# Kaleidoscope

Iteration 2 Progress Report

2019-04-11, version 1.0

## Document Revision History

**Rev. 1.0 2019-04-11: Initial Version**

## Summary

For iteration 2, we started binding the model, view, and controller components together. Pressing buttons on the view is able to communicate with the controller, which then is able to communicate with the model. Background was able to get the key combination working, and allowing the user to interact with the program behind a transparent window. We were also able to have a notepad widget open up.

## Questions

1. What were the main difficulties so far?

HTML and CSS code on a MacOS does not look the same on Windows or Linux, so to ensure that the main application can look as uniform as possible, there had to be workarounds such as making the scrollbars invisible. The main application and the settings page has most of the HTML and CSS finished. All that needs to be done is have JavaScript be added to make the two pages more reactive, which will need to be learned.

Background was able to get the the key combinations to open up the overlay to work. So far we cannot run the C code for the key combination on any operating system other than Windows. Not everyone is able to have the current version of the code with all the implemented functionalities, because the C code would not allow the main app to run on MacOS or Linux.

For the Main App, there have been growing pains with integrating the View, Model, and Controller in the MVC architecture. The Model and Controller are well-integrated, but there have been some issues binding the Controller to the View. We are able to register some of the events that occur in the UI, but we are currently hardly able to modify the DOM (document object model). This is especially difficult as there is no good point to start at. Implementations in that regard will need to start by populating the UI with dummy data and verifying that it can be modified.

1. Were there any features you did not implement as planned, and why? Are you pushing some features to later iterations, and if so, why?

Initially, the plan was to have the Controller do a lot of the required logic, but we have been able to do a lot of logic in the Model. This makes the Controller more of a wrapper for the Model. It mainly calls a Model function and then calls a View function if necessary.

For the overlay, we planned on having one big window with widgets inside, but now we are generating windows for each widget and placing them throughout the screen. This has made the implementation process much easier, and gives us the ability to interact with the widgets.

1. What tests did you prepare for this iteration, and what are they covering? Did the tests you wrote deviate from your plan? What features are you not testing yet? Did you use any test frameworks, such as JUnit, the Android Monkey, Selenium, etc.?

So far, we have unit tests for the Model and Controller of the MVC architecture and integration tests that integrate these two components. These tests ensure that a variety of actions in the UI are executed properly. These relate to pressing different buttons and ensuring that Model data is created, saved in the local file system, deleted, or duplicated. These tests do not deviate from the plan, but some tests are currently not implemented. System tests need to be written for the Main App in addition to the Overlay.

For a testing framework we used Mocha and Chai. Mocha is used to structure/outline unit tests, and Chai is used as an assertion library. Istanbul was used as a code coverage tool. We plan to also use Cucumber for writing system tests, and the UI will be driven with Spectron.

1. Give a sample of code coverage tool output both before and after adding to your test suite. What did you learn from the code coverage data? How did you use this information to expand your test suite and improve coverage?

Output is on the next page.





These two outputs show the difference in code coverage for adding a test on the Model.js file. Initially, the coverage is 86.36% for statements, 50% for branches, and 75% for functions. After a new test is added, the coverage increases to 95.45% for statements and 91.67% for functions, leaving branches the same. From this change it can be deduced that the Model is almost nearly completely tested, but requires another test or two. In general, it seems that the Model is well-tested, but testing should focus on the Controller and Preset files as those could use a significant increase in coverage. Specifically, even though the Model utilizes the Preset class, it has become apparent that the Preset class is not well-tested at all.