# Spring Boot
# Unit Testing using JUnit + Mockito :-

**Mock:--** It is a component which acts like
=> Client to make request.
=> Behaves as Container.
=> Supports object creation and Destroy.
=> Uses Proxy design Pattern.
=> Supports HTTP calls, No need of using any client (**RestTemplate, HttpClient**… etc).

Mocking is implemented using
1. EasyMock
2. Mockito

=>Spring boot uses JUnit 4 + Mockito 2 Integration for UnitTesting of Applications.
=>Here Mockito supports,
  a. HTTP Request Creation.
  b. Making HTTP Client (GET/POST…).
  c. Execute code using Proxies.
  d. GetResult and store in HTTP Response objects.

=>Here JUnit is used for
  a. Writing Test cases with annotations.
  b. Verify Result using assert methods (Return PASS/FALL).

**Step#1:-** Create one Spring Starter Project and define RestController with methods
and URLs added.

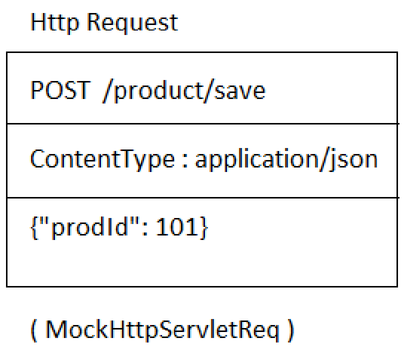
**Step#2:-** Define UnitTesting class (Test case) under src/test/java.

**a.** Autowire MockMvc (C) Object[acts as Http Client and supports container communication].

**b.** Call perform method to make Http Request but construct Request object at same time that returns as **MockHttpServletRequest** (use **RequestBuilder** static methods
and call).

**Ex:-** mockMvc.perform(post("/product/save").header("Content-Type", "application/json").content("{\"prodId\":101,....}")).andReturn();

**c.** Here andReturn() submits HttpRequest above one looks like.

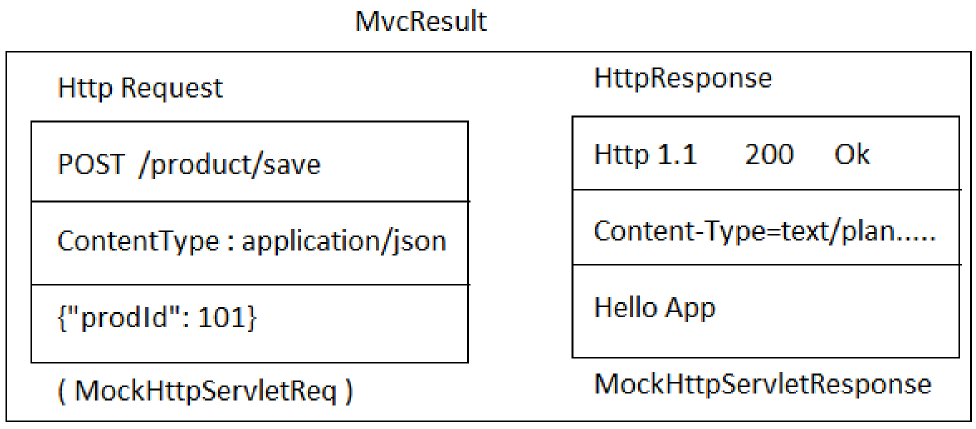


**d.** Both Request and Response objects are stored in “MvcResult” as,

a. MockHttpServletRequest

b. MockHttpServletResponse.

