Crashcode

Andrew Feng1, Ashwin Soorkea2, Jeffrey Buonamici3, Mikael Samvelian4, Mounir Maarabani5, Neha Pal6,

Rezhnd Raveendrakumar7, Ruthvik Prasad Shandilya8, The Phi Nguyen9

1stu-id, 2 stu-id, 3stu-id, 4stu-id, 5stu-id, 640070634, 740028067, 840049141, 940015643

***Abstract -* In this paper we followed the hybrid approach of Scrum and Extreme Programming to better understand the process of software development. We realized the importance of pair programming as well as Refactoring. Beginning from the first step of software requirements specification to the last step of Evolution, customer plays equally important role as the development team.**

# INTRODUCTION[[1]](#footnote-1)

I

n this project, the main objective was to learn about the process of building a software system in a large team. We were exposed to many phases involved in this process such as specification, designing, development and testing. The overall objective was to build a classified site similar to “Kijiji” where user can buy or sell goods/services. Some fundamental features that the site was intended to include are posting an ad, creating a personal account, contacting the seller, etc.

We followed the Scrum Agile method and adopted certain Extreme Programming (XP) practices such as user stories, refactoring, pair programming. Daily scrum meetings were organized. These meetings were documented in the wiki page on GitHub. In these meetings, we were discussing progress and delegated tasks to be done for each sprint. We communicated using Slack and Discords when we weren’t able to have a face to face meetings. Each Sprint followed a fixed length of 2 weeks. Within each sprint the TA’s (who served as the stakeholders) gave a set of deliverables to complete. Our development team needed to fulfill their wishes as well add features to the website to make it more user friendly. Further information about “CrashCode” architecture and design will be discussed throughout the article.

# Architecture and Design

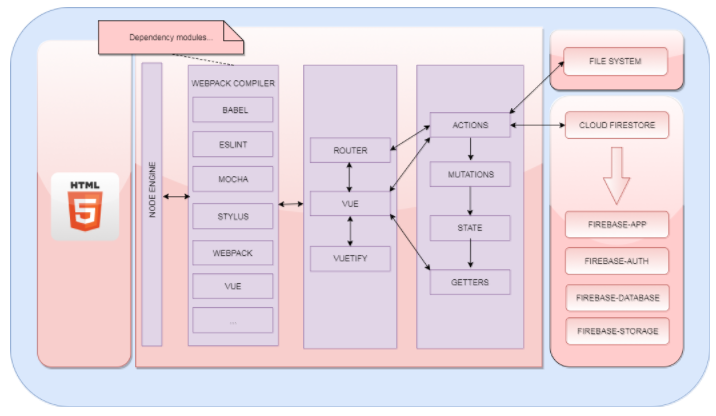
Our web based application follows the layered style client server architecture. The diagram below provides an overview of the client server model by using Node JS.

## node2

The client sends HTTP requests to the Node JS server, which processes the request and replies with a HTTP response to the client.

## Conceptual Architecture of our system.

The block diagram below shows the conceptual architecture of our application.



As can be seen from the diagram, our application uses 4 modules of the Firebase Cloud.

Firebase-app: core components to initialize firebase.

Firebase-auth: authentication feature linked to firebase.

Firebase-database: data model of the application.

Firebase-storage: used to store blob model.

The main modules of our application are:

Vue – modules for the layout templates.

Vuetify – modules representing the material components.

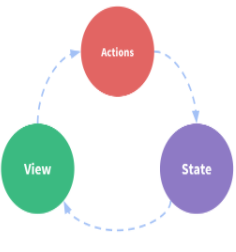
Router – modules for controlling the routing of pages.

VuEx container – controlling the actions, mutations, state and getters modules.

## Application Architecture – Flux pattern

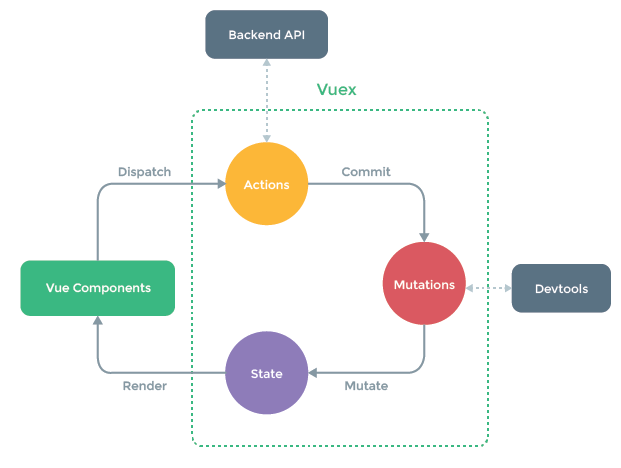
Flux is a design pattern just like MVC which controls data flow of an application.

