Classifieds

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

Rezhnd Raveendrakumar7, Ruthvik Prasad Shandilya8, The Phi Nguyen9

127546807, 2 40012400, 327415834, 440003178, 540005151, 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.**

*Keywords: Agile, Extreme Programming, Refactoring, Scrum.*

# 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.

Extreme Programming (XP) is the most widely used agile methodology. XP shares the values espoused by the Agile Manifesto for Software Development but goes further to specify a simple set of practices[1]. In this project, 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 “Classifieds” architecture and design will be discussed throughout the article.

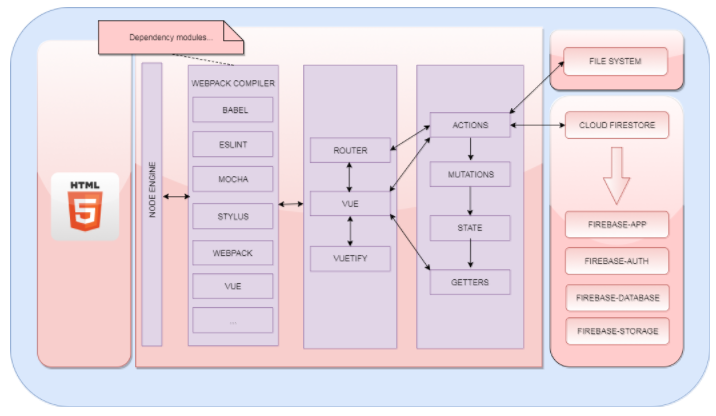
# Architecture and Design

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.

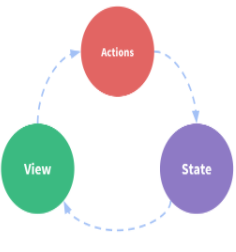
**Router** – modules for controlling the routing of pages.

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

## Application Architecture – Flux pattern

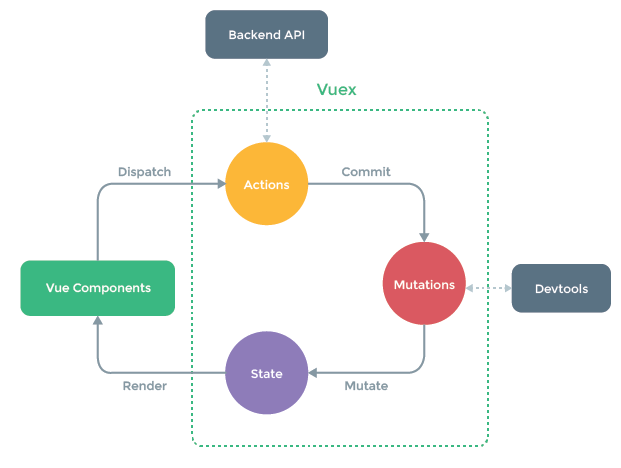
Flux is a design pattern just like MVC which controls data flow of an application.

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[2].

* **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 [3].
* **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 [4].
* **Travis ci**- Travis CI  is a hosted, distributed [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration" \o "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 [5].
* **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[6].

## Implementation

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.

##### **Variable Naming Style**: The camel case 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.

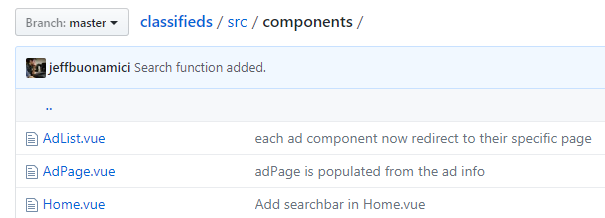
Example: var ad Page = { }

##### **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

##### **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:



##### **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.

##### **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 us to safe code and complex algorithm will be detected by the plugin with a warning during the compilation phase.

# Process

* 1. *Requirement and specification:*

1. Any new features or issues to the application are been analyzed, broken down into a well-defined task and they are registered on GitHub as an issue.
2. Once the issue is created, labels illustrating the Priority, Points, Type, Risk is assigned.
3. All the tasks are being assigned to all members of the team.
   1. *Development:*
4. The first thing after getting all the requirements and analyzing it, is to decide what kind of architecture we want to use for our project (Client-Server architecture) as shown previously.
5. Once the architecture is decided, we must decide on which design we should use for the web application. In our case, we used the Flux pattern which is based on the concept of one-way data flow as shown previously.
6. When the architecture and design are being decided, the next step will be the implementation.
7. We must decide to code in which language for the frond end and back end language, the framework, the database, etc.
8. For the development process we follow a git-flow like pattern for GIT for code development and monitoring.
9. Any team discussion related to the issue is opened on Slack if required.
10. Comments are added to the issue on GitHub to show progress or any problems related to it

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.
   1. *Validation:*
2. Once the code is developed and unit test created for an issue, the code is committed to the branch.
3. 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.
4. 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.
5. There will be a code review phase with feedbacks before the merge is accepted.
6. Once accepted, the merge is done in the “develop” branch.
7. Once all the testing was done, including unit tests, component tests and CI, the software went through an acceptance test.
8. Once the acceptance test is passed, the software from “dev” is merged into the “master”.
   1. *Evolution:*
9. The code was refactored to ensure the understandability.
10. Any additional requirements and specifications were being analyzed and implemented to evolve the software.
11. The software stability is enforced to maintain the customer’s satisfaction.

