong format (Request will not reach Controller)
- If `username` Parameter doesn't exist, Filter returns error message to the User and doesn't forward Request to Controller.
- This is achieved by simply `returning` from Filter, instead of calling `chain.doFilter(request, response)` to forward Request.
- Filter generates Response by calling `response.getWriter().println("Parameter username missing")`.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

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

MyController.java

```
package com.ivoronline.springboot_filter_blockrequests.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@RequestParam String username) {
        System.out.println("Hello from Controller");
        return "Hello " + username;
    }
}
```

MyFilter.java

```
package com.ivorononline.springboot_filter_blockrequests.filters;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class MyFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

        //LOG
        System.out.print("CHECKING REQUEST PARAMETERS: " + request.getQueryString());

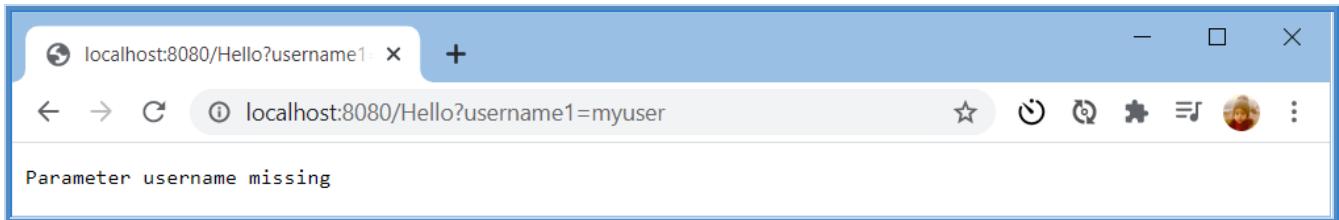
        //GET REQUESTS PARAMETERS
        String parameterUsername = request.getParameter("username");

        //CHECK REQUESTS PARAMETERS
        if(parameterUsername == null) {
            System.out.println(" (Parameter username missing)");
            response.getWriter().println("Parameter username missing");
            return;
        }
        System.out.println(" (Parameter username found)");

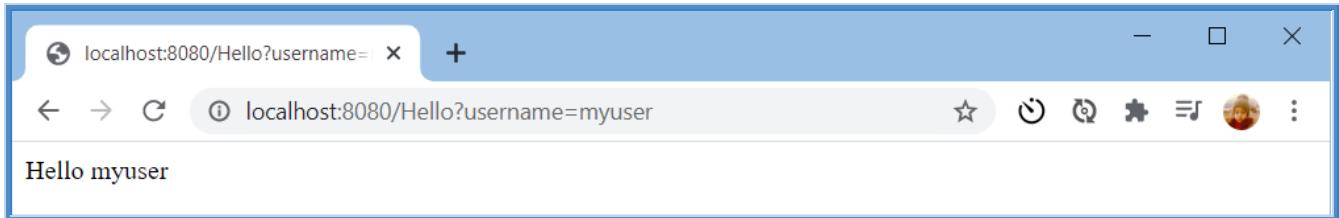
        //FORWARD TO CONTROLLER
        chain.doFilter(request, response);
    }
}
```

Results

<http://localhost:8080>Hello?username1=myuser>



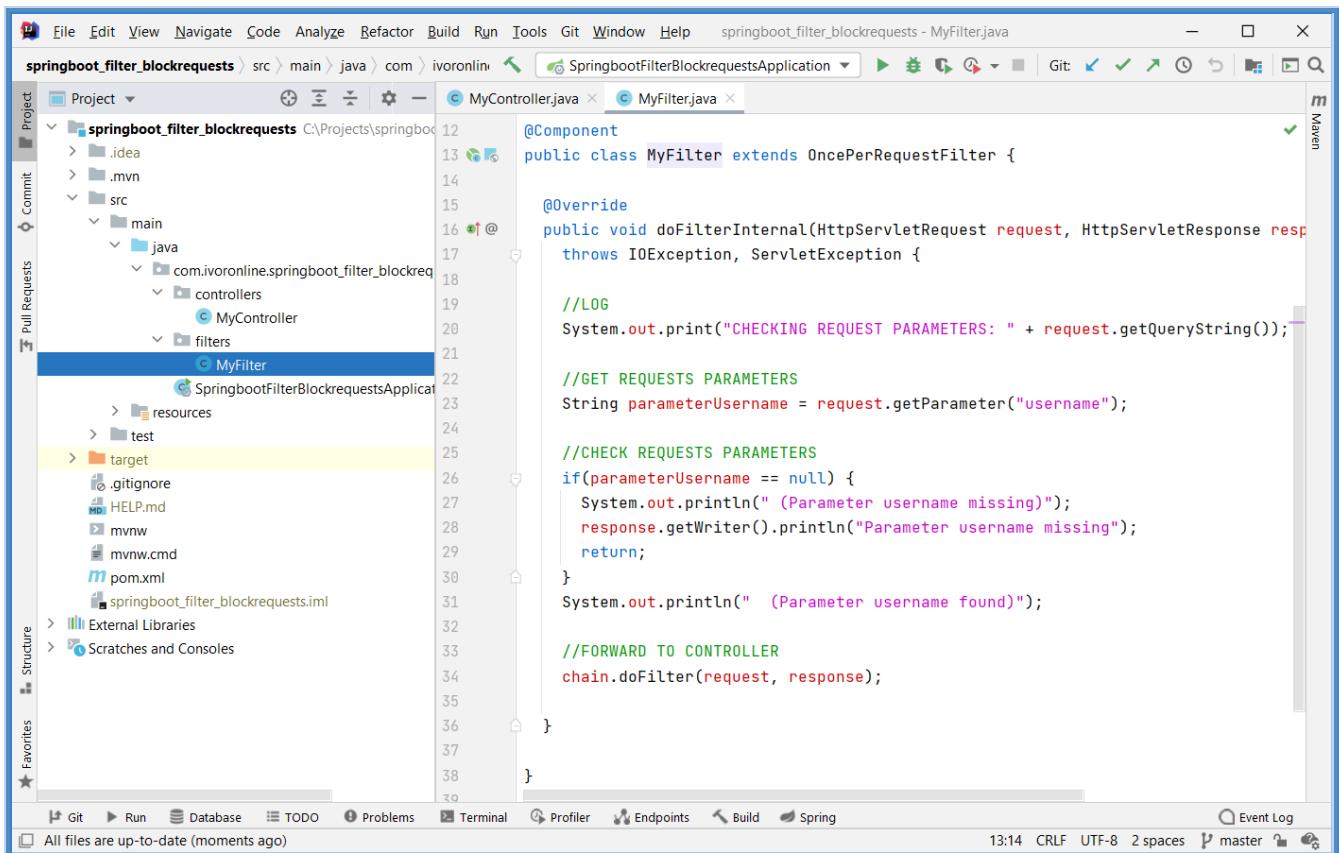
<http://localhost:8080>Hello?username=myuser>



Console

```
CHECKING REQUEST PARAMETERS: username1=myuser (Parameter username missing)
CHECKING REQUEST PARAMETERS: username=myuser (Parameter username found)
Hello from Controller
```

Application Structure



pom.xml

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

2.1.5 HTTP Request Parameters - Get

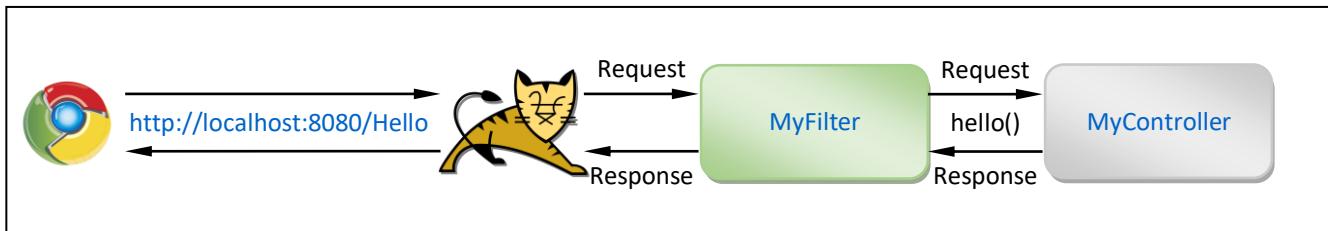
Info

[G]

- This tutorial shows how to use Filter to get HTTP Request/Response Parameters.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

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

MyController.java

```
package com.ivoronline.springboot_filter_gethttprequestresponseparameters.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("/Public/Hello")
    public String publicHello() {
        System.out.println("Controller: Code from Controller");
        return "Hello from Controller";
    }
}
```

MyFilter.java

```
package com.ivorononline.springboot_filter_gethttprequestresponseparameters.filters;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class MyFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

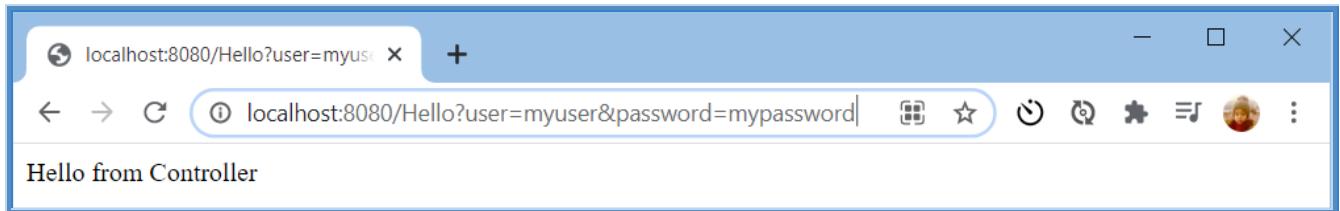
        //LOG HTTP REQUEST PARAMETERS: http://localhost:8080>Hello?user=myuser&password=mypassword
        System.out.println("\nREQUEST PARAMETERS: =====");
        System.out.println("RequestURL: " + request.getRequestURL()); //http://localhost:8080/Public/Hello
        System.out.println("RequestURI: " + request.getRequestURI()); // /Public/Hello
        System.out.println("ServletPath: " + request.getServletPath()); // /Public/Hello
        System.out.println("QueryString: " + request.getQueryString()); // user=myuser&password=mypassword
        System.out.println("Parameter: " + request.getParameter ("user")); // myuser
        System.out.println("Method: " + request.getMethod()); // GET
        System.out.println("Protocol: " + request.getProtocol()); // HTTP/1.1
        System.out.println("ServerName: " + request.getServerName()); // localhost
        System.out.println("ServerPort: " + request.getServerPort()); // 8080

        //DIVIDES HTTP REQUEST AND RESPONSE CODE
        chain.doFilter(request, response);

        //LOG HTTP RESPONSE PARAMETERS
        System.out.println("\nRESPONSE PARAMETERS: =====");
        System.out.println("Status: " + response.getStatus()); //200
        System.out.println("ContentType: " + response.getContentType()); //text/html;charset=UTF-8
    }
}
```

Results

<http://localhost:8080>Hello?user=myuser&password=mypassword>

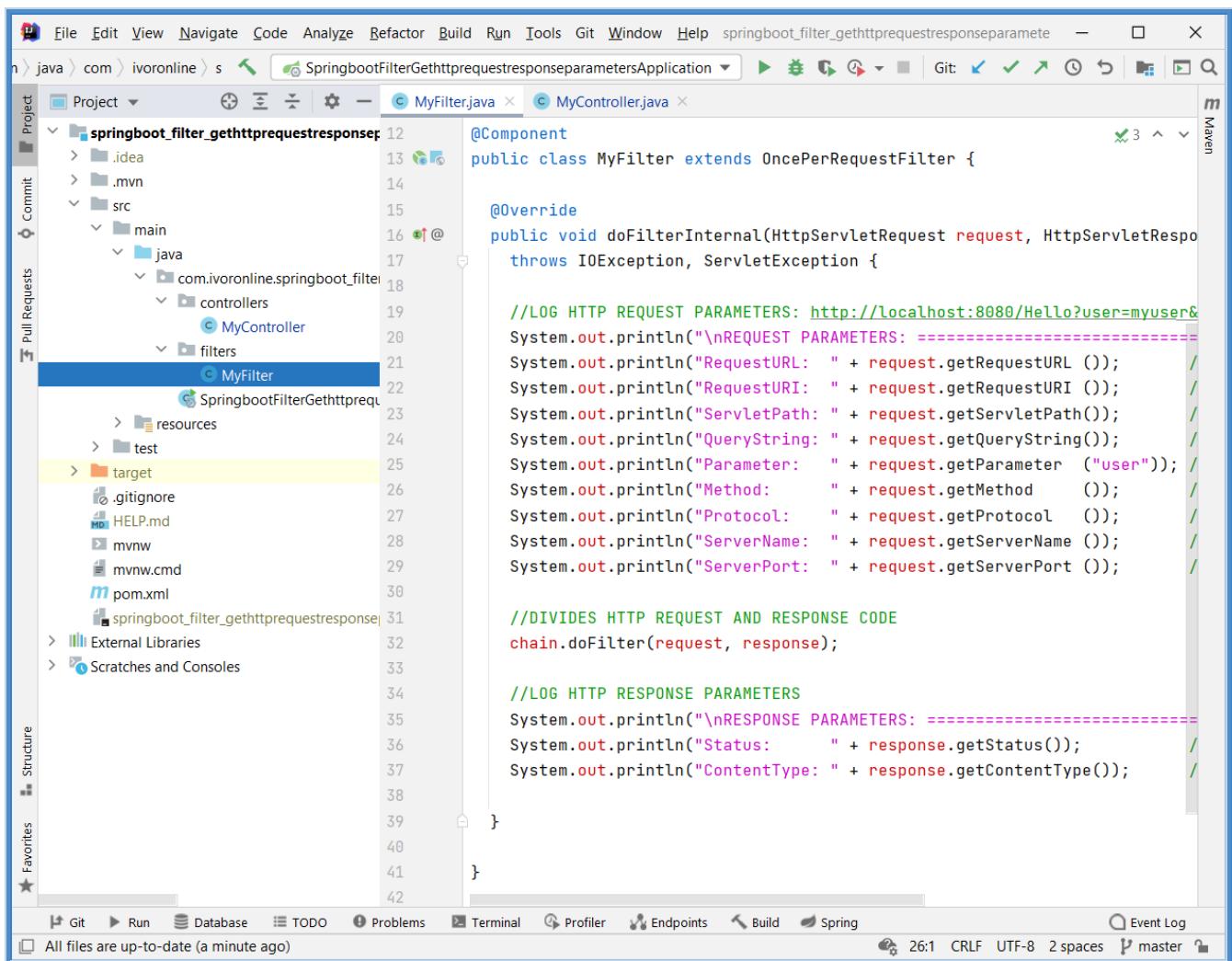


Console

```
REQUEST PARAMETERS: =====
RequestURL: http://localhost:8080>Hello
RequestURI: /Hello
ServletPath: /Hello
QueryString: user=myuser&password=mypassword
Parameter: myuser
Method: GET
Protocol: HTTP/1.1
ServerName: localhost
ServerPort: 8080

RESPONSE PARAMETERS: =====
Status: 200
ContentType: text/html;charset=UTF-8
```

Application Structure



pom.xml

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

2.1.6 HTTP Request Parameters - Add

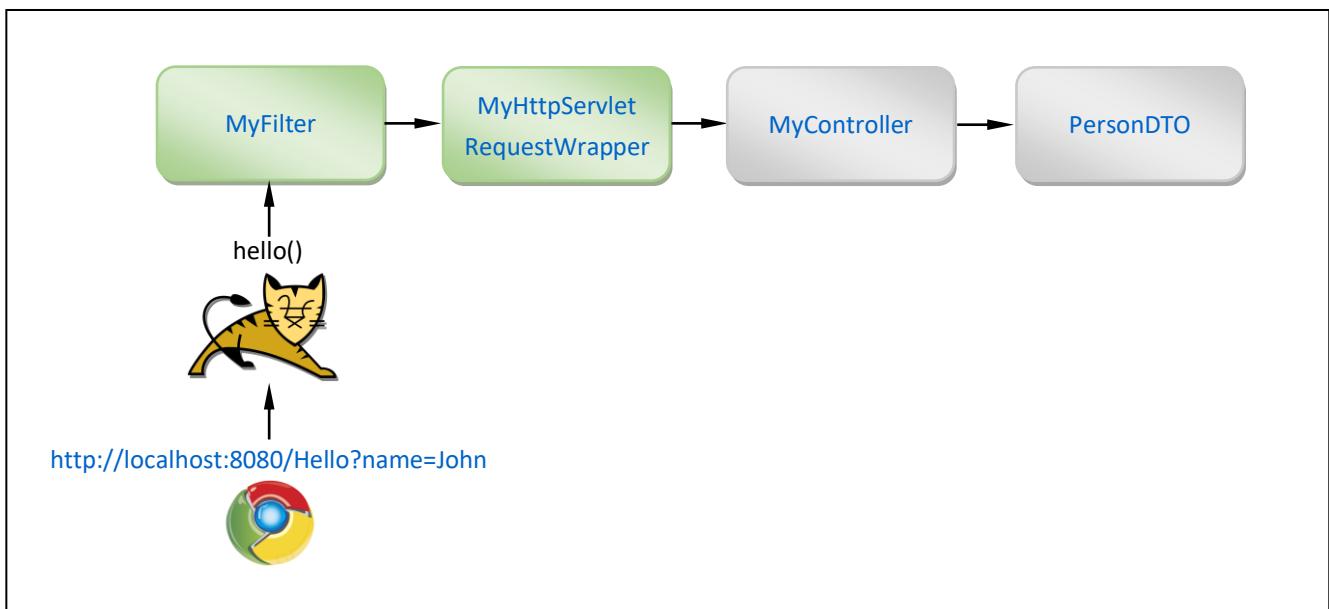
Info

[G]

- This tutorial shows how to "virtually" add HTTP Request Parameters by
 - creating new Request (that extends HttpServletRequestWrapper)
 - overriding Methods that return HTTP Request Parameters (getParameterNames, getParameterValues)
- Then we will load HTTP Request Parameters into **PersonDTO** Properties to confirm that they are visible to Controller.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_filter_addrequestparameters` (add Spring Boot Starters from the table)
- Create Package: `DTO` (inside main package)
 - Create Class: `PersonDTO.java` (inside DTO package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `filters` (inside main package)
 - Create Class: `MyHttpServletRequestWrapper.java` (inside filters package)
 - Create Class: `MyFilter.java` (inside filters package)

PersonDTO.java

```
package com.ivornline.springboot_filter_addrequestparameters.DTO;

public class PersonDTO {

    //PROPERTIES
    public String name;           //EXISTING REQUEST PARAMETER
    public String age;            //ADDED REQUEST PARAMETER

    //SETTERS (Used for Deserialization)
    public void setName(String name) { this.name = name; }
    public void setAge (String age ) { this.age  = age; }

}
```

MyController.java

```
package com.ivornline.springboot_filter_addrequestparameters.controllers;

import com.ivornline.springboot_filter_addrequestparameters.DTO.PersonDTO;
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(PersonDTO personDTO) {
        return personDTO.name + " is " + personDTO.age + " years old";
    }

}
```

MyHttpServletRequestWrapper.java

```
package com.ivornline.springboot_filter_addrequestparameters.filters;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;

public class MyHttpServletRequestWrapper extends HttpServletRequestWrapper {

    //PARAMETER MAP WITH ORIGINAL AND ADDITIONAL REQUEST PARAMETERS
    Map<String, String[]> extendedParameterMap = new HashMap<>();

    //=====
    // CONSTRUCTOR
    //=====
    public MyHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);

        //CREATE ADDITIONAL REQUEST PARAMETERS
        String[] age = {"20"};

        //ADD ADDITIONAL REQUEST PARAMETERS TO NEW MAP
        extendedParameterMap.put("age", age);

        //ADD EXISTING REQUEST PARAMETERS TO NEW MAP
        extendedParameterMap.putAll(request.getParameterMap());
    }

    //=====
    // GET PARAMETER NAMES
    //=====
    @Override
    public Enumeration<String> getParameterNames() {
        return Collections.enumeration(extendedParameterMap.keySet());
    }

    //=====
    // GET PARAMETER VALUES
    //=====
    @Override
    public String[] getParameterValues(String name) {
        return extendedParameterMap.get(name);
    }
}
```

MyFilter.java

```
package com.ivornline.springboot_filter_addrequestparameters.filters;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class MyFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

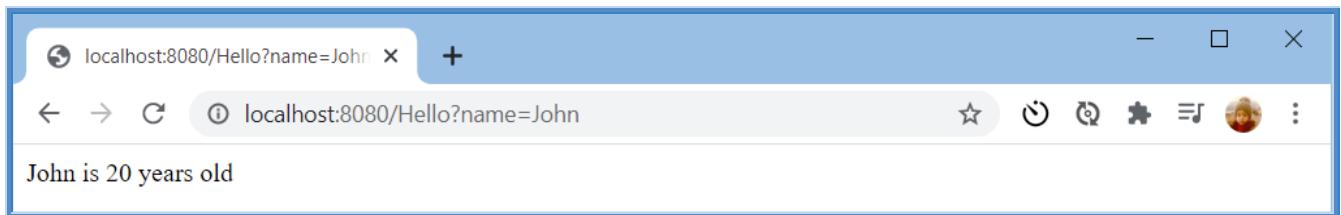
        //CREATE REPLACEMENT REQUEST
        HttpServletRequest myHttpServletRequestWrapper = new MyHttpServletRequestWrapper(request);

        //FORWARD REPLACEMENT REQUEST (With additional Request Parameters)
        chain.doFilter(myHttpServletRequestWrapper, response);
    }
}
```

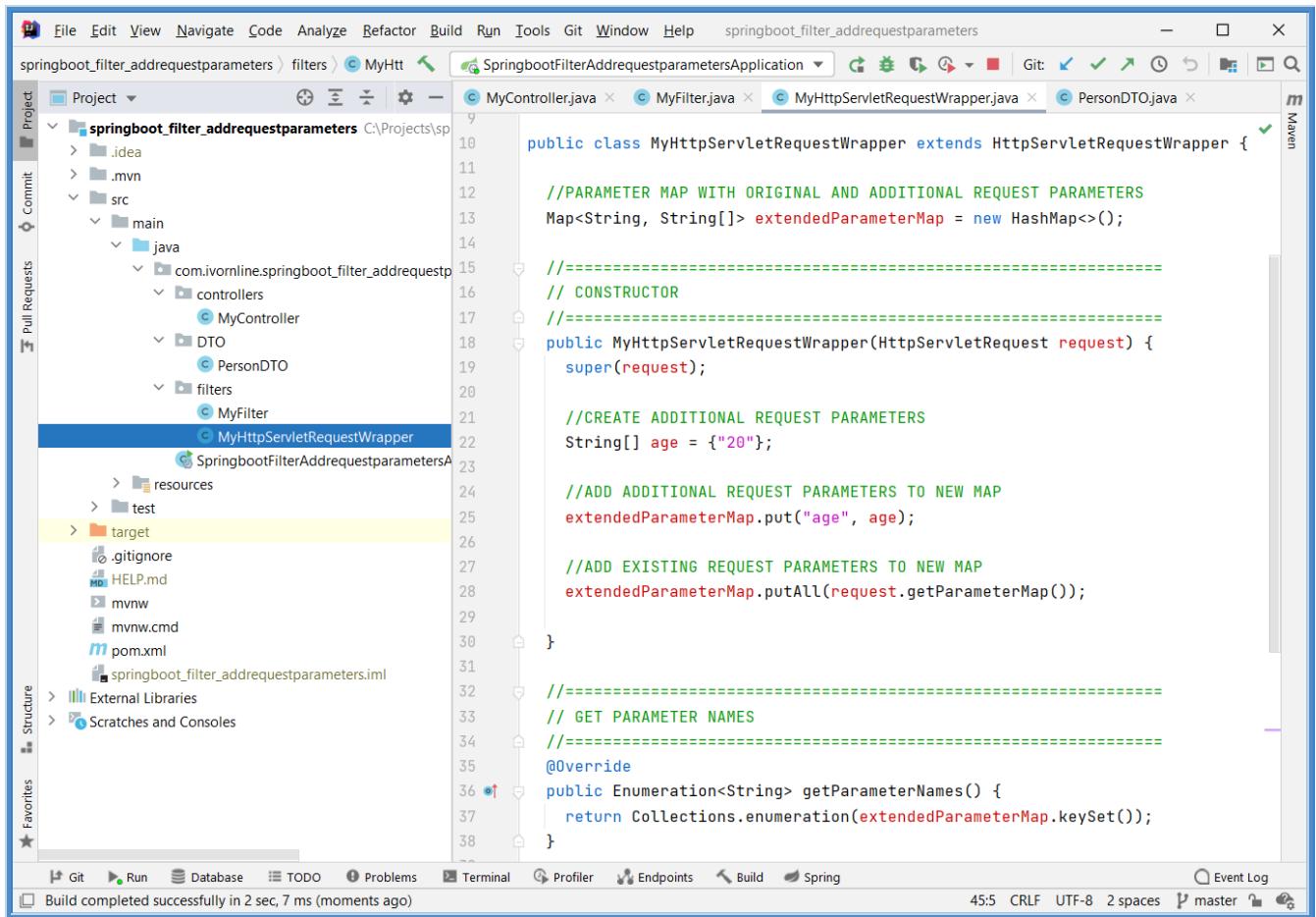
Results

<http://localhost:8080>Hello?name=John>

(Filter adds age Parameter)



Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

2.1.7 HTTP Request Parameters - Edit

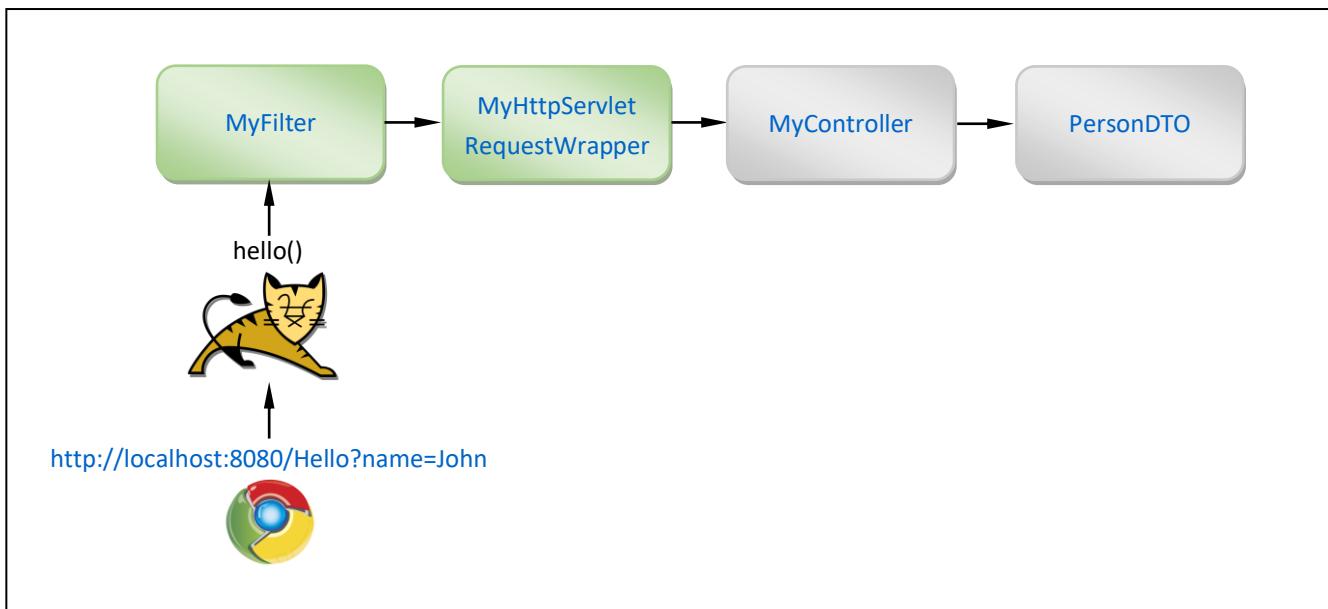
Info

[G]

- This tutorial shows how to "virtually" change value of HTTP Request Parameter by
 - creating new Request (that extends HttpServletRequestWrapper)
 - overriding Methods (getParameterNames, getParameterValues)
 - to return our own parameter Map (with changed values)
- In the Filter we will change the value of `height` HTTP Request Parameter **from "1,67" to "1.67"**. This way Controller will be able to automatically convert and load that value into `PersonDTO` Float Property `height`. Without this we would need to [Deserialize from Request Parameters - Using Setters](#) bloating `PersonDTO`.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_filter_requestparameters_edit` (add Spring Boot Starters from the table)
- **Create Package:** `DTO` (inside main package)
 - **Create Class:** `PersonDTO.java` (inside `DTO` package)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside `controllers` package)
- **Create Package:** `filters` (inside main package)
 - **Create Class:** `MyHttpServletRequestWrapper.java` (inside `filters` package)
 - **Create Class:** `MyFilter.java` (inside `filters` package)

PersonDTO.java

```
package com.ivornline.springboot_filter_addrequestparameters.DTO;

public class PersonDTO {

    //PROPERTIES
    public String name;           //EXISTING REQUEST PARAMETER
    public String age;            //ADDED REQUEST PARAMETER

    //SETTERS (Used for Deserialization)
    public void setName(String name) { this.name = name; }
    public void setAge (String age ) { this.age  = age; }

}
```

MyController.java

```
package com.ivornline.springboot_filter_addrequestparameters.controllers;

import com.ivornline.springboot_filter_addrequestparameters.DTO.PersonDTO;
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(PersonDTO personDTO) {
        return personDTO.name + " is " + personDTO.age + " years old";
    }

}
```

MyHttpServletRequestWrapper.java

```
package com.ivornline.springboot_filter_addrequestparameters.filters;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;

public class MyHttpServletRequestWrapper extends HttpServletRequestWrapper {

    //PARAMETER MAP WITH ORIGINAL AND ADDITIONAL REQUEST PARAMETERS
    Map<String, String[]> extendedParameterMap = new HashMap<>();

    //=====
    // CONSTRUCTOR
    //=====
    public MyHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);

        //CREATE ADDITIONAL REQUEST PARAMETERS
        String[] age = {"20"};

        //ADD ADDITIONAL REQUEST PARAMETERS TO NEW MAP
        extendedParameterMap.put("age", age);

        //ADD EXISTING REQUEST PARAMETERS TO NEW MAP
        extendedParameterMap.putAll(request.getParameterMap());
    }

    //=====
    // GET PARAMETER NAMES
    //=====
    @Override
    public Enumeration<String> getParameterNames() {
        return Collections.enumeration(extendedParameterMap.keySet());
    }

    //=====
    // GET PARAMETER VALUES
    //=====
    @Override
    public String[] getParameterValues(String name) {
        return extendedParameterMap.get(name);
    }
}
```

MyFilter.java

```
package com.ivornline.springboot_filter_addrequestparameters.filters;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class MyFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

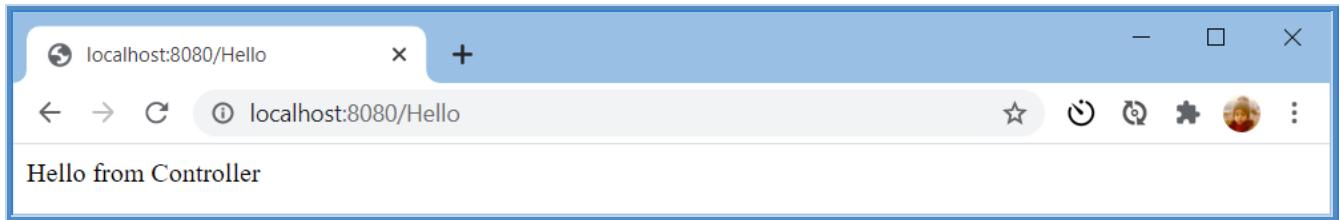
        //CREATE REPLACEMENT REQUEST
        HttpServletRequest myHttpServletRequestWrapper = new MyHttpServletRequestWrapper(request);

        //FORWARD REPLACEMENT REQUEST (With additional Request Parameters)
        chain.doFilter(myHttpServletRequestWrapper, response);
    }
}
```

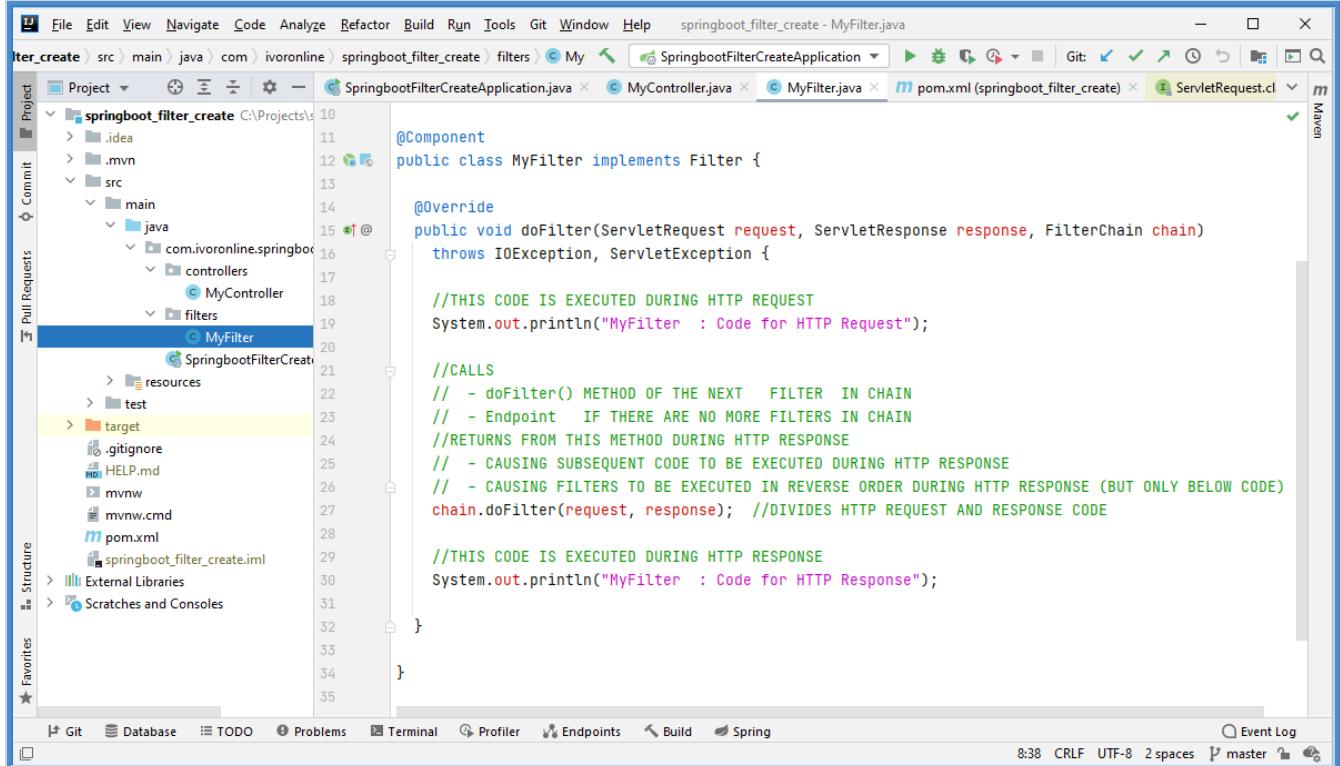
Results

<http://localhost:8080>Hello?name=John&height=1,67>

(Filter converts height to "1.67")



Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

2.1.8 Apply Filter only for specific URL

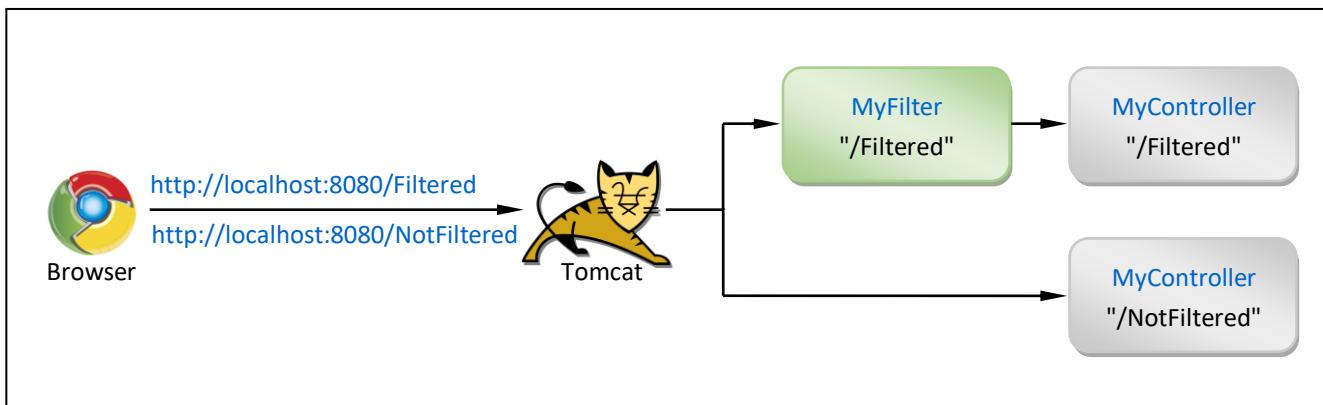
Info

[G]

- This tutorial shows how to create Filter that will be applied only to specific Endpoint/URL.
- This is defined while registering Filter.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

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

MyController.java

```
package com.ivoronline.springboot_filter_urlspecific.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/NotFiltered")
    public String notFiltered() {
        System.out.println("CONTROLLER: Hello from NOT Filtered Endpoint");
        return "Hello from NOT Filtered Endpoint";
    }

    @ResponseBody
    @RequestMapping("/Filtered")
    public String filtered() {
        System.out.println("CONTROLLER: Hello from Filtered Endpoint");
        return "Hello from Filtered Endpoint";
    }
}
```

MyFilter.java

```
package com.ivoronline.springboot_filter_urlspecific.filters;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class MyFilter extends OncePerRequestFilter {

    //=====
    // DO FILTER
    //=====

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        System.out.println("FILTER : Hello from Filter");
        chain.doFilter(request, response);
    }

    //=====
    // FILTER REGISTRATION BEAN
    //=====

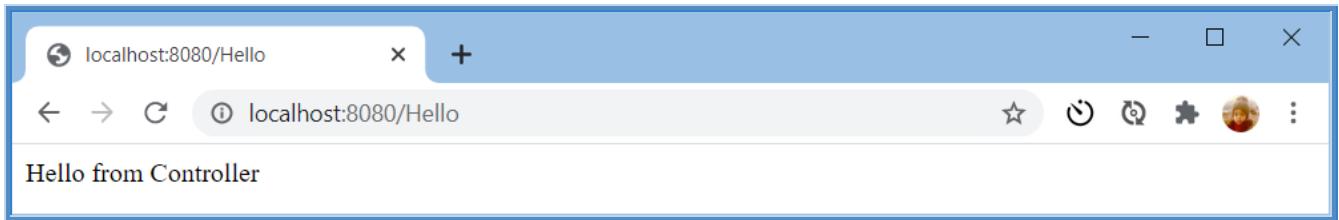
    @Bean
    public FilterRegistrationBean<MyFilter> loggingFilter(){

        //CREATE REGISTRATION BEAN
        FilterRegistrationBean<MyFilter> registrationBean = new FilterRegistrationBean<>();
        registrationBean.setFilter(new MyFilter());
        registrationBean.addUrlPatterns("/Filtered");

        //RETURN REGISTRATION BEAN
        return registrationBean;
    }
}
```

Results

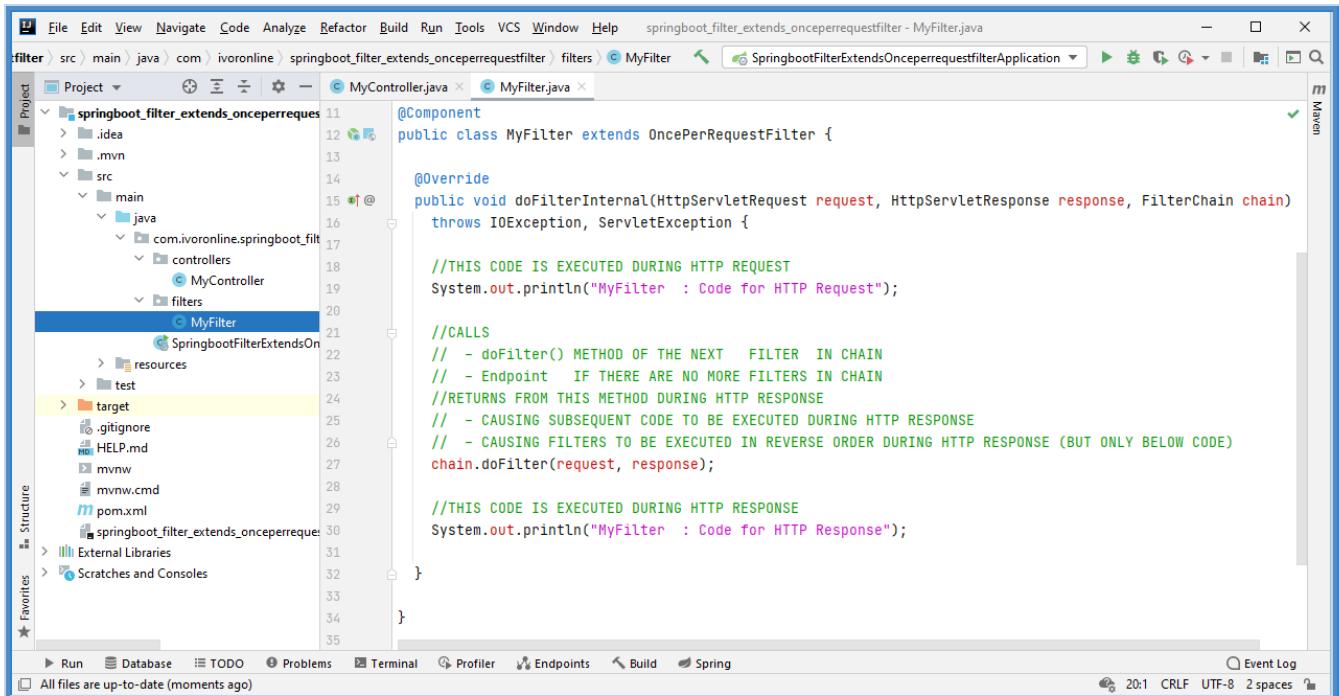
<http://localhost:8080>Hello>



Console

```
MyFilter : Code for HTTP Request
Controller: Code from Controller
MyFilter : Code for HTTP Response
```

Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

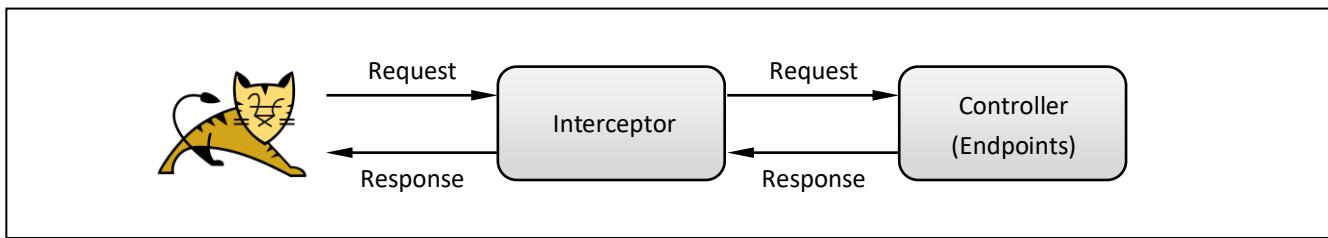
2.2 Interceptors

Info

[R]

- **Interceptor** is **Java Class** that is used to **intercept HTTP Requests** and **Responses**.
- Interceptors perform additional operations before sending
 - HTTP Request to the Controller (they can modify HTTP Request before it gets to Controller)
 - HTTP Response from the Controller (it can modify HTTP Response before it is returned to the User)
- Every Interceptor is **called twice** (because HTTP Requests & Responses pass through same Interceptors)
 - first during HTTP Request
 - then during HTTP Response
- Your custom **Interceptor Class** needs
 - to **implement HandlerInterceptor Interface** (so that Spring would know how to use it)
 - to **@Override preHandle()** Method (contains actual useful code performed by the Filter)
 - to **@Override postHandle()** Method (contains actual useful code performed by the Filter)
 - to **@Override afterCompletion()** Method (contains actual useful code performed by the Filter)
 - **@Component Annotation** (for Spring to detect it and create Interceptor Object)
- Interceptors are used for
 - **Security** (create Authentication Object from JWT Authorities)
 - **Logging** (log HTTP Requests/Responses: User, Endpoint, HTTP Response)
 - **Error Handling** (give feedback to User if HTTP Request has invalid Parameters/Format)

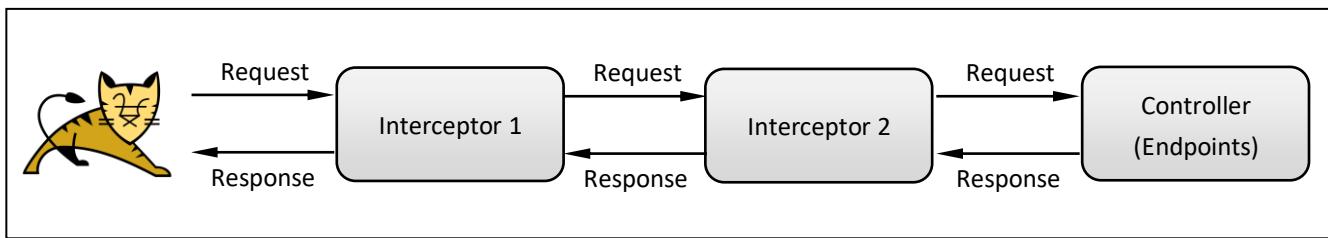
Filter is called twice



Interceptor Chain

- **Interceptor Chain** is ordered set of Interceptors that are
 - called in **sequence** before sending HTTP Request to the Controller
 - called in **reverse order** before sending HTTP Response from the Controller
- Interceptor Chain allows you to organize complex Interceptor Logic into multiple Interceptor Classes.

Interceptors are called in reverse order during HTTP Response



2.2.1 Create Interceptor - Implements HandlerInterceptor

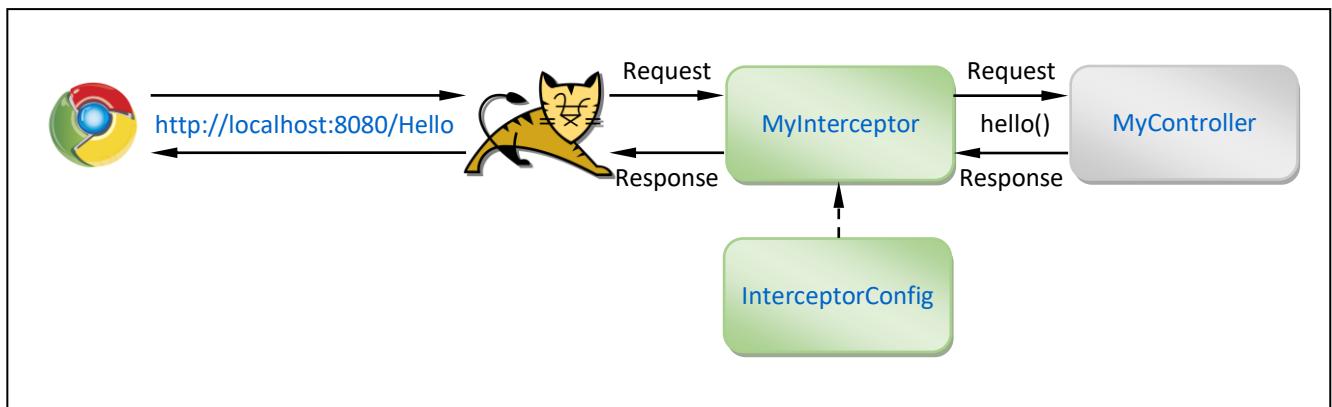
Info

[G]

This tutorial shows how to create Interceptor Class by using [implements HandlerInterceptor](#).

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** [springboot_filter_create](#) (add Spring Boot Starters from the table)
- **Create Package:** controllers (inside main package)
 - **Create Class:** [MyController.java](#) (inside controllers package)
- **Create Package:** interceptors (inside main package)
 - **Create Class:** [MyInterceptor.java](#) (inside controllers package)
 - **Create Class:** [InterceptorConfig.java](#) (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_interceptor.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() {
        System.out.println("Controller:");
        return "Hello from Controller";
    }
}
```

MyInterceptor.java

```
package com.ivoronline.springboot_interceptor.interceptors;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class MyInterceptor implements HandlerInterceptor {

    //=====
    // PRE HANDLE
    //=====

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
        System.out.print("MyInterceptor: preHandle()      ");
        System.out.println(request.getMethod());
        return true;
    }

    //=====
    // POST HANDLE
    //=====

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) {
        System.out.print("MyInterceptor: postHandle()      ");
        System.out.println(response.getStatus());
    }

    //=====
    // AFTER COMPLETION
    //=====

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
                                Exception exception) {
        System.out.print("MyInterceptor: afterCompletion() ");
        System.out.println(response.getStatus());
    }
}
```

InterceptorConfig.java

```
package com.ivoronline.springboot_interceptor.interceptors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

@Component
public class InterceptorConfig extends WebMvcConfigurationSupport {

    @Autowired MyInterceptor myInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(myInterceptor);
    }
}
```

Results

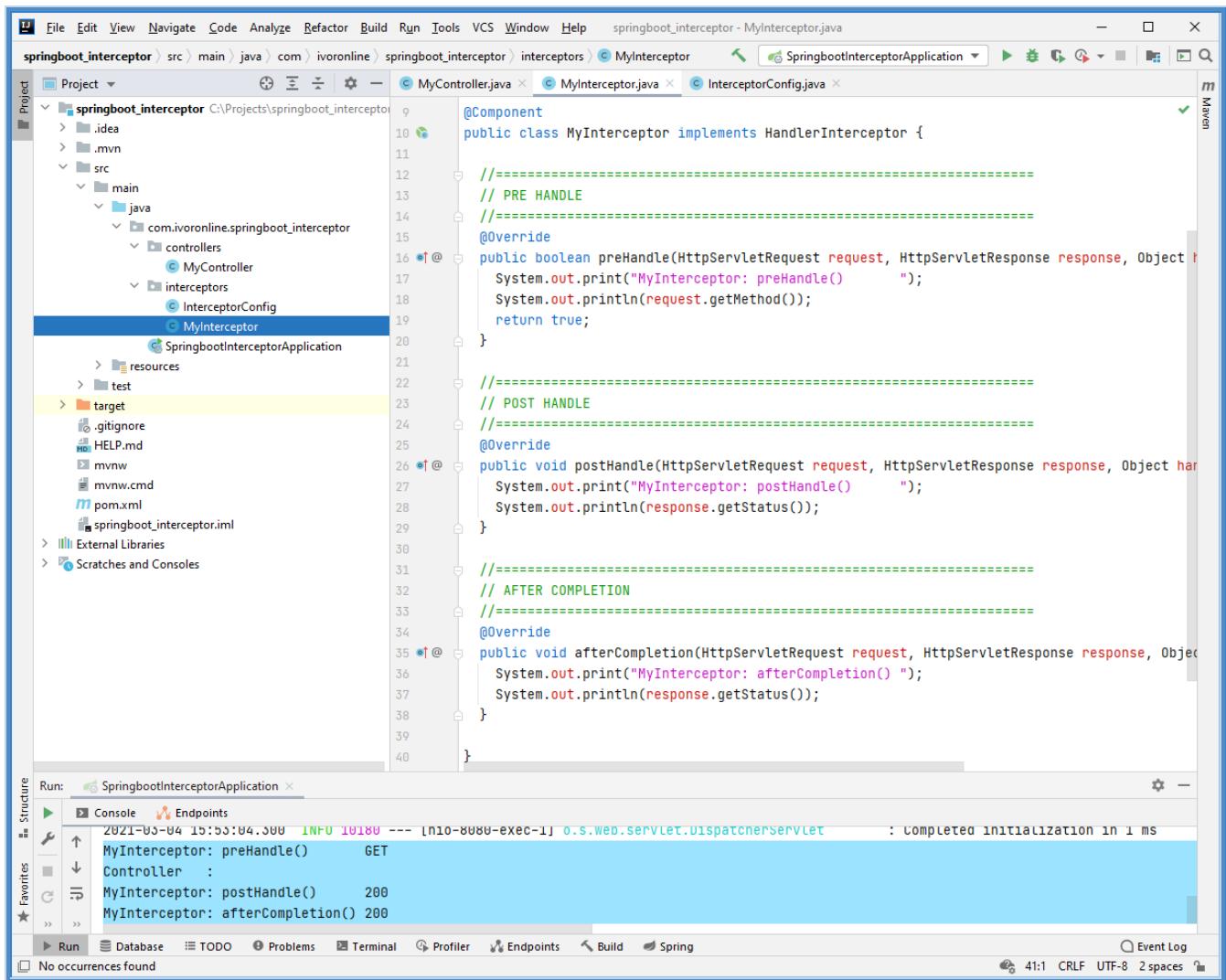
<http://localhost:8080>Hello>



Console

```
MyInterceptor: preHandle()      GET
Controller:
MyInterceptor: postHandle()     200
MyInterceptor: afterCompletion() 200
```

Application Structure



pom.xml

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

2.2.2 Create Interceptor Chain

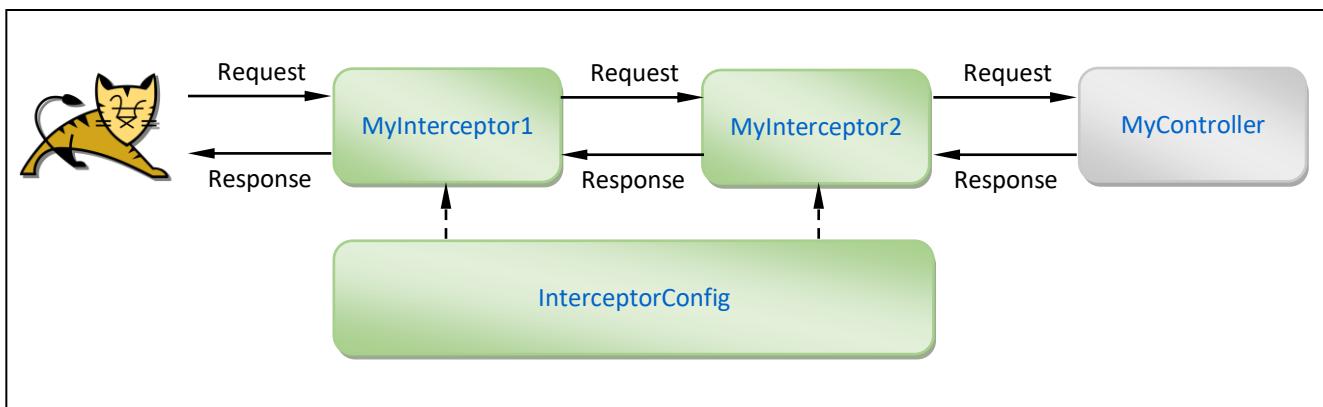
Info

[G]

- This tutorial shows how to create Interceptor Chain.
- Order in which they are added to configuration defines order in which their Methods are executed
 - their `prehandle()` Methods are executed during HTTP Request in that order
 - their `posthandle()` Methods are executed during HTTP Response in **reverse** order
 - their `afterCompletion()` Methods are executed during HTTP Response in **reverse** order

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- **Create Package:** controllers (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Package:** interceptors (inside main package)
 - **Create Class:** `MyInterceptor1.java` (inside controllers package)
 - **Create Class:** `MyInterceptor2.java` (inside controllers package)
 - **Create Class:** `InterceptorConfig.java` (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_interceptor.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() {
        System.out.println("Controller:");
        return "Hello from Controller";
    }
}
```

MyInterceptor1.java

```
package com.ivoronline.springboot_interceptor_chain.interceptors;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class MyInterceptor1 implements HandlerInterceptor {

    //=====
    // PRE HANDLE
    //=====

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
        System.out.print("MyInterceptor1: preHandle()      ");
        return true;
    }

    //=====
    // POST HANDLE
    //=====

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) {
        System.out.print("MyInterceptor1: postHandle()      ");
    }

    //=====
    // AFTER COMPLETION
    //=====

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
                               Exception exception) {
        System.out.print("MyInterceptor1: afterCompletion() ");
    }
}
```

MyInterceptor2.java

```
package com.ivoronline.springboot_interceptor_chain.interceptors;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class MyInterceptor2 implements HandlerInterceptor {

    //=====
    // PRE HANDLE
    //=====

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
        System.out.print("MyInterceptor2: preHandle()      ");
        return true;
    }

    //=====
    // POST HANDLE
    //=====

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) {
        System.out.print("MyInterceptor2: postHandle()      ");
    }

    //=====
    // AFTER COMPLETION
    //=====

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
                               Exception exception) {
        System.out.print("MyInterceptor2: afterCompletion() ");
    }
}
```

InterceptorConfig.java

```
package com.ivoronline.springboot_interceptor_chain.interceptors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

@Component
public class InterceptorConfig extends WebMvcConfigurationSupport {

    @Autowired MyInterceptor1 myInterceptor1;
    @Autowired MyInterceptor2 myInterceptor2;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //ORDER IN WHICH THEY ARE ADDED DEFINES ORDER IN WHICH THEIR METHODS ARE EXECUTED
        // - prehandle()      Methods are executed during HTTP Request  in that order
        // - posthandle()     Methods are executed during HTTP Response  in reverse order
        // - afterCompletion() Methods are executed during HTTP Response  in reverse order
        registry.addInterceptor(myInterceptor1);
        registry.addInterceptor(myInterceptor2);
    }
}
```

Results

<http://localhost:8080>Hello>



Console

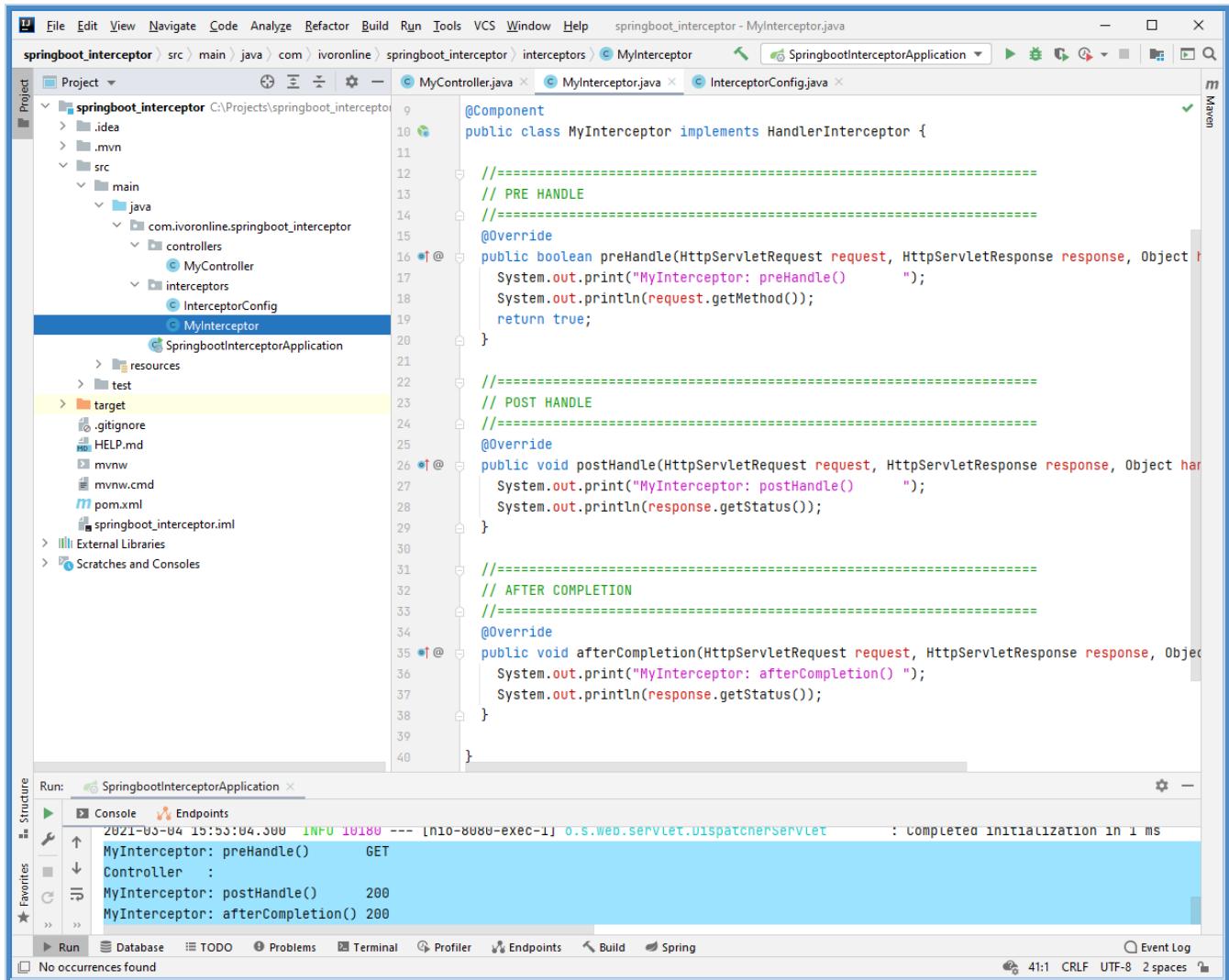
```
MyInterceptor1: preHandle()  
MyInterceptor2: preHandle()
```

Controller:

```
MyInterceptor2: postHandle()  
MyInterceptor1: postHandle()
```

```
MyInterceptor2: afterCompletion()  
MyInterceptor1: afterCompletion()
```

Application Structure



pom.xml

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

2.2.3 HTTP Request/Response - Get Parameters

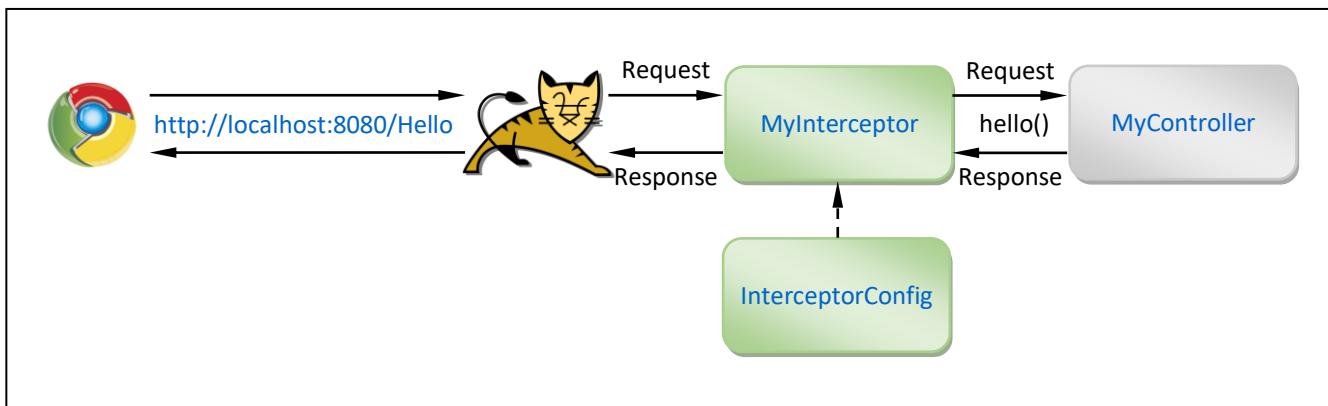
Info

[G]

- This tutorial shows how to use Interceptor to get HTTP Request/Response Parameters.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- Create Package:** controllers (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)
- Create Package:** interceptors (inside main package)
 - Create Class:** `MyInterceptor.java` (inside controllers package)
 - Create Class:** `InterceptorConfig.java` (inside controllers package)

MyController.java

```
package com.ivoronline.springboot_interceptor_gethttprequestresponseparameters.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() {
        System.out.println("Controller:");
        return "Hello from Controller";
    }
}
```

MyInterceptor.java

```
package com.ivoronline.springboot_interceptor_gethttprequestresponseparameters.interceptors;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class MyInterceptor implements HandlerInterceptor {

    //=====
    // PRE HANDLE
    //=====

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
        System.out.println("MyInterceptor: preHandle()      ");
        System.out.println(request.getMethod   ()); //GET
        System.out.println(request.getProtocol()); //HTTP/1.1
        System.out.println(request.getServerName()); //localhost
        System.out.println(request.getServerPort()); //8080
        System.out.println(request.getParameter("username")); //myuser
        return true;
    }

    //=====
    // POST HANDLE
    //=====

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) {
        System.out.println("MyInterceptor: postHandle()      ");
        System.out.println(response.getStatus()); //200
    }

    //=====
    // AFTER COMPLETION
    //=====

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
                               Exception exception) {
        System.out.println("MyInterceptor: afterCompletion() ");
        System.out.println(response.getStatus());
    }
}
```

InterceptorConfig.java

```
package com.ivoronline.springboot_interceptor_gethttprequestresponseparameters.interceptors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

@Component
public class InterceptorConfig extends WebMvcConfigurationSupport {

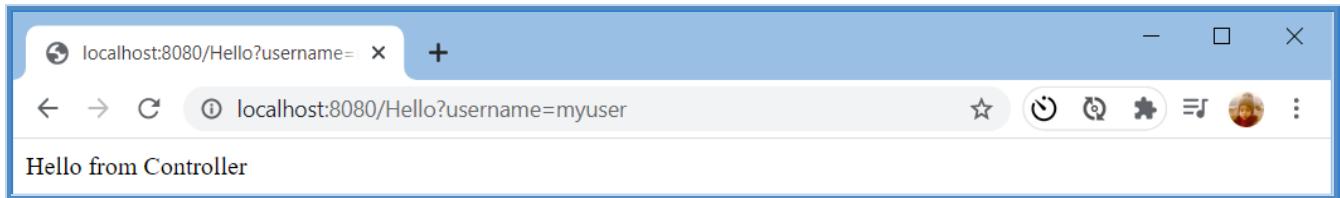
    @Autowired MyInterceptor myInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(myInterceptor);
    }

}
```

Results

<http://localhost:8080>Hello?username=myuser>



Console

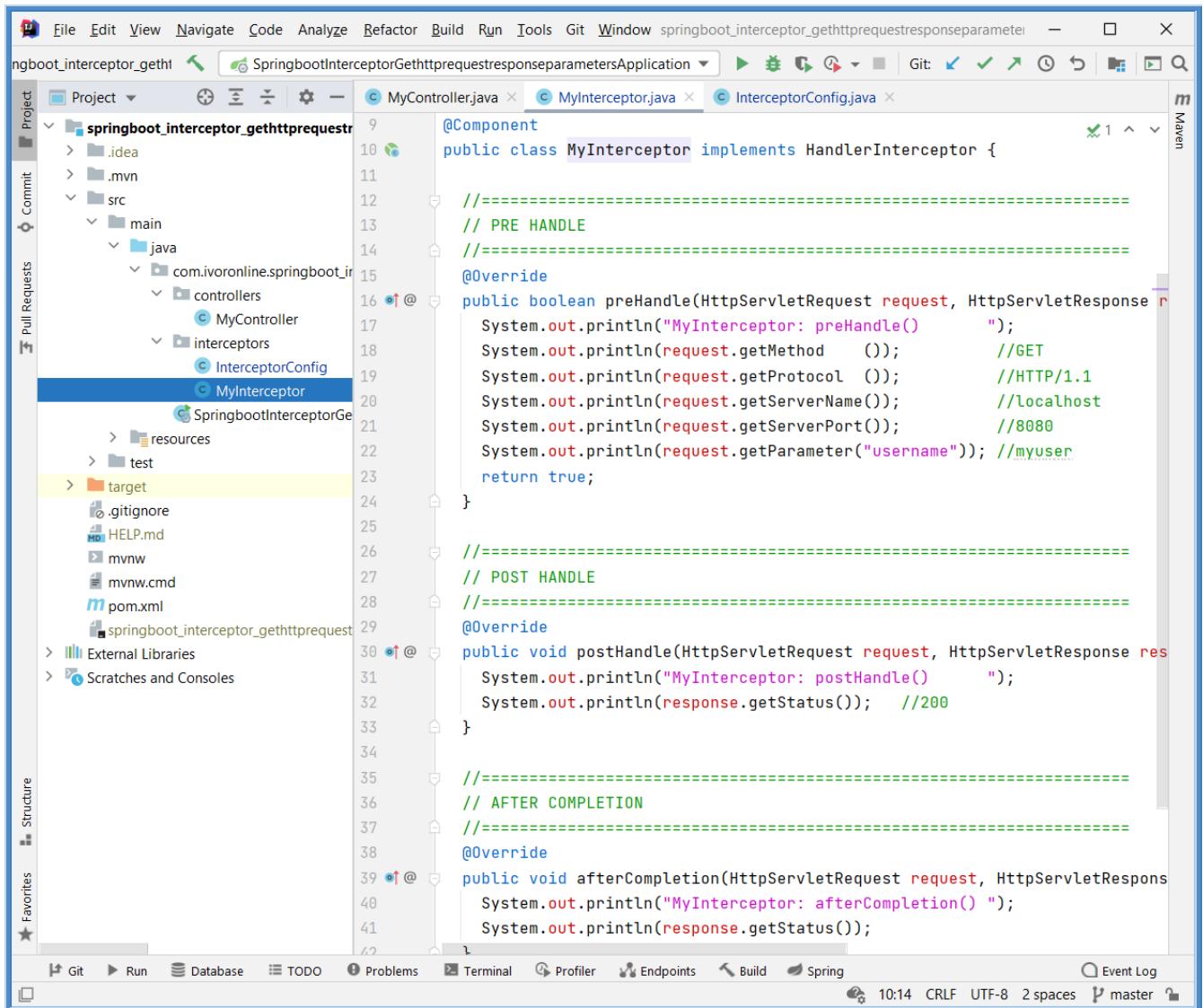
```
MyInterceptor: preHandle()
GET
HTTP/1.1
localhost
8080
myuser

Controller:

MyInterceptor: postHandle()
200

MyInterceptor: afterCompletion()
200
```

Application Structure



pom.xml

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

3 Logging

Info

- Following tutorials show how to use
 - Slf4j with Logback or Log4j to log events to Console, File or Database
 - AOP Annotations to separate logging from business logic

3.1 Slf4j

Info

- Following tutorials show how to use **SLF4J** with either
 - default **Logback** implementation (if no other logging dependencies can be detected in pom.xml)
 - **Log4j2** implementation (if you include Log4j2 dependencies in pom.xml)
- For its internal logging Spring Boot uses
 - **Jakarta Commons logging (JCL)** interface (equivalent to SLF4J)
 - **Logback** implementation (equivalent to Log4J)
- Available **log levels** are : ERROR, WARN, INFO, DEBUG, TRACE.
Selected Log Level will display log lines belonging to that Log Level and those **above** it.
Default log level is INFO which displays: ERROR, WARN, INFO.
- Logging can be **configured** through `application.properties` as shown below.
- Log Configuration allows you to choose
 - **Log Level** (display log lines belonging to that Log Level and those above it)
 - **Appender** (defines where to log data: Console, File, DatedFile. You can choose multiple destinations)
 - **Layout** (defines how to format log lines: Simple, Pattern, HTML)

Pattern Parameters

PARAMETER	EXAMPLE	DESCRIPTION
%p	INFO	Print priority of the logging event
%m	Some error occurred	Print message of the logging event
%c{2}	controllers.MyController	Print last 2 components of Class Path/Name
%d	2021-03-16 18:17:17,091	Print date
%%	%	Print %
%n	\n	Print new line character

