Testing Policy

Golf Course Mapper

Team Recursive Recursion

October 17, 2018

Contents

1 Introduction

1.1 Purpose

This *Testing Policy Document* is intended to be a guide on the testing procedure, tools and practices that are followed when working on the *Golf Course Mapper* project. The remainder of this section describes the different repositories used within the project and the deployment pipeline used. The remainder of this document then continues to describe the testing policies applied to each individual repository, excluding the documentation repository.

1.2 Repositories Structure

There are five (5) repositories that are used for the project: `web-app`, `mobile-app`, `ios-app`, `mapper-api` and `documentation`. A short description of the contents of each repository is listed below.

- `web-app`

  Contains web application subsystem used for mapping and managing golf courses. The web application is built using *Angular 6*.

- `mobile-app`

  Contains the *Android* app used for viewing golf courses. The mobile app is built using native Java code.

- `ios-app`

  Contains the *iOS* app used for viewing golf courses. The iOS app is built using native Swift code.

- `mapper-api`

  Contains the *Entity Framework Core* API used to manage and provide access to the database. The database used is *PostGIS*, an extention of *PostgreSQL*.

- `documentation`

  Contains all documentation relating to the project, including this file. Documents are written in LaTeX. The repository also contains all additional images and published PDF versions of the documents.

1.3 Deployment Pipeline

A continuous deployment pipeline is set up using *Azure DevOps* [1]. When code is pushed to the `master` branch of the `web-app` or `mapper-api` repositories, the deployment pipeline is triggered. Once triggered, the unit and integration tests are run for the repository. If the tests pass, the subsystem is built. If the build is successful, the build artifact is stored (used for reverting to previous builds

if necessary) and a *Docker* image is created. The server then pulls the new Docker image and performs any necessary maintenance before deploying the new release.

Figure 1 shows the release management page of Microsoft Azure DevOps, where the history of builds for both the Web App and API can be viewed.

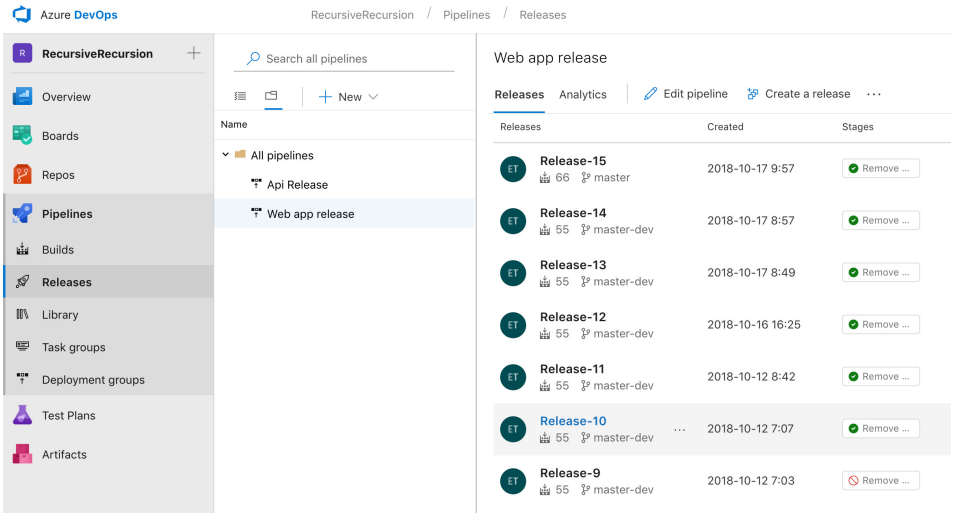

Figure 1: Continuous deployment

2 Testing Process

The following describes the process that is followed when testing code. The process described is the same for all repositories.

2.1 Targets

The following tests are performed to ensure that the system is operating correctly and satisfies the functional requirements, as outlined in the *Software Requirements Specification* (SRS) [2].

- *Unit Tests* are performed to ensure that individual functions and components of each system component performs correctly.
- *Integration Tests* are performed to ensure that the different subsystems operate correctly. This is further described in Section 5.
- *Acceptance Tests* are performed along with the client to ensure that the system meets the client's requirements and expectations.

The following tests are performed to ensure that the system complies with the non-functional system requirements, as outlined in the SRS.

- *Usability Testing* is performed to receive feedback on the UI design of the web and mobile applications. This also ensures that the UI is as user-friendly as possible.