# 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

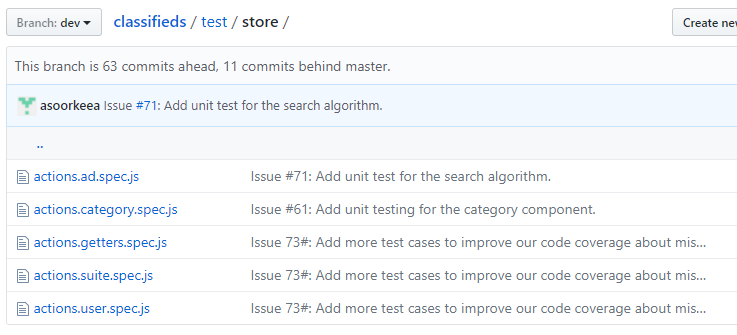
Based on our VuEx pattern, unit tests have been applied only to classes that contains our business logic implementation namely the \*.action classes.

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:



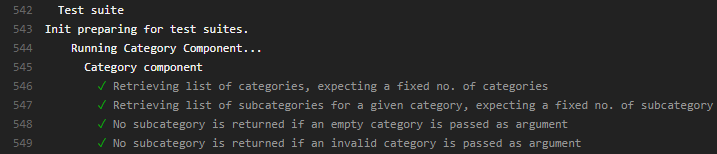
## Structure of Mocha Unit Test

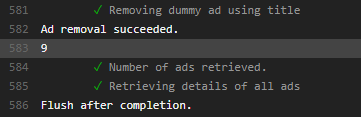
Each unit test is composed of 3 parts namely:

1. Setup: where the system is initialized with the test case 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.

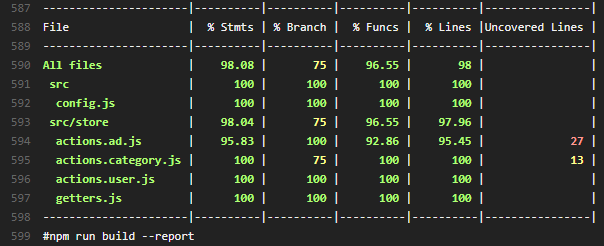




## Component Unit Test Coverage Report

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 types of calls.

Also all our tests are 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.

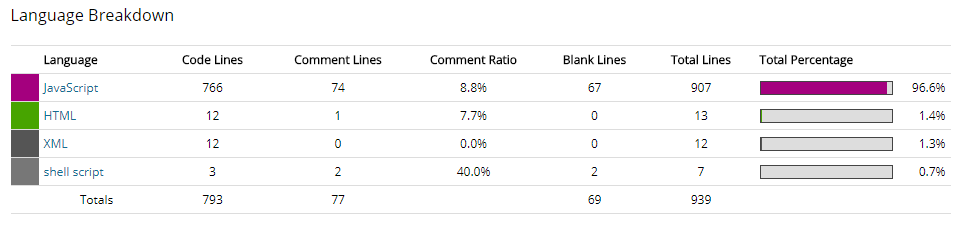
This step is currently being performed manually but an automatic tool such as the Selenium plugin can automate this process also.

# Metrics and explanations

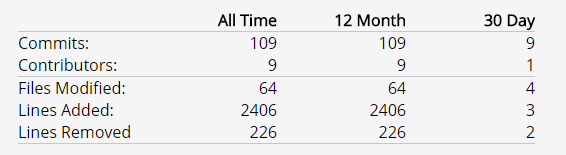
The following metrics have been derived based from the master repo in GitHub.

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.

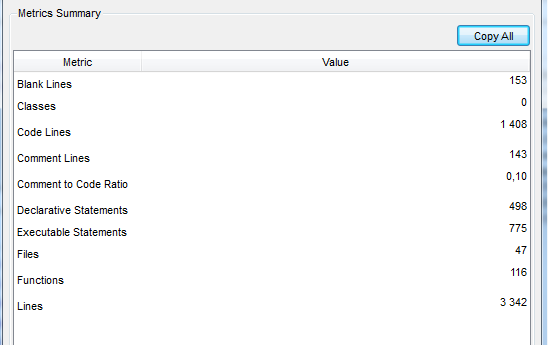
## Application Breakdown – metric code



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

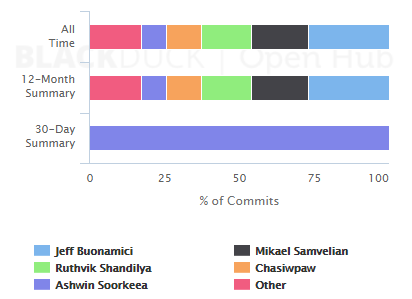


## Metric Summary – understand tool



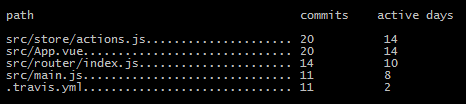
## Team Contribution

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.