Configuration through Properties

- How Properties are used is very confusing because of following three problems
 - each Appender has two references (instead of one)
 - `MyAppender1` is used to reference all the lines that belong to the same Appender
 - `MyConsoleAppender1Name` is used to reference Appender from the Logger (instead of using `MyAppender1`)
 - `appenderRef` has confusing name, it should be called `appenderRefs` since it holds references to multiple Appenders
 - `appenderRef` is followed by reference name (like `myappender1`) even though these names are never used

```
#APPENDERS
appender.MyAppender1.type = Console
appender.MyAppender1.name = MyConsoleAppender1Name

appender.MyAppender2.type = Console
appender.MyAppender2.name = MyConsoleAppender2Name
appender.MyAppender2.layout.type = PatternLayout
appender.MyAppender2.layout.pattern = MyAppender2: %d %p %c{2} %m %n

#LOGERS
logger.MyLogger1.name = com.ivoronline.springboot_log_log4j_config_xml.controllers
logger.MyLogger1.level = info
logger.MyLogger1.appenderRef.myappender1.ref = MyConsoleAppender1Name
logger.MyLogger1.appenderRef.myappender2.ref = MyConsoleAppender2Name

#DISABLE ROOT LOGGER
rootLogger.level = error
```

3.1.1 Using LoggerFactory

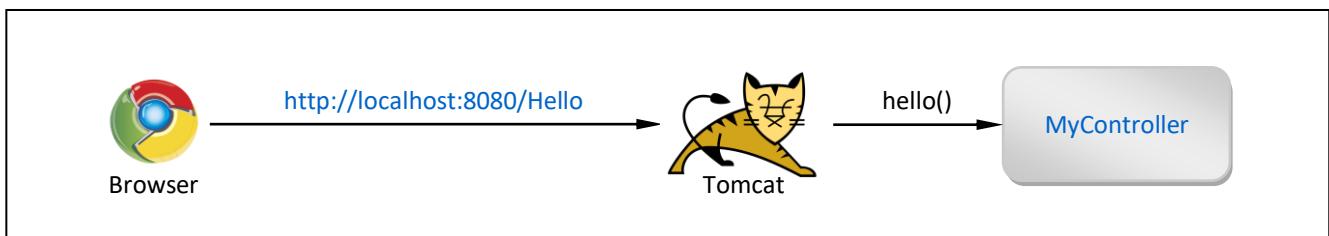
Info

[G] [R]

- This tutorial shows how to log with **Slf4J**.
- At this point we are not using any additional adjustments, so Slf4J will route our calls to default **Logback** implementation. Later if you decide to change underlying logger implementation, for instance to switch from Logback to Log4j, the way you would use SLF4J to log events would remain the same as described in this tutorial.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project: `springboot_log` (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_log.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

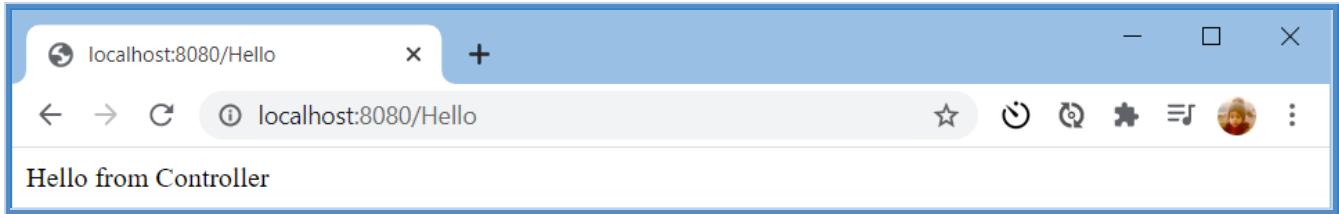
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

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

Results

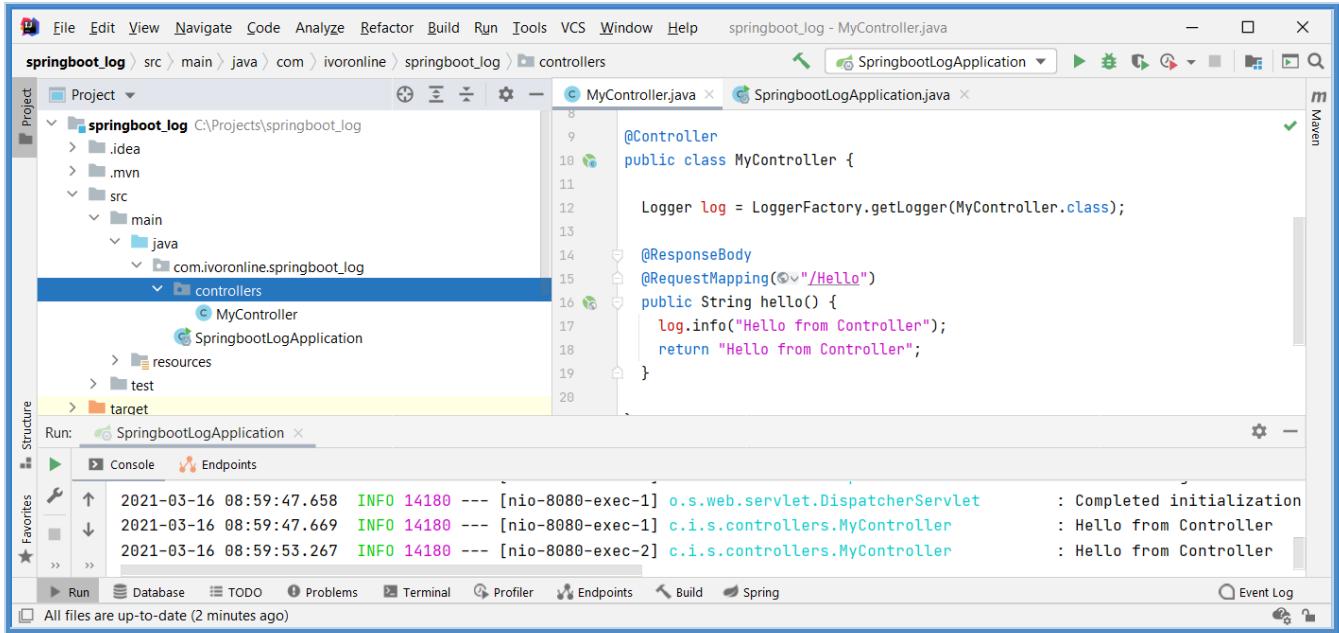
<http://localhost:8080>Hello>



Console

```
2021-03-16 08:59:53.267 INFO 14180 - [nio-8080-exec-2] c.i.s.controllers.MyController : Hello from Controller
```

Application Structure



pom.xml

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

3.1.2 Using @Slf4j

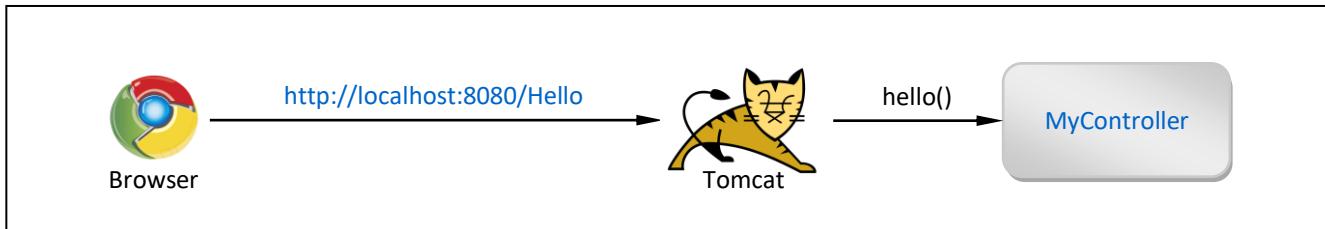
Info

[G] [R]

- This tutorial shows how to use Lombok's **@Slf4j** Annotation to work with Slf4J.
- This Annotation automatically generates following code so that we don't have to repeat it in every Class we want to log
`private static final org.slf4j.Logger log = org.slf4j.LoggerFactory.getLogger(LogExampleOther.class);`

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: <code>@RequestMapping</code> , Tomcat Server
Developer Tools	Lombok	Enables: @Slf4j (injects LoggerFactory)

Procedure

- Create Project:** `springboot_log_lombok` (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_log_lombok.controllers;

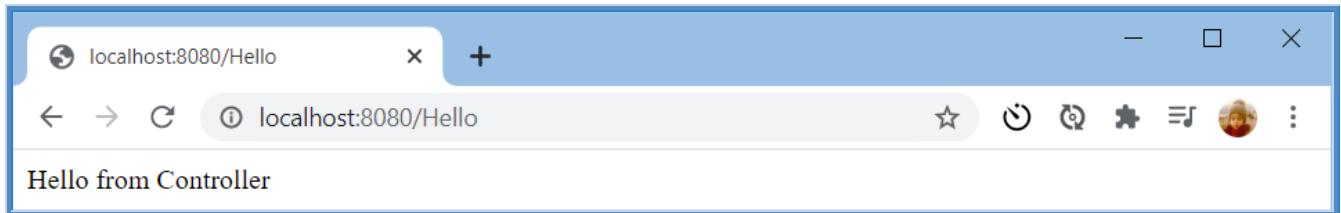
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

@Slf4j
@Controller
public class MyController {

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

Results

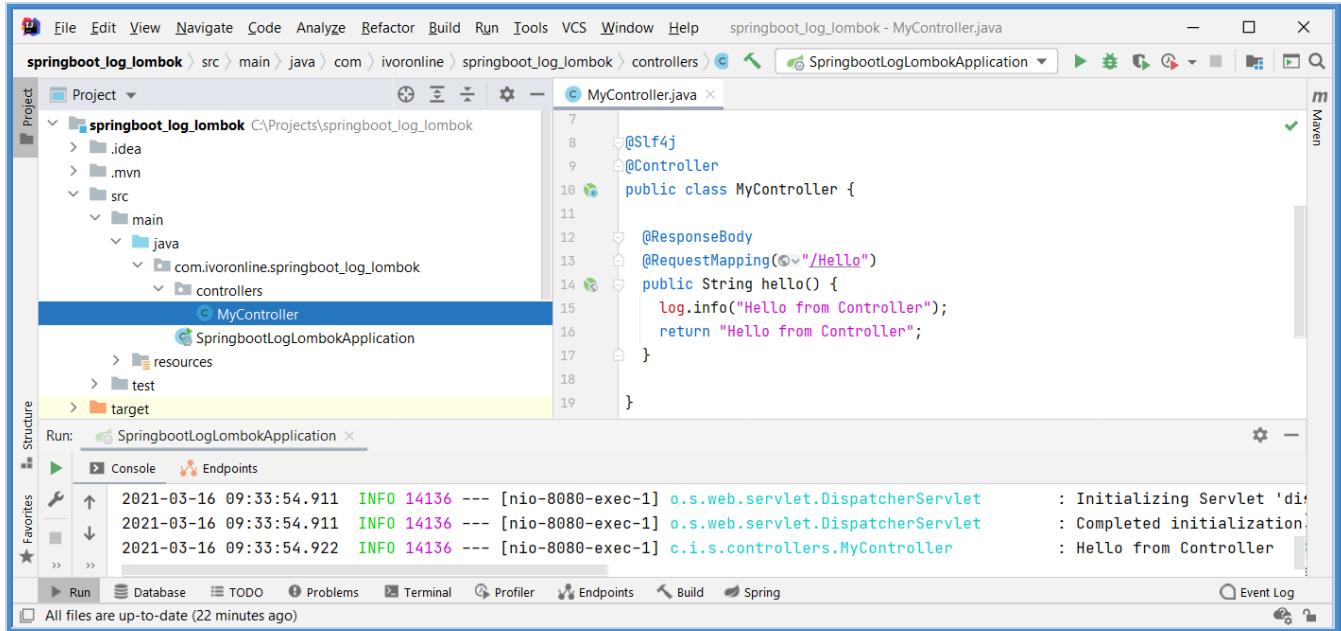
<http://localhost:8080>Hello>



Console

```
2021-03-16 08:59:53.267 INFO 14180 - [nio-8080-exec-2] c.i.s.controllers.MyController : Hello from Controller
```

Application Structure



pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

</dependencies>
```

3.1.3 Logback

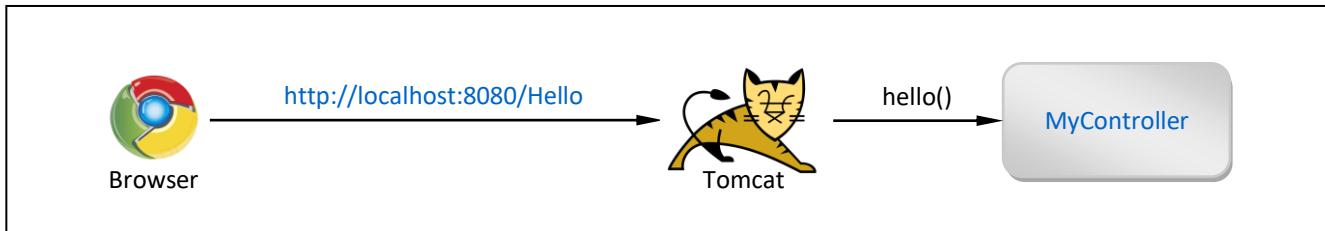
Info

[G] [R]

- Spring Boot by default uses **Logback** implementation for logging.
- Logback is included into every Spring Boot Starter therefore no additional configuration is needed for basic logging.
- Inside the Code we will use **slf4j** API to log Events and **slf4j** will forward these API calls to Logback.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project: `springboot_log` (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_log.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

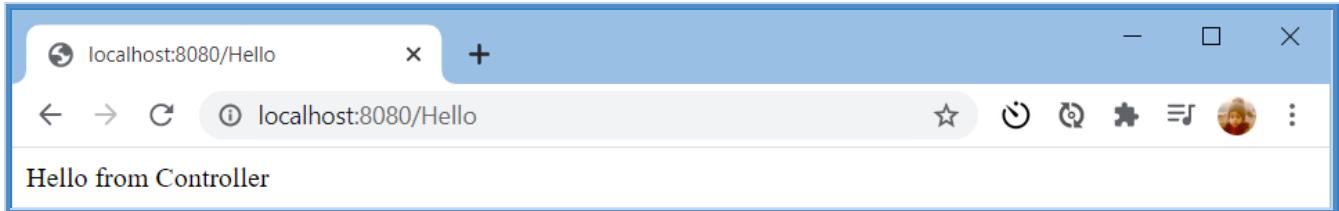
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

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

Results

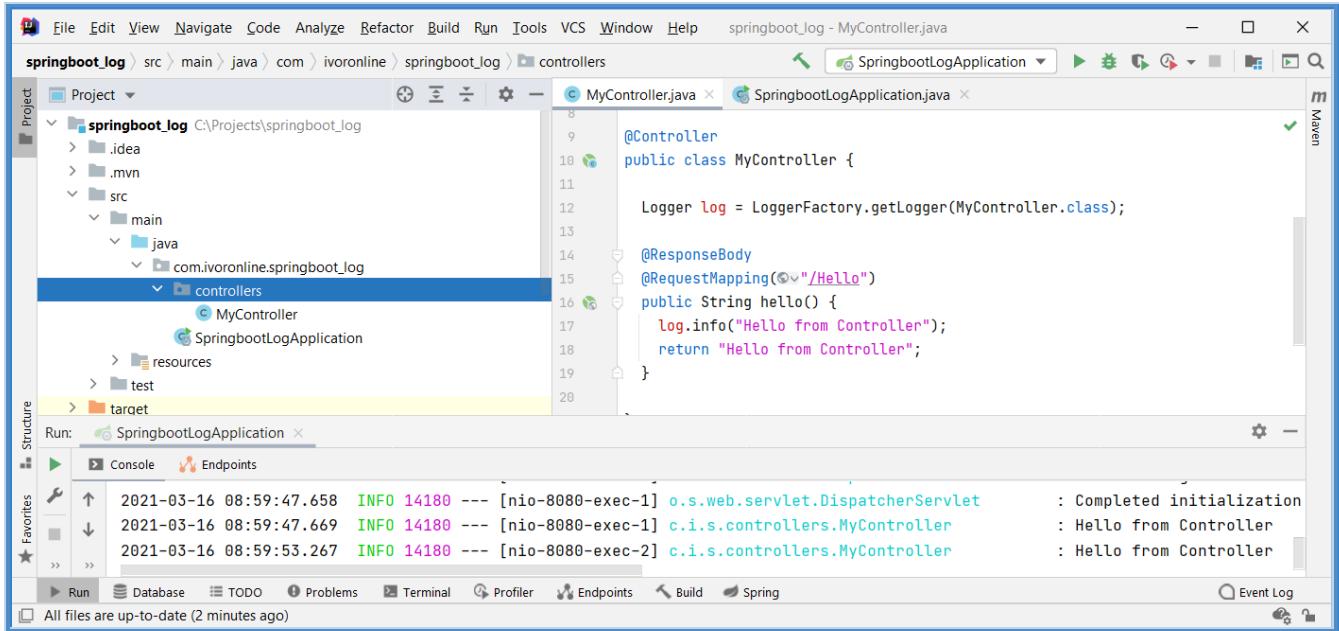
<http://localhost:8080>Hello>



Console

```
2021-03-16 08:59:53.267 INFO 14180 - [nio-8080-exec-2] c.i.s.controllers.MyController : Hello from Controller
```

Application Structure



pom.xml

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

3.1.4 Logback - Configure - XML

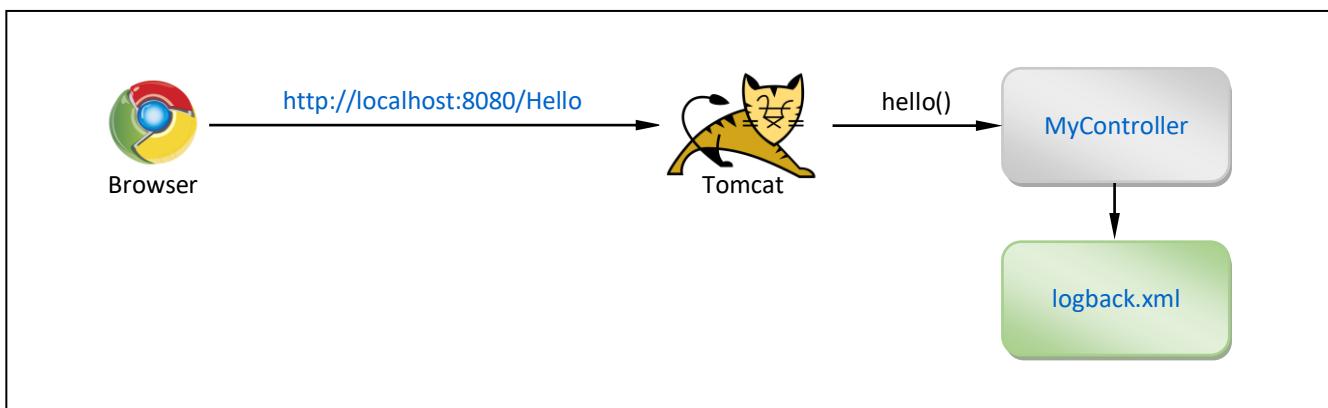
Info

[G]

- This tutorial shows how to configure Log4j by creating `log4j2.xml` file inside `resources` Directory.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_log_logback_config_xml` (add Spring Boot Starters from the table)
- Create File:** `logback.xml` (inside Directory resources)
- Create Package:** `controllers` (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)

`logback.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- CONSOLE APPENDER -->
    <appender name="MyConsoleAppenderName" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>My File Appender: %d %p %c{0} %m %n</Pattern>
        </encoder>
    </appender>

    <!-- FILE APPENDER -->
    <appender name="MyFileAppenderName" class="ch.qos.logback.core.FileAppender">

        <!-- FILE NAME -->
        <file>logs/File.log</file>

        <!-- ENCODER -->
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>My File Appender: %d %p %c{30} %m %n</Pattern>
        </encoder>

    </appender>

    <!-- ROLLING FILE APPENDER -->
    <appender name="MyRollingFileAppenderName" class="ch.qos.logback.core.rolling.RollingFileAppender">

        <!-- FILE NAME -->
        <file>logs/RollingFile.log</file>


```

```

<!-- ENCODER -->
<encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <Pattern>My File Appender: %d %p %c{30} %m %n</Pattern>
</encoder>

<!-- ROLLOVER POLICY (daily and when the file reaches 1MB) -->
<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <fileNamePattern>logs/archived/RollingFile_%d{dd.MM.yyyy}_%i.log</fileNamePattern>
    <timeBasedFileNamingAndTriggeringPolicy>
        <maxFileSize>10MB</maxFileSize>
    </timeBasedFileNamingAndTriggeringPolicy>
</rollingPolicy>

</appender>

<!-- LOGGER -->
<logger name="com.ivoronline.springboot_log_logback_config_xml.controllers" level="INFO">
    <appender-ref ref="MyConsoleAppenderName" />
    <appender-ref ref="MyFileAppenderName" />
    <appender-ref ref="MyRollingFileAppenderName"/>
</logger>

<!-- DISABLE DEFAULT LOGGER (since no appenders are defined) -->
<root level="info"> </root>

</configuration>

```

MyController.java

```

package com.ivoronline.springboot_log_logback_config_xml.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

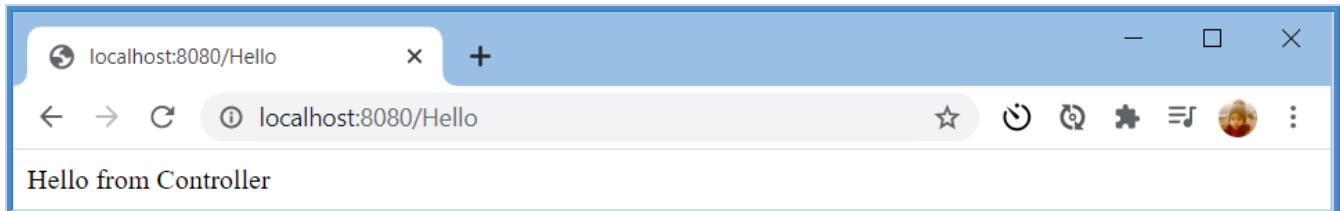
    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn ("Some warn occurred");
        log.info ("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}

```

Results

<http://localhost:8080>Hello>



Console

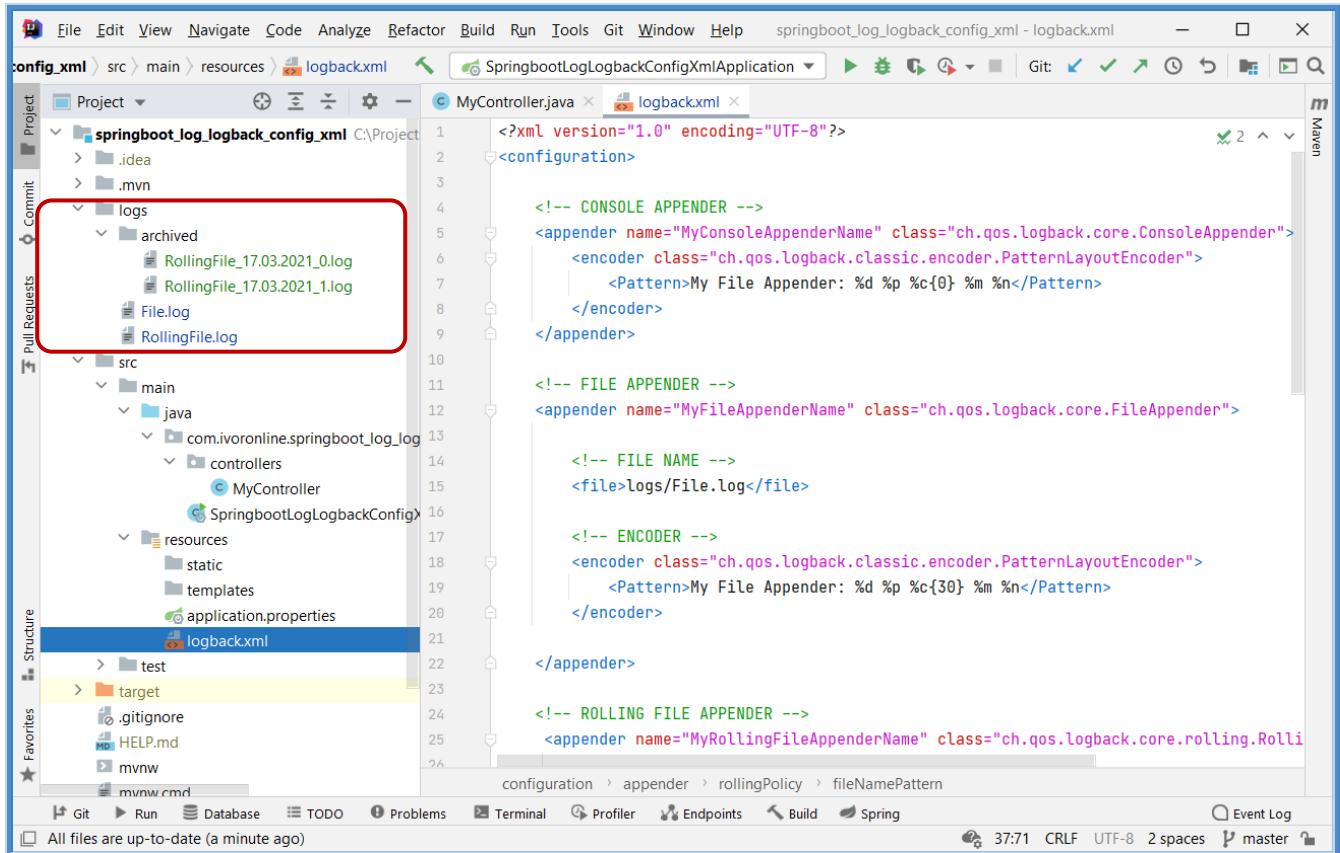
```
My File Appender: 2021-03-17 12:17:39,691 ERROR c.i.s.controllers.MyController Some error occured  
My File Appender: 2021-03-17 12:17:39,692 WARN c.i.s.controllers.MyController Some warn occured  
My File Appender: 2021-03-17 12:17:39,692 INFO c.i.s.controllers.MyController Some info occured
```

logs/File.log

(My File Appender: %d %p %c{2} %m %n)

```
My File Appender: 2021-03-17 12:17:39,691 ERROR c.i.s.controllers.MyController Some error occured  
My File Appender: 2021-03-17 12:17:39,692 WARN c.i.s.controllers.MyController Some warn occured  
My File Appender: 2021-03-17 12:17:39,692 INFO c.i.s.controllers.MyController Some info occured
```

Application Structure



pom.xml

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

3.1.5 Logback - Configure - XML - DB

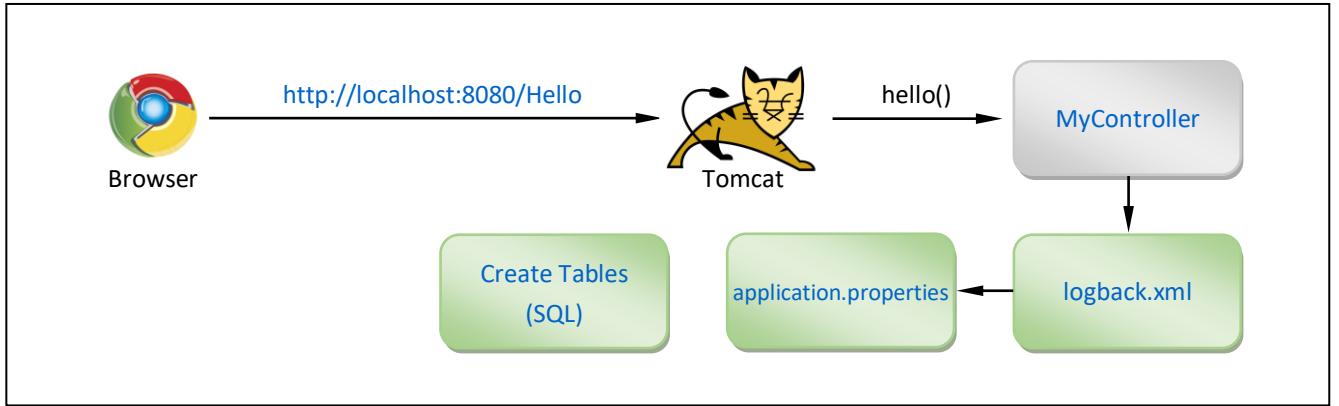
Info

[G] [R]

- This tutorial shows how to use **XML** configuration to configure **Logback** to log into **PostgreSQL Database**.
- It seems that the same can't be achieved through the **application.properties** alone.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: [springboot_log_logback_config_xml](#) (add Spring Boot Starters from the table)
- Edit File: [application.properties](#) (add DB source)
- Create File: [logback.xml](#) (inside Directory resources)
- Execute SQL: [SQL Create Tables](#) (manually create DB Tables in PostgreSQL DB)
- Create Package: [controllers](#) (inside main package)
 - Create Class: [MyController.java](#) (inside controllers package)

application.properties

```
# POSTGRES DATABASE
spring.datasource.url      = jdbc:postgresql://localhost:5432/Test
spring.datasource.username  = postgres
spring.datasource.password = letmein
spring.datasource.driver-class-name = org.postgresql.Driver

# JPA / HIBERNATE
spring.jpa.hibernate.ddl-auto = create-drop
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- IMPORT PROPERTIES -->
    <property resource="application.properties" />

    <!-- APPENDER -->
    <appender name="MyDBAppenderName" class="ch.qos.logback.classic.db.DBAppender">
        <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">
            <url> ${spring.datasource.url} </url>
            <user> ${spring.datasource.username} </user>
            <password> ${spring.datasource.password} </password>
            <driverClass> ${spring.datasource.driver-class-name} </driverClass>
        </connectionSource>
    </appender>

    <!-- LOGGER -->
    <logger name="com.ivorononline.springboot_log_logback_config_xml_db.controllers" level="INFO">
        <appender-ref ref="MyDBAppenderName"/>
    </logger>

</configuration>
```

Create Tables SQL

```
DROP TABLE    logging_event_property;
DROP TABLE    logging_event_exception;
DROP TABLE    logging_event;
DROP SEQUENCE logging_event_id_seq;

CREATE SEQUENCE logging_event_id_seq MINVALUE 1 START 1;

CREATE TABLE logging_event
(
    timestamp      BIGINT NOT NULL,
    formatted_message TEXT NOT NULL,
    logger_name    VARCHAR(254) NOT NULL,
    level_string   VARCHAR(254) NOT NULL,
    thread_name    VARCHAR(254),
    reference_flag SMALLINT,
    arg0           VARCHAR(254),
    arg1           VARCHAR(254),
    arg2           VARCHAR(254),
    arg3           VARCHAR(254),
    caller_filename VARCHAR(254) NOT NULL,
    caller_class   VARCHAR(254) NOT NULL,
    caller_method  VARCHAR(254) NOT NULL,
    caller_line    CHAR(4) NOT NULL,
    event_id       BIGINT DEFAULT nextval('logging_event_id_seq') PRIMARY KEY
);

CREATE TABLE logging_event_property
(
    event_id       BIGINT NOT NULL,
    mapped_key     VARCHAR(254) NOT NULL,
    mapped_value   VARCHAR(1024),
    PRIMARY KEY(event_id, mapped_key),
    FOREIGN KEY (event_id) REFERENCES logging_event(event_id)
);

CREATE TABLE logging_event_exception
(
    event_id       BIGINT NOT NULL,
    i              SMALLINT NOT NULL,
    trace_line    VARCHAR(254) NOT NULL,
    PRIMARY KEY(event_id, i),
    FOREIGN KEY (event_id) REFERENCES logging_event(event_id)
);
```

MyController.java

```
package com.ivoronline.springboot_log_logback_config_xml_db.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn ("Some warn occurred");
        log.info ("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello>



pgAdmin4

A screenshot of the pgAdmin 4 interface. The left sidebar shows a tree view of database objects under 'Test/postgres...'. A red box highlights the 'Tables (3)' section, which contains three entries: 'logging_event', 'logging_event_exception', and 'logging_event_property'. The 'logging_event' entry is selected. The right panel shows the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM public.logging_event
2 ORDER BY event_id ASC
```

The 'Data Output' tab is active, displaying a table with the following data:

	timestamp	formatted_message	logger_name	level_string	thread_name	refer...	arg0	arg1	arg...
1	1616058647105	Some error occurred	com.ivoronline.springb...	ERROR	http-nio-8080...	0	[null]	[null]	[n...
2	1616058647151	Some warn occurred	com.ivoronline.springb...	WARN	http-nio-8080...	0	[null]	[null]	[n...
3	1616058647193	Some info occurred	com.ivoronline.springb...	INFO	http-nio-8080...	0	[null]	[null]	[n...

SQL

(IntelliJ Database Console)

```
SELECT timestamp, formatted_message, level_string, caller_filename, caller_method, caller_line
FROM public.logging_event
```

logging_event

(IntelliJ Database Tool)

A screenshot of the IntelliJ Database Tool's data viewer. It shows a table with the following columns: timestamp, formatted_message, level_string, caller_filename, caller_method, and caller_line. The data is identical to the one shown in the pgAdmin screenshot:

	timestamp	formatted_message	level_string	caller_filename	caller_method	caller_line
1	1616058647105	Some error occurred	ERROR	MyController.java	hello	17
2	1616058647151	Some warn occurred	WARN	MyController.java	hello	18
3	1616058647193	Some info occurred	INFO	MyController.java	hello	19

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** Shows the project structure under "springboot_log_logback_config_xml_db". It includes ".idea", ".mvn", "src" (containing "main" and "resources"), and "application.properties".
- Editor:** Displays the content of "logback.xml". The XML code defines a database appender named "MyDBAppenderName" using a PostgreSQL connection. It also defines a logger for the package "com.ivoronline.springboot_log_logback_config_xml_db.controllers" at the INFO level, which refers to the database appender.
- Bottom Bar:** Shows various tool icons and status information like "Connected (2 minutes ago)", "Event Log", and "master".

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>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

</dependencies>
```

3.1.6 Logback - Configure - XML - DB - Custom

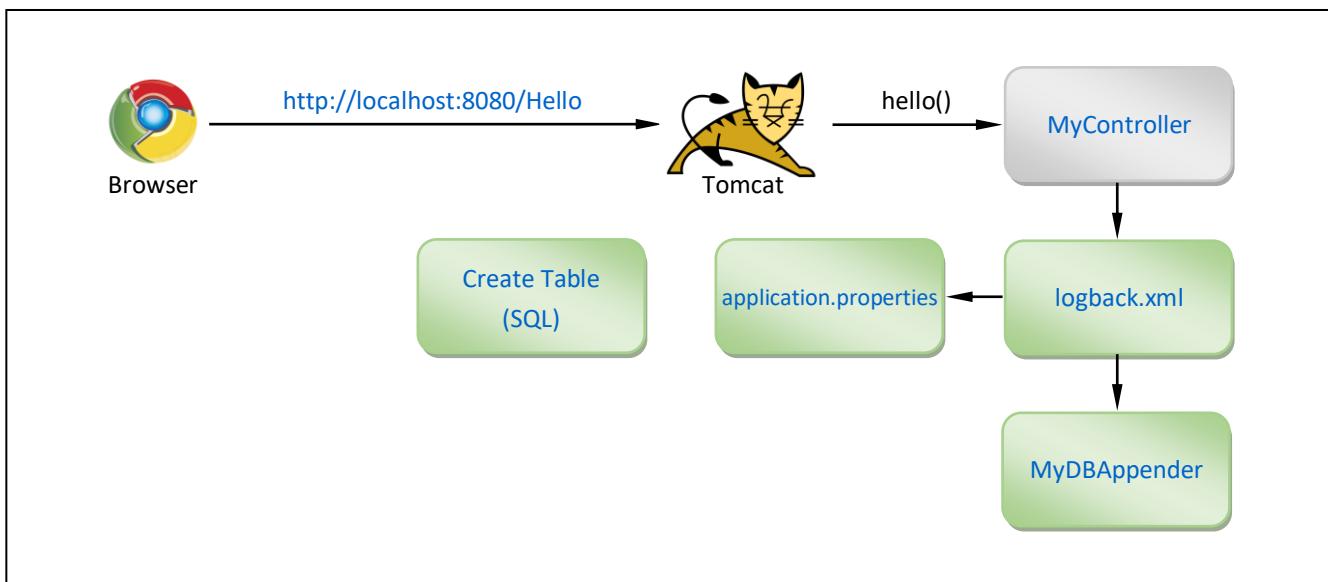
Info

[G] [R]

- This tutorial shows how to use **XML** configuration to configure **Logback** to log into **PostgreSQL Database**. We will use [MyDBAppender.java](#) to define custom table and columns to store log events.
- DB connection parameters need to be defined inside [application.properties](#) or application will not work. To prevent typing the same parameters again in the [logback.xml](#) we will reference those from [application.properties](#).

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server
SQL	Spring Data JPA	Enables: @Entity, @Id
SQL	PostgreSQL Database	Enables Hibernate to work with PostgreSQL DB

Procedure

- **Create Project:** `springboot_log_logback_config_xml` (add Spring Boot Starters from the table)
- **Edit File:** `application.properties` (add Database connection parameters)
- **Create File:** `logback.xml` (inside Directory resources)
- **Execute SQL:** `SQL Create Tables` (manually create DB Tables in PostgreSQL DB)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Package:** `log` (inside log package)
 - **Create Class:** `MyDBAppender.java` (inside controllers package)

application.properties

```
# POSTGRES DATABASE
spring.datasource.url          = jdbc:postgresql://localhost:5432/Test
spring.datasource.username       = postgres
spring.datasource.password       = letmein
spring.datasource.driver-class-name = org.postgresql.Driver

# JPA / HIBERNATE
spring.jpa.hibernate.ddl-auto   = create-drop
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- IMPORT PROPERTIES -->
    <property resource="application.properties" />

    <!-- MY DB APPENDER -->
    <appender name="MyDBAppenderName"
              class="com.ivoronline.springboot_log_logback_config_xml_db_custom.log.MyDBAppender">
        <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">
            <url>      ${spring.datasource.url}           </url>
            <user>      ${spring.datasource.username}     </user>
            <password>  ${spring.datasource.password}     </password>
            <driverClass> ${spring.datasource.driver-class-name} </driverClass>
        </connectionSource>
    </appender>

    <!-- LOGGER -->
    <logger name="com.ivoronline.springboot_log_logback_config_xml_db_custom.controllers" level="INFO">
        <appender-ref ref="MyDBAppenderName"/>
    </logger>

</configuration>
```

Create Table SQL

```
-- CREATE SEQUENCE
create sequence transaction_event_id_seq;

-- CREATE TABLE
create table transaction (
    event_id serial DEFAULT nextval('transaction_event_id_seq'::regclass),
    message  varchar,
    level    varchar,
    date     timestamp
);
```

MyDBAppender.java

```
package com.ivoronline.springboot_log_logback_config_xml_db_custom.log;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import ch.qos.logback.classic.db.DBAppender;
import ch.qos.logback.classic.spi.ILoggingEvent;

public class MyDBAppender extends DBAppender {

    //=====
    // GET INSERT SQL
    //=====

    @Override
    protected String getInsertSQL() {
        return "INSERT INTO TRANSACTION (event_id, timestamp, level, message) VALUES (DEFAULT, ?, ?, ?)";
    }

    //=====
    // SUB APPEND
    //=====

    @Override
    protected void subAppend(ILoggingEvent event, Connection connection, PreparedStatement preparedStatement)
        throws SQLException {

        //GET LOG EVENT DATA
        String level = event.getLevel().toString();
        String message = event.getMessage();

        //PREPARE STATEMENT
        preparedStatement.setTimestamp(1, Timestamp.valueOf(LocalDateTime.now()));
        preparedStatement.setString (2, level );
        preparedStatement.setString (3, message);

        //EXECUTE PREPARED STATEMENT
        preparedStatement.executeUpdate();

    }

}
```

MyController.java

```
package com.ivoronline.springboot_log_logback_config_xml_db_custom.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

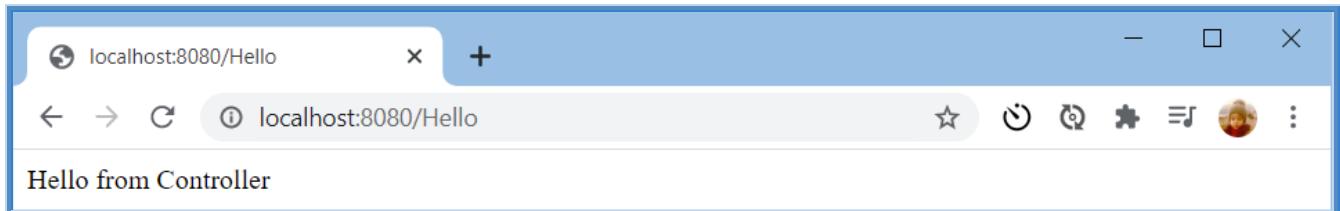
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn ("Some warn occurred");
        log.info ("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello>



logging_event

(IntelliJ Database Tool)

	event_id	date	message	level
1		29 2021-03-18 16:04:52.720884	Some error occured	ERROR
2		30 2021-03-18 16:04:52.777729	Some warn occured	WARN
3		31 2021-03-18 16:04:52.835090	Some info occured	INFO

Application Structure

A screenshot of the IntelliJ IDEA IDE. The left side shows the project structure for a Spring Boot application named "springboot_log_logback_config_xml_db_custom". The "src/main/java" directory contains "MyController.java" and "MyDBAppender.java". The "log" package contains "SpringbootLogLogbackConfigXmlDbCustomApplication.java". The "resources" folder includes "application.properties" and "logback.xml". The right side shows the code editor with the "MyDBAppender.java" file open, displaying Java code for a database appender.

```
public class MyDBAppender extends DBAppender {
    //=====
    // GET INSERT SQL
    //=====
    @Override
    protected String getInsertSQL() {
        return "INSERT INTO TRANSACTION (event_id, level, message) VALUES (DEFAULT, ?, ?)";
    }

    //=====
    // SUB APPEND
    //=====
    @Override
    protected void subAppend(ILoggingEvent event, Connection connection, PreparedStatement preparedStatement)
        throws SQLException {
        //GET LOG EVENT DATA
        String level = event.getLevel().toString();
        String message = event.getMessage();

        //PREPARE STATEMENT
        preparedStatement.setString(1, level);
        preparedStatement.setString(2, message);

        //EXECUTE PREPARED STATEMENT
        preparedStatement.executeUpdate();
    }
}
```

```
<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>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

</dependencies>
```

3.1.7 Logback - Configure - Properties

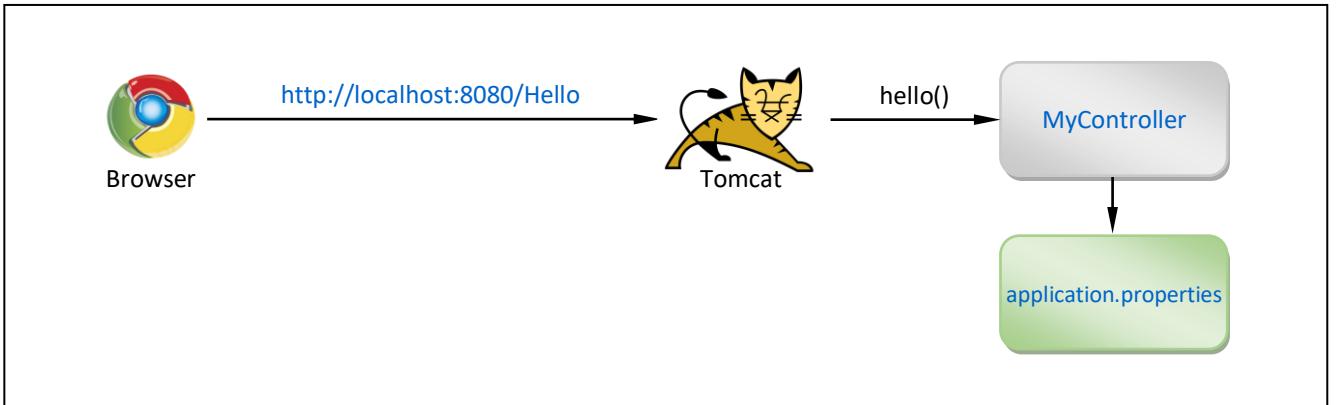
Info

[G] [R]

- This tutorial shows how to use `application.properties` to configure **Logback**.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_log_configuration` (add Spring Boot Starters from the table)
- **Edit File:** `application.properties` (inside main package)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)

application.properties

```
# CREATE GROUP
logging.group.mygroup      = com.ivoronline, com.ivoronline.springboot_log_logback_config_properties

# LOG LEVELS (for Group, Root, Package, Class)
logging.level.mygroup      = error
logging.level.root          = error
logging.level.com.ivoronline = error
logging.level.com.ivoronline.springboot_log_logback_config_properties.controllers.MyController = info

# CONSOLE LOGGER
logging.pattern.console     = My Console Log: %d %p %c{0} %m %n

# FILE LOGGER
logging.file.name           = logs/FIle.log
logging.pattern.file         = My File Log: %d %p %c{0} %m %n

# ROLLING FILE LOGGER
logging.logback.rollingpolicy.file-name-pattern = logs/archived/RollingFile_%d{dd.MM.yyyy}_%i.log
logging.logback.rollingpolicy.max-history        = 30
logging.logback.rollingpolicy.max-file-size      = 1KB
logging.logback.rollingpolicy.total-size-cap     = 10MB
logging.logback.rollingpolicy.clean-history-on-start = false
```

MyController.java

```
package com.ivoronline.springboot_log_logback_config_properties.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

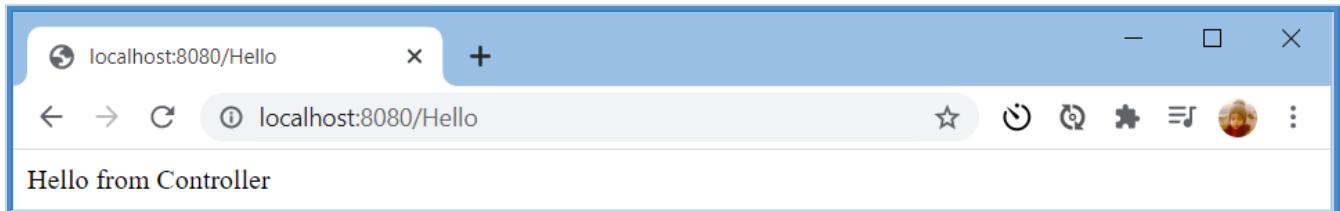
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn("Some warn occurred");
        log.info("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello>



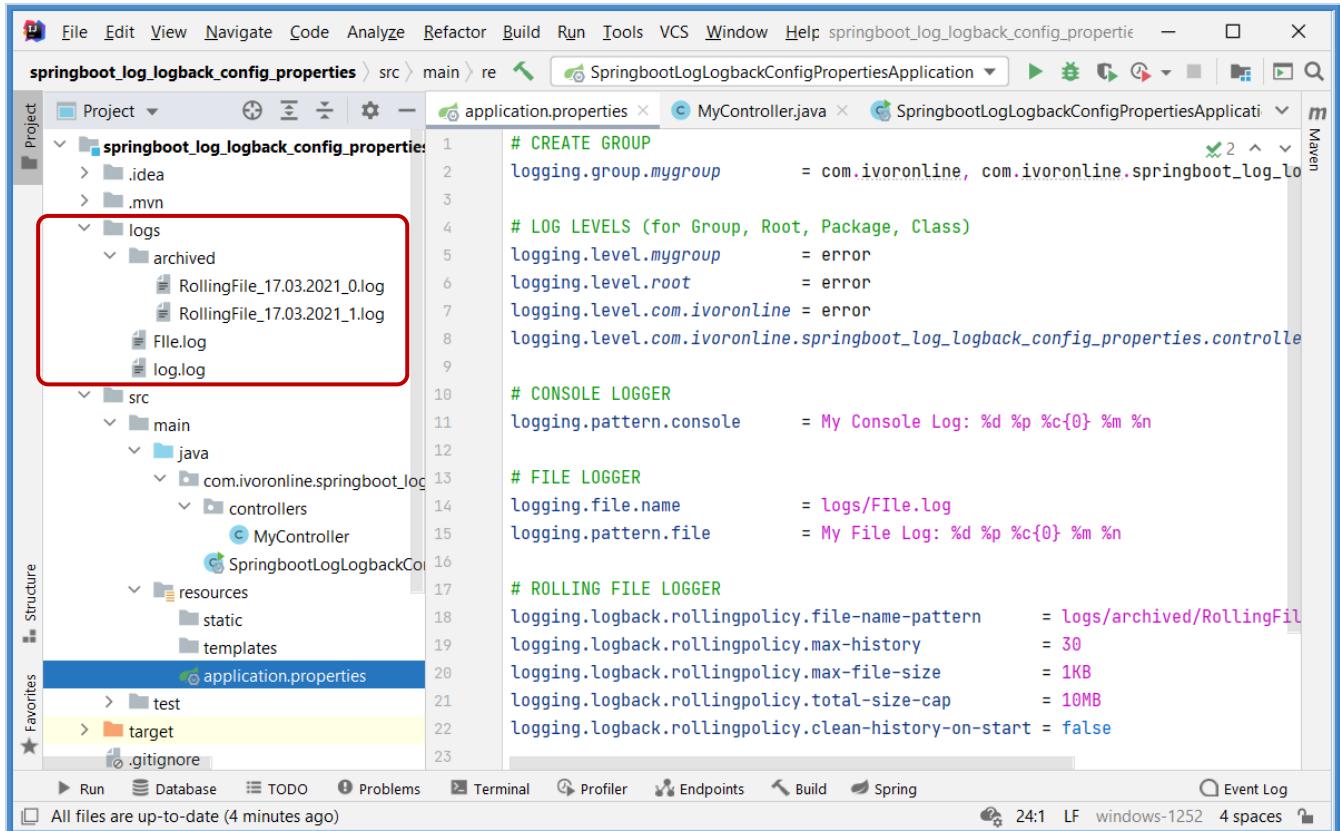
Console

```
My Console Log: 2021-03-17 15:00:31,057 ERROR MyController Some error occurred  
My Console Log: 2021-03-17 15:00:31,059 WARN MyController Some warn occurred  
My Console Log: 2021-03-17 15:00:31,059 INFO MyController Some info occurred
```

logs\log.log

```
My File Log: 2021-03-17 15:00:31,057 ERROR MyController Some error occurred  
My File Log: 2021-03-17 15:00:31,059 WARN MyController Some warn occurred  
My File Log: 2021-03-17 15:00:31,059 INFO MyController Some info occurred
```

Application Structure



pom.xml

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

3.1.8 Log4j

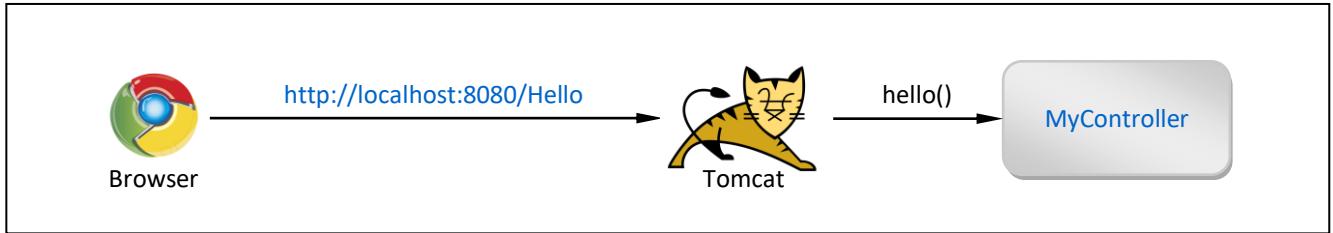
Info

[G] [R]

- This tutorial shows how to use SLF4J with **Log4j**.
- Changes are only needed in **pom.xml**. (add Log4j dependencies, exclude default logging from every Spring Boot Starter)

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_log` (add Spring Boot Starters from the table)
- **Edit File:** `pom.xml` (add Log4j dependencies, exclude default logging from every Starter)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)

pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

MyController.java

```
package com.ivoronline.springboot_log_log4j.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

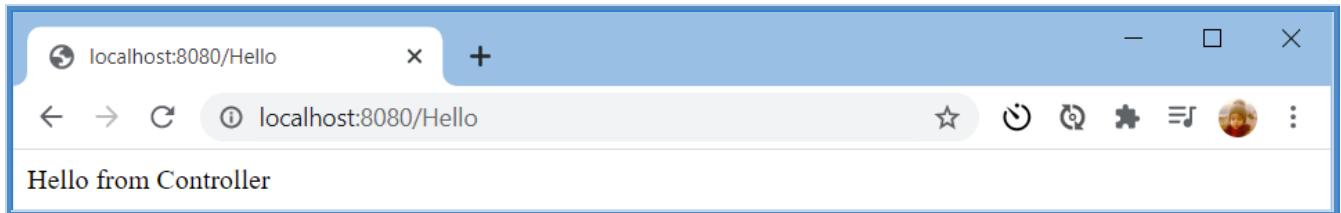
@Controller
public class MyController {

    private static final Logger log = LoggerFactory.getLogger(MyController.class);

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

Results

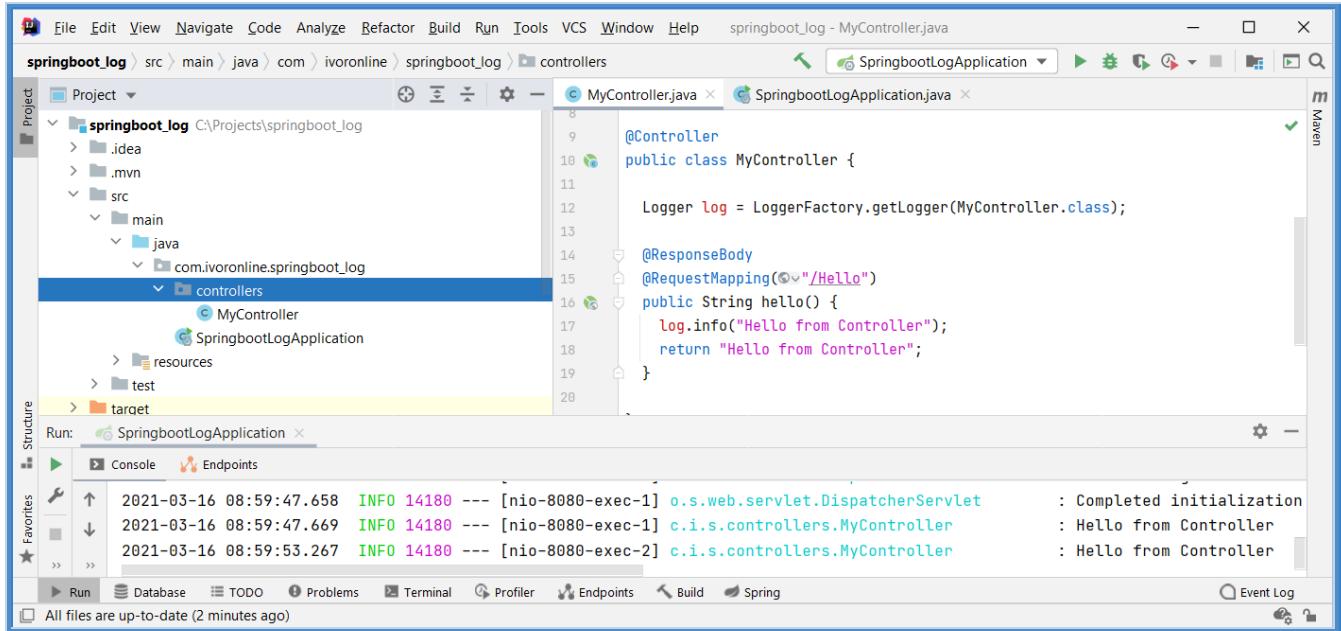
<http://localhost:8080>Hello>



Console

```
2021-03-16 08:59:53.267 INFO 14180 - [nio-8080-exec-2] c.i.s.controllers.MyController : Hello from Controller
```

Application Structure



pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

3.1.9 Log4j - Configure - XML

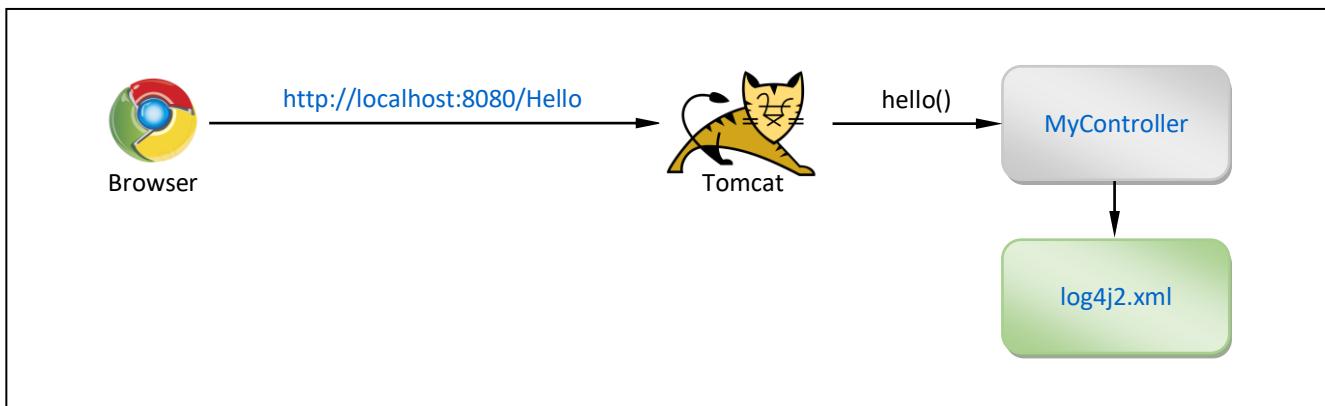
Info

[G] [R] [R]

- This tutorial shows how to configure Log4j by creating `log4j2.xml` file inside `resources` Directory.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project:** `springboot_log_log4j_config_xml` (add Spring Boot Starters from the table)
- Edit File:** `pom.xml` (add Log4j dependencies, exclude default logging from Starter)
- Create File:** `log4j2.xml` (inside Directory resources)
- Create Package:** `controllers` (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)

pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>

    <Appenders>

        <!-- CONSOLE APPENDER -->
        <Console name="MyConsoleAppenderName"> </Console>

        <!-- FILE APPENDER -->
        <File name="MyFileAppenderName" fileName="logs/File.log">
            <PatternLayout pattern="My File Appender: %d %p %c{2} %m %n" />
        </File>

        <!-- ROLLING FILE APPENDER -->
        <RollingFile name      = "MyRollingFileAppenderName"
                      fileName     = "logs/RollingFile.log"
                      filePattern = "logs/${date:MM-yyyy}/RollingFile_%d{dd.MM.yyyy}_{i}.log">

            <!-- PATTERN LAYOUT -->
            <PatternLayout pattern="My Rolling File Appender: %d %p %c{2} %m %n"/>

            <!-- ROLLOVER POLICY (on startup, daily and when the file reaches 10MB) -->
            <Policies>
                <OnStartupTriggeringPolicy          />
                <SizeBasedTriggeringPolicy size="10 MB" />
                <TimeBasedTriggeringPolicy         />
            </Policies>

        </RollingFile>

    </Appenders>

    <Loggers>

        <!-- LOGGER -->
        <Logger name="com.ivoronline.springboot_log_log4j_config_xml.controllers" level="info">
            <AppenderRef ref="MyConsoleAppenderName"   />
            <AppenderRef ref="MyFileAppenderName"       />
            <AppenderRef ref="MyRollingFileAppenderName"/>
        </Logger>

        <!-- PREVENT DEFAULT LOGGER (since no appenders are defined) -->
        <Root level="info"></Root>

    </Loggers>

</Configuration>
```

MyController.java

```
package com.ivoronline.springboot_log_log4j_config_xml.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

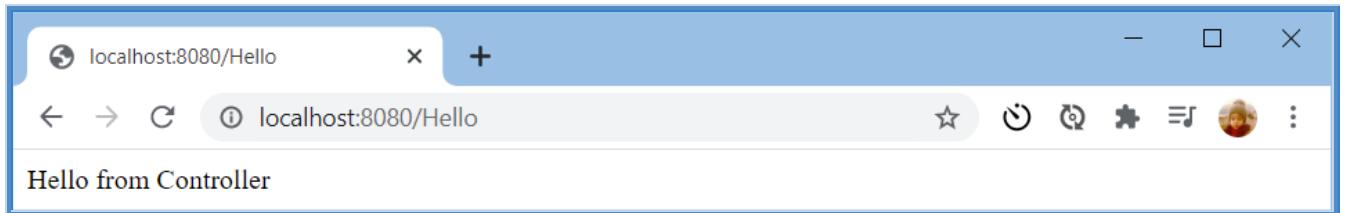
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn ("Some warn occurred");
        log.info ("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello>



Console

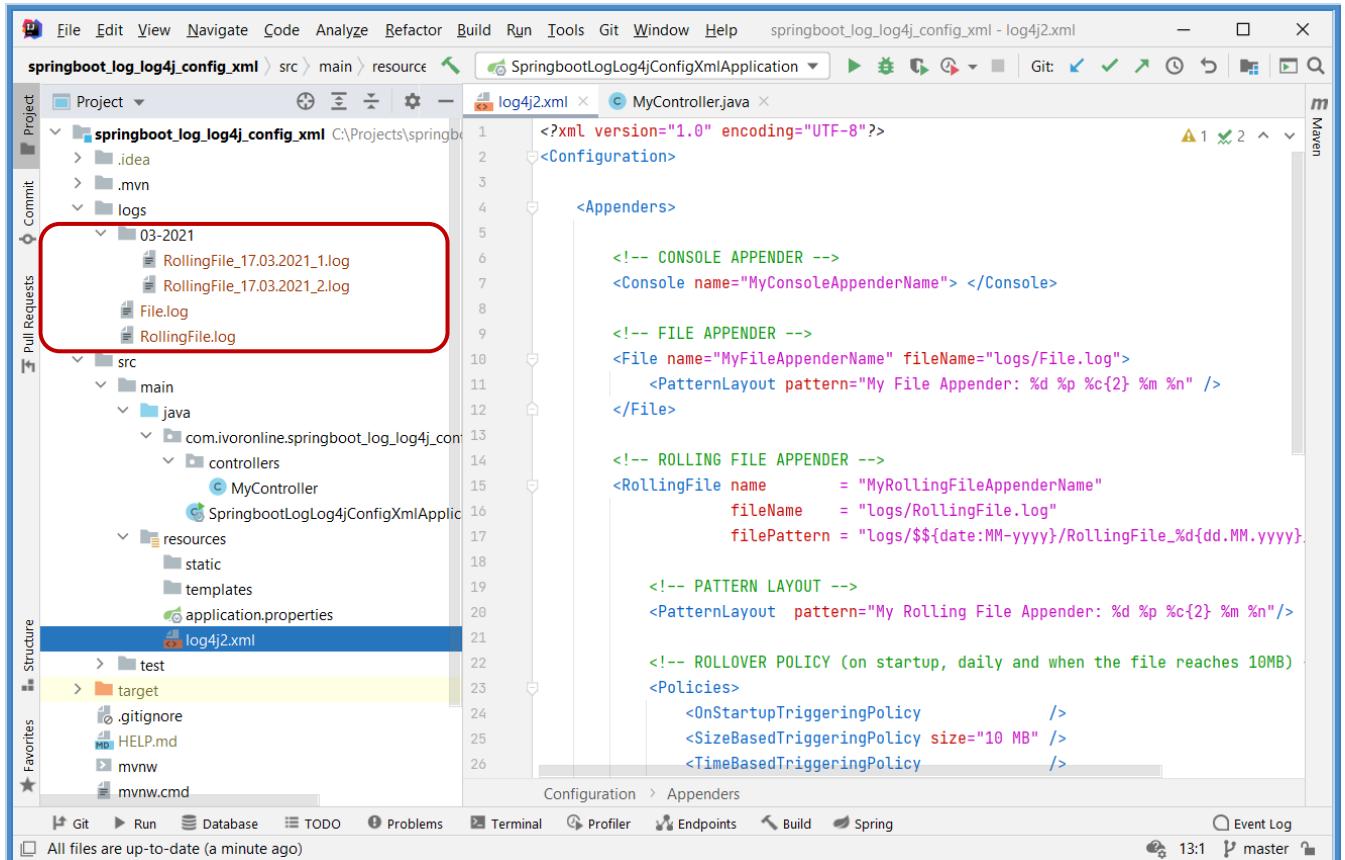
```
Some error occurred  
Some warn occurred  
Some info occurred
```

logs/app.log

(My File Appender: %d %p %c{2} %m %n)

```
My File Appender: 2021-03-16 21:56:24,216 ERROR controllers.MyController Some error occurred  
My File Appender: 2021-03-16 21:56:24,219 WARN controllers.MyController Some warn occurred  
My File Appender: 2021-03-16 21:56:24,220 INFO controllers.MyController Some info occurred
```

Application Structure



```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

3.1.10 Log4j - Configure - Properties

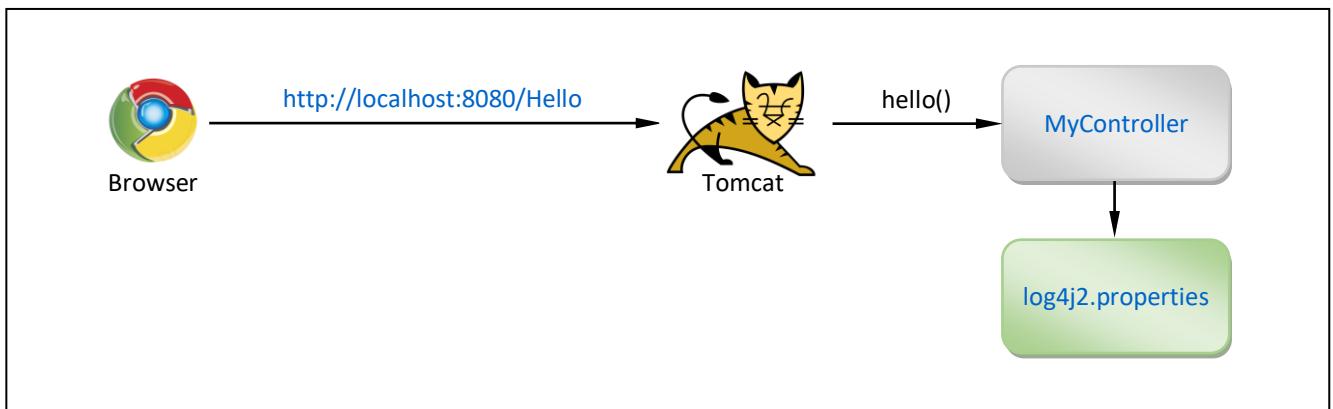
Info

[G] [R]

- This tutorial shows how to configure Log4j by creating **log4j2.properties** file inside **resources** Directory.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project:** `springboot_log_log4j_config_properties` (add Spring Boot Starters from the table)
- Edit File:** `pom.xml` (add Log4j dependencies, exclude default logging)
- Create File:** `log4j2.properties` (inside resources directory)
- Create Package:** `controllers` (inside main package)
- **Create Class:** `MyController.java` (inside controllers package)

pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

log4j2.properties

```
# CONSOLE APPENDER
appender.MyConsoleAppender.type = Console
appender.MyConsoleAppender.name = MyConsoleAppenderName

# FILE APPENDER
appender.MyFileAppender.type = File
appender.MyFileAppender.name = MyFileAppenderName
appender.MyFileAppender.fileName = logs/File.log
appender.MyFileAppender.layout.type = PatternLayout
appender.MyFileAppender.layout.pattern = My File Appender: %d %p %c{2} %m %n

# ROLLING FILE APPENDER
appender.rolling.type = RollingFile
appender.rolling.name = MyRollingFileAppenderName
appender.rolling.fileName = logs/RollingFile.log
appender.rolling.filePattern = logs/archived/RollingFile_%d{dd.MM.yyyy}_%i.log
appender.rolling.layout.type = PatternLayout
appender.rolling.layout.pattern = %d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
appender.rolling.policies.type = Policies
appender.rolling.policies.time.type = TimeBasedTriggeringPolicy
appender.rolling.policies.time.interval = 1
appender.rolling.policies.time.modulate = true
appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
appender.rolling.policies.size.size = 1KB
appender.rolling.strategy.type = DefaultRolloverStrategy
appender.rolling.strategy.max = 20

# LOGERS
logger.MyLogger1.name = com.ivoronline.springboot_log_log4j_config_properties.controllers
logger.MyLogger1.level = info
logger.MyLogger1.appendRef.myappendRef1.ref = MyConsoleAppenderName
logger.MyLogger1.appendRef.myappendRef2.ref = MyFileAppenderName
logger.MyLogger1.appendRef.myappendRef3.ref = MyRollingFileAppenderName

# DISABLE ROOT LOGGER
rootLogger.level
```

MyController.java

```
package com.ivoronline.springboot_log_log4j_config_properties.controllers;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

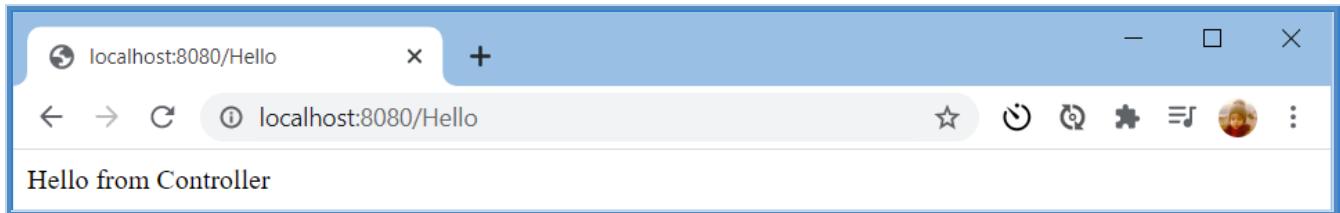
@Controller
public class MyController {

    Logger log = LoggerFactory.getLogger(MyController.class);

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        log.error("Some error occurred");
        log.warn ("Some warn occurred");
        log.info ("Some info occurred");
        log.debug("Some debug occurred");
        log.trace("Some trace occurred");
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello>



Console

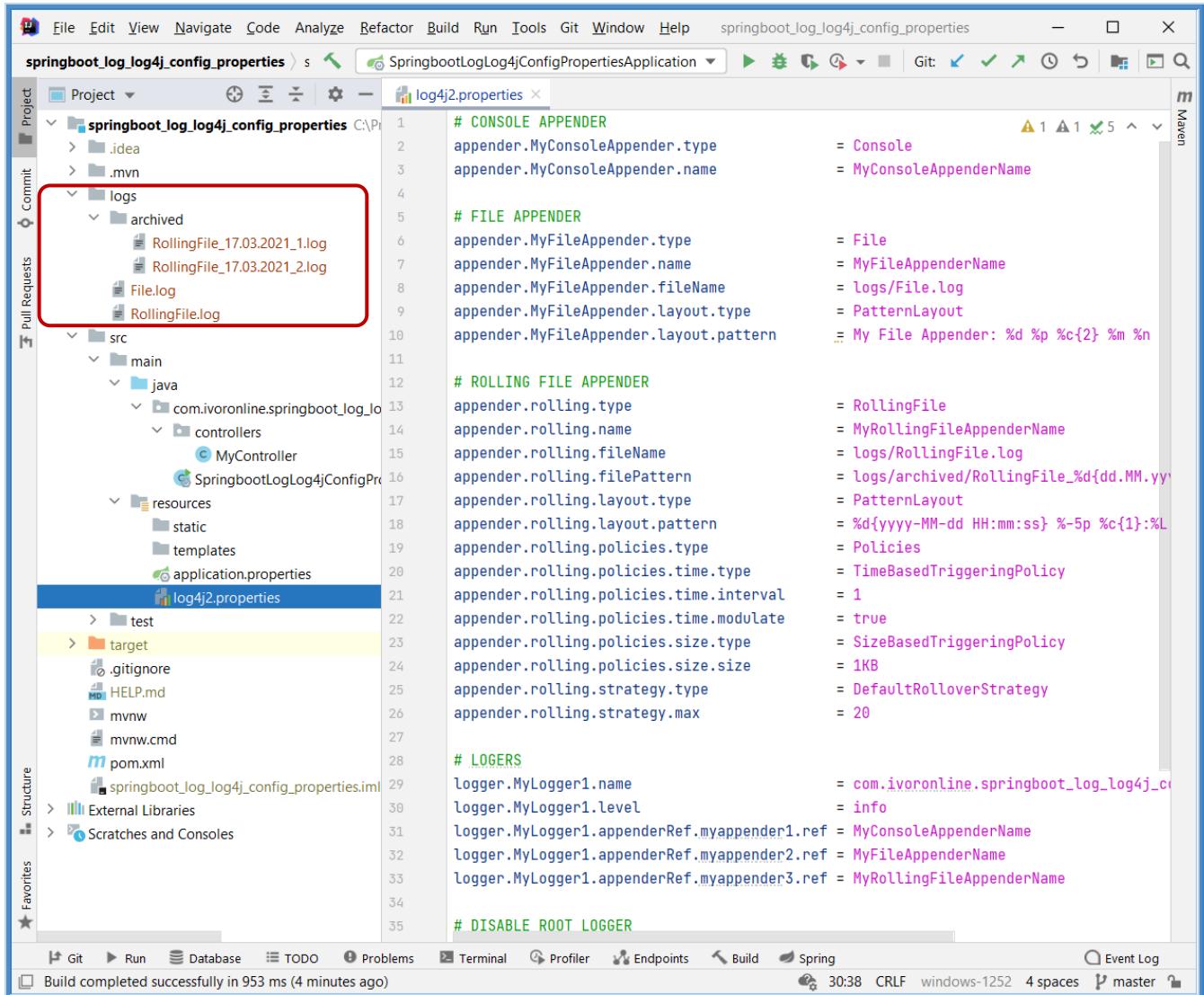
```
Some error occurred  
Some warn occurred  
Some info occurred
```

logs/app.log

(My File Appender: %d %p %c{2} %m %n)

```
My File Appender: 2021-03-16 21:56:24,216 ERROR controllers.MyController Some error occurred  
My File Appender: 2021-03-16 21:56:24,219 WARN controllers.MyController Some warn occurred  
My File Appender: 2021-03-16 21:56:24,220 INFO controllers.MyController Some info occurred
```

Application Structure



```
<dependencies>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

</dependencies>
```

3.2 AOP Annotations

Info

[R]

- Following tutorials show how to use AOP Annotations to log events.
- AOP stands for Aspect Orientated Programming.
- AOP is a way for adding behavior to existing code without modifying that code.
- By using Annotations business logic code isn't polluted with logging code.

