

Members: Dave Nandlall & Matthew Balwant

College Event Website

Group 15

# Table of Contents

Project Description ………………………………………………………………………………………………….......….. 2

Graphical User Interface…………………………………………………………………………………………….……... 3

Entity Relationship Model……………………………………………………………………………………………...… 4

Relational Model……………………………………………………………………………………………………….....….. 5

Population of Database Tables…………………………………………………………………………………………. 6

SQL Examples and Results....................................................................................................... 8

Conclusions & Observations.................................................................................................... 13

# Project Description

The main focus of this project was to create a website that allowed student to create and view events given his/her university. In addition, students were also allowed to form Research Student Organization Groups (RSO) within their granted university. Initial assumptions assumed that there were mainly three different types of users, with different levels of access to the website. At the highest level was the super administrator who was in charge of creating a university profile and in charge of approving events, proposed by an administrator. An administrator account would be the next level of access, lower than the super admin but higher than that of a student. Administrators were allowed to own RSO groups and create public, private or rso events. At the lowest level were student accounts, which by default had no permissions to modify anything. Students can only join/view events, comment/rate, and request to form an RSO group.

In order to make the project more realistic, some constraints had to be put in place in order for the website to accomplish its goal. Our plan was to create instead 4 levels of access, designating one user above a super administrator privilege. This user was the webmaster, who was in charge of overseeing the entire site. The webmaster would approve/disapprove a super administrator, who would in turn control an entire university. The super administrator is the next level of access and in this instance we decided that there is no need for multiple super administrators. Each university only has one super administrator, and this super admin would select other candidates to become administrators. Once the administrators were selected, they would form the appropriate actions for the students to follow.

Furthermore, in order to make all this work, we structured our site on a first come first served basis. The first account registered for a university becomes a super admin by default. This would have to be approved by the webmaster. The following accounts are registered by default as administrators given a particular number of admins, we allow for a university. The number of admins are chosen based on the amount of students registered per university. As the number of students grow, the amount of administrators also expands, dividing the privileges of overseeing student activities equally. Admin accounts are requests, and the one super admins for that university has to grant an admin account permission. Once super admin and admin accounts are created for a given university, every other account by default are registered as a student account. The system is developed in a hierarchical format to ensure given conditions are met.

# Graphical User Interface (GUI)

Platform & Development Tools

The main development platforms we chose to develop our GUI were Linux and Mac OS X. In addition, we did perform testing on Windows 10 machines. All of the source code was written and compiled in IntelliJ which is a cross-platform integrated development environment. In addition, some other tools such as Notepad++ and Sublime text editors were used to modify HTML and CSS files. Furthermore, our group created a GitHub repository to collaborate on the project together.

Languages

The development languages we chose to create the front end of the website were mainly HTML5, CSS3, and Bootstrap. These provided the front-end framework for a majority of our website graphical user interface form elements. In addition, we used JavaScript, jQuery and AJAX programming languages to provide behavior to elements on our website. Lastly, in regards to the backend development, we chose Java as our primary language. All of the server side actions were written in Java with the help of different imported Java libraries.

Database Management System

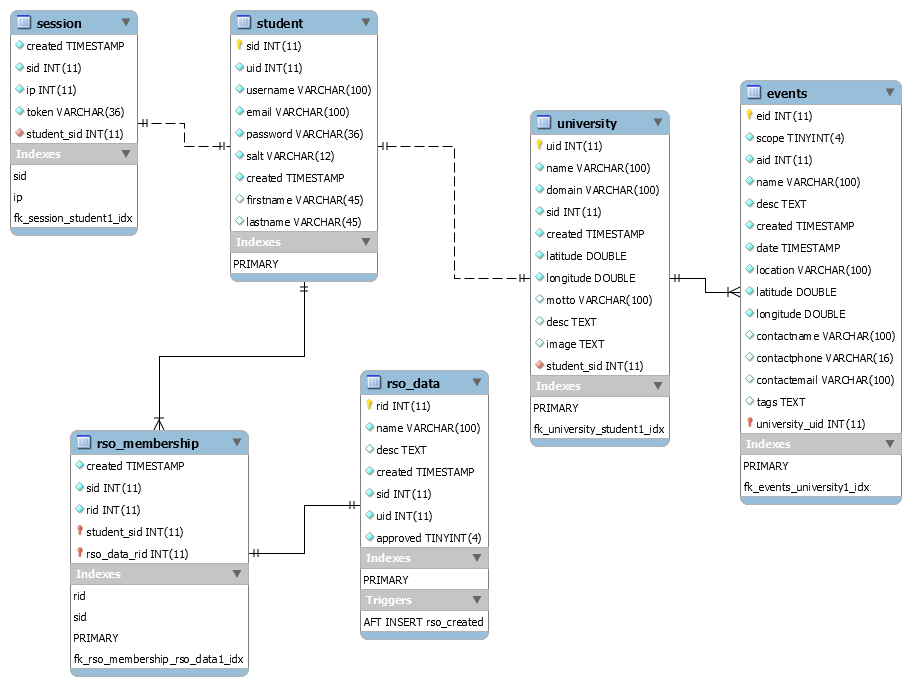
The DBMS we chose to use was MariaDB which is a fork of MySQL. We chose MariaDB due to the fact that it provided more features and better performance in developing our website.