It looks like



**e.** Enable this Mock and Test process using Annotations :

@RunWith(SpringRunner.class)
@WebMvcTest

**Code:-- RestController:--**

```
package com.app.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/emp")
public class EmployeeRestController {

        @GetMapping("/data")
        public String getData() {
            return "Hello";
        }
}
```

**Code:- Test class:--**

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
```

```
@RunWith(SpringRunner.class)
@WebMvcTest
@SpringBootTest
public class JUnitMockitoApplicationTests {

        @Autowired
        private MockMvc mockMvc;

        @Test
        public void testEmpData() throws Exception {
                MvcResult
result=mockMvc.perform(get("/emp/data")).andReturn();
                //MockHttpServletRequest req =result.getRequest();
                MockHttpServletResponse  resp =result.getResponse();
                assertEquals("Hi", resp.getContentAsString());
        }
}
```

=>Right click => Run as => JUnit Test

## Spring Boot with JUnit and Mockito:--

**##Testing : ReST CURD Application ##**

=>To implement Unit Testing, we need to follow 4 steps. Given below,

1. Construct Http Request using **RequestBuilders**.
2. Execute Http Request using **MockMvc**
3. Store Response along with Request in **MvcResult**.
4. Use assert API (JUnit) to verify actual details with expected values.

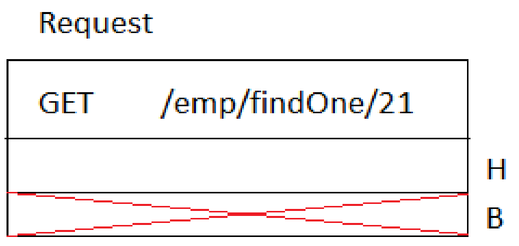
**Step#1:-** To construct Request Object use RequestBuilder and call static method
(Http Method Type).

MockMvcRequestBuilders.post(…)….;

=>It returns HttpServletRequest object.

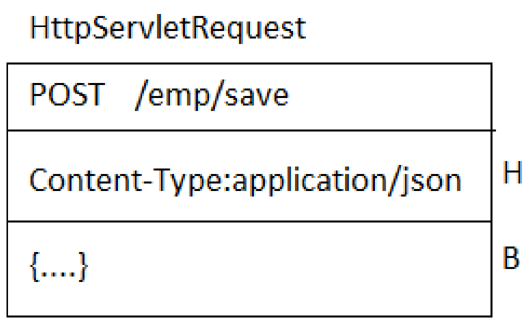
MockMvcServletRequestBuilder

**Ex#1 (GET):-**



MockHttpServletRequestBuilder request =
MockMvcRequestBuilders.get ("/emp/findOne/21");

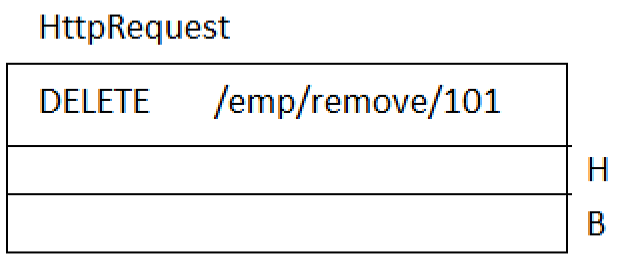
**Ex#2 (POST):--**



=>Equal code of above Diagram is

```
MockHttpServletRequestBuilder   request = MockMvcRequestBuilders
        .post("/emp/save")
        .header("Content-Type", "application/json")
        //.contentType("application/json")
        //contentType(MediaType.APPLICATION_JSON)
        .content("{...}");
```

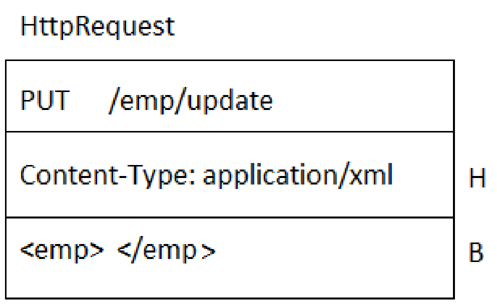
**Ex#3 (DELETE):-**



=>Equal code

```
MockHttpServletRequestBuilder   request =
MockMvcRequestBuilders.delete("/emp/remove/101");
```

**EX#4 (PUT):-**



=>Equal code

```
MockHttpServletRequestBuilder   request = MockMvcRequestBuilders
        .post("/emp/save")
        .contentType("application/xml")
        .content("<emp>...</emp>");
```

**Step#2:-** Execute Request call using MockMvc.

