
# ASSUMPTION UNIVERSITY

Vincent Mary School of Science and Technology

Department of Computer Science

CS3446 Fundamentals of Cloud Computing - Term Project

AU Open House

By


| | |
|---|---|
| CHATCHAWAN YOOJUIE | 571-5298 |
| CHAWAN VATTANALAP | 573-7444 |
| THAYAWAT THATHONG | 571-7532 |
| ARTISD CHANYAWADEE | 571-2266 |


Semester 1/2018

# Table of Contents

# 1. Introduction

The main objective of this project is to create an application to provide information for high school students who are coming to the open house event.

# 2. Application Features

There are two types of users that will use this application, which are:

1. Student - able to view infos, attend the event and play mini game
2. Authority
    a. Staff - able to manage the events information and mini game
    b. Admin - able to use every function in the application

## 2.1 Student features

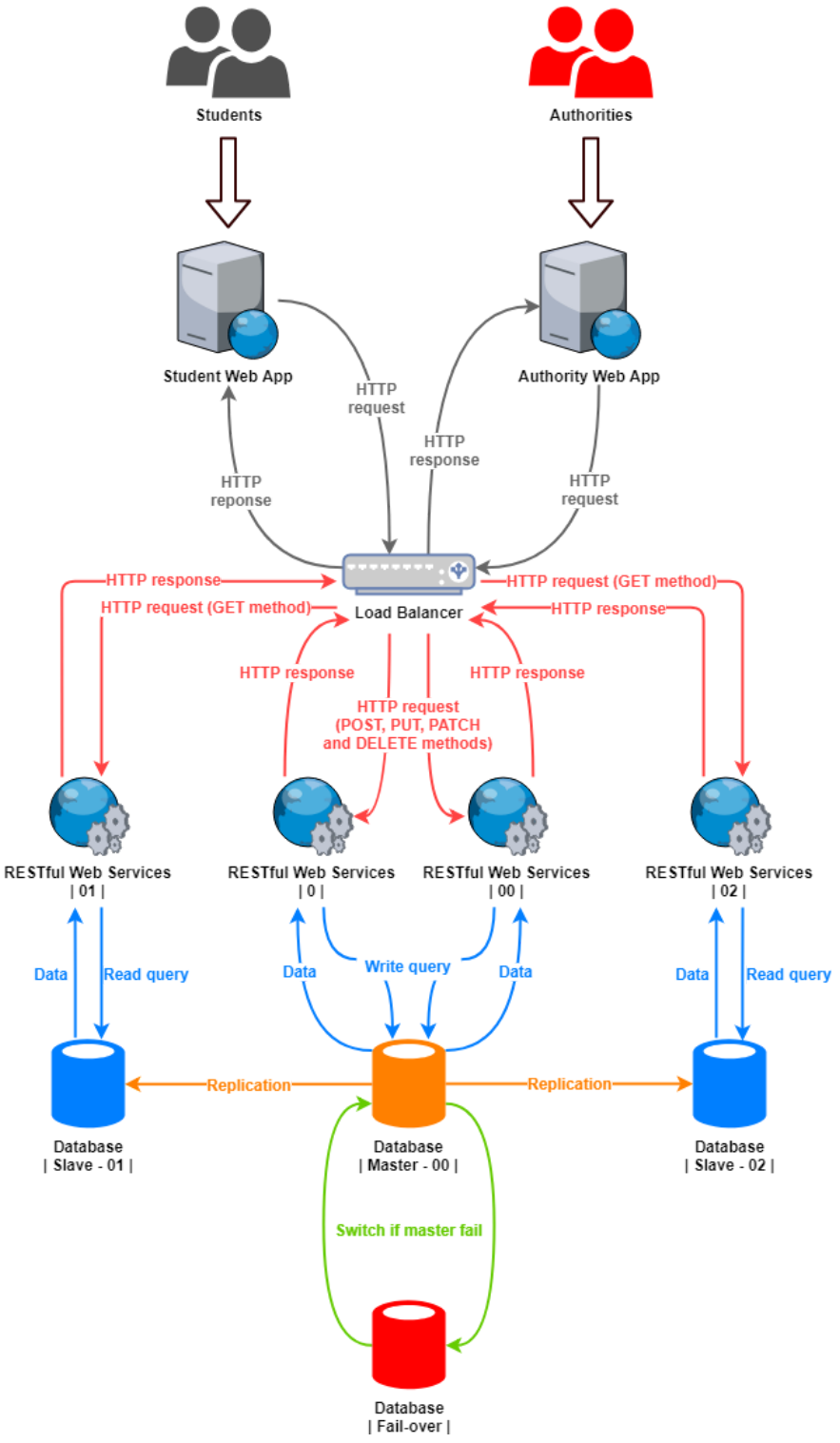
1. Account Registration
    a. Sign in via Facebook or Google
2. View faculty information
    a. Location (use google api)
    b. List of majors
    c. Website
3. Event Registration
    a. Event details
    b. Location (use google api)
    c. Live comment
4. Mini Game
    a. Detail of the question (multiple-choice)

