Jesse Alsing and Chandler Douglas

Database Driven events page for universities

Group #16

COP 4710 Fall 2018

Contents

[Project Description / Overview 2](#_Toc530776145)

[GUI 2](#_Toc530776146)

[ER-Model 10](#_Toc530776147)

[The Relational Data Model 14](#_Toc530776148)

[Table Populations 14](#_Toc530776149)

[SQL Examples and Results 14](#_Toc530776150)

[Constraint Enforcements 21](#_Toc530776151)

[Software Installation instructions 21](#_Toc530776152)

[Advanced Features 22](#_Toc530776153)

[Conclusions and Observations 22](#_Toc530776154)

[Database Performance 22](#_Toc530776155)

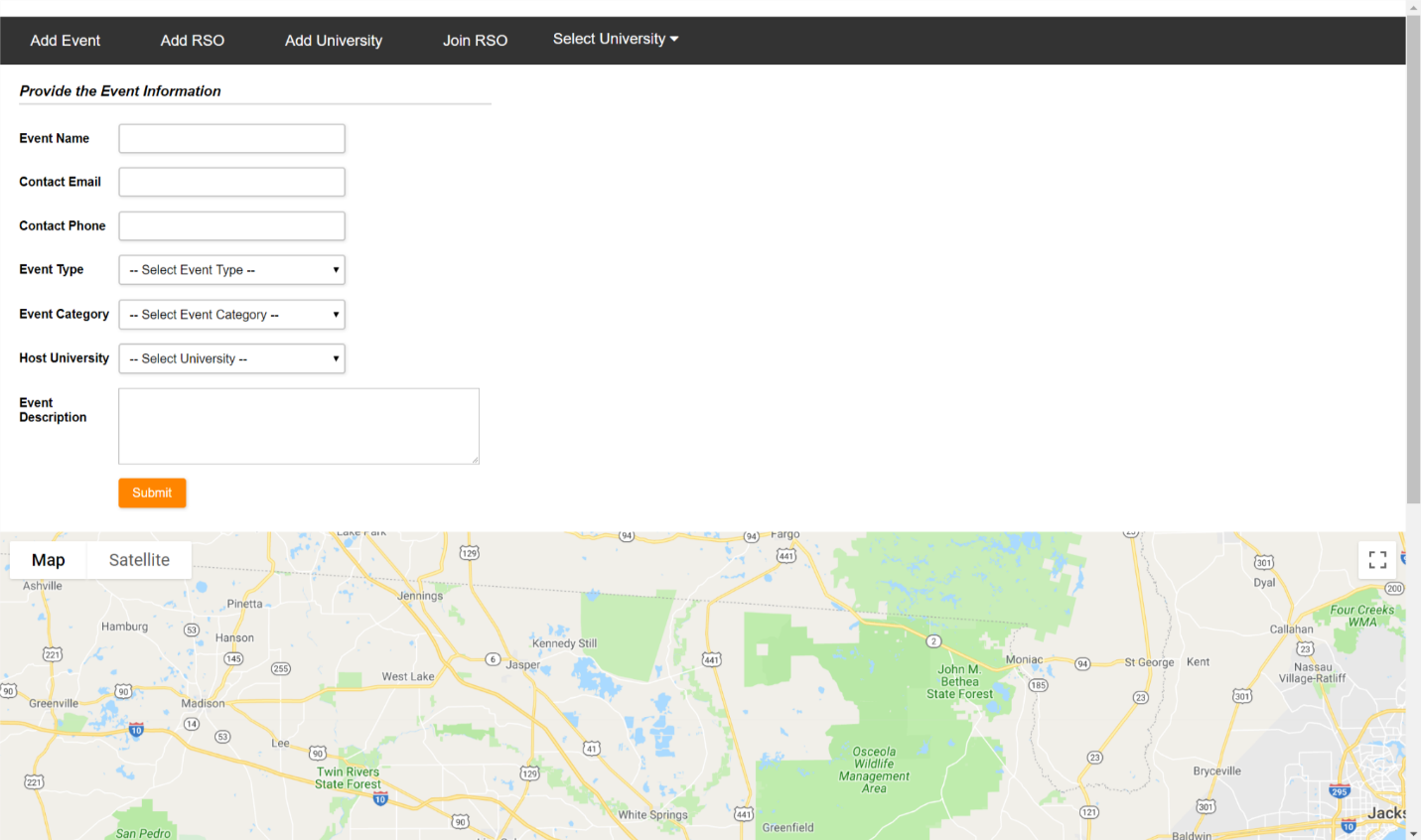
[Desired Features and Functionality 22](#_Toc530776156)

