WebServices: A simple REST WS

Created by Lasse Jenssen

Home

Spring Initializer

http://start.spring.io

Spring Initializer

Command Line w/Curl

```
curl -G https://start.spring.io/starter.tgz \
  -d javaVersion=8 \
  -d dependencies=web \
  -d name=SpringRestDemo \
  -d groupId=no.hvl.dat152 \
  -d artifactId=SimpleRestDemo \
  -d packageName=no.hvl.dat152.demo \
  -d type=maven-project \
  -o simple-rest-demo.tar.gz
```

```
bash-3.2$ curl -G https://start.spring.io/starter.tgz \
> -d javaVersion=8 \
> -d dependencies=web \
> -d name=SpringRestDemo \
> -d groupId=no.hvl.dat152 \
> -d artifactId=SimpleRestDemo \
> -d packageName=no.hvl.dat152.simple-rest-demo \
> -d type=maven-project \
> -o simple-rest-demo.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
                                 Dload  Upload   Total   Spent    Left  Speed
100 61670  100 61670    0     0   192k      0 --:--:-- --:--:-- --:--:--  198k

bash-3.2$ ls -ltr
total 128
-rw-r--r--  1 lassejenssen  staff  61670 Oct 25 08:16 simple-rest-demo.tar.gz
```

```
bash-3.2$ tar xvf simple-rest-demo.tar.gz
x mvnw.cmd
x mvnw
x pom.xml
x .mvn/
x .mvn/wrapper/
x .mvn/wrapper/maven-wrapper.properties
x .mvn/wrapper/maven-wrapper.jar
x src/
x src/main/
x src/main/resources/
x src/main/resources/templates/
x src/main/resources/static/
x src/main/resources/application.properties
x src/main/java/
x src/main/java/no/
x src/main/java/no/hvl/
x src/main/java/no/hvl/dat152/
x src/main/java/no/hvl/dat152/demo/
x src/main/java/no/hvl/dat152/demo/SpringRestDemoApplication.java
x src/test/
x src/test/java/
x src/test/java/no/
x src/test/java/no/hvl/
```

```
mvn clean package -D skipTests spring-boot:run
```

Spring Boot and pom.xml

Default: Tomcat Embedded Server

```
<parent>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-parent</artifactid>
  <version>2.7.5</version>
  <relativepath></relativepath>
</parent>

<dependencies>
  <dependency>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-starter-web</artifactid>
  </dependency>

  <dependency>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-starter-test</artifactid>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Spring Boot & REST WebService

Demo

1. Starting WebService without RestController
2. Change from Tomcat to Jetty Embedded Server
3. Creating a simple rest controller (hello)
4. Calling rest web service :

```
curl -G http://localhost:8080/hello?name=Bill
```

Spring Boot and pom.xml

Changing: Using Jetty Embedded

```
<dependencies>
  <dependency>
      <groupid>org.springframework.boot</groupid>
      <artifactid>spring-boot-starter-web</artifactid>
      <exclusions>
         <exclusion>
            <groupid>org.springframework.boot</groupid>
            <artifactid>spring-boot-starter-tomcat</artifactid>
         </exclusion>
      </exclusions>
   </dependency>

   <dependency>
      <groupid>org.springframework.boot</groupid>
      <artifactid>spring-boot-starter-jetty</artifactid>
   </dependency>

  ...
</dependencies>
```

```java
package no.hvl.dat152.demo.controller;

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

@RestController
public class HelloController {

  @RequestMapping(value = "/hello", produces = MediaType.TEXT_PLAIN_VALUE)
  public String hello(String name) {
    return "Hello to " + name;
  }