## 2.2 Authority features

1. Account Registration
    a. Sign in with AU's google account
2. Event management
    a. Add/Edit/Remove events
    b. View the list of student that will attend the events
3. Mini Game management
    a. Add/Edit/Remove games
4. Account management (**ADMIN ONLY**)
    a. Approve staff account
    b. Add/Edit/Remove account

# 3. System Architecture

The overall architecture of the system is as follows:



## 3.1 Web Application

There are two web applications for each type of user which are students and authorities. It is a client side application that based on hybrid application framework called Ionic.

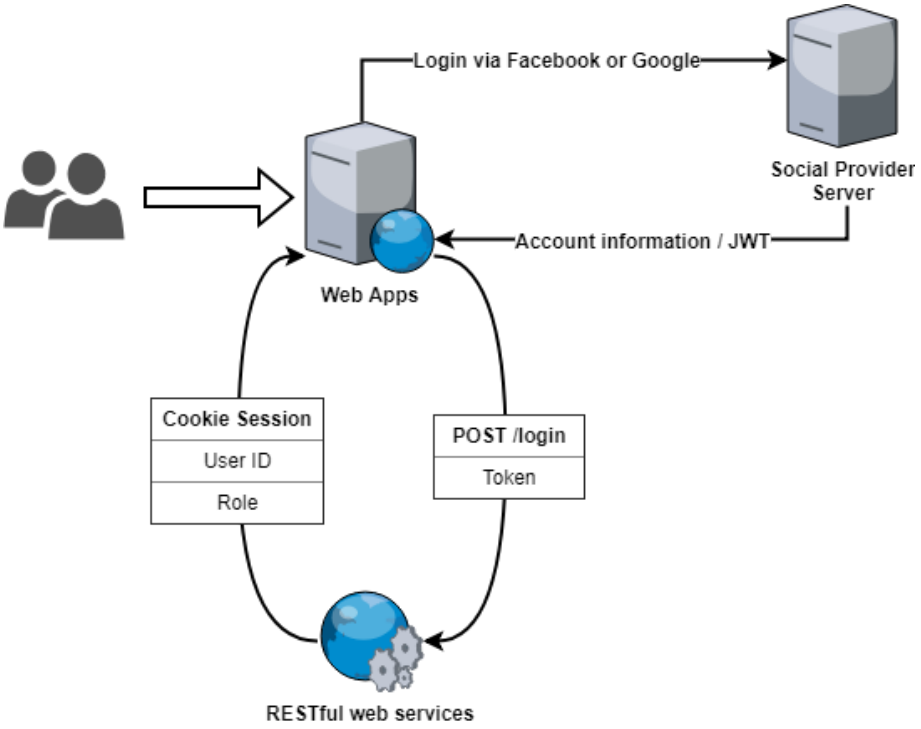
The architecture of the web application is as follow:

| | |
|---|---|
| **Framework** | Ionic: Hybrid application framework based on AngularTS |
| **APIs** | - Google Map: Used to display the location of events<br>- Au Open House: our web services with security access |
| **Authentication** | - Firebase Authentication: Login via Facebook and Google<br>- Firebase Token: Used for logging-in to our backend web services |
| **Hosting** | Firebase Hosting: Hosted on Google server<br>- Student: https://auopenhousestudent.firebaseapp.com<br>- Authority: https://auopenhouseadmin.firebaseapp.com |
| **Source Code** | GitHub: Version control<br>- Student: https://github.com/Cwanyo/AUOpenHouseStudent<br>- Authority: https://github.com/Cwanyo/AUOpenHouseAdmin |

## 3.2 RESTful Web Services

The backend web services that provide all the information according to the user request. It have a security filters to restrict access to certain information or functions based on user roles. The implementation of the authentication system on the web services is based on the Firebase authentication.