2.2 Procedure

Unit tests must be written and passed before code can be merged to the `master-dev` branch. This ensures that code on the dev branch is at least functionally correct. For details on how unit tests are performed for the Android app and Zoning API, see the sections 4 and 5, respectively.

Integration tests must be passed before code can be pushed to the the `master` branch for deployment. This ensures that code on the master branch is always in a usable state. For details on how integration tests are performed, see Section 5.

Acceptance tests are performed monthly with the clients to verify that the system complies with their requirements.

Usability testing is performed at least once close to the initial release of the system. For details on the usability test performed, see section 3.1.

3 Web App Testing

This section describes the testing policies adopted in the `web-app` repository. Being a subsystem that mainly acts as a frontend, only usability tests are performed. The history of the tests are shown in Section 3.2.

3.1 Usability Testing

In order to satisfy the usability requirement of the system, it is necessary to perform a usability test to ensure that the system is user-friendly enough to non-technical users. A formal usability test is performed, asking testers to perform specific tasks on the system.

Testers are then asked to provide critical feedback on the system, which is then analyzed. The results are then used to identify key issues in the design of the user interface.

3.2 History

Figure 2 illustrates the usability test performed on 20 September, 2018. Six testers from various backgrounds were asked to perform a serious of tasks on the web application, without any prior introduction to the user interface. The testers were then given a form to fill out, asking them to provide any feedback. The results of the test have been compiled in the Usability Report [3].

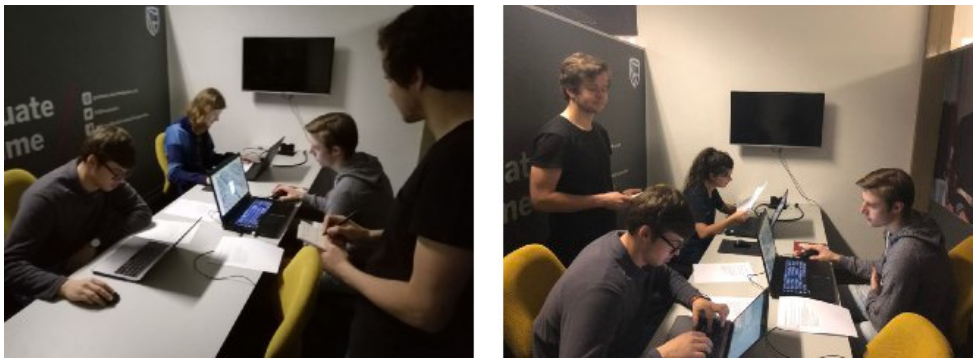

Figure 2: Usability test in progress

4 Mobile App Testing

This section describes the testing policies adopted in the `mobile-app` repository. The tools used for testing are described in Section 4.1. The layout of the test cases and history of the tests are shown respectively in Sections 4.2 and 4.3.

4.1 Tools

The mobile app uses *JUnit* to perform unit tests on the Android app. The built-in unit test functionality of *Android Studio* is used to automate the testing process. The tests are run by selecting UnitTests in the build options menu. This method is used for simplicity and ease of use. As example, when changes are made to the app, the developer can make the changes and easily run the unit tests afterwards to ensure that the existing functionality is still correct. New tests can also easily be added to test new functionality.

4.2 Test Cases

The tests are located in the `/app/src/test/` folder, specifically in the `UnitTest.java` source file.

4.3 History

Figure 3 shows the unit tests being executed in Android Studio.

