NAU NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

Software Testing Plan

April 3, 2022

Team Teacher To-Do

Project Sponsor: Chris Aungst

Graduate Mentor: Italo Santos

Faculty Mentor: Michael Leverington

Version: 1.0.0

Team Members:

Sam Gerstner (Team Lead)

Alexander Frenette

Noah Nannen

Shlok Sheth

Table of Contents

Introduction

In Arizona, the number of teachers has been steadily dropping. The Arizona Department of education has created a new program to help halt this steady decline, known as the Student Teacher Intern Certificate, or STIC. The NAU (Northern Arizona University) College of Education has a program in place to help speed these soon-to-be student teachers along. However, their program is not efficient and takes many hours to review and implement over the semester. We are working to mitigate the amount of time spent on this program by putting the power into the hands of the students by way of a web application.

For our team, we have created a student teacher intern tracking system that allows students to track requirements and be able to fulfill these requirements on their own terms. After this part of the process is complete, administrators will be able to look at the student's requirements and decide which ones are able to be marked as completed and which ones are to be rejected. By doing this, we change the dynamic of the original setup of the process. Initially it was the administrator having to review every student to ensure they are on track. But with our system, the students are the ones who have the responsibility. The administrator will have to monitor all the students, check the pending tasks, and ensure they are up to the program standards to approve them.

Software testing for our project has two purposes. The first is to ensure our project has met all requirements from the project sponsor. We set up tests to make sure the product is satisfactory and able to carry out the day-to-day activities that are needed to maintain the project's usability throughout its life cycle. The second purpose is to test the project's security and stability, making sure it can continue working on its purpose and will not break under certain conditions or be dismantled by outside users. Software testing is necessary for our project, as we want to be sure that we are delivering a safe, secure, reliable product to our client, and not inadvertently shipping a product that is defective or inaccurate to what the client would have wanted.

Because of the professional and high-traffic nature of the project, we have a lot of tests built directly into our project, be it in JavaScript, the Java controller classes, AWS package checkers, or even Boolean flags to check if code was executed. We have many of our tests at every code execution in our project, ensuring that if any part of our code were to be even slightly off, the code would stop before it snowballed into a major issue.

Unit Testing

Unit testing is a type of testing used in software development that focuses on individual units or components of a piece of software or system. Unit testing is typically performed by developers before the system is fully integrated and tested. Unit tests are automated and run each time the code is compiled to ensure that the new code does not break existing functionality. Primary goals of unit testing include isolating a section of code, verifying the correctness of the code, to help developers understand the code base and make changes quickly.

In this project we will make use of the JUnit unit testing library for Java. JUnit is a unit testing framework for the Java programming language and is widely used by Java developers for both small and large-scale projects. JUnit has become the most popular unit testing framework for Java and many Java IDEs have built-in support for writing and running JUnit tests.

For our project we have been using Maven. Maven is a build tool that operates on the concepts of build lifecycles. Some of the more important lifecycles are compile, test, and packaging. These names, as they suggest, must deal with compiling our source to runnable code, then testing our code against the JUnit tests we provide, and then finally packaging our source as a runnable Jar.

While unit testing is used to expose faults in logic, it is therefore our objective to create unit tests that we believe to be edge cases to break our code. Using this strategy, it is our objective to uncover edge cases that were not considered during our initial implementation. To verify the correctness of our initial logic and implementation we also find it important to consider code coverage. Considering code coverage allows us to isolate sections of code that might be incorrect, or simply unnecessary, and require further investigation. While there is no magic number for an acceptable percentage of code coverage, we believe that a good model to follow is that of Google. In a blog post titled Code Coverage Best Practices, they set the standard of 60% as acceptable, 75% as commendable, and 90% as exemplary. These standards are ones we feel comfortable following and will be implemented in our project.

Our project can be broken into two sub projects. The first being our web application, which is responsible for delivering the web page to our users. Our other sub project is our API. Our API is responsible for most of our business logic and handles data management. In other words, our web app is our view, and our API can be thought of as our model and controller. Each of these two projects thus have completely different testing approaches. The API is easy to mechanize and implement JUnit testing. Our Web app however requires a different approach.