# Entity Relationship Model



The figure above shows the entity relationship model that we used in order to structure our website. Some things to notice are that the entities are shown in blue, relationships are shown in green and the attributes are shown in purple. In this layout, we show the webmaster who is responsible for creating super admins. The way the system works is that the first account that creates a university profile, becomes the single super admin for that university. A university profile can only be created if one is not made already. Then the super admin chooses the users to be admins for the university, and once all admin accounts are satisfied, all other users are students. The webmaster approves all super admins, granting them permission to oversee the actions of a university profile. In turn, superadmin approve admin functions and admins approve student actions. In regards to the relationships, some to note are that at most one event is organized by an RSO, a student must be enrolled in at most one university, and RSO’s have a strong relationship with a student due to the fact that students are the ones who submit an RSO request.

# Relational Model



The figure above shows our relational model for our database. The relational model was created in MySQL workbench, using our database script. Some relationships to note are that there is one-to-one relationship with session and student. Session is a table used to store information about a student through the login process. There are a one-t0-many relationships with student and RSO membership, due to the fact that the student can create and join many RSO. RSO data is a different table used to store RSO data for a single RSO membership, so this has a one-to-one relationship. The student and university tables have a one-to-one relationship because a student can only be registered under one university. Lastly, university and events have a one-to-many relationship, because a university can have multiple events occurring at the same time.

