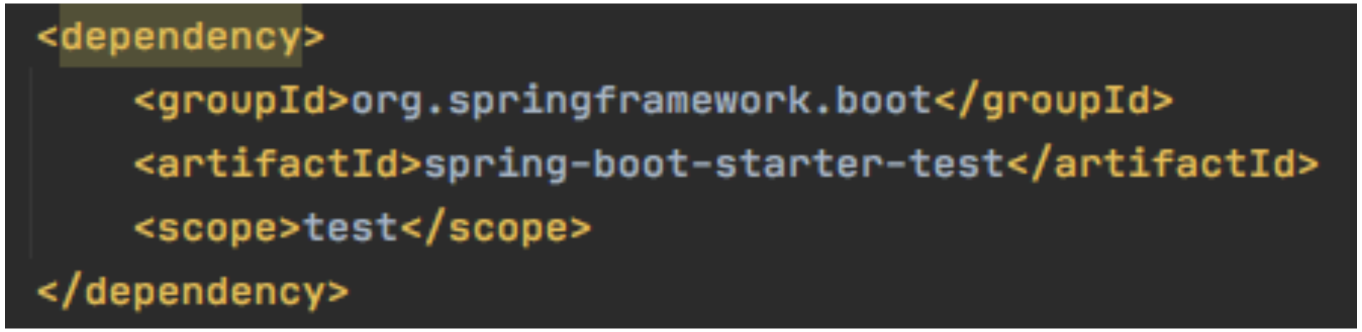
Spring Boot Starter Test adds

- Junit for unit tests
- SpringTest & SpringBoot Test for integration test
- AssertJ for Java asserts
- Hamcrest for matcher
- Mockito for mockups
- JSONassert for assertions on JSON structures
- JsonPath for Xpath on JSON structures

