Report Requirements

**Introduction**

**Architecture and Design**

**Technologies used and Implementation**

**Process**

**Testing**

**Meaningful metrics and explanation (at least one process and 2 code)**

**Lessons learned**

**Teams’s contribution**

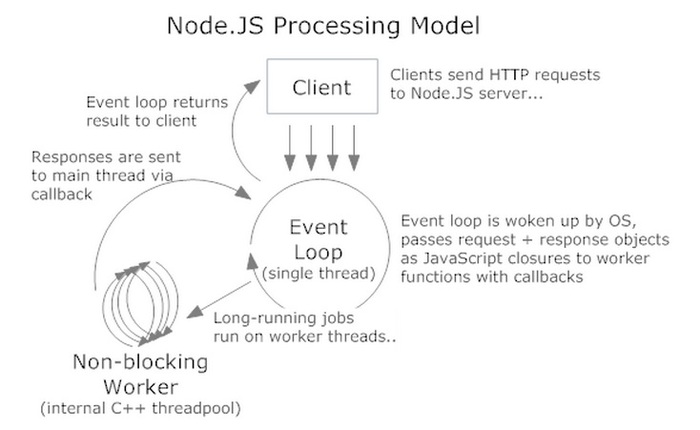
# Introduction

# Architecture and Design

## Layered Style Client Server

Our web based application follows the layered style client server architecture.

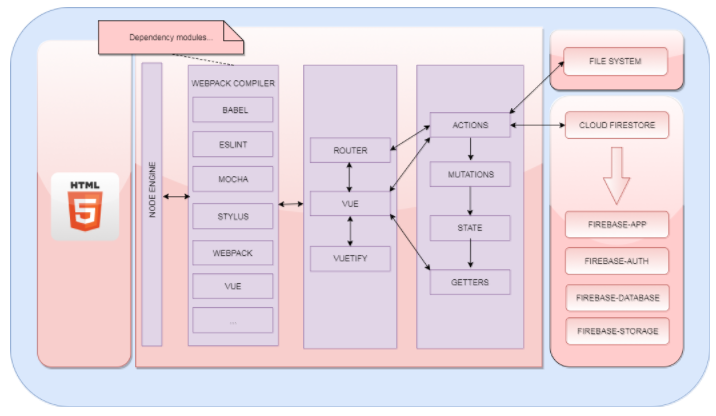
The diagram below provides an overview of the client server model by using Node JS.



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.

1. Firebase-app: core components to initialize firebase.
2. Firebase-auth: authentication feature linked to firebase.
3. Firebase-database: data model of the application.
4. Firebase-storage: used to store blob model.

The main modules of our application are:

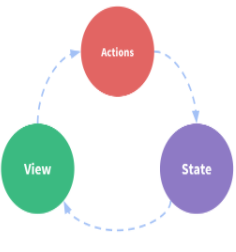
1. Vue – modules for the layout templates.
2. Vuetify – modules representing the material components.
3. Router – modules for controlling the routing of pages.
4. VuEx container – controlling the actions, mutations, state and getters modules.

## Application Architecture – Flux pattern

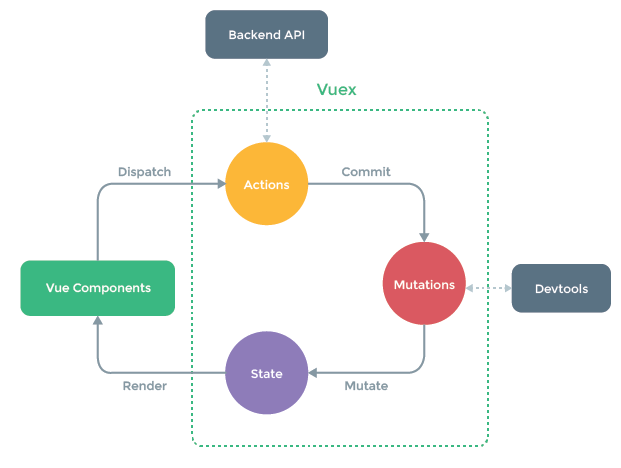
Flux is a design pattern just like MVC which controls data flow of an application.

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.

The one-way data flow is represented as the following diagram:



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



# Technologies used and Implementation

## Implementation Process

The following tools have been used for the Implementation Process namely:

* GIT for the version management.
* WEBPACK as a build system for managing our modules for the JavaScript application.
* NPM for installation and managing module dependencies.
* GITHUB for repository sharing and other features like collaboration, issue tracking… etc.
* FIREBASE for data storage.
* TRAVIS CI for continuous builds and integration testing.

Our implementation process follows the key concepts such as understandability, reusability and efficiency.

## Understandability

The following lists the principles of coding that are used in our application.

1. **Variable Naming Style**: The camelcase style has been used whenever a variable needed to be declared, that is the first letter of the word is always in lower case and then any additional words would be chained to the previous word with a capital letter.

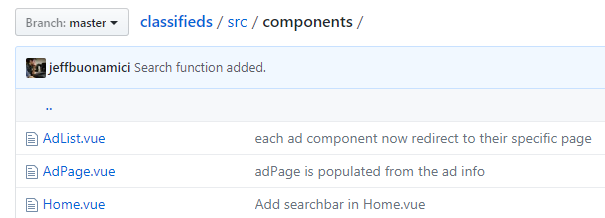
Example: var adPage = { }

1. **Class Naming Style**: Similar to variable naming style, except that the first letter of the word is uppercase. Since Vue JS is being used as the front-end framework, the extensions of the files will be *.vue*

Example: UserProfile.vue

1. **Folder structure:** The REACT way of grouping pages as a component is being used. Below is an example of the folder structure of our application.

