College Event Website

University Of Central Florida

gROUP rEPORT – COP4710 SPRING 2017

gROUP 37: jOSHUA thORSON & sTEVEN pARRA

2017

Table of Contents

[Project Description 1](#_Toc480208447)

[GUI 1](#_Toc480208448)

[ER-Model 1](#_Toc480208449)

[Relational Data Model 1](#_Toc480208450)

[SQL: 1](#_Toc480208451)

[SQL Statements and Code 1](#_Toc480208452)

[Conclusion 1](#_Toc480208453)

[Database Performance 1](#_Toc480208454)

[Desired Features 1](#_Toc480208455)

[Problems Encountered 1](#_Toc480208456)

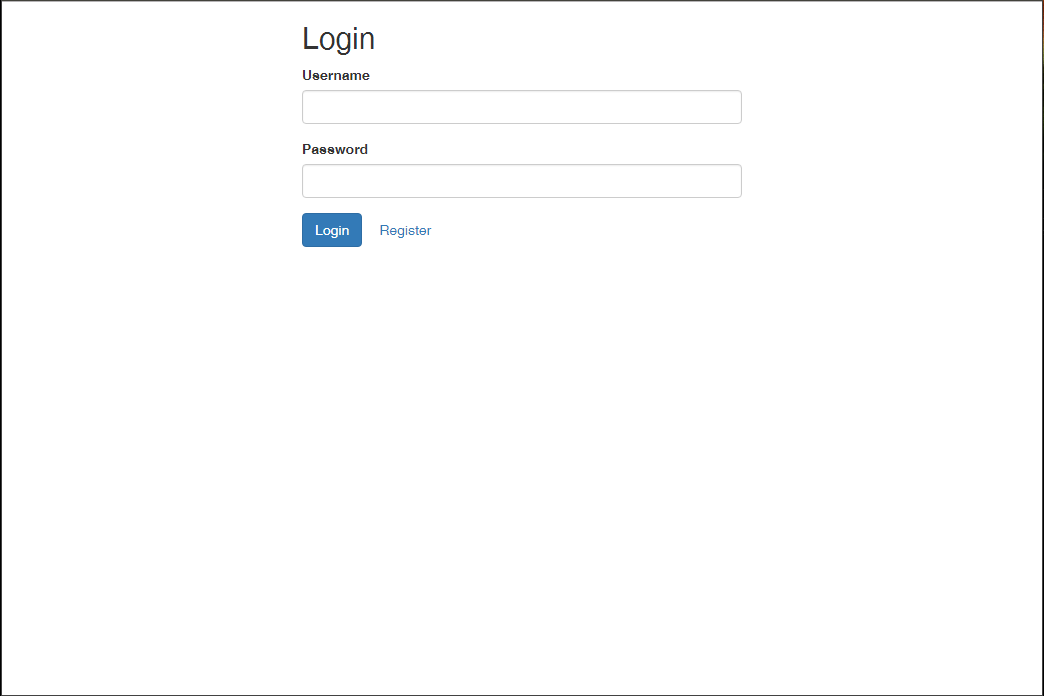
[Summary 1](#_Toc480208457)