```
MvcResult result = mockMvc.perform(request).andReturn();
```

**Step#3:-** Get Request/Response Object from Mvc.

```
MockHttpServletRequest   req = result.getRequest();
MockHttpServletResponse   resp = result.getResponse();
```

**Project Creation step by Step:--**

Step#1:- Define one CURD application using one database and RestController.

Step#2:- Check all operations using POSTMAN Screen.

Step#3:- Define one test profile for properties or yml.
=>application-test.properties

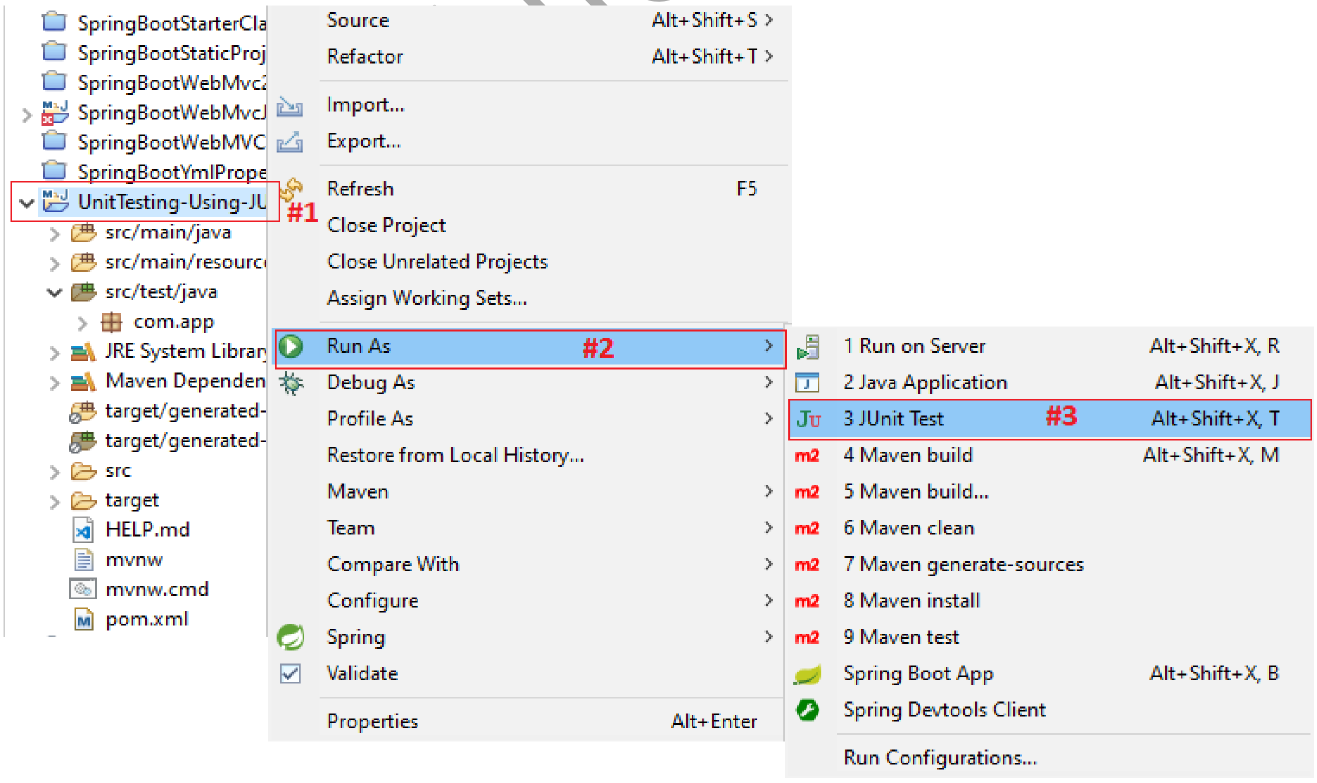
Step#4:- Define one class under src/test/java folder and apply annotations.

Step#5:- Apply below annotations over test class.
@SpringBootTest(webEnvironment=WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource("classpath:application-test.properties")

