Spring Boot Testing

Introduction

Contents

- Resolve Time Zone Issue
- Database Initialization
- Custom Queries Exercises
- Spring Boot Starter Test
- Spring Data JPA - @DataJpaTest

Resolve Time Zone Issue

jdbc:mysql://localhost:3307/money-transfer-app?**serverTimezone=UTC**

Database Initialization

spring.jpa.hibernate.ddl-auto

- `none`: The default for `MySQL`. No change is made to the database structure.
- `update`: Hibernate changes the database according to the given entity structures.
- `create`: Creates the database every time but does not drop it on close.
- `create-drop`: Creates the database and drops it when `SessionFactory` closes.
- `validate`: simply validates and fails if there is an issue

Custom Queries Exercises

1. Add birthdate to user using Flyway DB
2. Add user filtering endpoints
    a. Find by firstname or lastname or email
    b. Fetch top ten matching
    c. Make parameters optional
    d. Ignore case
3. Add transaction filtering logic by transaction id
4. Add transaction monitoring columns. They should be automatically saved
    a. Created date
    b. Modified date
5. Add active column to users table
    a. Write custom query to fetch active users
6. Filter transactions by currency

How Do You Write Tests?

Spring Data JPA @DataJpaTest

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class UserRepositoryTest {
  @Test
  public void testFindByFirstName() throws Exception {
    User mockUser = new User();
    mockUser.setFirstName("Chyngyz");
    mockUser.setLastName("Aitmatov");
    this.entityManager.persist(mockUser);

    Iterable<User> users = this.repository.findByFirstName("Chyngyz");
    List<User> userList= new ArrayList<>();
    users.forEach(userList::add);
    assertThat(userList).isNotEmpty();
    assertThat(userList).contains(mockUser);
  }
}
```

https://gist.github.com/nursultanturdaliev/09c4c5fc6fe531351ff6d0d8d1732447

H2 Database Engine

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2 MB jar file size

Application Properties

application.properties

```
spring.profiles.active= dev
```

application-dev.properties

```
spring.jpa.generate-ddl = true
spring.jpa.hibernate.ddl-auto=none

spring.datasource.url=jdbc:mysql://localhost:3307/money-transfer-app?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
```

application-test.properties

```
spring.jpa.generate-ddl = true
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.database= HSQL
```

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.

Spring Boot Starter Test

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

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.

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.

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

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

```
@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!")));
  }
}
```

Exercises

1. Test Fetch Transaction Endpoint Not Found Case
2. Test UserRepository with @DataJPATest
3. Test TransactionController by loading specific context

References

- https://spring.io/guides/gs/testing-web/
- https://www.baeldung.com/integration-testing-in-spring
-