Examples

```
Signature method      = joinPoint.getSignature();           //String com.ivoronline...MyController.hello()
String   classPathName = method.getDeclaringTypeName();    //com.ivoronline...controllers.MyController
String   methodName    = method.getName();                  //hello

Class   methodClass   = method.getDeclaringType();         //class com.ivoronline...controllers.MyController
String   classPathName2 = methodClass.getName();            //com.ivoronline...controllers.MyController
String   packageName   = methodClass.getPackageName();       //com.ivoronline...controllers
String   className     = methodClass.getSimpleName();        //MyController
```

3.2.1 Log Start End

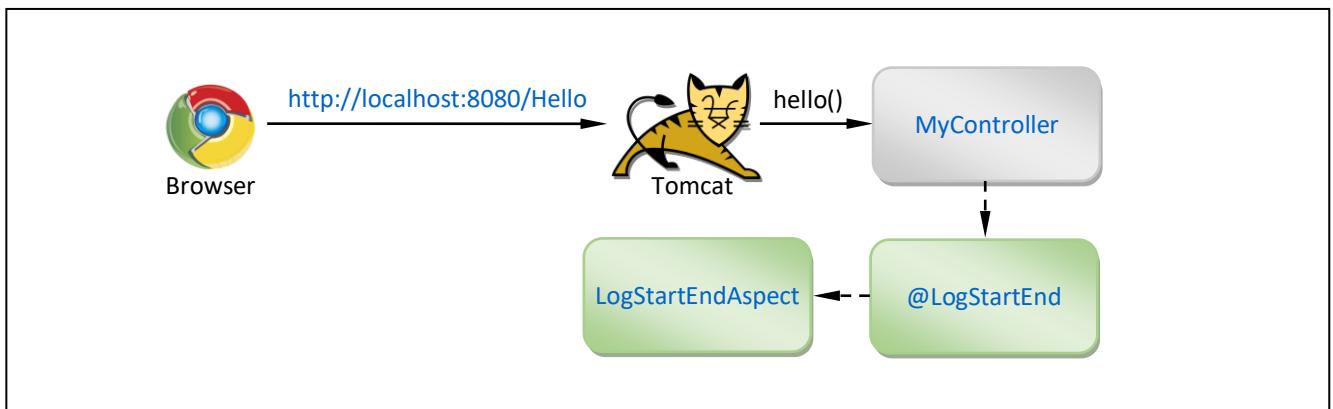
Info

[G] [R]

- This tutorial shows how to use AOP Annotation to Log Method's Execution Time.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_aop_logging` (add Spring Boot Starters from the table)
- Edit File: `pom.xml` (add AOP dependency)
- Create Package: `log` (inside main package)
 - Create Class: `LogStartEnd.java` (inside log package)
 - Create Class: `LogStartEndAspect.java` (inside log package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)

pom.xml

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

LogStartEnd.java

```
package com.ivorononline.springboot_aop_log_startend.log;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface LogStartEnd { }
```

LogStartEndAspect.java

```
package com.ivoronline.springboot_aop_log_startend.log;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.Signature;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LogStartEndAspect {

    @Around("@annotation(LogStartEnd)")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

        //GET METHOD PARAMETERS
        String methodName = joinPoint.getSignature().getName(); //hello
        String className = joinPoint.getSignature().getDeclaringType().getSimpleName(); //MyController

        //EXECUTE BEFORE METHOD
        System.out.println("STARTED METHOD: " + className + "." + methodName + "()");

        //EXECUTE METHOD
        Object proceed = joinPoint.proceed();

        //EXECUTE AFTER METHOD
        System.out.println("ENDED METHOD: " + className + "." + methodName + "()");

        //RETURN METHOD RESULT
        return proceed;
    }
}
```

MyController.java

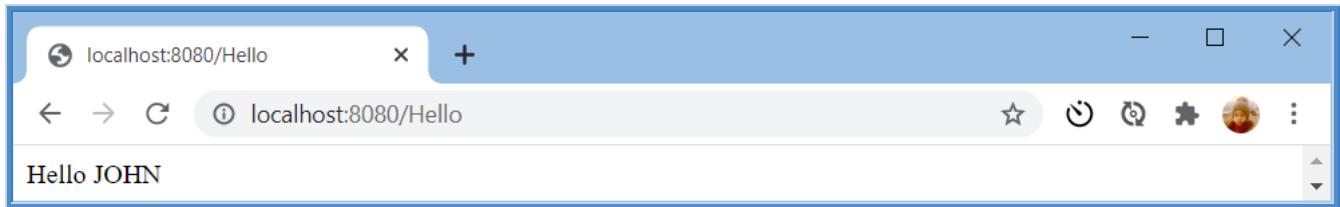
```
import com.ivoronline.springboot_aop_log_startend.log.LogStartEnd;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

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

Result

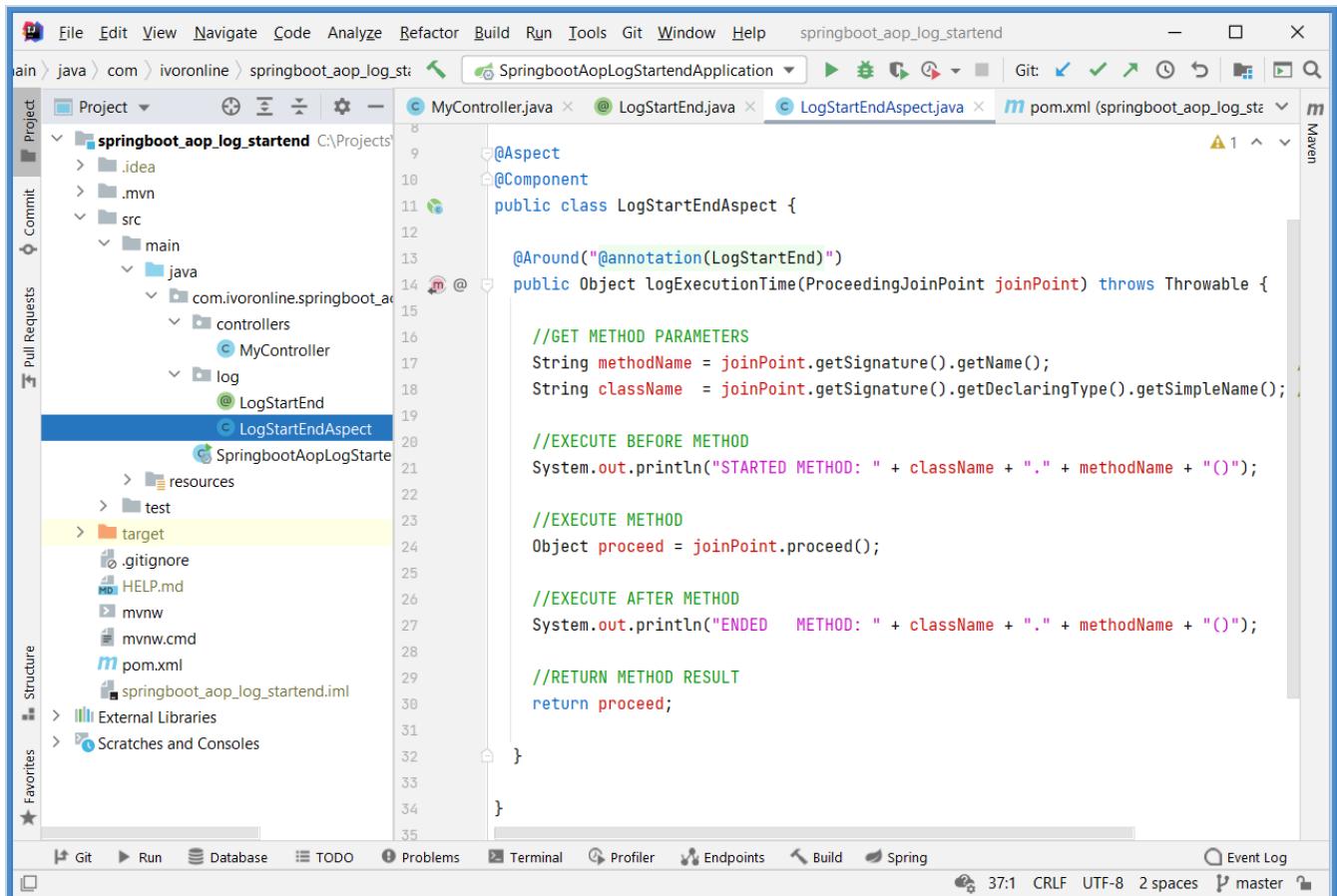
<http://localhost:8080>Hello>



Console

```
STARTED METHOD: MyController.hello()
ENDED   METHOD: MyController.hello()
```

Run Test Class: PersonTest



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-aop</artifactId>
    </dependency>

</dependencies>
```

3.2.2 Log Execution Time

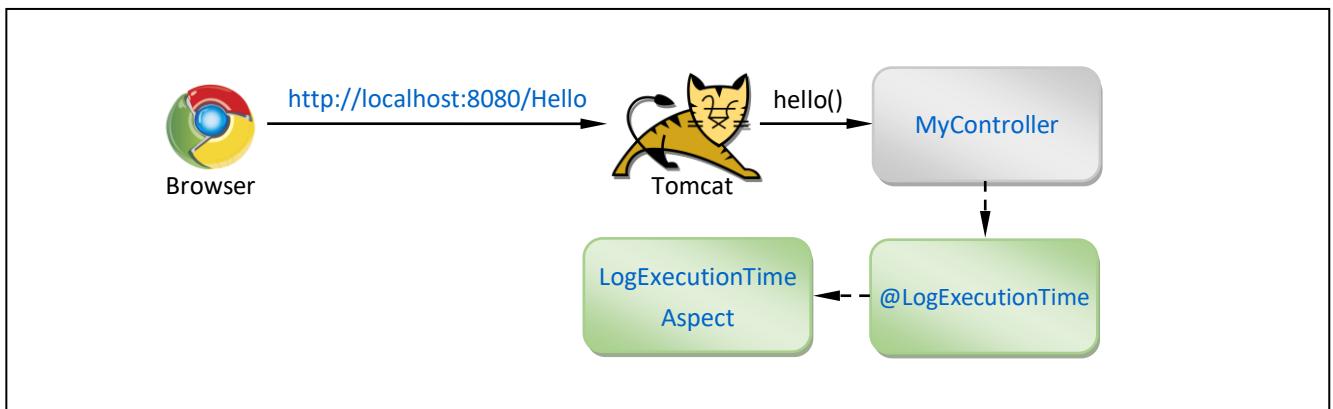
Info

[G] [R]

- This tutorial shows how to use AOP Annotation to Log Method's Execution Time.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_aop_logging` (add Spring Boot Starters from the table)
- Edit File: `pom.xml` (add AOP dependency)
- Create Package: `log` (inside main package)
 - Create Class: `LogExecutionTime.java` (inside log package)
 - Create Class: `LogExecutionTimeAspect.java` (inside log package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)

pom.xml

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

LogExecutionTime.java

```
package com.ivoronline.springboot_aop_logging.log;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface LogExecutionTime { }
```

LogExecutionTimeAspect.java

```
package com.ivoronline.springboot_aop_logging.log;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LogExecutionTimeAspect {

    @Around("@annotation(LogExecutionTime)")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

        //EXECUTE BEFORE METHOD
        long start = System.currentTimeMillis();

        //EXECUTE METHOD
        Object proceed = joinPoint.proceed();

        //EXECUTE AFTER METHOD
        long executionTime = System.currentTimeMillis() - start;
        System.out.println(joinPoint.getSignature() + " executed in " + executionTime + "ms");

        //RETURN METHOD RESULT
        return proceed;
    }
}
```

MyController.java

```
package com.ivoronline.springboot_aop_logging.controllers;

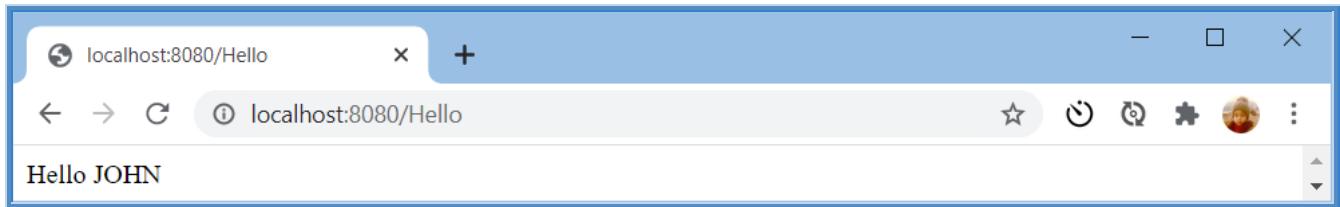
import com.ivoronline.springboot_aop_logging.log.LogExecutionTime;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

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

Result

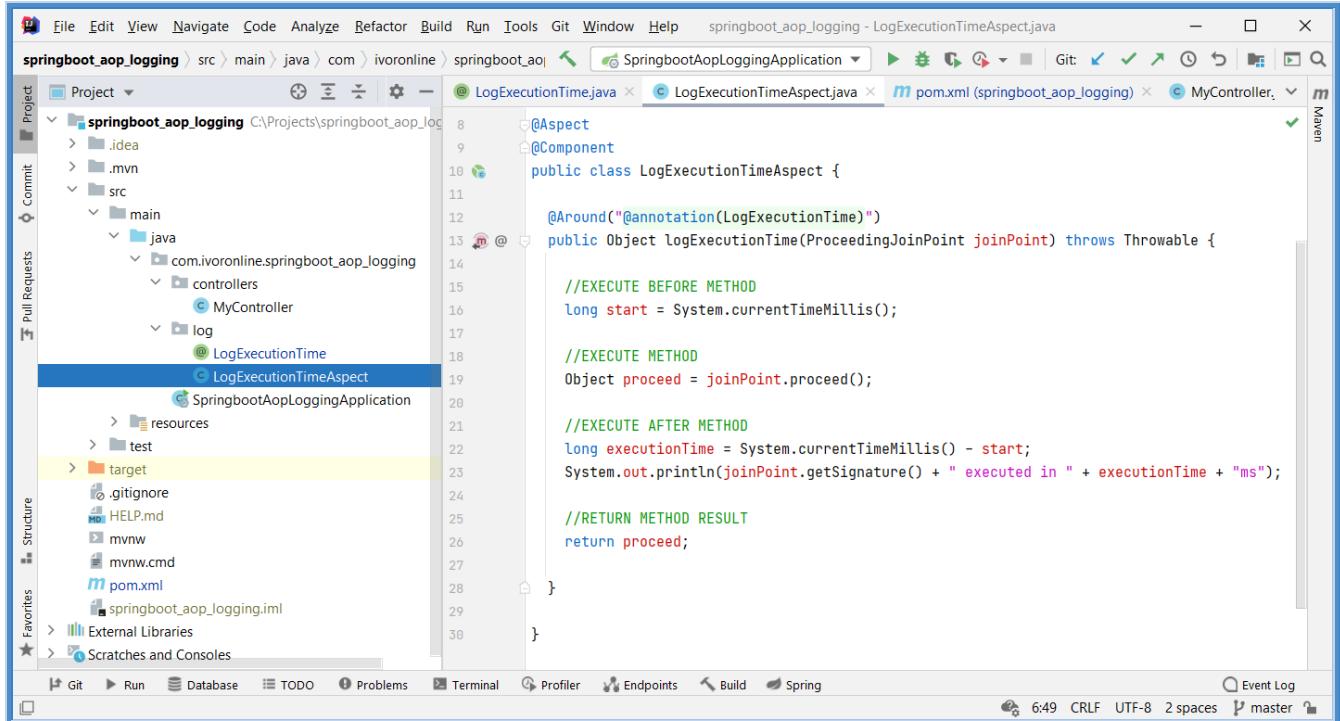
<http://localhost:8080>Hello>



Console

```
String com.ivoronline.springboot_aop_logging.controllers.MyController.hello() executed in 10ms
```

Run Test Class: PersonTest



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-aop</artifactId>
    </dependency>

</dependencies>
```

4 Testing

Info

- Following tutorials show how to use different technologies to test Application.

4.1 JUnit 5

Info

- Following tutorials show how to use **JUnit 5** to test Application.
- Both Application and Test Classes will have the same Package structure (for easier navigation) where
 - **Application Classes** are organized under **src\main**
 - **Test Classes** are organized under **src\test**
- You can **Create Test Class** either
 - **Manually** (manually create Package and Test Class under **src\test\java**)
 - **Automatically** (IntelliJ automatically recreates Packages and creates Test Class under **src\test\java**)
- You can **Run Tests** by
 - Right Clicking on the Package or Class (inside Project Hierarchy)
 - Right Clicking on the Class or Method Name (inside opened Class File)
 - Clicking left from Class or Method Name (inside opened Class File)

Application Structure

(created Test Packages and Classes)

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_junit". The "main" directory contains ".idea", ".mvn", and "src". The "src/main/java/com.ivoronline.springboot_junit/entities" package contains a "Person" class. The "src/test/java/com.ivoronline.springboot_junit/entities" package contains a "PersonTest" class, which is currently selected and shown in the code editor.
- Code Editor:** Displays the Java code for "PersonTest.java". The code includes imports for static assertions, a test method named "setName" with a test annotation, and a verification step using assertEquals.
- Status Bar:** Shows "Tests passed: 1 (today 11:51 AM)".
- Bottom Bar:** Includes icons for Run, Database, TODO, Problems, Terminal, Profiler, Endpoints, Build, Spring, and Event Log, along with file encoding settings (8:8 CRLF UTF-8 2 spaces).

4.1.1 Create Test Class - Manually

Info

- This tutorial shows how to manually create Test Class.
- We want Test Classes to be placed inside the same Package structure as the Classes that are being tested.
- For that reason we will also manually create the same Package structure (in our example it is just the entities Package).

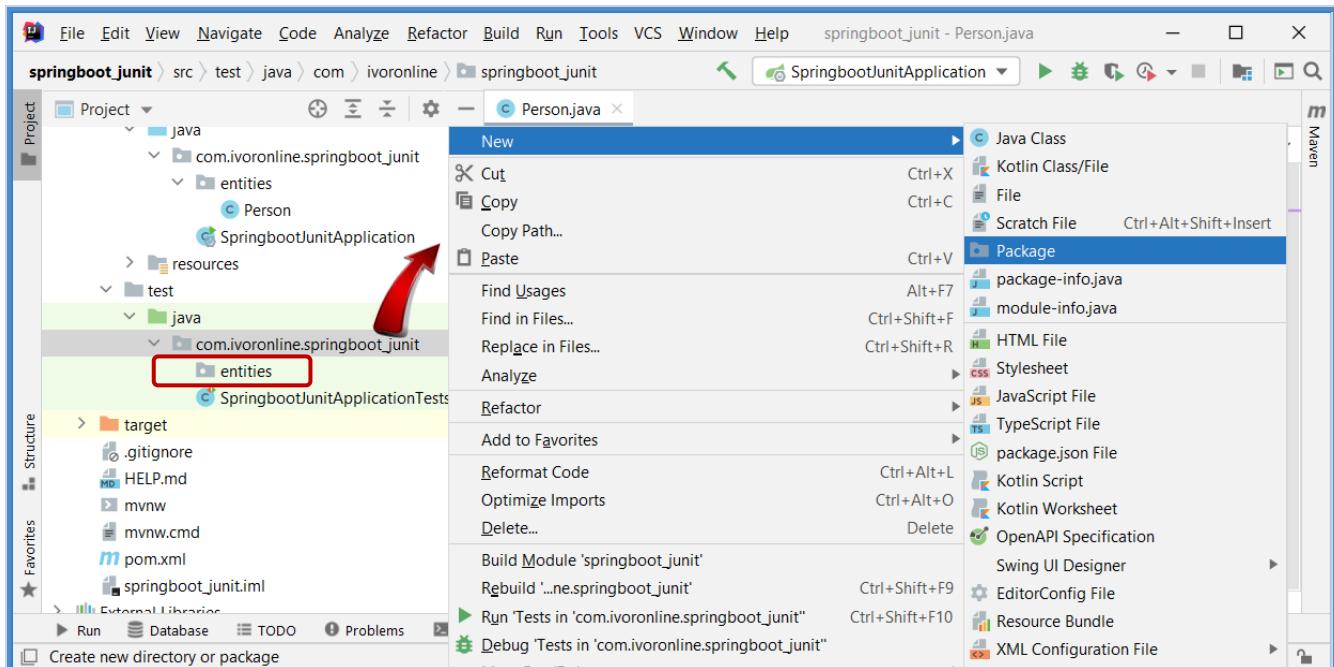
Content

Create Package
Create Test Class
Result

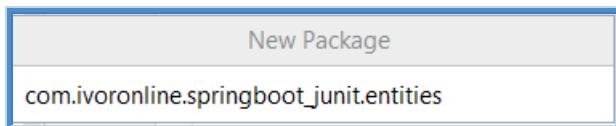
Create Package

- RC on src\test\java\com\ivoronline\springboot_junit
- New
- Package
- Name: com.ivoronline.springboot_junit.entities

RC on src\test\java\com\ivoronline\springboot_junit - New - Package



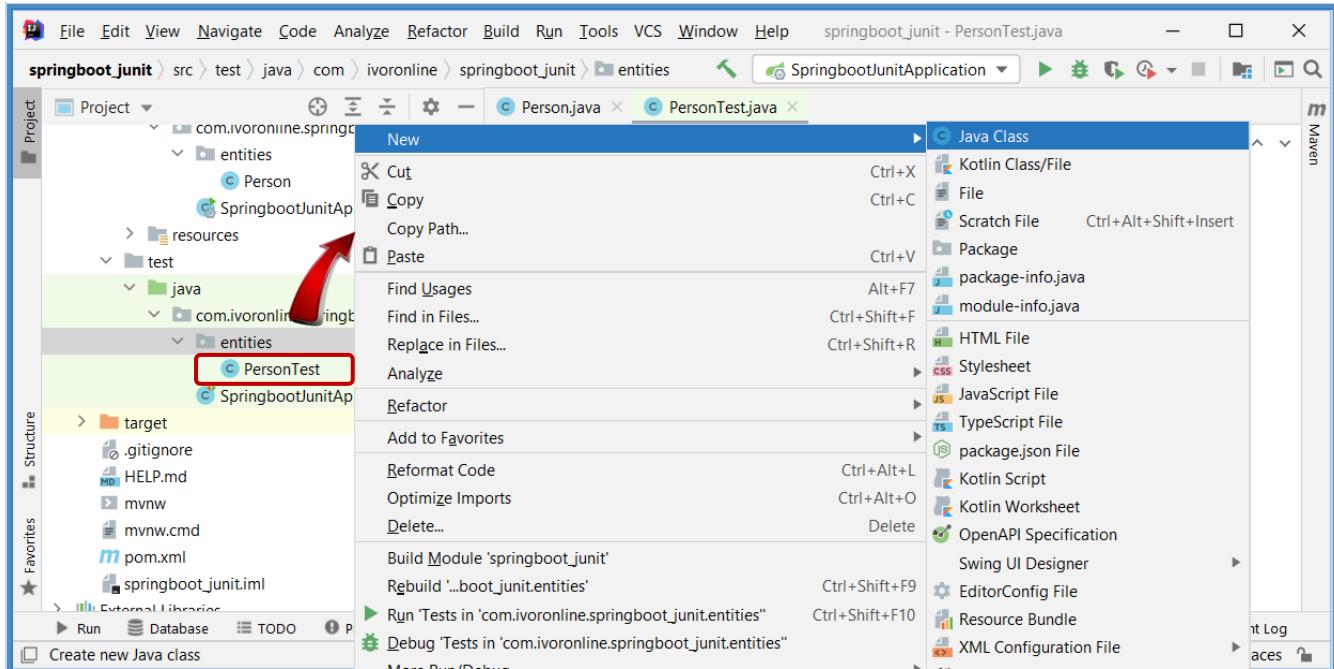
entities



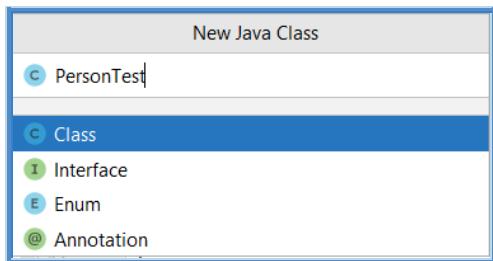
Create Test Class

- RC on entities
- New
- Java Class
- Name: **PersonTest**
- Copy [PersonTest.java](#)

RC on entities - New - Java Class



PersonTest



PersonTest.java

```
package com.ivoronline.springboot_junit.entities;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PersonTest {

    @Test
    void setName() {

        //PERFORM ACTION
        Person person = new Person();
        person.setName("john");

        //TEST RESULT
        assertEquals("JOHN", person.name);

    }

}
```

Result

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_junit". The "entities" package in the "Person" class is selected.
- Code Editor:** Displays the PersonTest.java file. The code is as follows:

```
package com.ivoronline.springboot_junit.entities;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PersonTest {

    @Test
    void setName() {
        //PERFORM ACTION
        Person person = new Person();
        person.setName("john");

        //TEST RESULT
        assertEquals("JOHN", person.name);
    }
}
```

The code editor includes syntax highlighting for Java and JUnit annotations. The status bar at the bottom indicates a successful build: "Build completed successfully in 2 sec, 147 ms (54 minutes ago)".

4.1.2 Create Test Class - Automatically

Info

- This tutorial shows how IntelliJ can automatically create Test Class by selecting Methods you want to test.
- IntelliJ will create new Test Class under `main\test\com.ivoronline.springboot_junit`
 - created Test Class will have the same name as the Class that contains selected Methods but with additional **Test** suffix
(if selected Methods were from the Class **Person** then IntelliJ will create Test Class named **PersonTest**)
 - IntelliJ will also recreate the same Packages
(if Class Person was in the Package **entities**, then Class PersonTest will also be placed in the Package entities)

Content

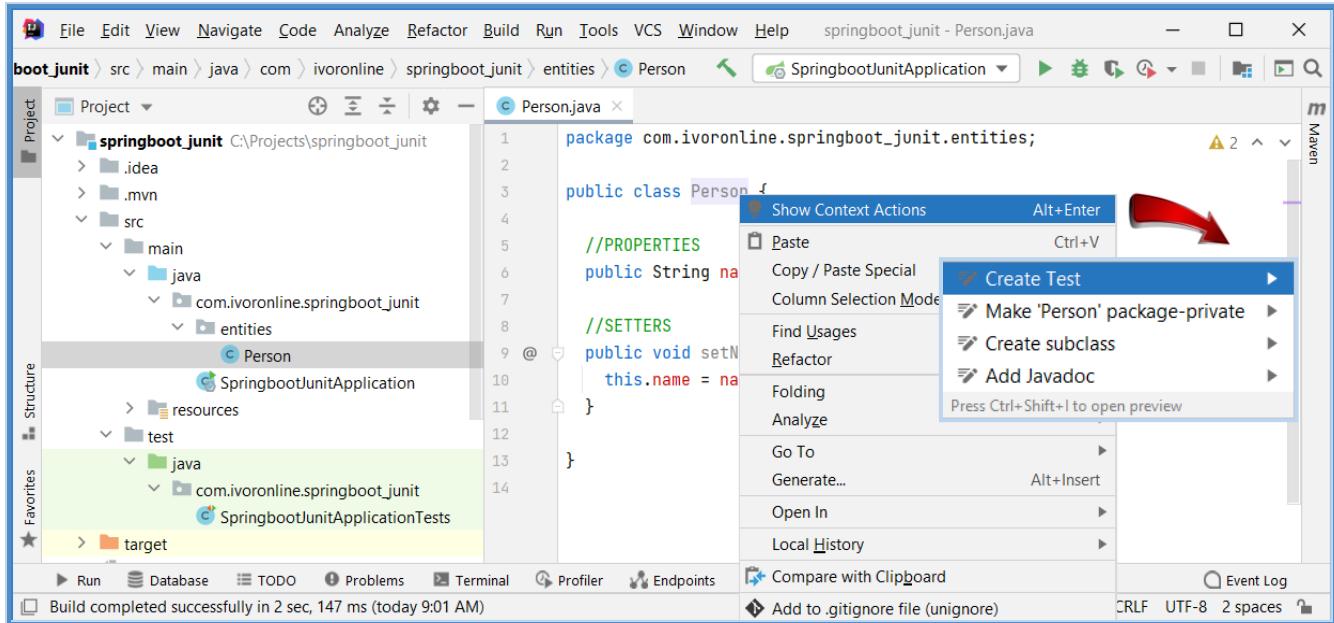
Create Test Class

Result

Create Test Class

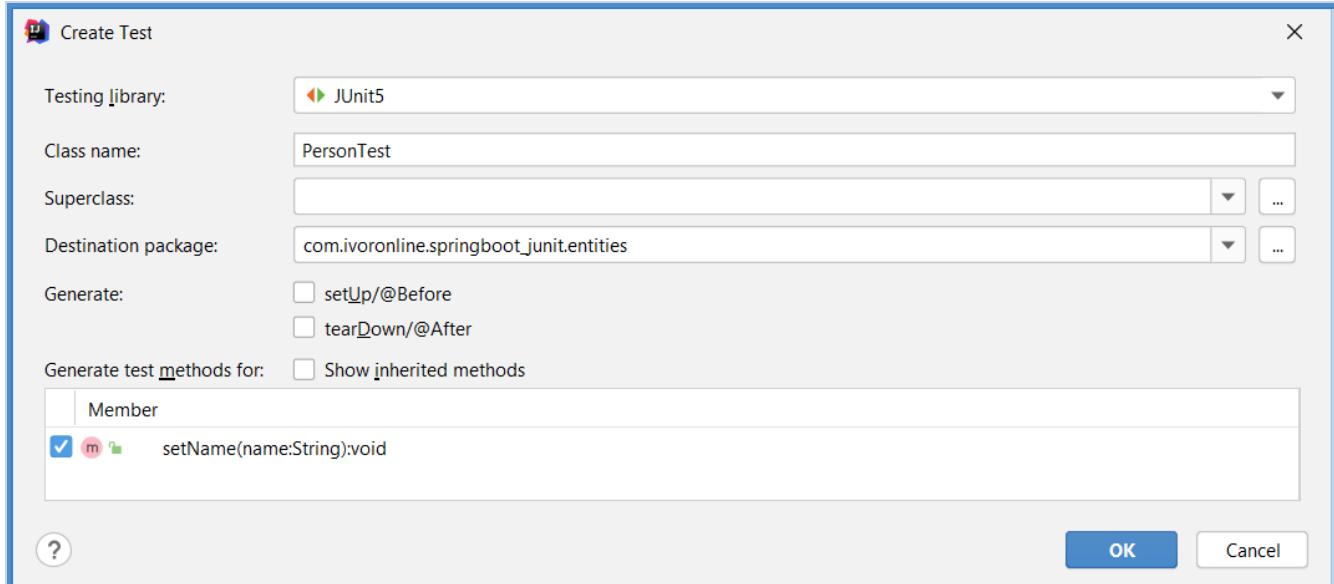
- Choose one
 - RC on Class - Show Context Actions (on Class which Methods you want to test)
 - LC on Class - Alt + Enter
- Create Test
- Select Methods (which you want to test)
- OK (creates Test Class under the same Package structure as Class being tested)

RC on Class



Select Methods

(which you want to test)



Result

Automatically created Test Class

(with the same Package structure)

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_junit". The "entities" package contains a "Person" class, which is selected. The "test" package contains a "PersonTest" class, which is also selected.
- Code Editor:** Displays the content of the "PersonTest.java" file. The code is as follows:

```
package com.ivoronline.springboot_junit.entities;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PersonTest {

    @Test
    void setName() {
        //PERFORM ACTION
        Person person = new Person();
        person.setName("john");

        //TEST RESULT
        assertEquals("JOHN", person.name);
    }
}
```

The code editor shows syntax highlighting for Java and JUnit annotations. The "Person" class is also visible in the editor, though it's not the active tab.

4.1.3 Run Tests

Info

- This tutorial shows how to run all Tests within selected Test **Package** or **Class**, or how to run a single Test **Method**.
- You can Run Tests by
 - Right Clicking on the Package or Class (inside Project Hierarchy)
 - Right Clicking on the Class or Method Name (inside opened Class File)
 - Clicking left from Class or Method Name (inside opened Class File)

Content

[Run Tests within Package](#)

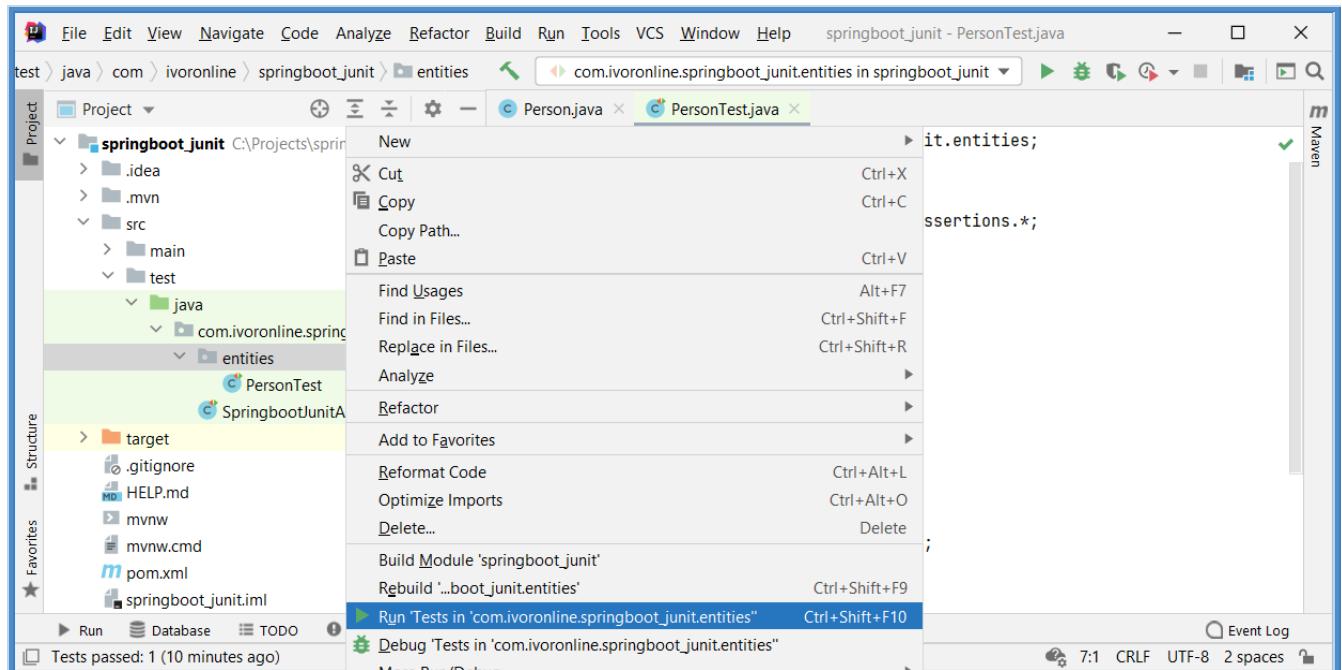
[Run Tests within Class](#)

[Run specific Test Method](#)

Run Tests within Package

- RC on entities
- Run Tests in com.ivoronline.springboot_junit.entities

RC on Test Package - Run Tests in com.ivoronline.springboot_junit.entities

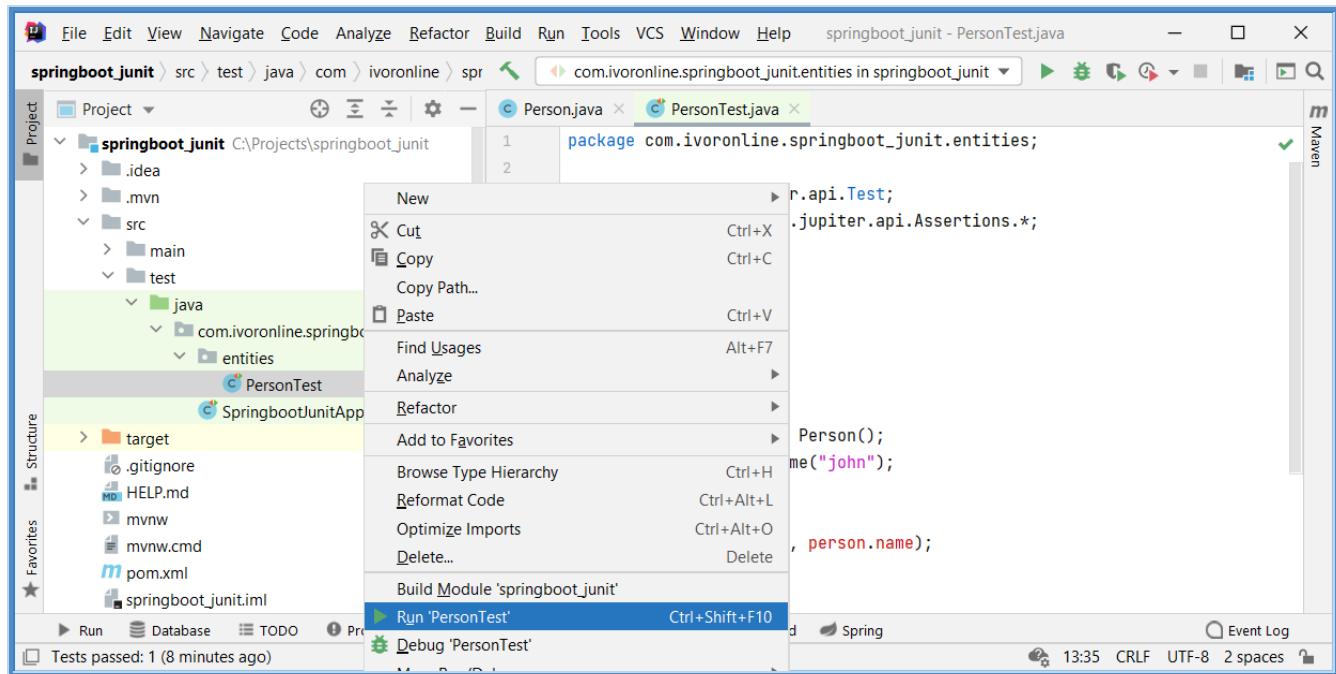


Run Tests within Class

- Option 1
 - RC on Test Class
 - Run PersonTest
- Option 2
 - Open Test Class
 - RC on Class Name
 - Run PersonTest
- Option 3
 - Open Test Class
 - Click on Run Test (left from Class Name)
 - Run PersonTest

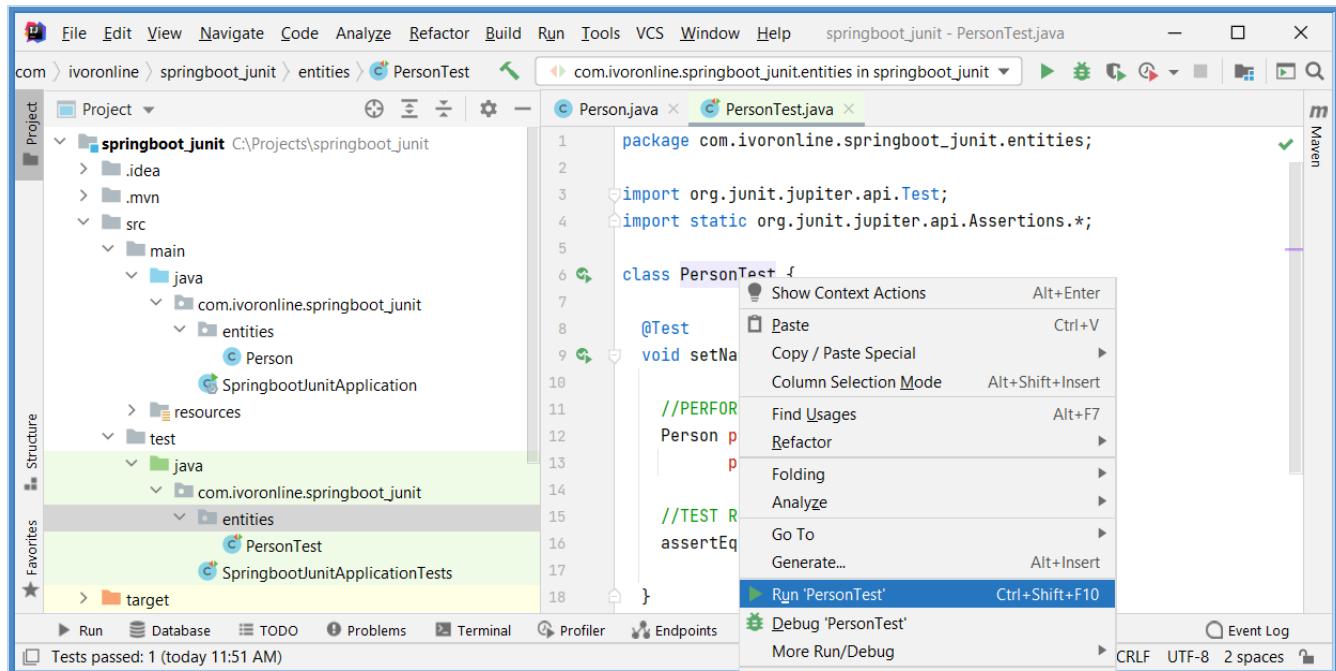
Option 1: RC on Test Class - Run PersonTest

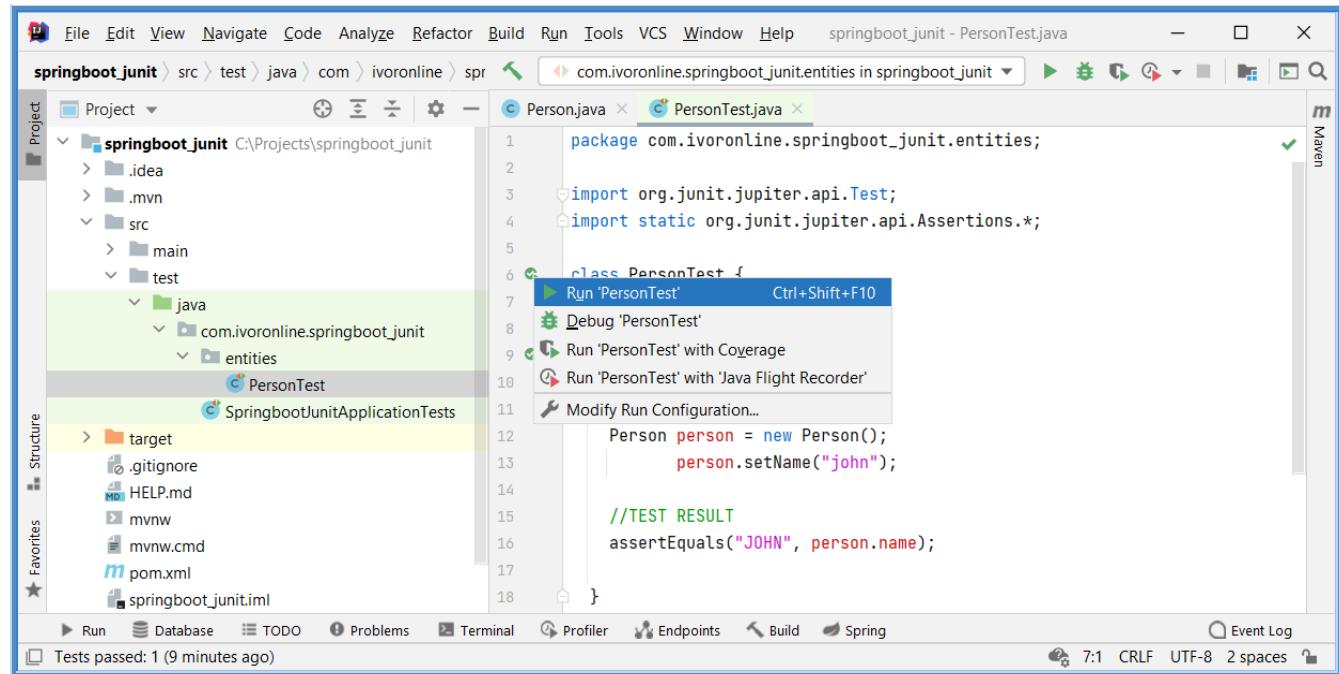
(inside Project Explorer)



Option 2: Open Test Class - RC on Class Name - Run PersonTest

(inside opened Class File)

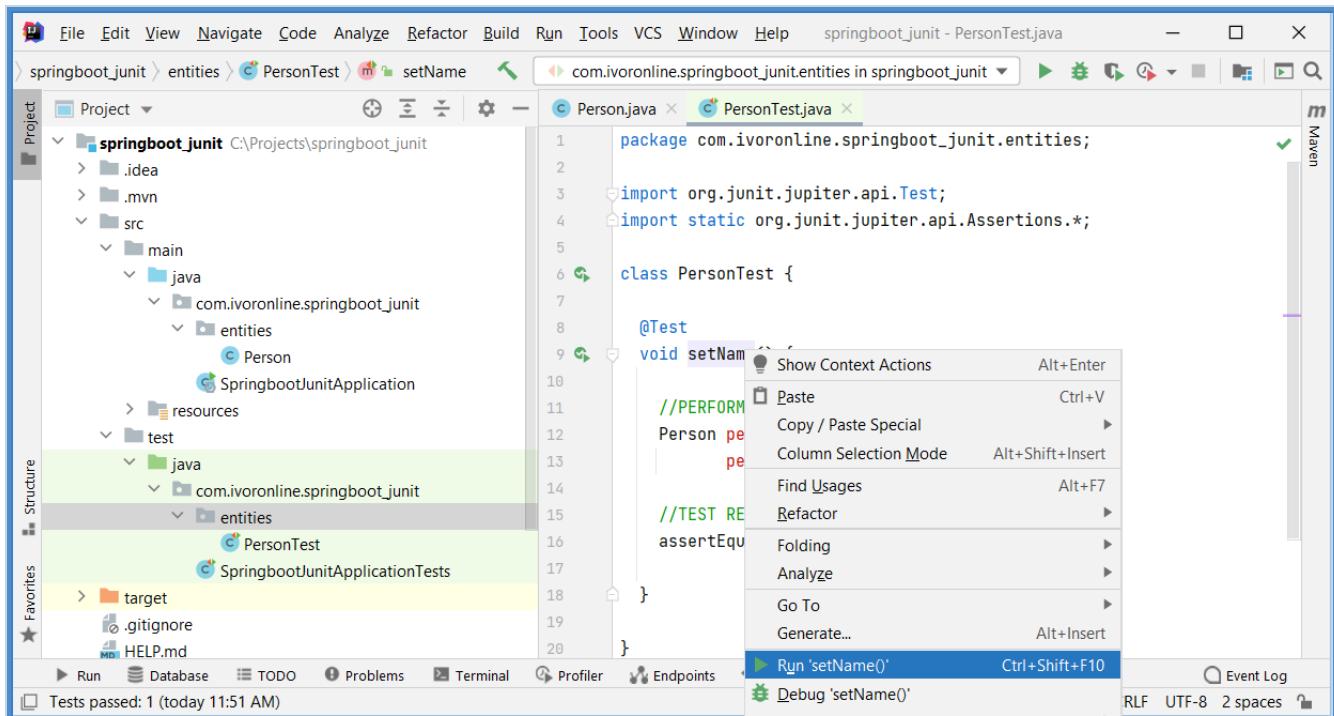




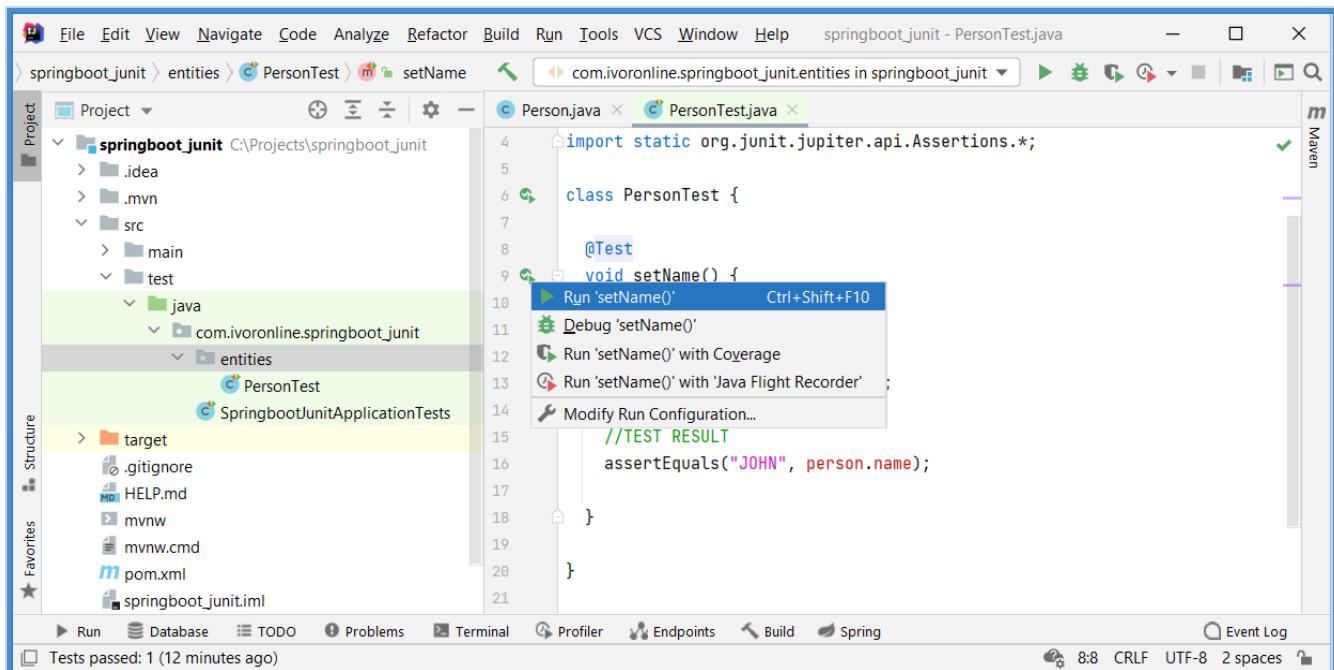
Run specific Test Method

- Option 1
 - Open Test Class
 - RC on Method Name
 - Run setName()
- Option 2
 - Open Test Class
 - Click on Run Test (left from Method Name)
 - Run setName()

Option 1: Open Test Class - RC on Method Name - Run setName()



Option 2: Open Test Class - Click on Run Test (left from Method Name) - Run setName()



4.1.4 Example

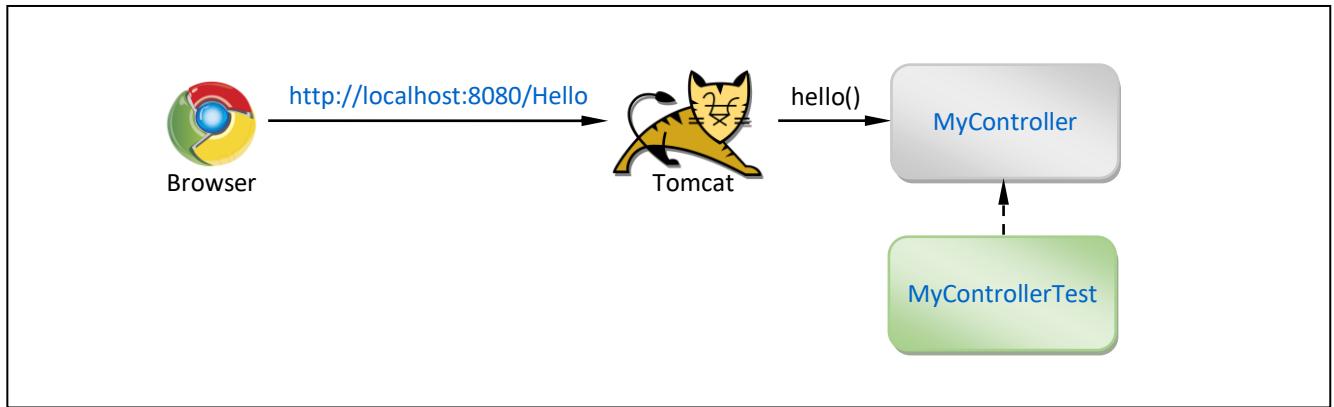
Info

[G]

- This tutorial shows a basic example of JUnit 5 Test Method.
- Tutorial isn't using `@SpringBootTest` meaning that `@Autowired` isn't working which is why we use `new MyController()`.
- This approach is useful when we don't need Spring Boot Context so that tests could run faster.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- [Create Project:](#) `springboot_junit` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit.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";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit.controllers;

import com.ivoronline.springboot_junit.controllers.MyController;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class MyControllerTest {

    @Test
    void hello() {

        //PERFORM ACTION
        MyController myController = new MyController();
        String result = myController.hello();

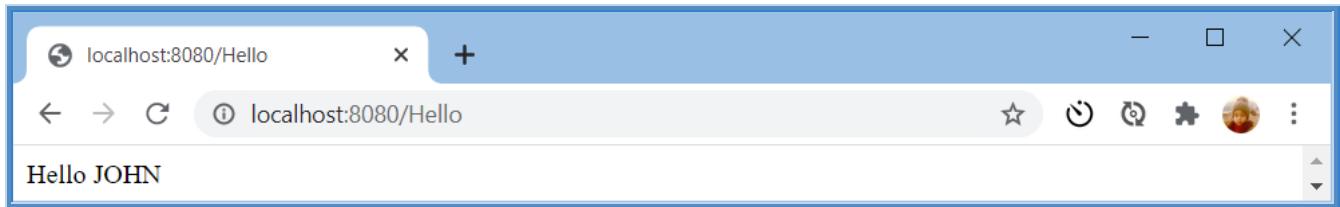
        //TEST RESULT
        assertEquals("Hello from Controller", result);

    }

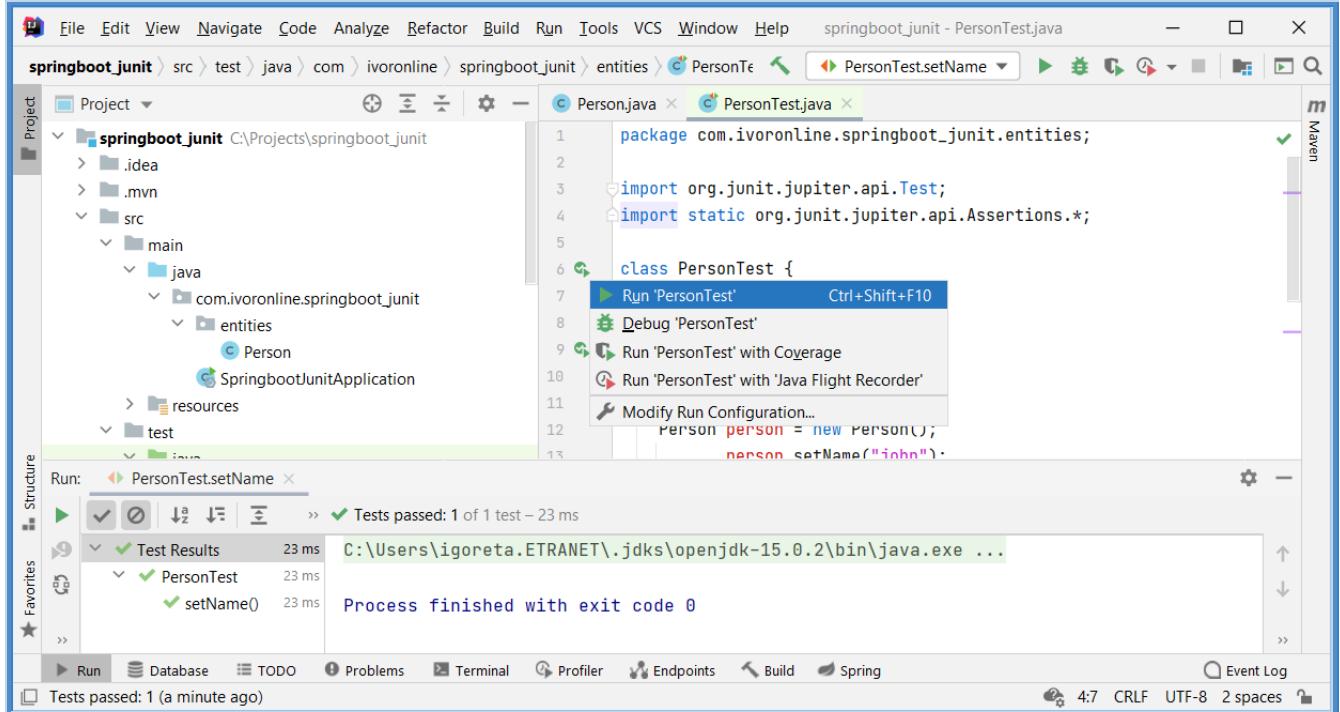
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

4.1.5 @SpringBootTest

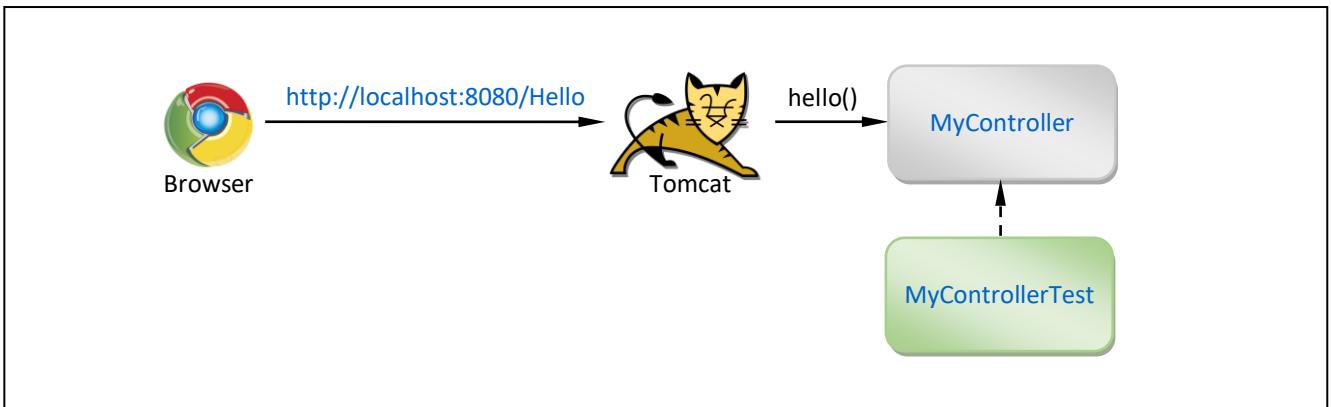
Info

[G]

- This tutorial shows how to use `@SpringBootTest` to create Spring Boot **Context** for testing. This instantiates Controller and other Beans making `@Autowired` work inside Application and `@Test` Methods. Otherwise
 - `@Autowired` wouldn't work neither inside the Application nor in the Test Methods
 - therefore we would need to use `Controller controller = new Controller()` inside the `@Test` Method instead
 - and if inside Controller we have `@Autowired Repository repository, repository` will not get instantiated
- Creating Spring Boot Context is time consuming operation but it is done only once (and not for every Test Method). In other words Context gets cached between Test Methods.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_junit_test_controller` (add Spring Boot Starters from the table)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Test Class:** `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit_test_controller.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";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_test_controller.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

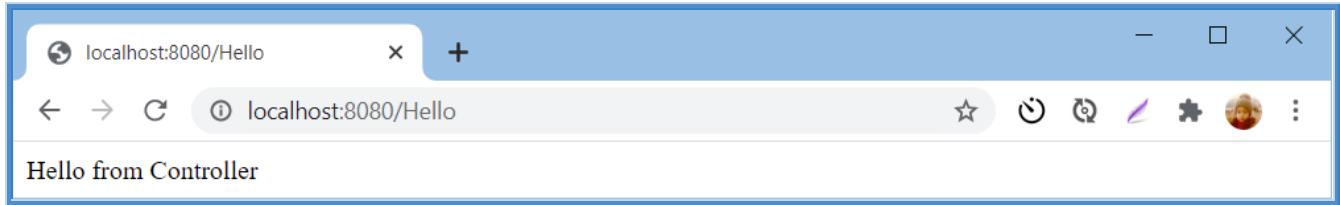
    @Autowired MyController myController;

    @Test
    void hello() {
        String result = myController.hello();
        assertEquals("Hello from Controller", result);
    }

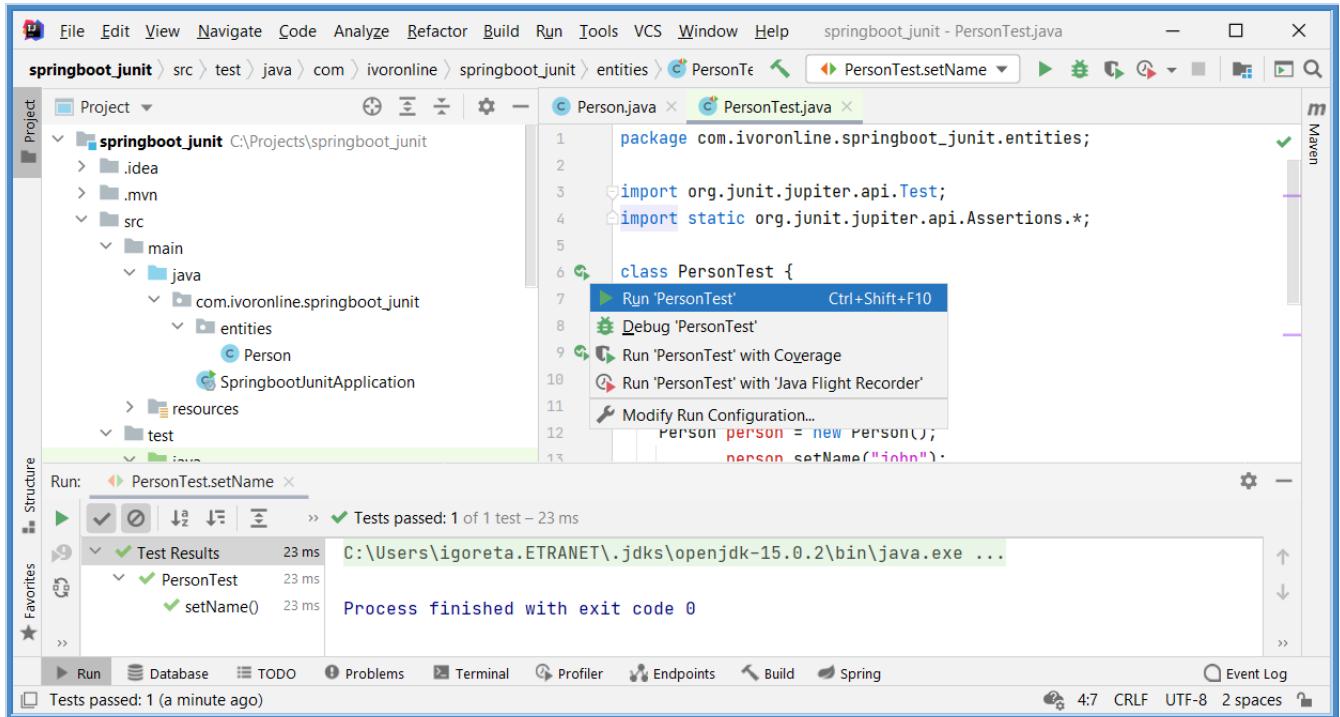
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

4.1.6 @Test

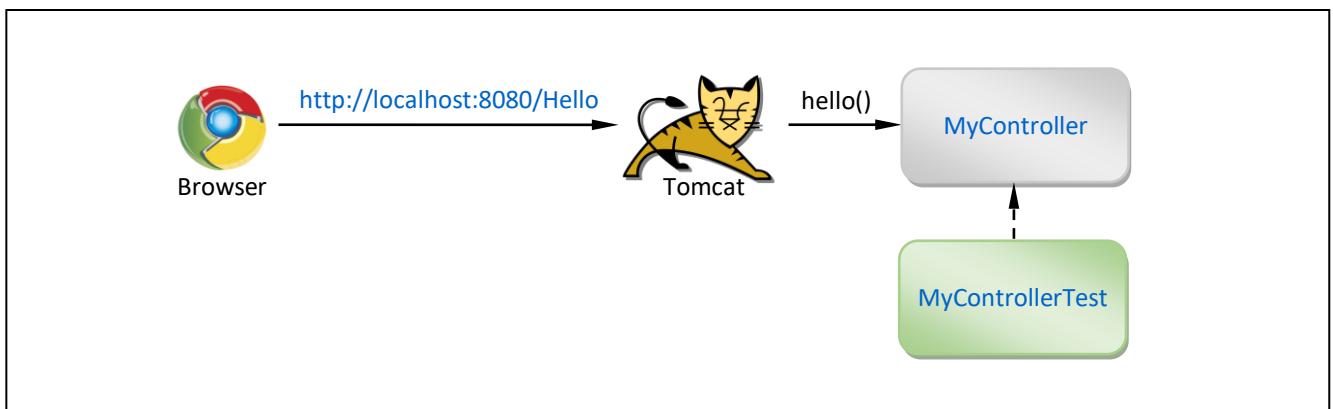
Info

[G]

- This tutorial shows how to use `@Test` Annotation to indicate that Method is Junit Test Method.
 - This is needed to distinguish Test Methods from
 - Helper Methods (Helper Methods are used/called by Test Methods)
 - Application Methods (if you are adding Test Methods to business Classes instead of having separate Test Classes)
- That way when you select a Class to run Test Methods Java would know which Methods it needs to run.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- [Create Project:](#) `springboot_junit` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit.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";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_test.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    @Autowired MyController myController;

    @Test
    void hello() {

        //PERFORM ACTION
        String result = myController.hello();

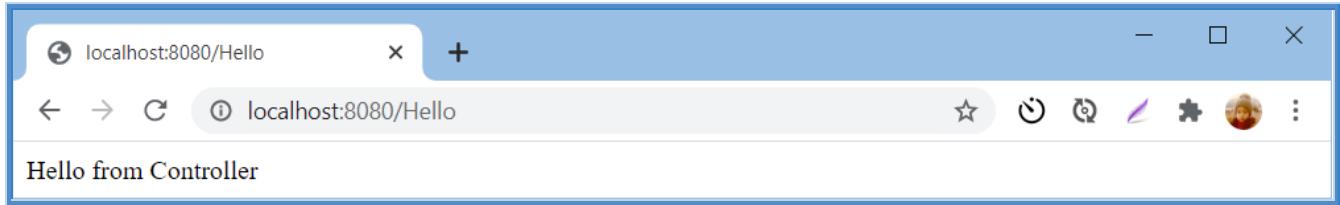
        //TEST RESULT
        assertEquals("Hello from Controller", result);

    }

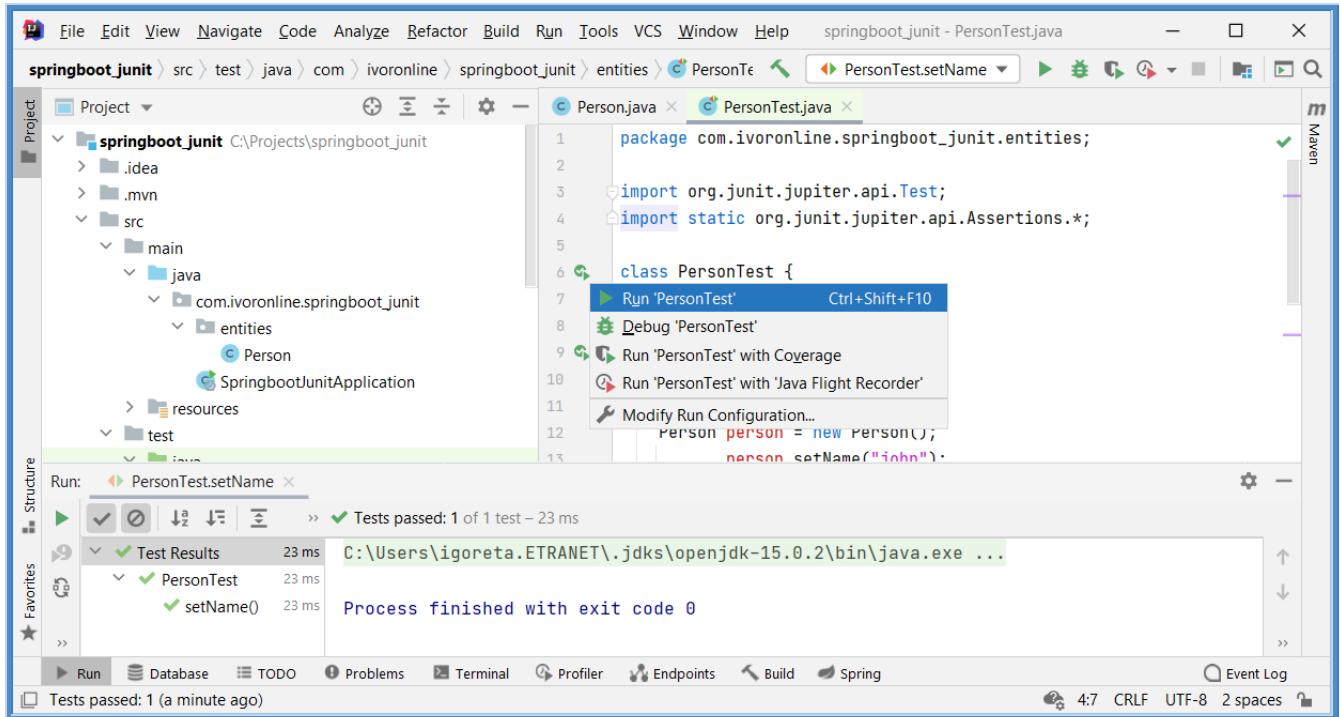
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