API Unit Tests

Our testing methodology for our API is not to simply test each function. It makes more sense for us to test end points of the API itself. While there may be many functions that are called when retrieving a user, the API consumer only conceptualizes the single endpoint GET::/users. This leads us to believe that due to our limited resources, it is more efficient to begin by testing the endpoints themselves, and upon encountering errors, then further exploration can occur if necessary. For our API it is difficult to establish traditional boundary values. There is no minimum or maximum acceptable value, instead our API relies of existences of relationships. For if a user were to modify a task resource, we are only really concerned with the existence of the resource. If a resource does not exist, that would result in an error, while a resource that does exist would result in a successful operation. Due to lack of boundaries, it is also the case that equiveillance portioning will not be possible. One might argue that the partition could be the programs that the students are in. Again, this is not the case, as the logic in our system does not have the concept of student programs, and instead only cares of the existance of a relationship between student user, and program. The actual instance of the program is not of significance, nor does it change the logic of our system.

One of the critical aspects of testing our API is initializing our backing data with predetermined data. To do this, we have created a module that will allow repeatable system initialization. This means we will generate students, requirements, majors, and all other database entries to model real-life use. This then allows us to validate the correctness of the database insertion, database retrieval, and modification. Having this module is critical in the evaluation of correctness as it reduces the duplication of code within the API testing itself and can be used by the Web Application as well. This is because our progenerated model will be used for creation of requests while testing our DB Helper as well.

Some of our endpoints/units which will be tested are as follows. For Further reference of any of our units being tested, please reference our source repositories at https://github.com/TeacherTodo

URL: */students/*

Method: GET – will return a list of students, we will be testing the resulting students based off of the pre-generated data we have discussed.

Method: POST – will create a user, our input will be a generated user, and the expected result is to have our user inserted into the table, which will be checked using DB_Helper

URL: */students/:id:*

Method: GET – will return a student, we will be testing the resulting student based off of the pre-generated data we have discussed.

URL: */requirements/*

Method: GET – will return a list of requirements, we will be testing the resulting requirements based off of the pre-generated data we have discussed.

Method: POST – will create a new requirement, our input will be a generated requirement, and the expected result is to have our requirement inserted into the table, which will be checked using DB_Helper

URL: */programs/*

Method: GET – will return a list of programs, we will be testing the resulting programs based off of the pre-generated data we have discussed.

Method: POST – will create a new program, our input will be a generated program, and the expected result is to have our program inserted into the table, which will be checked using DB_Helper

NAU NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

Web Application DB Helper Unit Tests

The web application component of our project contains a class titled DB_Helper that contains supporting functions for the application that send HTTP requests to the API to get needed application data. These functions heavily use both the OkHttp library for sending HTTP requests as well as the Jackson JSON Data Bind library for serializing and deserializing JSON objects. These functions fit into one of four main categories which will help us further section off code to be tested.

Student Functions: *isRegisteredStudent(), getStudent(), getAllStudents(), createStudent(), editStudent()*

Administrator Functions: *getAllAdmins(), createAdminUser(), deleteAdmin()*

Document Functions: *getFileContent(), uploadFileContent(), getAllDocs(), getPendingDocs(), createDocument(), editDocument(), deleteDocument()*

Requirement Instance Functions: *getStudentRequirements(), createRequirementInstance(), editRequirementInstance()*

Admin CRUD Functions: *getAllMajors(), getAllTerms(), getAllRequirementStatuses(), getAllApprovalStatuses(), getAllRequireemnts(), getRequirementByID(), getMajorByName(), getTermByName(), createRequirement(), createMajor(), deletMajor(), deleteRequirement()*

Testing all the functions in this class should be straight forward. If the function accepts a JSON request, first we validate that the request could be deserialized into the appropriate Java object for the request. Next, we confirm that the OkHttp library was able to submit the request to the API and got a response back from the API. Finally, we verify that the response from the API can be deserialized into the appropriate Java object of interest. Each one of these checks will need their own assert statement within

the unit test; This means that each unit test for these functions will contain at least two but up to three different assert statements. JUnit contains a wide variety of assertion types that will allow us to fine-tune our unit tests to check for extremely specific conditions within the application. Because OkHttp returns a generic response object, these different assertion types will be particularly useful.

One aspect in which our unit tests will not be as helpful, is if there is a Java exception thrown from a DB_Helper class method. These methods throw a generic Java exception, and unit tests will not help in diagnosing a specific exception, we will only see that the test failed. This means that if a unit test fails due to an exception, the root cause cannot be determined by the unit test and must be manually investigated by a developer. We may be able to improve the testability of these methods by changing how exceptions are thrown from these methods, but for this iteration of the project, our current architecture will suffice.