# Project Description

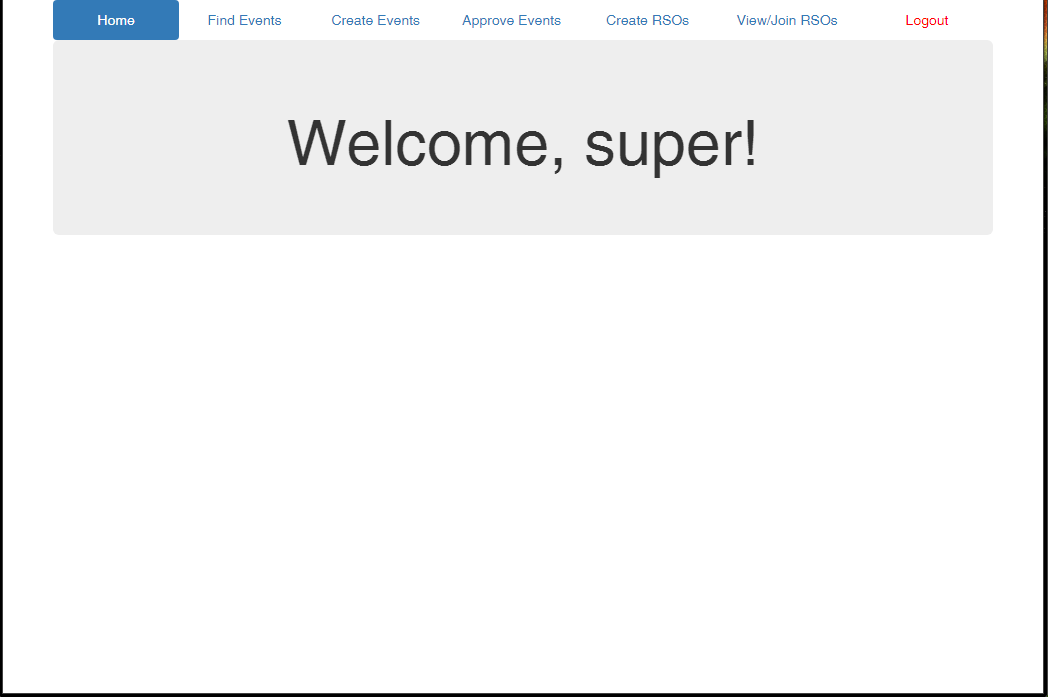
We were tasked with creating an Event web application that allows students to create events, view events, and join clubs. There are three different user levels to this web application. The ‘User’, who is able to login to the web application and view public events. The ‘Student’ who is able to join an RSO and view private events for their university. The ‘Admin’, who is able to create events private, public, or RSO. The ‘Super Admin’ who is able to create universities for users to create RSOs for. Events have certain details such as description, location, time, comments, and rating for each commenter. Overall, this project is aimed centralizing a place to view, create, and manage events and to help students of different universities find events that they would like to attend.

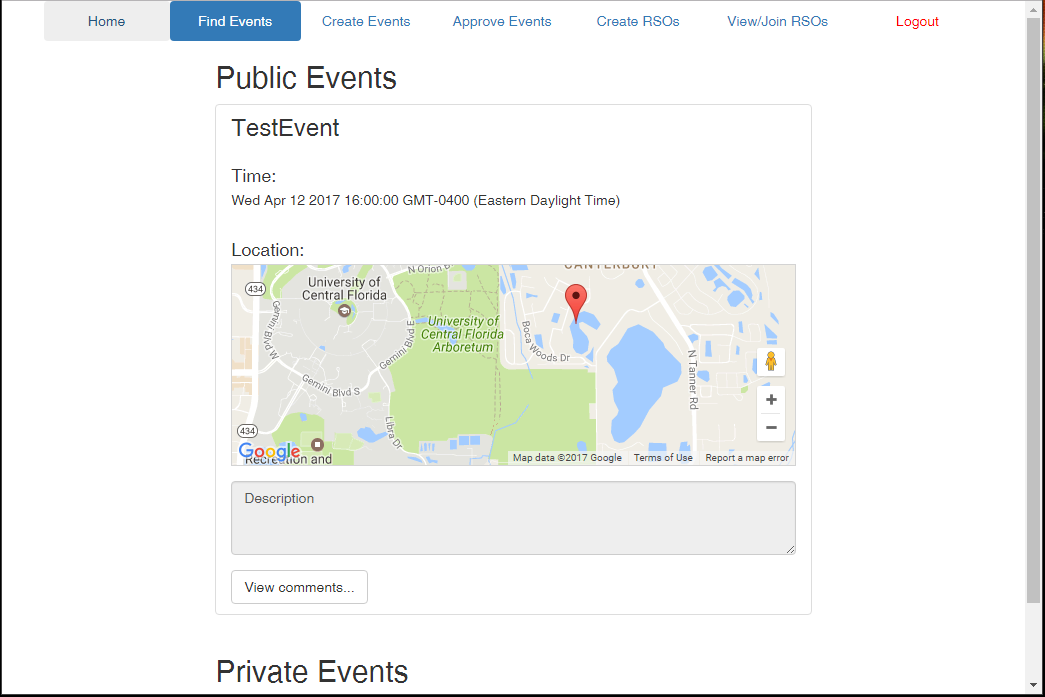
# GUI

Login Page:

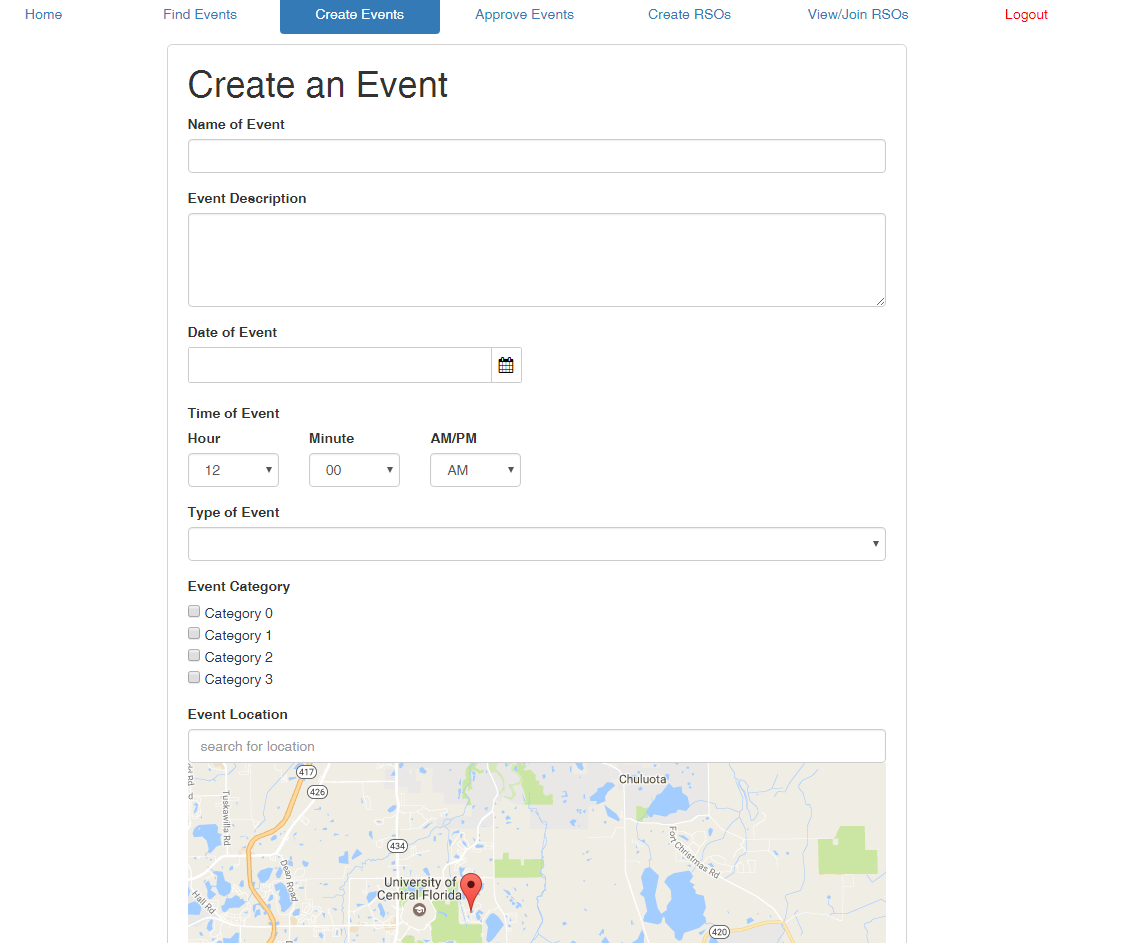


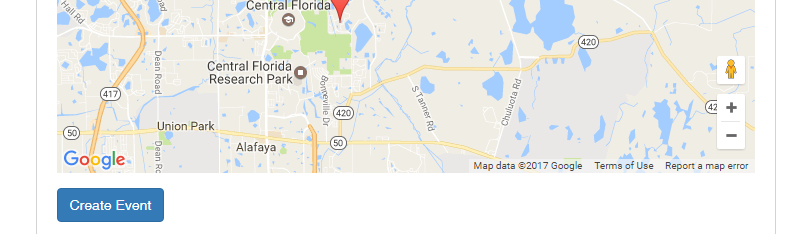
Super admin logged in (also homepage):

  
Find Events(Public Events):

