**Slopes Testing and Validation**

**Introduction**

Slopes is a mobile app that provides users with up to date mountain information to help them make an informed decision about which mountain to board or ski that day. Slopes is being developed with the Ionic framework, and will be deployed onto both iOS and Android. Our app is written in HTML, Javascript, CSS, and has a database in mySQL. Since we are developing in several languages with different components, we will have to test the code in modules as well as a whole system.

**Verification Strategy**

Slopes is a user interface (UI) driven application. The backend is relatively straightforward, so our focus was on making the user interface as clean and usable as possible. Our team is fortunate to be made up of skiers and snowboarders, so we based our initial UI on our individual needs. We developed slopes with the end users in mind: skiers and snowboarders. We used our teams background in skiing and snowboarding to gather feedback to help us develop a UI that is streamlined and gives the user the information they want, fast.

In order to get feedback, we developed GUI mock-ups and a very early interactive version of our app. These gave the feedback group a good idea of how everything will be laid out, where they can find the information they’re looking for, what the final product will likely look like, and more. We we met on a weekly basis to examine the design of the app and ensure that it was on track to match our requirements and design.

**Non-Functional Testing and Results**

In order to test the performance of our app, put it through extensive testing on multiple devices, spanning different OSes. Our plan was to test scalability by adding more information to our database to see how it affects speed and general performance. However, because we did not get a change to automate adding mountains to our database we did not get a chance to test scalability.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test #** | **Requirement Purpose** | **Action/ Input** | **Expected Result** | **Actual Result** | **P/F** | **Notes** |
| 1 | Giving the user information they request | User Requests to look at data | Data is loaded onto page | Data is properly loaded and displayed | P |  |
| 2 | User is able to input personal information | User Requests to update information | Fields where the user can input information are opened | Sliders are displayed, and information can be changed | P |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | App suggests appropriate mountains based on the user’s personal info | User requests to see the suggested mountains | Appropriate list of mountains is displayed according to the user’s personal information | List of mountains is displayed, ordered by users preferences. | P |  |
| 4 | App refreshes mountain information so it is up to date | App checks if user has refreshed or recently opened app | App refreshes the data from online accordingly | Data is refreshed if the device is connected to the internet | P |  |
| Unit Tests | Unit tests for app services, to ensure that each service function works as described | Specced tests written by developers, run using Karma | All unit tests run over Karma pass. | All unit tests run over Karma pass. | P |  |
| Integration Tests | Integration testing on app controllers, to ensure that they work together | Specced tests written by developers run using Karma/ngMock | All tests pass. | Tests for certain controllers are failing - bug fixing in progress. | F | Known bug, work on it is in progress, and will be finished by the final demo date. Tests influenced by this bug are skipped for now |

**Functional Testing Strategy**

We will be performing unit, integration, system, and manual testing on our app at various intervals. For unit testing, individually tested each function we created or edited before adding it to the master branch. For integration testing, we ran integration tests every time someone finished a section of code and wanted to add it into the master branch. Additionally, we ran integration tests once a week on a schedule to prevent bugs from propagating. System testing happened multiple times a week. Manual testing happened only in the last several sprints, to smooth out any bugs that exist in the UI, not the overall functionality.

To ease the testing process, all tests were written using Jasmine and ngMock, and were run automatically using Karma and the PhantomJS browser. Tests that are not supported using these plugins (high level system tests) were done manually.

Our team used the bug tracking software, Jira, to keep track of tasks to be completed, bugs that had been found, and suggestions for implementation. The categories of bugs were as follows: High, Medium, Low, and Critical. High priority was for bugs that could stop a user from using the rest of the app, medium priority was for bugs that affect the UI or functionality, but don’t stop the user from using the rest of the app. Low priority was for bugs that are fairly trivial in nature (such as misaligned text, wrong colour font, etc). Critical priority was for bugs that caused crashes and/or locked the user out of the app. The statuses for our bug priorities were To Do, In Progress, and Done. To Do was for bugs that we haven’t begun to fix, In Progress was for when a developer is working on a bug so multiple developers don’t try to fix the same bug, and Done was for when the bug has been tested and verified as fixed.

**Adequacy Criterion**

Our adequacy criteria are as follows:

* Make sure that every module has been tested on its own, so we know that each individual module does its job correctly so it doesn’t cause problems in the larger piece of software
* Make sure the existing tests provide adequate coverage, so we don’t end up with edge case bugs and crashes
* Make sure that each major component (ie database, displaying weather data, data parsing) of the app is extensively blackbox tested, so each one interacts with prompts and returns the outputs we expect
* Make sure UI is sufficient first with automated UI testing, then with as many third party users as possible
* Make sure that the user interface has been tested by third party, so we can find UI bugs that we didn’t catch in automated testing that a real user might find
* Make sure that the final application has been tested on multiple network qualities, because the average user might be in a poor network area, be offline or have inconsistent network connection and we want to ensure the apps behaviour is consistent