Step#6:- Use MockMvc dependency (autowire) in Test class.

Step#7:- Define @Test methods under Test class.

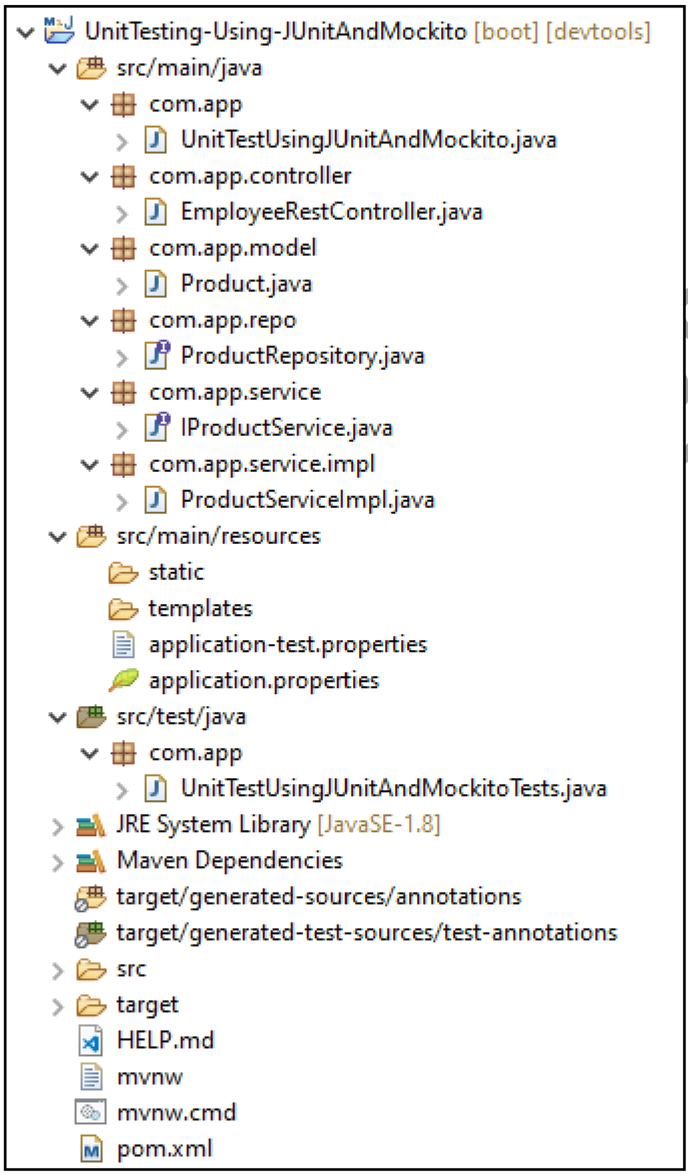
Step#8:- Run Test class using JUnit Test.
=>Right click on Project > Run As > JUnit Test.



**NOTE:--**

**1>@TestPropertySource:-** It is used to load properties/yml file into UnitTest
**2>@AutoConfigureMockMvc:-** It Define all Mock Beans related to environment
  (Ex: Datasource, ConnectionPool, Cache…).
**3>@SpringBootTest:-** It define beans and injects them based on relations (Objects for: RestControllers, Services, Repos…etc).
**4>@WebMvcTest:-** Works only for @RestControllers without service and other dependencies.

**1. Folder Structure of JUnit + Mockito Testing:--**