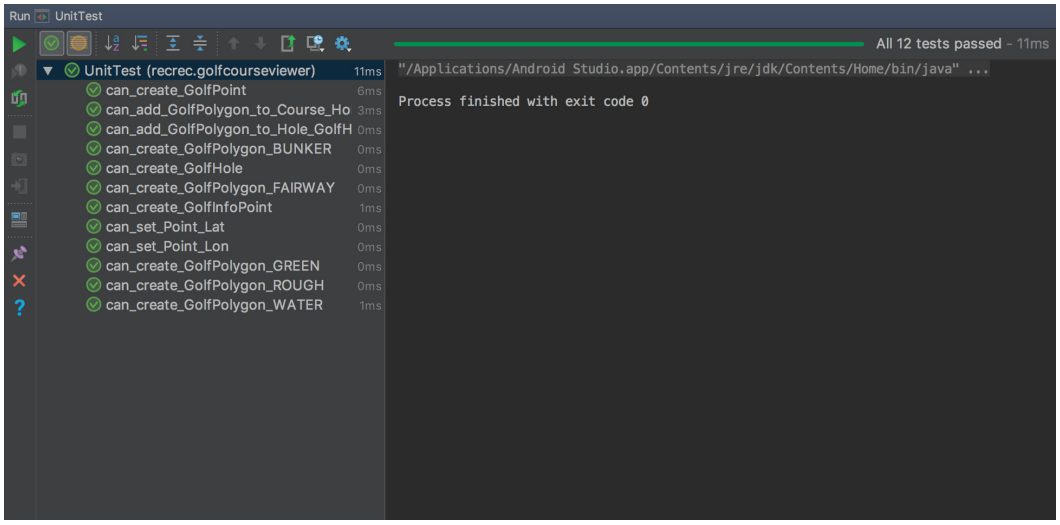

Figure 3: JUnit tests being executed

5 Zoning API Testing

This section describes the testing policies adopted in the `mapper-api` repository. The tools used for testing are described in Section 5.1. The layout of the test cases and history of the tests are shown respectively in Sections 5.2 and 5.3.

5.1 Tools

Moq is used to validate input to function calls and produce expected output for functions that are not currently being tested. This allows you to focus on only a specific API function at a time, without worrying about it failing due to problems in other functions.

xUnit is then used to execute both unit and integration tests by making use of Moq to single out a specific API endpoint (and mocking the ouput of the others) and then asserting that the output of the tested function is as expected. The combination of these two tools allows powerful testing.

The tests are automatically run by the Microsft Azure DevOps pipeline, described in Section 1.3.

5.2 Test Cases

The tests are located in the `/Test/TestSuite/API` folder. Every controller (and therefore, every use case) is tested in a separate `.cs` file, and the unit tests and integration tests are also separated into different files. As an example, the User controller has test cases in the `UserControllerUnitTests.cs` and `UserControllerIntegrationTests.cs` files.

5.3 History

Figure 4 shows the continuous integration being performed on Microsft Azure DevOps. You can see the deployment pipeline of building, testing and publishing the API.

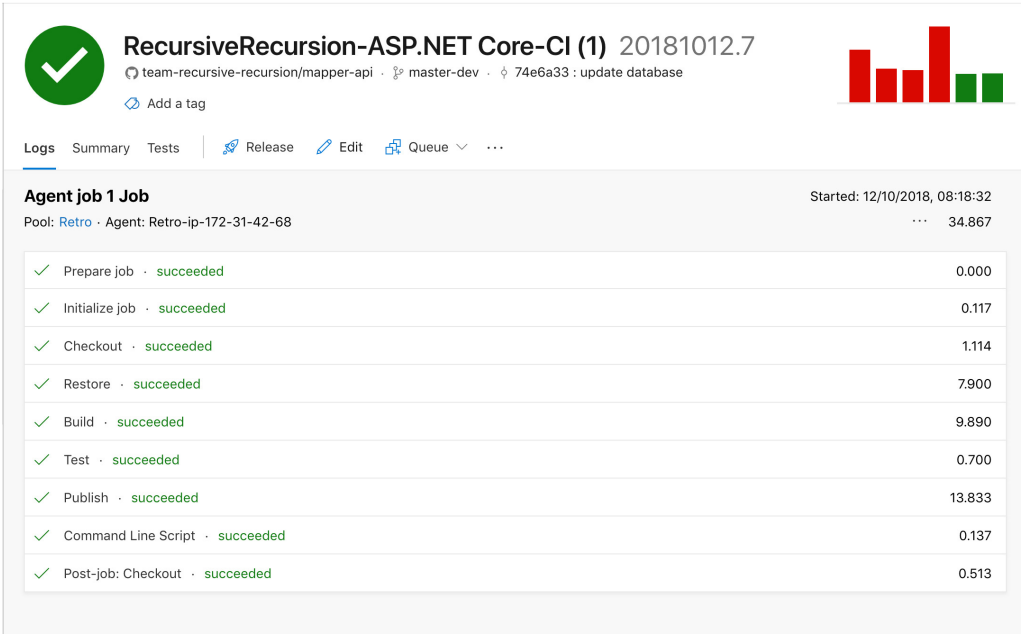

Figure 4: Continuous integration

6 References

1. Microsoft Azure DevOps (`https://devops.azure.com`)
2. Software Requirements Specification (`https://github.com/team-recursive-recursion/documentation/raw/master/publish/requirements.pdf`)
3. Usability Report (`https://github.com/team-recursive-recursion/documentation/raw/master/publish/usabilitytest.pdf`)