Week 14: JUnit

blog.bookly.online

jeanne May 7, 2020

Hi everyone

We created our Testplan! You can find it here: Testplan.md
We decided to do 3 different kind of tests:

- Function Testing: Cucumber, JUnit
- Database Integrity Test: JUnit + Mockito
- Field Test

Our goal is to achieve 20% test coverage. Our start date for implementing tests is the 13.5.2020.

Inline-Update: We integrated the visibility of our tests in gitlab. See our live coverage badge here:

coverage 29.00%

test coverage on master-branch

Function Testing

We already implemented BDD tests for the use cases of the first semester. You can find our implemented step definition for Cucumber here. The corresponding feature files are here. For the Java Backend we want to implement JUnit Tests.

Database Integrity Test

To ensure a correct data processing for creating, reading, updating and deleting, we want to implement JUnit Tests in combination with the Framework Mockito for our backend API.

Field Test

We are especially excited to plan the field test. As we are implementing an app for kids, we want to do the test session with children. We can imagine that children with an age between 10 to 12 fit perfect into our user group. We are in contact with a teacher that would help us with her school class to test our app.

But first we want to realize a few more use cases and clean up our styling. Then we will plan, what kind of remote tool we will use for the test session, what questions we want to ask and how we prepare the system for them. We will update you as soon as we planned more

Gitlab-CI: Test Stage

Since last semester we have a working CI for testing, building and deploying our application. You can find the corresponding configuration file here.

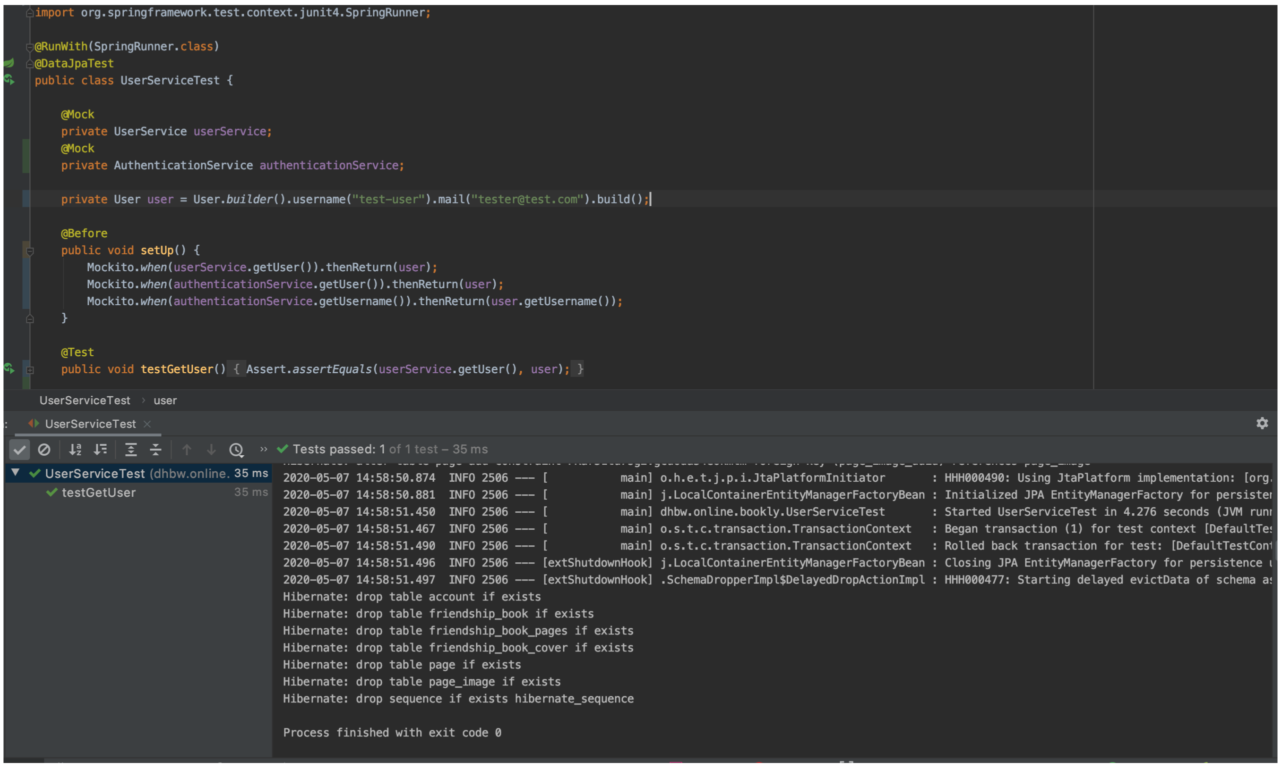
If you want to see the logging of any test, just go to our pipelines and click on a test stage you're interested in. Here's an example:
https://gitlab.com/project_bookly/bookly/-/jobs/542554917