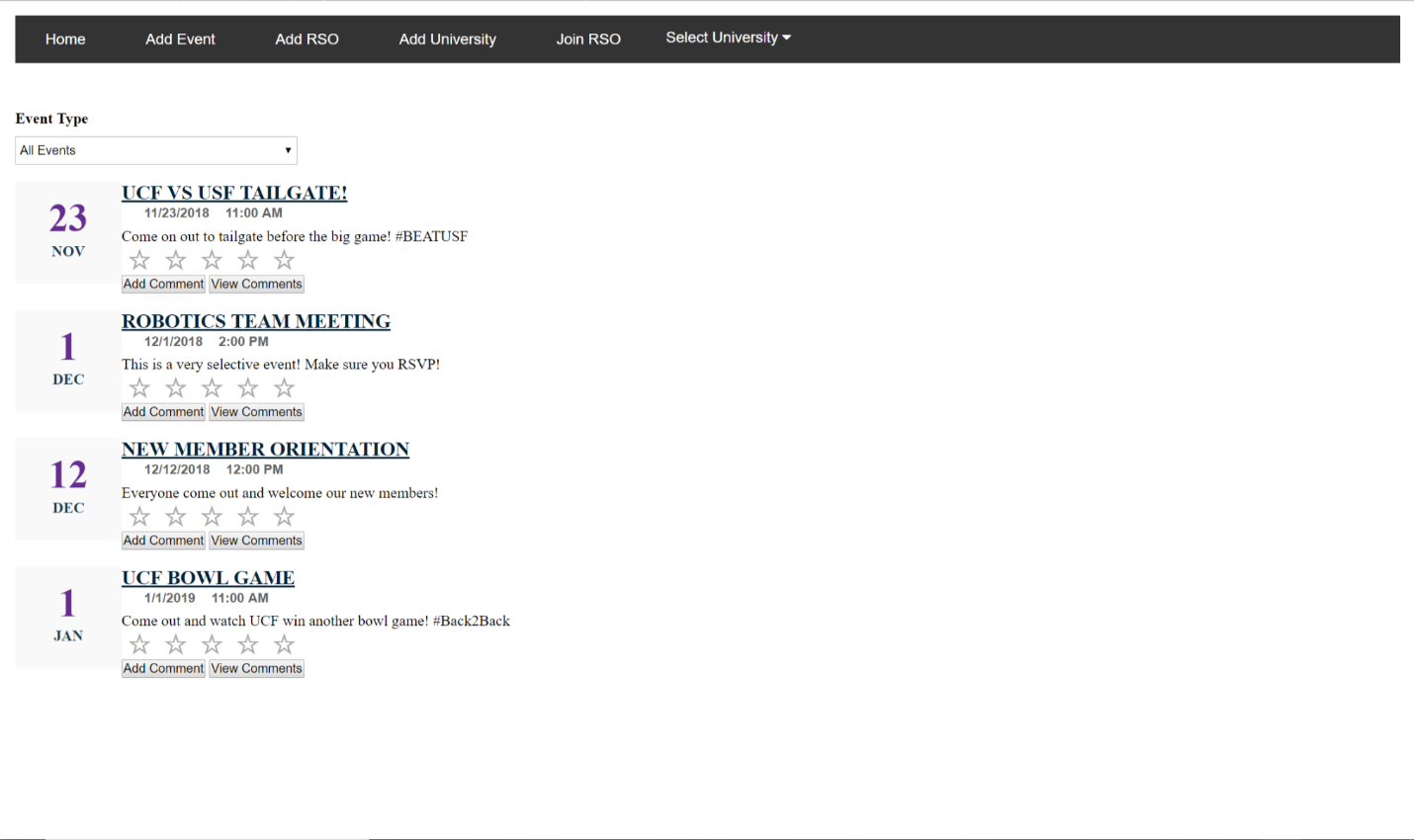
[Problems Encountered and Lessons Learned 22](#_Toc530776157)