Authentication flow is as follows:




First user will have to login on the web application via a social provider using Facebook or Google account. And if the user logged-in successfully, the token will be retrieved from the Firebase authentication service.

The token that used during this process is known as JSON Web Token (JWT). Basically, it is a JSON object that is defined in RFC 7519 as a safe way to represent a set of information between two parties. The token is composed of a header, a payload, and a signature. In the header, it will defined which type of hashing algorithm will be used to create the JWT signature component. For the payload, it will contain the user personal information associated with the user's social account which includes information such as user's email address, name,

profile photo, and most importantly, the user ID. And for the signature component, it is the result from hashing the payload with a asymmetric key (public key). By doing this, It verify the authenticity that the sent token was actually created by an authentic source.

And then, the token will be attached in the body of the HTTP request and sent to web services for logging into our backend web services. Once the web services receive this HTTP request, it will read the request's body and retrieve the token. Then, it will verify the token by performing the same hashing algorithm with a asymmetric key (private key), and finally compare the signature obtained from own hashing operation against the signature from the token.

If the token is verified, the web services will responds with a generated cookie session that contain a user ID and role in the system. With this generated session, the user will be able to access the data without having to login and re-verify the token again. Also with this session, the web services can filter and limit the accessibility depending on the user role. Moreover, the system will use the user ID that retrieved from the token's payload to identify and link the logged-in social account to the data in our database.

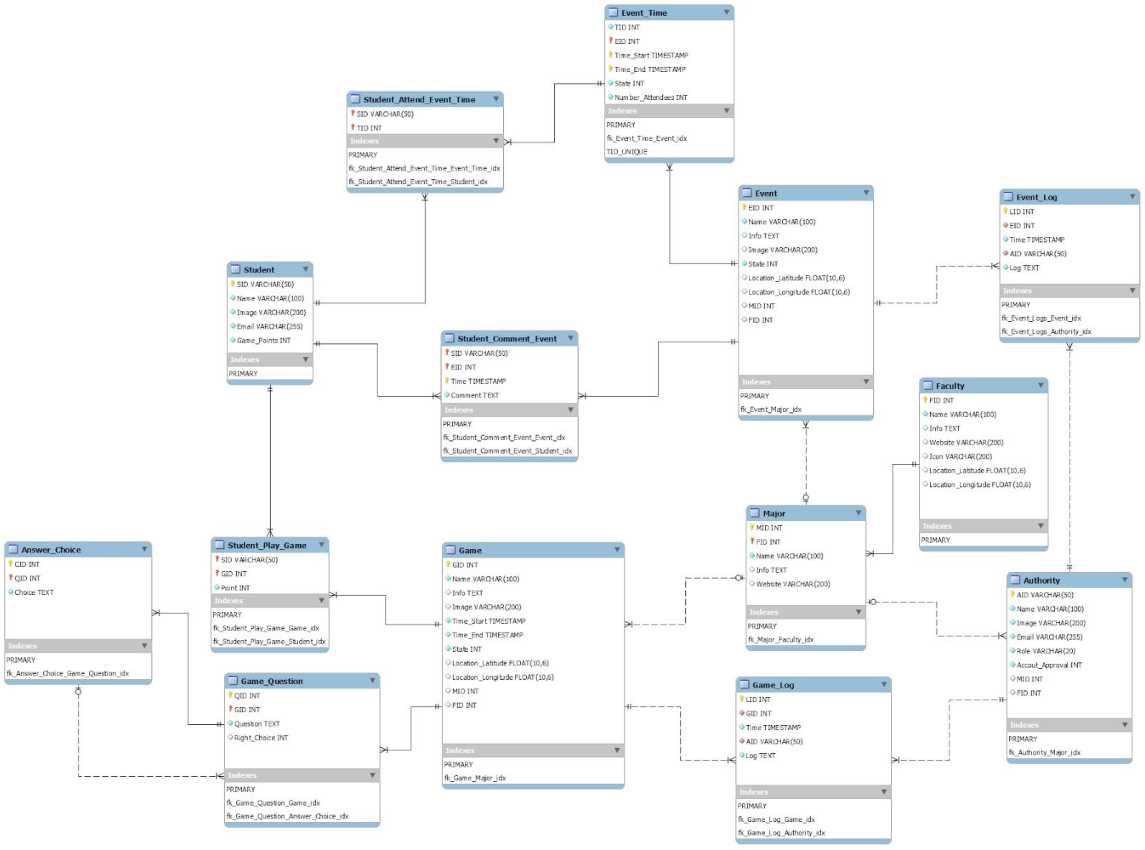
The architecture of the RESTful web services is as follow:

| | |
|---|---|
| **Framework** | Node.js with Express - Web framework based on Javascript |
| **Authentication** | - Firebase Admin: Used for token verification |
| **Hosting** | Heroku Hosting: Hosted on Amazon server<br>- Master 1: https://auopenhouse-0.herokuapp.com<br>- Master 2: https://auopenhouse-00.herokuapp.com<br>- Slave 1: https://auopenhouse-01.herokuapp.com<br>- Slave 2: https://auopenhouse-02.herokuapp.com |
| **Source Code** | GitHub: Version control<br>- AuOpenHouseAPI: https://github.com/Cwanyo/AuOpenHouseAPI |

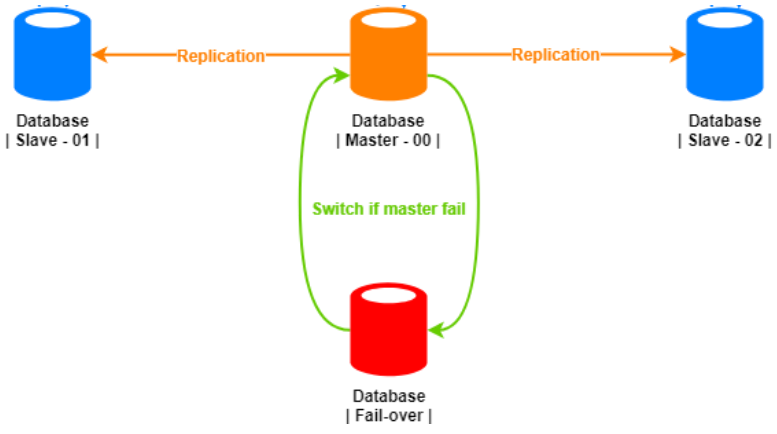
## 3.3 Database

The relational database that contain all the information of the AU open house system.

AU open house - ERR diagram is as follows:




In order to handle the workload and fail tolerant, we will use master-slave replication with a extra failover node.



The data from the master node will be replicated to the slave node automatically. Note that all write access can only be done on master, not on the slave nodes. Hence, all the write access will be handled by the master node and all the read access will be handled by the slave nodes. By doing, we can split the workload among multiple databases. Moreover, there will be additional backup node in case of failure in the master node.

| | |
|---|---|
| **Master - 00** | Handle write access |
| **Slave - 01** | Handle read access |
| **Slave - 02** | Handle read access |
| **Fail-over** | Automatic switch if master fail |

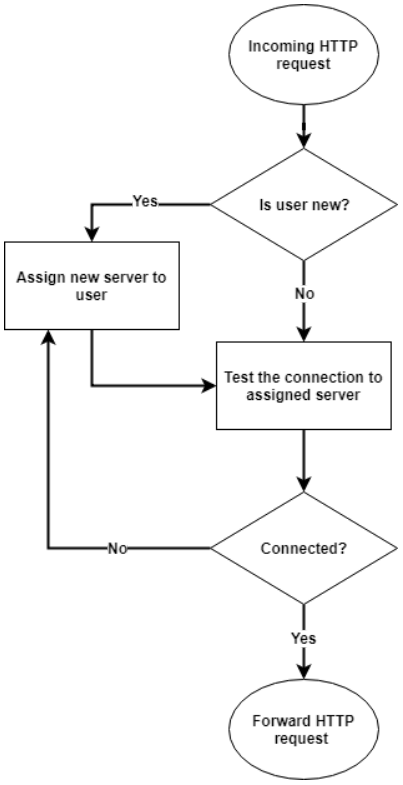
The architecture of the database is as follow:

| | |
|---|---|
| **DBMS** | MySQL |
| **Hosting** | Cloud SQL: Hosted on Google server |