Last semester we already filmed our first runnable cucumber tests. You can watch it here

https://youtu.be/hBR4Yb2V8zU

Proof of working JUnit Tests

As you can see we just implemented a mocked test for our UserService and our AuthenticationService. We will soon have more tests to run We used here the Junit4 SpringRunner of Spring's JUnit Integration too, to start the application context and autowire our services later on.



```java
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@DataJpaTest
public class UserServiceTest {

    @Mock
    private UserService userService;
    @Mock
    private AuthenticationService authenticationService;

    private User user = User.builder().username("test-user").mail("tester@test.com").build();

    @Before
    public void setUp() {
        Mockito.when(userService.getUser()).thenReturn(user);
        Mockito.when(authenticationService.getUser()).thenReturn(user);
        Mockito.when(authenticationService.getUsername()).thenReturn(user.getUsername());
    }

    @Test
    public void testGetUser() { Assert.assertEquals(userService.getUser(), user); }
```

UserServiceTest › user
UserServiceTest
Tests passed: 1 of 1 test – 35 ms
UserServiceTest (dhbw.online. 35 ms
testGetUser 35 ms

```
2020-05-07 14:58:50.874  INFO 2506 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.
2020-05-07 14:58:50.881  INFO 2506 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persister
2020-05-07 14:58:51.450  INFO 2506 --- [           main] dhbw.online.bookly.UserServiceTest       : Started UserServiceTest in 4.276 seconds (JVM runn
2020-05-07 14:58:51.467  INFO 2506 --- [           main] o.s.t.c.transaction.TransactionContext   : Began transaction (1) for test context [DefaultTes
2020-05-07 14:58:51.490  INFO 2506 --- [           main] o.s.t.c.transaction.TransactionContext   : Rolled back transaction for test: [DefaultTestCont
2020-05-07 14:58:51.496  INFO 2506 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence u
2020-05-07 14:58:51.497  INFO 2506 --- [extShutdownHook] .SchemaDropperImpl$DelayedDropActionImpl : HHH000477: Starting delayed evictData of schema as
Hibernate: drop table account if exists
Hibernate: drop table friendship_book if exists
Hibernate: drop table friendship_book_pages if exists
Hibernate: drop table friendship_book_cover if exists
Hibernate: drop table page if exists
Hibernate: drop table page_image if exists
Hibernate: drop sequence if exists hibernate_sequence

Process finished with exit code 0
```

Proof of working JUnit Tests

We added the dependencies for JUnit and Mockito to the Maven-POM of our backend. You can find the complete document here: https://gitlab.com/project_bookly/bookly/-/blob/master/backend/pom.xml

```xml
<!-- Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
            <version>1.10.19</version>
            <scope>test</scope>
        </dependency>
        .....
```