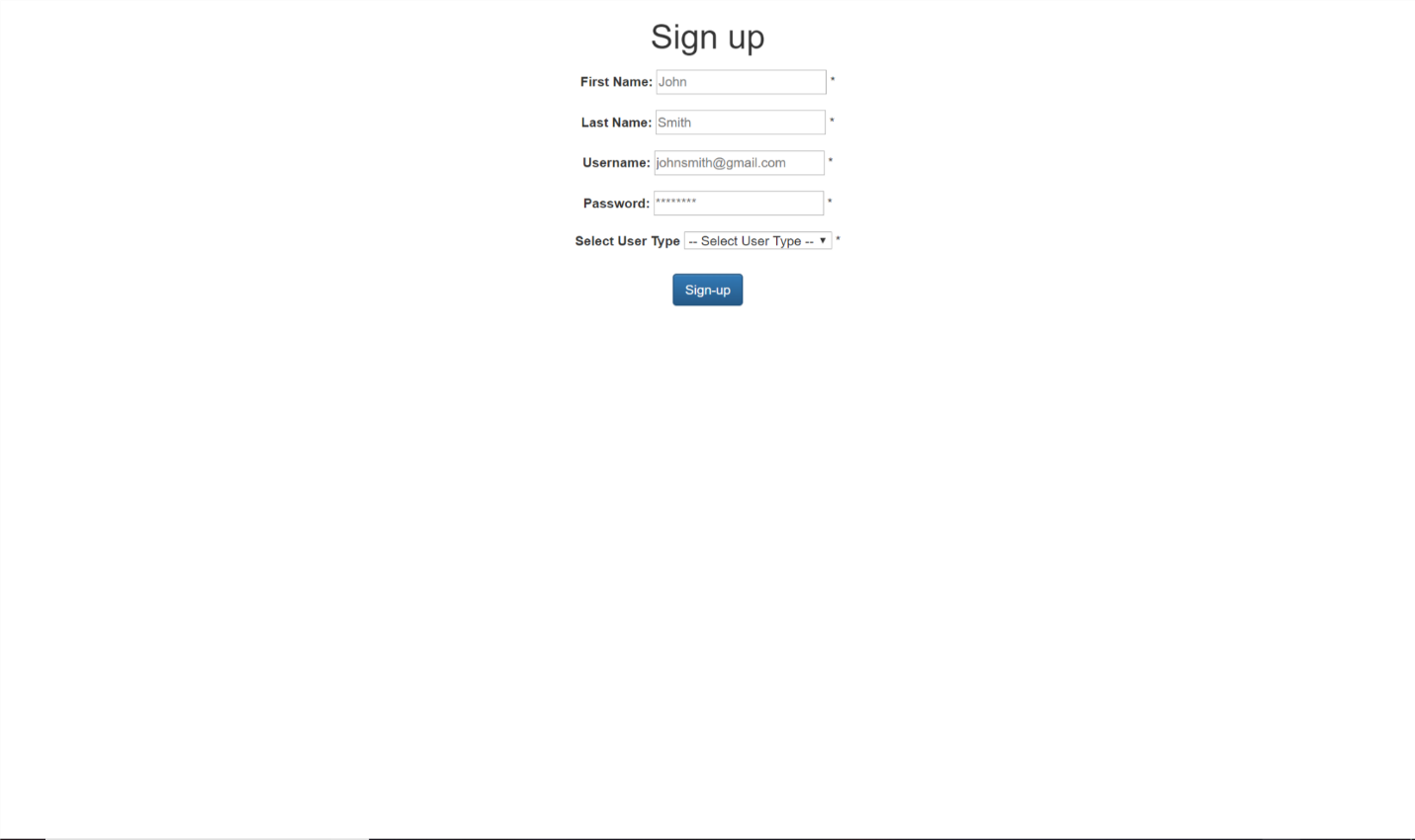
# Project Description / Overview

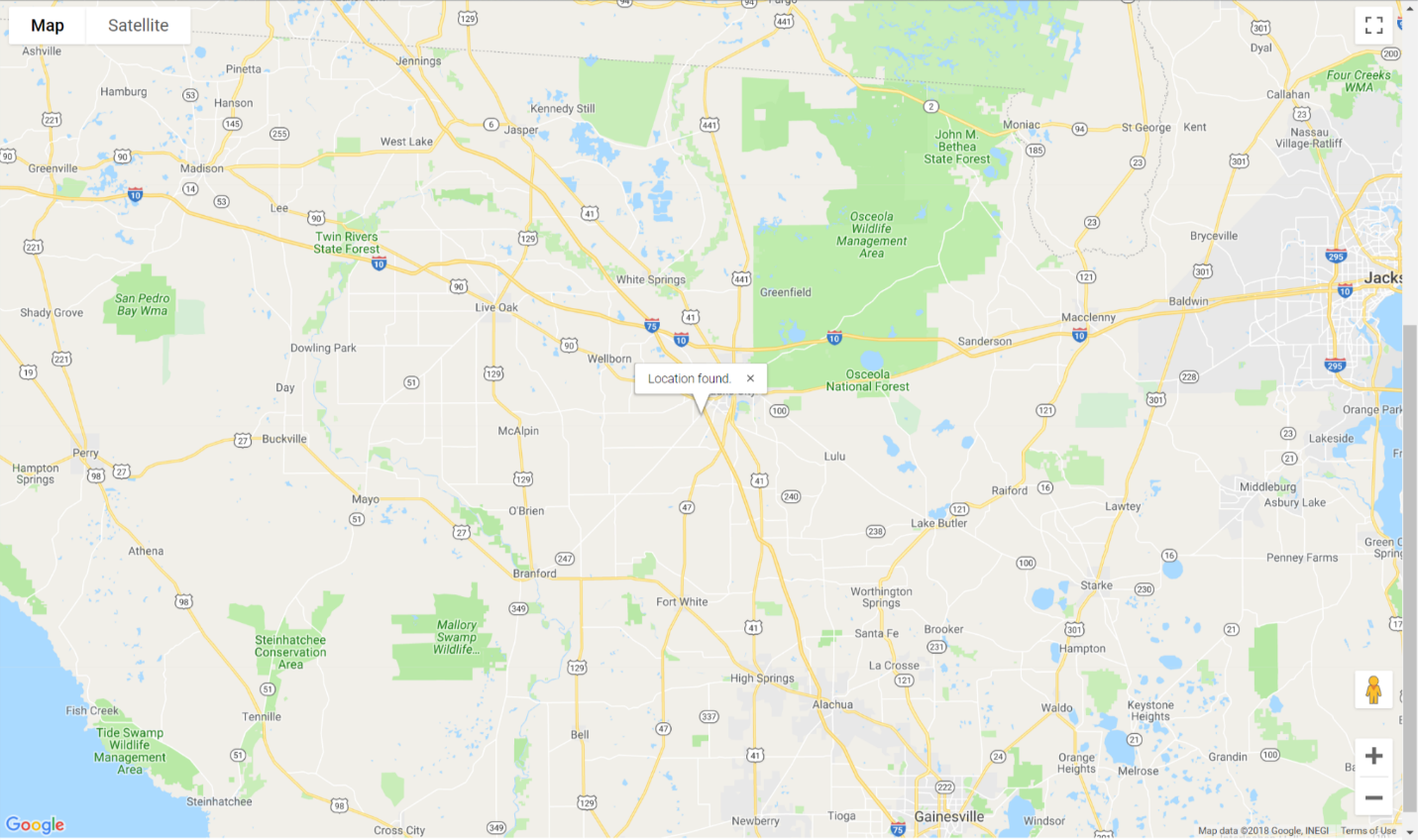
This project gives students and university admins a way to consolidate events from different universities into a single website. This website allows users from any university to create and edit events along with joining and creating Registered Student Organizations for multiple universities. The website has a simple UI to allow any level of technical savvy access to a university’s events. We accomplished this goal by building a website using the Model, View, Controller (MVC) architecture in the C# language. The front-end of the project is built entirely in .cshtml (a hybrid of C# and HTML) and it uses JavaScript to connect to a REST API Controller layer. From there, we’ve built a Contracts project that the front-end requests are converted to. These Contracts (named Request and Response, respectfully) are sent to the Database Connection layer where we use ADO.NET to insert into the Database, hosted in SQL Server and interfaced with SQL Server Management Studio.