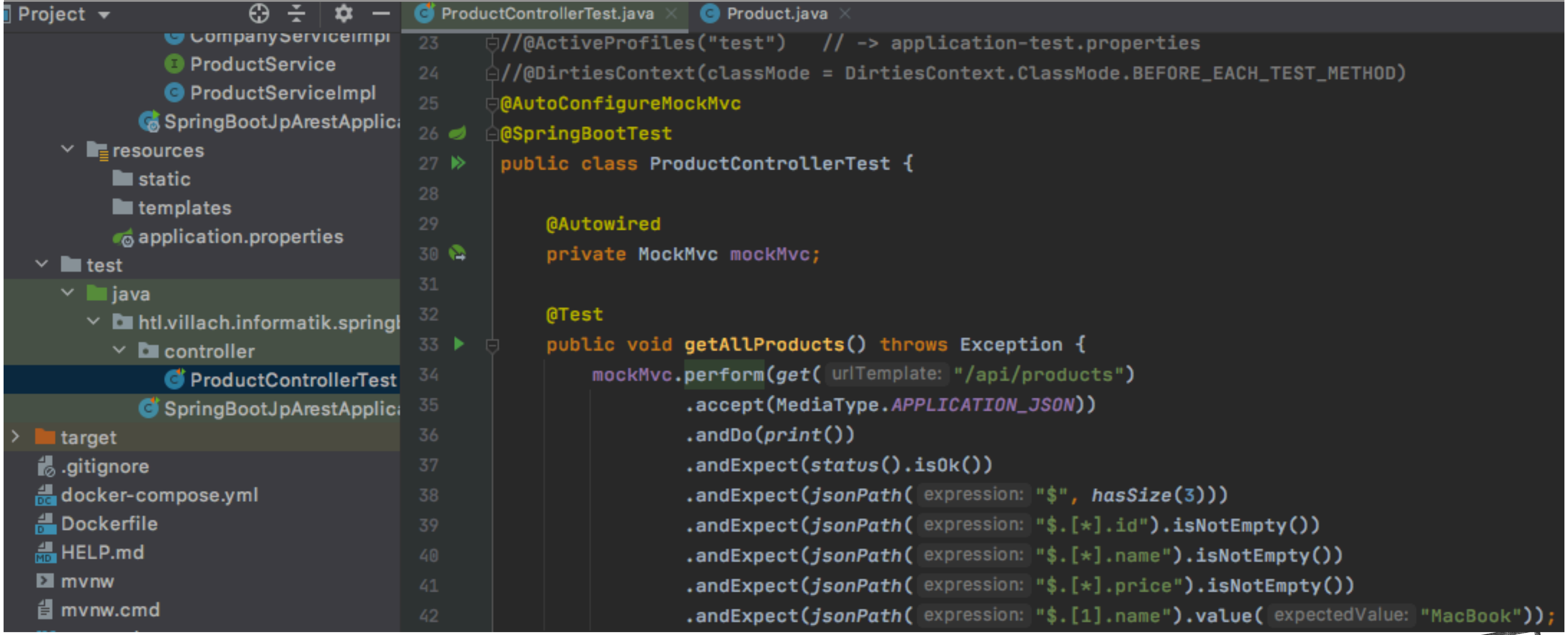


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

and integration tests by

- @JsonTest
- @WebMvcTest
- @DataJpaTest
- @JdbcTest
- @DataMongoTest
- @RestClientTest

mockMvc for controllers CRUD ops



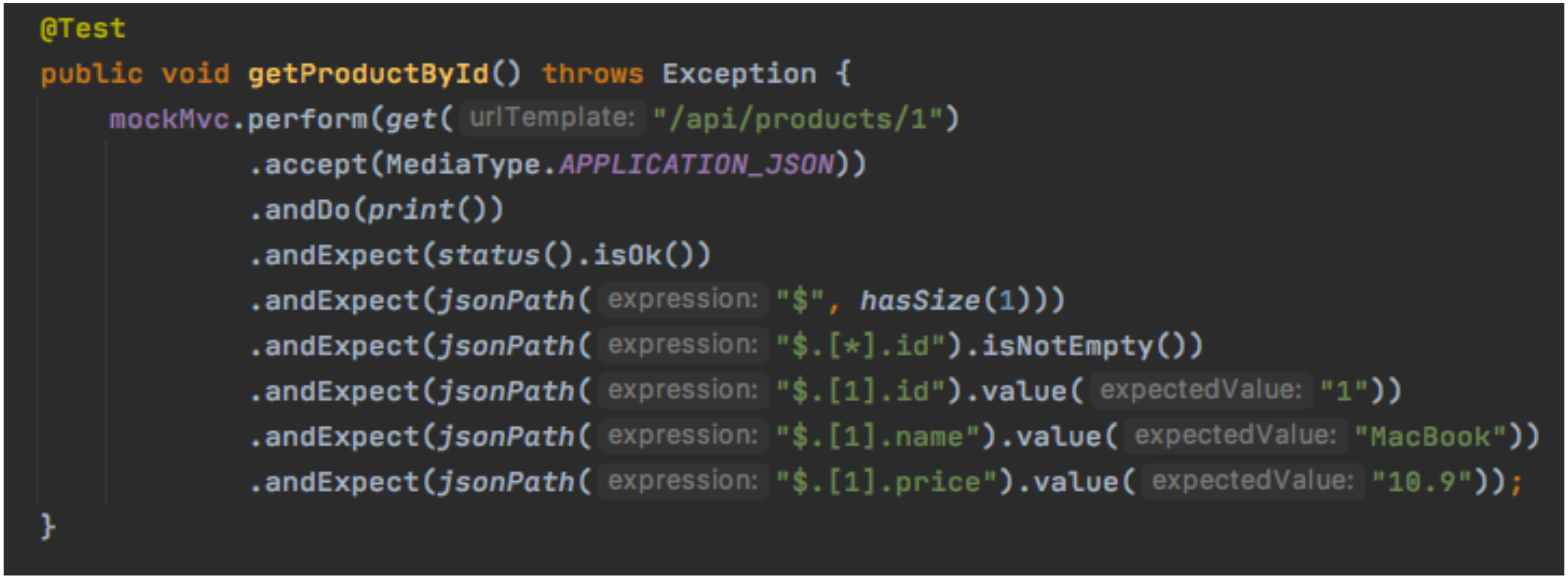
Project

CompanyServiceImpl
ProductService
ProductServiceImpl
SpringBootJpArestApplic
resources
static
templates
application.properties
test
java
htl.villach.informatik.springl
controller
ProductControllerTest
SpringBootJpArestApplic
target
.gitignore
docker-compose.yml
Dockerfile
HELP.md
mvnw
mvnw.cmd