# Population of Database Tables

## Populated University Tables (2 universities)

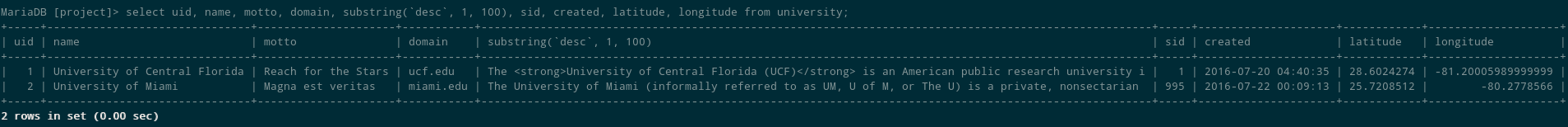


Figure 1 – University profile information in the database

## Populated Students Table (20 students)

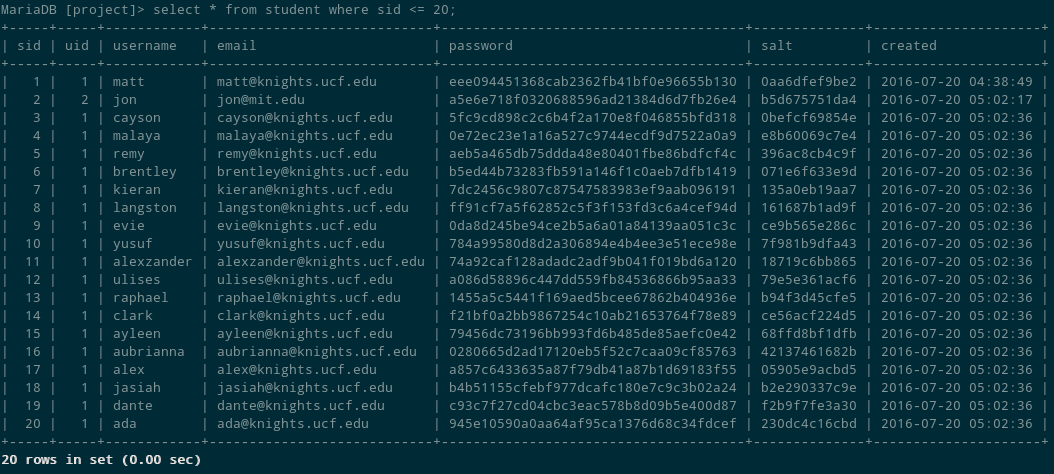


Figure 2 – Student information in the database

## Populated RSO Tables (3 RSO’s)

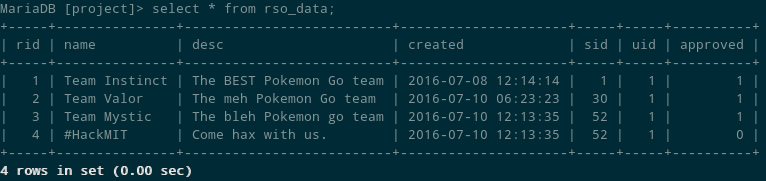
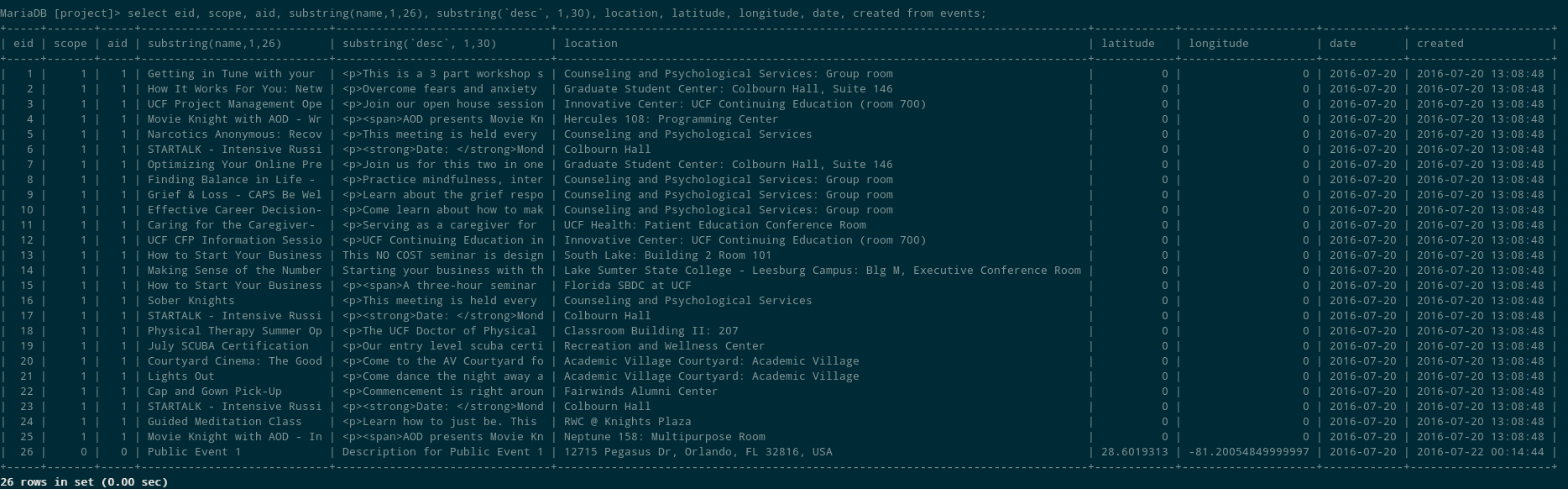


Figure 3 – Example of RSO group information

## Populated Events Table (5 Events)

This screenshot shows how event information is populated within our database. The events table contains a lot of information, so two screenshots are provided to display all columns in our table.



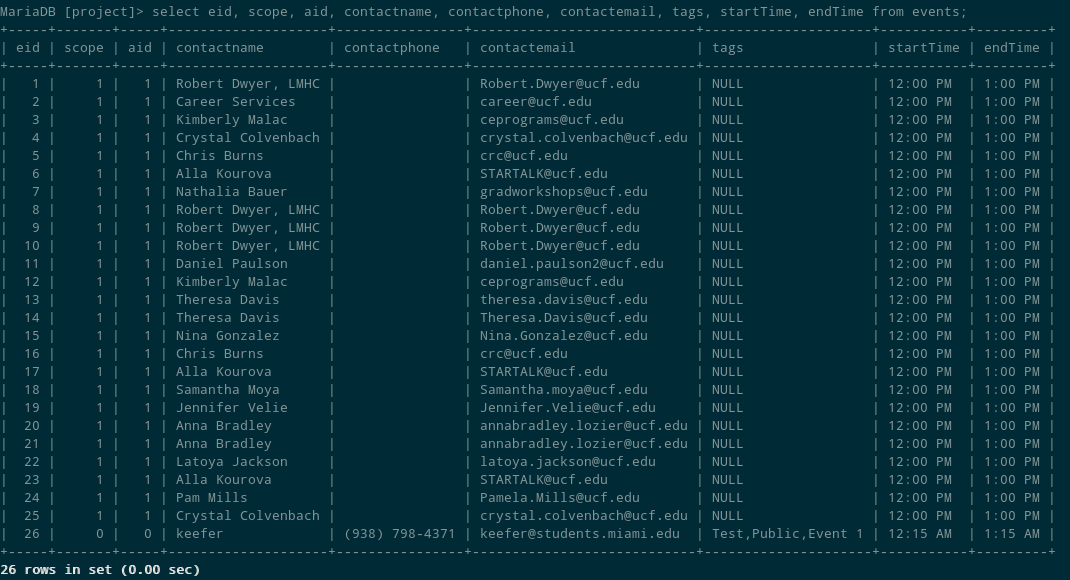


Figure 4 & 5 – The event information stored in the database.