Integration Testing

Integration testing is the idea of testing multiple pieces of code together to see if they work with each other and can transmit the necessary data back and forth. Integration testing consists of individual components combined and tested as a whole group to ensure integration across the application. Our goal for testing the capstone project would be to ensure correct integration of functions across the functionality of the application. Another goal is to validate the data flow and communication protocols between different controllers of the program. After unifying our project will the different pieces of application, we have a complete product to test. The main objective for us is to determine that each controller can fit to each other like a puzzle piece and the information being shared is maintained correctly.

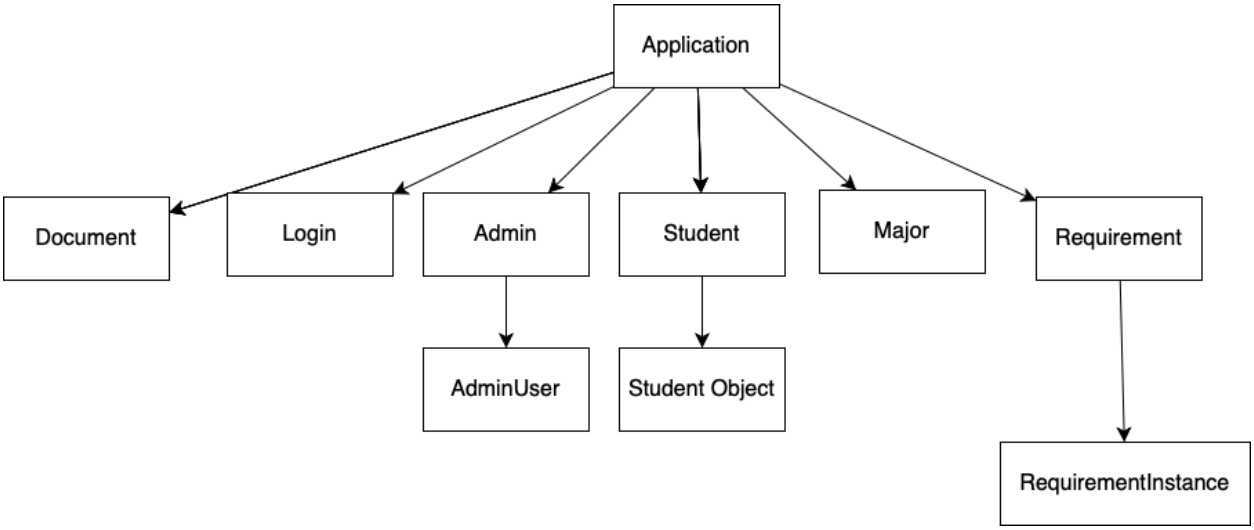

Figure 1.1

From this image we can see that the application is using a lot of controllers to support the function and provide feasibility. Although the way to check the information being loaded to each controller and being returned from the controller, which involves integration testing.

Our first take on implementing integration testing would be using a set of print statements and printing out the variable values from the function parameters and return values of functions in the controllers. Anyhow this will require us to map down the cases of logging in as a student, administrator, as a user not in the database. This will involve checking if the database and the information being displayed on the website are the same and valid. Furthermore, if we observe the source code of the web application there are a

lot of tunnels in it, tunnels are a space where the variable from a particular method and function, is being passed on to another function for further use. Our aim currently is to verify the information being sent over is correct.

Integration testing related to our capstone project can be defined for 5 different web pages, requirements page, major page, user page and document page. The requirement page is objected to add and update requirements for s for the students, it allows the CRUD operations although this needs to be reflected in the database and to the system. Integration testing in reference to this can be done by adding the requirements and mocking a student user to see if the new or update requirement is being reflected in the to do list. The admin web page has the ability to add admin users to the system and delete them. This can be tested by adding multiple admin users and checking if that is being reflected in the SQL database. The student portal webpage displays all the requirements and has the ability to upload documents and mark status of certain tasks, this webpage can be tested too. These are different pages that communicate with the database, on which the whole program relies on.

Integration testing is the next step of unit testing, it ensures that all the parts of the program when put together work perfectly and there are no bugs involved. Our capstone project obviously is made of different components. The first component that the user (student/admin) comes across is the login page, the information from this is passed on to other web pages using a cookie, this cookie has the OAuth token so that the user does not have to log in once they have been redirected to the next page. As part of integration testing this can be tested by checking that the cookie value does not change for the entire session and each controller is accessing the same cookie using the same cookie ID.