4.1.7 @Disable

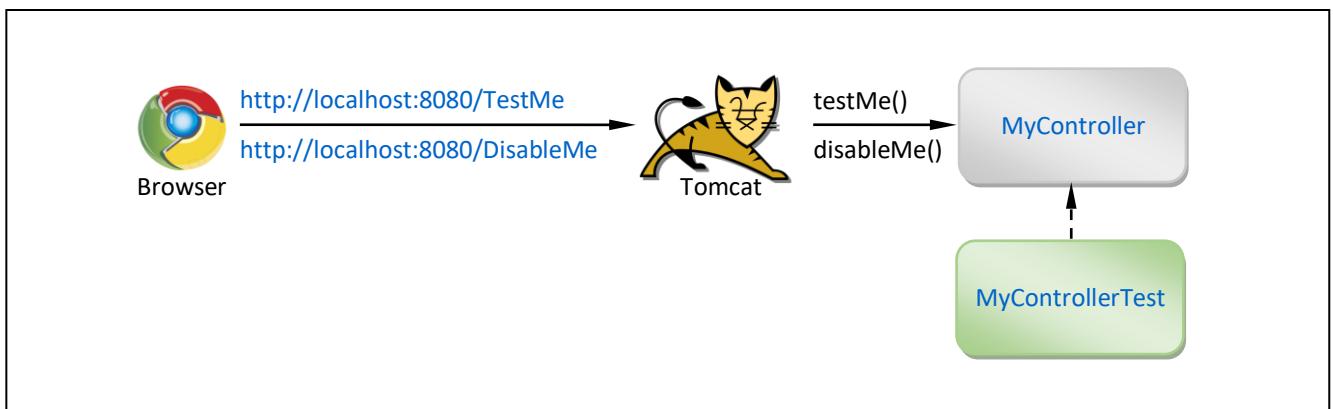
Info

[G]

- This tutorial shows how to use `@Disable` Annotation to skip specific Test Methods which is useful when you
 - want to concentrate on other Test Methods (and don't want to wait for disabled Test Methods to execute)
 - know that Test Method would fail because it is testing functionality that is still under development or being debugged
- In this example we will create two Test Methods to test each Controller's Endpoint.
But one Test Method will be `@Disable` and therefore will not get executed.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_junit_test_controller` (add Spring Boot Starters from the table)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside controllers package)
- **Create Test Class:** `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit_ignore.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("/TestMe")
    public String testMe() {
        return "Hello from TestMe";
    }

    @ResponseBody
    @RequestMapping("/DisableMe")
    public String disableMe() {
        return "Hello from DisableMe";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_ignore.controllers;

import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

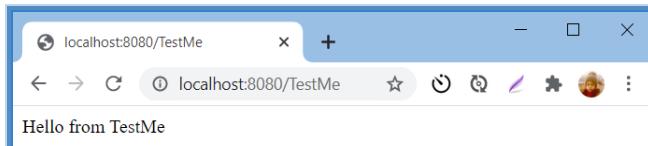
    @Autowired MyController myController;

    @Test
    void testMe() {
        String result = myController.testMe();
        assertEquals("Hello from TestMe", result);
    }

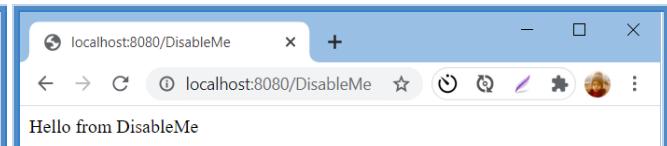
    @Disabled
    void disableMe() {
        String result = myController.disableMe();
        assertEquals("Hello from DisableMe", result);
    }
}
```

Result

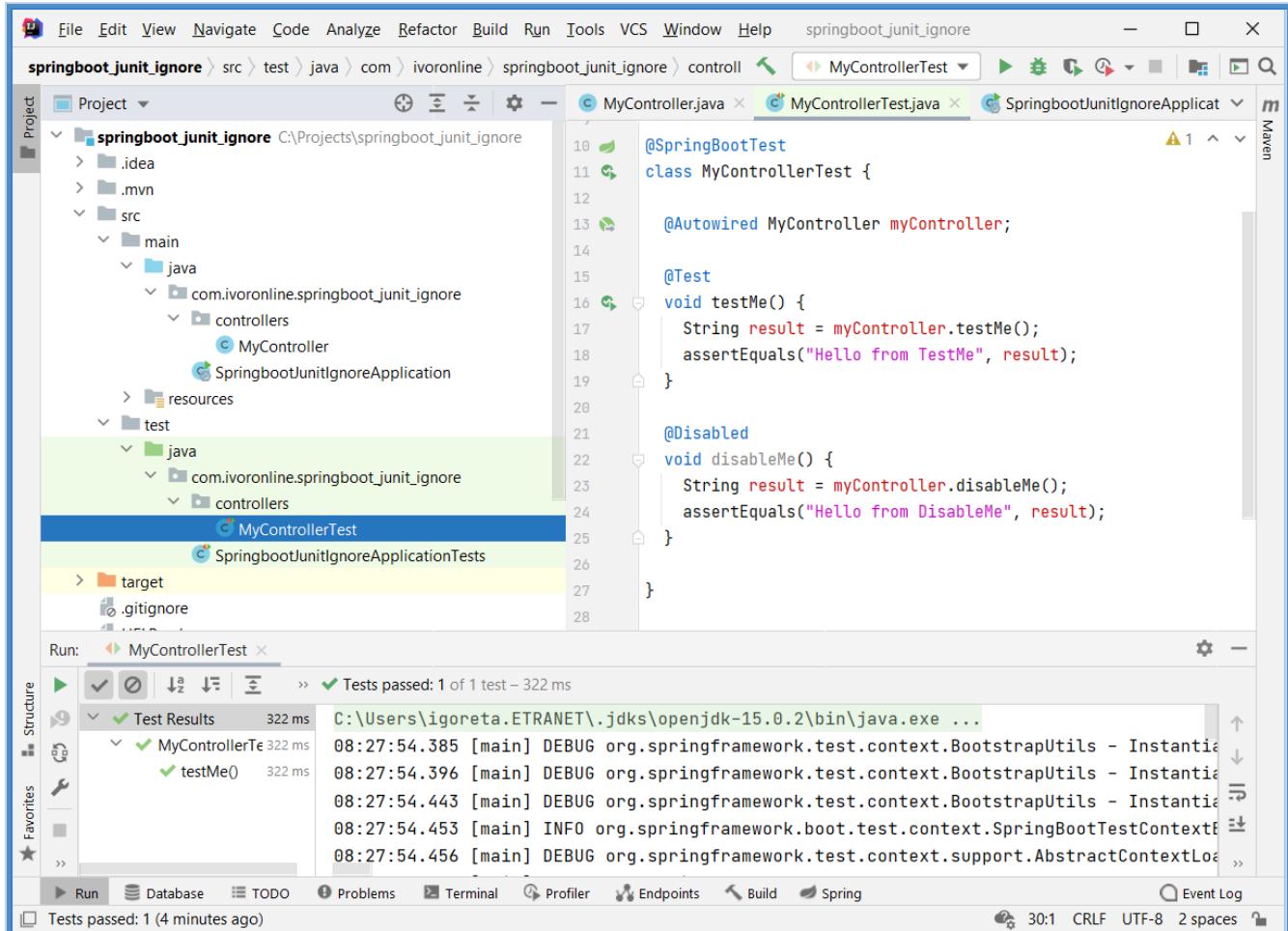
<http://localhost:8080/TestMe>



<http://localhost:8080/DisableMe>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.1.8 @BeforeAll, @AfterAll, @BeforeEach, @AfterEach

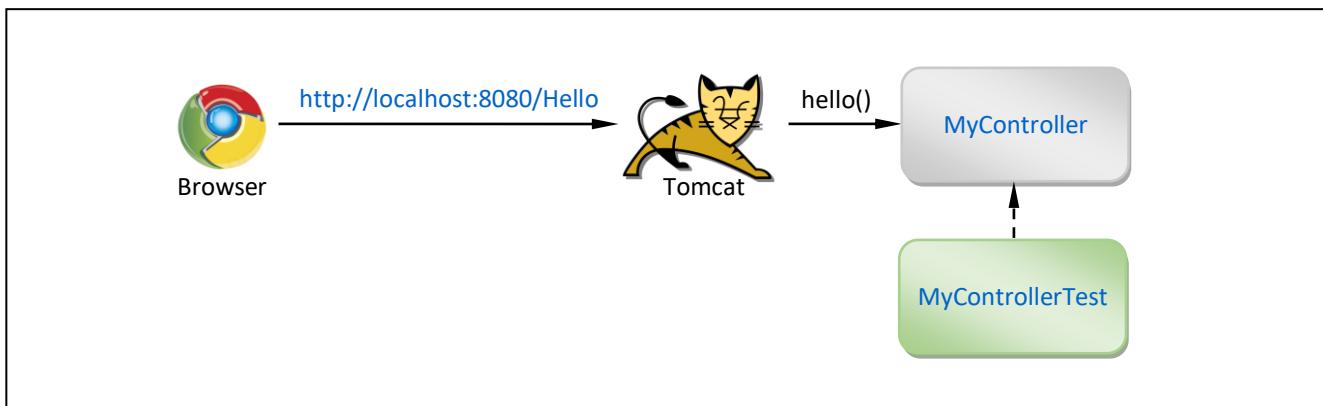
Info

[G]

- This tutorial shows how to use `@Disable` Annotation to skip specific Test Methods which is useful when you
 - want to concentrate on other Test Methods (and don't want to wait for disabled Test Methods to execute)
 - know that Test Method would fail because it is testing functionality that is still under development or being debugged
- In this example we will create two Test Methods to test each Controller's Endpoint.
But one Test Method will be `@Disable` and therefore will not get executed.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_junit_test_controller` (add Spring Boot Starters from the table)
- Create Package:** controllers (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)
- Create Test Class:** `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit_beforeafter.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";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_beforeafter.controllers;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    static int i;
    @Autowired MyController myController;

    @BeforeAll
    public static void callBeforeAllTests() {
        i = 100;
        System.out.println("callBeforeAllTests() " + i);
    }

    @BeforeEach
    public void callBeforeEachTest() {
        i = i + 20;
        System.out.println("callBeforeEachTest() " + i);
    }

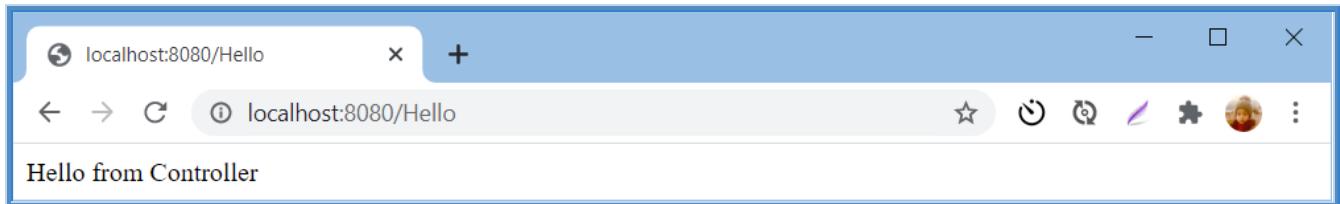
    @Test
    void hello() {
        i = i + 3;
        System.out.println("Test Method: hello() " + i);
        String result = myController.hello();
        assertEquals("Hello from Controller", result);
    }

    @AfterEach
    public void callAfterEachTest() {
        i = i - 20;
        System.out.println("callAfterEachTest() " + i);
    }

    @AfterAll
    public static void callAfterAllTests() {
        i = i - 100;
        System.out.println("\ncallAfterAllTests() " + i);
    }
}
```

Result

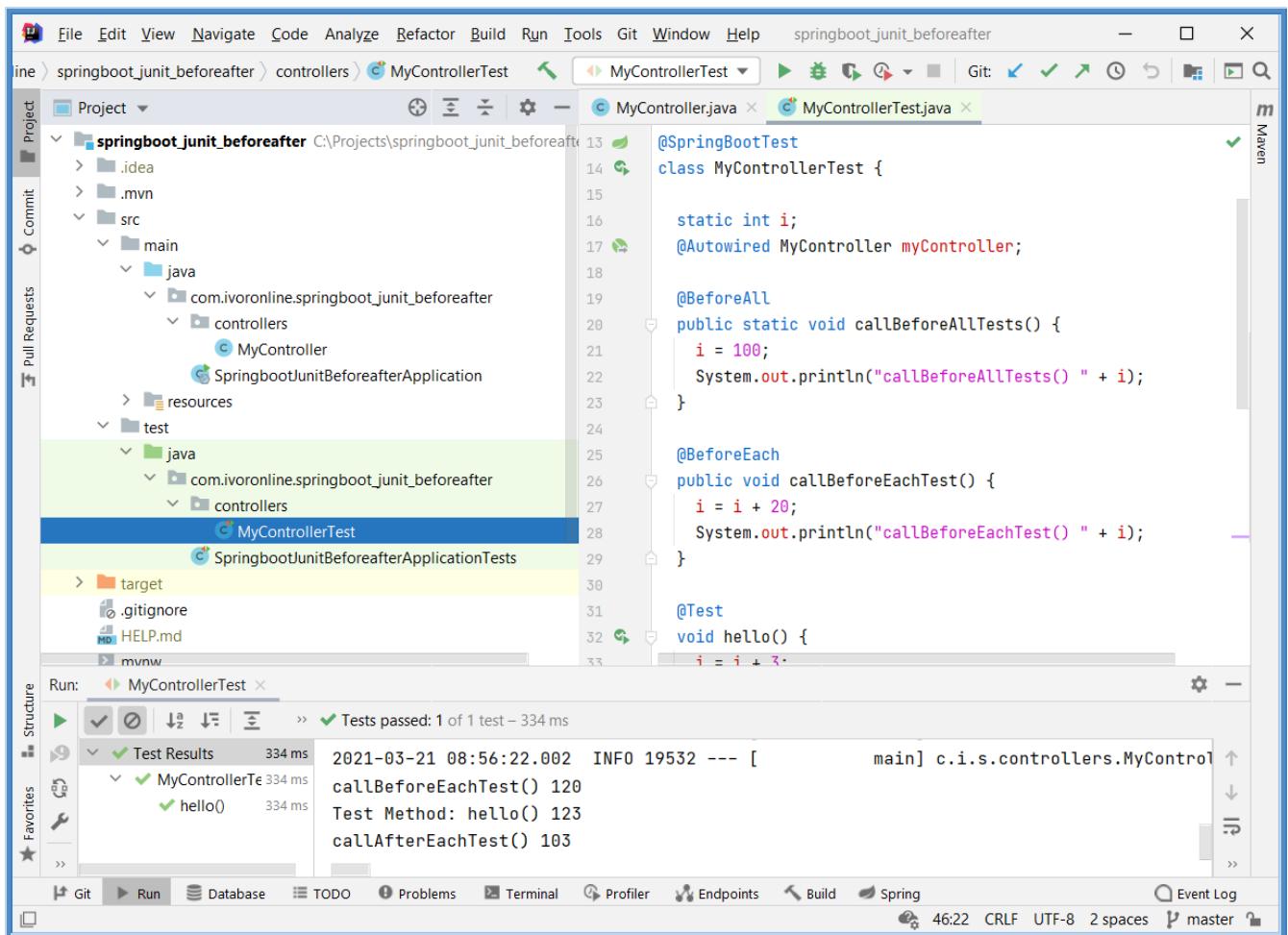
<http://localhost:8080>Hello>



Console

```
callBeforeAllTests() 100  
  
callBeforeEachTest() 120  
Test Method: hello() 123  
callAfterEachTest() 103  
  
callAfterAllTests() 3
```

Run Test Class: MyControllerTest.java



pom.xml

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

4.1.9 assert()

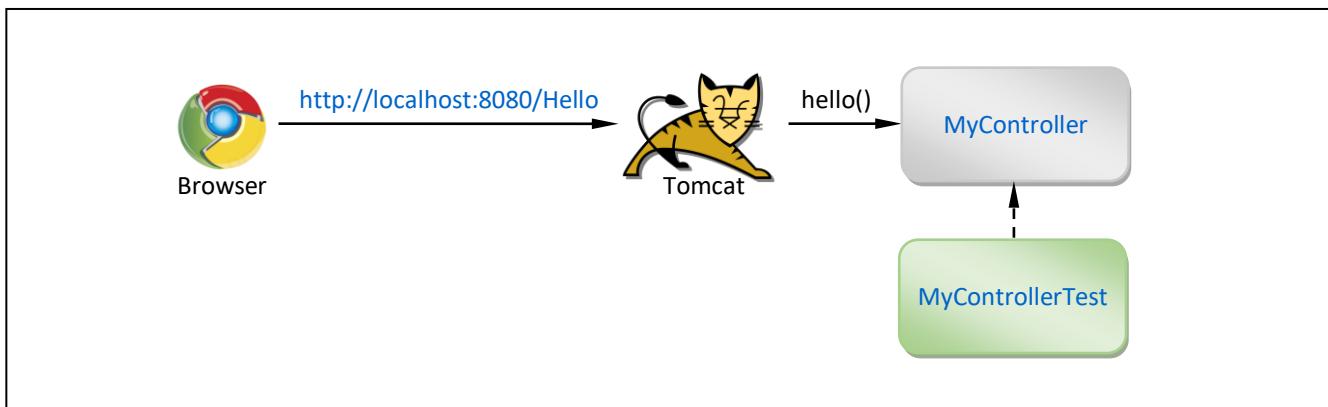
Info

[G]

- This tutorial shows how to use different `assert()` Methods which are used to assert/evaluate/compare provided values.
- If values are don't match then Test ends in **Failure** at the **first** assert that didn't pass.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project:** `springboot_junit` (add Spring Boot Starters from the table)
- Create Package:** `controllers` (inside main package)
 - Create Class:** `MyController.java` (inside controllers package)
- Create Test Class:** `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit.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";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_assert.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    @Autowired MyController myController;

    //=====
    // HELLO
    //=====

    @Test
    void hello() {

        //PERFORM ACTION
        String result = myController.hello();

        //TEST RESULT
        assertEquals("Hello from Controller", result);

    }

    //=====
    // ASSERTS PASS
    //=====

    @Test
    void assertsPass() {
        System.out.println("Inside testPass()");
        assertEquals(5, 5);
        assertNotEquals(5, 4);
        assertTrue(true);
        assertFalse(false);
        assertNull(null);
        assertNotNull(1);
    }

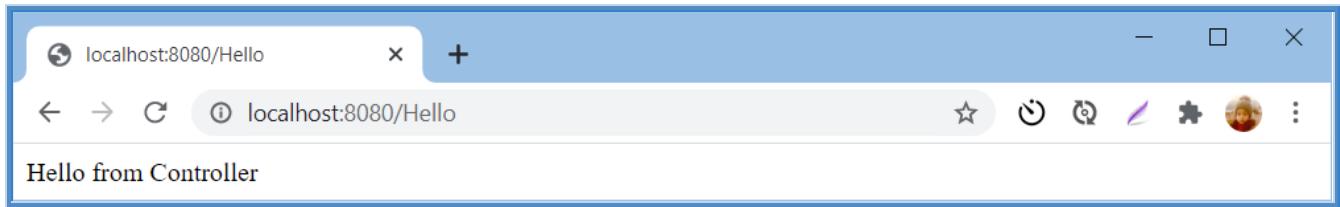
    //=====
    // ASSERTS FAIL
    //=====

    @Test
    public void assertsFail() {
        System.out.println("Inside testFail()");
        Integer i = 1;
        assertEquals(5, 4); //java.lang.AssertionError: expected:<5> but was:<4>
        assertNotEquals(5, 5); //java.lang.AssertionError: Values should be different. Actual: 5
        assertTrue(false); //java.lang.AssertionError:
        assertFalse(true); //java.lang.AssertionError:
        assertNull(i); //java.lang.AssertionError: expected null, but was:<1>
    }

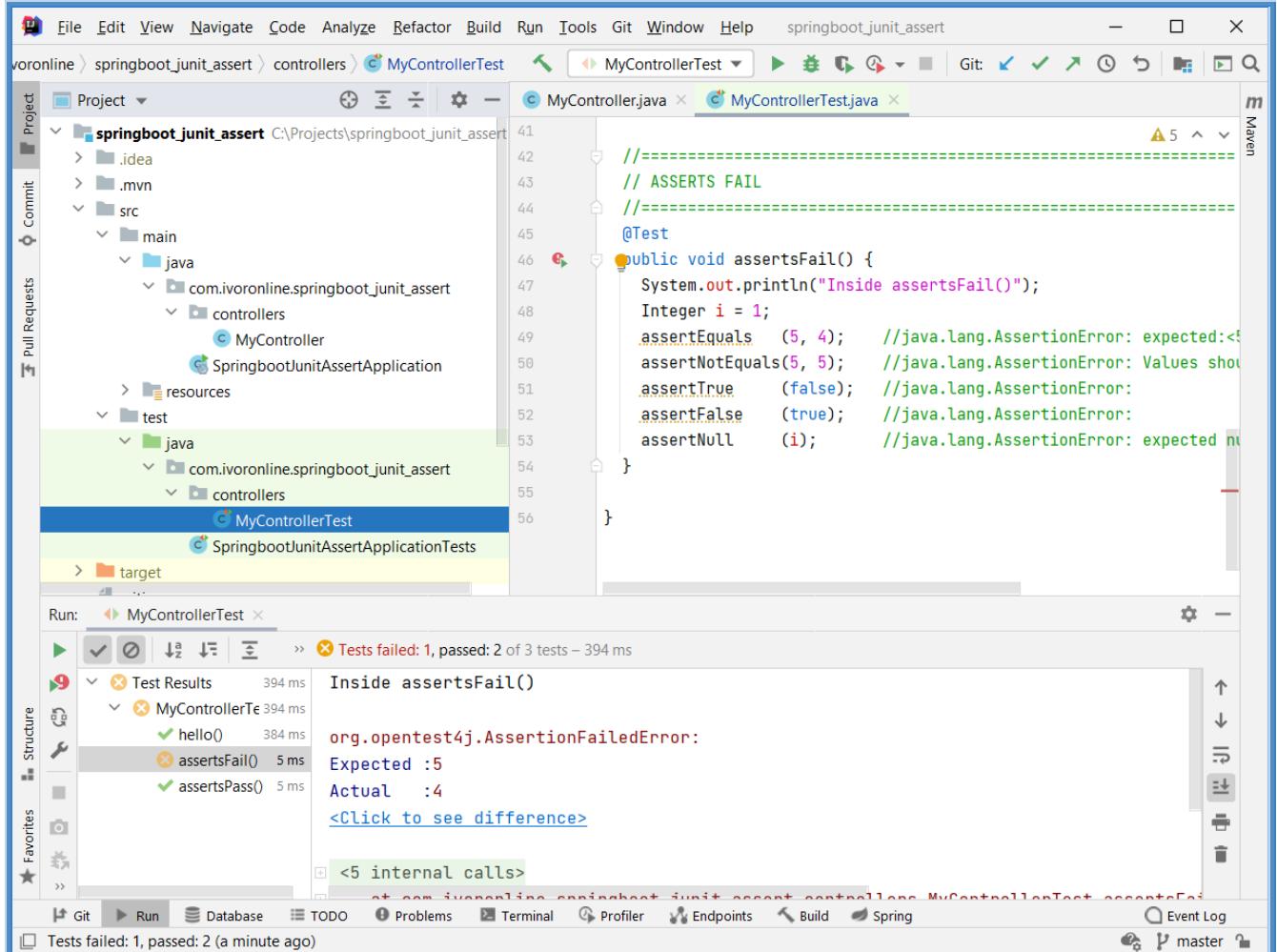
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

4.1.10 assertThrows()

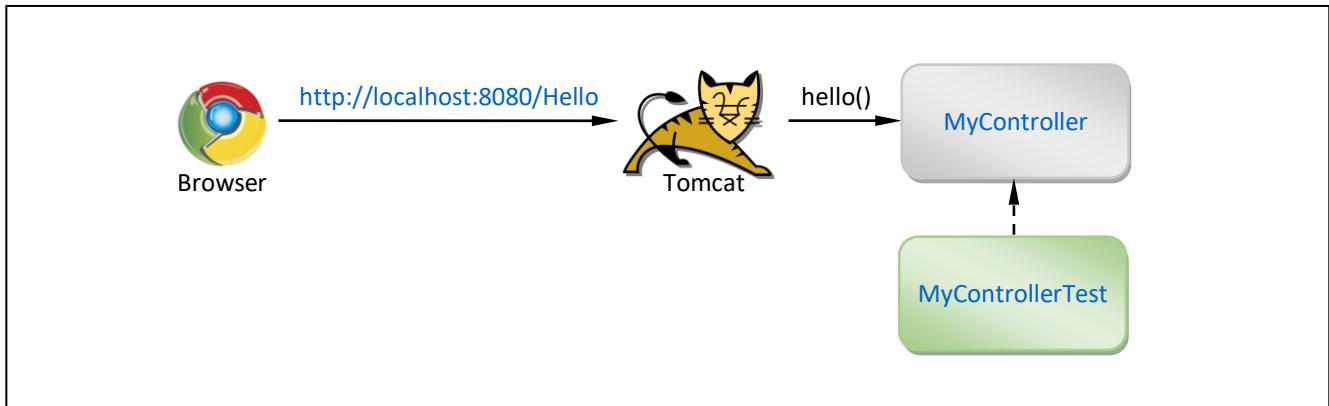
Info

[G] [R]

- This tutorial shows how to use different `assert()` Methods which are used to assert/evaluate/compare provided values.
- If values are don't match then Test ends in **Failure** at the **first** assert that didn't pass.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- [Create Project:](#) `springboot_junit` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit_exception.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() throws Exception {
        if (1 == 1) { throw new IllegalArgumentException("Argument is missing"); }
        return "Hello from Controller";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_exception.controllers;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    @Autowired MyController myController;

    //=====
    // assertThrows() - Lambda
    //=====
    @Test
    void hello1() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> myController.hello() ); //Check Type
        assertEquals("Argument is missing", exception.getMessage()); //Check Message
    }

    //=====
    // assertThrows() - Executable
    //=====
    @Test
    void hello2() {

        //CHECK TYPE
        Exception exception = assertThrows(IllegalArgumentException.class, new Executable() {
            @Override
            public void execute() throws Throwable {

```

```

        myController.hello();
    }
});

//CHECK MESSAGE
assertEquals("Argument is missing", exception.getMessage());

}

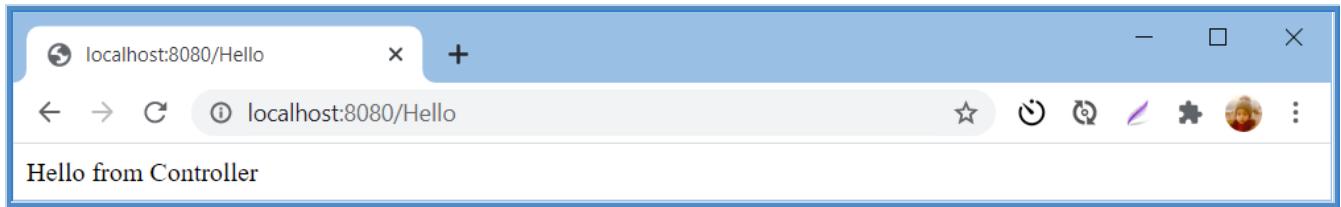
//=====================================================================
// TRY - CATCH
//=====================================================================

@Test
void hello3() {
    try {
        myController.hello();
        fail();                                     //Fail if no Exception was thrown
    }
    catch (IllegalArgumentException exception) {
        assertEquals("Argument is missing", exception.getMessage()); //Catch Exceptions of expected Type
        //Fail if Message is wrong
    }
    catch(Exception exception) {
        fail();                                     //Fail if Exception is of wrong Type
    }
}
}

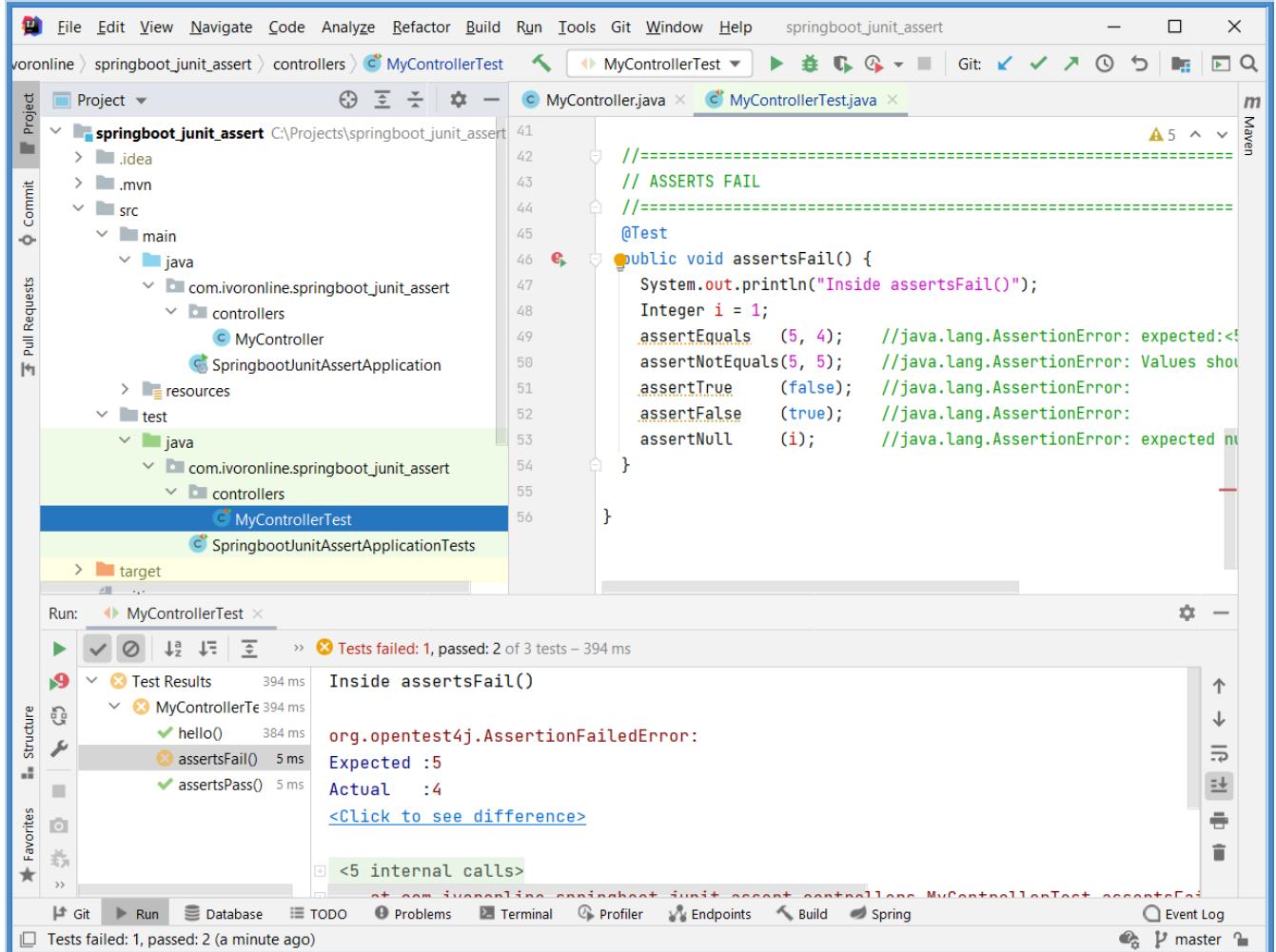
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2 Mockito

Info

- Following tutorials show how to use Mockito. (inside JUnit Tests)
- Mockito is used to **Mock Dependencies** (Classes and Methods that are called by the Method being tested)
 - first you **Mock Class** (you need to Mock Class in order to be able to Mock its Methods)
 - then you **Mock Methods** (of Mocked Class)
- To **Mock Method** means to **specify return value for specific Input Parameters.**

To Mock Method first you need to Mock Class to which this Method belongs to.

Class can be Mocked using either **@Mock** or **@Spy**.

In both cases Mocked Method returns what is defined by Mockito and not what the actual implementation would return.

Mock Object

```
@Mock PersonRepository personRepositoryMock;  
@Spy PersonRepository personRepositoryMock;
```

Inject Mocked Objects

(in place of **@Autowired**)

```
@InjectMocks MyController myController;
```

Mock Method

```
when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));  
given(personRepositoryMock.getPersonById(1)).willReturn(new Person(1, "Susan", 50)); //Alias for when()  
doReturn(new Person(1, "Susan", 50)).when(personRepositoryMock).getPersonById(1); //Alternative for when()
```

@Mock vs @Spy

[R]

- The difference is that with
 - @Mock** Real Method is **never called**
 - @Spy** Real Method is **always called** & executed but return value is intercepted & Mocked Value is returned instead
- We could say that
 - @Mock** creates **fully** Mocked Object (Object only has Mocked Methods defined by Mockito)
 - @Spy** creates **partially** Mocked Object (Object has combination of Real and Mocked Methods)
- The other subtle difference is that with **@Spy** Real Methods are **always called** - calls to Real Methods are not blocked.
 - If Method was Mocked it will return what was defined by Mockito after it gets executed.
 - If Method was not Mocked it will return what was defined by its actual implementation.

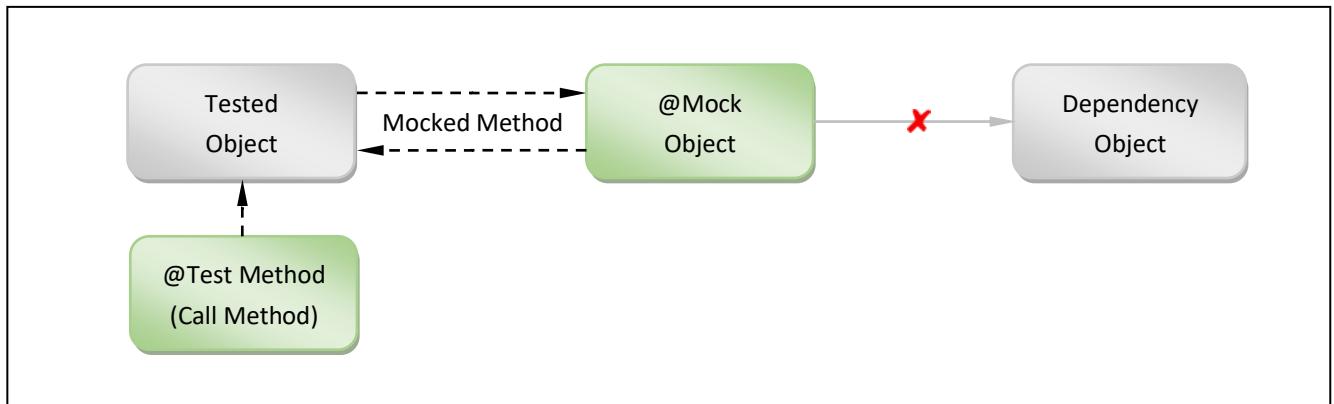
Syntax

```
//MOCK DEPENDENCY CLASS (choose @Mock or @Spy)  
@Mock PersonRepository personRepositoryMock;  
@Spy PersonRepository personRepositorySpy;  
  
//INJECT MOCKS (into Class being tested where @autowired is used)  
@InjectMocks MyController myController;  
  
//MOCK METHOD OF DEPENDENCY CLASS (Methods are mocked in the same way for both @Mock and @Spy)  
when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));  
when(personRepositorySpy.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));
```

@Mock

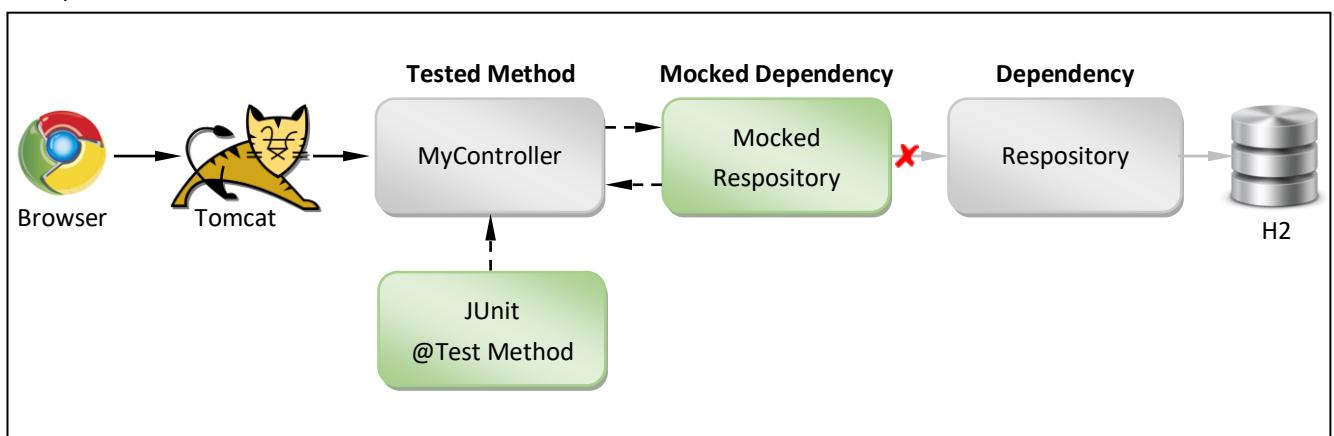
- `@Mock` creates fake Object without any Methods except those that you introduce through `when()` and `doReturn()`.
- In `@Mock` Object default behavior of the Method is to do nothing.
- `@Mock` is used for **Functional Testing** - to test functionality of a component in isolation from other components.
- This means that during tests Mocked Methods are called instead of real ones.
Calls to Mocked Methods are intercepted by Mockito which then returns whatever we specify.
This way we can test how tested Method would behave when Dependent Methods return certain values.
- Such approach is useful during parallel development of interdepended components.
Each developer can test his component by Mocking dependent components that are still under development.

`@Mock`



- For instance if we are testing Endpoint that calls Repository, then we would Mock Repository to return specific value.
- That way we can test Endpoint's functionality without having access to the actual Repository or the underlying Database.
- And we can also test how Endpoint will behave upon receiving different values from the Repository without having to adjust Database to actually return these values.

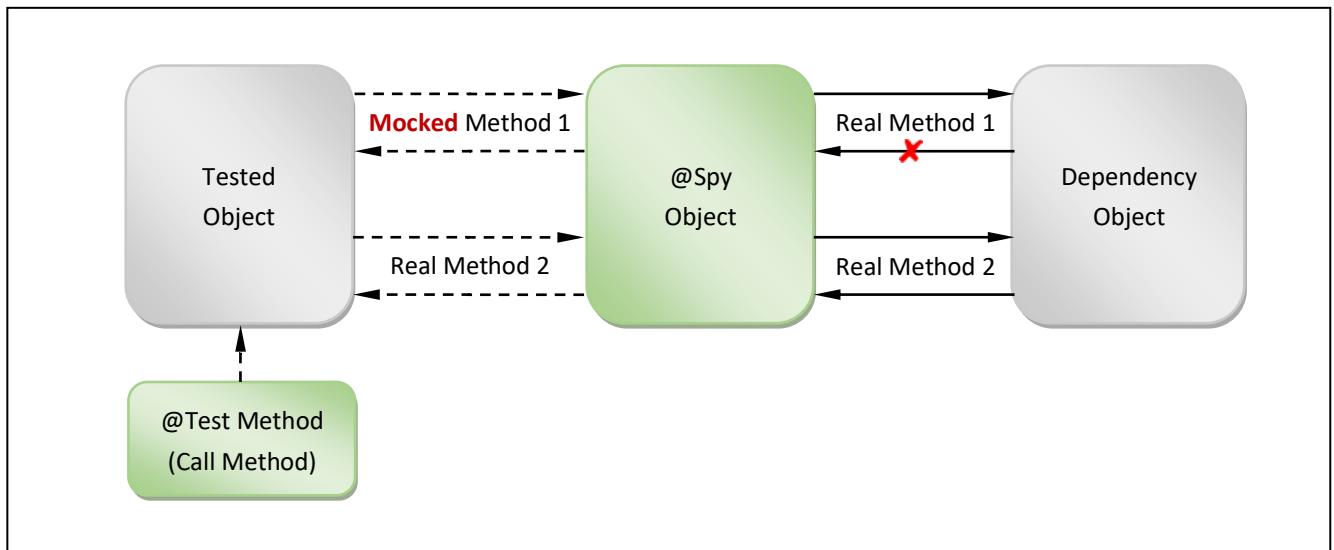
Example



@Spy

- `@Spy` creates Wrapper Object around real/existing Object that has its own working Methods.
Then you use `when()` or `doReturn()` to change return value just for some of those Methods.
- In `@Spy` Object default behavior of the Method is the one implemented by the Application.
- `@Spy` is used for **Integration Testing** - to test how your component interacts/integrates with other components.
So with `@Spy` you can test one component at the time by Mocking calls to those other components not being tested.

`@Spy`



4.2.1 Mock - Object - @Mock

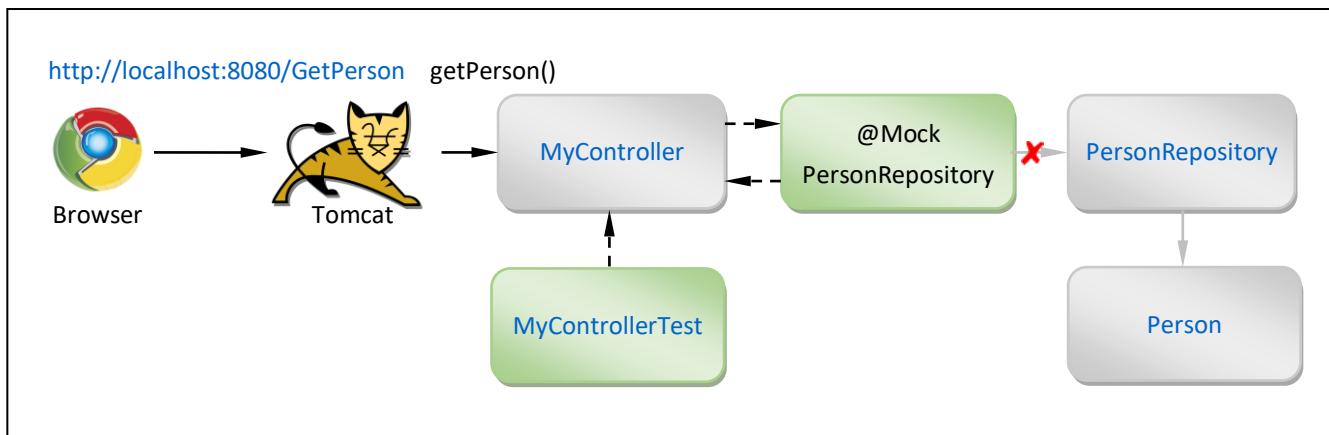
Info

[G] [R]

- This tutorial shows how to use
 - `@Mock` to **Mock Class**
 - `@InjectMocks` to **Inject Instance of Mocked Class**
 - `when()` to **Mock Method of Mocked Class**
- `@InjectMocks` looks for `@Autowired PersonRepository personRepository` & inserts instance of Mocked Class in Controller
When Controller calls `personRepository.getPersonById(1)` then Mocked Method of Mocked Class is called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@SpringBootTest
class MyControllerTest {
    @Mock      PersonRepository personRepositoryMock;
    @InjectMocks MyController     myController;
```

Procedure

- Create Project: springboot_mockito_mock (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
 - Create Class: Person.java (inside entities package)
- Create Package: repositories (inside main package)
 - Create Class: PersonRepository.java (inside entities package)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)
- Create Package: controllers (inside test package src\test\java\com.ivoronline.springboot_junit)
 - Create Test Class: MyControllerTest.java (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_mock.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_mock.repositories;

import com.ivoronline.springboot_mockito_mock.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivoronline.springboot_mockito_mock.controllers;

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

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_mock.controllers;

import com.ivoronline.springboot_mockito_mock.entities.Person;
import com.ivoronline.springboot_mockito_mock.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD (getPersonById(1) returns Susan instead John for id=1)
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER ENDPOINT
        String result = myController.getPerson(1);

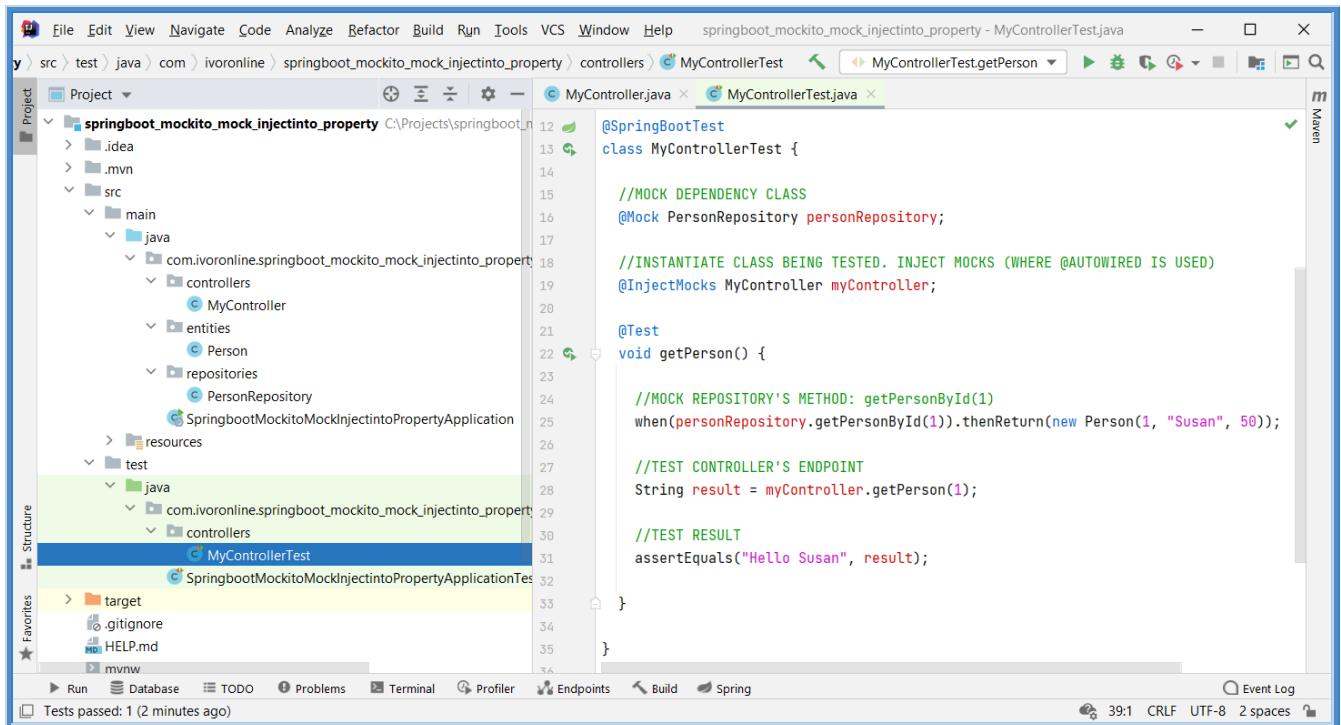
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.2 Mock - Object - @MockBean

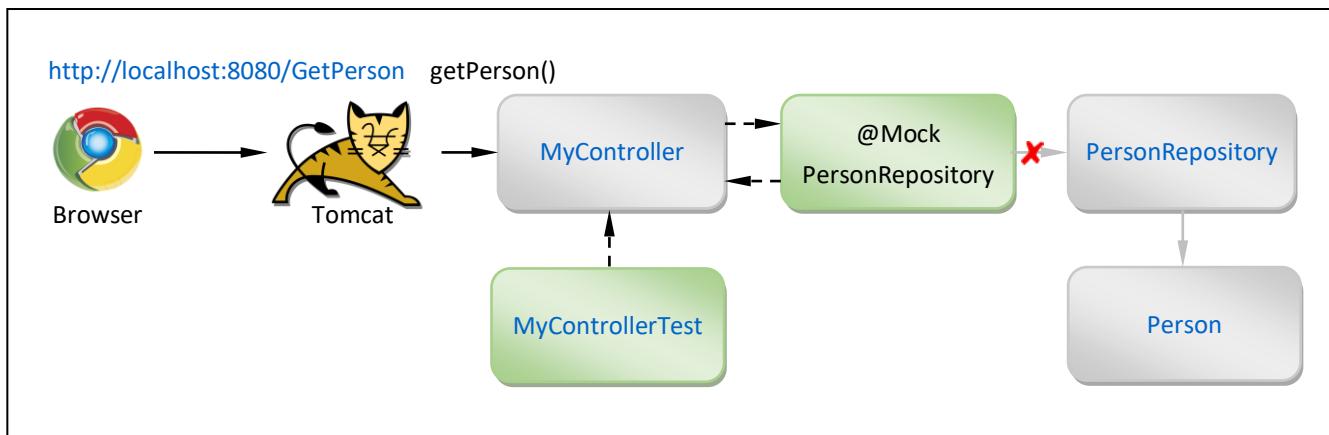
Info

[G] [R]

- This tutorial shows how to use
 - `@MockBean` to **Mock Bean**
 - `@Autowired` to **Inject Instance of Mocked Bean**
 - `when()` to **Mock Method of Mocked Class**
- `@InjectMocks` looks for `@Autowired PersonRepository personRepository` & inserts instance of Mocked Class in Controller
When Controller calls `personRepository.getPersonById(1)` then Mocked Method of Mocked Class is called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@SpringBootTest
class MyControllerTest {
    @MockBean PersonRepository personRepositoryMock;
    @Autowired MyController myController;
```

Procedure

- [Create Project:](#) `springboot_mockito_mockbean` (add Spring Boot Starters from the table)
- [Create Package:](#) `entities` (inside main package)
 - [Create Class:](#) `Person.java` (inside entities package)
- [Create Package:](#) `repositories` (inside main package)
 - [Create Class:](#) `PersonRepository.java` (inside entities package)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Package:](#) `controllers` (inside test package)
 - [Create Test Class:](#) `MyControllerTest.java` (inside controllers package)

Person.java

```
package com.ivoronline.springboot_mockito_mockbean.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_mockbean.repositories;

import com.ivoronline.springboot_mockito_mockbean.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_mockbean.controllers;

import com.ivorononline.springboot_mockito_mockbean.entities.Person;
import com.ivorononline.springboot_mockito_mockbean.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_mockbean.controllers;

import com.ivoronline.springboot_mockito_mockbean.entities.Person;
import com.ivoronline.springboot_mockito_mockbean.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import org.springframework.boot.test.mock.mockito.MockBean;

import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY BEAN (gets injected through @Autowired)
    @MockBean PersonRepository personRepositoryMock;

    //INJECT INSTANCE OF MyController
    @Autowired MyController myController;

    @Test
    void getPerson() {

        //MOCK METHOD: getPersonById(1)
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

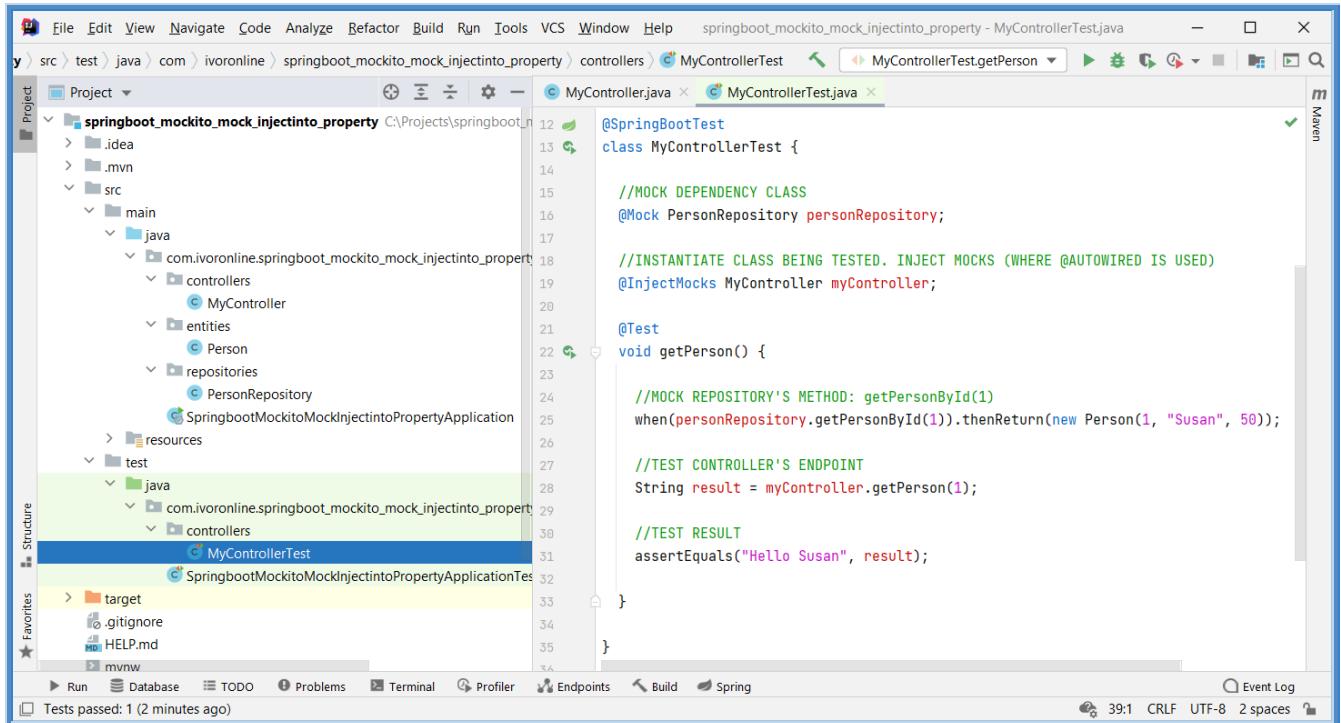
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.3 Mock - Object - @Spy

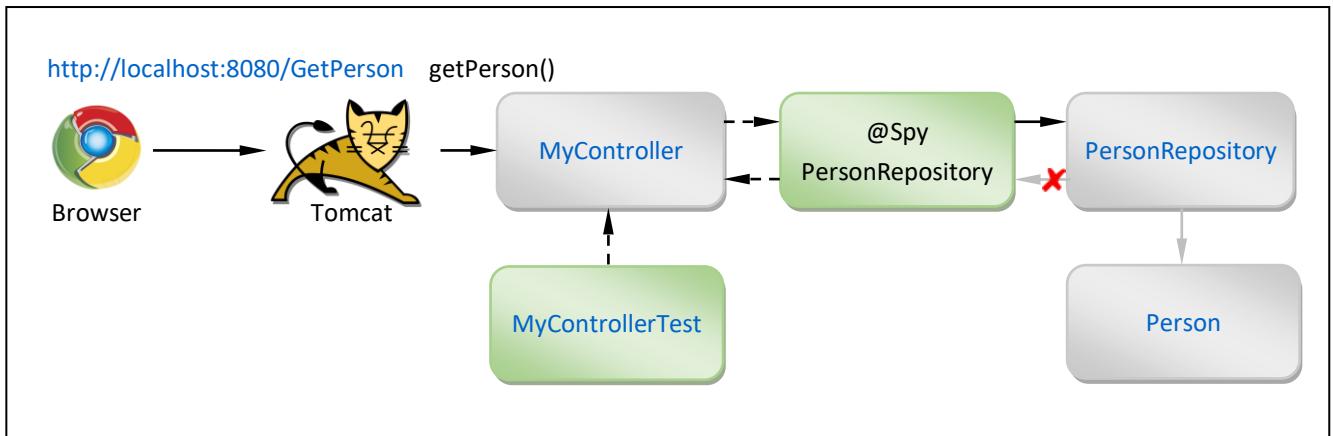
Info

[G]

- This tutorial shows how to use
 - **@Spy** to **Create Wrapper Object** `@Spy PersonRepository personRepositorySpy`
 - **@InjectMocks** to **Inject Wrapper Object** `@InjectMocks MyController myController`
 - **when()** to **Specify Method Return Value** `when(personRepositorySpy.getPersonById(1)).thenReturn(...);`
- **@Spy** creates **Wrapper Object** around **Existing Object** (Instance of Dependency Class) and intercepts calls & return values
 - every **Method call is forwarded** to a real Method of that **Existing Object**
 - but **return value is blocked** - instead return value defined by Mockito is returned

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@SpringBootTest
class MyControllerTest {
    @Spy      PersonRepository personRepositoryMock;
    @InjectMocks MyController     myController;
```

Procedure

- Create Project: `springboot_mockito_spy` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package)
 - Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package)
 - Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package `src\test\java\com.ivoronline.springboot_junit`)
 - Create Test Class: `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_spy.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_spy.repositories;

import com.ivoronline.springboot_mockito_spy.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID (this is method is called)
    public Person getPersonById(Integer id) {
        System.out.println("getPersonById()");
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivoronline.springboot_mockito_spy.controllers;

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

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_spy.controllers;

import com.ivoronline.springboot_mockito_spy.entities.Person;
import com.ivoronline.springboot_mockito_spy.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Spy;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Spy PersonRepository personRepositoryMock;

    //INJECT SPIES (where @autowired is used)
    @InjectMocks MyController myController;

    //ENDPOINT
    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER ENDPOINT
        String result = myController.getPerson(1);
        System.out.println(result);

        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

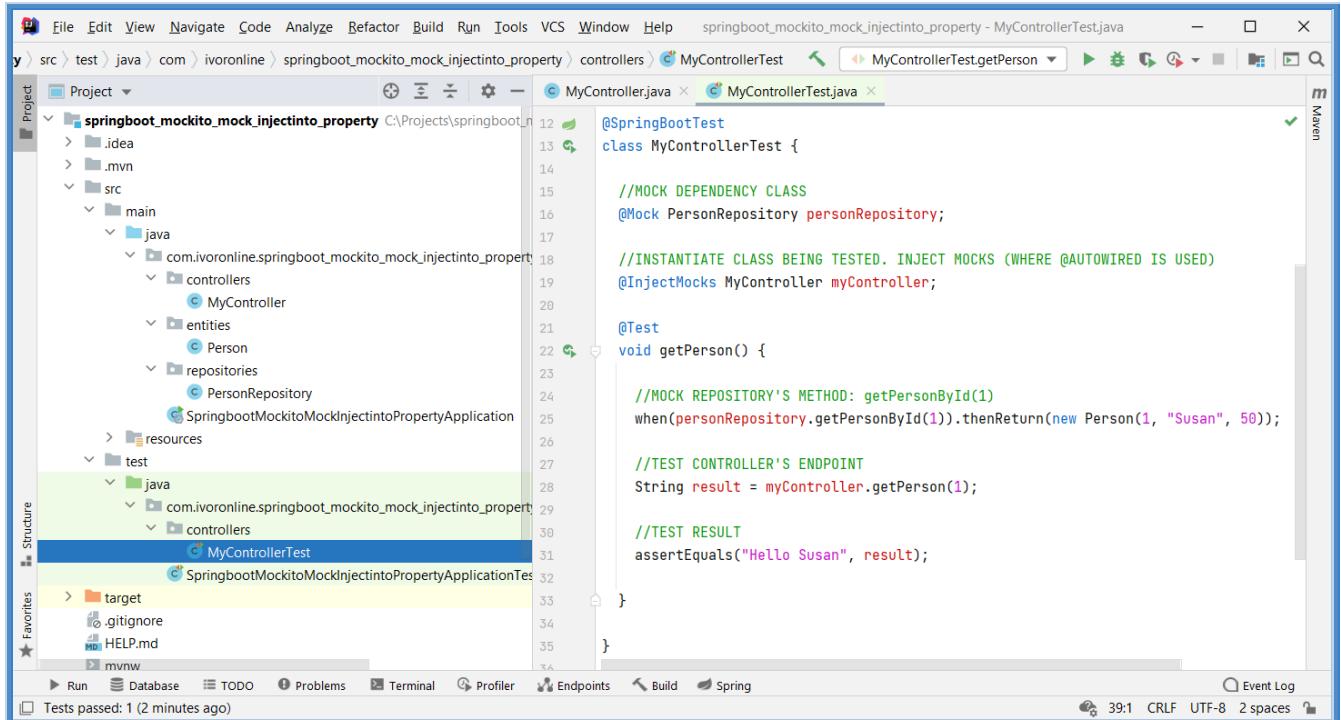
<http://localhost:8080/GetPerson?id=1>



Console

```
getPersonById()
Susan is 50 years old
```

Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.4 Mock - Object - @Spybean

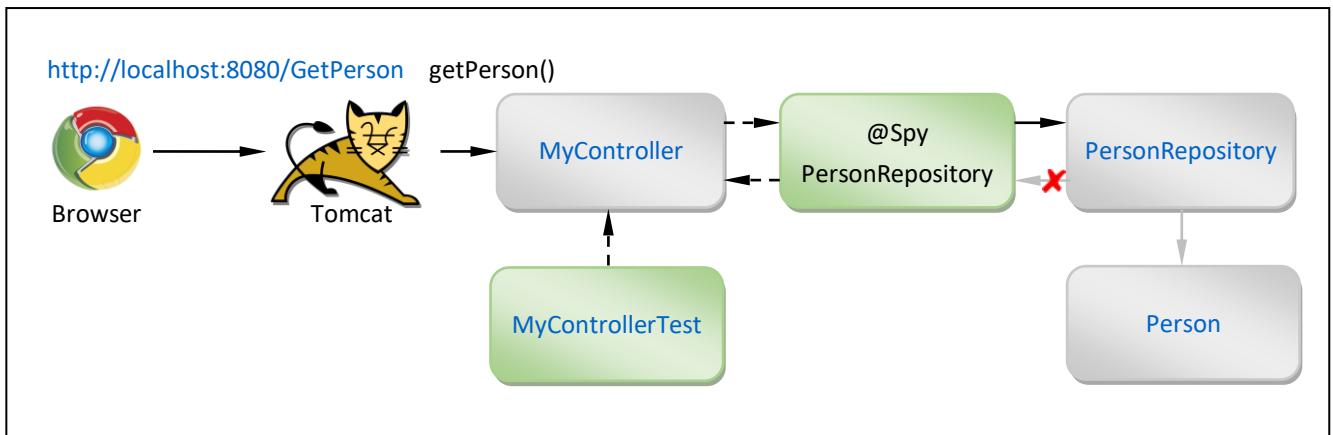
Info

[G]

- This tutorial shows how to use
 - **@SpyBean** to **Create Wrapper Object** `@SpyBean PersonRepository personRepositorySpy`
 - **@Autowired** to **Inject Wrapper Object** `@Autowired MyController myController`
 - **when()** to **Specify Method Return Value** `when(personRepositorySpy.getPersonById(1)).thenReturn(...);`
- **@SpyBean** creates **Wrapper Object** around **Existing Object** (Instance of Dependency Class) & intercepts calls & return value
 - every **Method call is forwarded** to a real Method of that **Existing Object**
 - but **return value is blocked** - instead return value defined by Mockito is returned

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@SpringBootTest
class MyControllerTest {
    @SpyBean    PersonRepository personRepositoryMock;
    @Autowired MyController     myController;
```

Procedure

- Create Project: springboot_mockito_spybean (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
 - Create Class: Person.java (inside entities package)
- Create Package: repositories (inside main package)
 - Create Class: PersonRepository.java (inside entities package)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)
- Create Package: controllers (inside test package src\test\java\com.ivoronline.springboot_junit)
 - Create Test Class: MyControllerTest.java (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_spybean.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_spybean.repositories;

import com.ivoronline.springboot_mockito_spybean.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID (this is method is called)
    public Person getPersonById(Integer id) {
        System.out.println("getPersonById()");
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_spybean.controllers;

import com.ivorononline.springboot_mockito_spybean.entities.Person;
import com.ivorononline.springboot_mockito_spybean.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivorononline.springboot_mockito_spybean.controllers;

import com.ivorononline.springboot_mockito_spybean.entities.Person;
import com.ivorononline.springboot_mockito_spybean.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import org.springframework.boot.test.mock.mockito.SpyBean;

import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @SpyBean PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @Autowired MyController myController;

    @Test
    void getPerson() {

        //MOCK METHOD: getPersonById(1)
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

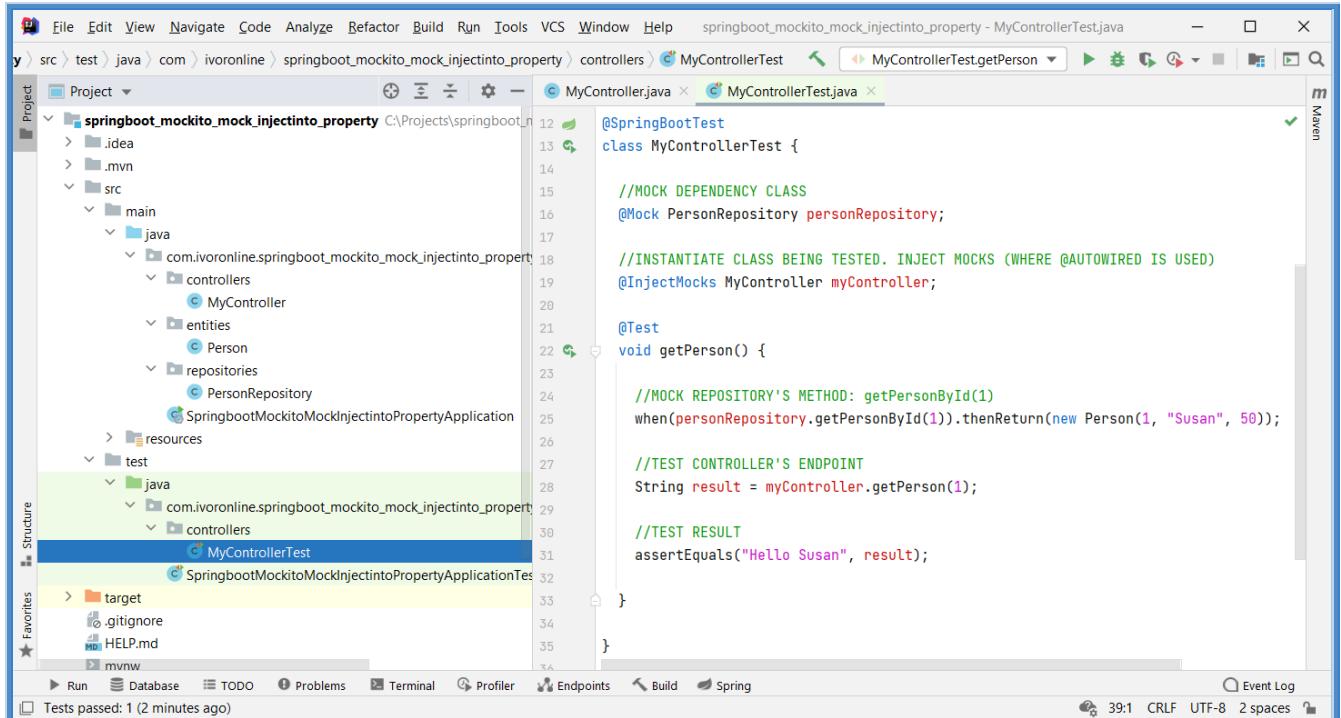
<http://localhost:8080/GetPerson?id=1>



Console

```
getPersonById()
Susan is 50 years old
```

Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.5 Mock - Method - when()

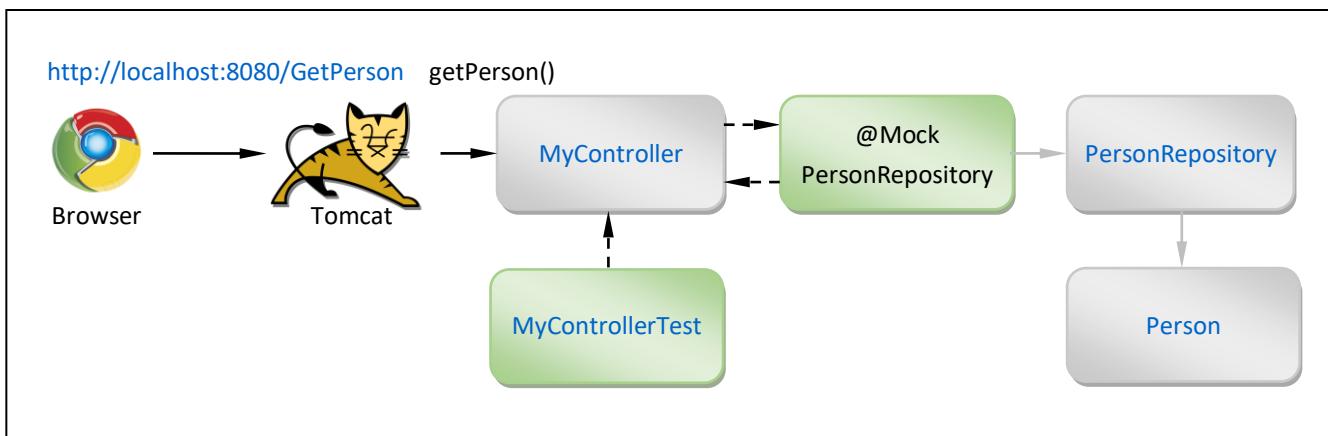
Info

[G]

- when() is used to **Mock Method** by specifying what should Method return for specific Input Parameters.
- When Tested Method calls **Mocked Method**, **Mocked Method** will return value defined by Mockito's when() and not the value from the real Method's Implementation.
- when() is used to Mock Methods for both @Mock and @Spy Objects.
- doReturn() is alternative to when() with different behavior in certain situations.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
when (personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));
given(personRepositoryMock.getPersonById(1)).willReturn(new Person(1, "Susan", 50)); //Alias for when()
```

Procedure

- Create Project: `springboot_mockito_when` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package)
 - Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package)
 - Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package)
 - Create Test Class: `MyControllerTest.java` (inside controllers package)

`Person.java`

```
package com.ivoronline.springboot_mockito_when.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

`PersonRepository.java`

```
package com.ivoronline.springboot_mockito_when.controllers;

import com.ivoronline.springboot_mockito_when.entities.Person;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_when.controllers;

import com.ivorononline.springboot_mockito_when.entities.Person;
import com.ivorononline.springboot_mockito_when.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_when.controllers;

import com.ivoronline.springboot_mockito_when.entities.Person;
import com.ivoronline.springboot_mockito_when.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks
    MyController myController;

    @Test
    void getPerson() {

        //MOCK METHOD: getPersonById(1)
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