ProductControllerTest.java | Product.java

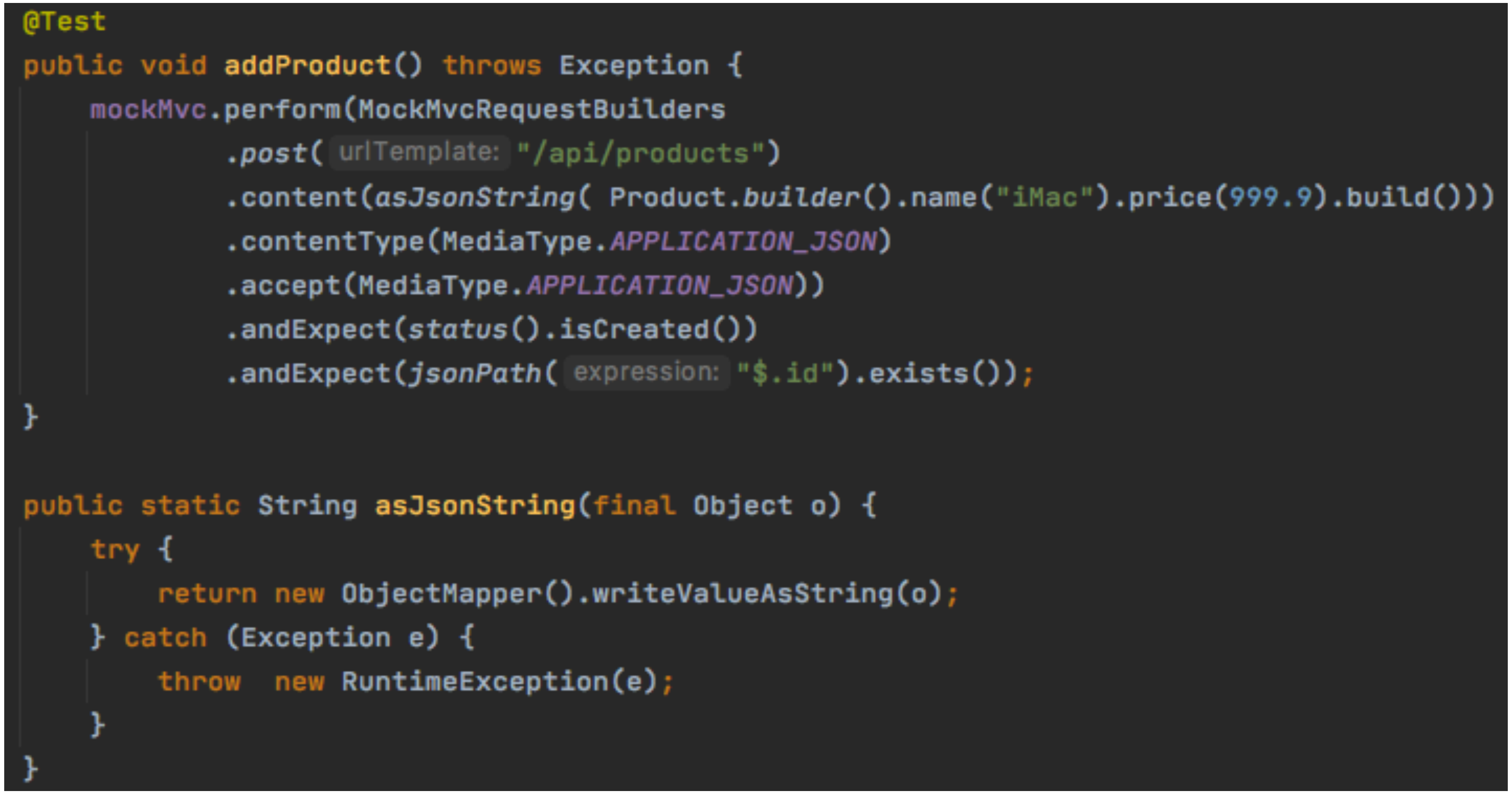
```java
//@ActiveProfiles("test")   // -> application-test.properties
//@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
@AutoConfigureMockMvc
@SpringBootTest
public class ProductControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void getAllProducts() throws Exception {
        mockMvc.perform(get( urlTemplate: "/api/products")
                .accept(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath( expression: "$", hasSize(3)))
                .andExpect(jsonPath( expression: "$.[*].id").isNotEmpty())
                .andExpect(jsonPath( expression: "$.[*].name").isNotEmpty())
                .andExpect(jsonPath( expression: "$.[*].price").isNotEmpty())
                .andExpect(jsonPath( expression: "$.[1].name").value( expectedValue: "MacBook"));
```