}
```

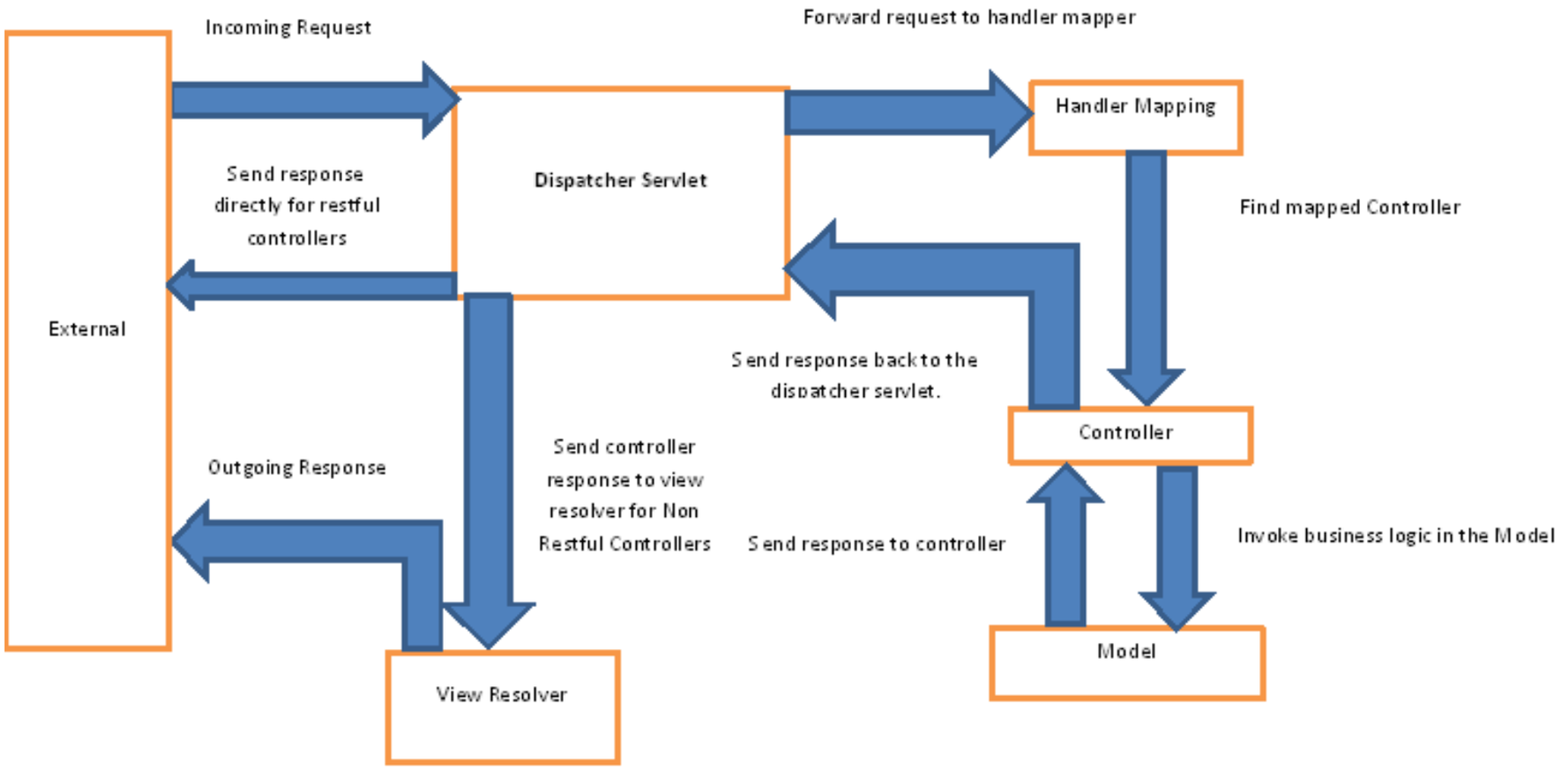


Fig 1 MVC Architecture flow

```java
package no.hvl.dat152.demo.controller;

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

@RestController
public class HelloController {

  @RequestMapping(value = "/hello", produces = MediaType.TEXT_PLAIN_VALUE)
  public String hello(String name) {
    return "Hello to " + name;
  }