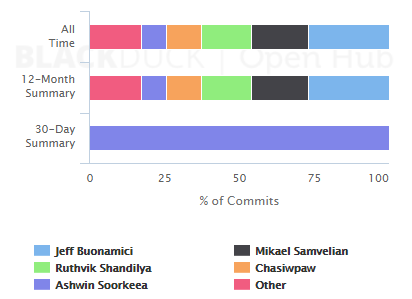
# lessons learned

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.



Following is a list of all user stories given to our development team from the customer.

User Story #1: Registration and Sign In and Password strength

User Story #2: Browsing Categories

User Story #3: Account Information

User Story #4: Ad Posting Options and Scripts

User Story #5: Browsing Categories and Search

User Story #6: Refactoring and Clean up

User Story #7: Documentation and Search by Description

User Story #8: Website Security

User Story #9: Case sensitive Searching

|  |  |  |
| --- | --- | --- |
| Small | Medium | Large |
|  |  |  |

We have represented Contribution of Team Members for each given user story as shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Team Members** | **US#1** | **US#2** | **US#3** | **US#4** | **US#5** | **US#6** | **US#7** | **US#8** | **US#9** |
| **Andrew Fang** |  |  |  |  |  |  |  |  |  |
| **Ashwin Soorkeea** |  |  |  |  |  |  |  |  |  |
| **Jeffrey Buonamici** |  |  |  |  |  |  |  |  |  |
| **Mikael Samvelian** |  |  |  |  |  |  |  |  |  |
| **Moun-ir Maarabani** |  |  |  |  |  |  |  |  |  |
| **Neha Pal** |  |  |  |  |  |  |  |  |  |
| **Rezhnd Raveendrakumar** |  |  |  |  |  |  |  |  |  |
| **Ruthvik Shandilya** |  |  |  |  |  |  |  |  |  |
| **The Phi Nguyen** |  |  |  |  |  |  |  |  |  |

Application Installation and Installation of Software Process Tools.

Application Setup #1: Firebase Database

Application Setup #2: Node JS

Application Setup #3: Application Setup (Webpack, NPM…etc)

Application Setup #4: GitHub

Software Process #5: CI Tools

Software Process #6: Slack Integration

Software Process #7: Code Coverage

Following table shows the contribution of team members for above mentioned processes and tasks:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Contributors | A#1 | A#2 | A#3 | A#4 | S#5 | S#6 | S#7 |
| Andrew Fang |  |  |  |  |  |  |  |
| Ashwin Soorkeea |  |  |  |  |  |  |  |
| Jeffrey Buonamici |  |  |  |  |  |  |  |
| Mikael Samvelian |  |  |  |  |  |  |  |
| Mounir Maarabani |  |  |  |  |  |  |  |
| Neha Pal |  |  |  |  |  |  |  |
| Rezhnd Raveendrakumar |  |  |  |  |  |  |  |
| Ruthvik Shandilya |  |  |  |  |  |  |  |
| The Phi Nguyen |  |  |  |  |  |  |  |

We can now classify contribution of team members members as follows:

#1: Backend Development

#2: Front End UI Development

#3: Unit testing Integration

#4: Feature Documentation

#5: Analysis/Design (Before implementation of a feature)

#6: System Integration

#7: Peer Review

#8: Release Management

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Contributors | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 |
| Andrew Fang |  |  |  |  |  |  |  |  |
| Ashwin Soorkeea |  |  |  |  |  |  |  |  |
| Jeffrey Buonamici |  |  |  |  |  |  |  |  |
| Mikael Samvelian |  |  |  |  |  |  |  |  |
| Mounir Maarabani |  |  |  |  |  |  |  |  |
| Contributors | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 |
| Neha Pal |  |  |  |  |  |  |  |  |
| Rezhnd Raveendrakumar |  |  |  |  |  |  |  |  |
| Ruthvik Shandilya |  |  |  |  |  |  |  |  |
| The Phi Nguyen |  |  |  |  |  |  |  |  |

# conclusion

The goal of this project was to understand and implement the Scrum and Extreme Programming methods of agile software development. After working on this project we can say that it is a good approach to follow the Extreme Programming and agile methodologies for developing a software. User stories help to identify the tasks to be done for each sprint. Working in a team environment has its own pros and cons. A team work is what matters at the end so it is important to maintain a healthy work environment. Scrum meetings help every team member to know the actual status of development as well as to know the efficiencies of other team members. But at the same time it is not easy to setup scrum meetings with such a large team.

Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown [7].

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://www.nku.edu/~sakaguch/msis655/lindstrom2004.pdf
2. <https://en.wikipedia.org/wiki/Vue.js>
3. https://en.wikipedia.org/wiki/Npm\_(software)
4. <https://github.com/vuetifyjs/vuetify>
5. https://en.wikipedia.org/wiki/Travis\_CI
6. https://en.wikipedia.org/wiki/Firebase
7. http://m.adaptivesd.com/articles/cross\_oct02.pdf

1. . [↑](#footnote-ref-1)