Spring Boot Testing 2

End To End Testing
TestRestTemplate & Database Prepopulating
Unit Testing

Contents

Exercises

1. Add user monitoring columns. They should be automatically saved
    a. Created date
    b. Modified date
2. Add deleted column to users table
    a. Write custom query to fetch not deleted users

Spring Boot Starter Test

- JUnit 4: The de-facto standard for unit testing Java applications.
- Spring Test & Spring Boot Test: Utilities and integration test support for Spring Boot applications.
- AssertJ: A fluent assertion library.
- Hamcrest: A library of matcher objects (also known as constraints or predicates).
- Mockito: A Java mocking framework.
- JSONassert: An assertion library for JSON.
- JsonPath: XPath for JSON.

Simplest Spring Boot Test Structure

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MoneyTransferAppApplicationTests {
  @Test
  public void contextLoads() {
  }
}
```

The `@SpringBootTest` annotation tells Spring Boot to go and look for a main configuration class (one with `@SpringBootApplication` for instance), and use that to start a Spring application context.

End To End Test [Starts Application]

```
import static org.assertj.core.api.Assertions.assertThat;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class HttpRequestTest {

   @LocalServerPort
   private int port;

   @Autowired
   private TestRestTemplate restTemplate;

   @Test
   public void userFetchEndpointWithNonExistentUserIdShouldReturn404() throws Exception {
      assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/users/1/",
            String.class)).contains("Not Found");
   }
}
```

https://gist.github.com/nursultanturdaliev/f8318c9644995ba06048410007841a1d

Simple Sanity Check

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MoneyTransferAppApplicationTests {

  @Autowired
  private UserController userController;

  @Test
  public void contextLoads() {
    assertThat(userController).isNotNull();
  }
}
```

The `@Autowired` annotation is interpreted by the Spring and the controller is injected before the test methods are run. We are using AssertJ (`assertThat()` etc.) to express the test assertions.

Initializing Database Before Running Tests

1. Create `data.sql` file under tests root folder
   a. Spring Boot automatically executes commands inside this file
2. Create platform specific files
   a. `schema-${platform}.sql`
      i. `hsqldb`, `h2`, `oracle`, `mysql`, `postgresql`
3. Using @Sql command
   a. Class level
   b. Method level

End To End Testing
TestRestTemplate

- getForObject
- getForEntity
- postForEntity
- delete
- put
- exchange

TestRestTemplate **getForObject**

```
@Test
public void testFetchAllUsersEndpoint() {
   assertThat(this.restTemplate.getForObject(HTTP_LOCALHOST + port +
"/api/users/",
         User[].class)).hasSize(2);
}
```

https://gist.github.com/nursultanturdaliev/6ba686b944a750f6a2ec0db9d8c5f7fa

TestRestTemplate **getForEntity**

```
public void testFetchUserEndpointWithExistingUser() {

   User firstUser = this.restTemplate.getForObject(HTTP_LOCALHOST + port + "/api/users/1",
User.class);

   assertThat(firstUser).hasFieldOrPropertyWithValue("firstName", "Nursultan");

   assertThat(firstUser).hasFieldOrPropertyWithValue("lastName", "Turdaliev");

}
```

TestRestTemplate **postForEntity**

```
@Test
public void testCreateUserEndpoint() throws URISyntaxException {
   URI url = new URI(HTTP_LOCALHOST + port + "/api/users/");
   ResponseEntity<User> userResponseEntity = this.restTemplate
         .postForEntity(url, new User("Tokon", "Mamytov"), User.class);

   assertThat(userResponseEntity.getStatusCode()).isEqualTo(HttpStatus.CREATED);
   assertThat(userResponseEntity.getBody()).hasFieldOrPropertyWithValue("firstName", "Tokon");
   assertThat(userResponseEntity.getBody()).hasFieldOrPropertyWithValue("lastName", "Mamytov");
}
```

Insert Data Into Database Using @Sql Method Level

```
@Test
@Sql(executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD, scripts = "classpath:delete-users.sql")
@Sql(executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD, scripts =
"classpath:testRunSpecificMethodRelatedSql.sql")
public void testRunSpecificMethodRelatedSql() throws URISyntaxException {

   URI url = new URI(HTTP_LOCALHOST + port + "/api/users/");
   ResponseEntity<User> userResponseEntity = this.restTemplate.getForEntity(url + "100",User.class);

   assertThat(userResponseEntity.getStatusCode()).isEqualTo(HttpStatus.OK);
   assertThat(userResponseEntity.getBody()).hasFieldOrPropertyWithValue("firstName", "Azamat");
   assertThat(userResponseEntity.getBody()).hasFieldOrPropertyWithValue("lastName", "Baimatov");
}
```