## Populated Comments Table (3 Comments)

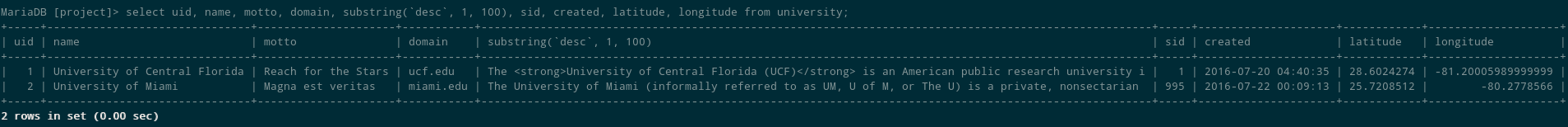


Figure 6 – The messages table with sample comment data.

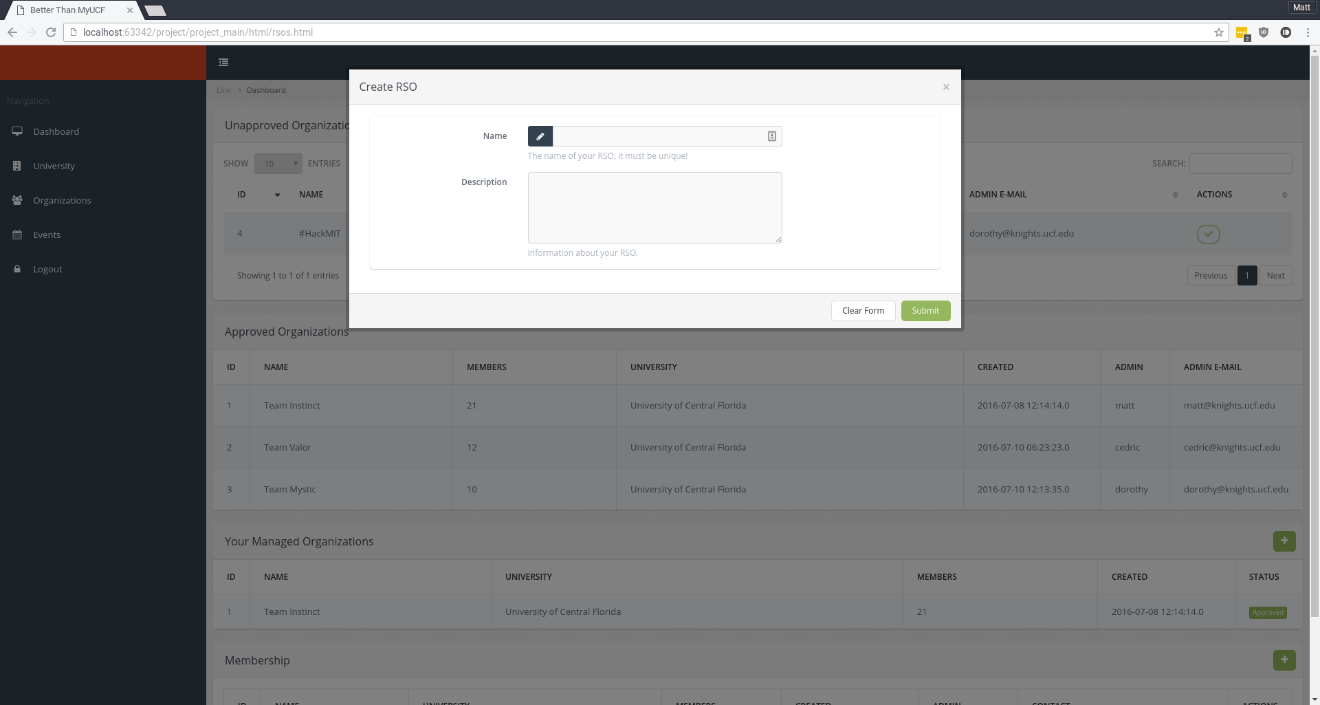
# SQL Examples and Results

Creating a new RSO (Create RSO form)

The process of creating a new RSO is performed with the following SQL insert. The query takes all the user entered form data, along with other attributes and inserts that into the RSO data table in the database. Once the student creates the RSO, it awaits approval and is displayed in both the student and admin panels with the current status.