}
```

```
bash-3.2$ curl -G http://localhost:8080/hello?name=Bill
Hello to Bill
```

MediaType.APPLICATION_JSON_VALUE

Demo

1. Introducing a Greeting class w/Counter and Greeting content.

2. Create new WebService "Greeting" configured to return Json.

3. Calling rest web service :

```
curl -G http://localhost:8080/greeting?name=Bill
```

```java
package no.hvl.dat152.demo.model;

public class Greeting {
   private final long id;
   private final String content;

   public Greeting(long id, String content) {
      this.id = id;
      this.content = content;
   }

   public long getId() {
      return id;
   }

   public String getContent() {
      return content;
   }
}
```

```java
@RestController
public class GreetingController {

  private static final String template = "Hello, %s!";
  private final AtomicLong counter = new AtomicLong();

  @RequestMapping(value = "/greeting",
                  produces = MediaType.APPLICATION_JSON_VALUE)
  public Greeting greeting(
    @RequestParam(value = "name", defaultValue = "World") String name) {

    return
      new Greeting(counter.incrementAndGet(), String.format(template, name));
  }

}
```

```java
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping(value = "/greetings/{name}",
                    produces = MediaType.APPLICATION_JSON_VALUE)
    public Greeting greetings(
        @PathVariable(value = "name") Optional< String> name) {

        return
            new Greeting(counter.incrementAndGet(),
                    String.format(template, name.orElseGet(() -> "World")));
    }

}
```

```java
@RestController
public class GreetingController {

   private static final String template = "Hello, %s!";
   private final AtomicLong counter = new AtomicLong();

   @RequestMapping(value = "/greetings/{name}",
                   produces = MediaType.APPLICATION_JSON_VALUE)
   public Greeting greetings(
      @PathVariable(value = "name") Optional< String> name) {

      return
         new Greeting(counter.incrementAndGet(),
                   String.format(template, name.orElseGet(() -> "World")));
   }

}
```

```
bash-3.2$ curl -G http://localhost:8080/greeting?name=Bill
{"id":1,"content":"Hello, Bill!"}bash-3.2$
```