Our application uses VuEx, a state management pattern, to maintain a centralized store for all the components in the application. It implements a subset of the Flux pattern and is based on the concept of one-way data flow.



Above diagram represents the one-way data flow.

Following diagram represents the model of VuEx tailored specifically for Vue.js



# Technologies Used and Implementation

## Technologies used

For developing our software our team members (development team) decide to use following technologies:

|  |  |
| --- | --- |
| **Front End** | **Vue Js, npm, Vuetify** |
| **Continuous Integration** | **Travis ci** |
| **Back-End** | **Firebase** |

## **Vue Js-** Vue.js is a popular JavaScript front-end framework that was built to organize and simplify web development. It is an open-source progressive JavaScript framework for building user interfaces. Integration into projects that use other JavaScript libraries is made easy with Vue because it is designed to be incrementally adoptable. Vue can also function as a web application framework capable of powering advanced single-page applications[1].

* **Npm-** npm is a [package manager](https://en.wikipedia.org/wiki/Package_manager) for the [JavaScript](https://en.wikipedia.org/wiki/JavaScript) programming language. It is the default package manager for the JavaScript runtime environment [Node.js](https://en.wikipedia.org/wiki/Node.js). It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry [2].
* **Vuetify-** Vuetify is a semantic component framework for Vue. It aims to provide clean, semantic and reusable components that make building your application a breeze. Vuetify supports all modern browsers. From mobile to laptop to desktop, you can rest assured that your application will work as expected [3].
* **Travis ci**- Travis CI  is a hosted, distributed [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration) service used to build and test software projects hosted at [GitHub](https://en.wikipedia.org/wiki/GitHub" \o "GitHub) Travis CI is configured by adding a file named .travis.yml, which is a [YAML](https://en.wikipedia.org/wiki/YAML) format text file, to the root directory of the repository. This file specifies the programming language used, the desired building and testing environment, and various other parameters[4].
* **Firebase -** Firebase provides a realtime database and backend as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud[5].

## Implementation

# Process

Our application process work as follows:

1. Any feature or issue to the application is registered as an issue in GitHub.
2. Once the issue is created, labels illustrating the Priority, Points, Type, Risk is assigned to it. An acceptance test wiki page is created to validate the change later on.
3. Members working on the issue is assigned to it.
4. Comments are added to the issue on Github to show progress or any problems related to it.
5. Any team discussion related to the issue is opened on Slack if required.

Our development process work as follows:

1. For the development process we follow a git-flow like pattern for GIT for code development and monitoring.

This means that we have 2 main branches (“master” for the production code and “develop” for the integration phase)

1. Once an issue is assigned to a member, the developer will create a branch and work on the issue till completion. Once completed, it is integrated in the “dev” branch by sending a pull request.
2. There will be a code review phase with feedbacks before the merge is accepted.
3. Once accepted, the merge is done in the “develop” branch.
4. Once the system testing is performed and validated, the code in the “develop” is integrated in the “master” for a production release.

Our integration process work as follows:

1. Once the code is developed and unit test created for an issue, the code is committed to the branch.
2. When a pull request is sent, our CI tool will run all the unit and component tests to validate for any regressions. It is only if all the unit tests have passed that the merge request will be accepted.
3. Once our CI tool have run and passes, a report of the unit tests covered together with a code coverage report will be generated. The report will give an indication of any missing tests based on the size of the code.

# Testing

The testing phase during our development cycle involves mainly:

1. Unit testing
2. Component testing
3. System testing

For the unit testing and component testing, an automated unit testing tool have been integrated to our application namely Mocha and Chai.

## Mocha and Chai unit test framework

Mocha is a testing framework for JavaScript and is very popular in the JavaScript community. It is easy to plug and also the tests can be setup easily. Chai is an assertion library which can be used with Mocha.