SQL Query

|  |
| --- |
| @SqlUpdate("insert into rso\_data (`name`, `desc`, `sid`, `uid`) values (:name, :desc, :s.sid, :s.uid)") |



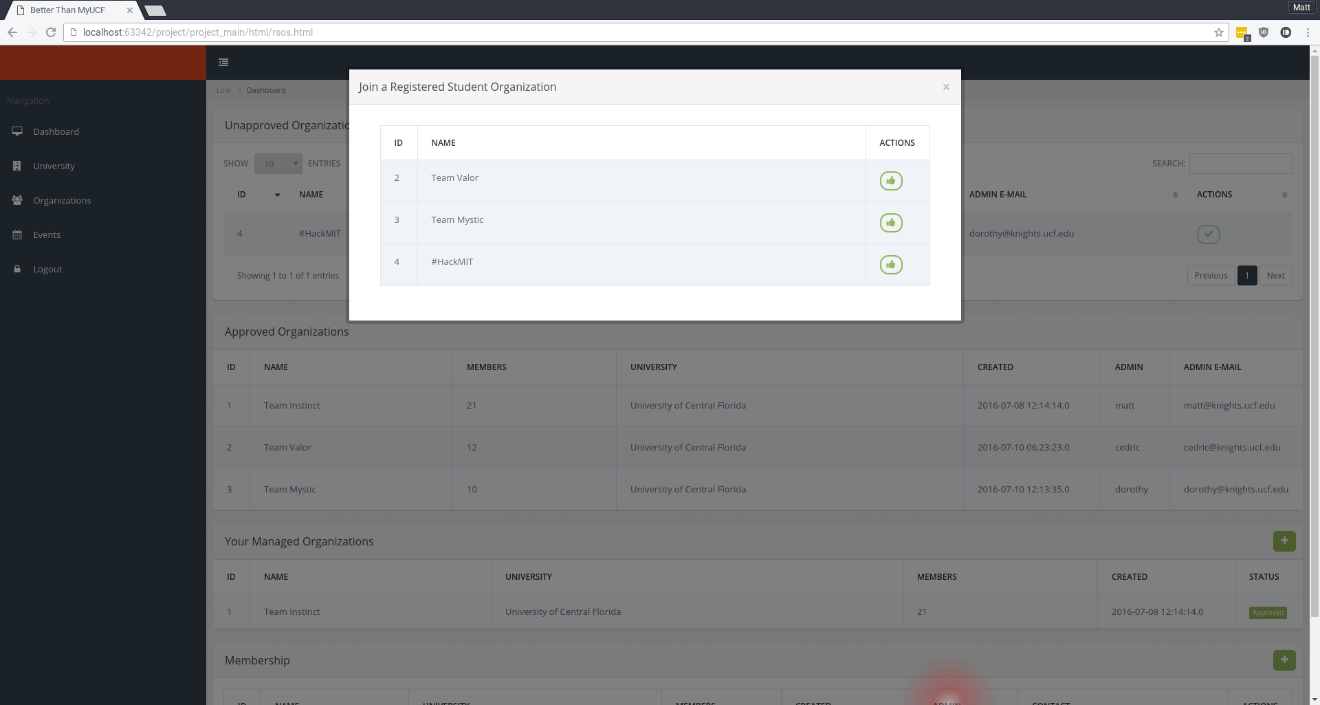
Inserting a new student into an RSO (Join RSO)

The query below is responsible for inserting a new student into an RSO. The query works for both creating and joining a new RSO. The student is inserted into the database as a member of that RSO, with the RSO id and the student id. Under the organizations tab, the student will be shown a modal window asking him/her which RSO they would like to join. The student then selects the RSO to be joined and is notified that his/her membership is awaiting approval by an administrator. If the student wishes to unjoin an RSO, they are allowed to do so on their student dashboard panel.

SQL Query

@SqlUpdate("insert into rso\_membership (`rid`, `sid`) values (:rid, :s.sid)")

