Testing Plan

Michael Riess, Jonah Kubath, Matthew Peter

# **1 Unit Testing**

**1.1 Overview**

Because we are using both Electron and Angular-6 for the vending kiosk application, it is more intuitive and easy to split the unit testing between the two.

The first tests will use Jasmine and Karma for the Angular-6 specific code. These tests ensure that all events are handled properly, and that all functions return correct results or perform the correct actions for all possible input scenarios. Tests are written in the spec typescript files for each component.

The second testing framework we will use, called Spectron, focuses on the Electron specific dimensions of the testing. Spectron will be used to test the base functionality of the program. Examples of this include ensuring that the window opens to the correct size, that program has access to the OS’s file system, and that the program receives accurate mouse/touch events.

All tests, those for both sets, will be run using the ‘make test’ command. Results will be printed for only those tests which fail, and the results will also be written to a log file.

Component-specific unit testing will be performed after any change to the code in order to ensure that all bugs and breaking changes are corrected immediately. Additionally, full, end-to-end unit testing will be performed before each major and minor version release.

**1.2 Testing Frameworks and Tools**

Spectron is a promise-based testing framework for Electron built on ChromeDriver and WebDriverIO. This framework was chosen because it is free and open source, and because it has full support for all Electron APIs.

Jasmine is a lightweight javascript testing framework that relies on no other Javascript frameworks. We chose to use Jasmine because it is small, and uses simple, easy to read syntax. Additionally, because the application was built using the Angular-CLI, it automatically generates an end-to-end testing suite with all the necessary files for each angular component. This suite includes the Jasmine framework by default.

Karma is a framework-agnostic javascript testing tool built by the AngularJS team in order to make testing simpler. As with Jasmine, Karma is a default part of the Angular-CLI testing suite.

**1.3 Documentation**

* Jasmine: https://jasmine.github.io/
* Karma: https://karma-runner.github.io/2.0/index.html
* Spectron: https://electronjs.org/spectron

**2 Usability testing**

**2.1 Overview**

In order to reach optimal usability, we will have both the client and a select group of participants test the applications. The client will ensure that the agreed upon features and standards have been met.

**2.2 Results**

When it came to the web browser, they were asked to do the following:

* Open a new tab
* Go to a web page of their choice
* Scroll up/down the web page
* Use a search engine to find a web page and select a link
* Open as many tabs as they could and determine why they were limited
* Attempt to close the web browser

Along with the web browser, we also had some tasks for them regarding the vending machine app:

* Scroll between the products
* Select a product
* Identify the product’s name, description, and price
* Change the quantity of the selected product
* Cancel the purchase of a product
* Purchase a product

While the testing was going on, all we did was tell the user what task to perform next. They weren’t told whether they had performed a given task correctly and we didn’t answer any questions they had regarding the current task.

Overall, the testing for most of the users went very well once they figured out how everything was laid out. The tasks that took users the most time to complete were scrolling up/down a web page and going to another web page for the first time. Most users attempted to scroll up/down the web page simply by swiping up/down the page rather than using the scrollbar (which is understandable given that this works on most touch devices) and tended to look towards the bottom of the screen for a touch keyboard when it came time to enter the web address (which is also understandable given that this is where the touch keyboard pops up on most touch devices). Other than these tasks, users appeared to have little difficulty when it came to understanding how the web browser and vending machine app worked.

**3 Security testing**

**3.1 Overview**

Because the vending machine kiosk will be handling transactions and user’s internet browsing, there are many potential security issues. Those associated with transactions are outside the scope of this project and will be handled by third-party software in a later version. In order to protect customers who use the kiosk’s browser, personal data is not logged, and users are unable to download anything or visit known bad websites. Access to any part of the system other than those applications that are explicit features of the kiosk is limited to those users who have maintenance or administrative accounts and are logged in.

Security testing will be performed before major version releases.

**3.2 Steps for Ensuring Security**

* Ensure that only absolutely necessary network ports are open, namely, those used to communicate between the vending machine and the center of operations, and those used in internet browsing.
* All communication between vending machines and the center of operations is encrypted and uses java-web tokens for API calls.
* All purchases will be handled by a third-party who is responsible for ensuring the privacy and security of customer data with regards to transactions.
* No customer-specific purchase data will be logged.
* Users will not have browsing history, usernames, passwords, etc logged.
* The cache will be cleared, and all tabs closed, after the browser is not in use.
* Users will not be able to download anything.
* Users will be limited in the number of browser tabs they can have open which will help to prevent possible security issues associated with excessive memory use.
* A well-maintained, third-party blacklist of blocked websites will be used in the browser.
* Users will not be able to exit out of any of the programs associated with the vending machine kiosk.
* Users will be unable to access the filesystem via the browser.
* Access to the vending machine OS will be limited to users with an administrative or maintenance account, and who are logged in.

**4 Accessibility Testing**

**4.1 Overview**

The accessibility testing is, by nature of the project’s scope, limited to color contrast testing for the vending application. For this, we will use the Colour Contrast Check for ensuring satisfaction of Checkpoint 2.2 of the W3C. The requirement reads, “Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen”.

The vending portion of the kiosk app has been modeled after traditional vending machines and other digital vending machines, and so matches the industry standard for things like size of product images. Additionally, the product images are to be treated as immutable for obvious trademark reasons. The ad and browser portions are also outside of the scope for accessibility testing since their content is determined by those who purchase advertising space and those currently browsing.

Lastly, as the application does not use sound, no accessibility testing is required for deaf or hearing-impaired users.