```
UnitTesting-Using-JUnitAndMockito [boot] [devtools]
  src/main/java
    com.app
      UnitTestUsingJUnitAndMockito.java
    com.app.controller
      EmployeeRestController.java
    com.app.model
      Product.java
    com.app.repo
      ProductRepository.java
    com.app.service
      IProductService.java
    com.app.service.impl
      ProductServiceImpl.java
  src/main/resources
    static
    templates
    application-test.properties
    application.properties
  src/test/java
    com.app
      UnitTestUsingJUnitAndMockitoTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

**pom.xml:--**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.8.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>UnitTesting-Using-JUnitAndMockito</artifactId>
	<version>1.0</version>
	<name>UnitTesting-Using-JUnitAndMockito</name>
	<description>Spring Boot Test Application</description>

	<properties>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<!-- <optional>true</optional> -->
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>com.oracle</groupId>
			<artifactId>ojdbc6</artifactId>
			<version>11.2.0</version>
		</dependency>
		<!-- <dependency>
			<groupId>org.mockito</groupId>
			<artifactId>mockito-all</artifactId>
			<scope>test</scope>
		</dependency> -->
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
</project>
```

**Coding Step by Step:--**

**Step#1: In “application.properties” & “application-test.properties” add bellow code:-**

```
server.port=2019
##DataSource##
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql:thin:@localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root
#Spring Data JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL55Dialect
spring.jpa.properties.hibernate.format_sql=true
```

**Step#2: Model class (Product.class):--**

```
package com.app.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Data
@Table
public class Product {

        @Id
        @GeneratedValue
        private Integer prodId;
        private String prodCode;
        private Double prodCost;
        private String vendorCode;
}
```

**Step#3: Repository Interface:--**

```
package com.app.repo;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> { }
```

**Step#4: Service Interface:--**

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService  {

        public Integer saveProduct(Product p);
        public void deleteProduct(Integer prodId);
        public Product getProductById(Integer prodId);
        public List<Product> getAllProducts();
        public boolean isProductExist(Integer id);
}
```

**Step#5: ServiceImpl class:--**

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;

@Service
public class ProductServiceImpl implements IProductService{

        @Autowired
        private ProductRepository repo;
```

```
	public Integer saveProduct(Product p) {
		p=repo.save(p);
		Integer prodId=p.getProdId();
		return prodId;
	}

	public void deleteProduct(Integer prodId) {
		repo.deleteById(prodId);
	}

	public Product getProductById(Integer prodId) {
		Optional<Product> p=repo.findById(prodId);
		if(p.isPresent()) {
			return p.get();
		}else {
			return new Product();
		}
	}
	public List<Product> getAllProducts() {
		List<Product> prods=repo.findAll();
		return prods;
	}
	@Override
	public boolean isProductExist(Integer id) {
		return repo.existsById(id);
	}
}
```

**Step#6: RestController:--**

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Product;
import com.app.service.IProductService;

@RestController
@RequestMapping("/product")
public class EmployeeRestController {

        @Autowired
        private IProductService service;
  //1. Post Method
        @PostMapping("/register")
        public ResponseEntity<?> saveProduct(@RequestBody Product product)
{
                ResponseEntity<?> response=null;
                try {
                        Integer stdId=service.saveProduct(product);
           response = new ResponseEntity<String>(stdId+"-Inserted",
HttpStatus.OK);
                } catch (Exception e) {
                        e.printStackTrace();
           response = new
responseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
                }
                return response;
        }
//2. Get Method
        @GetMapping("/get/{id}")
        public ResponseEntity<?> showOneProducts(@PathVariable Integer id) {
                ResponseEntity<?> response=null;
                boolean exist=service.isProductExist(id);
                if(exist) {
                        Product s=service.getProductById(id);
                        response=new ResponseEntity<Product>(s, HttpStatus.OK);
                }  else {
                        response=new
ResponseEntity<>(HttpStatus.NO_CONTENT);
                }
                return response;
        }
```

**//3. Get Method for all data**

```
        @GetMapping("/all")
        public ResponseEntity<?> showAllProducts() {
                ResponseEntity<?> response=null;
                List<Product> Products=service.getAllProducts();
                if(Products!=null && !Products.isEmpty()) {
         response=new ResponseEntity<List<Product>>(Products,
HttpStatus.OK);
                } else {
                        response=new
ResponseEntity<>(HttpStatus.NO_CONTENT);
                }
                return response;
        }
```