Output Screenshot – Student Panel Join/Unjoin an RSO



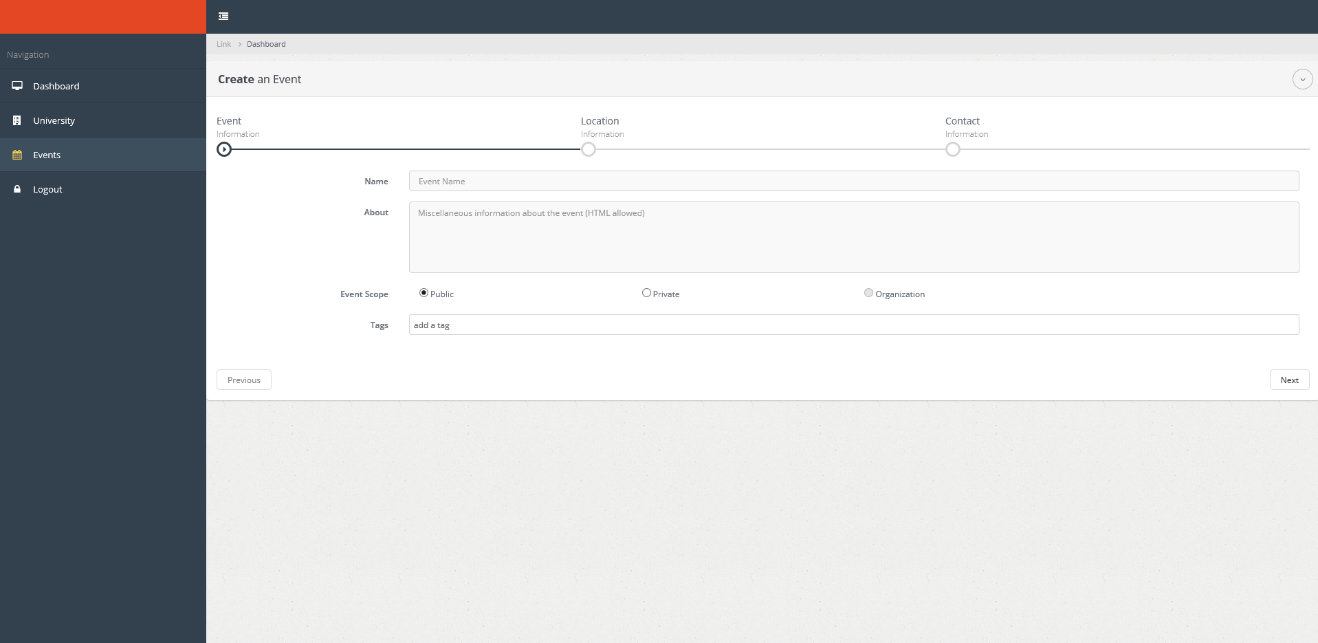
Creating an Event

The following query is used to create new events for a given university. The query is a simple insert that stores form data for a given event entered by a user. Example screenshots of the create event form are given below. The create event form is similar to a wizard, which navigates the user throughout the process of creating an event. For the screenshot we will only show the first page of the event screen. The screenshot after will demonstrate how a created event is displayed to the user.