Example:



1. **ESLint Coding Rules**: ESLint contains a list of rules which helps structure our code for ease of maintenance and evolution.

ESLint contains rules that prevent syntax or logic errors, variable declaration or even ways of coding to avoid problems.

Our application uses the ESLint plugin with checks turned on so that we enforce these rules for every commit which help maintains consistency and integrity of our application code.

1. **ES6 features**: Our application uses the ES6 features which makes our coding consistent.

## Reusability

By using the VuEx state management pattern, it allows us to increase code reuse since our modules are separated and it allows us to work on separate items concurrently.

## Efficiency

By using the ESLint plugins and with the check turned on, it allows use to safe code and complex algorithm will be detected by the plugin with a warning during the compilation phase.

# Process [Draft]

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

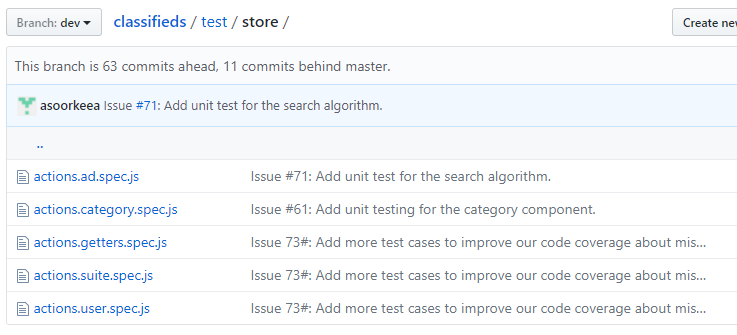
Based on our VuEx pattern, unit tests have been applied only to classes that contains our business logic implementation namely the \*.action classes.

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:



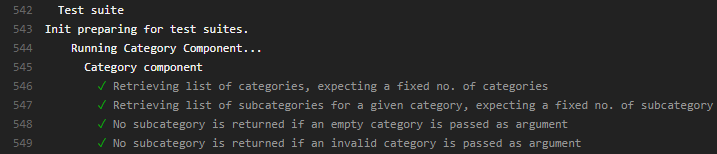
## Structure of Mocha Unit Test

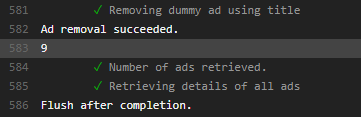
Each unit test is composed of 3 parts namely:

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.

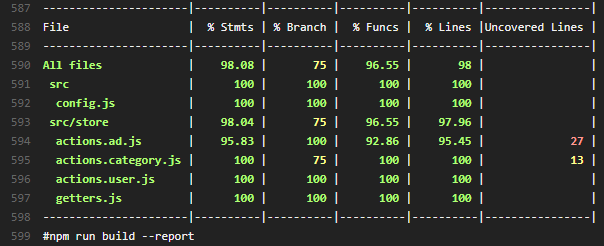




## Component Unit Test Coverage Report

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.

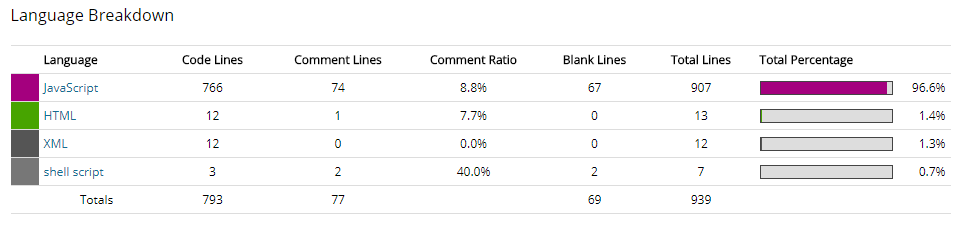
This step is currently being performed manually but an automatic tool such as the Selenium plugin can automate this process also.

# Meaningful metrics and explanation

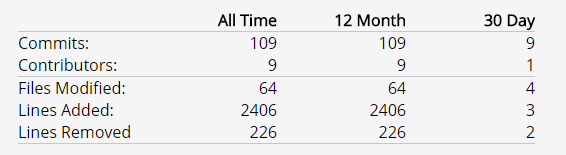
The following metrics have been derived based from the master repo in GitHub.

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

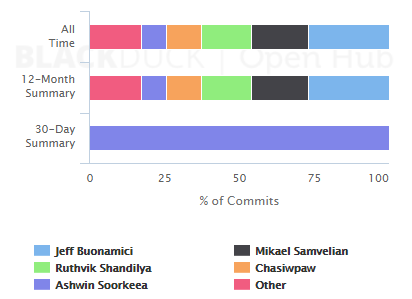


The table below shows process metrics about no. of changes in files in a 30 day and 12 months period.



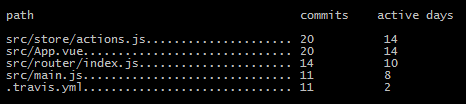
## Team Contribution

The table below shows the contribution of our team.



## File Breakdown

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.

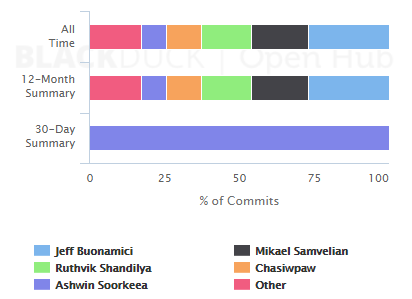
We have had 109 commits made by 9 contributors representing 793 lines of code. Our application is written mostly in Javascript and a low number of source code comments based on the % of the findings. We have a code base with a short history with the commit done in September 2017 and the application is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

# Lessons learned

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. 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 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. |  |
|  |  |

## References

Software metrics:

1. <https://github.com/tj/git-extras>
2. <https://www.openhub.net/p/classifieds341>