```
bash-3.2$ curl -G http://localhost:8080/greeting?name=Bill | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    33    0    33    0     0   1465      0 --:--:-- --:--:-- --:--:--  3000
{
   "content" : "Hello, Bill!",
   "id" : 2
}
```

MediaType.APPLICATION_JSON_VALUE

Why JSON over XML?

- JSON is **faster** because it is designed specifically for data interchange.
  - JSON encoding is terse (requires less bytes)
  - JSON parsers are less complex (requires less processing time and memory overhead)
  - XML is slower, because it is designed for a lot more than just data interchange
- JSON **less complex** (easier to read and handle)

Web Services: WADL vs WSDL

Web Application Description Language.

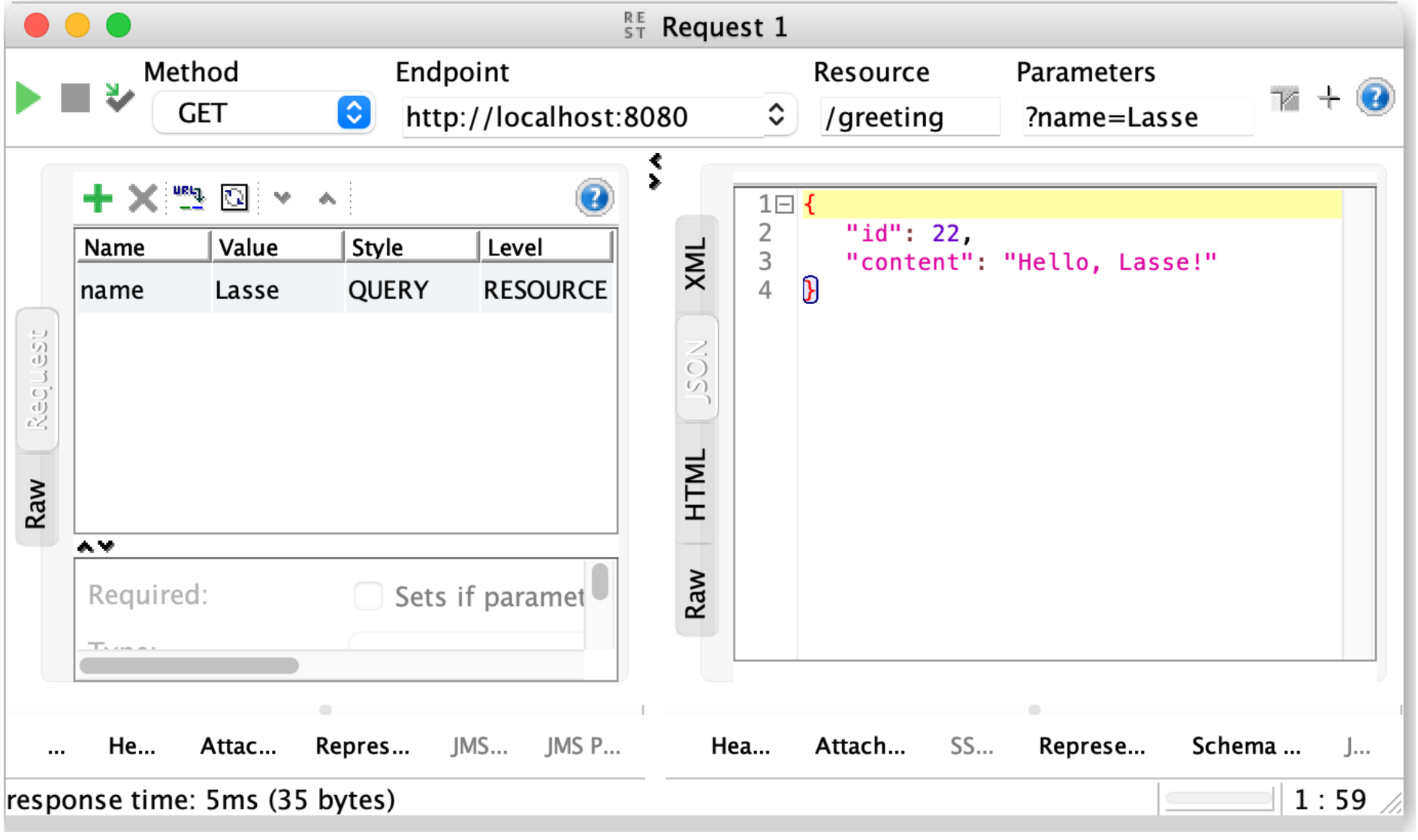
- A machine readable XML description of HTTP based web-services.
- Lightweight.
- Easier to understand, and easier to write than WSDL.
- Some implementations of JAX-RS includes tools to auto generate WADL:

  (For instance Jersey, Apache CFX)
- Spring has Spring REST Doc (does not produce WADL, but documentation)

Note! We are not going to gernerate WADL desciptions or use Spring REST Doc.

Demo: Testing our Greeting API

SoapUI



Unit Testing WebServices (JUnit5)

Testing REST API in Spring Boot

- JUnit is one of the most popular unit-testing frameworks in the Java ecosystem.
- JUnit 5: Goal of supporting new features in Java 8 and above.
- Direct support to run Unit tests on the JUnit Platform in Eclipse and IntelliJ.
- ... or just run by *"mvn test"* (or invoked in other goals)
- To skip test in Maven: *mvn package -DskipTests* (use with care)

Source: https://www.baeldung.com/junit-5

Unit Testing WebServices (JUnit5)

JUnit 5

The Architecture of Junit5 can be divided into 3 different projects:

- **JUnit Platform**:
  - provides an API to launch the tests from either the IDE's, Build Tools or Console.
  - defines a TestEngine API which enables development of new Testing Frameworks on top of the Platform.
- **JUnit Jupiter**: provides a TestEngine implementation to run JUnit5 tests on the platform.
- **JUnit Vintage**: provides a TestEngine implementation to support backward compatibility for tests written with Junit3 and Junit4

Maven Dependencies

Unit Testing WebServices