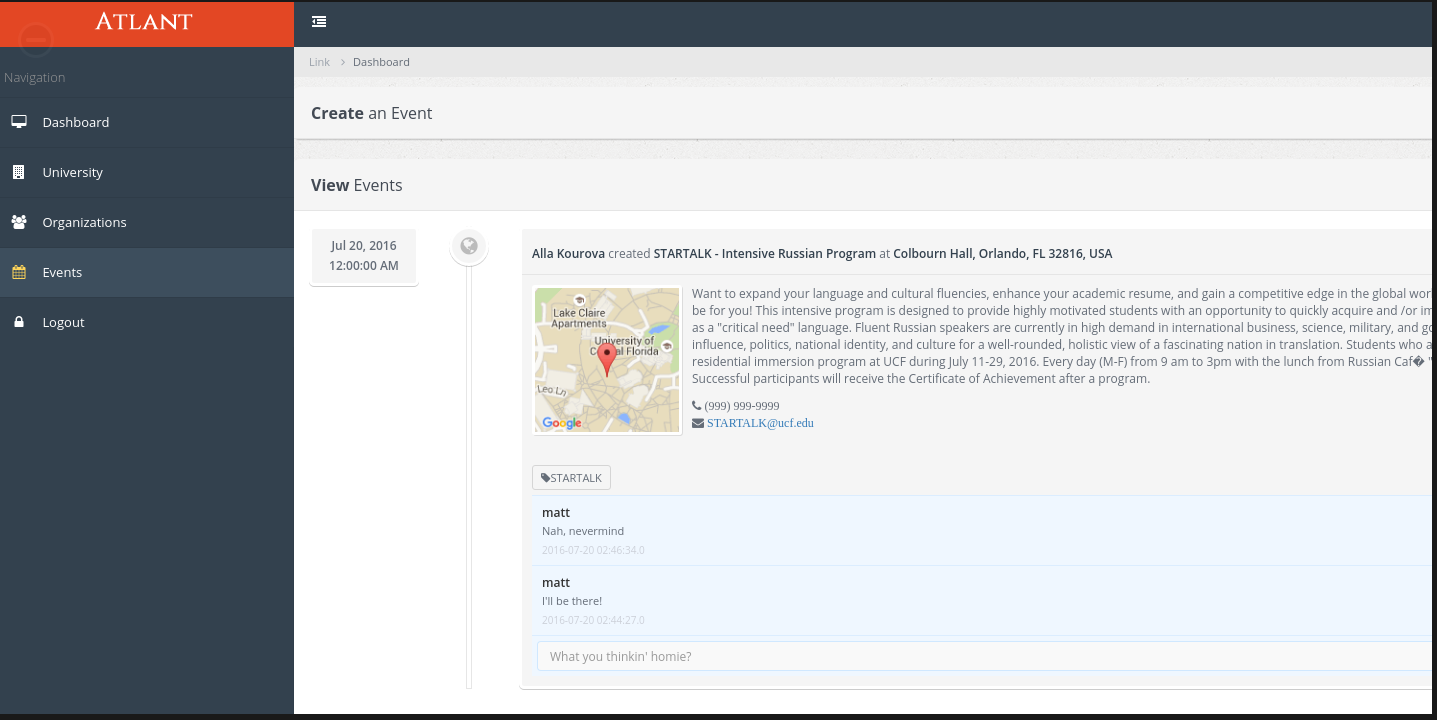
SQL Query

|  |
| --- |
| @SqlUpdate("insert into events (`eid`, `scope`, `aid`, `name`, `desc`, `date`, `startTime`, `endTime`, `location`, `latitude`, `longitude`, `contactname`, `contactphone`, `contactemail`, `tags`) values (:e.eid, :e.scope, :e.aid, :e.name, :e.desc, :e.date, :e.startTime, :e.endTime, :e.location, :e.latitude, :e.longitude, :e.contactname, :e.contactphone, :e.contactemail, :e.tags)") |

Output Screenshot – Event Form



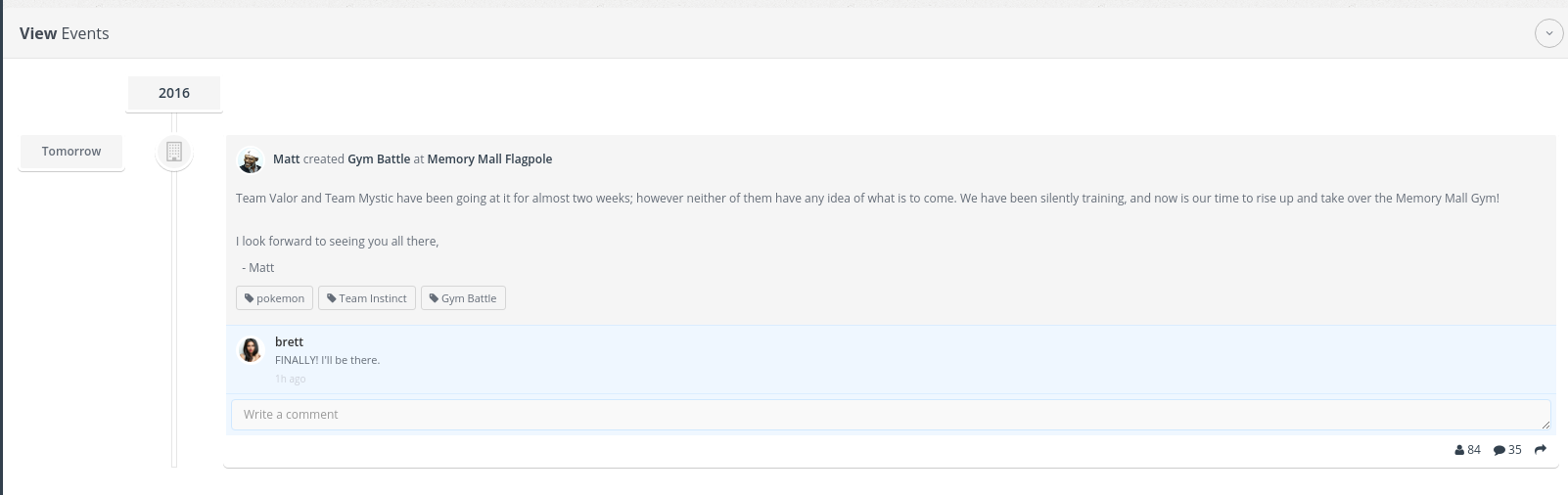
Output Screenshot – Created Event



Creating, Adding & Modifying Comments

|  |
| --- |
| The first query listed below handles both creating and adding comments. Creating and adding comments perform the same functionality so one query was sufficient enough to satisfy. The users simply enter a comment into the text box on the form, and presses the enter key to submit the comment.  SQL Query |
| @SqlUpdate("insert into message (`eid`, `sid`, `message`) values (:eid, :sid, :message)") |

Output Screenshot – Create, Modify and Add Comments



Displaying Events – Public, Private, and RSO

The SQL queries to display the events were standard SELECT statements. In order to place constraints on the type of event that is to be viewed, we used an integer value to allow access to view the event. The scope variable changed according to the type of event posted. The value zero was used for the scope when it was a public event, 1 for a private event and 2 for RSO events. The university id was also passed to the query to select the appropriate university hosting the event.