CRUD‘s read by id



```
@Test
public void getProductById() throws Exception {
    mockMvc.perform(get( urlTemplate: "/api/products/1")
            .accept(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$", hasSize(1)))
            .andExpect(jsonPath( expression: "$.[*].id").isNotEmpty())
            .andExpect(jsonPath( expression: "$.[1].id").value( expectedValue: "1"))
            .andExpect(jsonPath( expression: "$.[1].name").value( expectedValue: "MacBook"))
            .andExpect(jsonPath( expression: "$.[1].price").value( expectedValue: "10.9"));
}
```

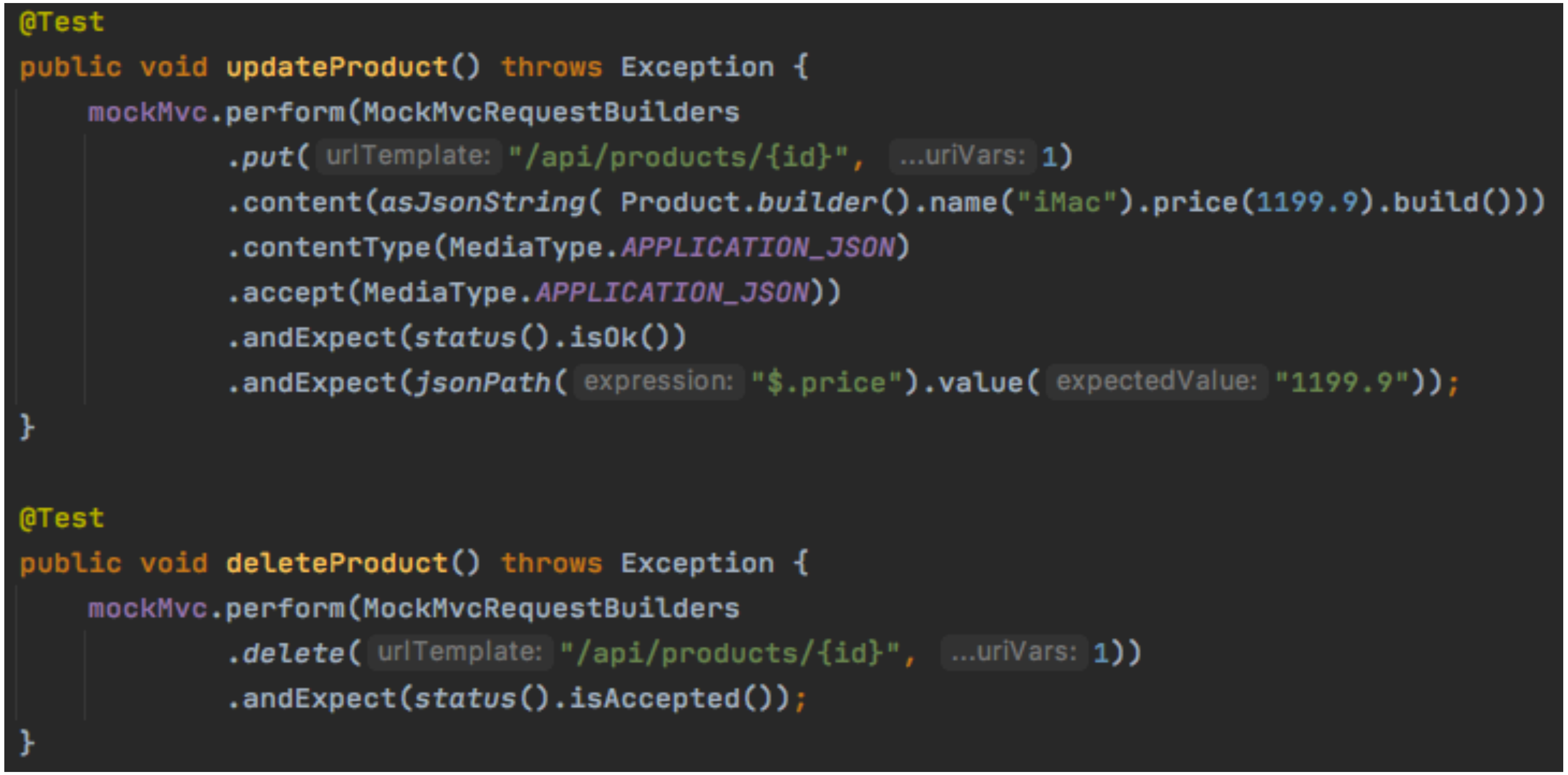
CRUD’s create



```java
@Test
public void addProduct() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "/api/products")
            .content(asJsonString( Product.builder().name("iMac").price(999.9).build()))
            .contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.APPLICATION_JSON))
            .andExpect(status().isCreated())
            .andExpect(jsonPath( expression: "$.id").exists());
}

public static String asJsonString(final Object o) {
    try {
        return new ObjectMapper().writeValueAsString(o);
    } catch (Exception e) {
        throw  new RuntimeException(e);
    }
}
```

