Assignment 4 – Acceptance Testing

SENG301 Software Engineering II

Neville Churcher Morgan English Fabian Gilson

1st April 2022

Learning objectives

By completing this assignment, students will demonstrate their knowledge of:

- translating acceptance criteria into automated acceptance tests;
- writing automated acceptance tests.

To this end, students will use the following technologies:

- the *Java* programming language with `gradle` dependency and building tool;
- the *Java Persistence API* with *Hibernate* and *sqlite*;
- the *Cucumber* and *Mockito* stubbing frameworks;
- the Apache *Log4j2* logging facility;

Next to this handout, a `zip` archive is given where the code base used for *Labs 3* to *5* has been updated for the purpose of this assignment.

1 Domain, story and acceptance criteria

This Assignment builds upon the *Wordle App* used during term 1 labs. The code base from *Lab 5* has been extended and modified to fit the purpose of this assignment. The user stories of current code base are listed in Section 1.1. A walkthrough of the code base is given in Section 1.2. Your tasks are described in Section 2 and the submission rules are stated in Section 3.

The domain model is reproduced in Figure 1[1]. Only domain classes are represented (i.e. we do not reproduce the Dictionary API classes for this assignment).

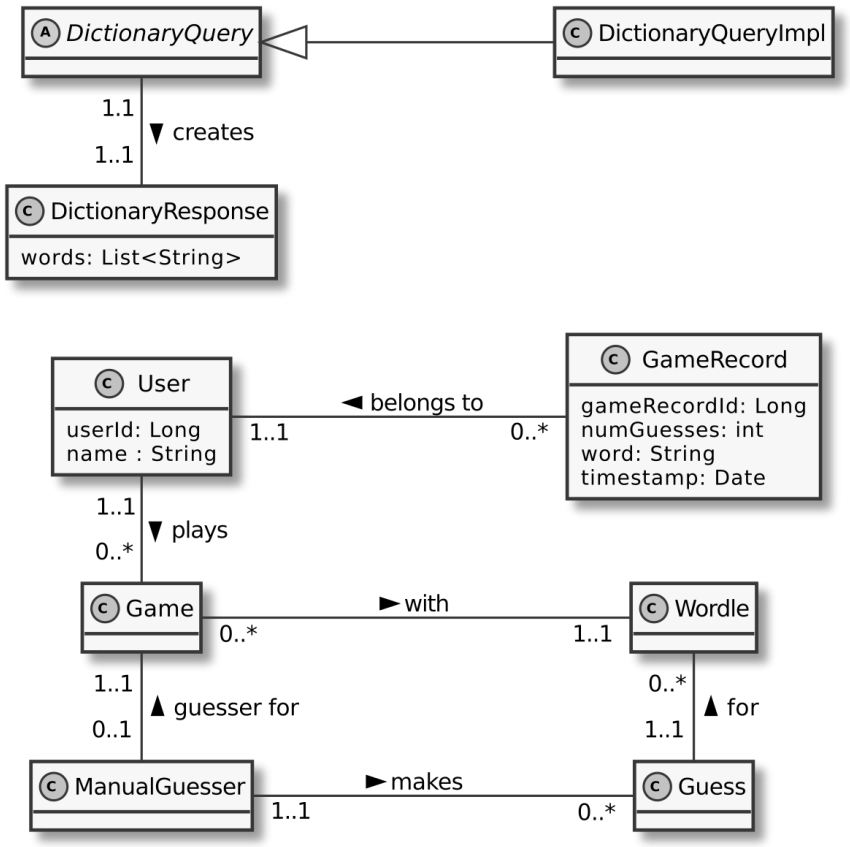

Figure 1: Domain model of *Wordle App* (updated from lab 5)

1.1 User stories of current code base

Let's assume your product owner came with the following user stories and acceptance criteria. For the sake of simplicity, we identified only one class of users for the system, called *"individual"*. This list expands on the user stories defined for the labs. Next to each AC below, we note which lab they relate too and if acceptance tests have been implemented already, *e.g.,* U2 AC3 should have been implemented in lab 3, but we supply a model answer in the attached code. Additionally, for U3, we supply the `features` file with the ACs in *Gherkin* syntax already.

U1 **As a player, I want to record the outcome of my wordle game so that I can keep track of my previous games.**

- AC.1 I can create a game record with a user, a word, and a number of guesses that are not empty. ***[Implemented in lab 4]***
- AC.2 Two game records may exist with the same attributes but must have different timestamps. ***[Implemented in lab 4]***
- AC.3 A game record can not have 0 or fewer guesses. ***[Implemented in lab 4]***

[1]Squares next to class attributes denote their `private` visibility. See Section 3.5 of `http://plantuml.com/guide`.

U2 **As a player, I want to retrieve possible guesses so that I can see possible options.**
- AC.1 I can get a list of possible words given the letters I know. ***[Implemented in lab 5]***
- AC.2 I can get a list of all possible words. ***[Implemented in lab 5]***
- AC.3 If no words match given the letters I know none will be returned. ***[Implemented in lab 5]***

U3 **As a player, I want to play a game by suggesting words one after the other so that I know how many guesses it took to guess a word.**
- AC.1 Every time I make a valid guess, the guess count is incremented. ***[Gherkin written]***
- AC.2 I cannot make a guess with strictly less than five letters. ***[Gherkin written]***

U4 **As a player, I want to see a colour coded reply from my guesses so that I know what characters are correctly guessed, placed or neither.**
- AC.1 A correctly guessed word will have all its letters coloured in green.
- AC.2 A correctly guessed and placed letter is coloured in green, a correctly guessed but not placed letter is coloured in yellow and other incorrect letters are coloured in gray.