@Sql query execution on class level

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
@Sql({ "drop_schema.sql", "schema.sql", "data.sql" })
public class UsersApiTest {
```

SQL file locations

classpath:delete-users.sql => **src/test/resources/delete-users.sql**

classpath:testRunSpecificMethodRelatedSql.sql => **src/test/resources/testRunSpecificMethodRelatedSql.sql**

Exercise

1. Cover user endpoints
    a. User search endpoint
    b. User update endpoint
    c. User delete endpoint
    d. Fetch user transactions endpoint
2. Cover transactions endpoint with end-to-end tests
3. Create create-transaction endpoint
    a. Cover this endpoint with tests

Unit Testing

```java
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public Iterable<User> fetchAllUsers() {

        return userRepository.findAll();
    }
}
```

```java
@SpringBootTest
@RunWith(SpringRunner.class)
public class UserServiceTest {

    @Autowired
    private UserService userService;

    @Test
    public void testFetchAll() {
        userService.fetchAllUsers();
    }
}
```

Unit Testing Removing Dependencies

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTestImprovedVersionTwo {

   @Autowired
   private UserRepository userRepository;

   private UserService userService;

   @Before
   public void initUserService()
   {
      userService = new UserService(userRepository);
   }

   @Test
   public void testFetchAll() {
      assertThat(userService.fetchAllUsers()).isNotEmpty();
   }
}
```

```
@Service
public class UserService {


   private final UserRepository userRepository;

   public UserService(UserRepository userRepository)
   {
      this.userRepository = userRepository;
   }

   public Iterable<User> fetchAllUsers() {

      return userRepository.findAll();
   }
}
```

Unit Testing Mocking External Dependencies

```
public class UserServiceTestCorrectVersion {

   private UserRepository userRepository = Mockito.mock(UserRepository.class);

   private UserService userService;

   @Before
   public void initUserService() {
      userService = new UserService(userRepository);
   }

   @Test
   public void testFetchAll() {

      List<User> userList = new ArrayList<>();
      userList.add(new User("Askar", "Akaev"));

      when(userRepository.findAll()).thenReturn(userList);
      assertThat(userService.fetchAllUsers()).isNotEmpty();
   }
}
```

Without Starting Server

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class ApplicationTest {

  @Autowired
  private MockMvc mockMvc;

  @Test
  public void shouldReturnDefaultMessage() throws Exception {
    this.mockMvc.perform(get("/"))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(
            content()
                .string(containsString("Welcome Home!")
                )
        );
  }
}
```

Full Spring application context is started, but without the serve

https://gist.github.com/nursultanturdaliev/febbc3d3c4767a5e297b5400a01108df

Web Layer Test

```
@RunWith(SpringRunner.class)
@WebMvcTest
public class WebLayerTest {

  @Autowired
  private MockMvc mockMvc;

  @Test
  public void shouldReturnDefaultMessage() throws Exception {
    this.mockMvc.perform(get("/"))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(content().string(containsString("Welcome Home!")));
  }
}
```

https://gist.github.com/nursultanturdaliev/ec2a8bb3a8d140f4034fd6cac8e5f43a

Spring Boot is only instantiating the web layer, not the whole context.

Loading Specific Controller & Mocking

```
@RunWith(SpringRunner.class)
@WebMvcTest(HomeController.class)
public class ControllerContextTest {

   @Autowired
   private MockMvc mockMvc;

   @MockBean
   private HomeService service;

   @Test
   public void greetingShouldReturnMessageFromService() throws Exception {
      when(service.welcome()).thenReturn("Welcome Mock!");
      this.mockMvc.perform(get("/")).andDo(print()).andExpect(status().isOk())
            .andExpect(content().string(containsString("Welcome Mock!")));
   }
}
```

References

- https://spring.io/guides/gs/testing-web/
- https://www.baeldung.com/integration-testing-in-spring
- https://www.baeldung.com/rest-template
- https://www.baeldung.com/junit-before-beforeclass-beforeeach-beforeall
- https://reflectoring.io/unit-testing-spring-boot/