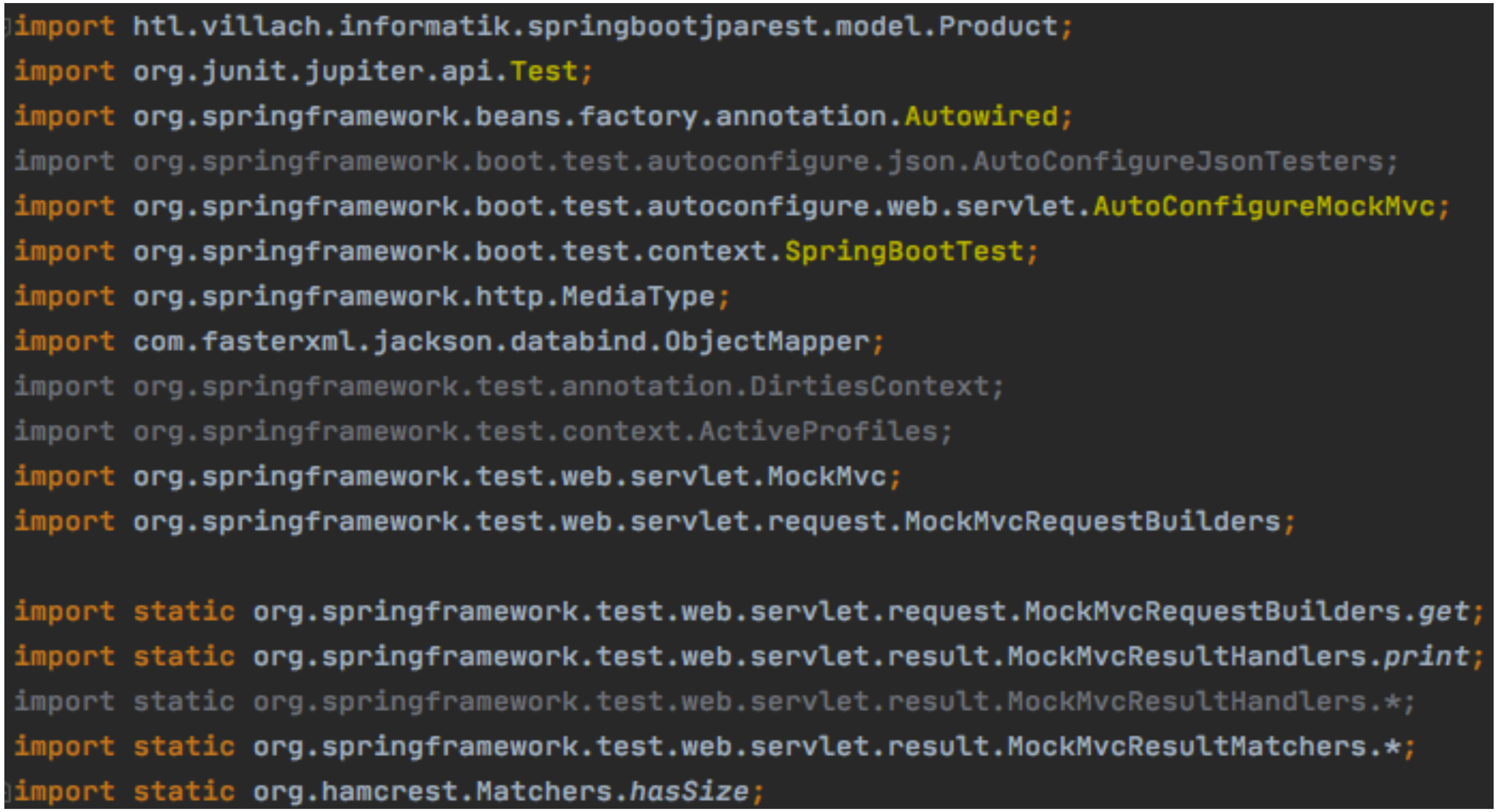
software inside

CRUD’s update an delete



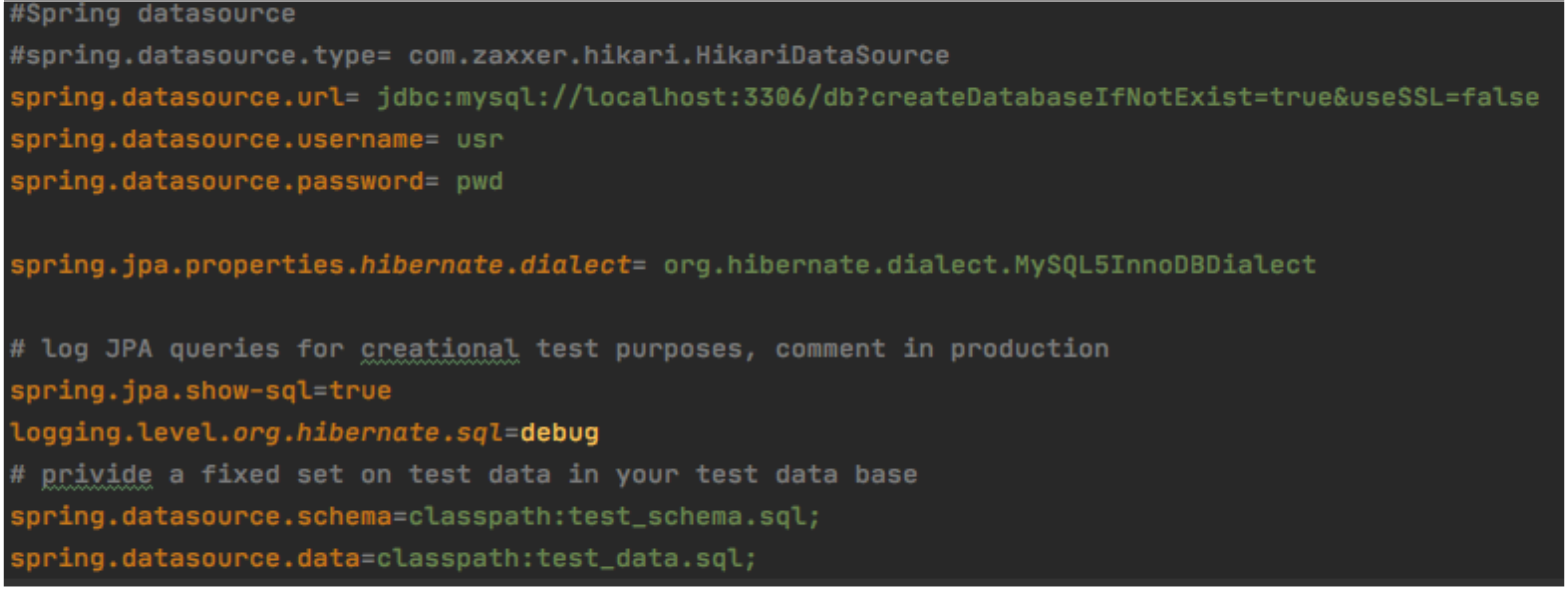
```
@Test
public void updateProduct() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders
            .put( urlTemplate: "/api/products/{id}", ...uriVars: 1)
            .content(asJsonString( Product.builder().name("iMac").price(1199.9).build()))
            .contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.price").value( expectedValue: "1199.9"));
}

@Test
public void deleteProduct() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders
            .delete( urlTemplate: "/api/products/{id}", ...uriVars: 1))
            .andExpect(status().isAccepted());
}
```

Needed imports



```
import htl.villach.informatik.springbootjparest.model.Product;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.json.AutoConfigureJsonTesters;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.hamcrest.Matchers.hasSize;
```

@ActiveProfiles("test") // -> application-test.properties



```
#Spring datasource
#spring.datasource.type= com.zaxxer.hikari.HikariDataSource
spring.datasource.url= jdbc:mysql://localhost:3306/db?createDatabaseIfNotExist=true&useSSL=false
spring.datasource.username= usr
spring.datasource.password= pwd

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect

# log JPA queries for creational test purposes, comment in production
spring.jpa.show-sql=true
logging.level.org.hibernate.sql=debug
# privide a fixed set on test data in your test data base
spring.datasource.schema=classpath:test_schema.sql;
spring.datasource.data=classpath:test_data.sql;
```