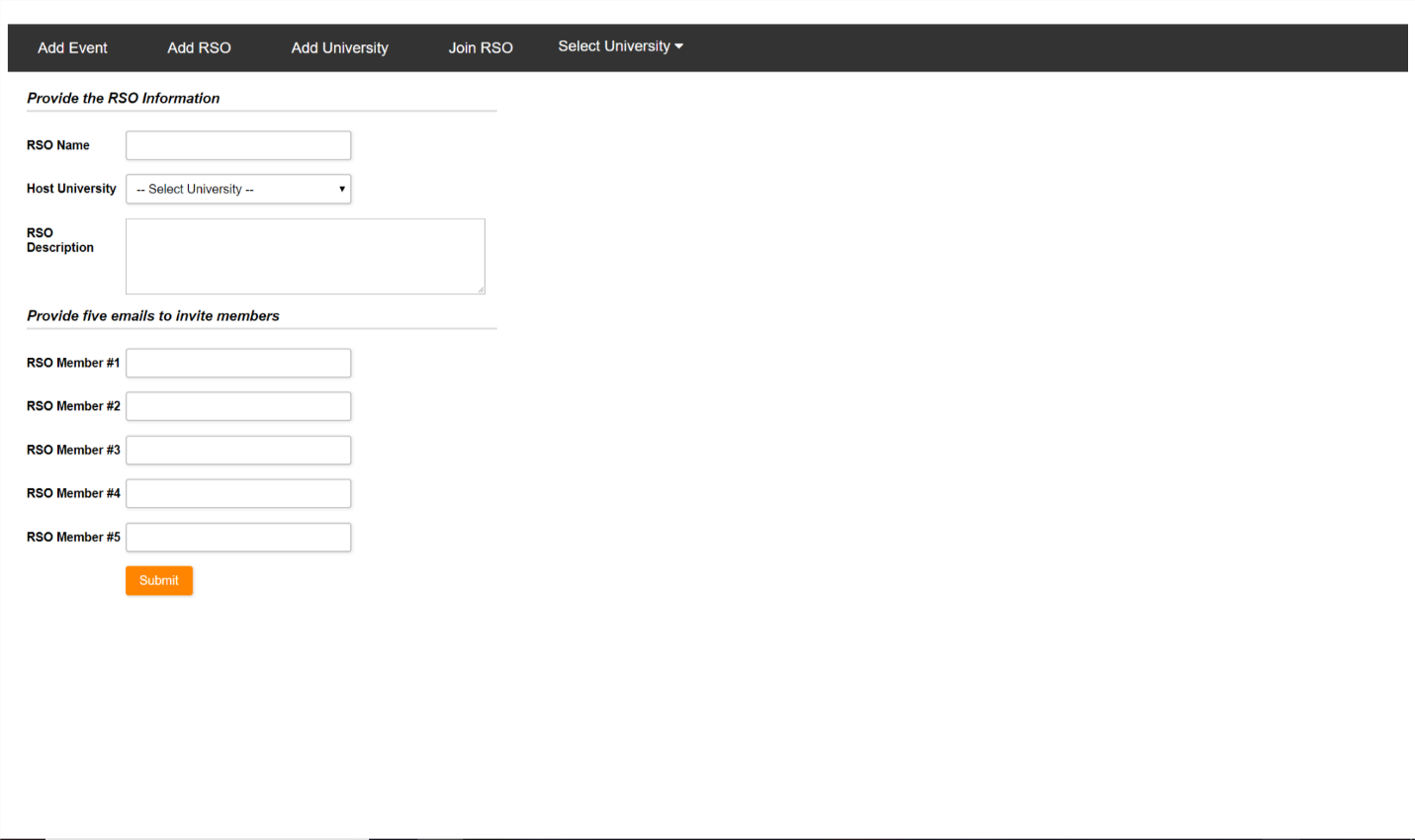
# GUI

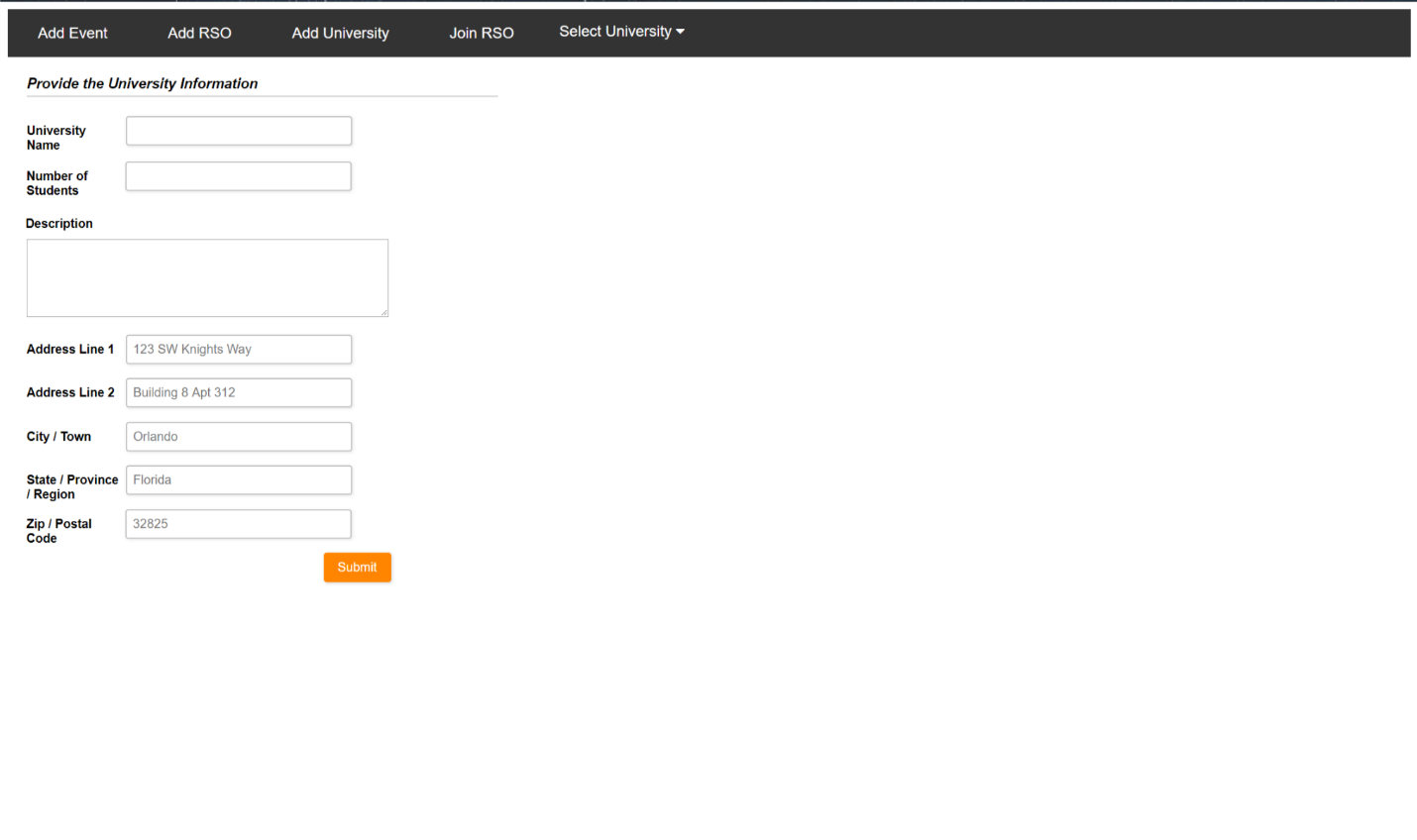
**The create event page with a Google Maps API with geolocation**