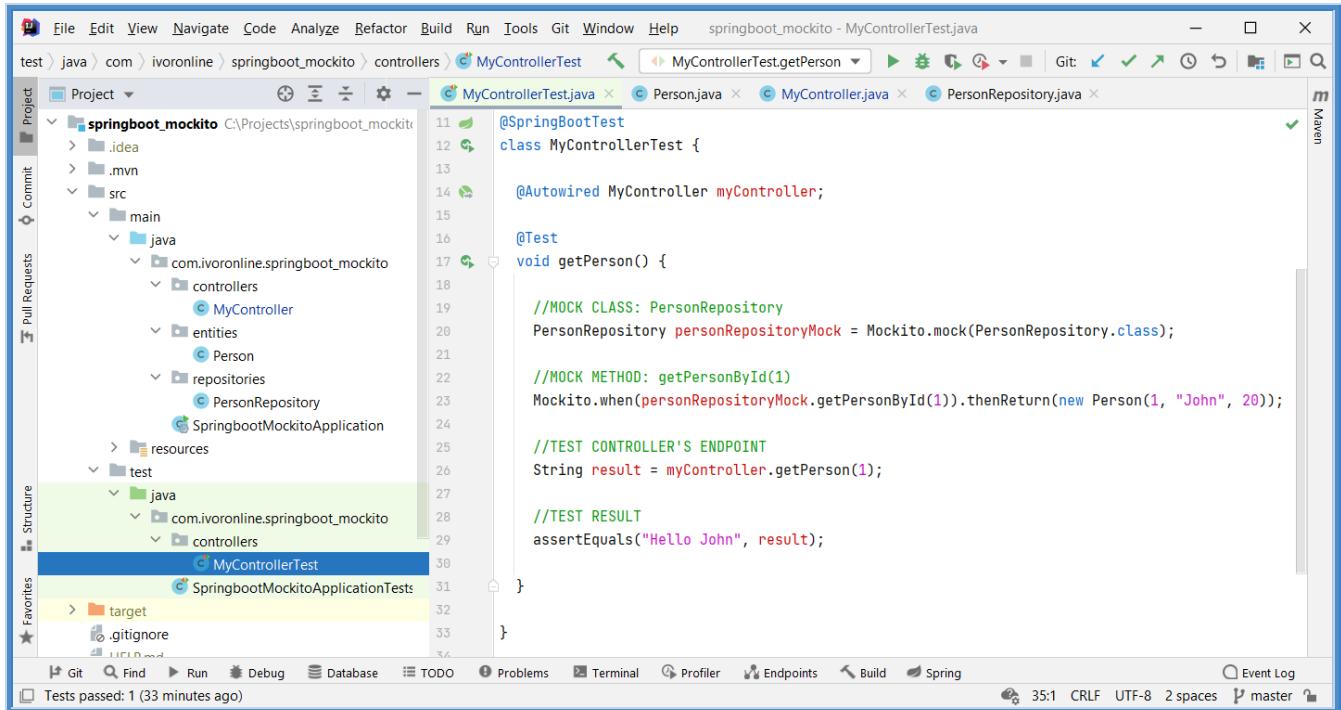
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.6 Mock - Method - given()

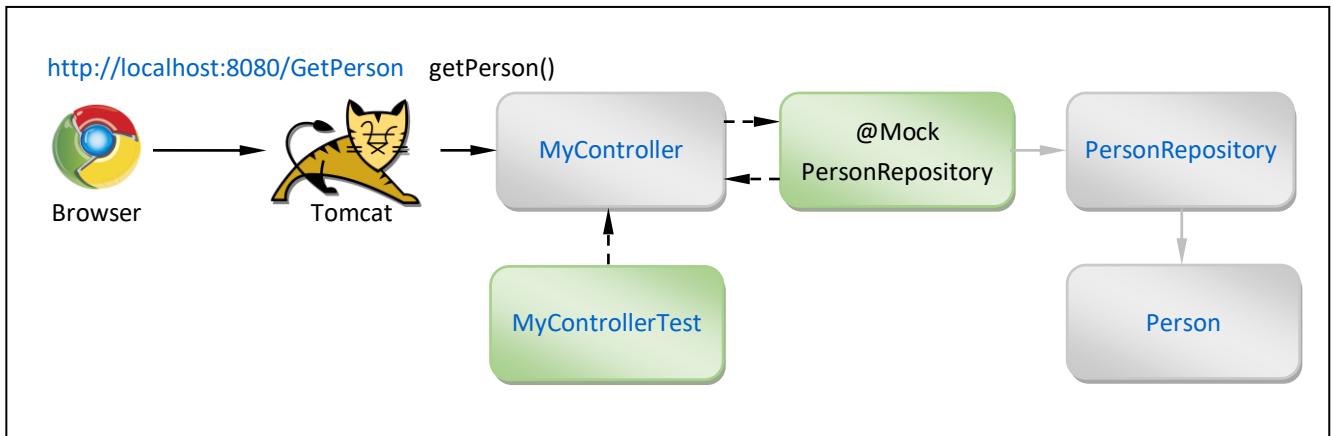
Info

[G]

- given() is alias for when() meaning that these two Methods do exactly the same thing but use different wording.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
when (personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));  
given(personRepositoryMock.getPersonById(1)).willReturn(new Person(1, "Susan", 50)); //Alias for when()
```

Procedure

- [Create Project:](#) `springboot_mockito_when` (add Spring Boot Starters from the table)
- [Create Package:](#) `entities` (inside main package)
 - [Create Class:](#) `Person.java` (inside entities package)
- [Create Package:](#) `repositories` (inside main package)
 - [Create Class:](#) `PersonRepository.java` (inside entities package)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Package:](#) `controllers` (inside test package)
 - [Create Test Class:](#) `MyControllerTest.java` (inside controllers package)

Person.java

```
package com.ivoronline.springboot_mockito_given.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_given.controllers;

import com.ivoronline.springboot_mockito_given.entities.Person;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_given.controllers;

import com.ivorononline.springboot_mockito_given.entities.Person;
import com.ivorononline.springboot_mockito_given.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_given.controllers;

import com.ivoronline.springboot_mockito_given.entities.Person;
import com.ivoronline.springboot_mockito_given.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;

import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.when;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    @Test
    void getPerson() {

        //MOCK METHOD: getPersonById(1)
        given(personRepositoryMock.getPersonById(1)).willReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

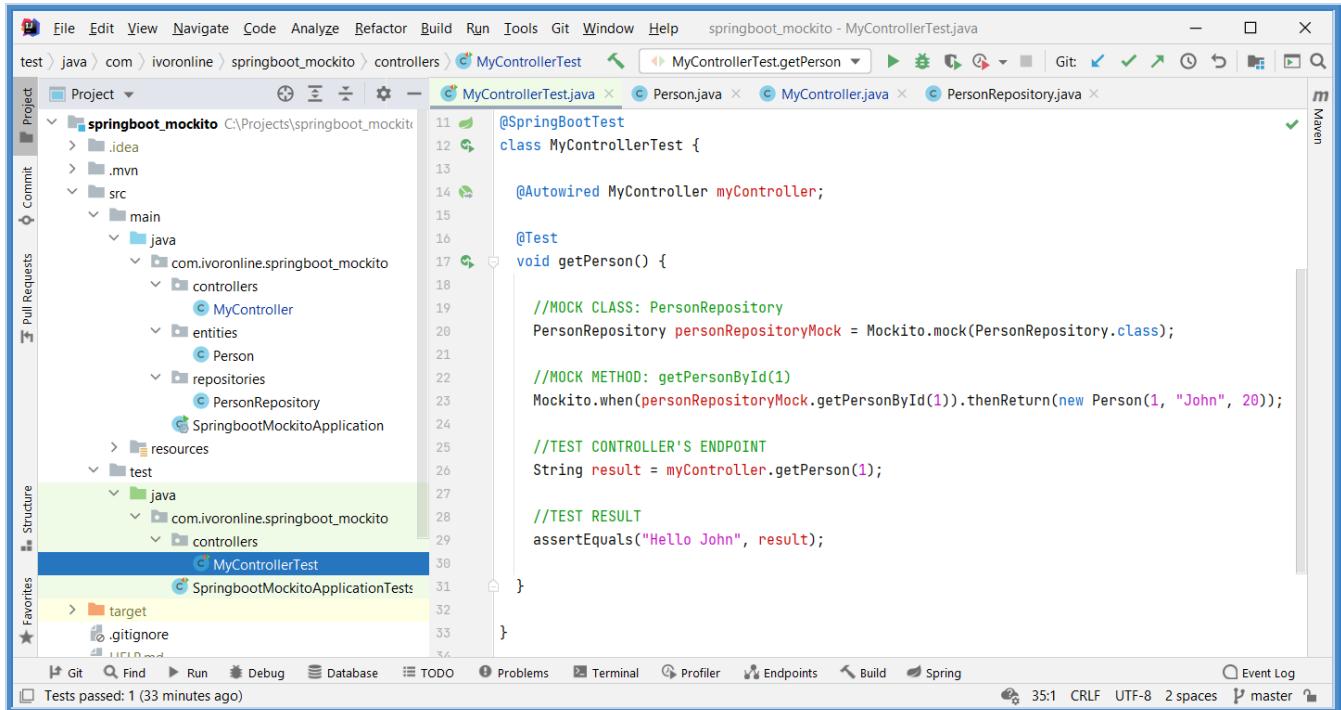
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.7 Mock - Method - doReturn()

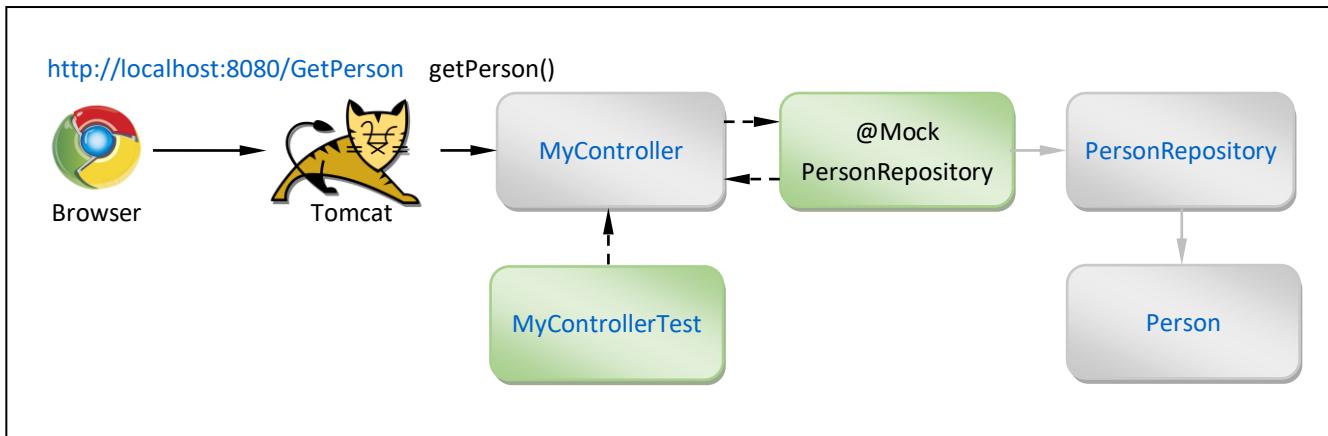
Info

[G]

- This tutorial shows how to use
 - `@Mock` to **Mock Class** `@Mock PersonRepository personRepositoryMock`
 - `@InjectMocks` to **Inject Instance of Mocked Class** `@InjectMocks MyController myController`
 - `doReturn()` to **Mock Method of Mocked Class** `doReturn(...).when(personRepositoryMock).getPersonById(1);`
- `@InjectMocks` looks for `@Autowired PersonRepository personRepository` & inserts instance of Mocked Class in Controller When Controller calls `personRepository.getPersonById(1)` then Mocked Method of Mocked Class is called.
- `when()` is alternative to `doReturn()` with different behavior in certain situations.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
doReturn(new Person(1, "Susan", 50)).when(personRepositoryMock).getPersonById(1);
```

Procedure

- Create Project: springboot_mockito_doreturn (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
 - Create Class: Person.java (inside entities package)
- Create Package: repositories (inside main package)
 - Create Class: PersonRepository.java (inside entities package)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)
- Create Package: controllers (inside test package src\test\java\com.ivoronline.springboot_junit)
 - Create Test Class: MyControllerTest.java (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_doreturn.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_doreturn.repositories;

import com.ivoronline.springboot_mockito_doreturn.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_doreturn.controllers;

import com.ivorononline.springboot_mockito_doreturn.entities.Person;
import com.ivorononline.springboot_mockito_doreturn.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivorononline.springboot_mockito_doreturn.controllers;

import com.ivorononline.springboot_mockito_doreturn.entities.Person;
import com.ivorononline.springboot_mockito_doreturn.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import static org.mockito.Mockito.doReturn;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

    //MOCK CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD (getPersonById(1) returns Susan instead John for id=1)
        doReturn(new Person(1, "Susan", 50)).when(personRepositoryMock).getPersonById(1);

        //TEST CONTROLLER ENDPOINT
        String result = myController.getPerson(1);

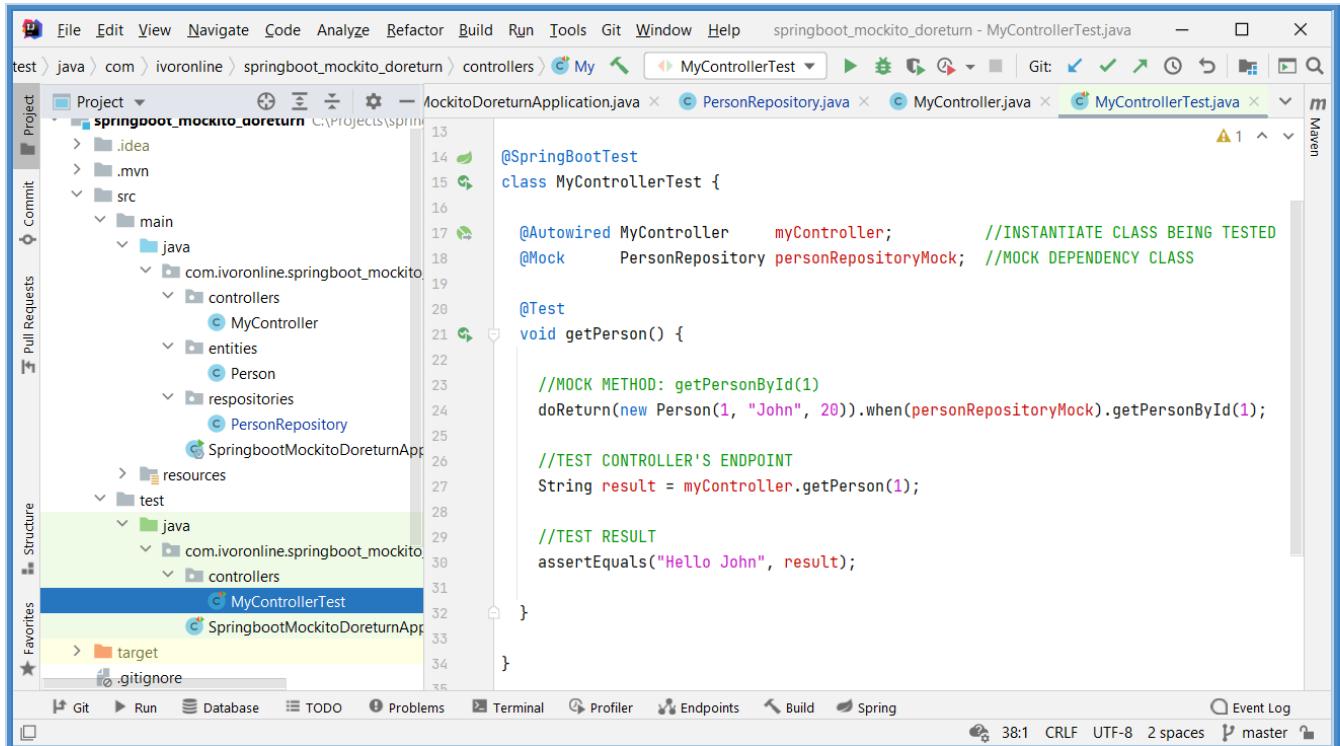
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.8 @InjectMocks

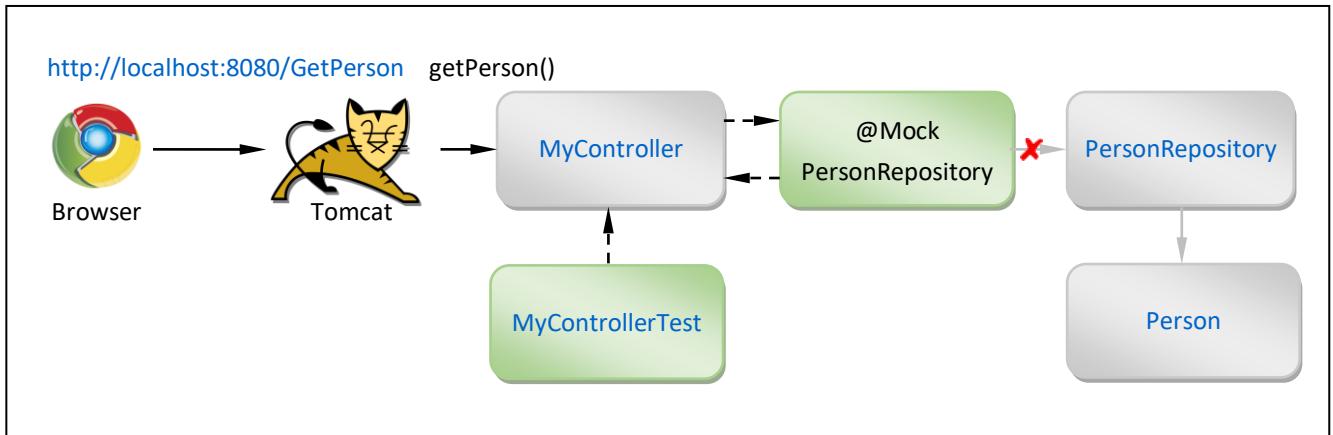
Info

[G]

- **@InjectMocks** is used to **Inject** **@Mock** and **@Spy Objects** into Class being tested.
- **@InjectMocks** looks for **@Autowired Annotation** inside the Class being tested and Injects **@Mock** and **@Spy Instances**.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@SpringBootTest
class MyControllerTest {
    @Mock PersonRepository personRepositoryMock;
    @InjectMocks MyController myController;
```

Procedure

- Create Project: springboot_mockito (add Spring Boot Starters from the table)
- Create Package: entities (inside main package)
 - Create Class: Person.java (inside entities package)
- Create Package: repositories (inside main package)
 - Create Class: PersonRepository.java (inside entities package)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)
- Create Package: controllers (inside test package src\test\java\com.ivoronline.springboot_junit)
 - Create Test Class: MyControllerTest.java (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito.repositories;

import com.ivoronline.springboot_mockito.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito.controllers;

import com.ivorononline.springboot_mockito.entities.Person;
import com.ivorononline.springboot_mockito.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito.controllers;

import com.ivoronline.springboot_mockito.entities.Person;
import com.ivoronline.springboot_mockito.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD (getPersonById(1) returns Susan instead John for id=1)
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER ENDPOINT
        String result = myController.getPerson(1);

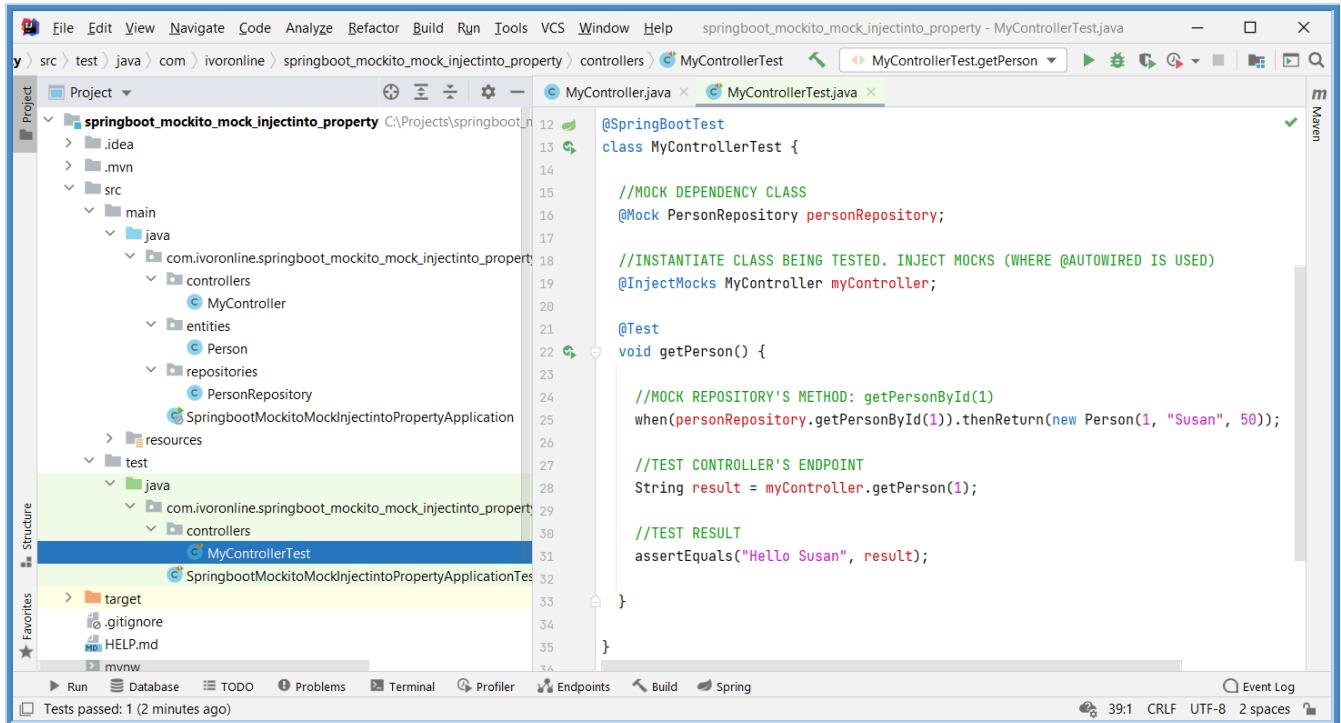
        //TEST RESULT
        assertEquals("Susan is 50 years old", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.9 verify()

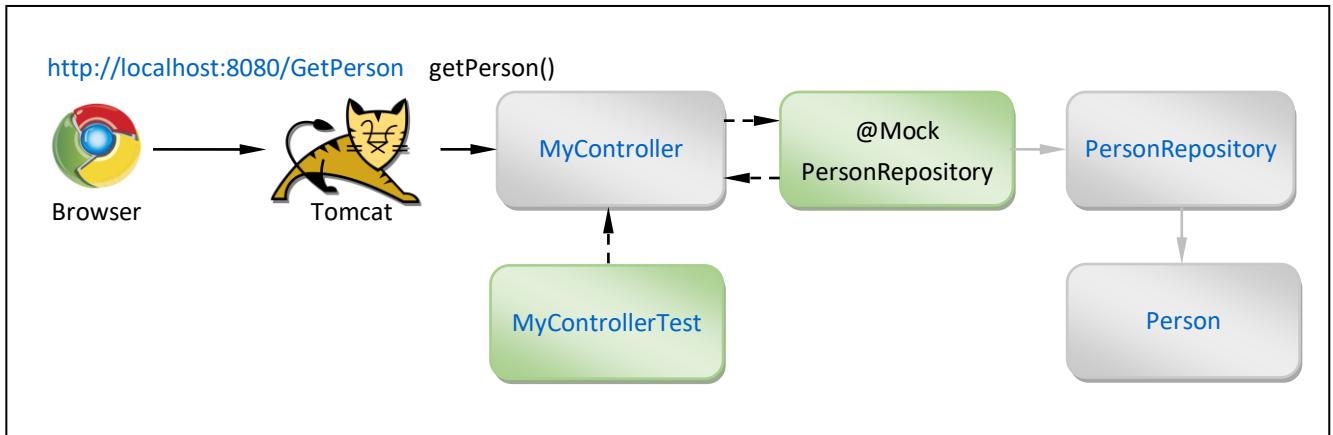
Info

[G]

- This tutorial shows how to use **verify()** to check how many times Mocked Method was called with specific Parameter.
- verify()** only counts number of calls that happened before **verify()** was called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
verify(personRepositoryMock, times(2)).getPersonById(1);
```

Second Parameter

PARAMETER	EXAMPLE	DESCRIPTION
No Parameter	verify(personRepositoryMock).getPersonById(1)	Exactly 1 time
times(2)	verify(personRepositoryMock, times(2)).getPersonById(1)	Exactly 2 times
atLeast(2)	verify(personRepositoryMock, atLeast(2)).getPersonById(1)	Twice or more
atLeastOnce()	verify(personRepositoryMock, atLeastOnce()).getPersonById(1)	Once or more
never()	verify(personRepositoryMock, never()).getPersonById(1)	Never

Procedure

- Create Project: `springboot_mockito_verify` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package)
 - Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package)
 - Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package)
 - Create Test Class: `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_verify.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_verify.repositories;

import com.ivoronline.springboot_mockito_verify.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(3, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_verify.controllers;

import com.ivorononline.springboot_mockito_verify.entities.Person;
import com.ivorononline.springboot_mockito_verify.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_verify.controllers;

import com.ivoronline.springboot_mockito_verify.entities.Person;
import com.ivoronline.springboot_mockito_verify.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    //ENDPOINT
    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //CALL REPOSITORY METHOD 2 TIMES
        myController.getPerson();
        myController.getPerson();

        //VERIFY THAT METHOD getPersonById() WAS CALLED EXACTLY 2 TIMES WITH PARAMETER 1
        verify(personRepositoryMock, times(2)).getPersonById(1);

    }

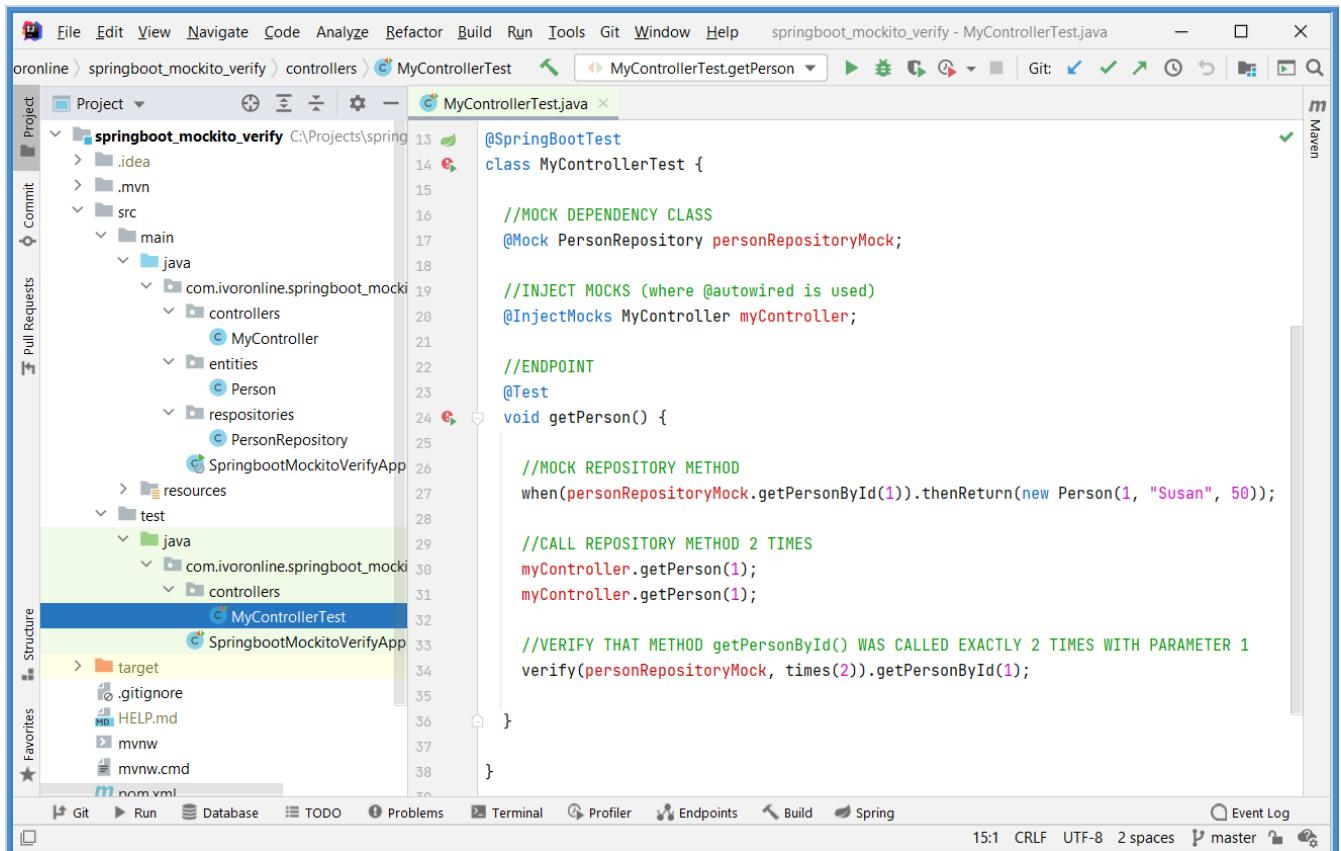
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.2.10 verify() - ArgumentMatchers

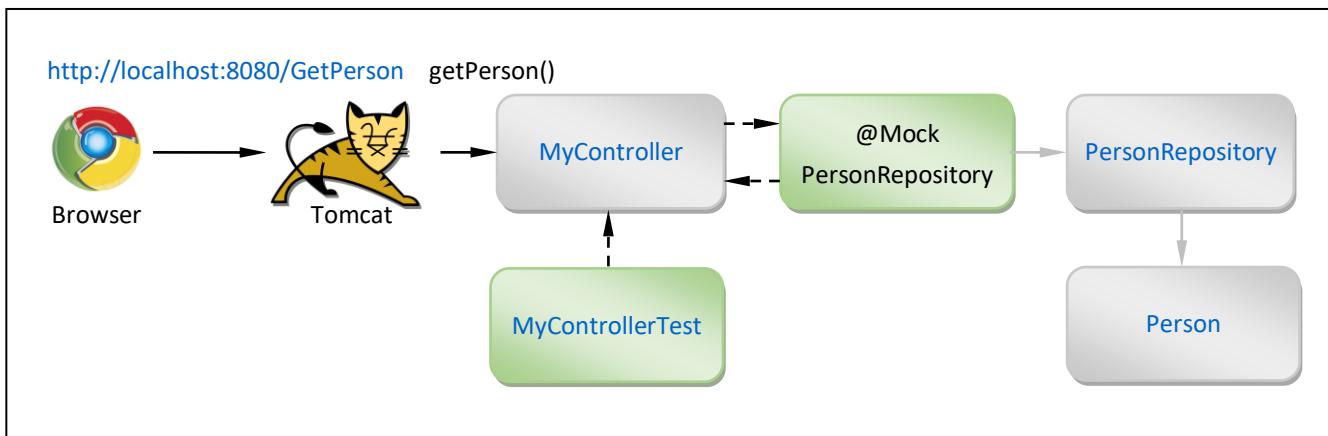
Info

[G]

- ArgumentMatchers is used in combination with verify() to specify how many times Mocked Method was called with specific Type of Parameter (like Integer) instead of specifying the exact value (like 1).

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
verify(personRepositoryMock, times(2)).getPersonById(ArgumentMatchers.any(Integer.class));
```

Parameters

PARAMETER	EXAMPLE	DESCRIPTION
any()	any(Integer.class)	Instance of specified Class
anyInt()	anyInt()	Instance of Integer Class

Procedure

— Create Project:	springboot_mockito_verify_argumentmatchers	(add Spring Boot Starters from the table)
— Create Package:	entities	(inside main package)
— Create Class:	Person.java	(inside entities package)
— Create Package:	repositories	(inside main package)
— Create Class:	PersonRepository.java	(inside entities package)
— Create Package:	controllers	(inside main package)
— Create Class:	MyController.java	(inside controllers package)
— Create Package:	controllers	(inside test package)
— Create Test Class:	MyControllerTest.java	(inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_verify_argumentmatchers.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_verify_argumentmatchers.repositories;

import com.ivoronline.springboot_mockito_verify_argumentmatchers.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(3, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_verify_argumentmatchers.controllers;

import com.ivorononline.springboot_mockito_verify_argumentmatchers.entities.Person;
import com.ivorononline.springboot_mockito_verify_argumentmatchers.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivorononline.springboot_mockito_verify_argumentmatchers.controllers;

import com.ivorononline.springboot_mockito_verify_argumentmatchers.entities.Person;
import com.ivorononline.springboot_mockito_verify_argumentmatchers.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.ArgumentMatchers;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    //ENDPOINT
    @Test
    void getPerson() {

        //MOCK REPOSITORY METHOD
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));
        when(personRepositoryMock.getPersonById(2)).thenReturn(new Person(1, "Jill", 60));

        //CALL REPOSITORY METHOD 2 TIMES WITH ANY INTEGER
        myController.getPerson(1);
        myController.getPerson(2);

        //VERIFY THAT METHOD getPersonById() WAS CALLED EXACTLY 2 TIMES WITH PARAMETER 1
        verify(personRepositoryMock, times(2)).getPersonById(ArgumentMatchers.any(Integer.class));
    }

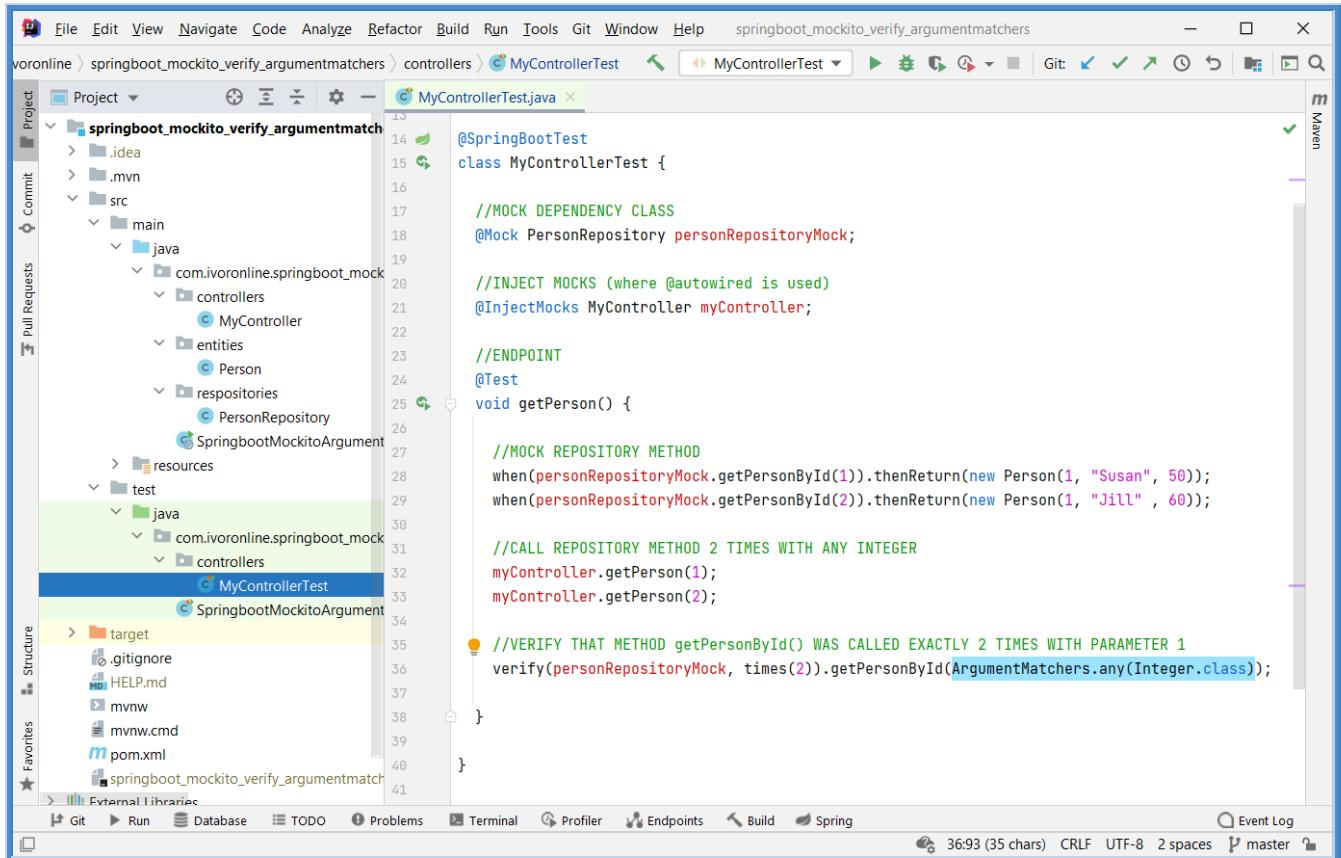
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



```
13  @SpringBootTest
14  class MyControllerTest {
15
16      //MOCK DEPENDENCY CLASS
17      @Mock PersonRepository personRepositoryMock;
18
19      //INJECT MOCKS (where @Autowired is used)
20      @InjectMocks MyController myController;
21
22      //ENDPOINT
23      @Test
24      void getPerson() {
25
26          //MOCK REPOSITORY METHOD
27          when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));
28          when(personRepositoryMock.getPersonById(2)).thenReturn(new Person(1, "Jill", 60));
29
30          //CALL REPOSITORY METHOD 2 TIMES WITH ANY INTEGER
31          myController.getPerson(1);
32          myController.getPerson(2);
33
34          //VERIFY THAT METHOD getPersonById() WAS CALLED EXACTLY 2 TIMES WITH PARAMETER 1
35          verify(personRepositoryMock, times(2)).getPersonById(ArgumentMatchers.any(Integer.class));
36      }
37
38  }
```

pom.xml

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

4.2.11 verify() - ArgumentCaptor

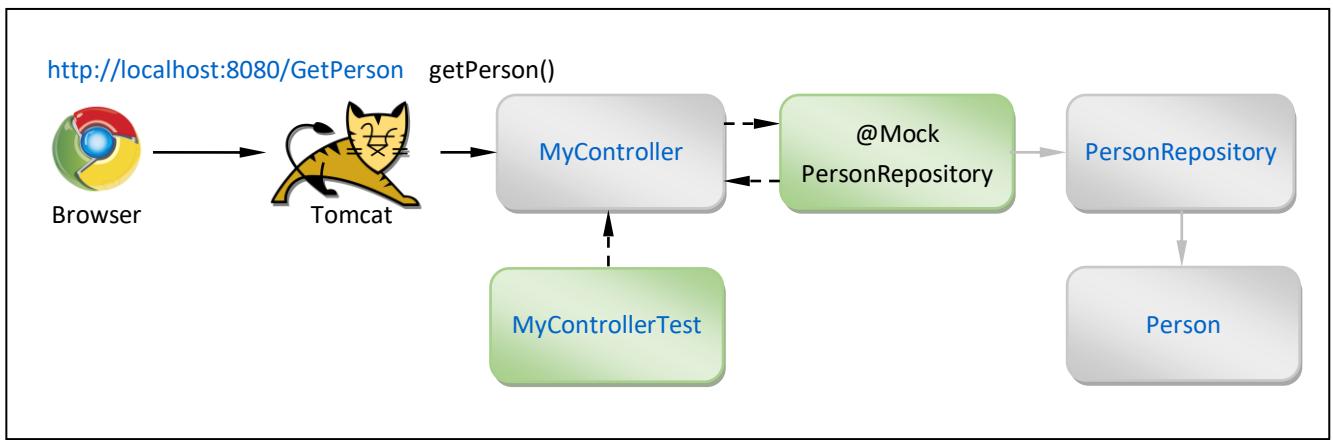
Info

[G]

- ArgumentCaptor is used in combination with `verify()` to capture arguments with which **Mocked Method** was called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
ArgumentCaptor<Integer> integerArgumentCaptor = ArgumentCaptor.forClass(Integer.class);
verify(personRepositoryMock, times(2)).getPersonById(integerArgumentCaptor.capture());
```

Procedure

- [Create Project:](#) `springboot_mockito_verify_argumentcaptor` (add Spring Boot Starters from the table)
- [Create Package:](#) `entities` (inside main package)
- [Create Class:](#) `Person.java` (inside entities package)
- [Create Package:](#) `repositories` (inside main package)
- [Create Class:](#) `PersonRepository.java` (inside entities package)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Package:](#) `controllers` (inside test package)
- [Create Test Class:](#) `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito_verify_argumentcaptor.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito_verify_argumentcaptor.repositories;

import com.ivoronline.springboot_mockito_verify_argumentcaptor.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(3, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_verify_argumentcaptor.controllers;

import com.ivorononline.springboot_mockito_verify_argumentcaptor.entities.Person;
import com.ivorononline.springboot_mockito_verify_argumentcaptor.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired PersonRepository personRepository;

    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);
        String name = person.name;
        Integer age = person.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivorononline.springboot_mockito_argumentcaptor.controllers;

import com.ivorononline.springboot_mockito_argumentcaptor.entities.Person;
import com.ivorononline.springboot_mockito_argumentcaptor.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.mockito.Mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepositoryMock;

    //INJECT MOCKS (where @autowired is used)
    @InjectMocks MyController myController;

    //ENDPOINT
    @Test
    void getPerson() {

        //CREATE CAPTOR THAT CAPTURES INTEGER ARGUMENTS
        //ArgumentCaptor<Integer> integerArgumentCaptor = ArgumentCaptor.forClass(Integer.class);

        //MOCK REPOSITORY METHOD
        when(personRepositoryMock.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));
        when(personRepositoryMock.getPersonById(2)).thenReturn(new Person(1, "Susan", 50));

        //CALL REPOSITORY METHOD
        myController.getPerson(1);
        myController.getPerson(2);

        //CAPTURE INTEGER ARGUMENT (from last call)
        ArgumentCaptor<Integer> integerArgumentCaptor = ArgumentCaptor.forClass(Integer.class);
        verify(personRepositoryMock, times(2)).getPersonById(integerArgumentCaptor.capture());

        //GET LAST CAPTURED ARGUMENT
        Integer argument = integerArgumentCaptor.getValue();

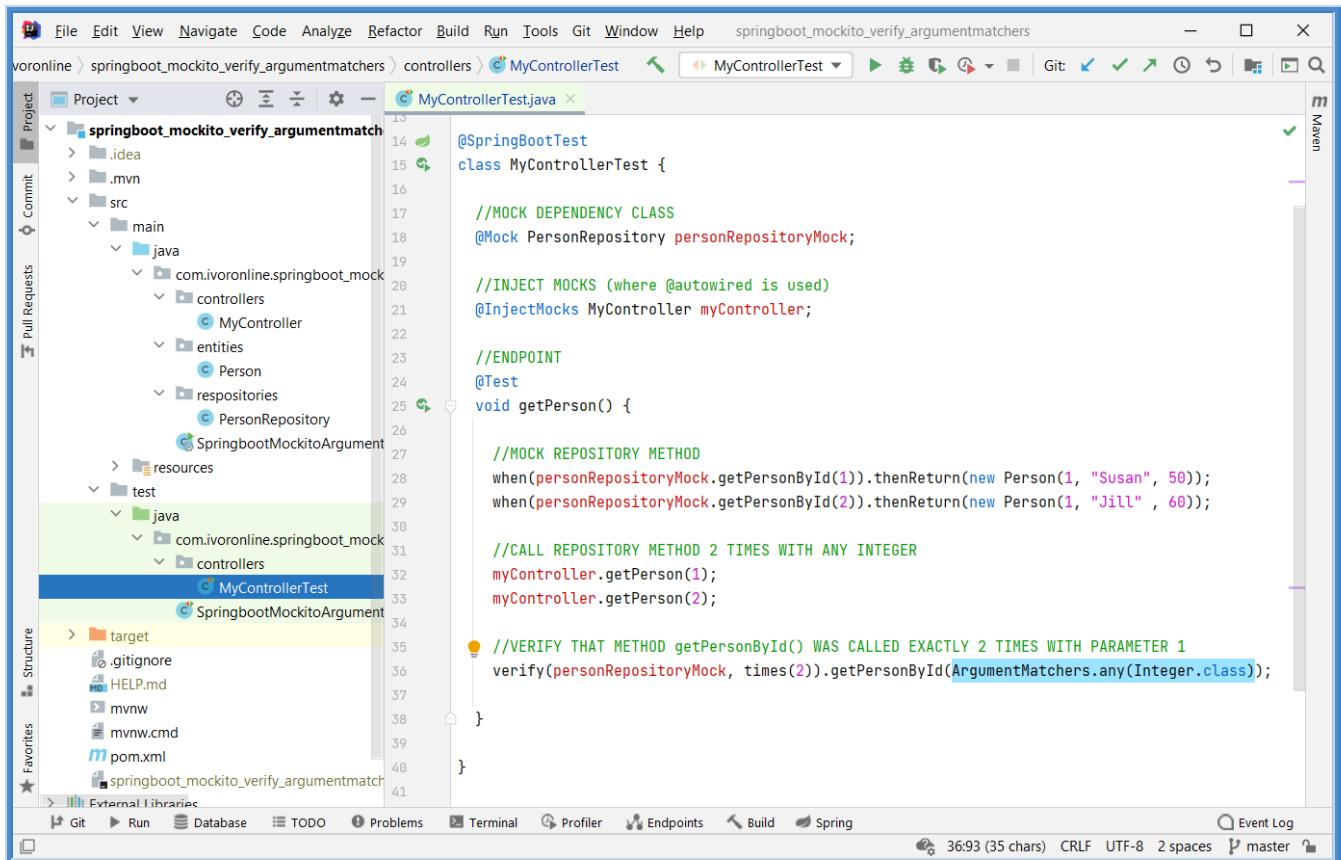
        //DISPLAY CAPTURED ARGUMENT
        System.out.println(argument);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.3 MockMVC

Info

- Following tutorials show how to use `MockMvc` to test Controllers by
 - performing HTTP Requests
 - verifying HTTP Responses

Syntax

```
mockMvc.perform(get("/Hello"))           //perform HTTP Request
    .andExpect(status().isOk());          //verify HTTP Response
```

Instantiate MockMvc

- `MockMvc` can be instantiated in different ways as shown below.

`@SpringBootTest, @AutoConfigureMockMvc, @Autowired`

```
@SpringBootTest
@AutoConfigureMockMvc
class MyControllerTest {
    @Autowired MockMvc mockMvc;
```

`@WebMvcTest, @Autowired`

```
@WebMvcTest
class MyControllerTest {
    @Autowired MockMvc mockMvc;
```

MockMvcBuilders

```
class MyControllerTest {
    MyController myController = new MyController();
    @Test
    void hello() throws Exception {
        MockMvc mockMvc = MockMvcBuilders.standaloneSetup(myController).build();
```

4.3.1 @WebMvcTest

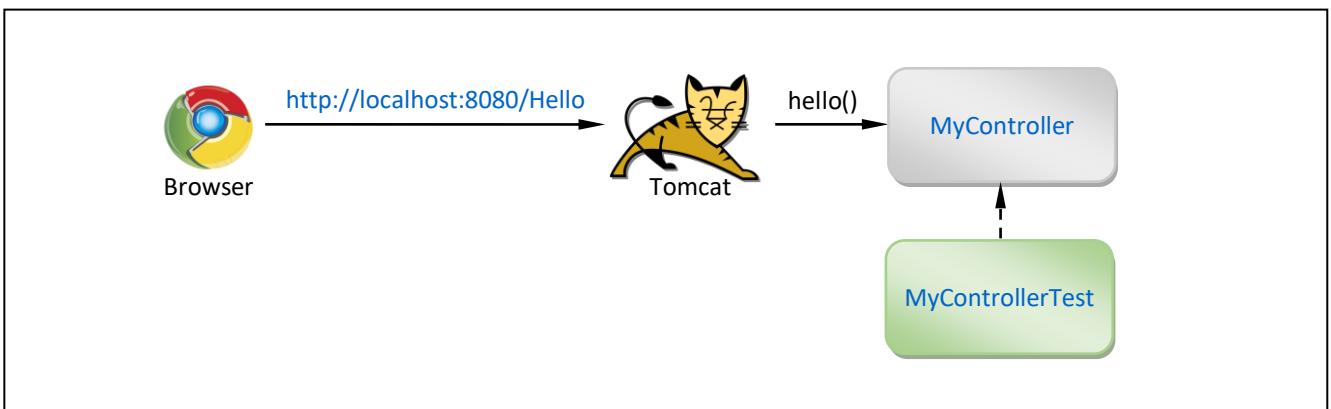
Info

[G] [R]

- This tutorial shows how to use `@WebMvcTest` which
 - creates Application Context that only contains Beans needed for testing web Controllers
 - instantiates `MockMvc` (as one such Bean)
- This means that `@WebMvcTest` will instantiate Classes that are Annotated with `@Controller`.
This means that you **can** use `@Autowired MyController myController` to inject Instance of `MyController`.
- But `@WebMvcTest` will not instantiate Classes that are Annotated with `@Entity`.
This means that you **can't** use `@Autowired PersonEntity personEntity` to inject Instance of `PersonEntity`.
- However you can use `@MockBean` or manually instantiate Classes that are not automatically pulled into Context.
Purpose of `@WebMvcTest` is to keep tests faster than when using `@SpringBootTest` which loads everything into Context.
But to also allow you to add additional instances if needed to fine tune your tests.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
@WebMvcTest
class MyControllerTest {
    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_webmvctest.controllers;

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() {
        return "Hello from Controller";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_webmvctest.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

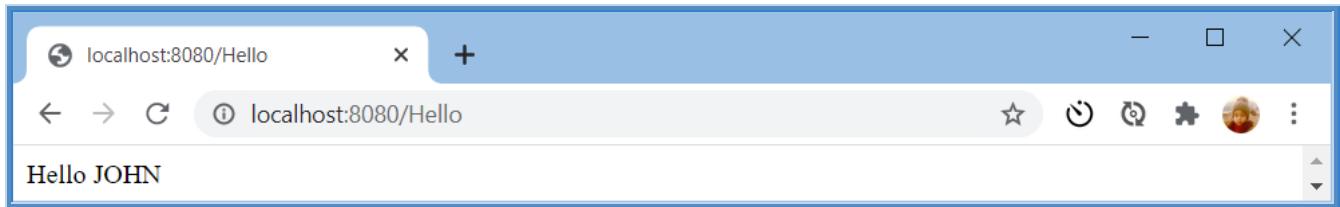
    @Autowired MockMvc mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {
        mockMvc.perform(get("/Hello"))
            .andExpect(status().isOk());
    }

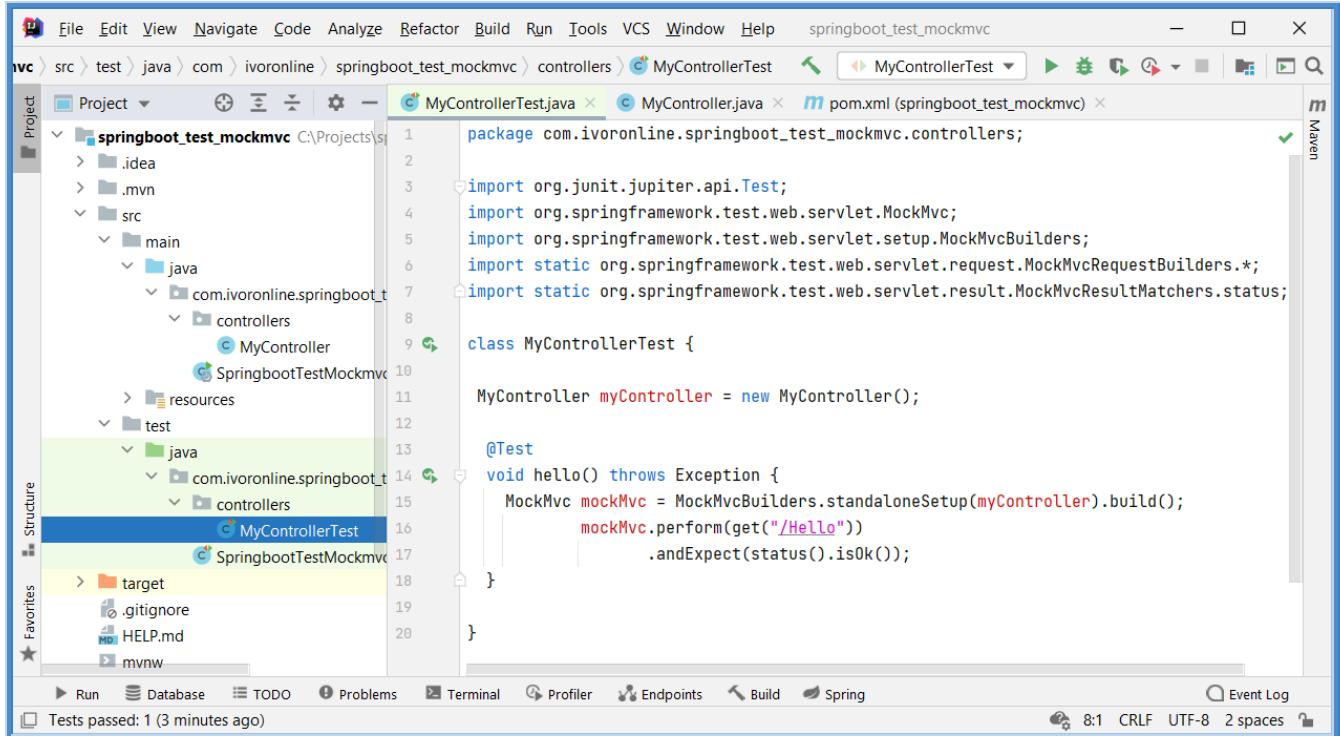
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

4.3.2 Test - Request - URL Mapping

Info

[G] [R]

- This tutorial shows how to test if Endpoint was called
 - for specific URL
 - with required HTTP Request Method: GET, POST, ...
- If Endpoint expects Parameters and you give none you get 400 Bad Request

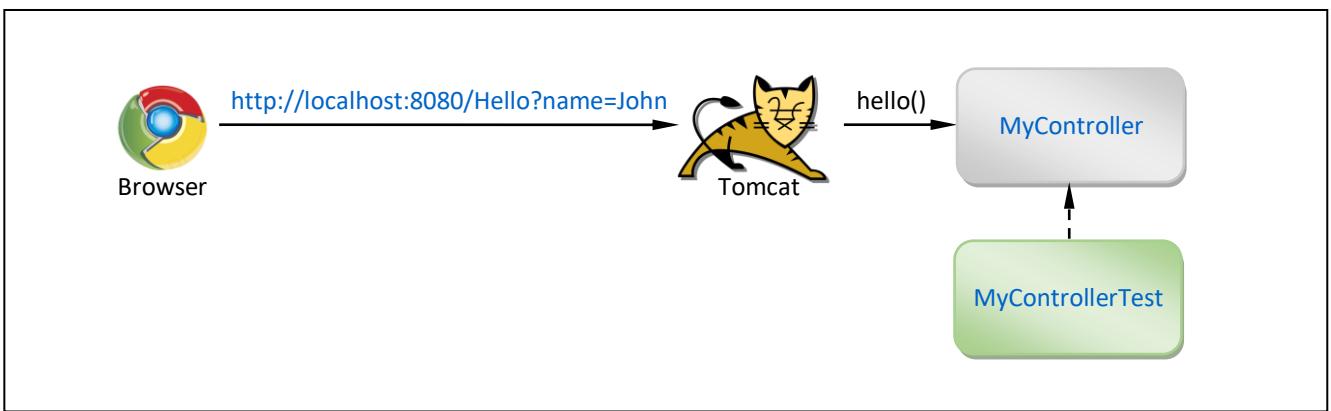
404 Page Not Found

405 Method Not Allowed

400 Bad Request

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
MockHttpServletRequestBuilder request = get("/Hello");           //CREATE REQUEST
mockMvc.perform(request).andExpect(status().isOk());          //PERFORM REQUEST. CHECK STATUS.
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_responsetatus` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_responsetatus.controllers;

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

@Controller
public class MyController {

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

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_responsetatus.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {

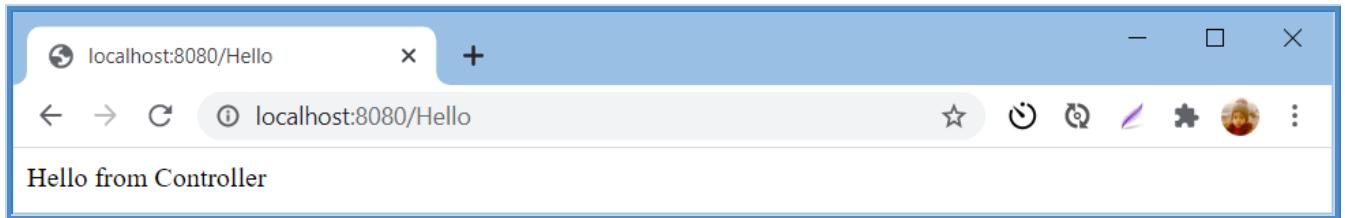
        //CREATE GET REQUEST
        MockMvcRequestBuilder request = get("/Hello");

        //PERFORM REQUEST
        mockMvc.perform(request).andExpect(status().isOk());

    }
}
```

Result

<http://localhost:8080>Hello>



Wrong URL

get("/Hello1?name=John")

```
MockHttpServletRequest:  
    HTTP Method = GET  
    Request URI = /Hello1  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 404  
    Error message = null  
  
java.lang.AssertionError: Status expected:<200> but was:<404>  
Expected :200  
Actual   :404 Page Not Found
```

Wrong Method

post("/Hello?name=John")

```
MockHttpServletRequest:  
    HTTP Method = POST  
    Request URI = /Hello  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 405  
    Error message = Request method 'POST' not supported  
    Headers = [Allow:"GET"]  
  
java.lang.AssertionError: Status expected:<200> but was:<405>  
Expected :200  
Actual   :405 Method Not Allowed
```

Run Test Class: MyControllerTest.java

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, Git, Window, Help.
- Toolbar:** Includes icons for Run, Stop, Refresh, and Git operations.
- Project Explorer (Left):** Shows the project structure:
 - Project node (selected).
 - springboot_test_mockmvc_responsetatus node (selected).
 - .idea folder.
 - .mvn folder.
 - src folder
 - main folder
 - java folder
 - com.ivoronline.springboot_test_mockmvc_responsetatus package
 - controllers folder
 - MyController.java (selected).
 - SpringbootTestMockmvcResponsetatusApplication.java
 - resources folder.
 - test folder
 - java folder
 - com.ivoronline.springboot_test_mockmvc_responsetatus package
 - controllers folder
 - MyControllerTest.java (selected).
 - target folder.
 - Code Editor (Right):** Displays the content of MyControllerTest.java.

```
package com.ivoronline.springboot_test_mockmvc_responsetatus;

import ...;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {
        mockMvc.perform(get("/Hello?name=John"))
            .andExpect(status().isOk());
    }
}
```
 - Bottom Navigation Bar:** Git, Run, Database, TODO, Problems, Terminal, Profiler, Endpoints, Build, Spring.
 - Status Bar:** Tests failed: 1, passed: 0 (4 minutes ago), 25:1, CRLF, UTF-8, 2 spaces, master.

pom.xml

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

4.3.3 Test - Request - Parameters

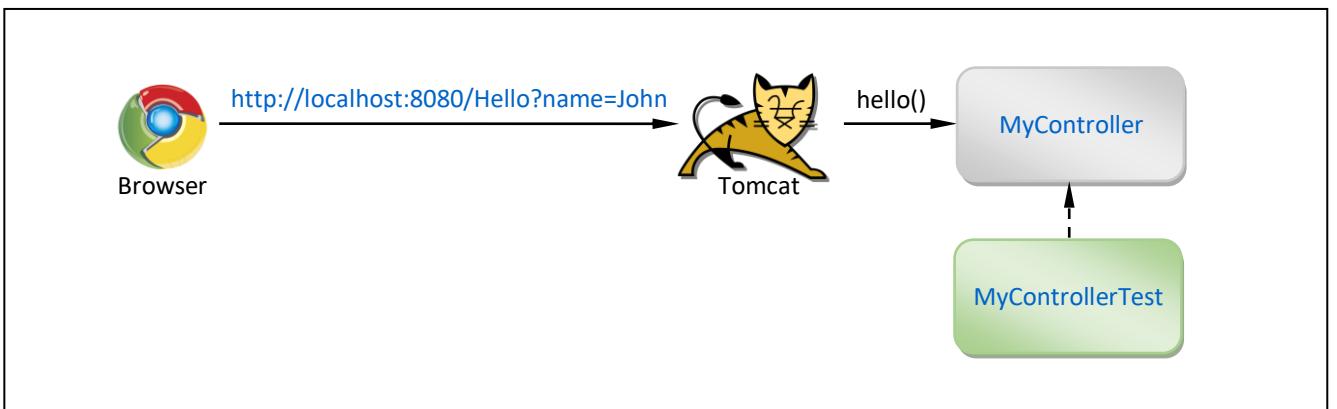
Info

[G] [R]

- This tutorial shows how to test if Endpoint was called with required HTTP Request Parameters [400 Bad Request](#).

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
MockHttpServletRequestBuilder request = get("/Hello?name=John");           //CREATE REQUEST
MockHttpServletRequestBuilder request = get("/Hello").param("name", "John"); //CREATE REQUEST (alternative)
mockMvc.perform(request).andExpect(status().isOk());                      //PERFORM REQUEST. CHECK STATUS.
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_responsetatus` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_requestparameters.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @GetMapping("/Hello")
    public String hello(@RequestParam String name) {
        return "Hello " + name;
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_requestparameters.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {

        //CREATE REQUEST
        //MockHttpServletRequestBuilder request = get("/Hello?name=John");
        MockHttpServletRequestBuilder request = get("/Hello").param("name", "John");

        //PERFORM REQUEST
        mockMvc.perform(request).andExpect(status().isOk());

    }
}
```

Result

<http://localhost:8080>Hello?name=John>

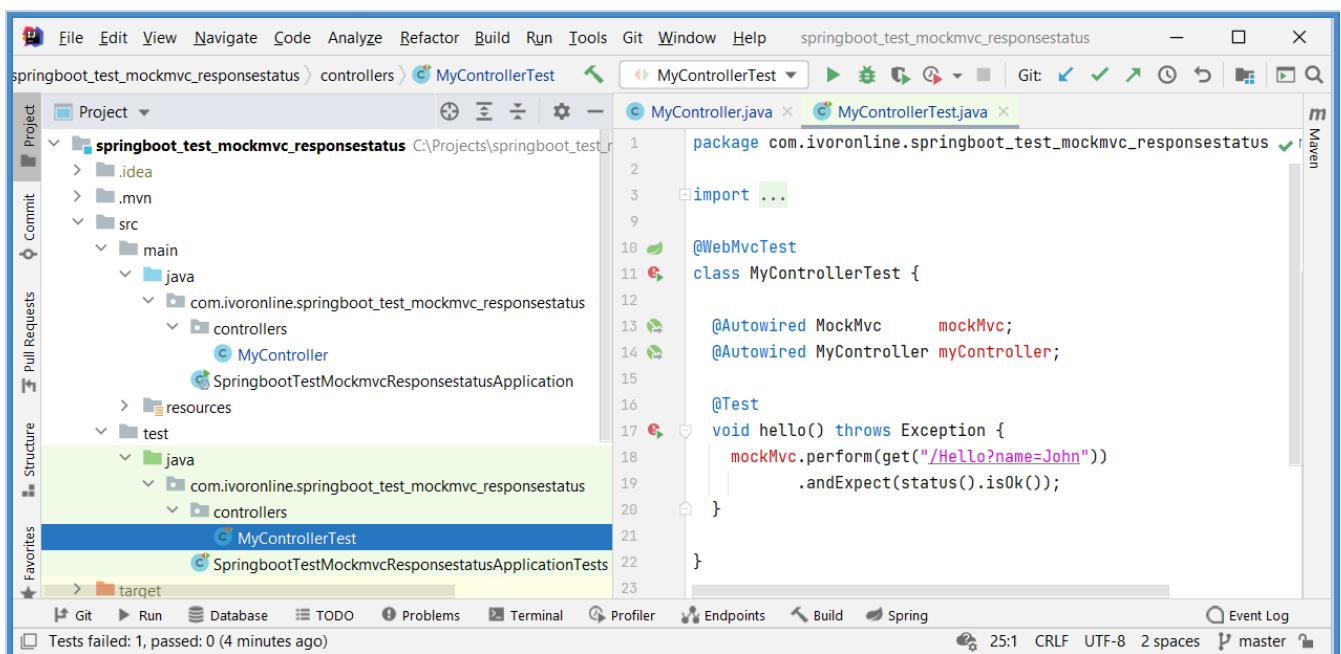


Missing Parameter

get("/Hello?name1=John")

```
MockHttpServletRequest:  
    HTTP Method = GET  
    Request URI = /Hello  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 400  
    Error message = Required String parameter 'name' is not present  
  
java.lang.AssertionError: Status expected:<200> but was:<400>  
Expected :200  
Actual   :400 Bad Request
```

Run Test Class: MyControllerTest.java



pom.xml

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

4.3.4 Test - Request - Path Variables

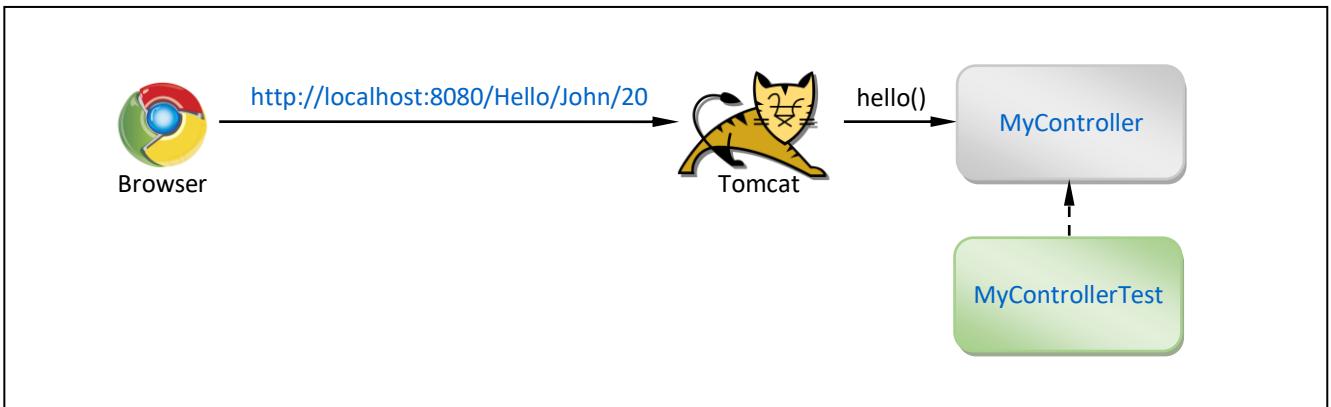
Info

[G] [R]

- This tutorial shows how to test if Endpoint was called with required Path Variables [404 Page Not Found](#).

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
MockHttpServletRequestBuilder request = get("/Hello/John/20");           //CREATE REQUEST
mockMvc.perform(request).andExpect(status().isOk());                      //PERFORM REQUEST. CHECK STATUS.
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_pathvariable` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_pathvariable.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @GetMapping("/Hello/{FirstName}/{age}")
    public String hello(
        @PathVariable("FirstName") String name, //IF NAMES ARE DIFFERENT
        @PathVariable String age //IF NAMES ARE THE SAME
    ) {
        return name + " is " + age + " years old";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_pathvariable.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {

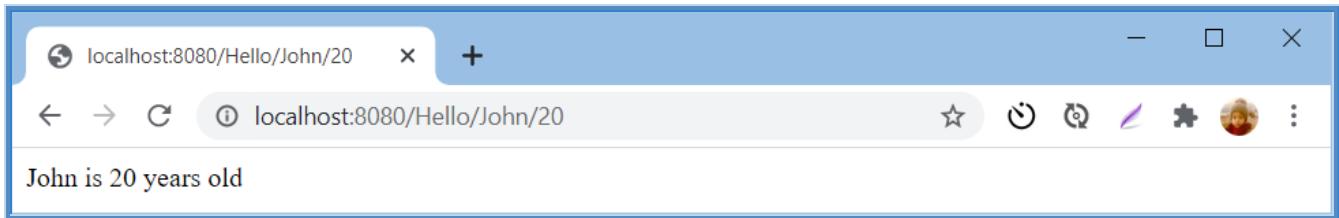
        //CREATE REQUEST
        MockMvcRequestBuilder request = get("/Hello/John/20");

        //PERFORM REQUEST
        mockMvc.perform(request).andExpect(status().isOk());

    }
}
```

Result

<http://localhost:8080>Hello/John/20>

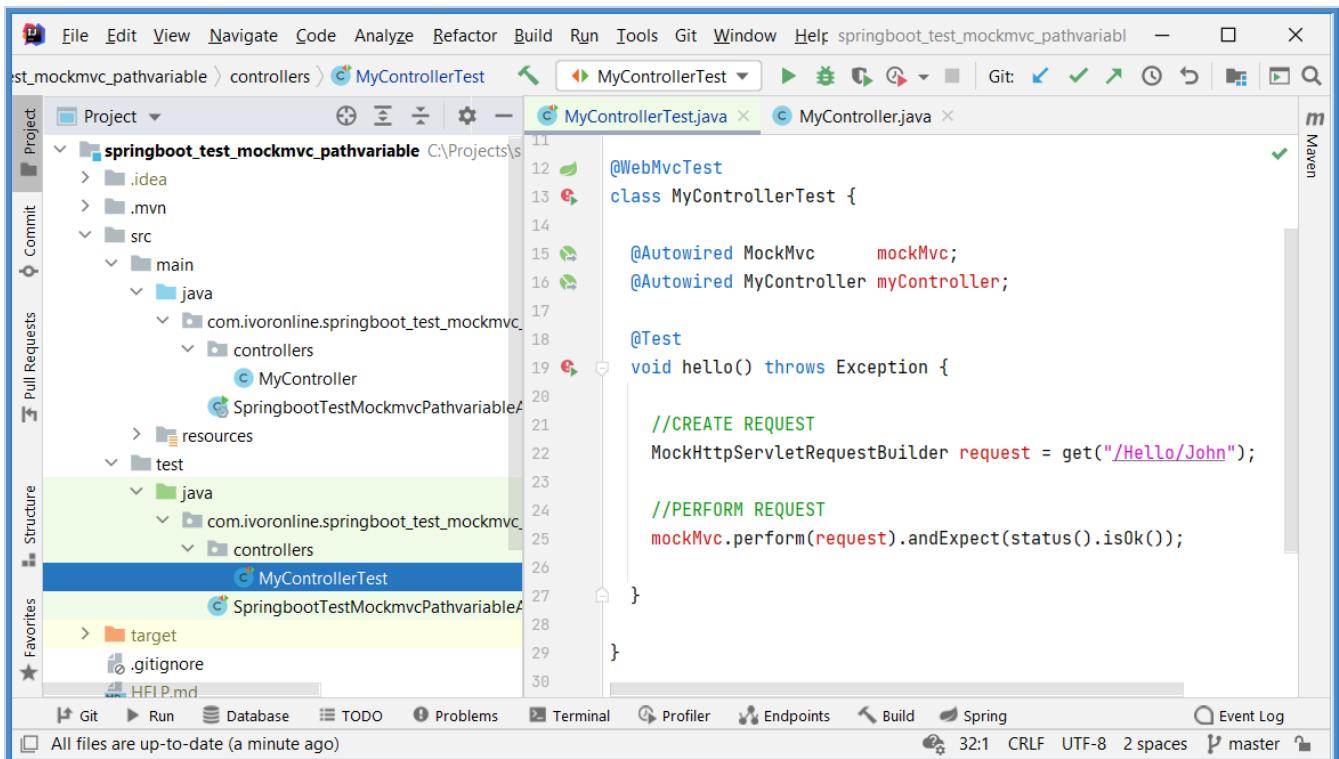


Missing Parameter

get("/Hello/John")

```
MockHttpServletRequest:  
    HTTP Method = GET  
    Request URI = /Hello/John  
    Parameters = {}  
  
MockHttpServletResponse:  
    Status = 404  
    Error message = null  
  
java.lang.AssertionError: Status expected:<200> but was:<404>  
Expected :200  
Actual   :404 Page Not Found
```

Run Test Class: MyControllerTest.java



pom.xml

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

4.3.5 Test - Request - Body - JSON

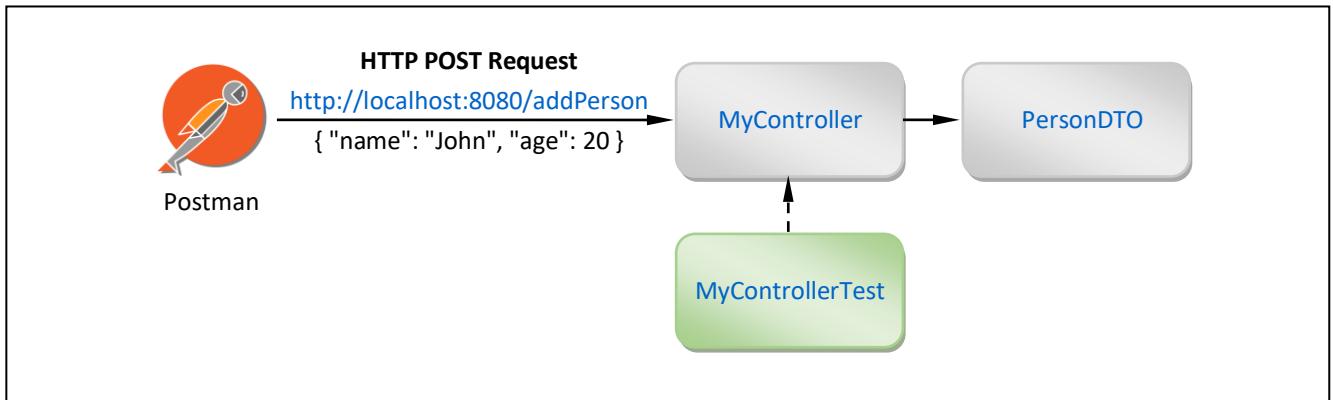
Info

[G] [R]

- This tutorial shows how to Desterilize DTO and send it to Controller through JSON Body.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
//CREATE REQUEST
MockHttpServletRequestBuilder request =
    post      ("/AddPerson")           //POST
    .contentType("application/json")   //Type of Body Content
    .content   (personDTOSerialized); //{{ "name": "John", "age": 20 }}

//PERFORM REQUEST. CHECK STATUS.
mockMvc.perform(request).andExpect(status().isOk());
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_jsonbody` (add Spring Boot Starters from the table)
- [Edit File:](#) `pom.xml` (add validation dependency)
- [Create Package:](#) `DTO` (inside main package)
 - [Create Class:](#) `PersonDTO.java` (inside package `DTO`)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

pom.xml

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

PersonDTO.java

```
package com.ivoronline.springboot_test_mockmvc_jsonbody.DTO;

import javax.validation.constraints.NotNull;

public class PersonDTO {
    @NotNull public String name;
    @NotNull public Integer age;
}
```

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_jsonbody.controllers;

import com.ivoronline.springboot_test_mockmvc_jsonbody.DTO.PersonDTO;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.validation.Valid;

@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/AddPerson")
    public String addPerson(@Valid @RequestBody PersonDTO personDTO) {

        //GET DATA FROM PersonDTO
        String name = personDTO.name;
        Integer age = personDTO.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_jsonbody.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ivoronline.springboot_test_mockmvc_jsonbody.DTO.PersonDTO;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;
    @Autowired ObjectMapper objectMapper;

    @Test
    void addPerson() throws Exception {

        //CREATE PERSON DTO
        PersonDTO personDTO      = new PersonDTO();
        personDTO.name = "John";
        personDTO.age  = 20;

        //SERIALIZE PERSON DTO INTO JSON STRING { "name": "John", "age": 20 }
        String personDTOSerialized = objectMapper.writeValueAsString(personDTO);

        //CREATE REQUEST
        MockHttpServletRequestBuilder request =
            post      ("/AddPerson")
            .contentType("application/json")
            .content   (personDTOSerialized);

        //PERFORM REQUEST
        mockMvc.perform(request).andExpect(status().isOk());

    }
}
```

Result

- Start Postman

POST

```
http://localhost:8080/addPerson
```

Headers

(add Key-Value)

```
Content-Type: application/json
```

Body

(option: raw)

```
{
  "name" : "John",
  "age"   : 20
}
```

Postman

The screenshot shows the Postman application window. In the top navigation bar, 'File', 'Edit', 'View', and 'Help' are visible. Below the bar, there are tabs for 'Home', 'Workspaces', 'Reports', and 'Explore'. A search bar says 'Search Postman'. On the right side of the header, there are icons for cloud, user, gear, and notifications, followed by 'Upgrade'. The main workspace shows a 'DTO / New Request' section with a 'GET' method and URL 'http://localhost:8080/AddPerson'. Below it, under 'Body', the JSON payload is defined:

```
1 {
2   "name" : "John",
3   "age"   : 20
4 }
```

The 'Pretty' tab is selected in the preview area, which displays the JSON as:

```
1 John is 20 years old
```

At the bottom of the preview area, status information is shown: 'Status: 200 OK Time: 99 ms Size: 184 B Save Response'. The bottom navigation bar includes 'Find and Replace', 'Console', and environment management buttons for 'Bootcamp', 'Runner', and 'Trash'.

Missing Parameter

personDTO.name = null

```
MockHttpServletRequest:
    HTTP Method = POST
    Request URI = /AddPerson
    Parameters = {}
    Headers = [Content-Type:"application/json;charset=UTF-8", Content-Length:"22"]
    Body = {"name":null,"age":20}

MockHttpServletResponse:
    Status = 400
    Error message = null

java.lang.AssertionError: Status expected:<200> but was:<400>
Expected :200
Actual    :400 Bad Request
```

Run Test Class: MyControllerTest.java

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure for "springboot_test_mockmvc_jsonbody". It includes a .idea folder, .mvn, and src directory containing main and test packages. The test package contains controllers, DTO, and application.properties files.
- Code Editor:** The main window displays the content of `MyControllerTest.java`. The code uses MockMvc to test a controller named `MyController` which handles `PersonDTO`.
- Toolbars and Status Bar:** The top bar has standard IntelliJ tabs like File, Edit, View, Navigate, etc. The bottom status bar shows the time as 33:44 and the branch as master.