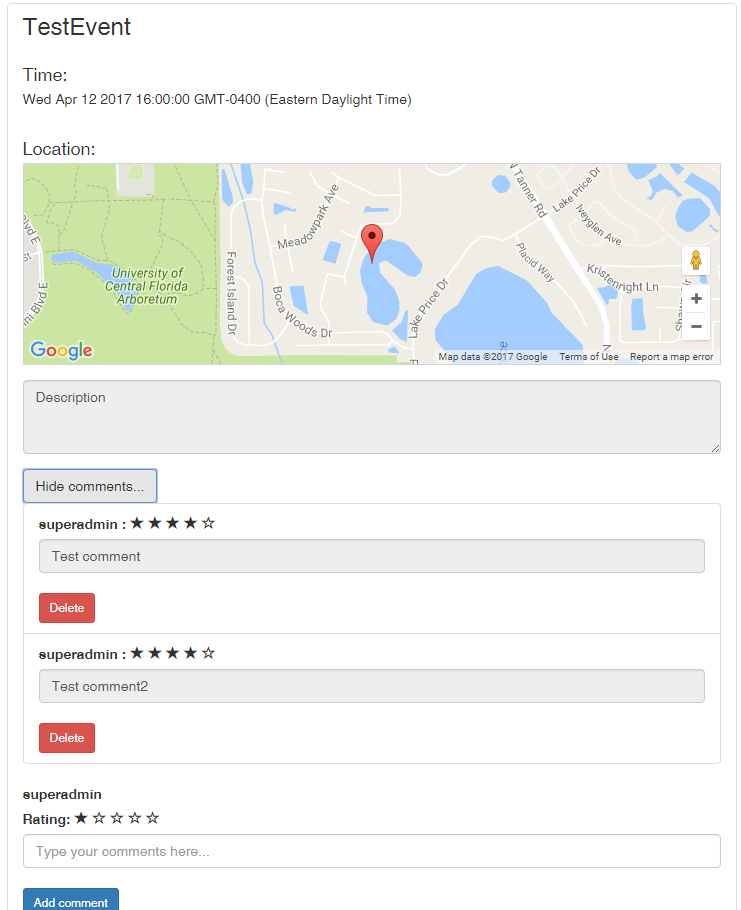


Create Event:

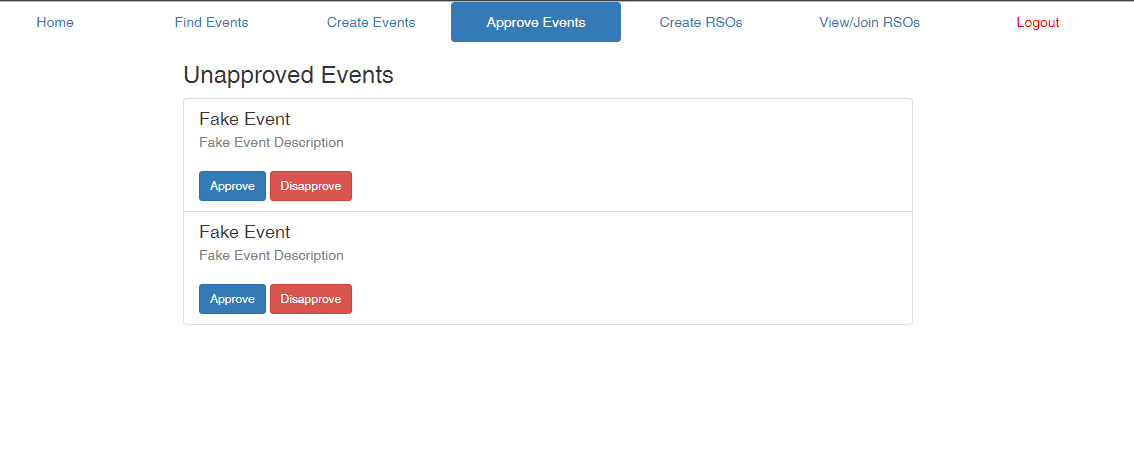




Event info and comments:

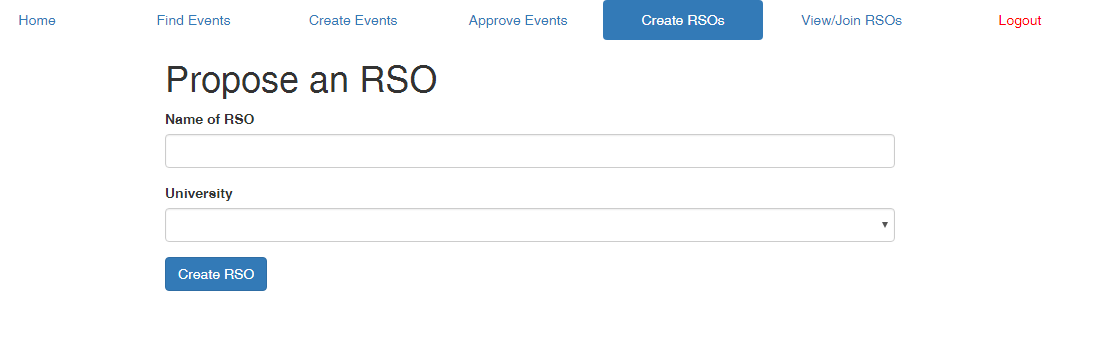


Approve Events (superadmin only):

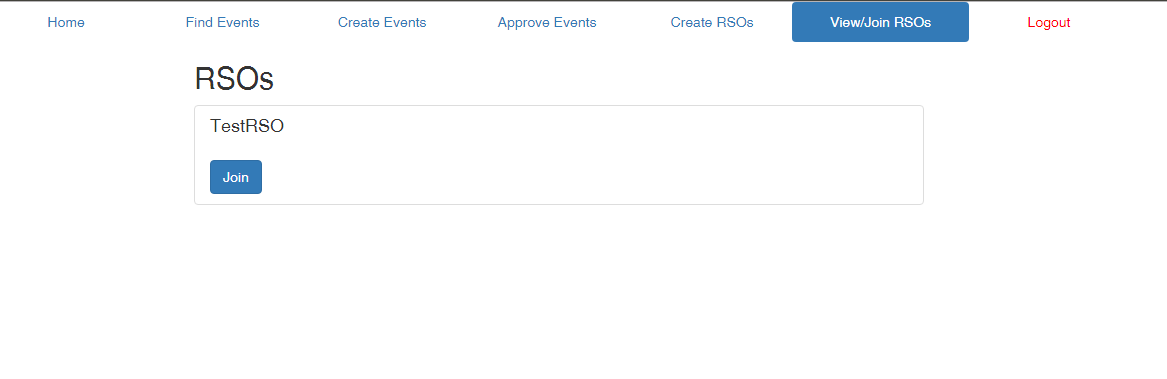


Create RSO (has drop down with all schools in DB):

View/Join



View/Join RSO:



# ER-Model



Figure X: This figure is our ER-Model diagram. We used this diagram to create our Relational data model. The note displays the information that can’t be shown by the ER-diagram.

# Relational Data Model

## SQL:

drop database if exists cop4710;

Create database cop4710;

use cop4710;

create table users (

uid bigint auto\_increment primary key,

username varchar(100) not null,

hashPass varchar(100) not null,

email varchar(100) not null,

phone varchar(100) not null,

firstName varchar(100) not null,

lastName varchar(100) not null,

role varchar(10) not null

);

create table superadmin (

uid bigint primary key,

foreign key(uid) references users(uid)

);

create table university(

unid bigint auto\_increment primary key,

uid bigint not null,

unName varchar(45),

location varchar(100),

description varchar(500),

numStudents integer,

foreign key (uid) references superadmin(uid)

);

create table student (

uid bigint primary key,

unid bigint not null,

foreign key(uid) references users(uid),

foreign key (unid) references university(unid)

);

create table admin (

uid bigint primary key,

unid bigint not null,

foreign key(uid) references users(uid),

foreign key (unid) references university(unid)

);

create table rso (

rid bigint auto\_increment primary key,

uid bigint not null,

unid bigint not null,

rsoName varchar(45),

rsoStatus varchar(10) default 'Pending',

foreign key (uid) references admin(uid),

foreign key (unid) references university(unid)

);

create table \_events (

eid bigint auto\_increment primary key,

uid bigint not null,

eventName varchar(45),

timedate datetime,

category varchar(45),

description varchar(500),

latitude varchar(45),

longitude varchar(45),

eventStatus varchar(20) default "Unapproved",

foreign key(uid) references admin(uid)