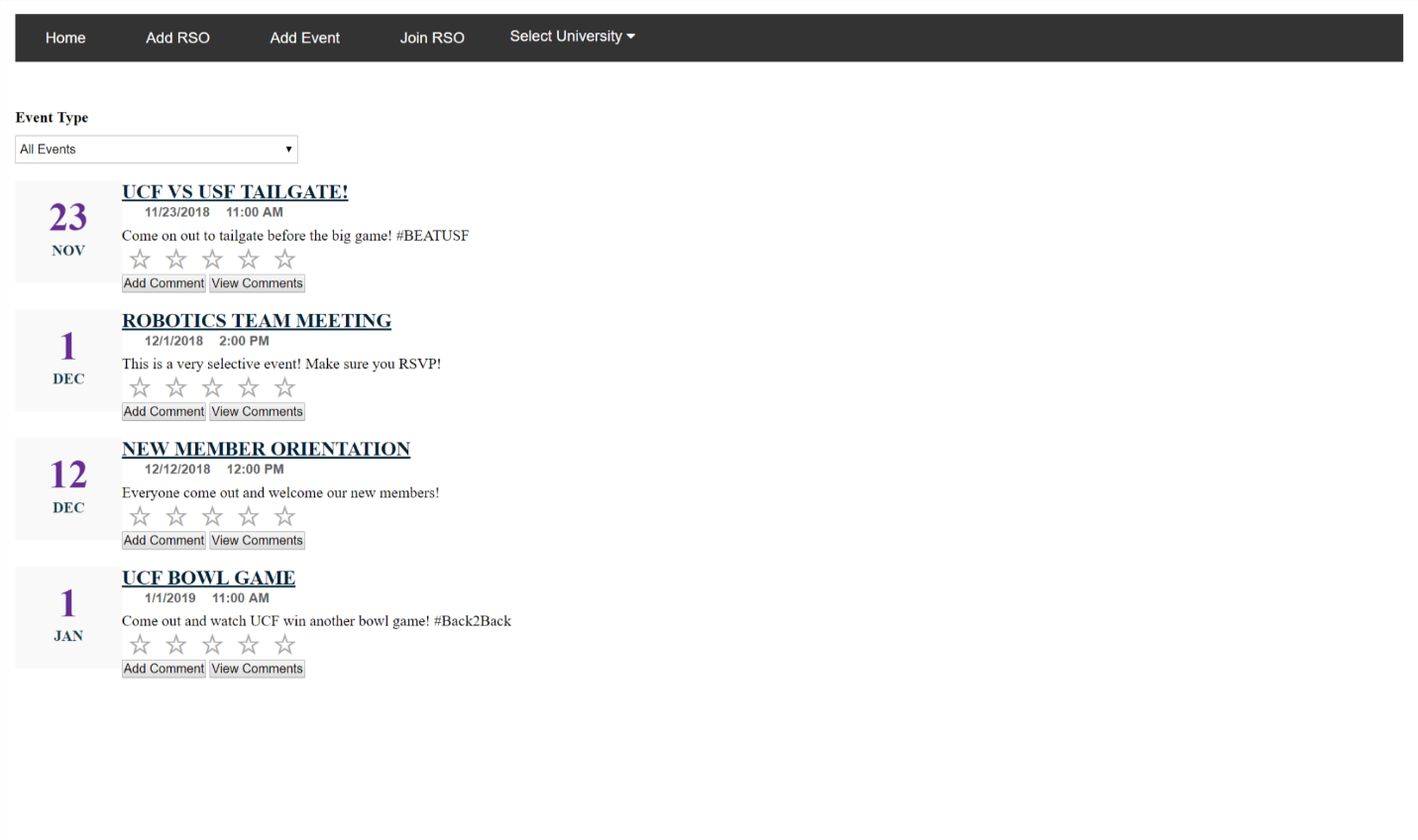
**The User Home Page with all of the Events**