```
14 @WebMvcTest
15 class MyControllerTest {
16
17     @Autowired MockMvc mockMvc;
18     @Autowired MyController myController;
19     @Autowired ObjectMapper objectMapper;
20
21     @Test
22     void addPerson() throws Exception {
23
24         //CREATE PERSON DTO
25         PersonDTO personDTO = new PersonDTO();
26         personDTO.name = "John";
27         personDTO.age = 20;
28
29         //SERIALIZE PERSON DTO INTO JSON STRING
30         String personDTOSerialized = objectMapper.writeValueAsString(personDTO);
31
32         //CREATE REQUEST
33         MockHttpServletRequestBuilder request =
34             post ("/AddPerson")
35             .contentType("application/json")
36             .content (personDTOSerialized);
37
38         //PERFORM REQUEST
39         mockMvc.perform(request).andExpect(status().isOk());
40
41     }
42
43 }
44 }
```

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-validation</artifactId>
    </dependency>
</dependencies>
```

4.3.6 Test - Response - Status

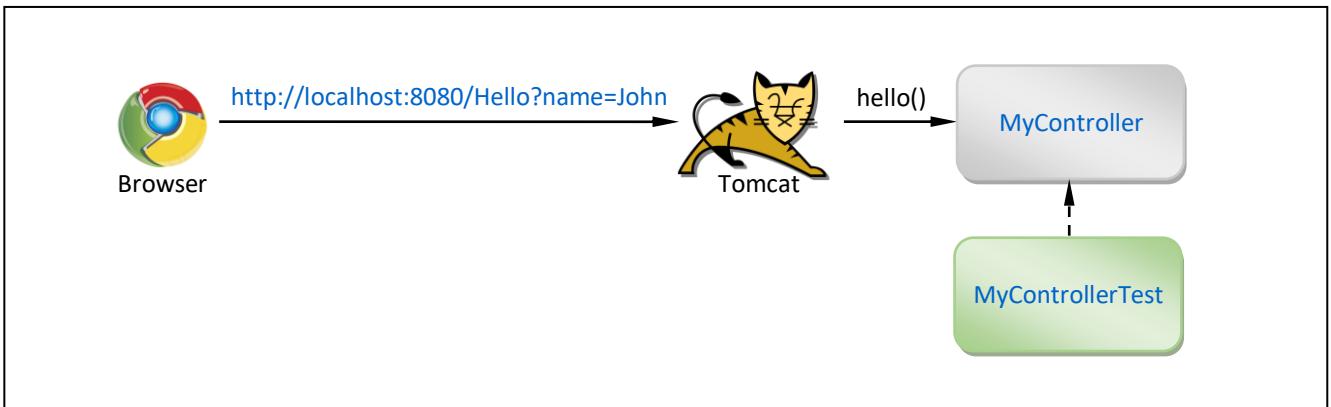
Info

[G] [R]

- This tutorial shows how to test different Response Statuses from Controller

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
//CREATE REQUEST
MockHttpServletRequestBuilder request = get("/Hello?name=John");

//PERFORM REQUEST
mockMvc.perform(request).andExpect(status().isOk());
mockMvc.perform(request).andExpect(status().is(200));
```

Response Statuses

STATUS	MESSAGE	METHOD	URL
200	OK	isOk()	get("/Hello?name=John")
400	BAD REQUEST	isBadRequest()	get("/Hello?name1=John")
404	PAGE NOT FOUND	isNotFound()	get("/Hello1?name=John")
405	METHOD NOT ALLOWED	isMethodNotAllowed()	post("/Hello?name=John")

Procedure

- [Create Project:](#) `springboot_test_mockmvc_responsetatus` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_requestparameters.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @GetMapping("/Hello")
    public String hello(@RequestParam String name) {
        return "Hello " + name;
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_responsetatus.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {

        //CREATE REQUEST
        MockMvcRequestBuilder request = get("/Hello?name=John");

        //PERFORM REQUEST
        mockMvc.perform(request).andExpect(status().isOk());
        mockMvc.perform(request).andExpect(status().is(200));

    }

}
```

Result

<http://localhost:8080>Hello?name=John>



Wrong URL

get("/Hello1?name=John")

```
MockHttpServletRequest:  
    HTTP Method = GET  
    Request URI = /Hello1  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 404  
    Error message = null  
  
java.lang.AssertionError: Status expected:<200> but was:<404>  
Expected :200  
Actual   :404 Page Not Found
```

Wrong Method

post("/Hello?name=John")

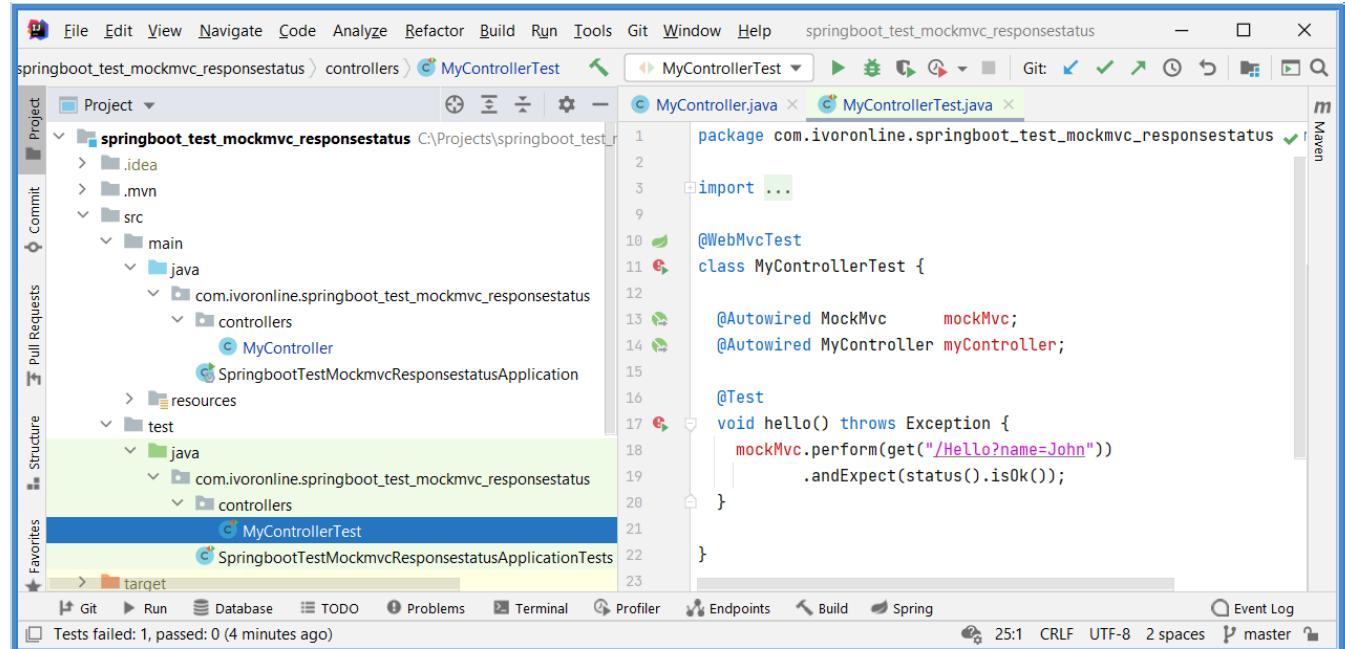
```
MockHttpServletRequest:  
    HTTP Method = POST  
    Request URI = /Hello  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 405  
    Error message = Request method 'POST' not supported  
    Headers = [Allow:"GET"]  
  
java.lang.AssertionError: Status expected:<200> but was:<405>  
Expected :200  
Actual   :405 Method Not Allowed
```

Missing Parameter

get("/Hello?name1=John")

```
MockHttpServletRequest:  
    HTTP Method = GET  
    Request URI = /Hello  
    Parameters = {name1=[John]}  
  
MockHttpServletResponse:  
    Status = 400  
    Error message = Required String parameter 'name' is not present  
  
java.lang.AssertionError: Status expected:<200> but was:<400>  
Expected :200  
Actual   :400 Bad Request
```

Run Test Class: MyControllerTest.java



```
package com.ivoronline.springboot_test_mockmvc_responsetatus;

import ...;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {
        mockMvc.perform(get("/Hello?name=John"))
            .andExpect(status().isOk());
    }
}
```

pom.xml

```
<dependencies>

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

</dependencies>
```

4.3.7 Test - Response - Body - Text

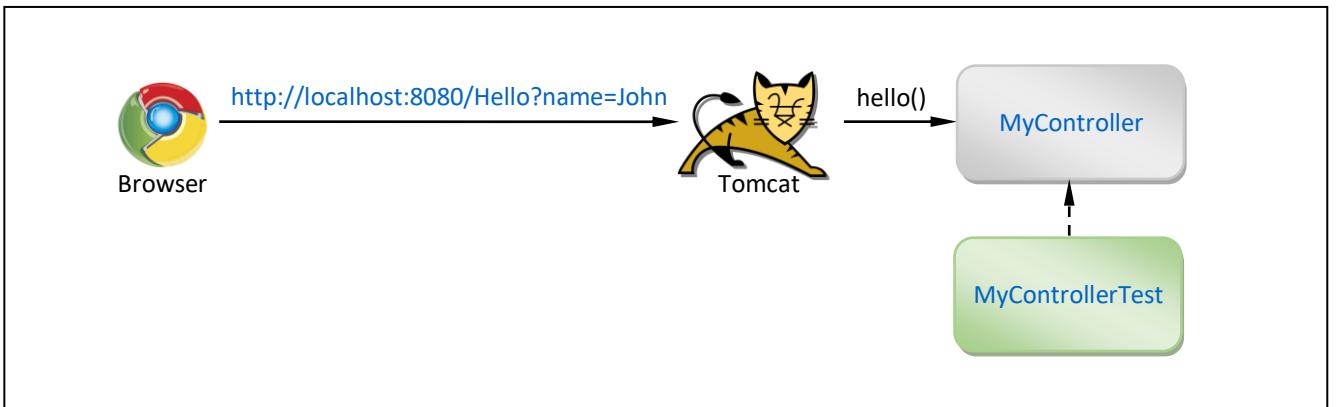
Info

[G] [R]

- This tutorial shows how to test if Controller returned valid Text in Response Body.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
//CREATE REQUEST
MockHttpServletRequestBuilder request = get("/Hello?name=John");

//PERFORM REQUEST. RETURN RESULT.
MvcResult mvcResult = mockMvc.perform(request).andReturn();
String responseBody = mvcResult.getResponse().getContentAsString();

//CHECK RESPONSE BODY
assertEquals("Hello John", responseBody);
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_response_body` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_response_body.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @GetMapping("/Hello")
    public String hello(@RequestParam String name) {
        return "Hello " + name;
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_response_body.controllers;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcHttpRequest;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;

    @Test
    void hello() throws Exception {

        //CREATE REQUEST
        MockMvcHttpRequest request = get("/Hello?name=John");

        //PERFORM REQUEST. RETURN RESULT.
        MvcResult mvcResult      = mockMvc.perform(request).andReturn();
        String      responseBody = mvcResult.getResponse().getContentAsString();

        //CHECK RESPONSE BODY
        assertEquals("Hello John", responseBody);

    }
}
```

Result

<http://localhost:8080>Hello?name=John>

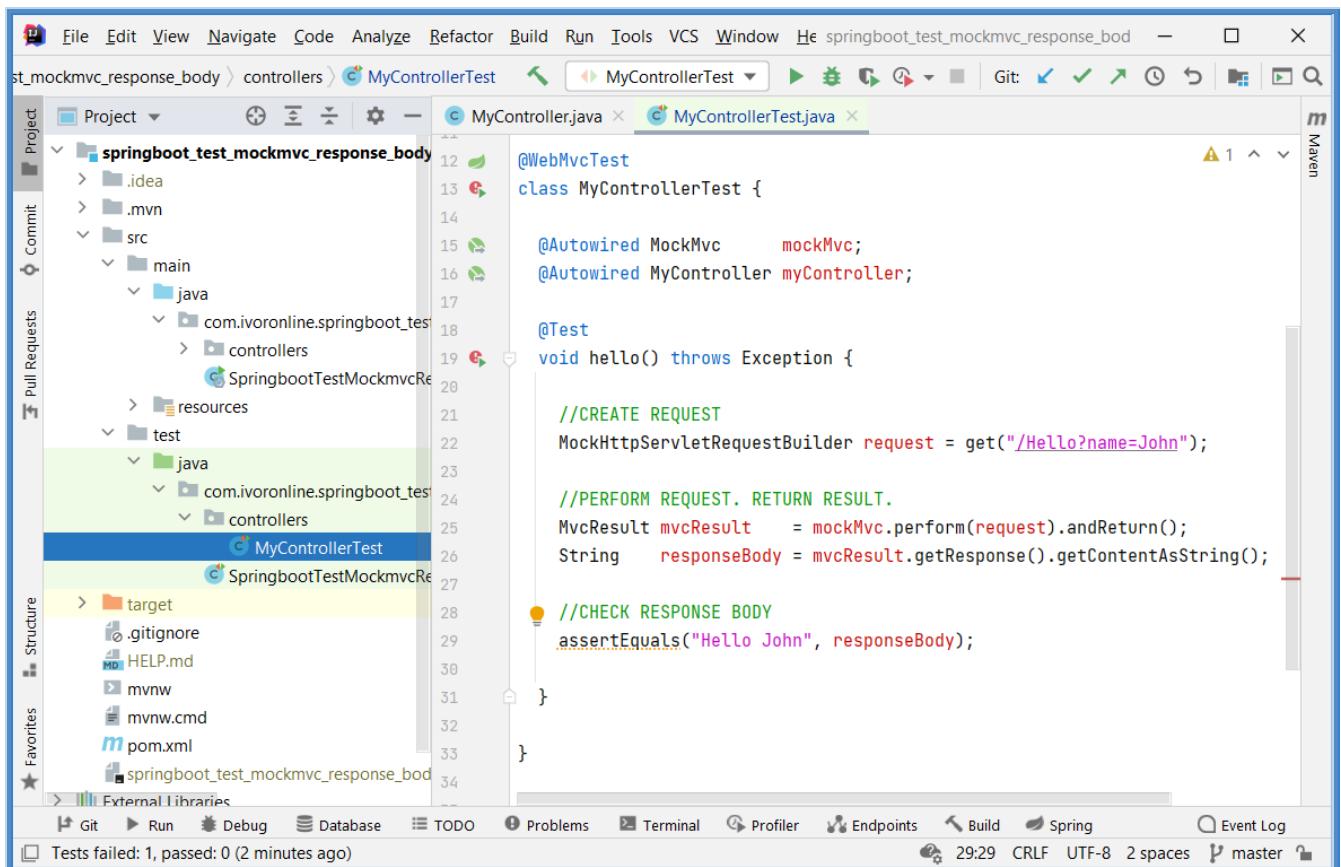


Wrong Response Body

```
assertEquals("Hello John1", responseBody)
```

```
org.opentest4j.AssertionFailedError:  
Expected :Hello John1  
Actual   :Hello John
```

Run Test Class: MyControllerTest.java



pom.xml

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

4.3.8 Test - Response - Body - JSON

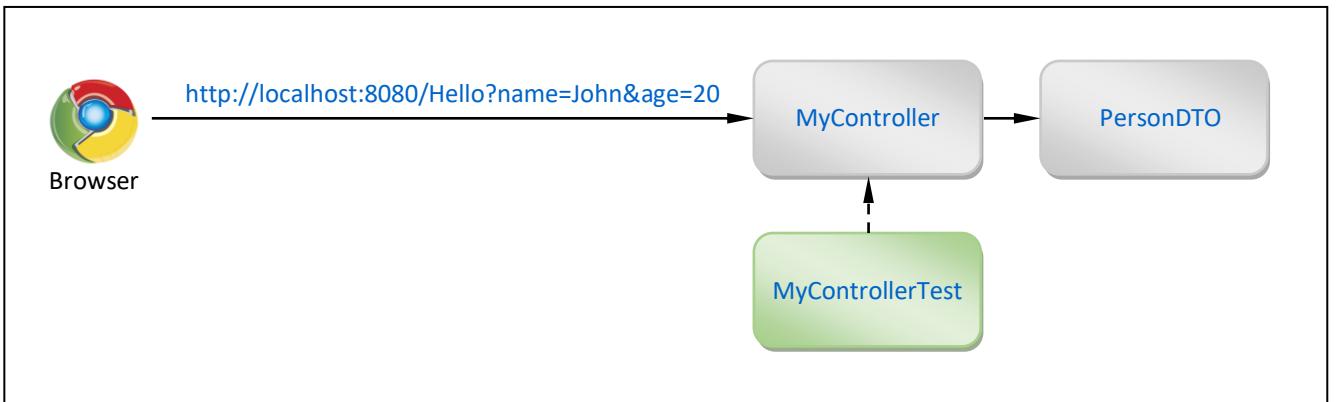
Info

[G] [R]

- This tutorial shows how to test if Controller returns valid JSON in Response Body.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Syntax

```
//CREATE REQUEST
MockHttpServletRequestBuilder request = get("/Hello?name=John&age=20");

//PERFORM REQUEST. RETURN RESULT.
MvcResult mvcResult = mockMvc.perform(request).andReturn();
String responseBody = mvcResult.getResponse().getContentAsString();

//GENERATE EXPECTED JSON
String expectedPersonDTOJSON = objectMapper.writeValueAsString(new PersonDTO("John", 20));

//CHECK RESPONSE BODY
assertEquals(expectedPersonDTOJSON, responseBody);
```

Procedure

- [Create Project:](#) `springboot_test_mockmvc_response_body_json` (add Spring Boot Starters from the table)
- [Create Package:](#) `DTO` (inside main package)
- [Create Class:](#) `PersonDTO.java` (inside package DTO)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

PersonDTO.java

```
package com.ivoronline.springboot_test_mockmvc_response_body_json.DTO;

public class PersonDTO {
    public String name;
    public Integer age;
}
```

MyController.java

```
package com.ivoronline.springboot_test_mockmvc_response_body_json.controllers;

import com.ivoronline.springboot_test_mockmvc_response_body_json.DTO.PersonDTO;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @ResponseBody
    @GetMapping("/Hello")
    public PersonDTO hello(@RequestParam String name, @RequestParam Integer age) {

        //CREATE PERSON DTO
        PersonDTO personDTO = new PersonDTO();
        personDTO.name = name;
        personDTO.age = age;

        //RETURN SOMETHING
        return personDTO;
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_test_mockmvc_response_body_json.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ivoronline.springboot_test_mockmvc_response_body_json.DTO.PersonDTO;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest
class MyControllerTest {

    @Autowired MockMvc      mockMvc;
    @Autowired MyController myController;
    @Autowired ObjectMapper objectMapper;

    @Test
    void hello() throws Exception {

        //CREATE REQUEST
        MockHttpServletRequestBuilder request = get("/Hello?name=John&age=20");

        //PERFORM REQUEST. RETURN RESULT.
        MvcResult mvcResult      = mockMvc.perform(request).andReturn();
        String   responseBody = mvcResult.getResponse().getContentAsString();

        //CREATE EXPECTED PERSON DTO
        PersonDTO expectedPersonDTO      = new PersonDTO();
        expectedPersonDTO.name = "John";
        expectedPersonDTO.age  = 20;

        String   expectedPersonDTOJSON = objectMapper.writeValueAsString(expectedPersonDTO);

        //CHECK RESPONSE BODY
        assertEquals(expectedPersonDTOJSON, responseBody);
    }
}
```

Result

<http://localhost:8080>Hello?name=John&age=20>

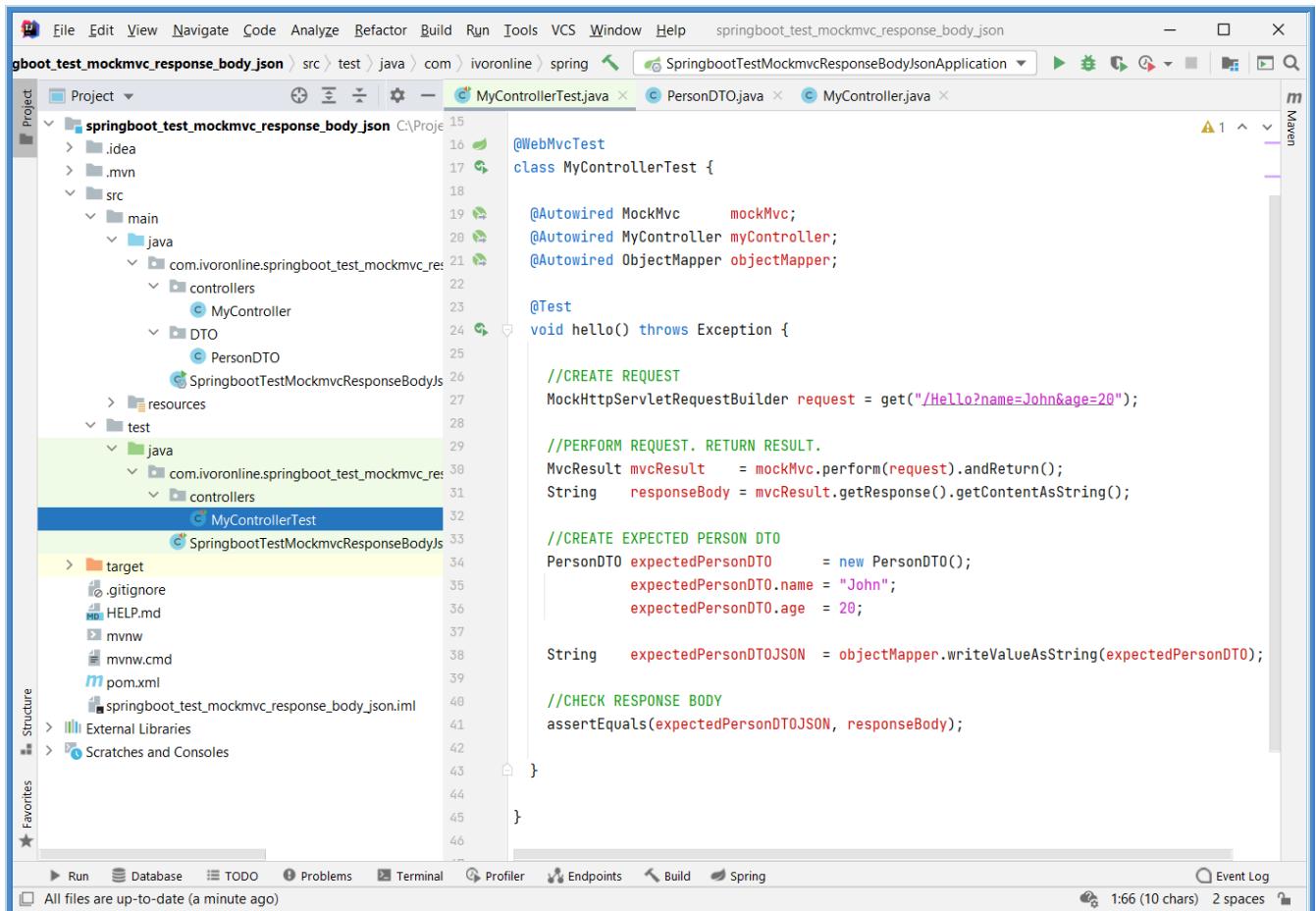


Wrong Response Body

get("/Hello?name=**John2**&age=20")

```
org.opentest4j.AssertionFailedError:  
Expected :{"name":"John","age":20}  
Actual   :{"name":"John2","age":20}
```

Run Test Class: MyControllerTest.java



pom.xml

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

5 Demo Applications

Info

- Following tutorials contain demo applications that show how to combine functionalities covered in previous tutorials.

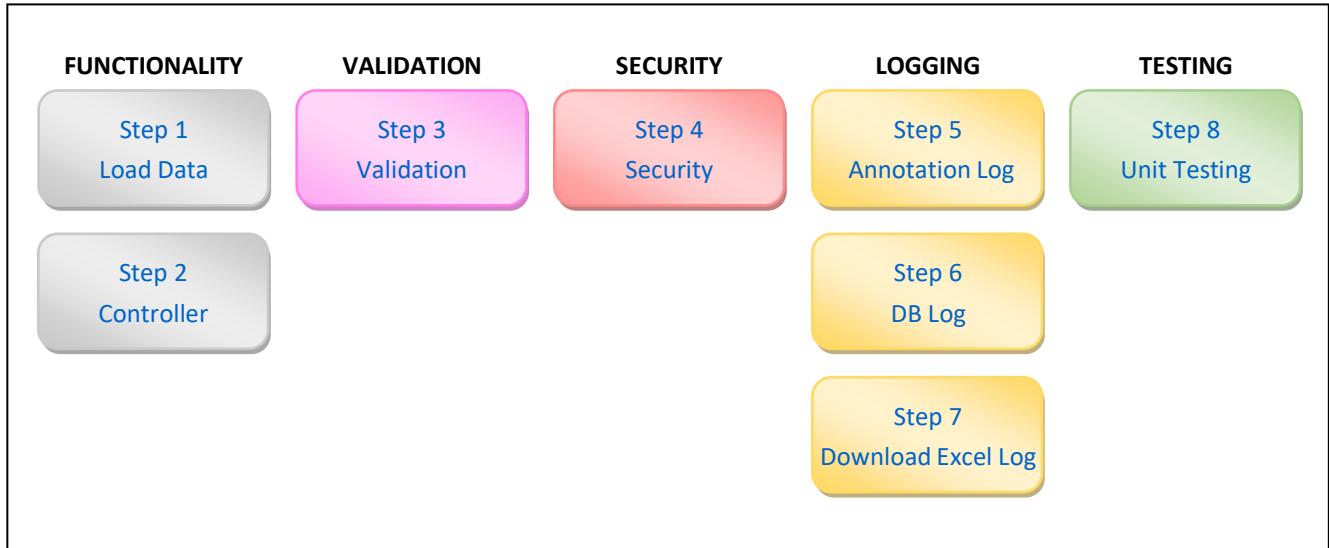
5.1 Currencies

Info

[G]

- Application loads currency exchange rates from HNB Remote Server using below URL into its own PostgreSQL DB
<http://api.hnb.hr/tecajn/v2?datum-primjene-od=2000-01-01&datum-primjene-do=2000-12-31>
- Then it provides its own Endpoints which allow you to request certain information about loaded currencies.
- To the Main Functionality we will add Validation, Logging and Unit Testing.
- Security is implemented in the most basic way just to show how to log which User accessed Application.

Application Steps



5.1.1 Step 1 - Load Data

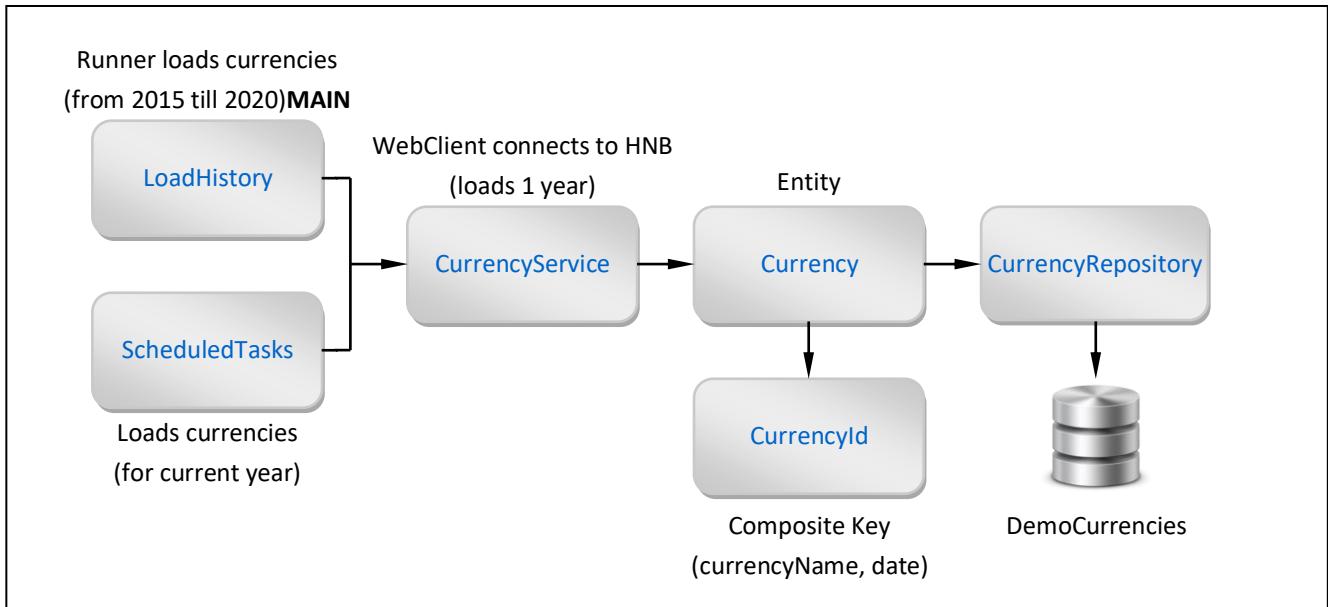
Info

[G]

- In the first step we will just load currencies into Database from the remote Server.
- Runner [LoadHistory](#) will be executed only once to load all the currencies until the current date.
- Scheduled Task [ScheduledTasks](#) will keep running to load new currencies every day.
- Both will be controlled from [application.properties](#).

Load Data

[\[Results\]](#)



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RequestMapping, Tomcat Server
Web	Spring Reactive Web	Enables: WebClient
SQL	Spring Data JPA	Enables: @Entity, @Id
SQL	PostgreSQL Database	Enables: Hibernate (to work with PostgreSQL DB)

Overview

- LoadHistory & ScheduledTasks call CurrencyService to load 1 year of exchange rates using below URL (year changes)
<http://api.hnb.hr/tecajn/v2?datum-primjene-od=2000-01-01&datum-primjene-do=2000-12-31>
- This returns Array of JSON Objects in the format shown below.
Each JSON Object is converted into Currency Entity and stored into DB.
- srednji_tecaj uses decimal comma 7,65 instead decimal point 7.65 and can't be stored directly in Double exchangeRate. Therefore CurrencyService
 - first loads srednji_teca into @Transient @JsonProperty("srednji_tecaj") public String exchangeRateString
 - and then replaces ',' with '.' and stores result in Currency's Double exchangeRate

Array of JSON Objects

```
[  
 {  
     "broj_tecajnjice" : "40",           //number  
     "datum_primjene"   : "2021-02-27",    //date  
     "drzava"          : "EMU",           //state  
     "drzava_iso"       : "EMU",           //stateISO  
     "sifra_valute"    : "978",           //currencyCode  
     "valuta"          : "EUR",            //currencyName  
     "jedinica"         : 1,               //units  
     "kupovni_tecaj"   : "7,559490",      //  
     "srednji_tecaj"   : "7,582237",      //exchangeRate  
     "prodajni_tecaj"  : "7,604984"  
 }  
 ...  
 ]
```

Class / Interface

NAME	DESCRIPTION
LoadHistory	Loads currencies from 2015 till 2020. Controlled by application.properties loadHistory = true .
ScheduledTasks	Loads currencies for current year. Controlled by application.properties loadCurrentYear = true .
CurrencyService	WebClient loads 1 year of currencies. Used by LoadHistory & ScheduledTasks.
Currency	Entity. @IdClass allows us to load Properties from JSON.
CurrencyId	Composite Primary Key for Currency from Properties: currencyName, date
CurrencyRepository	Contains JPQLs (for getting data from DB)

Procedure

- Create Project: [springboot_demo_currencies](#) (add Spring Boot Starters from the table)
- Edit File: [application.properties](#)
- Create Package: entities (inside main package)
 - Create Class: [CurrencyId.java](#) (inside package entities)
 - Create Class: [Currency.java](#) (inside package entities)
- Create Package: repositories (inside main package)
 - Create interface: [CurrencyRepository.java](#) (inside package repositories)
- Create Package: services (inside main package)
 - Create Class: [CurrencyService.java](#) (inside package controllers)
- Create Package: startuptasks (inside main package)
 - Create Class: [LoadHistory.java](#) (inside package startuptasks)
- Create Package: scheduledtasks (inside main package)
 - Create Class: [ScheduledTasks.java](#) (inside package scheduledtasks)

application.properties

```
# DATABASE: LOCAL
spring.datasource.url          = jdbc:postgresql://localhost:5432/DemoCurrencies
spring.datasource.username       = postgres
spring.datasource.password       = letmein
spring.datasource.driver-class-name = org.postgresql.Driver

# HIBERNATE
spring.jpa.hibernate.ddl-auto = update

# CUSTOM PROPERTIES
loadHistory                   = false
loadCurrentYear                = false
```

CurrencyId.java

```
package com.ivoronline.com.springboot_demo_currencies.business.entities;

import javax.persistence.Embeddable;
import java.io.Serializable;
import java.time.LocalDate;

@Embeddable
public class CurrencyId implements Serializable {

    //COMPOSITE PRIMARY KEY
    private String   currencyName;
    private LocalDate date;

    //REQUIRED NO ARGS CONSTRUCTOR
    public CurrencyId() { }

    //CONSTRUCTOR FOR findById(BookId)
    public CurrencyId(String currencyName, LocalDate date) {
        this.currencyName = currencyName;
        this.date         = date;
    }
}
```

Currency.java

```
package com.ivoronline.com.springboot_demo_currencies.business.entities;

import com.fasterxml.jackson.annotation.JsonProperty;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.IdClass;
import javax.persistence.Transient;
import java.time.LocalDate;

@Entity
@IdClass(CurrencyId.class)
public class Currency {

    //COMPOSITE PRIMARY KEY
    @Id @JsonProperty("valuta")      public String   currencyName;
    @Id @JsonProperty("datum_primjene") public LocalDate date;

    //STORED PROPERTIES
    @JsonProperty("broj_tecajnjice")   public Integer  number;
    @JsonProperty("drzava")           public String    state;
    @JsonProperty("drzava_iso")       public String    stateISO;
    @JsonProperty("sifra_valute")     public String    currencyCode;
    @JsonProperty("jedinica")         public Integer  units;
    @JsonProperty("jedinica")         public Double   exchangeRate; //Decimal point "7.65"

    //TRANSIENT PROPERTIES
    @Transient
    @JsonProperty("srednji_tecaj")    public String   exchangeRateString; //Decimal comma "7,65"
}

}
```

CurrencyRepository.java

```
package com.ivoronline.com.springboot_demo_currencies.business.repositories;

import com.ivoronline.com.springboot_demo_currencies.business.entities.Currency;
import com.ivoronline.com.springboot_demo_currencies.business.entities.CurrencyId;
import org.springframework.data.repository.CrudRepository;

public interface CurrencyRepository extends CrudRepository<Currency, CurrencyId> { }
```

CurrencyService.java

```
package com.ivorononline.com.springboot_demo_currencies.business.services;

import com.ivorononline.com.springboot_demo_currencies.business.entities.Currency;
import com.ivorononline.com.springboot_demo_currencies.business.repositories.CurrencyRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import java.time.Duration;
import java.util.List;

@Service
public class CurrencyService {

    @Autowired CurrencyRepository currencyRepository;

    //=====
    // STORE CURRENCIES
    //=====

    public void storeCurrencies(String url) {

        //LOG
        System.out.print(url + " ... ");

        //GET CURRENCIES FROM SERVER
        List<Currency> currencies = WebClient.create(url)
            .get()
            .retrieve()
            .bodyToFlux(Currency.class)
            .collectList()
            .block(Duration.ofMinutes(1));

        //REFORMAT EXCHANGE RATES
        for(Currency currency : currencies) {
            currency.exchangeRate = Double.parseDouble(currency.exchangeRateString.replace(",","."));
        }

        //STORE CURRENCIES
        currencyRepository.saveAll(currencies);

        //LOG
        System.out.println("DONE");
    }
}
```

LoadHistory.java

```
package com.ivorononline.com.springboot_demo_currencies.business.startup;

import com.ivorononline.com.springboot_demo_currencies.business.services.CurrencyService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class LoadHistory implements CommandLineRunner {

    @Autowired CurrencyService currencyService;

    @Value("${loadHistory}")
    private Boolean loadHistory;

    //=====
    // RUN
    //=====

    @Override
    public void run(String... args) throws Exception {

        //RETURN IF HISTORY SHOULD NOT BE LOADED
        if(!loadHistory) { return; }

        //LOG
        System.out.println("STARTED RUNNER: LoadHistory -----");

        //CONSTRUCT URL
        Integer sleep      = 10 * 1000;           //Sleep for 10 seconds
        Integer year       = 2015;                 //Start Date
        Date   currentDate = new Date();          //To calculate current Year
        Integer yearEnd    = 1900 + currentDate.getYear(); //Current Year
        while(year <= yearEnd) {
            String url = "http://api.hnb.hr/tecajn/v2?datum-primjene-od=" + year + "-01-01&datum-primjene-do=" +
year + "-12-31";
            currencyService.storeCurrencies(url);
            Thread.sleep(sleep);
            year++;
        }

        //LOG
        System.out.println("ENDED RUNNER: LoadHistory -----");
    }
}
```

ScheduledTasks.java

```
package com.ivorononline.com.springboot_demo_currencies.business.scheduled_tasks;

import com.ivorononline.com.springboot_demo_currencies.business.services.CurrencyService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;

import java.util.Date;

@Configuration
@EnableScheduling
public class ScheduledTasks {

    @Autowired CurrencyService currencyService;

    @Value("${loadCurrentYear}")
    private Boolean loadCurrentYear;

    //=====
    // LOAD CURRENT YEAR
    //=====

    @Scheduled(fixedDelay = 50000, initialDelay = 1000)
    public void loadCurrentYear() throws InterruptedException {

        //RETURN IF HISTORY SHOULD NOT BE LOADED
        if(!loadCurrentYear) { return; }

        //LOG
        System.out.println("STARTED TASK: loadCurrentYear() -----");

        //CONSTRUCT URL
        Date currentDate = new Date();
        Integer year      = 1900 + currentDate.getYear();
        String url       = "http://api.hnb.hr/tecajn/v2?datum-primjene-od=" + year + "-01-01&datum-primjene-
do=" + year + "-12-31";

        //GET CURRENCIES FOR CURRENT YEAR
        currencyService.storeCurrencies(url);

        //LOG
        System.out.println("ENDED TASK: loadCurrentYear() -----");
    }
}
```

Results

- Load History
 - Edit `application.properties`
 - `loadHistory = true`
 - `loadCurrentYear = false`
 - Start Application

Console

(`loadHistory = true`)

```
STARTED RUNNER: LoadHistory -----
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2015-01-01&datum-prmjene-do=2015-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2016-01-01&datum-prmjene-do=2016-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2017-01-01&datum-prmjene-do=2017-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2018-01-01&datum-prmjene-do=2018-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2019-01-01&datum-prmjene-do=2019-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2020-01-01&datum-prmjene-do=2020-12-31 ... DONE
ttp://api.hnb.hr/tecajn/v2?datum-prmjene-od=2021-01-01&datum-prmjene-do=2021-12-31 ... DONE
ENDED RUNNER: LoadHistory -----
```

- Load Current Year

- Edit `application.properties`
- `loadHistory = false`
- `loadCurrentYear = true`
- Start Application

Console

(`loadCurrentYear = true`)

```
STARTED TASK: loadCurrentYear() -----
2021 http://api.hnb.hr/tecajn/v2?datum-prmjene-od=2021-01-01&datum-prmjene-do=2021-12-31 ... DONE
ENDED TASK: loadCurrentYear() -----
```

Currency

(data loaded in DB Table)

	currency_name	date	currency_code	exchange_rate	number	state	stateiso	units
1	AUD	2021-01-01	036	4.750711	1	Australija	AUS	1
2	CAD	2021-01-01	124	4.821472	1	Kanada	CAN	1
3	CZK	2021-01-01	203	0.287345	1	Češka	CZE	1
4	DKK	2021-01-01	208	1.013713	1	Danska	DNK	1
5	HUF	2021-01-01	348	2.069425	1	Mađarska	HUN	100
6	JPY	2021-01-01	392	5.961764	1	Japan	JPN	100
7	NOK	2021-01-01	578	0.721385	1	Norveška	NOR	1
8	SEK	2021-01-01	752	0.753394	1	Švedska	SWE	1
9	CHF	2021-01-01	756	6.97128	1	Švicarska	CHE	1
10	GBP	2021-01-01	826	8.398918	1	Velika Britanija	GBR	1

Application Structure

The screenshot shows the IntelliJ IDEA interface with the project structure and code editor. The project structure on the left includes .idea, .mvn, and src folders. The src/main/java/com/ivoronline/com/springboot_demo_currencies/business/startup package contains LoadHistory.java, ScheduledTasks.java, CurrencyRepository.java, and application.properties. The code editor on the right displays LoadHistory.java:

```
10  @Component
11  public class LoadHistory implements CommandLineRunner {
12
13      @Autowired
14      CurrencyService currencyService;
15
16      @Value("true")
17      private Boolean loadHistory;
18
19      //=====
20      // RUN
21      //=====
22      @Override
23      public void run(String... args) throws Exception {
24
25          //RETURN IF HISTORY SHOULD NOT BE LOADED
26          if(!loadHistory) { return; }
27
28          //LOG
29          System.out.println("STARTED RUNNER: LoadHistory -----");
30
31          //CONSTRUCT URL
32          Integer sleep      = 10 * 1000;           //Sleep for 10 seconds
33          Integer year       = 2015;                 //Start Date
34
35          Data supportData = new Data();
36          supportData.setDate(sleep);
37          supportData.setYear(year);
38
39          //To calculate support Year
40
41      }
42
43  }
```

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-webflux</artifactId>
</dependency>

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

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>

</dependencies>
```

5.1.2 Step 2 - Controller

Info

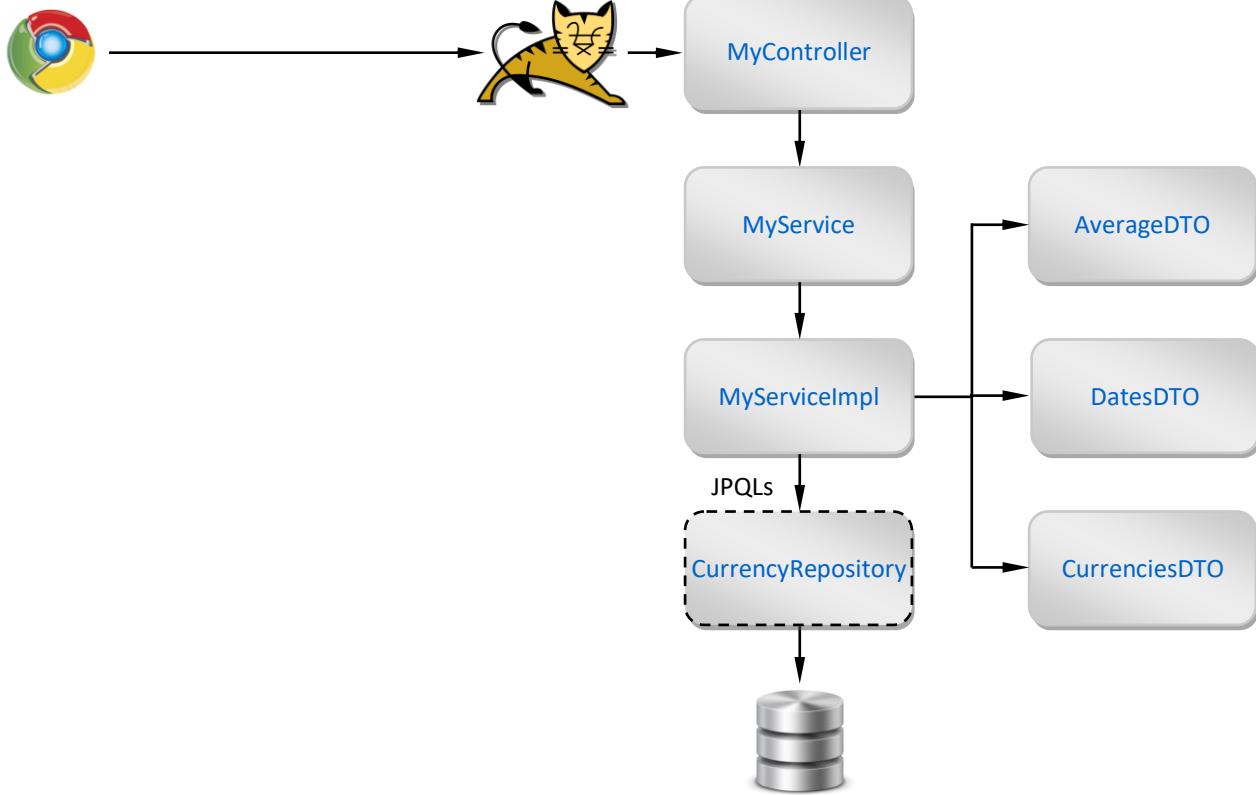
[G]

- In this step we are adding Controller with Endpoints that can give us some information about stored currencies.

Application Structure

[Results]

<http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2000-05-30&endDate=2010-05-30>
<http://localhost:8080/GetFirstLastDate?currencyName=EUR>
<http://localhost:8080>SelectAllCurrencyNames>



Procedure

- Create Package: DTO (inside main package)
- Create Class: CurrenciesDTO.java (inside DTO controllers)
- Create Class: AverageDTO.java (inside DTO controllers)
- Create Class: DatesDTO.java (inside DTO controllers)
- Inside Package: repositories
- Edit Class: CurrencyRepository.java
- Inside Package: services
- Create Interface: MyService.java (inside package services)
- Create Class: MyServiceImpl.java (inside package services)
- Create Package: controllers (inside main package)
- Create Class: MyController.java (inside package controllers)

CurrenciesDTO.java

```
package com.ivoronline.springboot_demo_currencies.business.DTO;

import java.util.List;

public class CurrenciesDTO {
    public List<String> allCurrencyNames;
}
```

AverageDTO.java

```
package com.ivoronline.springboot_demo_currencies.business.DTO;

public class AverageDTO {
    public Float average;
}
```

DatesDTO.java

```
package com.ivoronline.springboot_demo_currencies.business.DTO;

import java.time.LocalDate;

public class DatesDTO {
    public LocalDate firstDate;
    public LocalDate lastDate;
}
```

CurrencyRepository.java

(add JPQLs)

```
package com.ivoronline.com.springboot_demo_currencies.business.repositories;

import com.ivoronline.com.springboot_demo_currencies.business.entities.Currency;
import com.ivoronline.com.springboot_demo_currencies.business.entities.CurrencyId;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import java.time.LocalDate;
import java.util.List;

public interface CurrencyRepository extends CrudRepository<Currency, CurrencyId> {

    //GET ALL CURRENCY NAMES
    @Query(nativeQuery = true, value = "SELECT DISTINCT CURRENCY_NAME FROM CURRENCY")
    List<String> getAllCurrencyNames();

    //GET FIRST ENTRY
    @Query(nativeQuery = true, value = "SELECT MIN(date) FROM currency WHERE currency_name = :currencyName")
    LocalDate getFirstDate(String currencyName);

    //GET LAST ENTRY
    @Query(nativeQuery = true, value = "SELECT MAX(date) FROM currency WHERE currency_name = :currencyName")
    LocalDate getLastDate(String currencyName);

    //GET AVERAGE VALUE
    @Query(nativeQuery = true, value = "SELECT AVG(exchange_rate) FROM currency WHERE currency_name = :currencyName AND date >= :startDate AND date <= :endDate")
    Float getAverageValue(String currencyName, LocalDate startDate, LocalDate endDate);

}
```

MyService.java

```
package com.ivorononline.com.springboot_demo_currencies.business.services;

import com.ivorononline.com.springboot_demo_currencies.business.dto.AverageDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.CurrenciesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.DatesDTOResponse;
import org.springframework.stereotype.Component;

@Component
public interface MyService {
    CurrenciesDTOResponse getAllCurrencyNames();
    DatesDTOResponse getFirstLastDate(String currencyName);
    AverageDTOResponse getAverageValue(String currencyName, String startDate, String endDate);
}
```

MyServiceImpl.java

```
package com.ivorononline.com.springboot_demo_currencies.business.services;

import com.ivorononline.com.springboot_demo_currencies.business.dto.AverageDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.CurrenciesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.DatesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.repositories.CurrencyRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.time.LocalDate;
import java.util.List;

@Service
public class MyServiceImpl implements MyService {

    @Autowired CurrencyRepository currencyRepository;

    //=====
    // GET ALL CURRENCY NAMES
    //=====

    @Override
    public CurrenciesDTOResponse getAllCurrencyNames() {

        //GET CURRENCY NAMES
        List<String> allCurrencyNames = currencyRepository.getAllCurrencyNames();

        //CREATE DTO
        CurrenciesDTOResponse currenciesDTOResponse = new CurrenciesDTOResponse();
        currenciesDTOResponse.allCurrencyNames = allCurrencyNames;

        //RETURN DTO
        return currenciesDTOResponse;
    }

    //=====
    // GET FIRST LAST DATE
    //=====

    @Override
    public DatesDTOResponse getFirstLastDate(String currencyName) {

        //GET DATES
        LocalDate firstDate = currencyRepository.getFirstDate(currencyName);
        LocalDate lastDate = currencyRepository.getLastDate(currencyName);

        //CREATE DTO
        DatesDTOResponse datesDTOResponse = new DatesDTOResponse();
    }
}
```

```

        datesDTOResponse.firstDate = firstDate;
        datesDTOResponse.lastDate = lastDate;

    }

//=====
// GET AVERAGE VALUE
//=====

@Override
public AverageDTOResponse getAverageValue(String currencyName, String startDate, String endDate) {

    //CONVERT DATES
    LocalDate startDateConverted = LocalDate.parse(startDate);
    LocalDate endDateConverted = LocalDate.parse(endDate);

    //GET AVERAGE VALUE
    Float average = currencyRepository.getAverageValue(currencyName, startDateConverted, endDateConverted);

    //CREATE DTO
    AverageDTOResponse averageDTOResponse = new AverageDTOResponse();
    averageDTOResponse.average = average;

    //RETURN DTO
    return averageDTOResponse;
}

}

```

MyController.java

```
package com.ivorononline.com.springboot_demo_currencies.business.controllers;

import com.ivorononline.com.springboot_demo_currencies.business.dto.AverageDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.CurrenciesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.DatesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.services.MyService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@Controller
public class MyController {

    @Autowired
    MyService myService;

    //=====
    // GET ALL CURRENCY NAMES
    //=====

    @ResponseBody
    @GetMapping("/GetAllCurrencyNames")
    public CurrenciesDTOResponse getAllCurrencyNames() {
        return myService.getAllCurrencyNames();
    }

    //=====
    // GET FIRST LAST DATE
    //=====

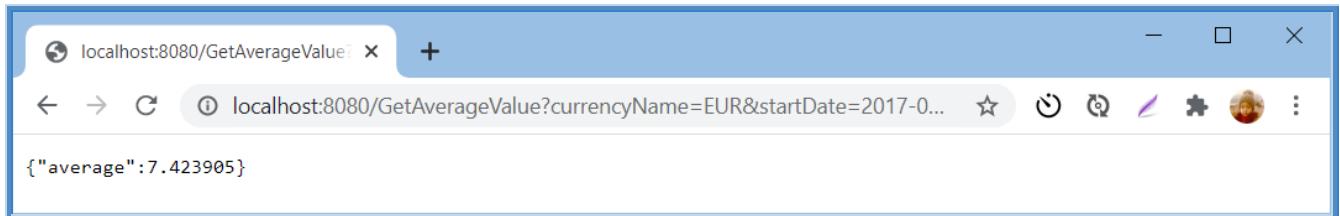
    @ResponseBody
    @GetMapping("/GetFirstLastDate")
    public DatesDTOResponse getFirstLastDate(@RequestParam String currencyName) {
        return myService.getFirstLastDate(currencyName);
    }

    //=====
    // GET AVERAGE VALUE
    //=====

    @ResponseBody
    @GetMapping("/GetAverageValue")
    public AverageDTOResponse getAverageValue(
        @RequestParam String currencyName,
        @RequestParam String startDate,
        @RequestParam String endDate
    ) {
        return myService.getAverageValue(currencyName, startDate, endDate);
    }
}
```

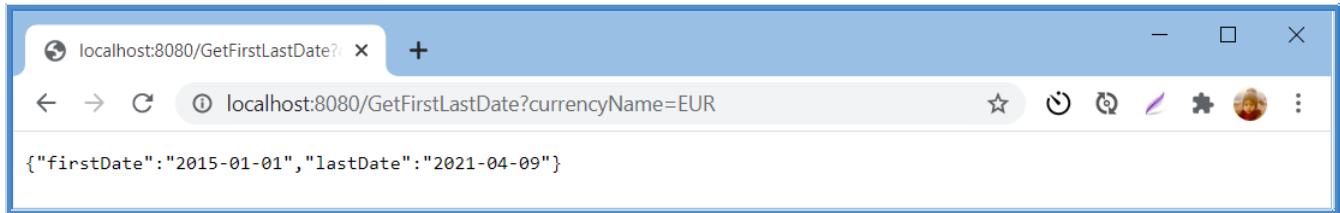
Results

<http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2017-05-30&endDate=2017-05-30>



The screenshot shows a browser window with the URL localhost:8080/GetAverageValue?currencyName=EUR&startDate=2017-05-30&endDate=2017-05-30. The response body contains the JSON object: {"average":7.423905}

<http://localhost:8080/GetFirstLastDate?currencyName=EUR>



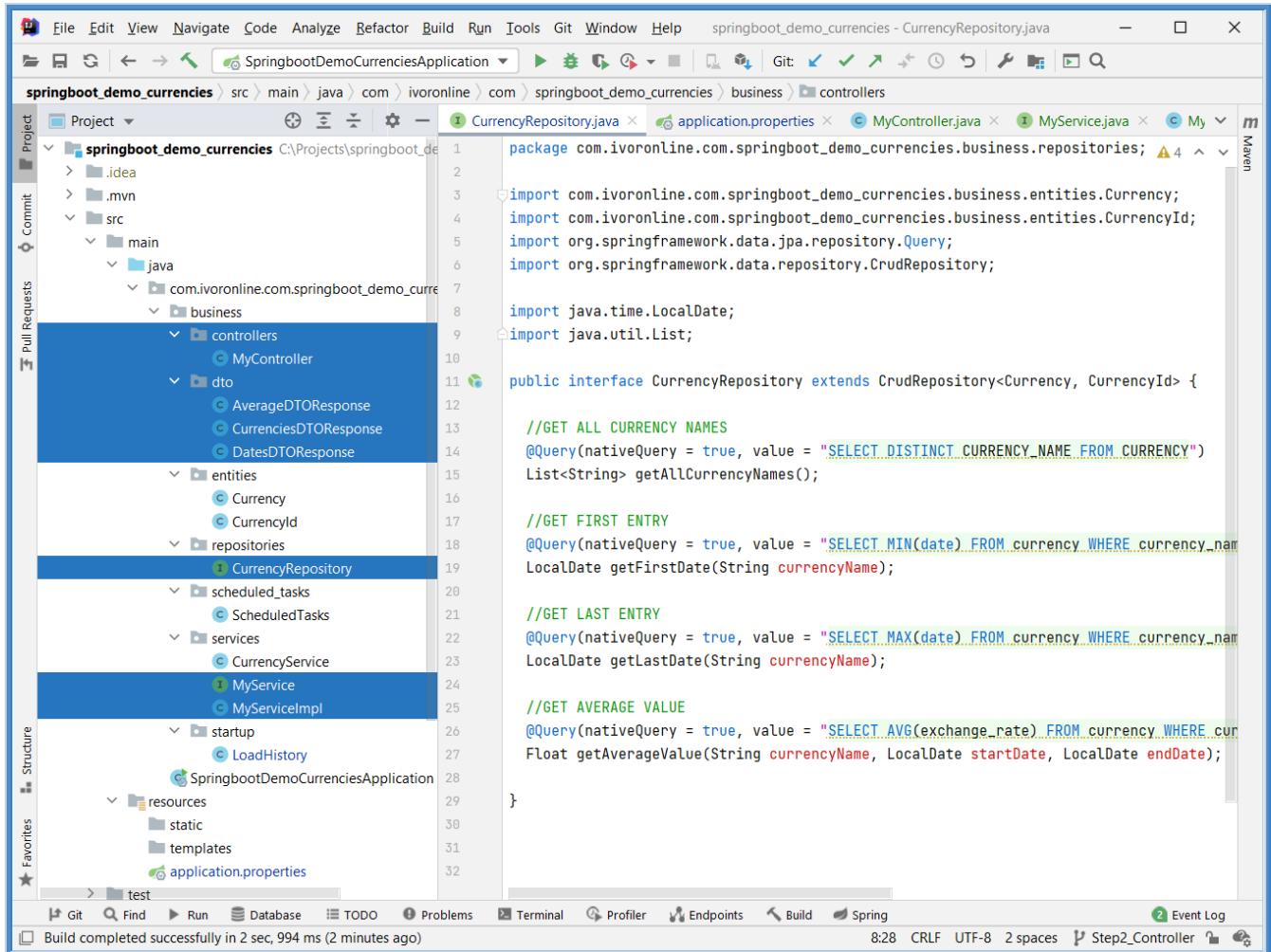
The screenshot shows a browser window with the URL localhost:8080/GetFirstLastDate?currencyName=EUR. The response body contains the JSON object: {"firstDate": "2015-01-01", "lastDate": "2021-04-09"}

<http://localhost:8080/GetAllCurrencyNames>



The screenshot shows a browser window with the URL localhost:8080/GetAllCurrencyNames. The response body contains the JSON object: {"allCurrencyNames": ["BAM", "SEK", "NOK", "JPY", "AUD", "GBP", "CHF", "HUF", "XDR", "CZK", "CAD", "DKK", "EUR", "USD", "PLN"]}

Application Structure



The screenshot shows the IntelliJ IDEA interface with the project `springboot_demo_currencies` open. The code editor displays the `CurrencyRepository.java` file, which contains the following code:

```
package com.ivoronline.com.springboot_demo_currencies.business.repositories;

import com.ivoronline.com.springboot_demo_currencies.business.entities.Currency;
import com.ivoronline.com.springboot_demo_currencies.business.entities.CurrencyId;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import java.time.LocalDate;
import java.util.List;

public interface CurrencyRepository extends CrudRepository<Currency, CurrencyId> {

    //GET ALL CURRENCY NAMES
    @Query(nativeQuery = true, value = "SELECT DISTINCT CURRENCY_NAME FROM CURRENCY")
    List<String> getAllCurrencyNames();

    //GET FIRST ENTRY
    @Query(nativeQuery = true, value = "SELECT MIN(date) FROM currency WHERE currency_name = :name")
    LocalDate getFirstDate(String currencyName);

    //GET LAST ENTRY
    @Query(nativeQuery = true, value = "SELECT MAX(date) FROM currency WHERE currency_name = :name")
    LocalDate getLastDate(String currencyName);

    //GET AVERAGE VALUE
    @Query(nativeQuery = true, value = "SELECT AVG(exchange_rate) FROM currency WHERE currency_name = :name")
    Float getAverageValue(String currencyName, LocalDate startDate, LocalDate endDate);
}
```

5.1.3 Step 3 - Validation

Info

[G]

- In this step we are adding Validation to Controller Endpoints.
- User will get error message if Endpoints are not called with all required parameters.