1.2 Walkthrough the code base

The code base used for *Labs 3* to *5* has been augmented with more features and acceptance tests from past labs have been implemented for you to compare with what you wrote during those labs[2]. You should already be familiar with most of that code.

1.2.1 README

The `zip` archive with the code also contains a `README` and a `LICENSE` file that you need to consult prior going deeper into the code. This `README` describes the content of the archive and how to run the project. The code is also extensively documented so take some time to read the *Javadoc*.

1.2.2 Gradle configuration file

Take some time to review the `build.gradle` file. You should be familiar with its content as it is a complete version of what was needed to complete *Lab 5*. **You are required to keep this file untouched, failing to do so would prevent us from marking your assignment and you will be awarded 0 marks for the assignment.**

1.2.3 Model layer

In that package, you will find the domain model presented in Figure 1. An extensive documentation is provided with pointers to external references, so you should take some time to understand how this works.

1.2.4 Accessor layer

Accessors have been implemented for `GameRecord` and `User`.

1.2.5 Solver API

From *Lab 5*, you learned how to use an external API to retrieve possible guesses using a simple HTTP server and transparent JSON deserialization. The classes from *Lab 5* have been updated a little to abstract the actual implementation from the expected behaviour of the service[3].

[2] Be aware that small modifications in the code may imply that the acceptance test you wrote may be slightly different to the ones in the supplied code base.

[3] Do not be misslead, this is no *Façade* pattern since only one service is provided where the *Façade* abstracts a series of complex services. See `https://refactoring.guru/design-patterns/facade` for more details.

1.2.6 Automated acceptance tests

You can take a look at the three `feature` files under `app/src/test/resources/gradle/cucumber` to see the scenarios covering all acceptance criteria for U1, U2 and U3. The test code is placed under `app/src/test/java/gradle/cucumber` (for U1 and U2). **Pay particular attention to the naming convention we used. We expect you to follow that naming.**

1.2.7 Common line client (CLI)

After having taken some time to review the code base, you can run the project to get a deeper understanding of its existing features. Check the `README` file for explanations on how to run the CLI code (i.e. `$ ./gradlew --console=plain run`). The `main` method is placed in the `App` class. For help running different commands, use the command *help* to see a list of all commands accepted by the CLI. When running the `App`, all commands are displayed.

2 Your tasks

2.1 Task 1 - Write acceptance tests for U3 [2 x 10 marks]

In the `u3-manual-guessing.feature` file, you will find two scenarios for U3, but no test code has been provided. You are required to implement these *Cucumber* scenarios (**without changing their definition**).

When assessing this task, we will verify that:

- the automated acceptance tests effectively check the expected behaviour expressed in both ACs (i.e. scenarios);
- the tests are self-contained, readable and follow the design practices taught in the course (e.g., Lecture 8, additional material and labs);
- the test passes (**a failing test awards 0 marks for that AC**).

HINT: The input `Scanner` can be mocked, see `https://stackoverflow.com/a/31635953/5463498`.

2.1.1 Task 2 - Write *Cucumber* scenarios in *Gherkin* syntax for "*U4 - Colour coding of guesses*" [2 x 5 marks] and implement these scenarios [2 x 10 marks]

In Section 1.1, acceptance criteria have been given for user story 4 relating to colourised guesses of words. You are required to:

- create a new `feature` file for that story (**remember to comply to the naming convention**);
- translate the two acceptance criteria of U4 into *Gherkin* syntax (i.e. *Cucumber* scenarios);
- implement the acceptance tests for both scenarios into a new test class;

When assessing this task, we will verify that:

- you have adequately translated the acceptance criteria, i.e. the **behavioural semantics** of the *Gherkin* scenario **is identical to the English version** given in Section 1.1;
- the automated acceptance tests effectively **check the expected behaviour expressed in the AC**;
- the tests are self-contained, readable and follow the design practices taught in the course (e.g., Lecture 8, additional material and labs);
- the tests pass (**a failing test awards 0 marks for that AC**).

3 Submission

You are expected to submit a `.zip` archive of your project named `seng301_asg4_lastname-firstname.zip` on Learn by **Friday 6 May 6PM**[4]. **No other format than `.zip` will be accepted**. Your archive must contain the updated source code (please remove the `.gradle`, `bin`, `build` and `log` folders, but keep the `gradle` - with no dots - folder); **do not remove the `build.gradle` file or we will not be able to assess your assignment and you will be awarded 0 marks**.

Your code:

1. **may not** import other libraries / dependencies than the ones currently in the `build.gradle` file;
2. **will not be evaluated** if it does not build straight away or fail to comply to 1. (*e.g.,* no or wrong `build.gradle` file supplied);
3. **will be** passed through an advanced clone detection tools (*i.e.* Txl/NiCad) that proved to be performing well on sophistically plagiarised code earlier.

The marking rubric is specified next to each task (overall **50 MARKS**), but is summarised as follow:

Task 1 write acceptance tests from given scenarios for U3 **2x10 MARKS**.

Task 2 translate both ACs from U4 into *Gherkin* syntax (*a.k.a. Cucumber scenarios*) **2x5 MARKS**, and write the tests for these two scenarios **2x10 MARKS**.

4 Tools

You only need the following tools to run and develop your program if you work on your own computer:

- an IDE, *e.g., IntelliJ IDEA* `https://www.jetbrains.com/idea/download/`
- *OpenJDK Java SDK* `https://openjdk.java.net/install/`
- for *Windows* users, setting up your environment variables for Java to be recognised: `https://stackoverflow.com/a/52531093/5463498`

The code you receive already contains the minimal binaries for `gradle` that will manage the dependencies for you. Please refer to the `README` shipped with the code for more details. You should be familiar with the process as it is the same as for term 1 labs.

[4] There is a one week grace period with no penatly. No further extension will be granted, unless special consideration.