Automatic Test Data

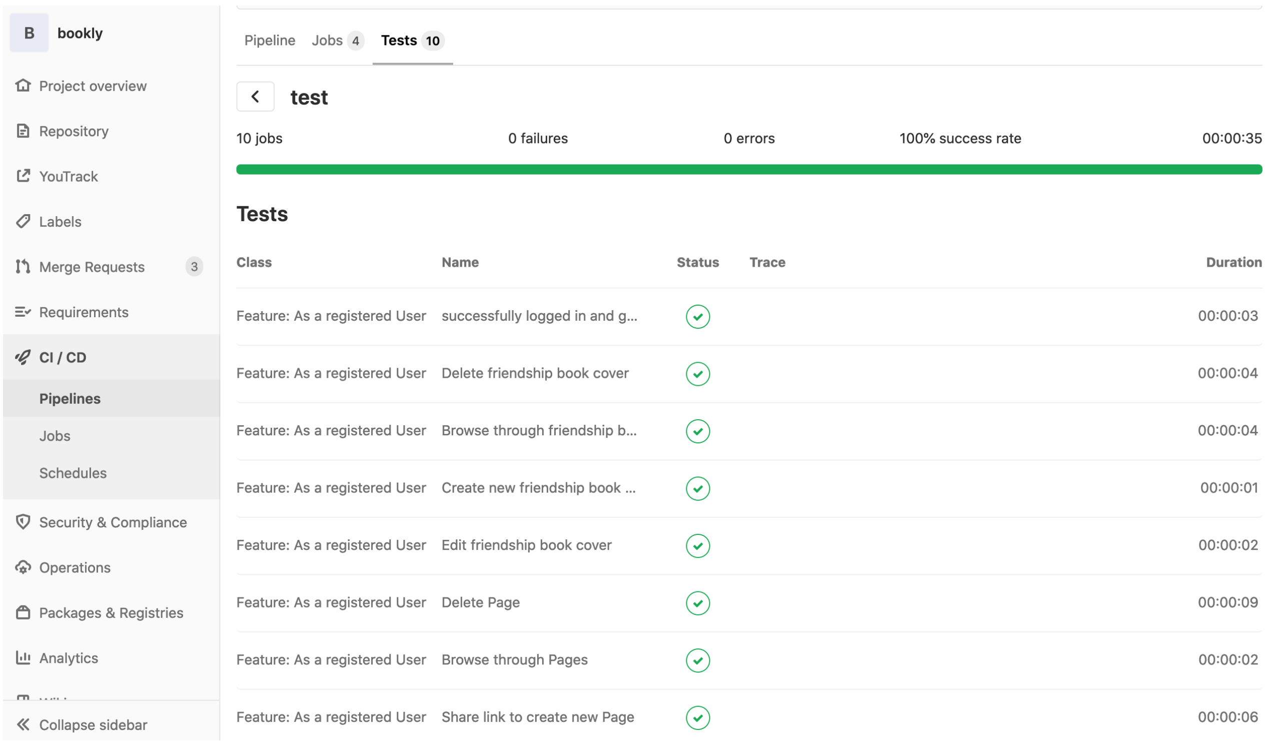
For our local deployment we already created some test data. It was necessary as our local H2 is not saving any data while running and is empty after a restart. So we implemented a Component that initialize the H2 at the beginning. We are using the same data for our test environment. You can look inside of the file here: https://gitlab.com/project_bookly/bookly/-/blob/master/backend/src/test/java/dhbw/online/bookly/TestData.java

As we are using the JUnit4 Spring Runner, the class is recognized as Component and automatically creates the data we need.

Gitlab: Test Coverage

We want to integrate the Maven Surefire Plugin for displaying the test coverage within Gitlab. After we implemented and integrated it into our project, we will update this blog post here

Update:
We integrated Surefire and JaCoCo into our project. We are now able to see a more detailed view of our passed/failed unit and cucumber tests by clicking to a specific pipeline.



test

10 jobs | 0 failures | 0 errors | 100% success rate | 00:00:35

Tests

Class	Name	Status	Trace	Duration
Feature: As a registered User	successfully logged in and g...	✓		00:00:03
Feature: As a registered User	Delete friendship book cover	✓		00:00:04
Feature: As a registered User	Browse through friendship b...	✓		00:00:04
Feature: As a registered User	Create new friendship book ...	✓		00:00:01
Feature: As a registered User	Edit friendship book cover	✓		00:00:02
Feature: As a registered User	Delete Page	✓		00:00:09
Feature: As a registered User	Browse through Pages	✓		00:00:02
Feature: As a registered User	Share link to create new Page	✓		00:00:06

Example of tests within a pipeline, generated by surefire

For the whole test coverage we created a badge for our master branch. The coverage is calculated by JaCoCo for our backend. We are extracting the coverage value from its result and are handing it over to Gitlab. The value is refreshed after a successful test stage and is visibile within the test pipeline or you can see our current test state here live:

coverage 29.00%

test coverage on master-branch

Deployment

As you can see in our .gitlab-ci.yml we are deploying manually. Even though we have automated testing, we prefer to deploy our system by clicking a button because it gives us immense satisfaction and happiness.