**//4. Delete Method**

```
        @DeleteMapping("/delete/{id}")
        public ResponseEntity<?> deleteProduct(@PathVariable Integer id) {
                ResponseEntity<?> response=null;
                boolean exist=service.isProductExist(id);
                if(exist) {
                        service.deleteProduct(id);
        response=new ResponseEntity<String>(id+"-Removed", HttpStatus.OK);
                }  else {
```

```
                response=new ResponseEntity<String>("Product NOT FOUND",
                        HttpStatus.BAD_REQUEST);
            }
            return response;
        }
```

**//5. Edit method**

```
        @PutMapping("/edit")
        public ResponseEntity<?> editProduct(@RequestBody Product product)
{
            ResponseEntity<?> response=null;
            Integer id=product.getProdId();
            boolean exist=service.isProductExist(id);
            if(exist) {
                service.saveProduct(product);
        response = new ResponseEntity<String>(id+"-Updated", HttpStatus.OK);
            }else {
              response = new ResponseEntity<String>("Product NOT FOUND",
                                HttpStatus.BAD_REQUEST);
            }
            return response;
        }
}
```

**Step#7:JunitAndMockitoTests class:--**

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.post;
```

```
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.put;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.
AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment= SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource(locations = "classpath:application-test.properties")
public class UnitTestUsingJUnitAndMockitoTests {

	@Autowired
	private MockMvc mockMvc;
//1. Post Method case
	@Test
	public void testProductSave() throws Exception {
	  MvcResult result = mockMvc.perform(post("/product/register")
		.contentType(MediaType.APPLICATION_JSON)
.content("{\"prodCode\":\"ABCD\",\"prodCost\":88.55,\"vendorCode\":
\"V11\"}"))
		.andReturn();
	  MockHttpServletResponse   resp = result.getResponse();
	  assertEquals(HttpStatus.OK.value(), resp.getStatus());
```

```
            System.out.println(resp.getContentAsString());
            assertNotNull(resp.getContentAsString());
        }
```

**//3. Put Method Test Case**

```
            @Test
            public void testProductPut()throws Exception {
                MvcResult result=mockMvc.perform(put("/product/edit")
                        .contentType(MediaType.APPLICATION_JSON)
                       .content("{\"prodId\":1, \"prodCode\":\"ABCDE\",\"
                        prodCost\":38.55,\"vendorCode\":
\"V14\"}")).andReturn();
            MockHttpServletResponse resp=result.getResponse();
            assertEquals(HttpStatus.OK.value(), resp.getStatus());
            System.out.println(resp.getContentAsString());
            assertNotNull(resp.getContentAsString());
            }
```

**//2. Get Method Test Case**

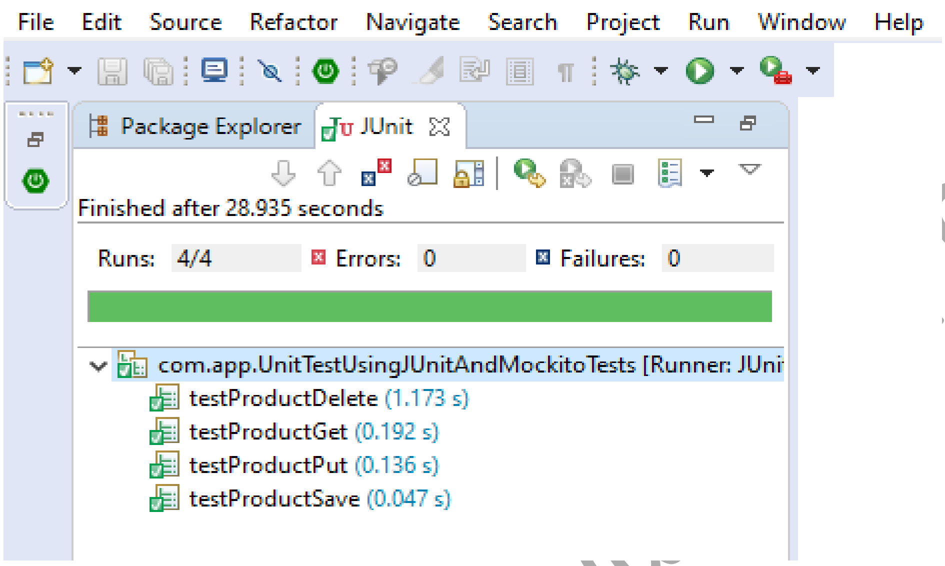
```
            @Test
            public void testProductGet() throws Exception {
        MvcResult result =
mockMvc.perform(get("/product/get/4")).andReturn();
                MockHttpServletResponse resp=result.getResponse();
                assertEquals(HttpStatus.OK.value(), resp.getStatus());
                assertNotNull(resp.getContentAsString());
            }
```

**//4. Delete Method Test Case**