## 3.4 Load Balancer

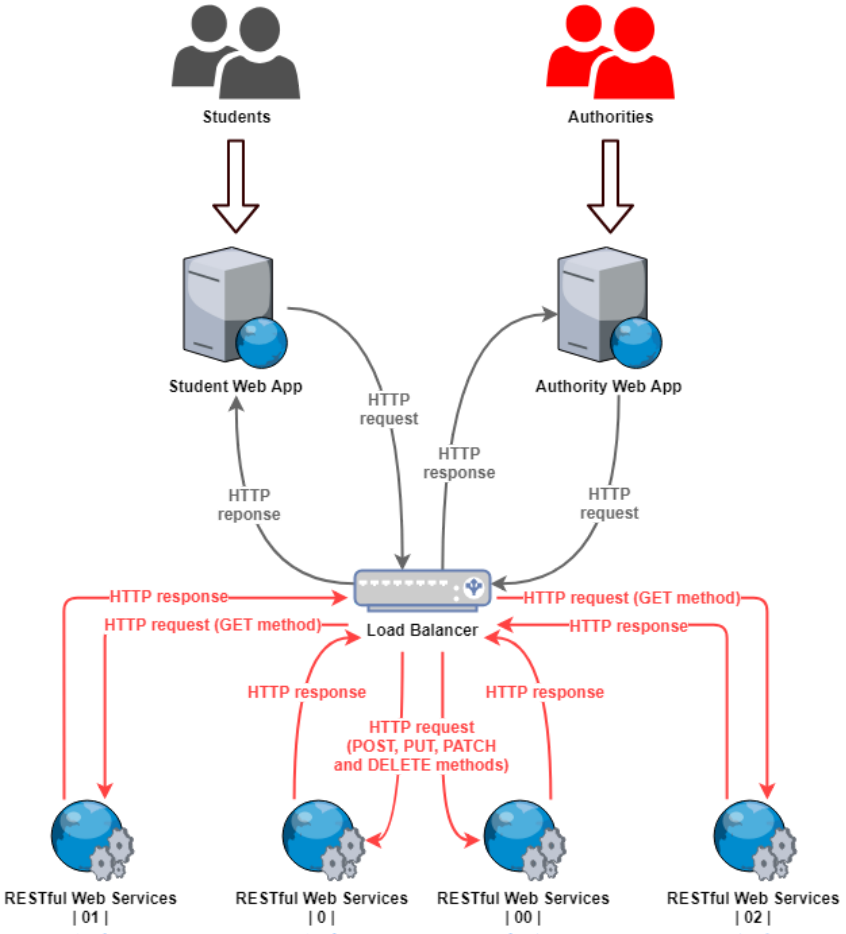
Load Balancing is created as a middle server that manage and distribute workloads among multiple RESTful web services servers.

The implementation of the load balancer is as follows:




The Load Balancer will check whether HTTP request is coming from a new user or not. If it is a new user, the server will be assigned to that particular user. With the assigned server, the load balancer will test the connection to check whether that assigned server is online and functional. And, if the test connection return the server status code lesser than 500, It will forward the HTTP request to that assigned server. On the other hand, if the test connection return the server status code equal or more then 500, it will re-assign a new server to user and test the connection once again.

Any HTTP request method that requires that database write access will be forced and forwarded to the RESTful web services - 0 and 00, which are connected with the master database. And for the HTTP request method that require only read access will be forwarded to the assigned slave server.




The architecture of the load balancer is as follow:

| | |
|---|---|
| **Framework** | Node.js with Express - Web framework based on Javascript |
| **Hosting** | Heroku Hosting: Hosted on Amazon server<br>- Load Balancer: https://auopenhouse-loadbalancer.herokuapp.com |
| **Source Code** | GitHub: Version control<br>- AuOpenHouseAPILoadBalancer:<br>https://github.com/Cwanyo/AuOpenHouseAPILoadBalancer |