Application Structure

[Results]

```
http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2000-05-30&endDate=2010-05-30  
http://localhost:8080/GetFirstLastDate?currencyName=EUR  
http://localhost:8080>SelectAllCurrencyNames
```



MyController

Procedure

- >Edit File: pom.xml (add validation dependency)
- Edit Class: MyController.java

pom.xml

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

MyController.java

```
package com.ivoronline.com.springboot_demo_currencies.business.controllers;

import com.ivoronline.com.springboot_demo_currencies.business.dto.AverageDTOResponse;
import com.ivoronline.com.springboot_demo_currencies.business.dto.CurrenciesDTOResponse;
import com.ivoronline.com.springboot_demo_currencies.business.dto.DatesDTOResponse;
import com.ivoronline.com.springboot_demo_currencies.business.services.MyService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

import javax.validation.constraints.NotBlank;

@Controller
public class MyController {

    @Autowired MyService myService;

    //=====
    // GET ALL CURRENCY NAMES
    //=====

    @ResponseBody
    @RequestMapping("/GetAllCurrencyNames")
    public CurrenciesDTOResponse getAllCurrencyNames() {
        return myService.getAllCurrencyNames();
    }

    //=====
    // GET FIRST LAST DATE
    //=====

    @ResponseBody
    @RequestMapping("/GetFirstLastDate")
    public DatesDTOResponse getFirstLastDate(@RequestParam @NotBlank String currencyName) {
        return myService.getFirstLastDate(currencyName);
    }

    //=====
    // GET AVERAGE VALUE
    //=====

    @ResponseBody
    @RequestMapping("/GetAverageValue")
    public AverageDTOResponse getAverageValue(
        @RequestParam @NotBlank String currencyName,
        @RequestParam @NotBlank String startDate,
        @RequestParam @NotBlank String endDate
    ) {
```

```
        return myService.getAverageValue(currencyName, startDate, endDate);
    }

//=====
// HANDLE EXCEPTIONS (it only catches first exception)
//=====

@ResponseBody
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(MissingServletRequestParameterException.class)
public String handleExceptions(MissingServletRequestParameterException exception) {

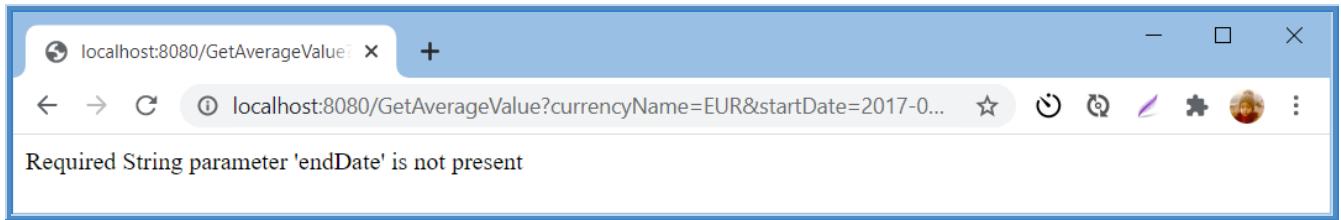
    //GET EXCEPTION DETAILS
    String parameterType = exception.getParameterType(); //String
    String parameterName = exception.getParameterName(); //name
    String message      = exception.getMessage();          //Required String parameter 'name' is not present

    //RETURN MESSAGE
    return message;
}

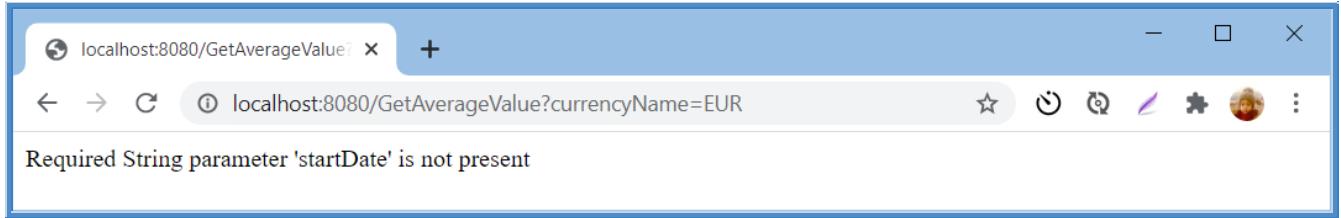
}
```

Results

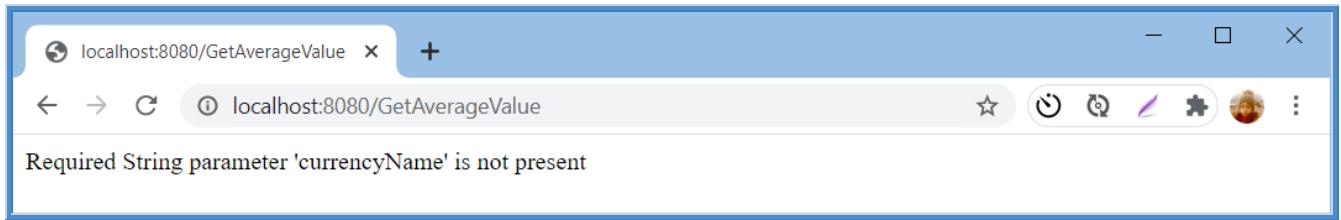
<http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2017-05-30>



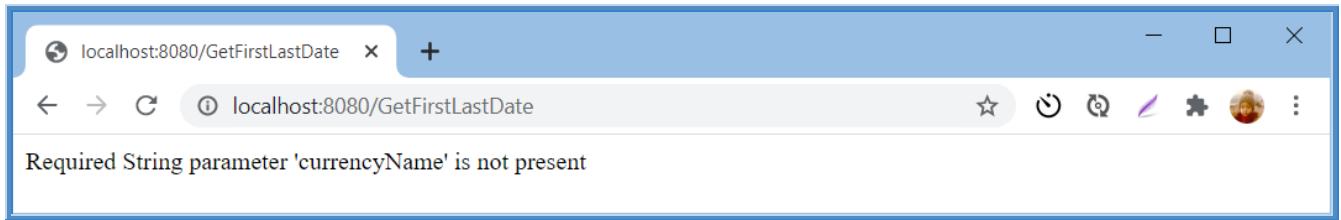
<http://localhost:8080/GetAverageValue?currencyName=EUR>



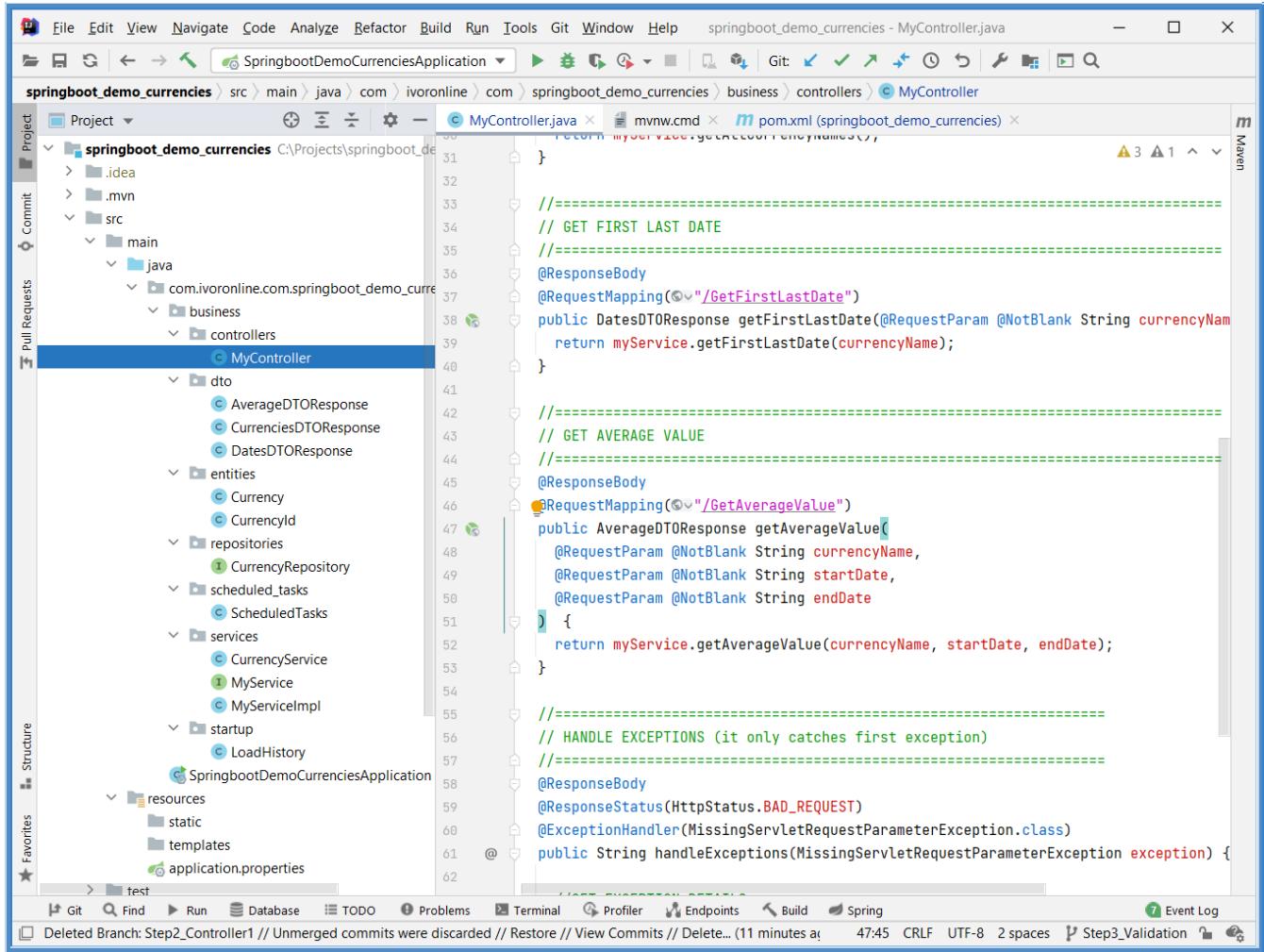
<http://localhost:8080/GetAverageValue>



<http://localhost:8080/GetFirstLastDate>



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-webflux</artifactId>
    </dependency>

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

</dependencies>
```

5.1.4 Step 4 - Security

Info

[G]

- In this step we will add basic Security for the purpose of showing Security related Logging and Testing in following steps.
- We will do this by adding a single user to [application.properties](#).

Application Structure

[Results]



application.properties

Procedure

- Edit File: [pom.xml](#) (add security dependency)
- Edit File: [application.properties](#) (add User Account)

pom.xml

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

application.properties

```
# DATABASE: LOCAL
spring.datasource.url      = jdbc:postgresql://localhost:5432/DemoCurrencies
spring.datasource.username  = postgres
spring.datasource.password = letmein
spring.datasource.driver-class-name = org.postgresql.Driver

# HIBERNATE
spring.jpa.hibernate.ddl-auto = update

# CUSTOM PROPERTIES
loadHistory                 = false
loadCurrentYear              = true

# SECURITY
spring.security.user.name   = myuser
spring.security.user.password = mypassword
spring.security.user.roles   = USER
```

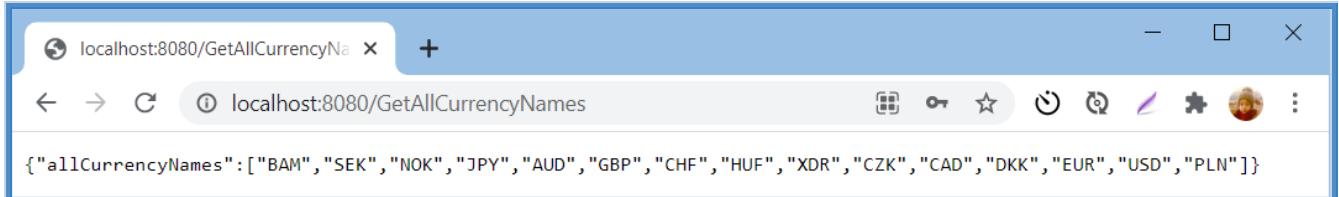
Results

- <http://localhost:8080/GetAllCurrencyNames>
- Username: myuser
- Password: mypassword
- Sign in

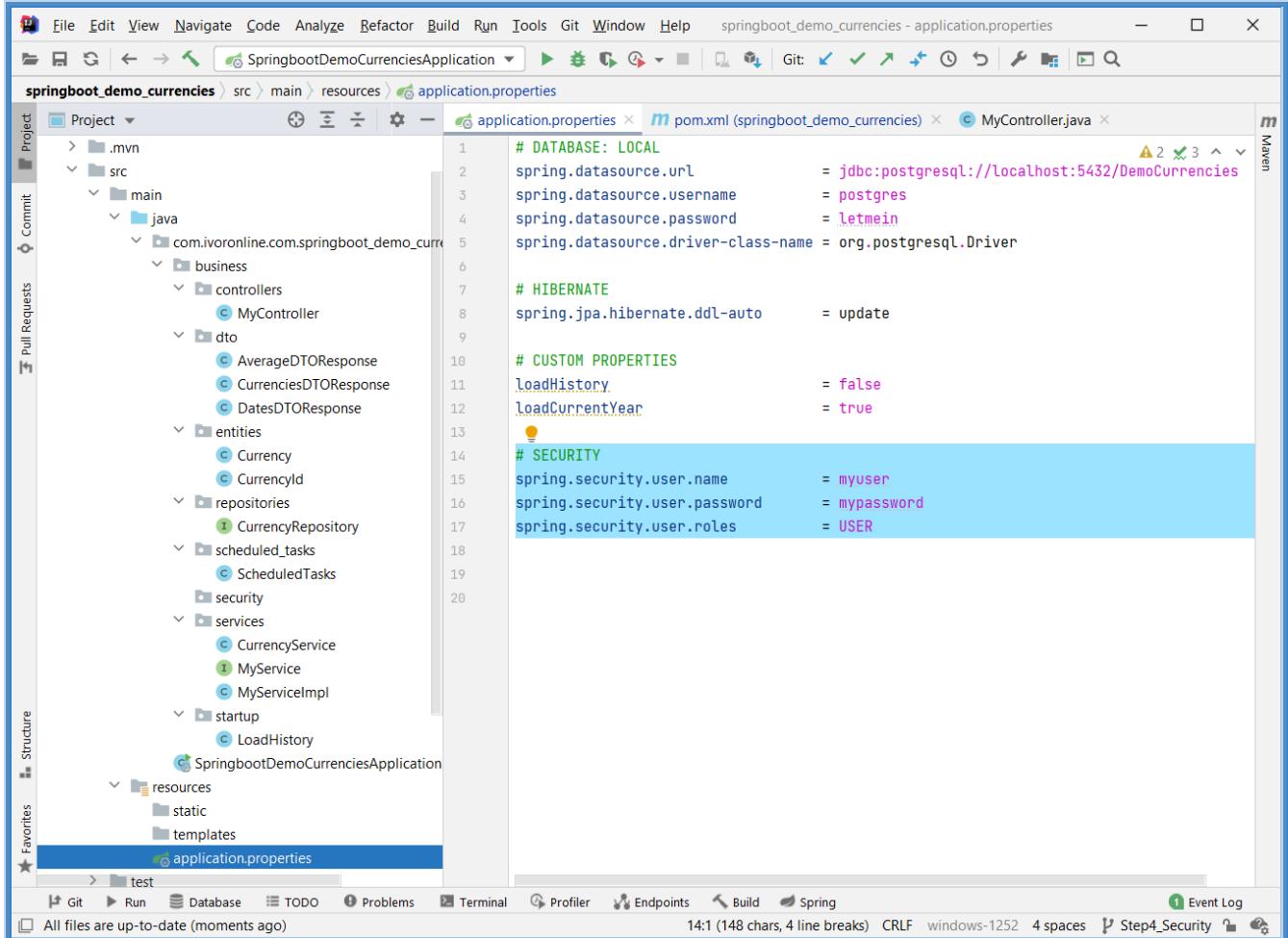
<http://localhost:8080/GetAllCurrencyNames> - myuser - mypassword



<http://localhost:8080/GetAllCurrencyNames>



Application Structure



```
<dependencies>

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

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

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

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

</dependencies>
```

5.1.5 Step 5 - Logging with Annotation into File

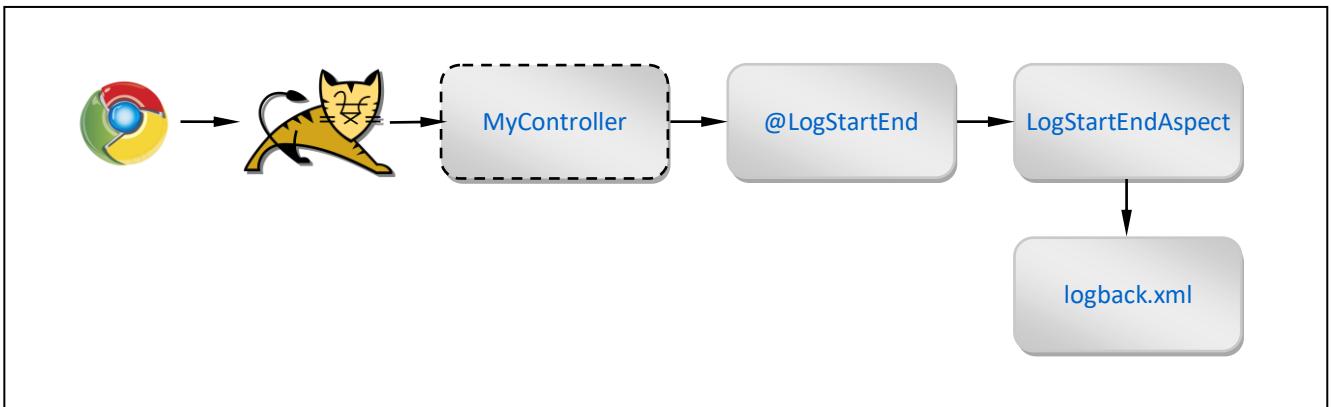
Info

[G]

- In this step we will create `@LogStartEnd` Annotation to log **start** and **end** of Controller Methods.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- **Create Project:** `springboot_filter_create` (add Spring Boot Starters from the table)
- **Edit File:** `pom.xml` (add Lombok dependency to support @Slf4j)
- **Create File:** `logback.xml` (in directory resources)
- **Create Package:** `loggerAOP` (inside main package)
 - **Create @interface:** `LogStartEnd.java` (inside loggerAOP package)
 - **Create Class:** `LogStartEndAspect.java` (inside loggerAOP package)
- **Edit Class:** `MyController.java` (add @LogStartEnd Annotations before Endpoint)

pom.xml

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- FILE APPENDER -->
    <appender name="MyFileAppenderName" class="ch.qos.logback.core.FileAppender">

        <!-- FILE NAME -->
        <file>logs/File.log</file>

        <!-- ENCODER -->
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>My File Appender: %d %p %c{30} %m %n</Pattern>
        </encoder>

    </appender>

    <!-- ROLLING FILE APPENDER -->
    <appender name="MyRollingFileAppenderName" class="ch.qos.logback.core.rolling.RollingFileAppender">

        <!-- FILE NAME -->
        <file>logs/RollingFile.log</file>

        <!-- ENCODER -->
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>My File Appender: %d %p %c{30} %m %n</Pattern>
        </encoder>

        <!-- ROLLOVER POLICY (daily and when the file reaches 1MB) -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>logs/archived/RollingFile_%d{dd.MM.yyyy}_%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>10MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>

    </appender>

    <!-- LOGGER -->
    <logger name="com.ivorononline" level="INFO">
        <appender-ref ref="MyFileAppenderName" />
        <appender-ref ref="MyRollingFileAppenderName"/>
    </logger>
</configuration>
```

LogStartEnd.java

```
package com.ivoronline.springboot_demo_currencies.loggerAOP;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface LogStartEnd { }
```

LogStartEndAspect.java

```
package com.ivoronline.springboot_demo_currencies.loggerAOP;

import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Slf4j
@Aspect
@Component
public class LogStartEndAspect {

    @Around("@annotation(LogStartEnd)")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

        //GET METHOD PARAMETERS
        String methodName = joinPoint.getSignature().getName(); //hello
        String className = joinPoint.getSignature().getDeclaringType().getSimpleName(); //MyController

        //EXECUTE BEFORE METHOD
        log.info("STARTED METHOD: " + className + "." + methodName + "()");

        //EXECUTE METHOD
        Object proceed = joinPoint.proceed();

        //EXECUTE AFTER METHOD
        log.info("ENDED METHOD: " + className + "." + methodName + "()");

        //RETURN METHOD RESULT
        return proceed;
    }
}
```

MyController.java

```
package com.ivorononline.com.springboot_demo_currencies.business.controllers;

import com.ivorononline.com.springboot_demo_currencies.business.dto.AverageDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.CurrenciesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.dto.DatesDTOResponse;
import com.ivorononline.com.springboot_demo_currencies.business.services.MyService;
import com.ivorononline.com.springboot_demo_currencies.loggerAOP.LogStartEnd;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotationResponseStatus;

import javax.validation.constraints.NotBlank;

@Controller
public class MyController {

    @Autowired MyService myService;

    //=====
    // GET ALL CURRENCY NAMES
    //=====

    @LogStartEnd
    @ResponseBody
    @RequestMapping("/GetAllCurrencyNames")
    public CurrenciesDTOResponse getAllCurrencyNames() {
        return myService.getAllCurrencyNames();
    }

    //=====
    // GET FIRST LAST DATE
    //=====

    @LogStartEnd
    @ResponseBody
    @RequestMapping("/GetFirstLastDate")
    public DatesDTOResponse getFirstLastDate(@RequestParam @NotBlank String currencyName) {
        return myService.getFirstLastDate(currencyName);
    }

    //=====
    // GET AVERAGE VALUE
    //=====

    @LogStartEnd
    @ResponseBody
    @RequestMapping("/GetAverageValue")
    public AverageDTOResponse getAverageValue(
        @RequestParam @NotBlank String currencyName,
        @RequestParam @NotBlank String startDate,
        @RequestParam @NotBlank String endDate
    ) {
        return myService.getAverageValue(currencyName, startDate, endDate);
    }

    //=====
    // HANDLE EXCEPTIONS (it only catches first exception)
    //=====

    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
```

```
@ExceptionHandler(MissingServletRequestParameterException.class)
public String handleExceptions(MissingServletRequestParameterException exception) {

    //GET EXCEPTION DETAILS
    String parameterType = exception.getParameterType(); //String
    String parameterName = exception.getParameterName(); //name
    String message      = exception.getMessage();           //Required String parameter 'name' is not present

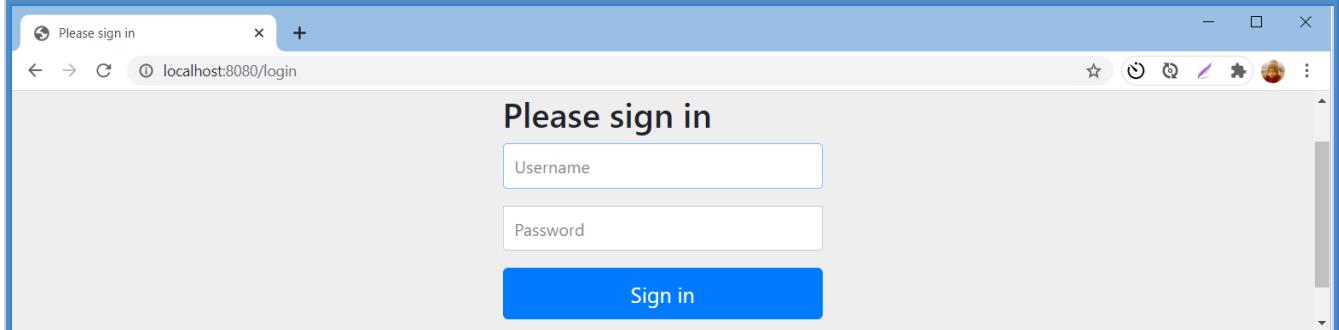
    //RETURN MESSAGE
    return message;

}
```

Results

- <http://localhost:8080/GetAllCurrencyNames>
- Username: myuser
- Password: mypassword
- Sign in

<http://localhost:8080/GetAllCurrencyNames> - myuser - mypassword



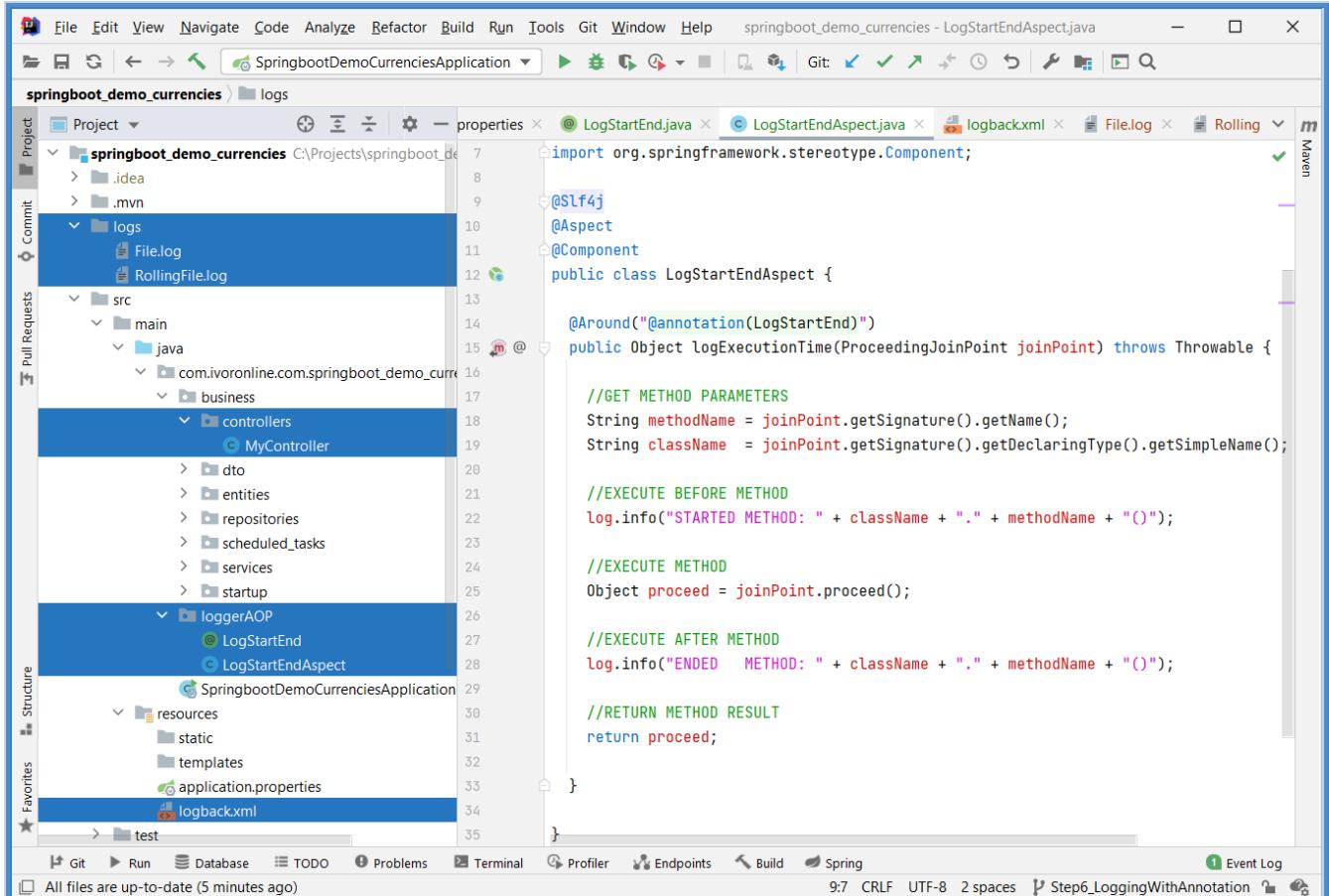
<http://localhost:8080/GetAllCurrencyNames>



File.log

```
My File Appender: 2021-04-09 09:56:49,700 INFO c.i.c.s.l.LogStartEndAspect STARTED METHOD: MyController.getAllCurrencyNames()
My File Appender: 2021-04-09 09:56:49,749 INFO c.i.c.s.l.LogStartEndAspect ENDED METHOD: MyController.getAllCurrencyNames()
```

Application Structure



```
<dependencies>

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

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

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

</dependencies>
```

5.1.6 Step 6 - Logging with Filter into DB

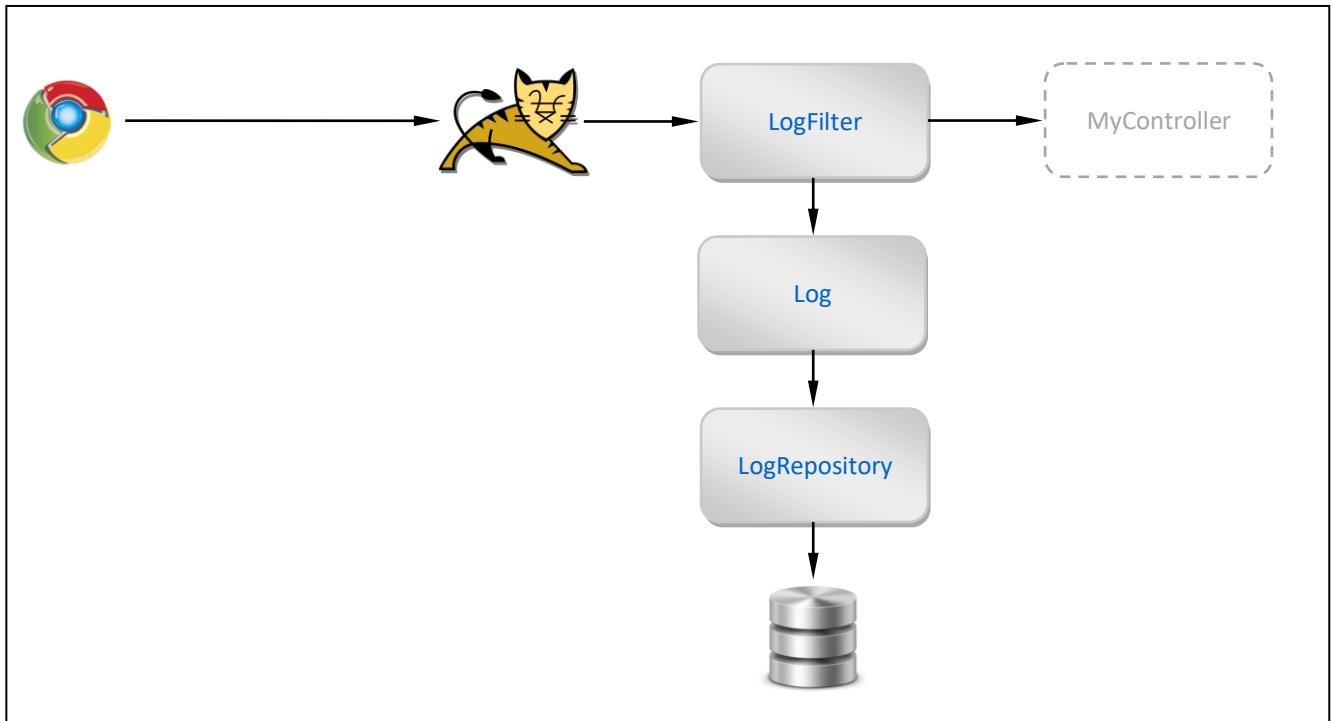
Info

[G]

- In this step we will use Filter to intercept HTTP Requests and log them into DB.
- Logging will be done manually (with custom Entity and Repository) rather than using Slf4j as in previous step.

Application Schema

[Results]



Procedure

- Create Project: `springboot_filter_create` (add Spring Boot Starters from the table)
- Create Package: `loggerDB` (inside main package)
- Create Package: `entities` (inside loggerDB package)
 - Create Class: `Log.java` (Entity)
- Create Package: `repositories` (inside loggerDB package)
 - Create Class: `LogRepository.java`
- Create Package: `filters` (inside loggerDB package)
 - Create Class: `LogFilter.java`

Log.java

```
package com.ivoronline.com.springboot_demo_currencies.loggerDB.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.time.LocalDate;

@Entity
public class Log {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String protocol;
    public String serverName;
    public String method;
    public String servletPath;
    public String queryString;
    public Integer serverPort;
    public String username;
    public Integer status;
    public LocalDate date;

}
```

LogRepository.java

```
package com.ivoronline.com.springboot_demo_currencies.loggerDB.repositories;

import com.ivoronline.com.springboot_demo_currencies.loggerDB.entities.Log;
import org.springframework.data.repository.CrudRepository;

public interface LogRepository extends CrudRepository<Log, Integer> { }
```

```

package com.ivorononline.com.springboot_demo_currencies.loggerDB.filters;

import com.ivorononline.com.springboot_demo_currencies.loggerDB.entities.Log;
import com.ivorononline.com.springboot_demo_currencies.loggerDB.repositories.LogRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.context.SecurityContextHolder;
import org.springframework.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.LocalDate;

@Component
public class LogFilter extends OncePerRequestFilter {

    @Autowired
    LogRepository logRepository;

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        //DIVIDES HTTP REQUEST AND RESPONSE CODE
        chain.doFilter(request, response);

        //GET USERNAME
        String username;
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) { username = ((UserDetails)principal).getUsername(); }
        else { username = principal.toString(); }

        //CREATE LOG
        Log log = new Log();
        log.username = username;
        log.date = LocalDate.now();
        log.protocol = request.getProtocol();
        log.serverName = request.getServerName();
        log.serverPort = request.getServerPort();
        log.servletPath = request.getServletPath();
        log.queryString = request.getQueryString();
        log.method = request.getMethod();
        log.status = response.getStatus();

        //STORE LOG
        logRepository.save(log);
    }
}

```

Results

<http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2017-05-30&endDate=2017-05-30>

The screenshot shows a browser window with the URL <http://localhost:8080/GetAverageValue?currencyName=EUR&startDate=2017-05-30&endDate=2017-05-30>. The response body contains the JSON object: {"average":7.423905}.

<http://localhost:8080/GetFirstLastDate?currencyName=EUR>

The screenshot shows a browser window with the URL <http://localhost:8080/GetFirstLastDate?currencyName=EUR>. The response body contains the JSON object: {"firstDate": "2015-01-01", "lastDate": "2021-04-09"}.

<http://localhost:8080/GetAllCurrencyNames>

The screenshot shows a browser window with the URL <http://localhost:8080/GetAllCurrencyNames>. The response body contains the JSON object: {"allCurrencyNames": ["BAM", "SEK", "NOK", "JPY", "AUD", "GBP", "CHF", "HUF", "XDR", "CZK", "CAD", "DKK", "EUR", "USD", "PLN"]}.

Log

(Database)

	<code>id</code>	<code>date</code>	<code>method</code>	<code>protocol</code>	<code>query_string</code>	<code>server_name</code>	<code>server_port</code>	<code>servlet_path</code>	<code>status</code>	<code>username</code>
1	2	2021-04-09	GET	HTTP/1.1	<null>	localhost	8080	/GetAllCurrencyNames	200	myuser
2	3	2021-04-09	GET	HTTP/1.1	currencyName=EUR	localhost	8080	/GetFirstLastDate	200	myuser
3	4	2021-04-09	GET	HTTP/1.1	currencyName=EUR&st..	localhost	8080	/GetAverageValue	200	myuser

Application Structure

The screenshot shows an IDE interface with the project `springboot_demo_currencies` open. The code editor displays `LogFilter.java` with the following content:

```
@Override
public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
throws IOException, ServletException {
    //DIVIDES HTTP REQUEST AND RESPONSE CODE
    chain.doFilter(request, response);

    //GET USERNAME
    String username;
    Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    if (principal instanceof UserDetails) { username = ((UserDetails)principal).getUsername(); }
    else { username = principal.toString(); }

    //CREATE LOG
    Log log = new Log();
    log.username = username;
    log.date = LocalDate.now();
    log.protocol = request.getProtocol();
    log.serverName = request.getServerName();
    log.serverPort = request.getServerPort();
    log.servletPath = request.getServletPath();
    log.queryString = request.getQueryString();
    log.method = request.getMethod();
    log.status = response.getStatus();

    //STORE LOG
    logRepository.save(log);
}
```

```
<dependencies>

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

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

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

</dependencies>
```

5.1.7 Step 7 - Download Excel with DB Log

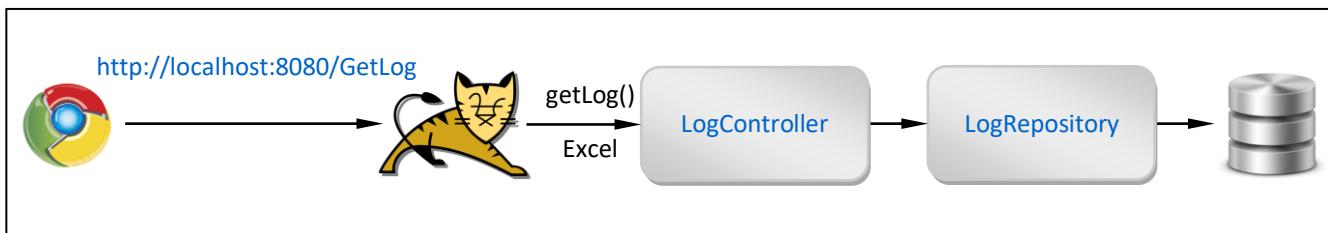
Info

[G]

- In this step we will add [LogController](#) with Endpoint which downloads Excel with entries from DB Log Table.

Application Schema

[Results]



Procedure

- Create Project:** [springboot_filter_create](#) (add Spring Boot Starters from the table)
- Edit File:** [pom.xml](#) (add Excel dependency)
- Edit Class:** [LogRepository.java](#) (add JPQ to fetch DB Log data)
- Create Package:** controllers (inside loggerDB package)
 - Create Class:** [LogController.java](#) (Entity)

pom.xml

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>3.15</version>
</dependency>
```

LogRepository.java

```
package com.ivoronline.com.springboot_demo_currencies.loggerDB.repositories;

import com.ivoronline.com.springboot_demo_currencies.loggerDB.entities.Log;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import java.time.LocalDate;
import java.util.List;

public interface LogRepository extends CrudRepository<Log, Integer> {

    //GET LOG
    @Query(value = "SELECT log FROM Log log WHERE log.username = :username AND log.date >= :startDate AND log.date <= :endDate")
    List<Log> getLog(String username, LocalDate startDate, LocalDate endDate);

}
```

LogController.java

```
package com.ivorononline.com.springboot_demo_currencies.loggerDB.controllers;

import com.ivorononline.com.springboot_demo_currencies.loggerDB.entities.Log;
import com.ivorononline.com.springboot_demo_currencies.loggerDB.repositories.LogRepository;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.mvc.method.annotation.StreamingResponseBody;

import javax.validation.constraints.NotBlank;
import java.time.LocalDate;
import java.util.List;

@Controller
public class LogController {

    @Autowired
    LogRepository logRepository;

    //=====
    // GET LOG
    //=====

    @ResponseBody
    @GetMapping("/GetLog")
    public ResponseEntity<StreamingResponseBody> getLog(
        @RequestParam @NotBlank String username,
        @RequestParam @NotBlank String startDate,
        @RequestParam @NotBlank String endDate
    ) {

        //CONVERT DATES
        LocalDate startDateConverted = LocalDate.parse(startDate);
        LocalDate endDateConverted = LocalDate.parse(endDate);

        //GET LOG
        List<Log> logs = logRepository.getLog(username, startDateConverted, endDateConverted);

        //CREATE EXCEL FILE
        Workbook workBook = new XSSFWorkbook();

        //CREATE TABLE
        Sheet sheet = workBook.createSheet("My Sheet");
        sheet.setColumnWidth(0, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(1, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(2, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(3, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(4, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(5, 12 * 256);           //10 characters wide
        sheet.setColumnWidth(6, 10 * 256);           //10 characters wide
        sheet.setColumnWidth(7, 40 * 256);           //10 characters wide
    }
}
```

```

        sheet.setColumnWidth(8, 90 * 256);           //10 characters wide

    //CREATE HEADER
    Row header = sheet.createRow(0);
    header.createCell(0).setCellValue("DATE");
    header.createCell(1).setCellValue("USERNAME");
    header.createCell(2).setCellValue("STATUS");
    header.createCell(3).setCellValue("METHOD");
    header.createCell(4).setCellValue("PROTOCOL");
    header.createCell(5).setCellValue("SERVER");
    header.createCell(6).setCellValue("PORT");
    header.createCell(7).setCellValue("PATH");
    header.createCell(8).setCellValue("QUERY");

    //POPULATE TABLE
    int rowIndex = 1;
    for(Log log : logs) {

        //CREATE ROW
        Row row = sheet.createRow(rowIndex++);

        //CREATE CELLS
        row.createCell(0).setCellValue(log.date.toString());
        row.createCell(1).setCellValue(log.username);
        row.createCell(2).setCellValue(log.status);
        row.createCell(3).setCellValue(log.method);
        row.createCell(4).setCellValue(log.protocol);
        row.createCell(5).setCellValue(log.serverName);
        row.createCell(6).setCellValue(log.serverPort);
        row.createCell(7).setCellValue(log.servletPath);
        row.createCell(8).setCellValue(log.queryString);

    }

    //DOWNLOAD EXCEL
    return ResponseEntity
        .ok()
        .contentType(MediaType.APPLICATION_OCTET_STREAM)
        .header(HttpHeaders.CONTENT_DISPOSITION, "inline;filename=\"Log.xlsx\"")
        .body(workBook::write);

}

//=====
// HANDLE EXCEPTIONS (it only catches first exception)
//=====

@ResponseBody
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(MissingServletRequestParameterException.class)
public String handleExceptions(MissingServletRequestParameterException exception) {

    //GET EXCEPTION DETAILS
    String parameterType = exception.getParameterType(); //String
    String parameterName = exception.getParameterName(); //name
    String message      = exception.getMessage();          //Required String parameter 'name' is not present

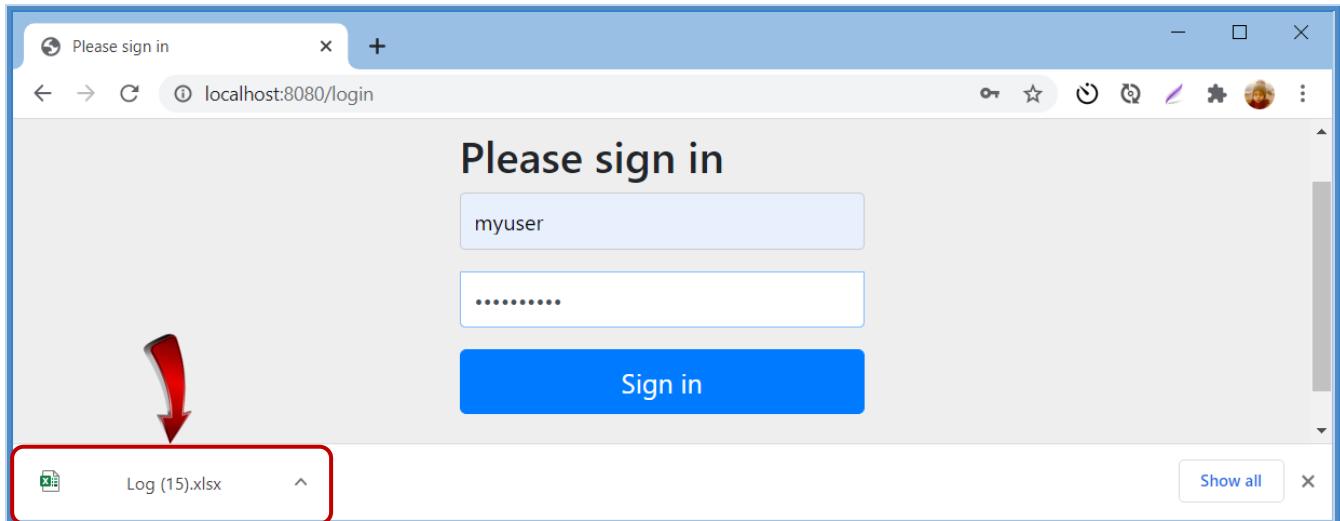
    //RETURN MESSAGE
    return message;

}
}

```

Results

<http://localhost:8080/GetLog?username=myuser&startDate=2021-03-01&endDate=2021-08-01>



Log.xlsx



The screenshot shows a Microsoft Excel spreadsheet titled "Log (14).xlsx". The data is organized into columns labeled A through I. Column A contains dates from April 9, 2021. Column B contains the user "myuser". Column C shows status codes (200) and methods (GET). Column D shows protocols (HTTP/1.1). Column E lists servers (localhost). Column F lists ports (8080). Column G lists paths corresponding to the queries. Column H contains the query parameters. Column I is empty.

	A	B	C	D	E	F	G	H	I
1	DATE	USERNAME	STATUS	METHOD	PROTOCOL	SERVER	PORT	PATH	QUERY
2	2021-04-09	myuser	200	GET	HTTP/1.1	localhost	8080	/GetAllCurrencyNames	
3	2021-04-09	myuser	200	GET	HTTP/1.1	localhost	8080	/GetFirstLastDate	currencyName=EUR
4	2021-04-09	myuser	200	GET	HTTP/1.1	localhost	8080	/GetAverageValue	currencyName=EUR&startDate=2017-05-30&endDate=2017-05-30
5	2021-04-09	myuser	200	GET	HTTP/1.1	localhost	8080	/GetLog	username=myuser&startDate=2021-03-01&endDate=2021-08-01

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, Git, Window, Help.
- Title Bar:** springboot_demo_currencies - LogController.java
- Toolbar:** Includes icons for file operations like Open, Save, Find, and Git.
- Project View (Left):**
 - Project: Shows the project structure with logs, src, main, and java folders.
 - src folder contains business, loggerAOP, loggerDB, and controllers.
 - controllers folder contains LogController.java.
 - loggerDB folder contains LogRepository.java.
 - main folder contains entities, filters, LogFilter, and repositories.
 - repositories folder contains LogRepository.java.
 - resources folder contains static, templates, application.properties, and logback.xml.
- Code Editor (Right):** The code for LogController.java is displayed:

```
27  @Controller
28  public class LogController {
29
30      @Autowired LogRepository logRepository;
31
32      //=====
33      // GET LOG
34      //=====
35
36      @ResponseBody
37      @RequestMapping(value = "/GetLog")
38      public ResponseEntity<StreamingResponseBody> getLog(
39          @RequestParam @NotBlank String username,
40          @RequestParam @NotBlank String startDate,
41          @RequestParam @NotBlank String endDate
42      ) {
43
44          //CONVERT DATES
45          LocalDate startDateConverted = LocalDate.parse(startDate);
46          LocalDate endDateConverted = LocalDate.parse(endDate);
47
48          //GET LOG
49          List<Log> logs = logRepository.getLog(username, startDateConverted, endDateConverted);
50
51          //CREATE EXCEL FILE
52      }
53  }
```
- Bottom Navigation Bar:** Git, Run, Database, TODO, Problems, Terminal, Profiler, Endpoints, Build, Spring.
- Status Bar:** All files are up-to-date (11 minutes ago), 26:1 CRLF UTF-8 2 spaces, Step7_DownloadExcelWithDBLog.

```
<dependencies>

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

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

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.15</version>
    </dependency>

</dependencies>
```

5.1.8 Step 8 - Unit Testing

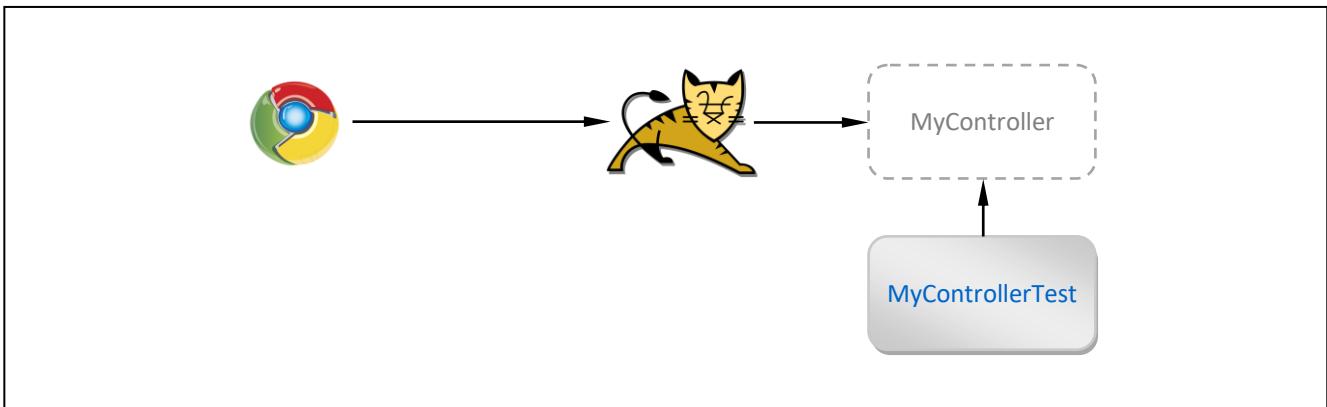
Info

[G]

- In this step we will show how to create Unit Tests to test Controller's Endpoints.

Application Schema

[Results]



Procedure

- Create Project: `springboot_filter_create` (add Spring Boot Starters from the table)
- Create Test Class: `MyControllerTest.java`

`MyControllerTest.java`

```
package com.ivoronline.springboot_demo_currencies.business.controllers;

import com.ivoronline.springboot_demo_currencies.business.repositories.CurrencyRepository;
import com.ivoronline.springboot_demo_currencies.business.services.MyService;
import com.ivoronline.springboot_demo_currencies.loggerDB.repositories.LogRepository;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@AutoConfigureMockMvc(addFilters = false)
@WebMvcTest/controllers = MyController.class)
class MyControllerTest {

    @Autowired MockMvc          mockMvc;
    @Autowired MyController      myController;

    @MockBean MyService          myService;
    @MockBean CurrencyRepository currencyRepository;
    @MockBean LogRepository      logRepository;

    //=====
    // TEST ENDPOINT URL: GET MyService
    //=====

    @Test
    void getAllCurrencyNames() throws Exception {
        mockMvc.perform(get("/GetAllCurrencyNames")).andExpect(status().isOk());
    }

    //=====
    // TEST ENDPOINT URL: GET GetFirstLastDate
    //=====

    @Test
    void GetFirstLastDate() throws Exception {
        mockMvc.perform(get("/GetFirstLastDate?currencyName=EUR")).andExpect(status().isOk());
    }

    //=====
    // TEST ENDPOINT URL: GET GetAverageValue
    //=====

    @Test
    void GetAverageValue() throws Exception {
        mockMvc.perform(get("/GetAverageValue?currencyName=EUR&startDate=2000-05-30&endDate=2010-05-
30")).andExpect(status().isOk());
    }

}
```

Results

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_demo_currencies". The "src/test/java" directory is expanded, showing the package "com.ivoronline.springboot_demo_currencies" and its sub-package "business.controllers". Inside "business.controllers", there are files: "MyControllerTest.java", "SpringbootDemoCurrenciesApplicationTest.java", and "MyController.java".
- Code Editor:** Displays the content of "MyControllerTest.java". The code uses Mockito annotations like @AutoConfigureMockMvc, @WebMvcTest, @Autowired, and @MockBean to test the "MyController" class. It includes test methods for endpoints like "/GetAllCurrencyNames" and "/GetFirstLastDate".
- Run Tab:** Shows the results of a run named "MyControllerTest". It indicates "Tests passed: 3 of 3 tests – 233 ms". The log output shows INFO messages from Logback.
- Bottom Status Bar:** Shows "Tests passed: 3 (6 minutes ago)" and the step number "Step8_[UnitTesting]".

```
<dependencies>

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

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

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

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

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

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.15</version>
    </dependency>

</dependencies>
```

6 Appendix

Info

- Following tutorials show how to use additional technologies that can be helpful while developing Spring Boot Application.

6.1 IntelliJ

Info

- Following tutorials show how to use IntelliJ to create Spring Boot Project since all tutorials are done with IntelliJ.
- But you can use any other IDE of your choice.

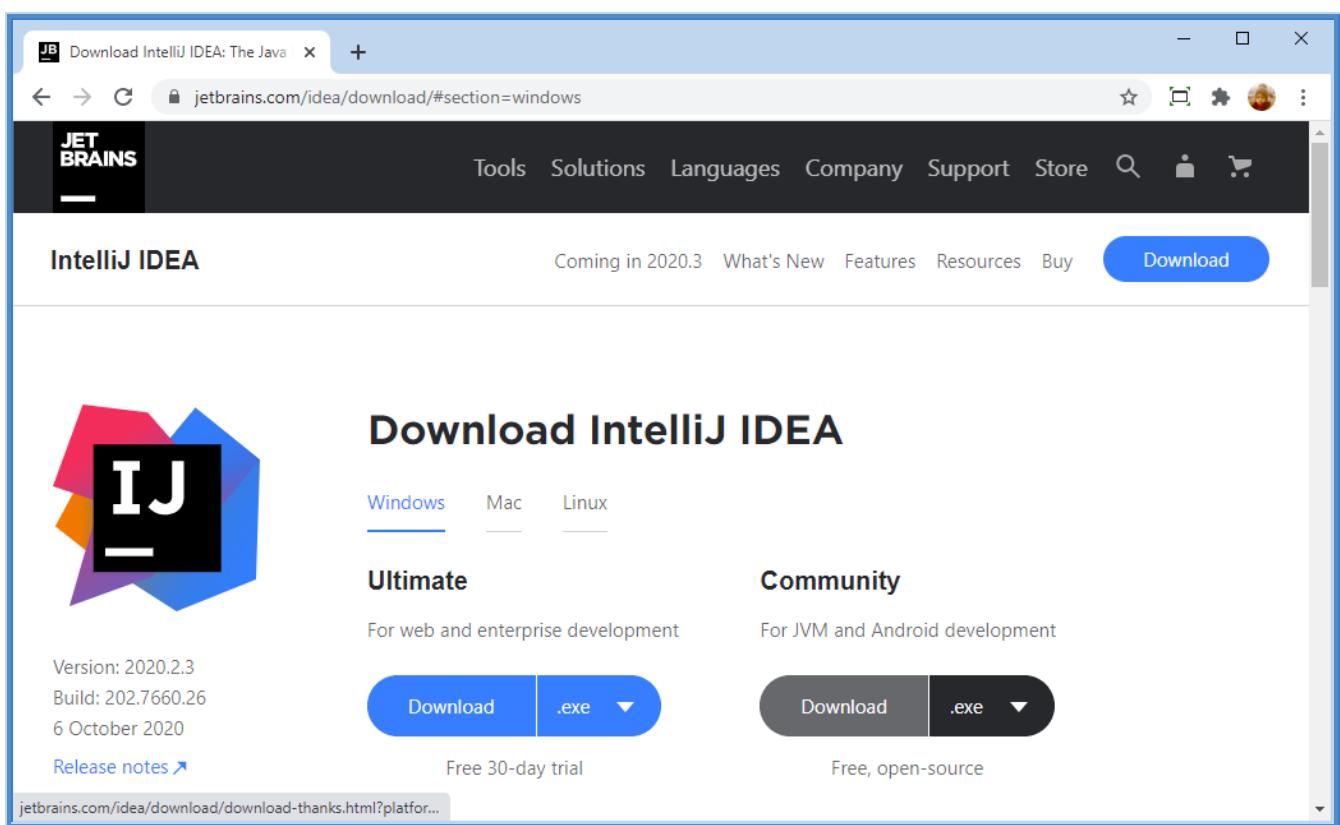
6.1.1 Install

Info

- This tutorial shows how to install IntelliJ IDEA 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

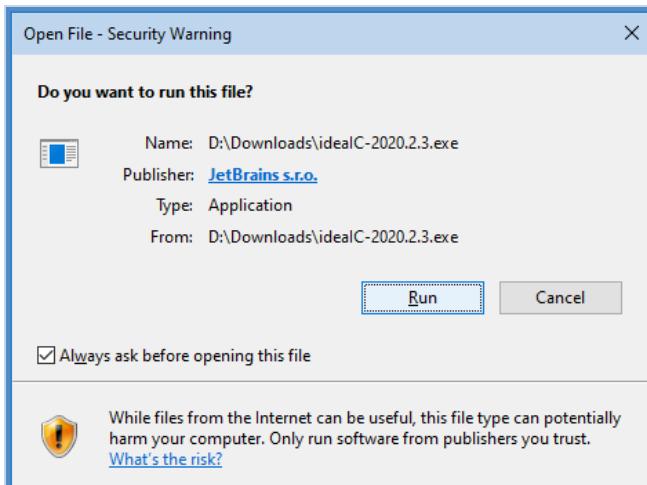
- <https://www.jetbrains.com/idea>
- Download
- Community
- Download
- Save as: D:\Downloads\idealC-2020.2.3.exe



Install IntelliJ IDEA

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

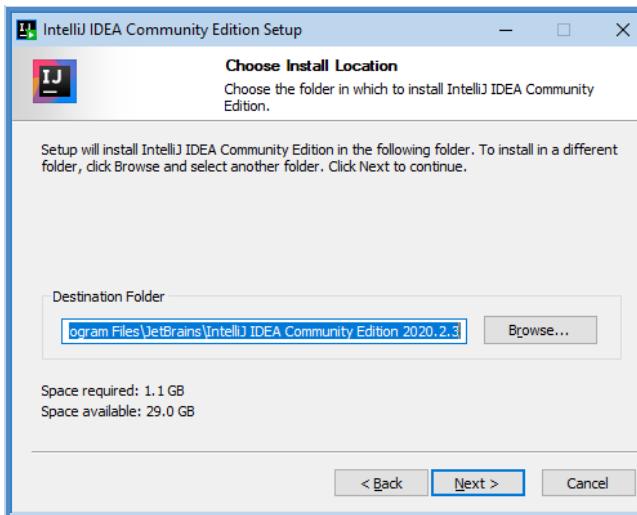
Run



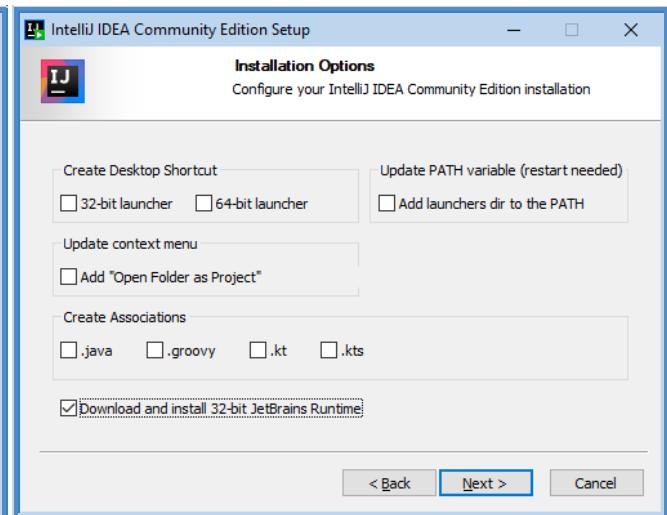
Next



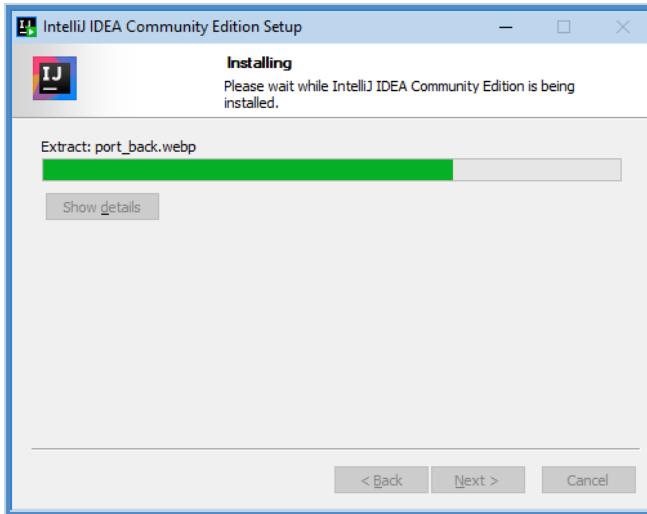
Next



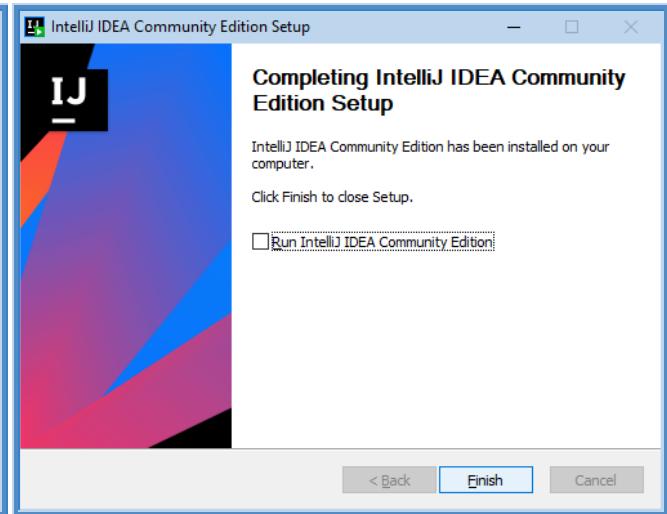
Next



Progress



Finish



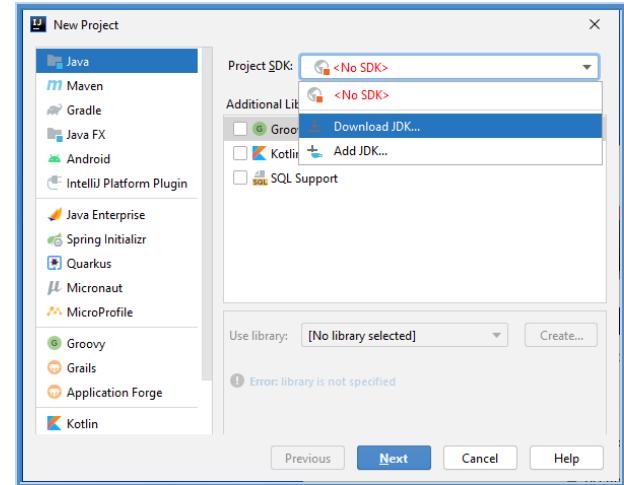
Download Java JDK

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

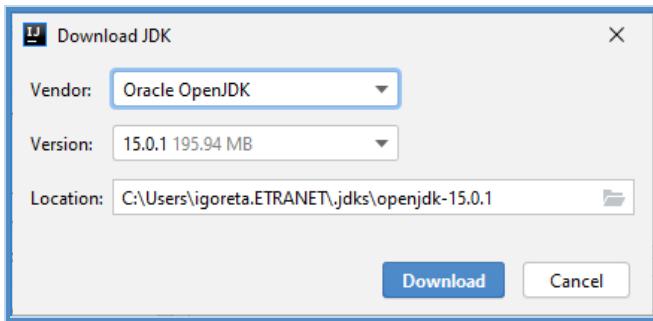
New Project



Download JDK



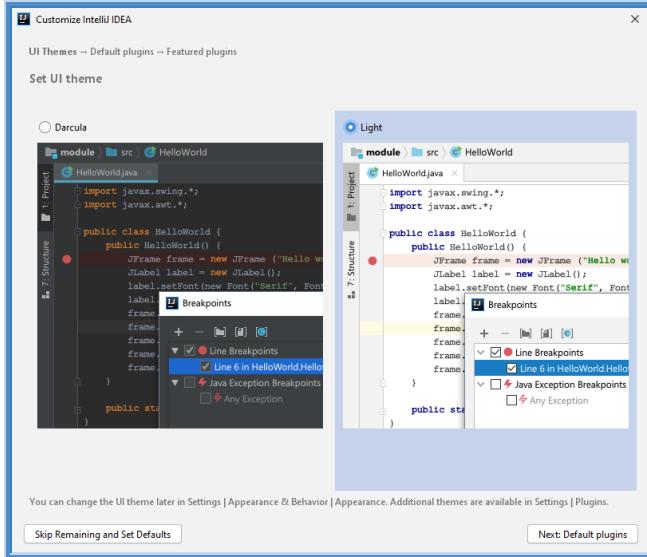
Download



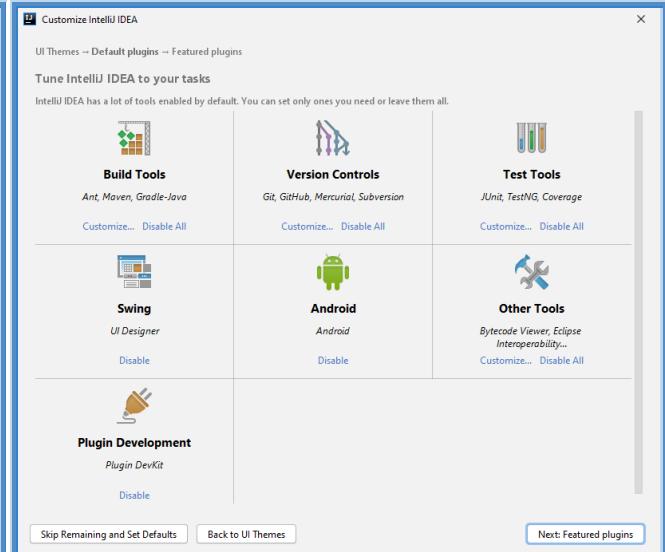
Customize

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

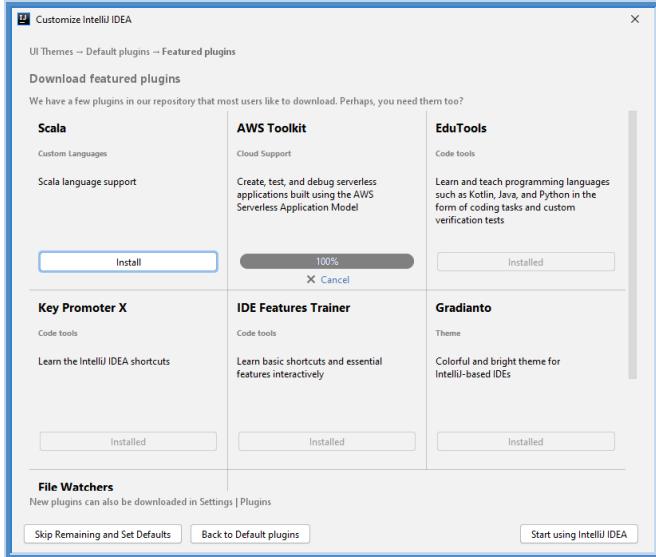
Light



Next



Start



6.1.2 Create Project

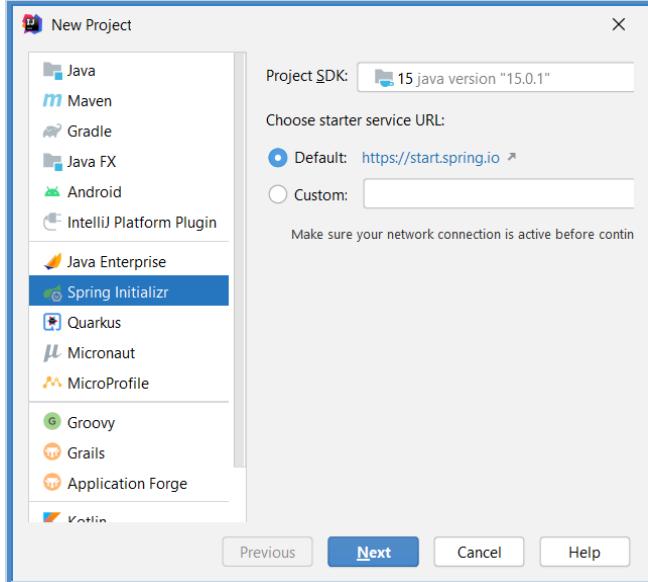
Info

- 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.](#)

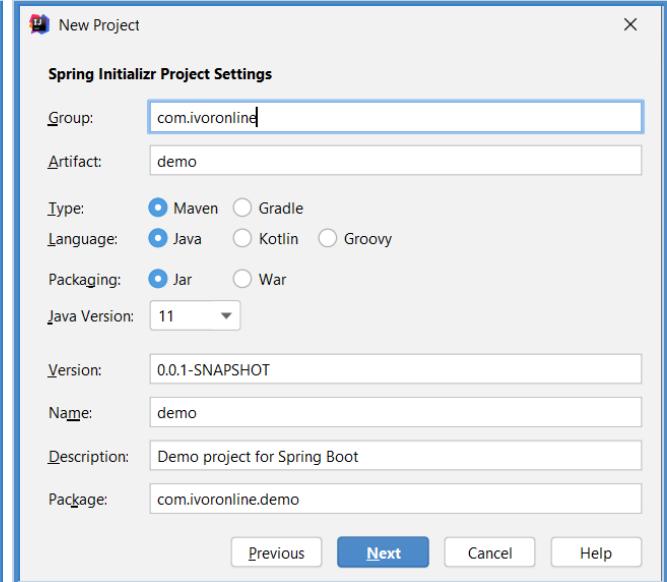
Create Spring Boot Project

- 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)

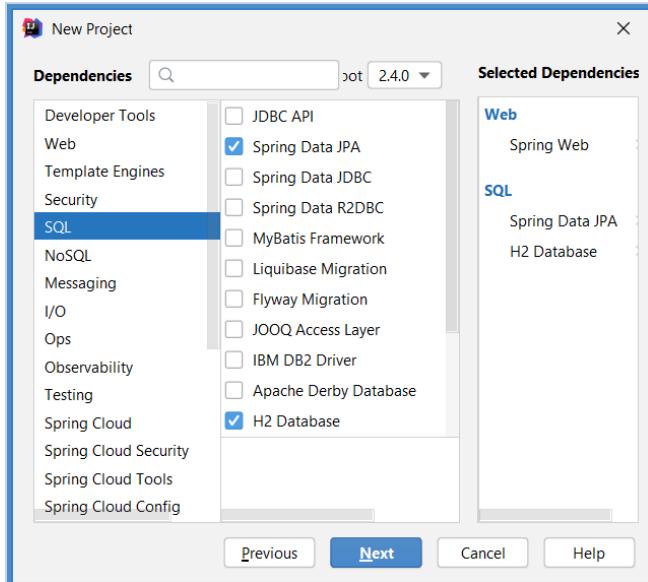
File - New - Project - Spring Initializr



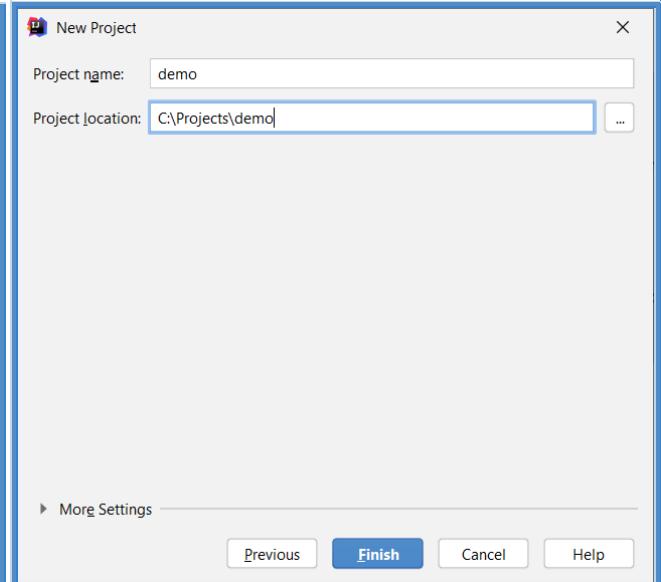
Project Settings



Dependencies



Project name



6.1.3 Run Application

Info

- This tutorial shows how to run Spring Boot Application.
- We will display 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;

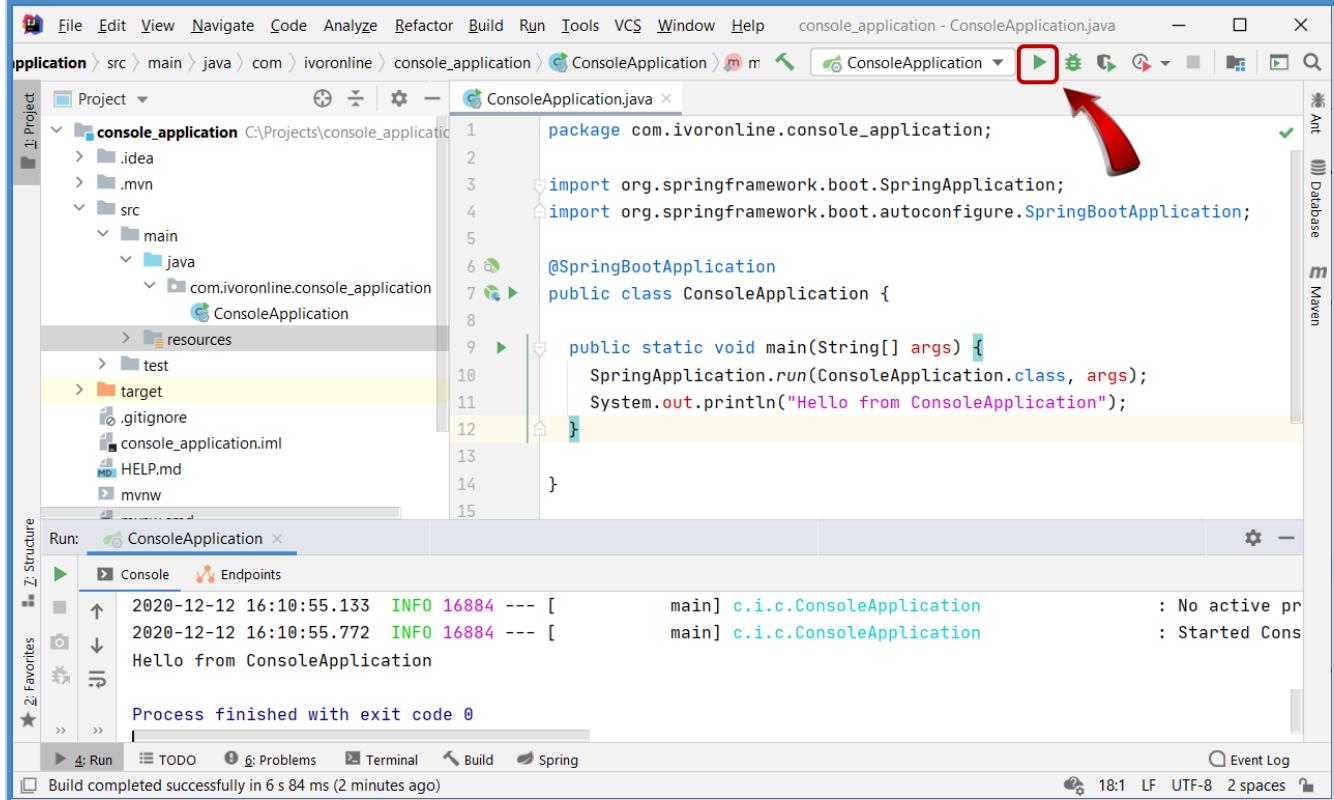
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@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
```

6.2 JUnit

Info

- Following tutorials show alternative approaches of using JUnit compared to those covered in the main chapter.
- These might be useful in certain scenarios or to understand code from other developers.