```
            @Test
            public void testProductDelete()throws Exception {
MvcResult
result=mockMvc.perform(delete("/product/delete/5")).andReturn();
                MockHttpServletResponse resp=result.getResponse();
```

```
                        assertEquals(HttpStatus.OK.value(), resp.getStatus());
                        System.out.println(resp.getContentAsString());
                        assertNotNull(resp.getContentAsString());
            }
}
```

**1>Output SCRENN Short of JUnit:--**

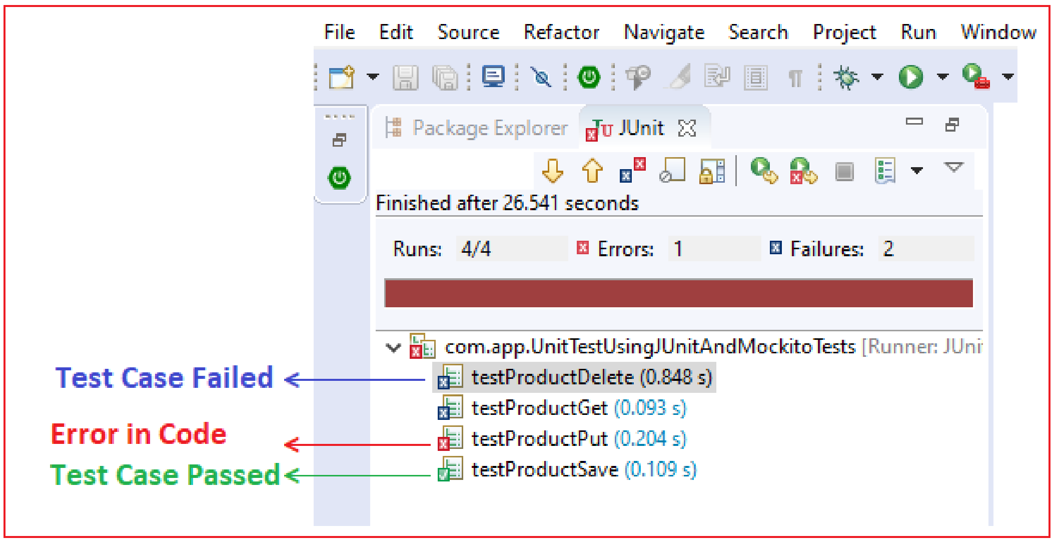


Finished after 28.935 seconds

Runs: 4/4 Errors: 0 Failures: 0

com.app.UnitTestUsingJUnitAndMockitoTests [Runner: JUni
testProductDelete (1.173 s)
testProductGet (0.192 s)
testProductPut (0.136 s)
testProductSave (0.047 s)

**2>Output Screen for all test cases.**



Finished after 26.541 seconds

Runs: 4/4 Errors: 1 Failures: 2

com.app.UnitTestUsingJUnitAndMockitoTests [Runner: JUni
testProductDelete (0.848 s)
testProductGet (0.093 s)
testProductPut (0.204 s)
testProductSave (0.109 s)

**NOTE:--** 1>Green Color(testProductSave) indicate Test cases passed.
2>Blue color(testProductDetele,testProductGet) indicates test cases failed.
3>Red color(testProductPut) indicate error in test class specific method code.

**FB: [https://www.facebook.com/groups/thejavatemple/](https://www.facebook.com/groups/thejavatemple/)**

**email: javabyraghu@gmail.com**