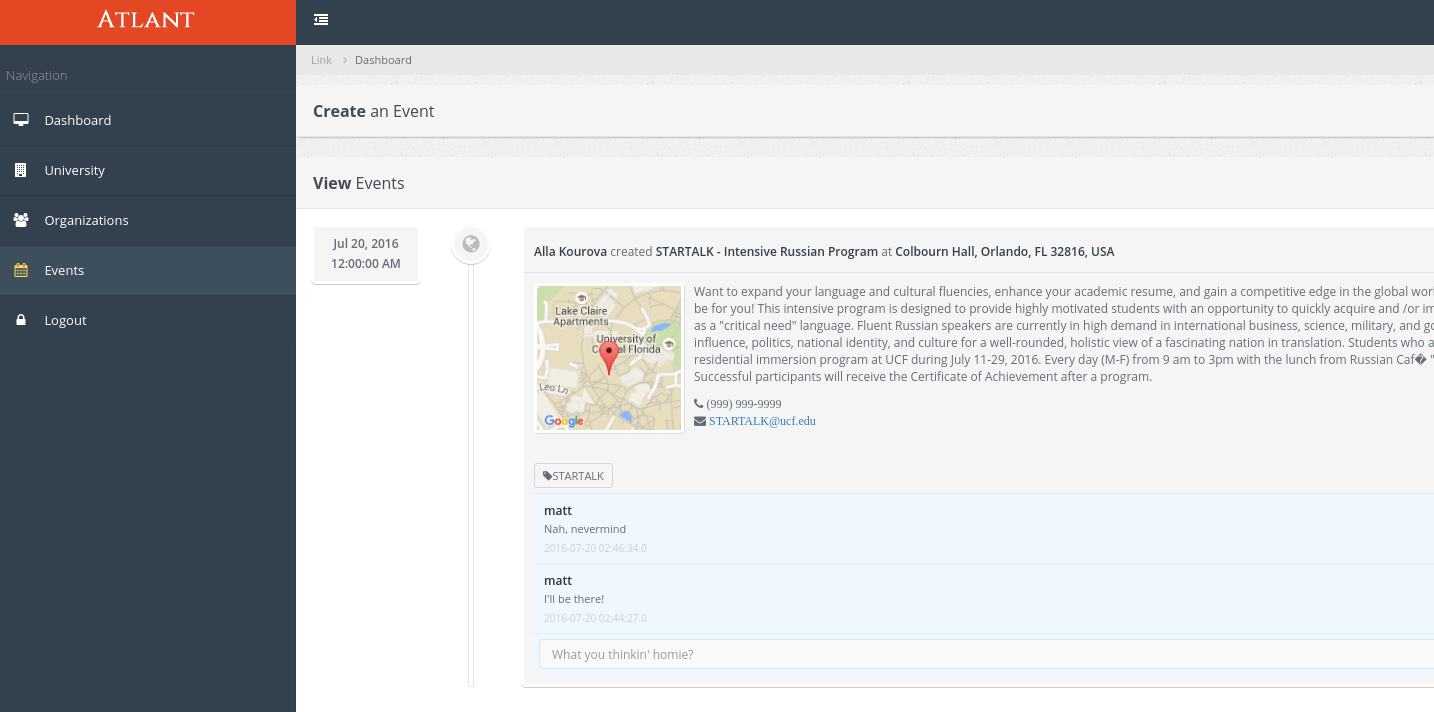
SQL Query

Public - @SqlQuery("select \* from events where scope = 0 and aid = :u")

Private - @SqlQuery("select \* from events where scope = 1 and aid = :u")

RSO - @SqlQuery("select \* from events where scope = 2 and aid = :u")

Output Screenshot – Displaying Events



# Conclusions & Observations

Database Performance

The performance of the database was very good in our opinion. The script written to populate the database with 1000+ users successfully executed and displayed the results in optimal time. The record keeping in the tables were maintained properly, avoiding duplicated entries. The indexing schema we created to search user information and other constraints worked flawlessly and performed well. The use of MariaDB as our DBMS made the database performance much better. There are options available with MariaDB that allowed our database to perform much more optimal than choosing a MySQL database.

Desired Features & Functionalities

In the process of creating this site, we felt that we accomplished the basic features and functionalities we set out to accomplish. The login system for our users was secure in that passwords were hashed and encrypted when stored into the database. One feature we explored was to be able to pull the RSS feed from the UCF events webpage. This would have been better at auto-populating the database with current events rather than us creating the events manually. We also wanted to incorporate social media integration within our sites to allow users to share events, but could not do so due to the development timeframe.

Problems Encountered

The most challenging issue we face when creating this website, were the constraints to place on the database. The initial setup of the hierarchy posed the most difficult. In order to satisfy the constraints, we had to modify the restrictions as we saw fit. We came into this project with experience in working with a database as well as knowledge of creating a web GUI. The structure of our database was carefully executed in order to prevent any data manipulation. In this regards, we feel that it was an advanced database application already. If we were to enhance our schema, we could possibly introduce advanced SQL statements to query our database, but in such a case as our website, our queries were sufficient enough.