);

create table publicEvent (

eid bigint primary key,

foreign key(eid) references \_events(eid)

);

create table privateEvent (

eid bigint primary key,

unid bigint not null,

foreign key(eid) references \_events(eid),

foreign key(unid) references university(unid)

);

create table rsoEvent (

eid bigint primary key,

rid bigint not null,

foreign key(rid) references rso(rid),

foreign key(eid) references \_events(eid)

);

create table comments(

uid bigint,

eid bigint,

body varchar(500),

rating integer,

timedate datetime default current\_timestamp,

primary key(uid, eid, timedate),

foreign key(uid) references users(uid),

foreign key(eid) references \_events(eid)

);

create table rsoMembers(

rid bigint,

uid bigint,

primary key(rid, uid),

foreign key(uid) references users(uid),

foreign key(rid) references rso(rid)

);

## SQL Statements and Code

**Statements are in Bold**

**Create RSO:**

function createRSO() {

var rso = {uid: rsoParam.uid, unid: rsoParam.unid, rsoName: rsoParam.rsoName};

//maybe use a procedure here instead of a sql statement

connection.query("**insert into rso set ?**", [rso], function(err, rows) {

connection.release();

if(err) {

deferred.reject(err.name + ': ' + err.message);

}

if(!err) {

var rsoMember = {uid: rsoParam.uid, rid: rows.insertId};

console.log("Adding uid: " + rsoParam.uid+" to rid: "+rows.insertId +" member group");

connection.query("**INSERT INTO rsoMembers SET ?**", [rsoMember], function(err, rows) {

if(err) {

deferred.reject(err.name + ': ' + err.message);

}

});

deferred.resolve();

}

**Insert student into RSO:**

function insertMember(rsoId, userId){

console.log("user Id: " + userId +", joining rso: " + rsoId);

var rso = {rid: rsoId, uid: userId};

connection.query(`**INSERT INTO rsoMembers set ?**`, [rso], function(err, rows) {

connection.release();

if(err) {

deferred.reject(err.name + ': ' + err.message);

}

deferred.resolve();

});

}

});

return deferred.promise;

}

**Create New Event:**

function create(eventParam) {

var deferred = Q.defer();

pool.getConnection(function(err, connection) {

if(err) {

connection.release();

deferred.reject(err.name + ': ' + err.message);

}

console.log(eventParam);

if(eventParam.type == 'RSO')

{

eventParam.status = 'Approved';

}

else

{

eventParam.status = 'Unapproved';

}

var event = {

uid: eventParam.uid,

eventName: eventParam.name,

timedate: eventParam.time,

category: eventParam.category,

description: eventParam.description,

latitude: eventParam.latitude,

longitude: eventParam.longitude,

eventStatus: eventParam.status

};

connection.query("**INSERT INTO \_events SET ?**", [event], function(err, rows) {

if(err){

deferred.reject(err.name +": "+ err.message);

}

**View events:**

**(Private)**

function getPrivateEvents(uid) {

var deferred = Q.defer();

console.log(uid);

pool.getConnection(function(err, connection) {

if(err) {

connection.release();

deferred.reject(err.name + ': ' + err.message);

}

connection.query(`**SELECT e.eid, e.eventName, e.description, e.category, e.latitude, e.longitude, e.timedate FROM \_events e, privateEvent p, student s**

**WHERE p.eid = e.eid AND e.eventStatus = 'Approved' AND (p.unid = s.unid AND s.uid = ?)**`, uid, function(err, rows) {

if(err){

deferred.reject(err.name +": "+ err.message);

} else if(rows.length > 0){

deferred.resolve(rows);

}else{

connection.query(`**SELECT e.eid, e.eventName, e.description, e.category, e.latitude, e.longitude, e.timedate FROM \_events e, privateEvent p, admin a**

**WHERE p.eid = e.eid AND e.eventStatus = 'Approved' AND (p.unid = a.unid AND a.uid = ?)**`, uid, function(err, rows) {

connection.release();

if(err){

deferred.reject(err.name +": "+ err.message);

}

deferred.resolve(rows);

});

}

});

});

return deferred.promise;