The admin must approve any documents uploaded by the student related to requirements, it has to be approved by the system admin in order to fulfill the requirement. Considering this the next step of integration testing would be to check the document when uploaded by the user is being attached to their user id and saved at the Amazon S3 document storage bucket properly and this file can be seen in the admin dashboard to be approved for the individual. Following this another test would be to see if the approved document or denied document can be reflected in the respective student's dashboard. The requirements of the program keep changing and hence we have implemented the feature to add, edit and delete requirements for all the majors. The major feature that we

must test is that the requirements that are being added by the admin are reflected in the students' dashboard for that major.

Usability Testing

Usability testing is a version of testing that relies on a user being able to be hands on with a product. It involves a closed test between a user and a facilitator, with the facilitator giving the user a task and seeing if they can complete it successfully and without too much trouble. The main goal is to see what users can do without trouble and where they may run into issues if the path to the task goal is not clear enough. This allows project developers (primarily UX developers) to be able to identify weak points in their designs and be able to create an experience directed towards ease of use for a user.

Our plan for Usability testing revolves around several tasks that users will complete. For our project, we will have two user identities: the student and the Administrator. These are the two possible types of users being able to access the web application. Many of these will be Student users, representing the substantial number of students that will be in the STIC program using the application to be able to get the certificate. While most of the program will be used quantitatively by Students, it will be used more qualitatively by Administrators. These will be the people who manage the student requirements and are responsible for maintaining student progress throughout the program. They will also maintain other Administrator users and ensure that only the correct people can access the Administrator landing page that displays the students' data.

Student:

The students' tasks will be basic. The student user is not meant to handle many technical-based items through our web-based app. For most of the students' experience, we want to keep the application as simple as possible. So, we have created tasks that focus on simplistic items on the webpage, making sure that they are as straightforward as possible.

Student Tasks:

1. Log In to a Student Account
2. Check a Task's Status
3. Upload a Document to a Task
4. Submit a Task

Administrator:

The Administrator's tasks will be more complex. Because of their heightened responsibilities, it is especially important that we can have Administrator users be able to handle each task easily and efficiently. The majority of the Administrator's task will have to do with editing requirements, something that should very rarely happen, however it is incredibly important to ensure that the process is clean and simple, as it can affect a student user's ability to earn their certificate. The other half of the Administrator's responsibility comes from approving Student tasks to be able to show the student as completed in their requirements.

Basic Administrator Tasks:

1. Log In to an Administrator account.
2. View a student's tasks.
3. Approve a student's task.
4. Reject a student's task.
5. Filter Students based on task completion.
6. Filter Students based on UID.

Editing Administrative Users Tasks:

7. Add a new Administrator User
8. Remove an Administrator User
9. View all Administrative Users

Editing Requirements Tasks:

10. Add a new Requirement.
11. Add a new Major.
12. Edit an Existing Requirement
13. Edit an Existing Major
14. View all Requirements.
15. View all Majors.

We will be using expert reviews, as we will be utilizing actual STIC program students and administrators for these tests. Since our product is not meant for public use, we believe that it would not be prudent to include focus groups in the project. We may be

utilizing some realistic user studies; however, we are unable to be sure of that. We will meet with our client to review his ideas for the software; however, we have many different administrative users that can be tested to ensure we have a product that is to their satisfaction. Once we can meet with the client, we will be able to meet with the other officials that are going to be using the product, the administrators that will be using the program. Since we are unable to access the student users through our testing plan, it will be up to the administrative users to confirm the student's usability. We are giving ourselves 4 weeks of preemptive testing and modifications to be able to modify our product to best serve its users. During these 4 weeks, we will be keeping our testing process up to date, and reviewing with the administrative users to ensure that all additions are made with the correct functionality in mind.

Conclusion

So, in review, our project has been extensively tested, and will continue to be in the future. We will allow administrators to access these tests to ensure the product can be checked for malfunctions at any time to support the project even after we stop working on it. Our unit tests will remain in place, with their outputs hidden but still able to be checked if necessary. Our integration testing, while internal, will be able to check and show the webpages working together with their Java controllers doing the same behind the scenes. The usability tests are going to be the only thing not easily accessible, but that is mostly due to them not being needed unless something goes horribly wrong. If usability tests are needed, they will remain, though they should not be necessary to a working product.

Overall, we have developed a working, suitably tested product that we are able to deliver to the College of Education, as our software tests will cover all our projects' uses and we will be able to pinpoint where any errors may be occurring, if at all.