```
<dependency>
    <groupid>org.junit.jupiter</groupid>
    <artifactid>junit-jupiter-engine</artifactid>
    <version>5.8.1</version>
    <scope>test</scope>
</dependency>
```

Maven Dependencies: Spring Boot Starter Test (always included)

Unit Testing WebServices (JUnit5)

```
<dependency>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-starter-test</artifactid>
    <scope>test</scope>
</dependency>
```

Demo: Example Test Code

Unit Testing WebServices (JUnit5)

Extend the DemoApplicationTests class ...

1. Create a test for failing web service ("morning" - not existing).
2. Create a test for sucessfull web service call ("greeting").
3. Create a test checking content type is Json.
4. Create a test checking increasing counter.

Demo: Example Test Code

Unit Testing WebServices (JUnit5)

```
1  package no.hvl.dat152.demo;
2
3  import org.junit.jupiter.api.Test;
4  import org.springframework.boot.test.context.SpringBootTest;
5
6  @SpringBootTest
7  class SpringRestDemoApplicationTests {
8
9    @Test
10   void contextLoads() {
11   }
12
13 }
```

Unit Testing WebServices (JUnit5)

Adding configuration for using a random port

```
1  package no.hvl.dat152.demo;
2
3  import org.junit.jupiter.api.Test;
4  import org.springframework.boot.test.context.SpringBootTest;
5
6  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
7  class SpringRestDemoApplicationTests {
8
9    @Test
10   void contextLoads() {
11   }
12
13 }
```

Unit Testing WebServices (JUnit5)

Create a test for failing web service ("morning" - not existing)

```java
package no.hvl.dat152.demo;

import ...
import org.junit.jupiter.api.Test;
import org.springframework.http.ResponseEntity;
import org.springframework.boot.test.web.client.TestRestTemplate;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class SpringRestDemoApplicationTests {
   @Autowired
   private TestRestTemplate template;

   @Test
   @DisplayName("Not existing URI (/morning)")
   public void checkWrongGreeting() throws Exception {
      ResponseEntity< Greeting> response =
               template.getForEntity("/morning", Greeting.class);

      assertTrue(response.getStatusCode().isError(),
               () -> "Not existing URI should return error status");
   }

}
```

Unit Testing WebServices (JUnit5)

Assertion vs Assumption

Assertion:

- The process of making sure some condition is met.

Assumption:

- Check the prerequisites of the test.
- If they are not available or do not have the expected value, there's no point in continuing with the test.

Unit Testing WebServices (JUnit5)

Demo: Assumption

```
1  @Test
2  @DisplayName("Wrong Path (/morning) with Assumption")
3  public void checkWrongGreetingAssumption() throws Exception {
4     assumeTrue("DEV".equals(System.getenv("APP_MODE")),
5                () -> "Aborting test: APP_MODE not set to 'DEV'");
6
7     ResponseEntity< Greeting> response =
8                 template.getForEntity("/morning", Greeting.class);
9
10    assertTrue(response.getStatusCode().isError());
11 }
```

Unit Testing WebServices (JUnit5)

Create a test for sucessfull web service call ("greeting")

```
1  @Test
2  @DisplayName("Check Greeting OK")
3  public void checkOkGreeting() throws Exception {
4     String name = "bill";
5     String greetingTemplate = "Hello, %s!";
6     String expectedValue = String.format(greetingTemplate, name);
7
8     ResponseEntity< Greeting> response = template.getForEntity("/greeting?name=
9
10    assertTrue(response.getStatusCode().is2xxSuccessful());
11    assertEquals(expectedValue, response.getBody().getContent());
12 }
```

Unit Testing WebServices (JUnit5)

Create a test checking content type is JSON

