Kyle Riggenbach

CS 320 Testing

Oct 15 2023

Aligned to the software requirements

As requested by the customer, I created unit tests for the contact, task, and appointment services throughout this course. To make sure the tests worked, I used a mix of black and white box testing methods to cover as much as possible that could go wrong. The black box testing is when data that is entered is checked to make sure that the correct result happened. The white box testing is looking at the code itself to make sure the logic and flow works. This generally aligns with what the client requested throughout the different projects in the course

Effective coverage percentage

I used JUnit testing to try and have the best coverage percentage as possible but I am sure there are additional things I could have done to make it better in terms of coverage. The amount of edge cases that could happen in software is astounding and making tests for all of them is just an ongoing thing that we try and cover the best we can.

Technically sound

I enjoyed the experience of writing the tests. Usually on projects in the past, I personally would be the one testing. These unit tests were helpful to make sure as much as possible was tested to be technically sound. I have more confidence in the resilience of these projects we created in this course because the testing was automated.

Code is efficient

To try and make sure the code is efficient as possible, I tried to refactor and reduce the code as much as possible. I have done some leetcode stuff and generally I tried to leverage the same techniques of minimizing the amount of loops and conditionals. This hopefully helped in creating decently efficient code in the end.

Software testing techniques

The majority of the testing techniques I used fall into the category of white and black box testing. Black box testing entails having data inputted and checking if the expected or appropriate output was generated. White box testing entails looking at the code itself and making sure the logic and flow are correct. Then JUnit was also very helpful to look at the edge cases and potential errors that could happen.

Not used techniques

We did not measure the performance of any of the apps which is a large reason to do testing. This usually includes checking for how the app will scale, how fast it is for the user, and how much memory certain things take up. Then security is also another large area we did not cover which would be stuff like sanitizing inputs and such.

Practical use

The practical use for the testing techniques I used is to check for the accuracy of the information coming in. Then to check for the overall efficiency of the code itself. Then they are

useful to find edge cases that need to be addressed like a hyphen in a name that accidentally gets sanitized out. If these techniques were not used, it would be hit or miss if manual testing was able to find all the potential errors before sending the code to production.

Mindset

While working on the code, I wanted to keep an open mind to using these testing methods. In all my previous projects with either school or a personal project, I manually tested everything. So getting familiar with an automated way to do it was nice to go through.

Limit Bias

I limit my bias by assuming everything I did in the code is going to produce an error. So making a test for as much as possible helps remove the bias that I am testing my own code.

Being disciplined

Right now how I view this is that I just need to get a minimal viable product (MVP) up to get familiar with how the thing I am building needs to work. Then I go into it expecting to have to do a full rebuild once I am more familiar with how this thing will work. So being disciplined in getting the full rebuild done is the most important part. It is possible to build the MVP then just go on using it without any refactoring or rebuilding.