}

**(Public)**

function getPublicEvents() {

var deferred = Q.defer();

pool.getConnection(function(err, connection) {

if(err) {

connection.release();

deferred.reject(err.name + ': ' + err.message);

}

connection.query("**SELECT e.eid, e.eventName, e.description, e.category, e.latitude, e.longitude, e.timedate FROM \_events e, publicEvent p WHERE p.eid = e.eid AND e.eventStatus = ?**", 'Approved', function(err, rows) {

connection.release();

if(err){

deferred.reject(err.name +": "+ err.message);

}

deferred.resolve(rows);

});

});

return deferred.promise;

}

**(RSO)**

function getRSOEvents(uid) {

var deferred = Q.defer();

pool.getConnection(function(err, connection) {

if(err) {

connection.release();

deferred.reject(err.name + ': ' + err.message);

}

connection.query("**SELECT e.eid, e.eventName, e.description, e.category, e.latitude, e.longitude, e.timedate FROM \_events e, rsoEvent r, rsoMembers m WHERE r.eid = e.eid AND r.rid = m.rid AND m.uid = ?**", [uid], function(err, rows) {

connection.release();

if(err){

deferred.reject(err.name +": "+ err.message);

}

deferred.resolve(rows);

});

});

return deferred.promise;

}

**Get Joinable RSOs query:**

function getJoinable(uid){

var deferred = Q.defer();

pool.getConnection(function(err, connection) {

if(err) {

connection.release();

deferred.reject(err.name + ': ' + err.message);

}

console.log("Getting all rso's for the user: "+uid);

connection.query("**SELECT r.rid, r.unid, r.rsoName FROM rso r WHERE NOT EXISTS (SELECT \* FROM rsoMembers WHERE r.rid = rid AND uid = ?**)", [uid], function(err, rows) {

//console.log(rows);

connection.release();

if(err) {

deferred.reject(err.name + ': ' + err.message);

}

deferred.resolve(rows);

});

});

return deferred.promise;

}

Example of a trigger used to mark a RSO as ‘active’

DROP TRIGGER IF EXISTS after\_insert\_rso\_member;

DELIMITER //

CREATE TRIGGER after\_insert\_rso\_member

AFTER INSERT ON `rsoMembers`

FOR EACH ROW BEGIN

UPDATE rso r1

JOIN

(

SELECT rid

FROM rsomembers

GROUP BY rid

HAVING COUNT(\*) > 4

) r2

ON r1.rid = r2.rid

SET r1.rsoStatus = 'Active';

END;

# Conclusion

## Database Performance

Due to the finite data that we had stored in we did not notice any poor database performance. We are hosting our data base on Amazons Relational Database system using the MySQL workbench to modify the different tables. Amazon’s RDS also allows us to monitor the just about every aspect of our relational database. Some of the different monitors include Write latency and read latency, which have never gone over 60ms and 12 ms respectively. The 60ms write latency was due to creating all of the tables.

## Desired Features

Some desired features that we implemented was the ability to use the google maps API to set a location and get the appropriate latitude and longitude for that location. Another feature that we could implement successfully was the rating system where each user who makes a comment on an event can rate (out of 5 stars) how successful the event was. For the login security we hashed all passwords of users on the backend that why no one could tap into our API and get user names and password of users. Some other features that we would have liked to implement would have been some sort of social media into our web application. We would have also liked to add a bit more pictures and styling to the user interface of the web application. I believe if we would have priorities a little better we would have been able to implement these features.

## Problems Encountered

The biggest hurdle of this project was learning all the different technologies (Javascript, MySQL, Angular 2, Node.js, Amazon Web Services) that we had to know to get this project to work successfully. Another problem that we encountered but didn’t have time to solve was how to reverse geocode a pair of coordinates (Latitude and longitude) into an address using the google API. We tried several different methods but came up unsuccessful.

## Summary

Choosing a language that we knew nothing about in the beginning of the project was a little daunting at first. We could have gone with some languages that we already knew like php and C#, however, we saw this project as an opportunity to learn a new language, node.js and Angular 2. We also could have just left the Relational database local, however, we decided to put it up on Amazons Web service. Overall, the knowledge acquired while doing this project was priceless. Before taking this class, we had a very small idea of how a website worked with a backend database and now we have a great deal of knowledge into how web applications work with a backend.