## Structure of Application Test

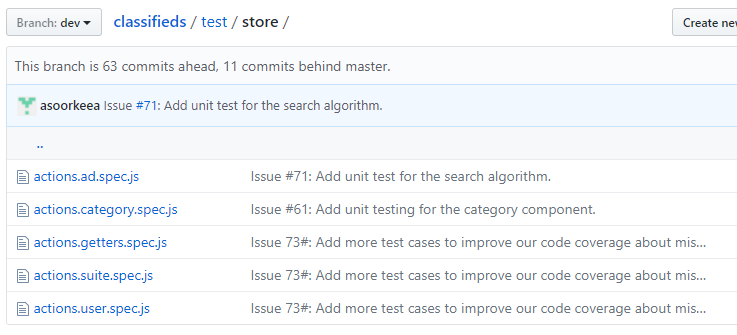
Based on our VuEx pattern, unit tests have been applied only to classes that contains our business logic implementation namely the \*.action classes.

For each \*.action file, a corresponding unit test class have been created to test the different operations available in the file.

For example, the following components are tested:

1. User component class
2. Ad component class
3. Category component class

Example:



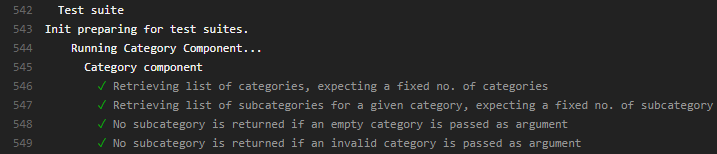
## Structure of Mocha Unit Test

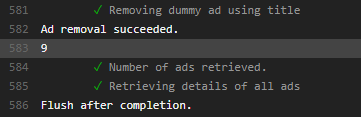
Each unit test is composed of 3 parts namely:

1. Setup: where the system is initialized with the testcase namely input and expected output.
2. Call: where the operation or method to be tested is called.
3. Assert: where the result of the operation is compared with the expected output.

Once the unit tests are created, it can be executed by using the “npm run tests” command which an alias of the actual command defined in the “package.json” file.

Example of an extract: where we can see the “before”, “test execution”, “after” and results of execution.

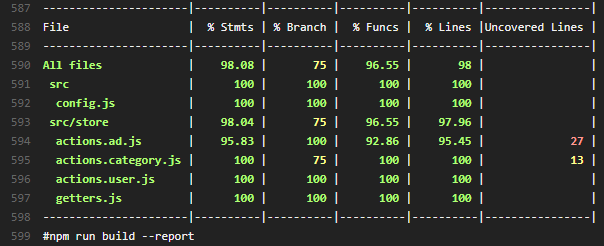




## Component Unit Test Coverage Report

Once the tests have been executed, a code coverage report is produced. The code coverage report is generated by the Istanbul plugins.

Example of the test coverage report:



The report produced provides a percentage of any uncovered tests and also shows which part of the code have not been tested.

## Challenges

Most of the JavaScript functions and API uses the Promises style and Asynchronous type. So the unit tests have to be tailored to handle these type of calls.

Also all our tests is performed on our live database. Dummy data has to be setup for the test cases and these data have to be destroyed at the end of the tests.

Our database is secured and only registered user is allow to perform update transaction. Unit tests have to be tailored for a “before execution” and “after execution” to handle the security issue.

## System Testing

The system testing is performed manually by using the Acceptance Tests defined in our Wiki Page. For each user story, one or more acceptance have been defined and for each release, the acceptance tests is covered and the results of the test displayed on the Wiki Page.

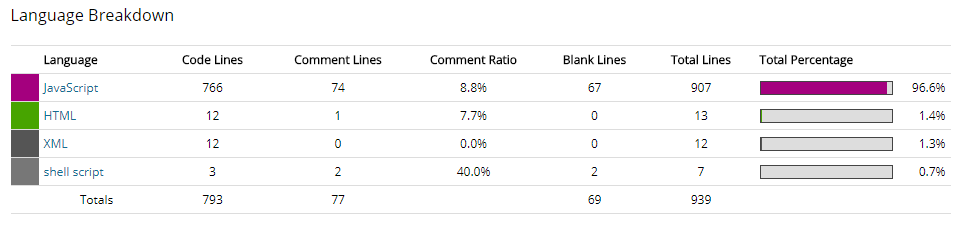
This step is currently being performed manually but an automatic tool such as the Selenium plugin can automate this process also.

# Metrics and explanations

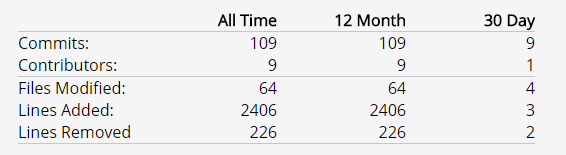
The following metrics have been derived based from the master repo in GitHub.

The table below shows the code metrics of our application, with total lines of code for each language with information regarding the comment and blank lines.