6.2.1 Test Exception

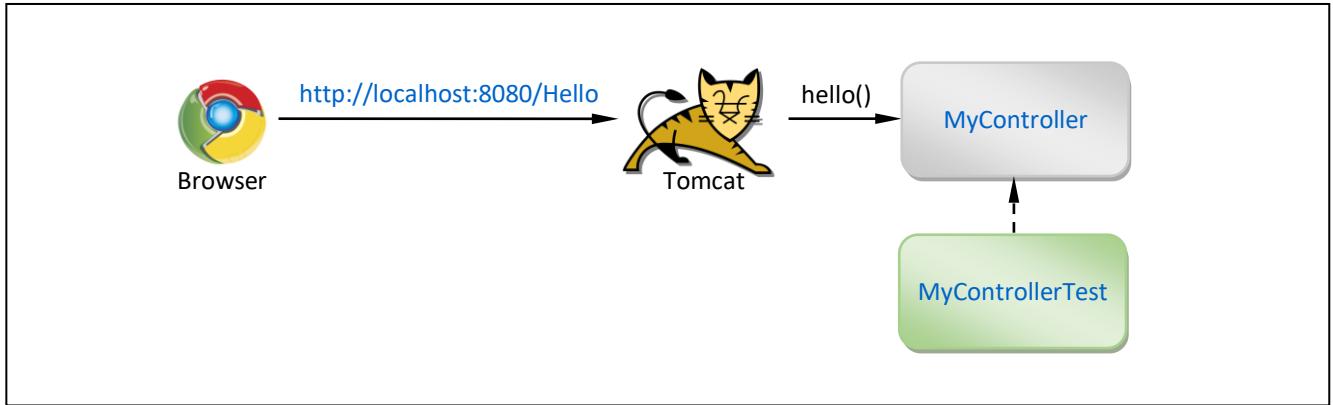
Info

[G] [R]

- **Includes Alternative Code:** Executable, try - catch
- This tutorial shows how to use different `assert()` Methods which are used to assert/evaluate/compare provided values.
- If values are don't match then Test ends in **Failure** at the **first** assert that didn't pass.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- [Create Project:](#) `springboot_junit` (add Spring Boot Starters from the table)
- [Create Package:](#) `controllers` (inside main package)
- [Create Class:](#) `MyController.java` (inside controllers package)
- [Create Test Class:](#) `MyControllerTest.java`

MyController.java

```
package com.ivoronline.springboot_junit_exception.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() throws Exception {
        if (1 == 1) { throw new IllegalArgumentException("Argument is missing"); }
        return "Hello from Controller";
    }

}
```

MyControllerTest.java

```
package com.ivoronline.springboot_junit_exception.controllers;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class MyControllerTest {

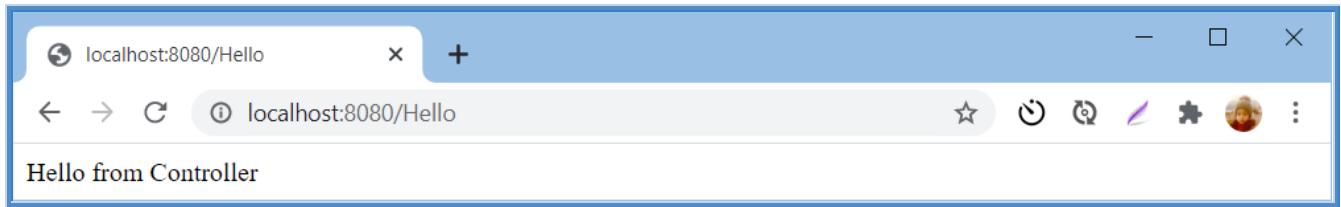
    @Autowired MyController myController;

    @Test
    void hello1() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> myController.hello() );//Type
        assertEquals("Argument is missing", exception.getMessage()); //Message
    }

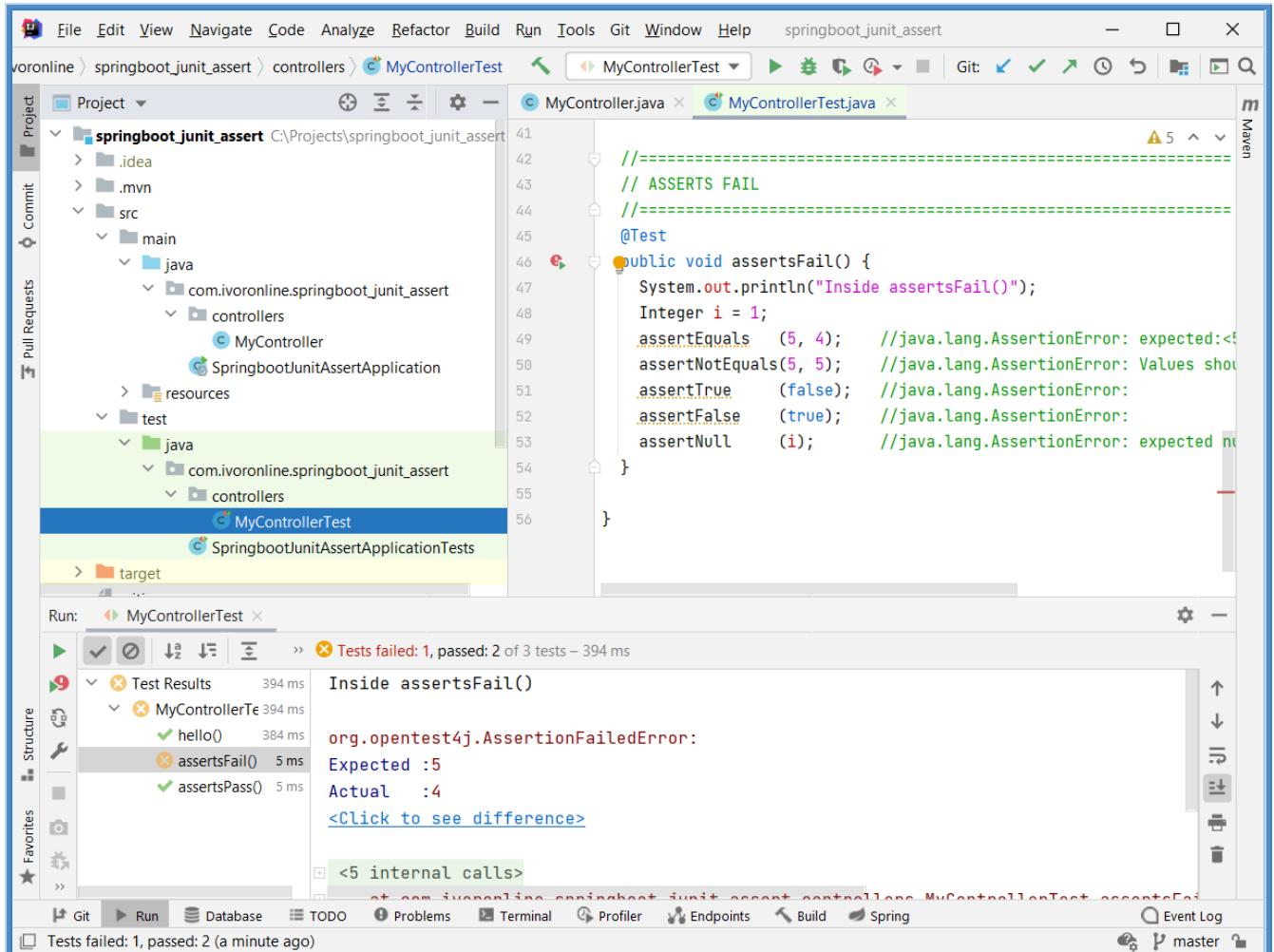
}
```

Result

<http://localhost:8080>Hello>



Run Test Class: PersonTest



pom.xml

```
<dependencies>

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

</dependencies>
```

6.3 Lombok

Info

- Following tutorials show how to use
 - Lombok API** contains Annotations that automatically generate helper methods (setters, getters, constructors)
 - Lombok Plugin**
 - prevents IntelliJ for showing errors while using Lombok (if it can't find explicit Constructor and such)
 - provides refactoring options to add/remove Lombok Annotations

Hiding Helper Methods

- The only useful part of the **Entity Class** are its **Properties** (from a business logic perspective).
- Helper Methods (like setters and getters) are just part of the plumbing/implementation so that Entity Class could fulfill its role to store some data/properties. Since these helper methods are the same for all Entities (and therefore do not provide any useful information) they are obfuscating true purpose of Entity Class. Therefore it is useful to hide helper methods (so that we can focus only on the Properties as we move between Entities).
- For that purpose we can use **Lombok API** which contains Annotations that inject helper Methods into Entity Class. This way helper Methods are no longer visible inside the Entity Class (allowing us to ignore them and focus on Properties).

Annotations

- @Data** Annotation is most useful since it generates typical boilerplate code for Entities.
@Data is used as shorthand for: @Getter, @Setter, @ToString, @EqualsAndHashCode, @RequiredArgsConstructor.
But you can still explicitly add any of these if you need to specify additional parameters (fine tune Annotation).

@Data Annotations

ANNOTATION	DESCRIPTION
@Data	<ul style="list-style-type: none">@Getter, @Setter, @ToString, @EqualsAndHashCode, @RequiredArgsConstructor (Constructor is not generated if one already exists)
@Getter	<ul style="list-style-type: none">Creates getter Methods for all Properties
@Setter	<ul style="list-style-type: none">Creates setter Methods for all non-final Properties
@ToString	<ul style="list-style-type: none">Creates toString Method from Class Name and Properties<ul style="list-style-type: none">Optional parameter to include Property's nameOptional parameter to call super toString Method
@EqualsAndHashCode	<ul style="list-style-type: none">Creates equals() and hashCode() Methods
@RequiredArgsConstructor	<ul style="list-style-type: none">Generate Constructor for final and @NonNull Properties

Additional Annotations

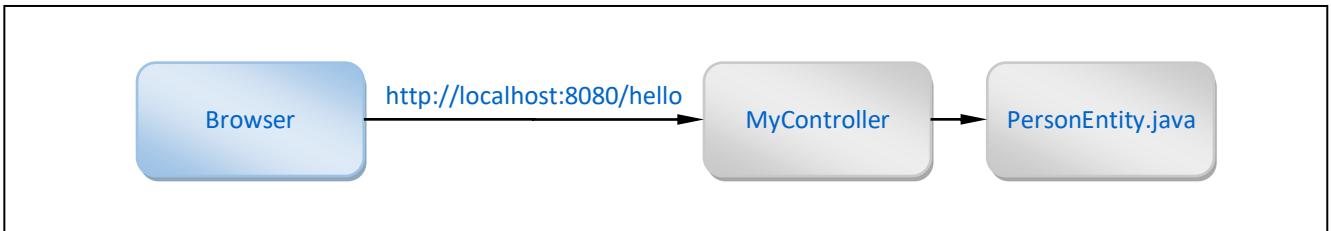
ANNOTATION	DESCRIPTION
@NoArgsConstructor	<ul style="list-style-type: none">Generate Constructor with no parameters (Throws error if there are final Properties)
@AllArgsConstructor	<ul style="list-style-type: none">Generate Constructor for all Properties (@NonNull Properties are checked)

6.3.1 Lombok API - Use

Info

- This tutorial shows how to include Lombok API in your project by selecting its [Spring Boot Starter](#).
- Alternatively you can add Maven dependency to [your pom.xml](#).
- This tutorial is exactly the same as [Entity - @Data \(Lombok\)](#).

Application Schema



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @RequestMapping. Includes Tomcat Server.
Developer Tools	Lombok	Enables @Data which generate helper methods (setters, getters, ...)

pom.xml

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

Procedure

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

PersonEntity.java

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

import lombok.Data;
import org.springframework.stereotype.Component;

@Data
@Component
public class PersonEntity {
    private Long id;
    private String name;
    private Integer age;
}
```

MyController.java

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

import com.ivoronline.springboot.entity_lombok.entities.PersonEntity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;

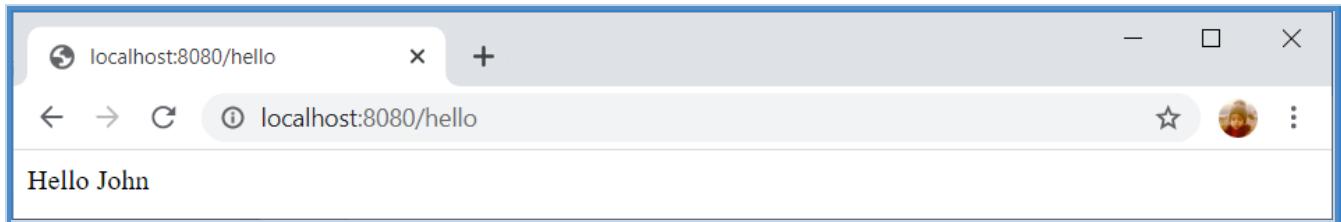
@Controller
public class MyController {

    @Autowired
    PersonEntity personEntity;

    @RequestBody
    @RequestMapping("/hello")
    public String sayHello() {
        personEntity.setName("John");
        String name = personEntity.getName();
        return "Hello " + name;
    }
}
```

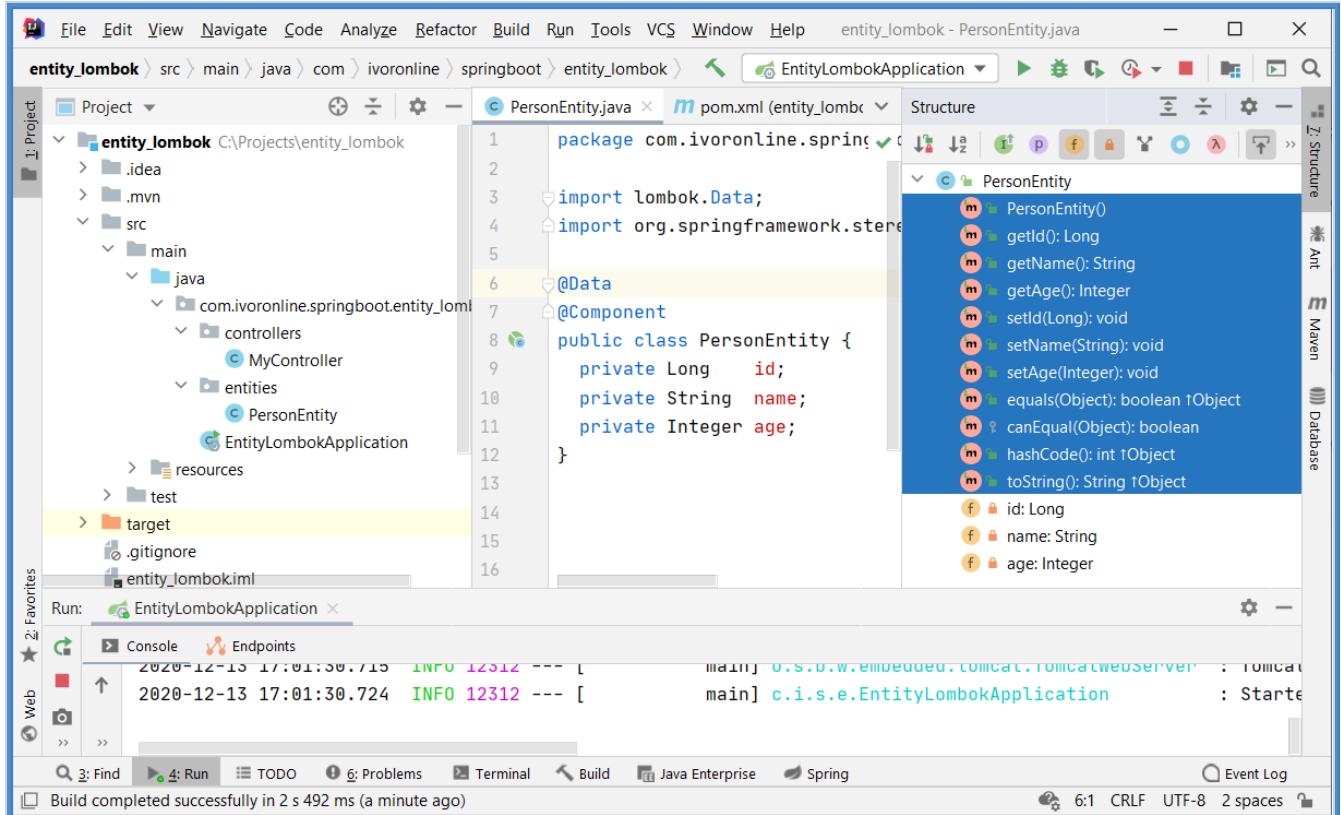
Results

<http://localhost:8080/hello>



Application Structure

(Lombok generated methods are shown under Structure View)



pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

</dependencies>
```

6.3.2 Lombok Plugin - Install for IntelliJ

Info

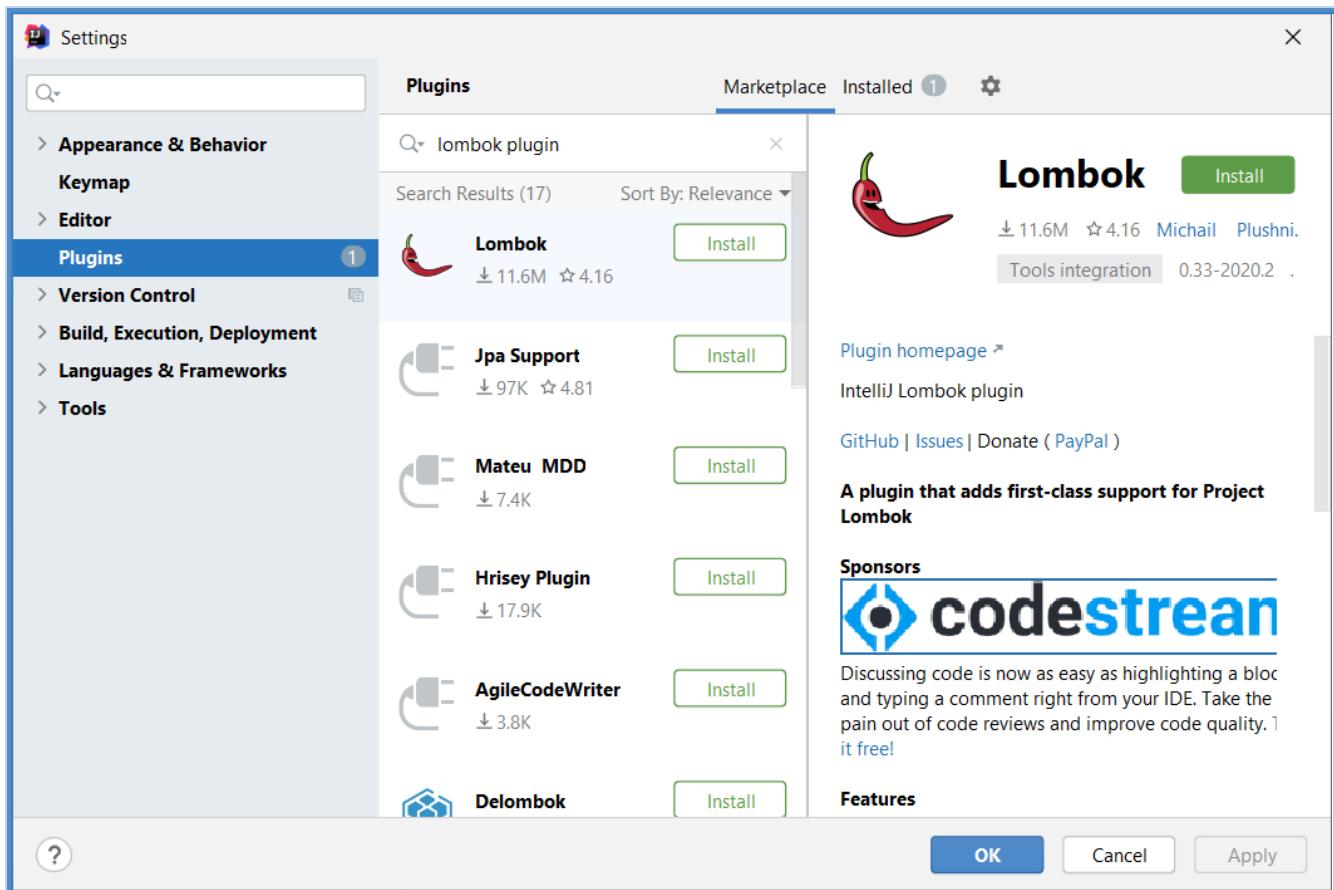
[R]

- This tutorial shows how to install Lombok **Plugin** for IntelliJ.

Install Lombok Plugin

- Start IntelliJ
- File
- Settings
- Plugins
- Search: Lombok Plugin
- Install
- Enable Annotation Processing
- Restart IntelliJ

Install Lombok Plugin



Enable

! Lombok Requires Annotation Processing

Do you want to enable annotation processors? [Enable](#)

6.3.3 Lombok Plugin - Lombok

Info

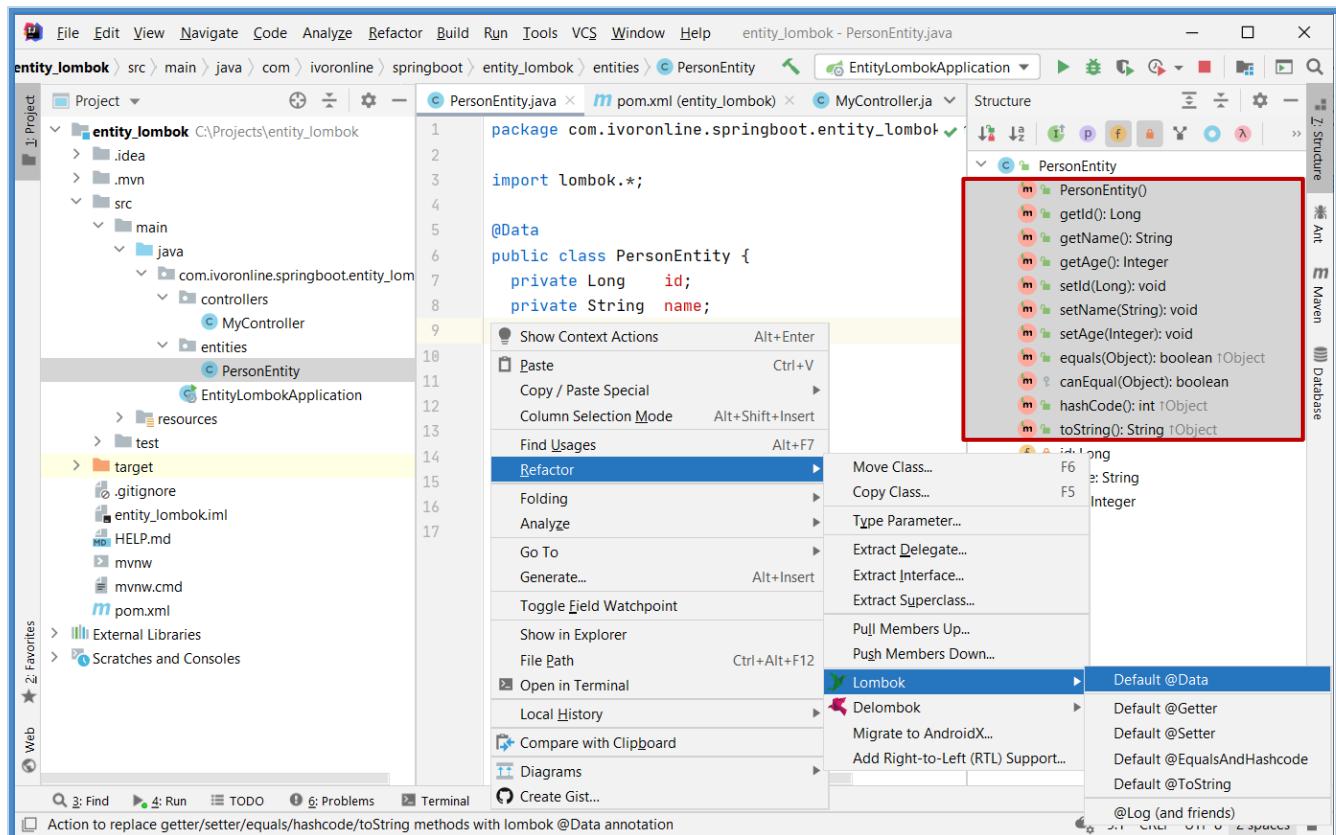
- This tutorial shows how to add Lombok Annotations through Lombok Plugin.
- Alternativly you can simply type these annotations manually in your code.
- Generated methods are visible in the Structure View: View - Tool Window - Structure
- You can use [Lombok Plugin - Delombok](#) to remove Lombok Annotations and insert these methods in the Class.

Delombok

- RC inside Class
- Refactor
- Lombok
- Default @Data

Delombok @Data

(generated methods are visible under Structure tab)



PersonEntity.java

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

import lombok.*;

@Data
public class PersonEntity {
    private Long id;
    private String name;
    private Integer age;
}
```

6.3.4 Lombok Plugin - Delombok

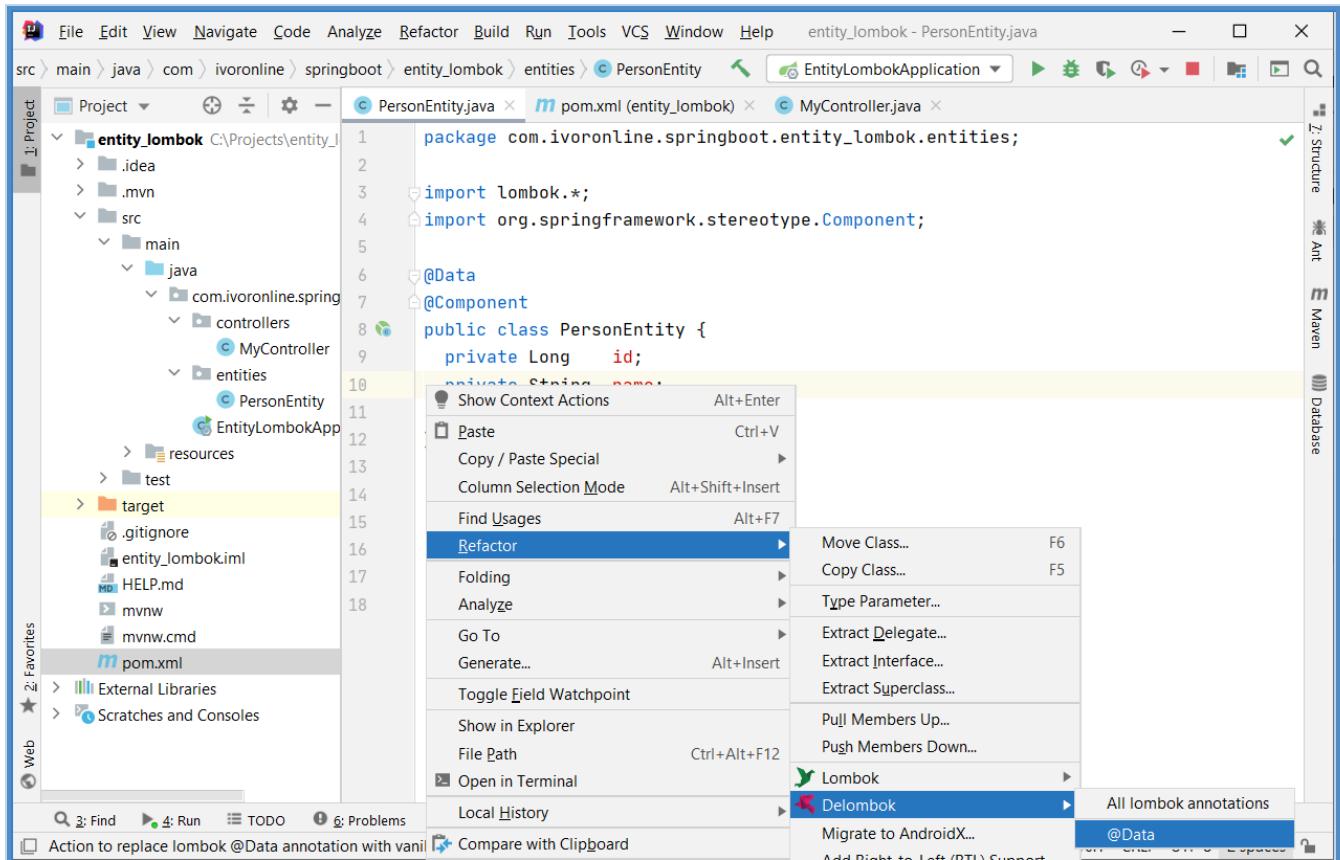
Info

- Lombok Plugin can be used to Delombok your Class.
- This will remove selected Lombok Annotation and replace it with the Lombok generated code.
- You can use this approach to generate helper methods even if you don't want them to be hidden.

Delombok

- RC inside Class
- Refactor
- Delombok
- @Data

Delombok @Data



PersonEntity.java

(Lomboked)

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

import lombok.*;

@Data
public class PersonEntity {
    private Long id;
    private String name;
    private Integer age;
}
```

```

package com.ivorononline.springboot.entity_lombok.entities;

public class PersonEntity {

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

    //NO-ARG CONSTRUCTOR
    public PersonEntity() { }

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

    //SETTERS
    public void setId (Long id) { this.id = id; }
    public void setName(String name) { this.name = name; }
    public void setAge (Integer age) { this.age = age; }

    //TOSTRING
    public String toString() {
        return "PersonEntity(id=" + this.getId() + ", name=" + this.getName() + ", age=" + this.getAge() + ")";
    }

    //EQUALS
    public boolean equals(final Object o) {
        if (o == this) return true;
        if (!(o instanceof PersonEntity)) return false;
        final PersonEntity other = (PersonEntity) o;
        if (!other.canEqual((Object) this)) return false;
        final Object this$id = this.getId();
        final Object other$id = other.getId();
        if (this$id == null ? other$id != null : !this$id.equals(other$id)) return false;
        final Object this$name = this.getName();
        final Object other$name = other.getName();
        if (this$name == null ? other$name != null : !this$name.equals(other$name)) return false;
        final Object this$age = this.getAge();
        final Object other$age = other.getAge();
        if (this$age == null ? other$age != null : !this$age.equals(other$age)) return false;
        return true;
    }

    //CANEQUAL
    protected boolean canEqual(final Object other) { return other instanceof PersonEntity; }

    //HASHCODE
    public int hashCode() {
        final int PRIME = 59;
        int Results = 1;
        final Object $id = this.getId();
        Results = Results * PRIME + ($id == null ? 43 : $id.hashCode());
        final Object $name = this.getName();
        Results = Results * PRIME + ($name == null ? 43 : $name.hashCode());
        final Object $age = this.getAge();
        Results = Results * PRIME + ($age == null ? 43 : $age.hashCode());
        return Results;
    }
}

```

6.4 Mockito

Info

- Following tutorials show alternative approaches of using Mockito compared to those covered in the main chapter.
- These might be useful in certain scenarios or to understand code from other developers.

6.4.1 @Mock - Into Property

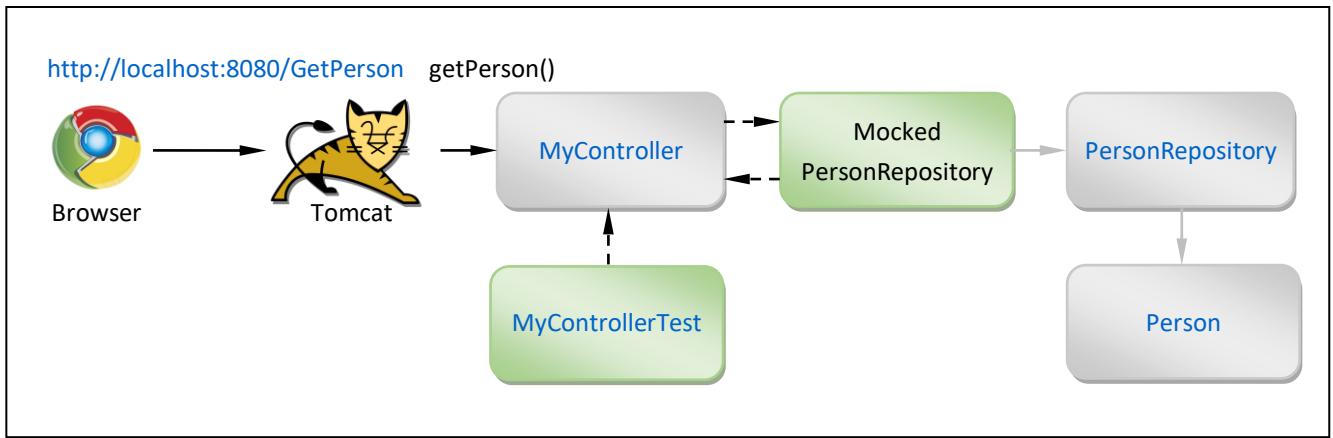
Info

[G]

- This tutorial shows how to use **@Mock to Mock Class** `@Mock PersonRepository personRepository`
- Then we need to inject Mocked Class into Controller that is being tested `@InjectMocks MyController myController` Mockito looks for `@Autowired PersonRepository personRepository` and inserts instance of a Mocked Class.
- Then we Mock Method `getPersonById(1)` by defining what it should return for Input Parameter `1`. Now when Controller calls `personRepository.getPersonById(1)` Mocked Method of a Mocked Class will be called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_junit` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package `src\test\java\com.ivoronline.springboot_junit`)
 - Create Test Class: `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito.repositories;

import com.ivoronline.springboot_mockito.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivoronline.springboot_mockito_mock_injectinto_property.controllers;

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

@Controller
public class MyController {

    //PROPERTIES
    @Autowired PersonRepository personRepository;

    //ENDPOINT
    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(1);

        //GREET PERSON
        return "Hello " + person.name;
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_mock_injectinto_property.controllers;

import com.ivoronline.springboot_mockito_mock_injectinto_property.entities.Person;
import com.ivoronline.springboot_mockito_mock_injectinto_property.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepository;

    //INSTANTIATE CLASS BEING TESTED. INJECT MOCKS (WHERE @AUTOWIRED IS USED)
    @InjectMocks MyController myController;

    @Test
    void getPerson() {

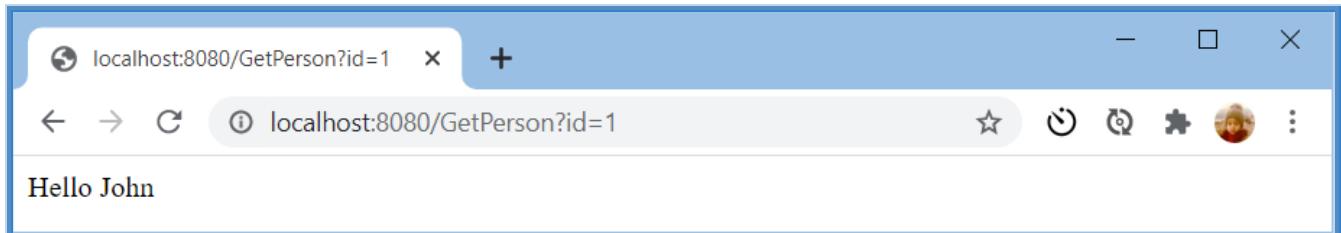
        //MOCK REPOSITORY'S METHOD: getPersonById(1)
        when(personRepository.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

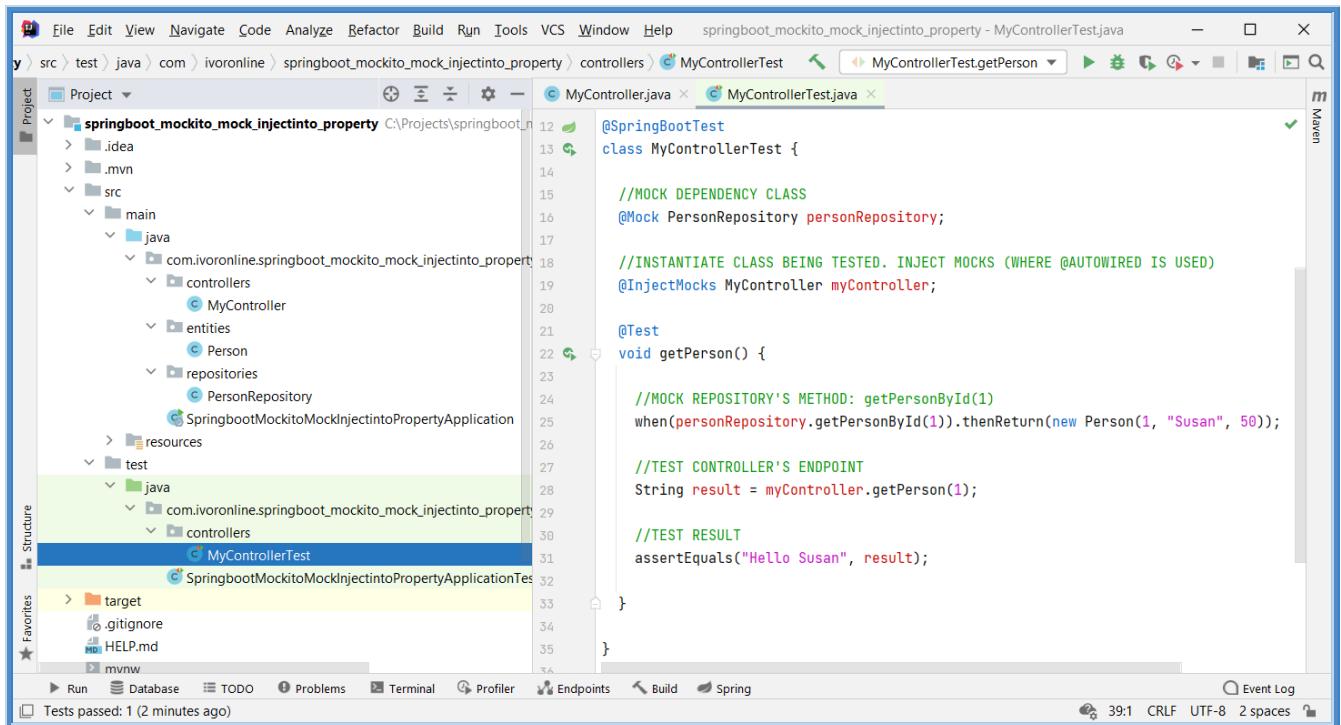
        //TEST RESULT
        assertEquals("Hello Susan", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

6.4.2 @Mock - Into Setter

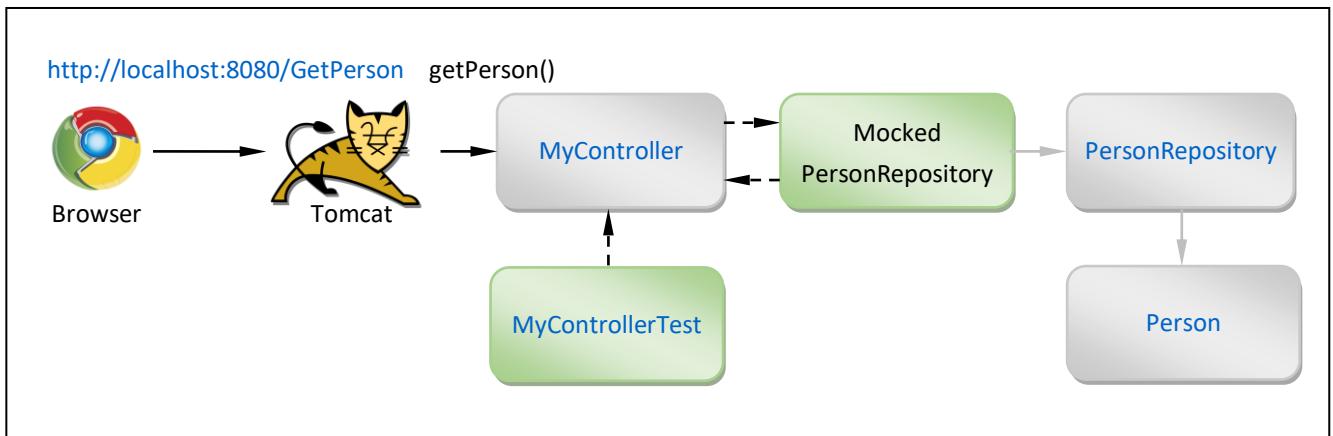
Info

[G]

- This tutorial shows how to use **@Mock** to **Mock Class**
Then we use Mocked Class to set Controller's Repository
`@Mock PersonRepository personRepository
myController.setPersonRepository(personRepository)`
- Then we Mock Method `getPersonById(1)` by defining what it should return for Input Parameter `1`.
Now when Controller calls `personRepository.getPersonById(1)` Mocked Method of a Mocked Class will be called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_junit` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package `src\main\java\com.ivoronline.springboot_junit`)
 - Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package `src\test\java\com.ivoronline.springboot_junit`)
 - Create Test Class: `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito.repositories;

import com.ivoronline.springboot_mockito.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mock_injectinto_setter.controllers;

import com.ivorononline.springboot_mock_injectinto_setter.entities.Person;
import com.ivorononline.springboot_mock_injectinto_setter.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    //PROPERTIES
    @Autowired PersonRepository personRepository;

    //SETTER
    @Autowired
    public void setPersonRepository(PersonRepository personRepository) {
        this.personRepository = personRepository;
    }

    //ENDPOINT
    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(1);

        //GREET PERSON
        return "Hello " + person.name;
    }
}
```

MyControllerTest.java

```
package com.ivorononline.springboot_mock_injectinto_setter.controllers;

import com.ivorononline.springboot_mock_injectinto_setter.entities.Person;
import com.ivorononline.springboot_mock_injectinto_setter.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepository;

    @Test
    void getPerson() {

        //MOCK REPOSITORY'S METHOD: getPersonById(1)
        when(personRepository.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //INSTANTIATE CONTROLLER. SET PROPERTY TO MOCKED INSTANCE.
        MyController myController = new MyController();
        myController.setPersonRepository(personRepository);

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

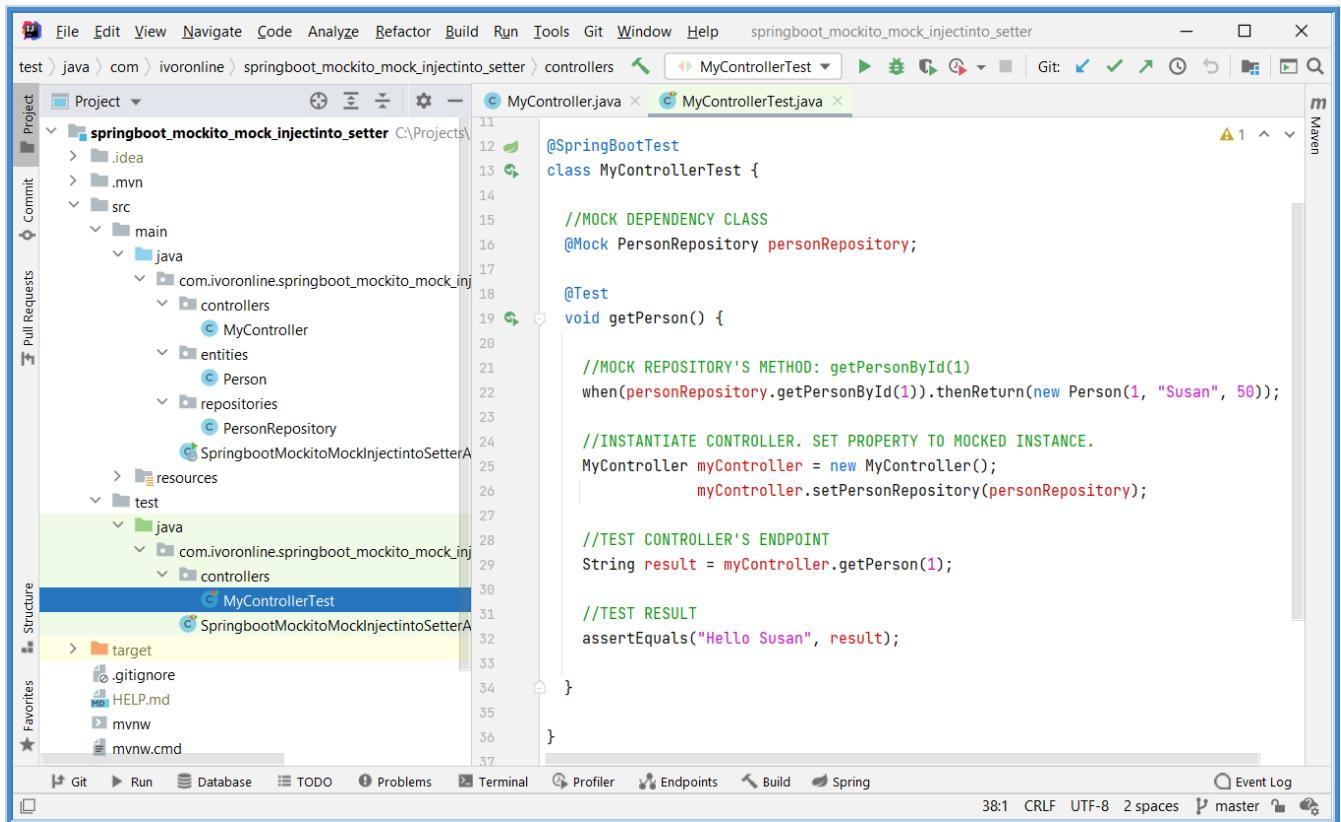
        //TEST RESULT
        assertEquals("Hello Susan", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

6.4.3 @Mock - Into Constructor

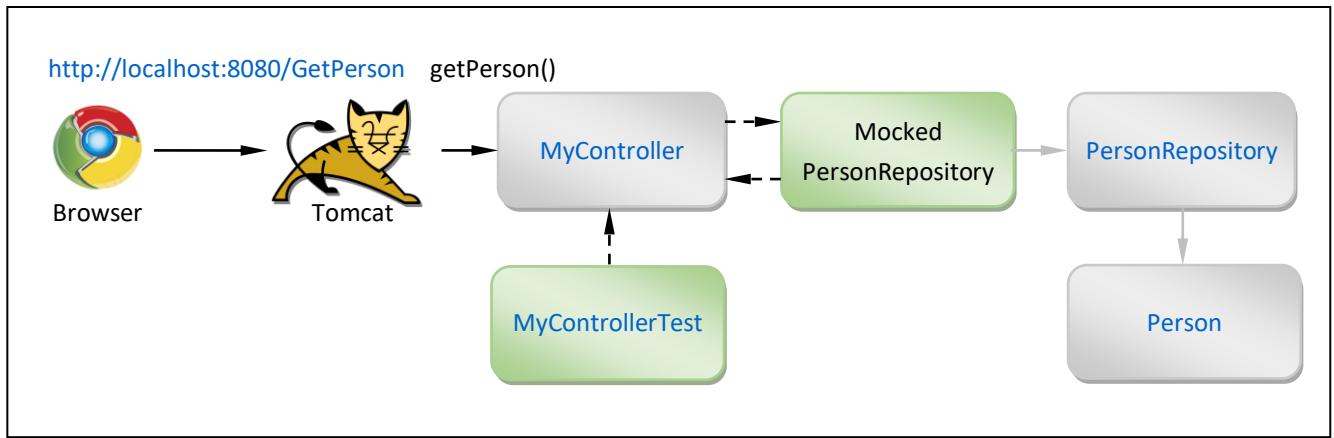
Info

[G]

- This tutorial shows how to use **@Mock to Mock Class**
 @Mock PersonRepository personRepository
 @InjectMocks MyController myController
- Then we use Mocked Class to instantiate Controller
 Mockito looks for **@Autowired PersonRepository personRepository** and inserts instance of a Mocked Class.
- Then we Mock Method **getPersonById(1)** by defining what it should return for Input Parameter **1**.
 Now when Controller calls **personRepository.getPersonById(1)** Mocked Method of a Mocked Class will be called.

Application Schema

[Result]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: Controller Annotations, Tomcat Server

Procedure

- Create Project: `springboot_junit` (add Spring Boot Starters from the table)
- Create Package: `entities` (inside main package src\main\java\com.ivoronline.springboot_junit)
- Create Class: `Person.java` (inside entities package)
- Create Package: `repositories` (inside main package src\main\java\com.ivoronline.springboot_junit)
- Create Class: `PersonRepository.java` (inside entities package)
- Create Package: `controllers` (inside main package src\main\java\com.ivoronline.springboot_junit)
- Create Class: `MyController.java` (inside controllers package)
- Create Package: `controllers` (inside test package src\test\java\com.ivoronline.springboot_junit)
- Create Test Class: `MyControllerTest.java` (inside entities package)

Person.java

```
package com.ivoronline.springboot_mockito.entities;

public class Person {

    //PROPERTIES
    public Integer id;
    public String name;
    public Integer age;

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

}
```

PersonRepository.java

```
package com.ivoronline.springboot_mockito.repositories;

import com.ivoronline.springboot_mockito.entities.Person;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.Map;

@Component
public class PersonRepository {

    //PROPERTY
    Map<Integer, Person> persons = new HashMap();

    //CONSTRUCTOR
    public PersonRepository() {
        persons.put(1, new Person(1, "John", 20));
        persons.put(2, new Person(2, "Bill", 30));
        persons.put(3, new Person(2, "Jack", 40));
    }

    //GET PERSON BY ID
    public Person getPersonById(Integer id) {
        return persons.get(id);
    }

}
```

MyController.java

```
package com.ivorononline.springboot_mockito_mock_injectinto_constructor.controllers;

import com.ivorononline.springboot_mockito_mock_injectinto_constructor.entities.Person;
import com.ivorononline.springboot_mockito_mock_injectinto_constructor.repositories.PersonRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    //PROPERTIES
    PersonRepository personRepository;

    //CONSTRUCTOR WITH AUTOWIRED INPUT PARAMETER (that gets stored into above Property)
    @Autowired
    public MyController(PersonRepository personRepository) {
        this.personRepository = personRepository;
    }

    //ENDPOINT
    @ResponseBody
    @RequestMapping("/GetPerson")
    public String getPerson(@RequestParam Integer id) {

        //GET PERSON
        Person person = personRepository.getPersonById(id);

        //GREET PERSON
        return "Hello " + person.name;
    }
}
```

MyControllerTest.java

```
package com.ivoronline.springboot_mockito_mock_injectinto_constructor.controllers;

import com.ivoronline.springboot_mockito_mock_injectinto_constructor.entities.Person;
import com.ivoronline.springboot_mockito_mock_injectinto_constructor.repositories.PersonRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.when;

@SpringBootTest
class MyControllerTest {

    //MOCK DEPENDENCY CLASS
    @Mock PersonRepository personRepository;

    @Test
    void getPerson() {

        //MOCK REPOSITORY'S METHOD: getPersonById(1)
        when(personRepository.getPersonById(1)).thenReturn(new Person(1, "Susan", 50));

        //INSTANTIATE CONTROLLER WITH MOCKED REPOSITORY
        MyController myController = new MyController(personRepository);

        //TEST CONTROLLER'S ENDPOINT
        String result = myController.getPerson(1);

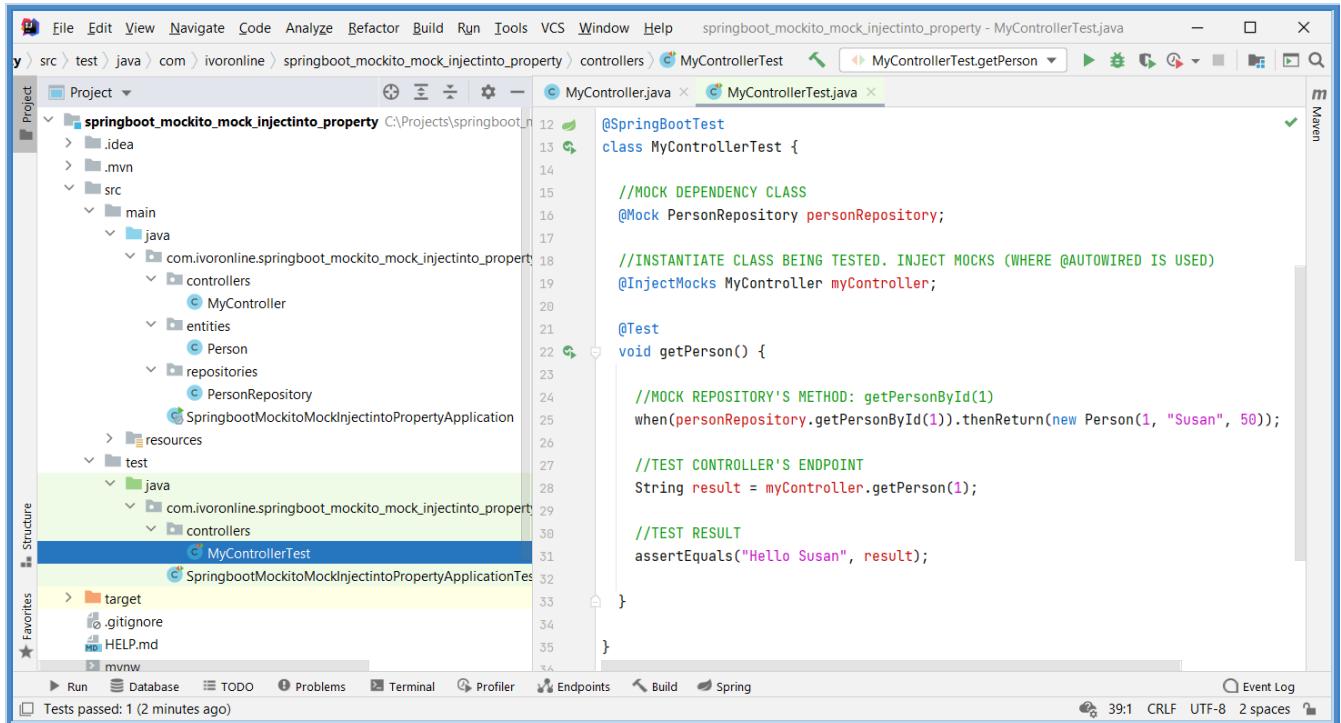
        //TEST RESULT
        assertEquals("Hello Susan", result);
    }
}
```

Result

<http://localhost:8080/GetPerson?id=1>



Run Test Class: MyControllerTest.java



pom.xml

```
<dependencies>

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

</dependencies>
```

6.5 Swagger

6.5.1 GET

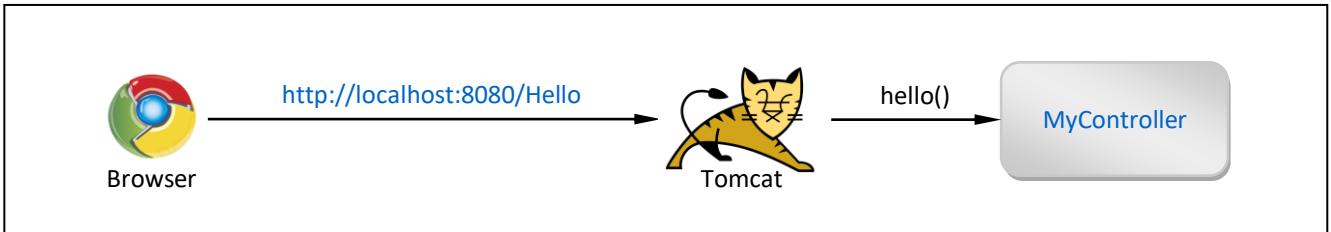
Info

[G]

- 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: <code>@RequestMapping</code> , Tomcat Server

Procedure

- **Create Project:** `springboot_swagger` (add Spring Boot Starters from the table)
- **Edit File:** `pom.xml` (add Swagger dependencies)
- **Edit Class:** `SpringbootSwaggerApplication.java` (add Swagger Annotation)
- **Create Package:** `controllers` (inside main package)
- **Create Class:** `MyController.java` (inside controllers package)

pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

SpringbootSwaggerApplication.java

```
package com.ivoronline.springboot_swagger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2
@SpringBootApplication
public class SpringbootSwaggerApplication {

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

MyController.java

```
package com.ivoronline.springboot_swagger.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/Hello")
    public String hello(@RequestParam String name, @RequestParam Integer age) {
        return name + " is " + age + " years old";
    }
}
```

Results

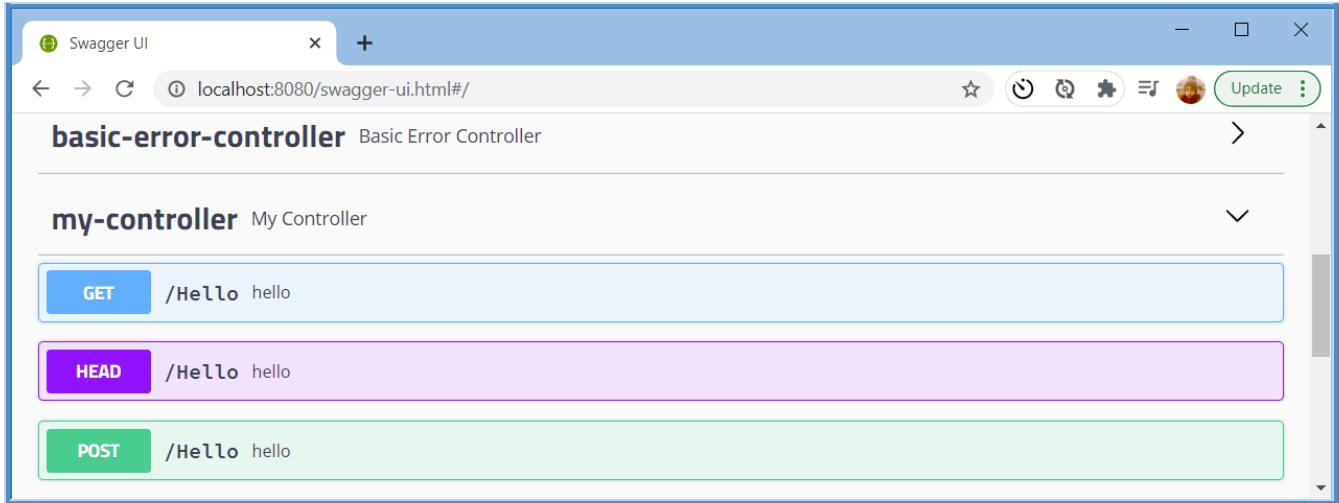
— <http://localhost:8080/swagger-ui.html#/>

- Expand: my-controller
- GET

— Try it out

- age: 20
- name: John
- Execute

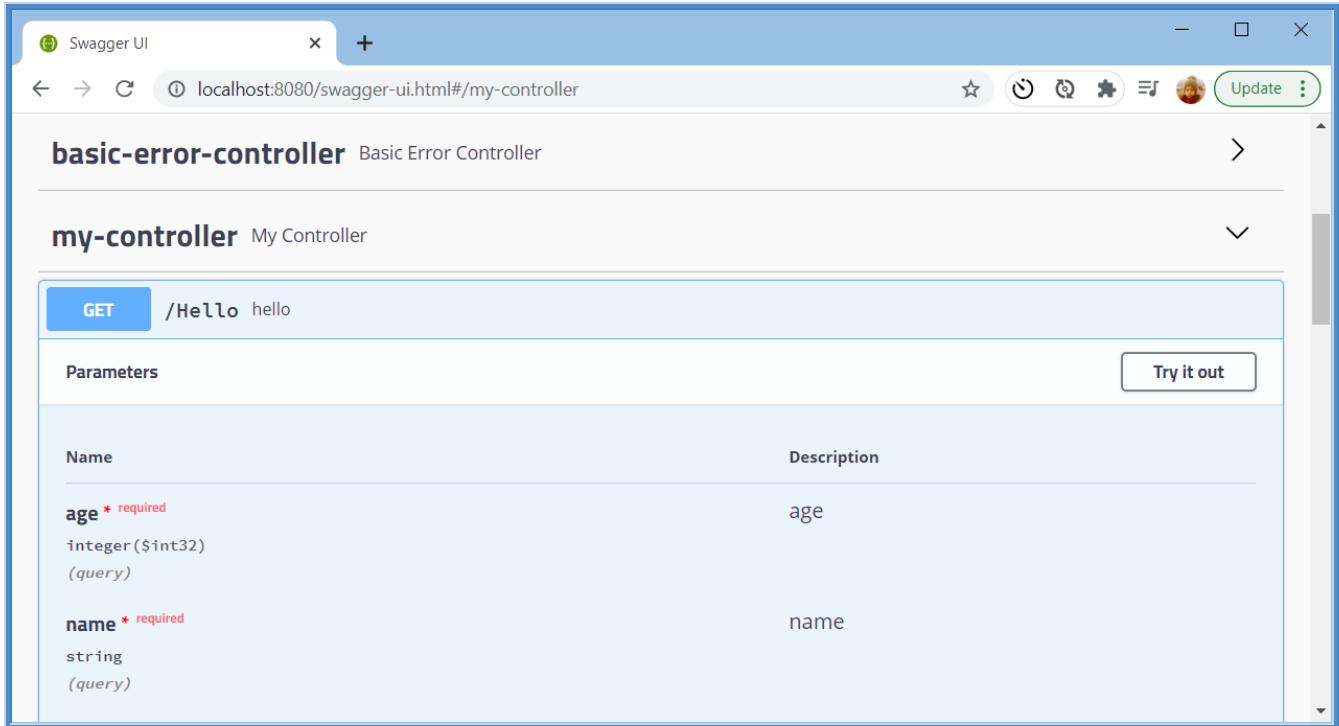
<http://localhost:8080/swagger-ui.html#/> - Expand: my-controller - GET



The screenshot shows the Swagger UI interface for a 'Basic Error Controller'. The 'my-controller' section is expanded, displaying three methods:

- GET** /Hello hello
- HEAD** /Hello hello
- POST** /Hello hello

Try it out



The screenshot shows the 'Try it out' view for the GET /Hello hello endpoint. It displays the following parameters:

Name	Description
age * required integer(\$int32) (query)	age
name * required string (query)	name

A 'Try it out' button is visible on the right side of the parameter table.

Enter Parameters - Execute

The screenshot shows the Swagger UI interface for executing a GET request to the endpoint `/Hello/hello`. The request parameters are defined as follows:

Name	Description
age * required integer(\$int32) (query)	age 20
name * required string (query)	name John

At the bottom, there are two buttons: "Execute" (blue) and "Clear" (white).

Responses

The screenshot shows the Swagger UI interface for viewing responses. It includes the following sections:

- Curl:** A command-line example:

```
curl -X GET "http://localhost:8080>Hello?age=20&name=John" -H "accept: */*"
```
- Request URL:** The URL for the request:

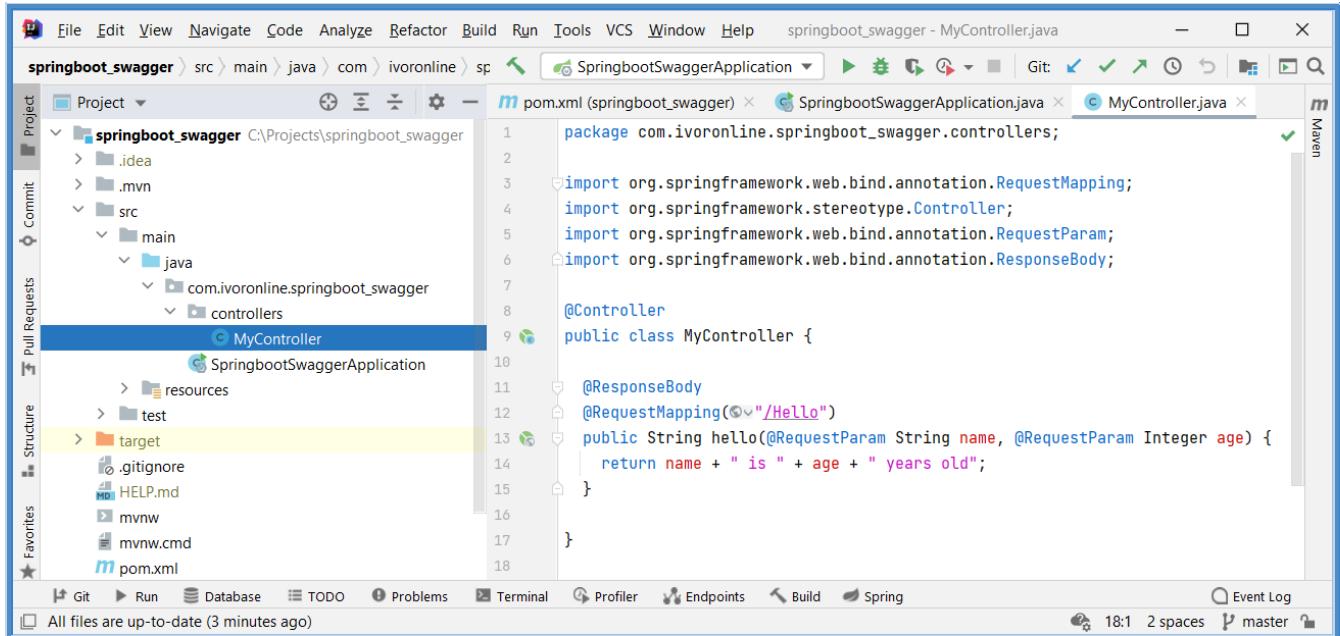
```
http://localhost:8080>Hello?age=20&name=John
```
- Server response:**
 - Code:** 200
 - Details:** Response body:

John is 20 years old

Download
 - Response headers:**

```
connection: keep-alive
content-length: 20
content-type: text/plain;charset=UTF-8
date: Mon, 15 Mar 2021 13:58:05 GMT
keep-alive: timeout=60
```
- Response content type:** `*/*`

Application Structure



pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>2.9.2</version>
    </dependency>

    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger-ui</artifactId>
        <version>2.9.2</version>
    </dependency>

</dependencies>
```

6.5.2 POST

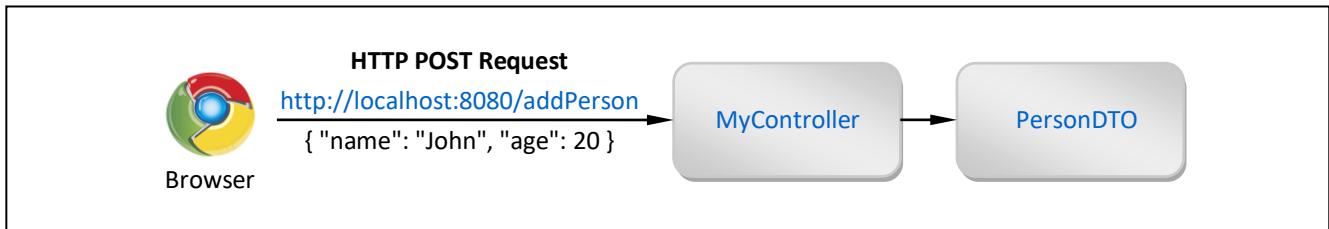
Info

[G]

- This tutorial shows how to use `@RequestBody` Annotation to **convert JSON** from HTTP Request into **Java Object** (DTO).

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** bootspring_http_requestbody (add Spring Boot Starters from the table)
- **Edit File:** pom.xml (add Swagger dependencies)
- **Edit Class:** SpringbootSwaggerApplication.java (add Swagger Annotation)
- **Create Package:** controllers (inside main package)
 - **Create Class:** MyController.java (inside package controllers)
- **Create Package:** DTO (inside main package)
 - **Create Class:** PersonDTO.java (inside package controllers)

pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

SpringbootSwaggerApplication.java

```
package com.ivoronline.springboot_swagger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2
@SpringBootApplication
public class SpringbootSwaggerApplication {

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

}
```

PersonDTO.java

```
package com.ivoronline.bootspring_http_requestbody.DTO;

public class PersonDTO {
    public Long id;
    public String name;
    public Integer age;
}
```

MyController.java

```
package com.ivoronline.bootspring_http_requestbody.controllers;

import com.ivoronline.bootspring_http_requestbody.DTO.PersonDTO;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/AddPerson")
    public String addPerson(@RequestBody PersonDTO personDTO) {

        //GET DATA FROM PersonDTO
        String name = personDTO.name;
        Integer age = personDTO.age;

        //RETURN SOMETHING
        return name + " is " + age + " years old";

    }
}
```

Results

- <http://localhost:8080/swagger-ui.html#/>
 - Expand: my-controller
 - POST
- Try it out
 - Copy body
 - Execute

Body

(option: raw)

```
{  
    "name" : "John",  
    "age"   : 20  
}
```

<http://localhost:8080/swagger-ui.html#/> - Expand: my-controller - POST

The screenshot shows the Swagger UI interface. In the top navigation bar, the URL is `localhost:8080/swagger-ui.html#/`. The main content area is titled "basic-error-controller" and "Basic Error Controller". Below it, under "my-controller" (My Controller), there are three method entries:

- GET** /AddPerson addPerson (highlighted in blue)
- HEAD** /AddPerson addPerson (highlighted in purple)
- POST** /AddPerson addPerson (highlighted in green)

Try it out

The screenshot shows the "Try it out" view for the POST /AddPerson addPerson method. The URL in the browser is `localhost:8080/swagger-ui.html#/my-controller/addPersonUsingPOST`. The method is highlighted in green. The "Parameters" section shows one parameter:

Name	Description
personDTO * required (body)	personDTO

The "Example Value" field contains the JSON object from the previous screenshot:

```
{  
    "name" : "John",  
    "age"   : 20  
}
```

A "Try it out" button is visible in the top right corner of the form.

Copy body - Execute

The screenshot shows the Swagger UI interface for a POST request to '/AddPerson'. The 'Parameters' section displays a required parameter 'personDTO' with a JSON example value:

```
{  
  "name" : "John",  
  "age" : 20  
}
```

The 'Content-Type' dropdown is set to 'application/json'. A large blue 'Execute' button is visible at the bottom.

Responses

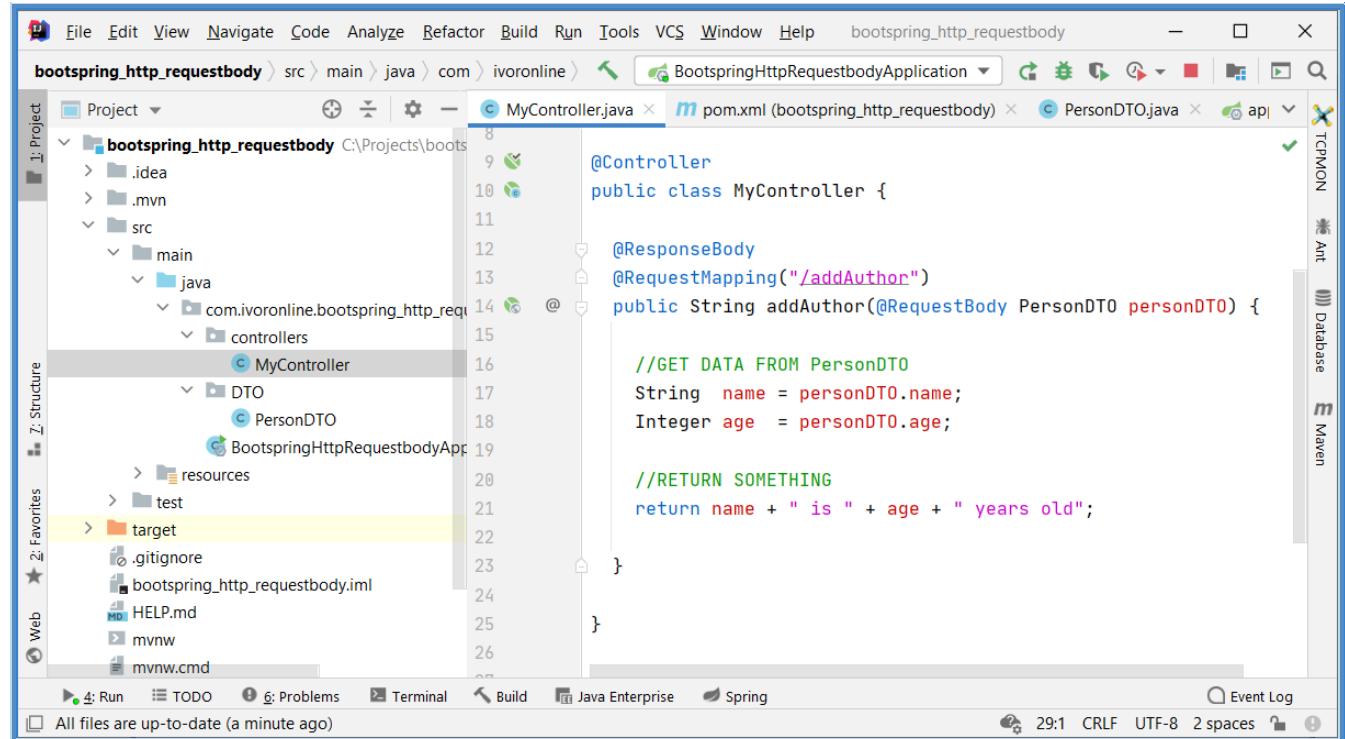
The 'Responses' panel shows a successful response (200) with the following details:

- Curl:**

```
curl -X POST "http://localhost:8080/AddPerson" -H "accept: */*" -H "Content-Type: application/json" -d "{ \"name\" : \"John\", \"age\" : 20}"
```
- Request URL:** <http://localhost:8080/AddPerson>
- Server response:**
 - Code:** 200
 - Response body:** John is 20 years old
 - Download** button
- Response headers:**

```
connection: keep-alive  
content-length: 20  
content-type: text/plain; charset=UTF-8  
date: Mon, 15 Mar 2021 15:10:24 GMT  
keep-alive: timeout=60
```

Application Structure



pom.xml

```
<dependencies>

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

    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>2.9.2</version>
    </dependency>

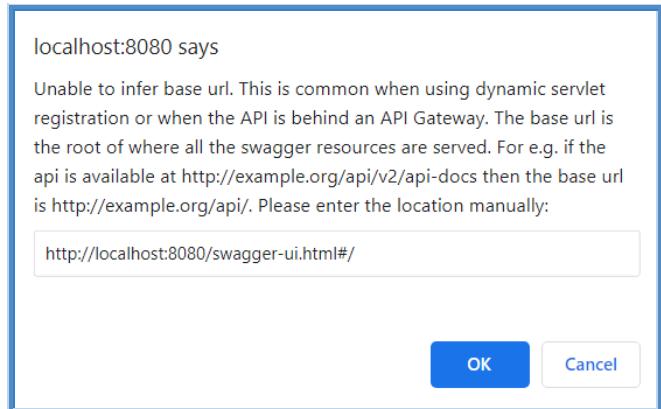
    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger-ui</artifactId>
        <version>2.9.2</version>
    </dependency>

</dependencies>
```

6.5.3 Error - Unable to infer base URL

Error Message

Error Message



Solution

- Add @EnableSwagger2 Annotation as shown below.

SpringbootSwaggerApplication.java

```
package com.ivoronline.springboot_swagger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2
@SpringBootApplication
public class SpringbootSwaggerApplication {

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