```
@Test
@DisplayName("Check content-type is JSON")
public void checkContentTypeJSON() throws Exception {
    String expectedType = MediaType.APPLICATION_JSON_VALUE;

    ResponseEntity< Greeting> response = template.getForEntity("/greeting", Gre

    assertEquals( expectedType,
                  response.getHeaders().getContentType().toString());
}
```

Unit Testing WebServices (JUnit5)

Create a test checking increasing counter

```
1 @Test
2 public void checkOkCounter() throws Exception {
3   ResponseEntity< Greeting> response1 = template.getForEntity("/greeting", Gree
4   ResponseEntity< Greeting> response2 = template.getForEntity("/greeting", Gree
5
6   long expectedValue = response1.getBody().getId() + 1;
7
8   assertEquals(expectedValue, response2.getBody().getId());
9 }
```

Unit Testing WebServices (JUnit5)

assumingThat and assertAll

```
1  @Test
2  @DisplayName("Check Greeting Content)")
3  public void checkGreetingContent() throws Exception {
4     String name = "bill";
5     String greetingTemplate = "Hello, %s!";
6     String expectedValue = String.format(greetingTemplate, name);
7
8     ResponseEntity< Greeting> response = template.getForEntity("/greeting?name=
9     Greeting greeting = response.getBody();
10
11    assumingThat(greeting != null,
12          () -> assertAll("Checking Greeting",
13             () -> assertEquals(expectedValue, greeting.getContent()),
14             () -> assertTrue(greeting.getId()>0)
15          )
16    );
17
18 }
```

Annotations

- **@DisplayName** defines a custom display name for a test class or a test method
- **@Disable**: disables a test class or method (previously @Ignore)
- **@Tag** declares tags for filtering tests
- **@BeforeEach** denotes that the annotated method will be executed before each test method (previously @Before)
- **@AfterEach** denotes that the annotated method will be executed after each test method (previously @After)
- **@BeforeAll** denotes that the annotated method will be executed before all test methods in the current class (previously @BeforeClass)
- **@AfterAll** denotes that the annotated method will be executed after all test methods in the current class (previously @AfterClass)

Demo: Run test in Eclipse (look at output).

Unit Testing WebServices (JUnit5)

@Tag

```
1  @Test
2  @DisplayName("Check Greeting Content)")
3  @Tag("UnitTests")
4  public void checkGreetingContent() throws Exception {
5     String name = "bill";
6     String greetingTemplate = "Hello, %s!";
7     String expectedValue = String.format(greetingTemplate, name);
8
9     ResponseEntity< Greeting> response = template.getForEntity("/greeting?name=
10    Greeting greeting = response.getBody();
11
12    assumingThat(greeting != null,
13          () -> assertAll("Checking Greeting",
14             () -> assertEquals(expectedValue, greeting.getContent()),
15             () -> assertTrue(greeting.getId()>0)
16          )
17    );
18
19 }
```

Unit Testing WebServices (JUnit5)

@Tag

```
mvn test -D groups=UnitTests

...
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.510 s
[INFO] Finished at: 2022-10-30T21:01:12+01:00
[INFO] ------------------------------------------------------------------------
```

Unit Testing WebServices

Exception Testing

```
1  @Test
2  @DisplayName("Can not divide on zero")
3  void checkExceptionDivisionByZero() {
4     String expectedErrorMessage = "/ by zero";
5
6     Throwable exception = assertThrows(ArithmeticException.class,
7        () -> { int result = 10/0; });
8
9     assertEquals(expectedErrorMessage, exception.getMessage());
10 }
11
12 @Test
13 @DisplayName("Illegal argument (Integer conversion)")
14 void checkExceptionIllegalArgument() {
15    String str = null;
16    assertThrows(IllegalArgumentException.class,
17          () -> { Integer.valueOf(str); },
18          () -> "Trying to covert null as Integer should result in IllegalArgum
19 }
```

Western Norway University of Applied Sciences

Next

Home