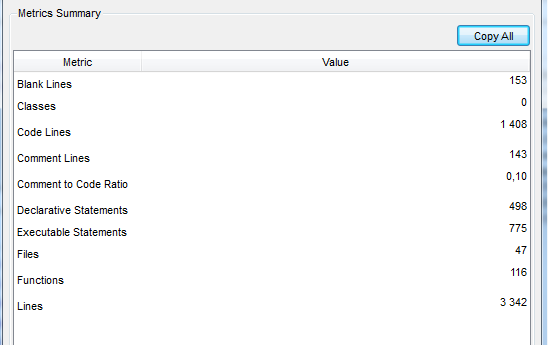
## Application Breakdown – metric code



The table below shows process metrics about no. of changes in files in a 30 day and 12 months period.

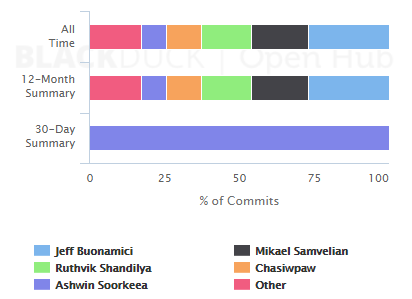


## Metric Summary – understand tool



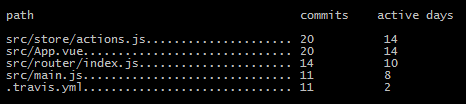
## Team Contribution

The table below shows the contribution of our team.



## File Breakdown – Process code

The table below shows the top 5 files that contains the most changes/commit and the number of days the files are active.



The metrics are not final and has been extracted on 2017-11-26.

## Summary

In a nutshell, here’s an overview of the results of the findings.

We have had 109 commits made by 9 contributors representing 793 lines of code. Our application is written mostly in JavaScript and a low number of source code comments based on the % of the findings. We have a code base with a short history with the commit done in September 2017 and the application is maintained by a large development team of 9 contributors.

The most active file is the “actions.js” which contains all the business logic of the application. It is the file that performs data manipulation between the application and firebase.

The metric code shown above has been derived by two different systems, the scitools “understand” which provides a summary of our code base and the “openhub” which reads the Github repository and produces metric information. Both tools show different results in terms of lines of code because the understand tool has been customized to read specific files whereas open hub applies its algorithm to the code base files only, by not taking into consideration test files, generated files and node modules plugins.

The process metric has been derived using git commands and using the extra set of commands.

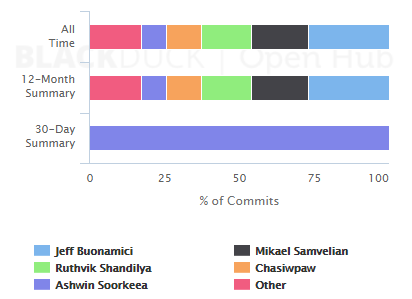
# lessons learned

Learning was the motive of this project and we learned many things throughout the process of this project, which includes:

1. GitHub might be a suitable platform for repository management and collaboration. However, an alternate tool such as Slack is required for instant communication. The decisions taken and conversations on Slack are not registered on Github. It has to be done manually.
2. Maintaining a constant velocity for sprints is difficult when the team is distributed and all contributors are not 100% percent on the project (work, not available, sick, etc. ).
3. 3. Most of the acceptance tests were performed by the development team instead of the TA. Any issues raised by our TA were through verbal communication and through the sprint report grade. The development team then has to create these issues on GitHub. To be efficient, these issues should have been raised through GitHub directly by our TA.

# team’s contribution

The table below shows the contribution of our team.



Team Lead / Scrum Master

Technical Expert (UI / Firebase)

Contributors/Developers

Process Integrators

|  |  |
| --- | --- |
| **User stories** | **Contributors** |
| **1.** |  |
| **2.** |  |
| **3.** |  |
| **4.** |  |
| **5.** |  |

# conclusion

conclusion of project here

Acknowledgment

We would like to thank Dr. Emad Shihab and Giancarlo Sierra for their continuous guidance and feedback in completion of this project.

References

1. <https://en.wikipedia.org/wiki/Vue.js>
2. https://en.wikipedia.org/wiki/Npm\_(software)
3. <https://github.com/vuetifyjs/vuetify>
4. https://en.wikipedia.org/wiki/Travis\_CI
5. https://en.wikipedia.org/wiki/Firebase

1. . [↑](#footnote-ref-1)