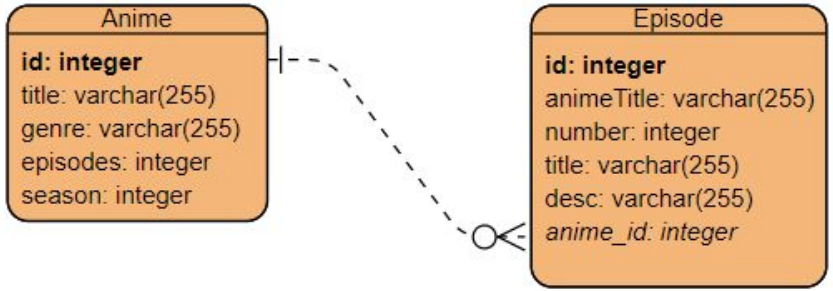
Hobby Web Application - AnimeList

Presented By Matthew Holmes

My Approach

My first task was to decide which hobby I'd base my project off of and I ended up choosing anime shows.

Secondly, I thought about what I'd want my web application to do & what entities for a database could be used to achieve this. I decided to have my app store information on the shows you're watching, and allow you to summarise what happens in each episode. Thus the entities I decided to use were 'Anime' and 'Episode'.



Sprint Plan

During my Sprint Plan, I split the project into 4 main areas that I used as epics for the sprint:

- Back-End: Areas that handed my database/tables & CRUD functionality
- Front-End: Areas that involved my webpage & how this would utilize my back-end
- Testing: My test suite, namely including Unit, Integration and Acceptance tests.
- Documentation: Additional tasks that aren't codebase related, e.g. ERD, UML, Risk Assessment

The goal of the sprint was to deliver a working web application that adhered to the MVP of the project.

Learned Technologies:

Spring API - An application framework and inversion of control container that assists in the coding of enterprise features & web applications. It presented me with a new way of coding by utilizing auto-generated code that handled a lot of the connecting of different modules.

HTML, CSS & JavaScript - A standard markup language and programming language respectively, used to build the structure and functionality of my webpage.

Selenium - A framework that assists with the testing of web applications by giving the user a playback tool and allowing them to write scripts in Java which can then be using to simulate webpage interactions.

Version Control & Testing

I handled version control with GitHub & GitBash, and followed the branch-feature model (where I'd create a new branch for each feature of my application) to organize my workflow.

My testing suite involved unit tests, integration tests & acceptance tests.

I performed my unit tests using JUnit & Mockito and achieved a 62% coverage. However, I large majority of the 38% is made-up of auto-generated code from methods such as hashCode() and equals() & excluding these methods brings the coverage percentage to 90+. Additionally, these method are only used for my JUnit assert statement, as opposed to being part of my application's functionality so I deemed this test coverage acceptable.

I performed my integration tests using Spring's MockMvc to mock requests & check the responses.

I performed my acceptance tests using JUnit & Selenium to run scripts that would run through the various use cases of my project.

Demonstration

Sprint Review

What I’ve achieved:

I have succeeded in creating a simple web application.

I have created a extensive suite of tests for the various classes and their methods.

What I’ve missed:

I haven’t applied static analysis to my code for detecting smells & refactoring.

Sprint Retrospective

What went well:

I did well in working with a variety of different technologies for the first time & producing a working web application nonetheless.

What could be improved:

I didn’t adhere to the set feature-branch model too well due to my inexperience with Spring. This led to me making changes on incorrect branches at times.

Conclusion

In conclusion, the project was a valuable experience that added many tools & much experience to my repertoire. While there is still room for improvement, such as stricter adherence to the version control branch model I was using and/or time management so that I could include static analysis & refactoring, I feel I've grown a lot over the course of this project.

Further steps I could take for future sprints would be to include what I missed such as static analysis of my codebase, as well as added stress testing to my test suite. Then I could think about expanding the functionality of my application.

Questions?