Team 28: DocuSign Integration

4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopting test strategy and instruments. In this overarching introduction, given an overview of the testing strategy. Emphasize any unique challenges to testing for your system/design.

4.1 Unit Testing

We will be using unit testing to prove that the code we are writing works properly before adding it to the code base. It will also ensure that no other parts of the code are broken as a result of any recent changes, and protects our codebase from regression. The primary tool that we will be using in our unit testing is mocking. Mocking will allow us to test individual functions in our code without having to worry about the test failing because an outside method isn't working properly.

4.2 Interface Testing

What are the interfaces in your design? Discuss how the composition of two or more units (interfaces) are being tested. Tools?

There are three interfaces within our design: The front-end, back-end, and the DocuSign API. The connection between the front-end and back-end will be automatically tested using unit tests and mocks, as well as functional tests. The same goes for the connection between the back-end and the DocuSign API.

4.3 Integration Testing

The standard need for integration testing does not directly apply to our project. Generally, integration testing is done by adding a new component into the greater system and using a service that is capable of imitating the actions an end user would take. Thus verifying that the new module is functional as a part of the whole. Our project is to provide a proof of concept for BuilderTrend to verify the viability of DocuSign. Thus they will be the ones integrating it into their systems.

Integration testing is not necessary as we are not "integrating" the project into something else. We will of course be performing system testing which is similar to integration testing, but limited to the project itself. Information on that may be found in the following section.

4.4 System Testing

To make sure that a complete system works, we will need a set of tests (unit, interface, and integration tests) that show complete functionality over the use cases described in the original plan. Main use cases that were listed in the requirements documents are going to be tested with unit

testing. This is to verify that the main use cases are working correctly with use cases that are independent of each other. For our project the main use cases would be the following:

- Have a way that a builder + contractor can electronically sign a document
- Allow multiple contractor signatures via the Docusign API
- Have a way that a homeowner + builder can electronically sign a document
- Thorough documentation about DocuSign's API and the use case solutions

With interface testing, we verify that use cases that have common functionality with each are working. In this area of testing, we may also identify places in the code where duplicate functionality exists and combine these into one piece of code. While we dont expect that to happen, it could happen because of the concurrent development that we have. In a more specific example, we will expect to see the integration of a front-end and back-end interface with our software. This is described more in the interface section of this document above.

While our project may not have integration testing, if we eventually get to the Buildertrend source code portion of the project, we may expect the need for integration testing with systems that Buildertrend already has to validate a working system.

4.5 Regression Testing

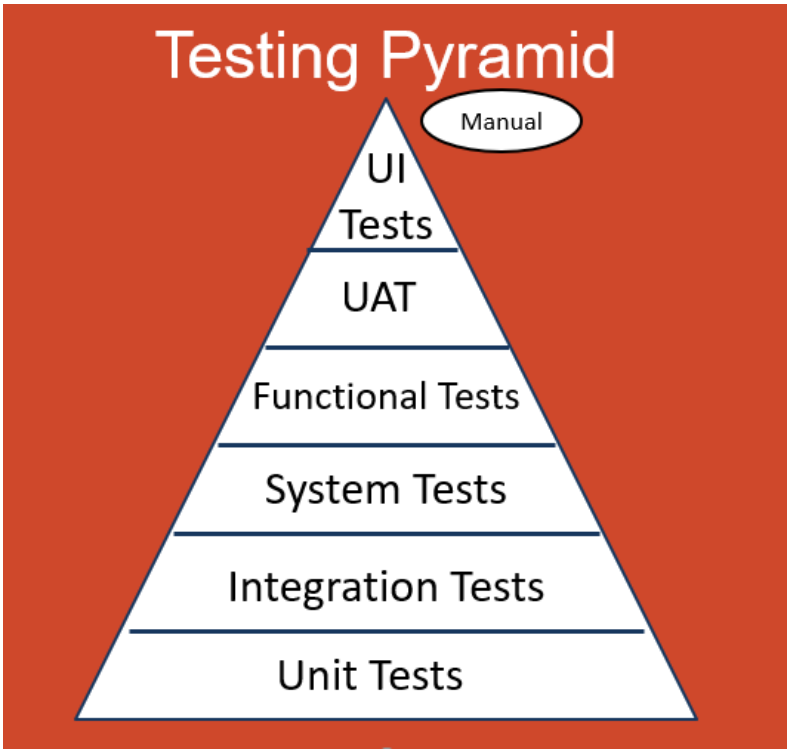
We will ensure that any new additions do not break functionality by breaking up our code into separate units. Each unit can then be modified and add the capability to use other units. By making sure that each unit talks and communicates with each other separately, we can then integrate new features without worrying about breaking old interfaces. We need to ensure that the requirements BuilderTrend wants are kept working while adding new features if they request them. To add, all of this is what will happen if BuilderTrend requests we integrate anything with their code. This is because they want us to deliver a working model to show itIs feasible. We will use mocking as stated above as our tools of testing.

4.6 Acceptance Testing

All requirements will be fully visible and usable in our minimum viable product - the web application. This will demonstrate the functionality that BuilderTrend requested, as well as the documentation to support the decisions we made and our recommendations for the future. We will present this to BuilderTrend, and ask that they test all of our functionality to see if it fits their requirements. We expect this to be an ongoing process next semester, so acceptance testing will consist of real usage and feedback from our client.

4.7 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.



Testing ensures that our project satisfies all the requirements specified by our client, and does so efficiently and as expected. By following a pattern similar to the testing pyramid above, we can easily verify all the use cases that BuilderTrend has mentioned. Also, we can more easily identify issues based on which layer of tests they affect. In the pyramid, the UI Tests layer maps to our acceptance tests - where real users (developers and clients) can test our application to ensure that it satisfies all requirements.