SOLENT UNIVERSITY

School of Media Arts and Technology
BSc (Hons) Software Engineering

[Name]
[Solent ID]

[Rideout Management System]

Commented [SP1]: Or any other proper name

Reflection Report

SSD-BCC608-AE2 (60%)

Contents

1 Project Management

2 Design

3 Development

4 Testing

To support testing; the following methods were used:

- Unit testing
- Integration testing
- Code reviews
- Test plan

Unit testing and integration testing was implemented using JUnit in the API. Due to the fact that there was not a large amount of business logic in the system; not a large amount of unit tests were required to reach full coverage — however; an example of such a unit test was testing the salting and hashing of passwords, and the matching of a password against the salted and hashed password.

Many integration tests were created in the API to test the RESTful interface. Examples of this include testing the retrieval of ride outs using the path; testing the registration of users etc. As well as this; inte-gration tests were created to ensure that the API would only peform actions with the required parameters, and that the API would output the correct HTTP status codes on failures etc.

Code reviews were performed before every pull request into dev from feature branches; and from dev into master. These code reviews ensured that code was readable, and performed what was required. As part of the code-review; the web application would also be tested against any new features that were being merged; for example if a feature implemented joining a ride out was being merged, this would be tested from a user's perspective to ensure that the feature worked as intended, and also failed as intended (i.e. handles errors correctly).

A formal test plan was created to test the final system against the use-cases, to ensure that the system both meets the requirements gathered during the design phase; and that there are no serious bugs or issues.

5 Integration

To ensure that testing and deployment went smoothly throughout the development of this project; continuous-integration/continuous-deployment was implemented.

Initially TravisCI was decided upon as it was free, and ran unit and integration tests without prob-lems; however it soon became cumbersome to deploy the new versions of the quickly changing project manually.

To counter this; it was decided to move to CircleCI. With CircleCI it became possible to move from continuous-integration to continuous-deployment; where a workflow was created to:

Commented [SP2]: Please research on **continuous integration**

1. Run unit and integration tests
2. Build a .war file
3. Build a docker image (using the .war file)
4. Upload the docker image to DockerHub
5. Patch the Kubernetes deployment to use the latest version from the docker image

Commented [SP3]: You can follow your own stpes to integration/deployment

Implementing this work-flow improved development speed by a large amount; as it was no longer necessary to contact a member of the team to update their server image to run a new version to test the current deployment; it was no longer necessary to set up a local environment to test development branches on, as one could now run the docker image; and it was no longer required to patch the server with the new version of the application.

The choice of using Kubernetes also gave multiple benefits; such as seamless patching, as Kuberenetes was configured to load a new pod of the new image before shutting down the old version, meaning that from the user's perspective there is 0 downtime; and also the ability to run multiple instances of different versions of the application — for example; the server was configured with two Kubernetes deployment: one running the docker image created by the 'master' branch; and one running the docker image created by the 'dev' branch. Doing this allowed easier testing of new features, before merging dev and master; and also meant that if a recent merge broke a feature in the 'dev' branch; there was always still a working 'master' branch.

6 Personal Evaluation

6.1 Product

I think that the product that was produced was reasonable given the time-constraints of the project; how-ever I believe that, in hindsight through the development of the product, many areas could have been improved upon to ensure that every team member was on the same page; and that the requirements could be understood by everybody.

By agreeing upon the requirements of the system at the start of the project, but only designing each aspect of the system as it went along; many issues were encountered that could have been avoided, such as missing aspects of features, or unnecessary features that were created as a product of miscommuni-cation before the implementation of such features.

6.2 Personal Contribution

I believe that I served the team well; through discussions of technology choices; design choices; and through supporting the team using various tools such as implementing the CI/CD system, configuring the Kubernetes server etc.

I do, however, believe that due to the lack of formal design during the development of the system; that I allowed scope creep to occur, such as the map display of the ride out route; which I believe is a good feature, however unnecessary and took up a large amount of resource to implement, which could have been spent ensuring that the base minimum viable product was implemented first.

7 Appendix

Appendices such as design work; meeting notes; class diagrams; source code; testing; etc. are contained within the .zip file submitted alongside this report.

Commented [SP4]: Those must in zipped folder

References

Driessen, Vincent (2010). A Successful Git Branching Model. URL: https://nvie.com/posts/a-successful-git-branching-model/.

SmartBear (2019). SwaggerUI. URL: https://swagger.io/tools/swagger-ui/.