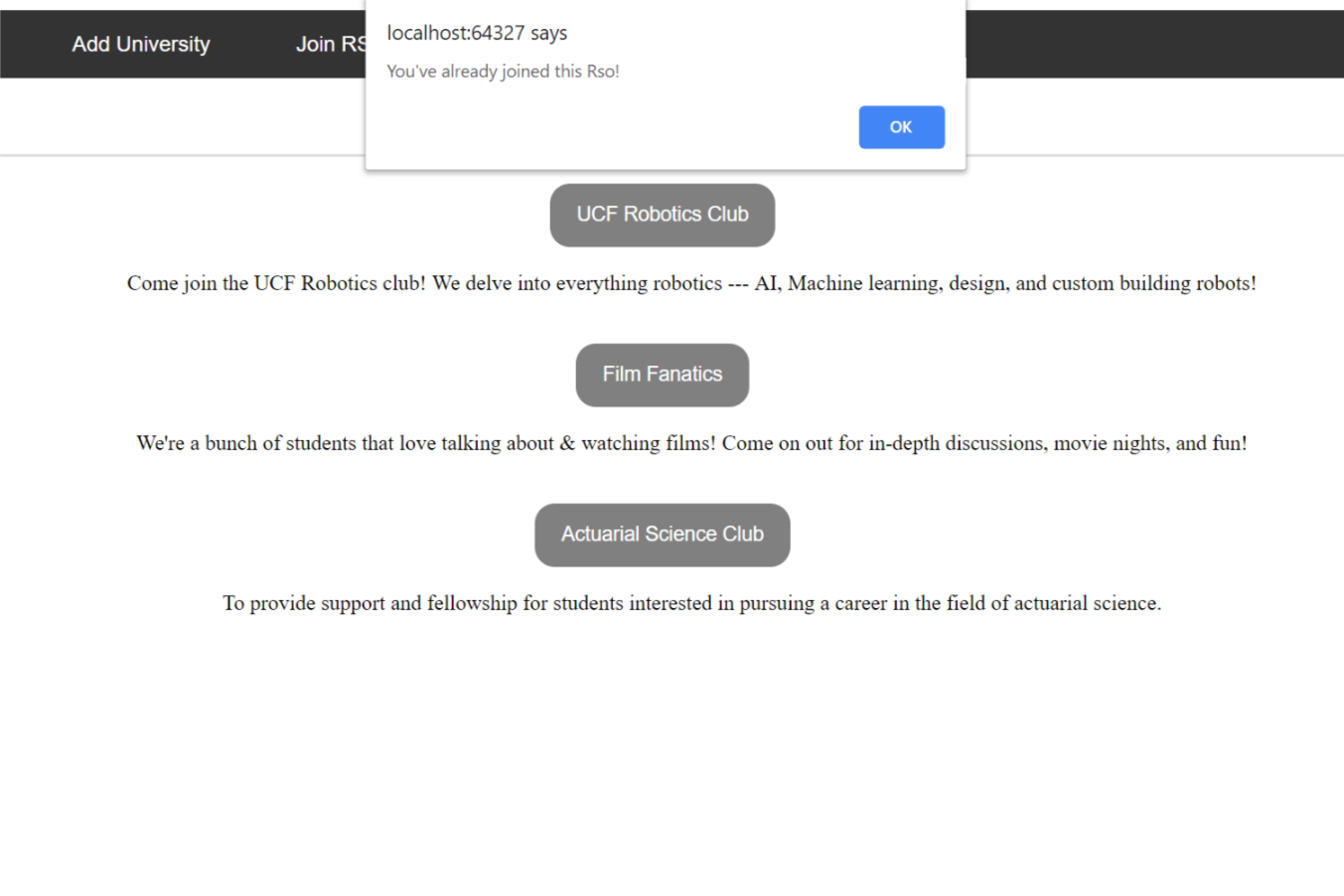
**The User Sign Up Page**

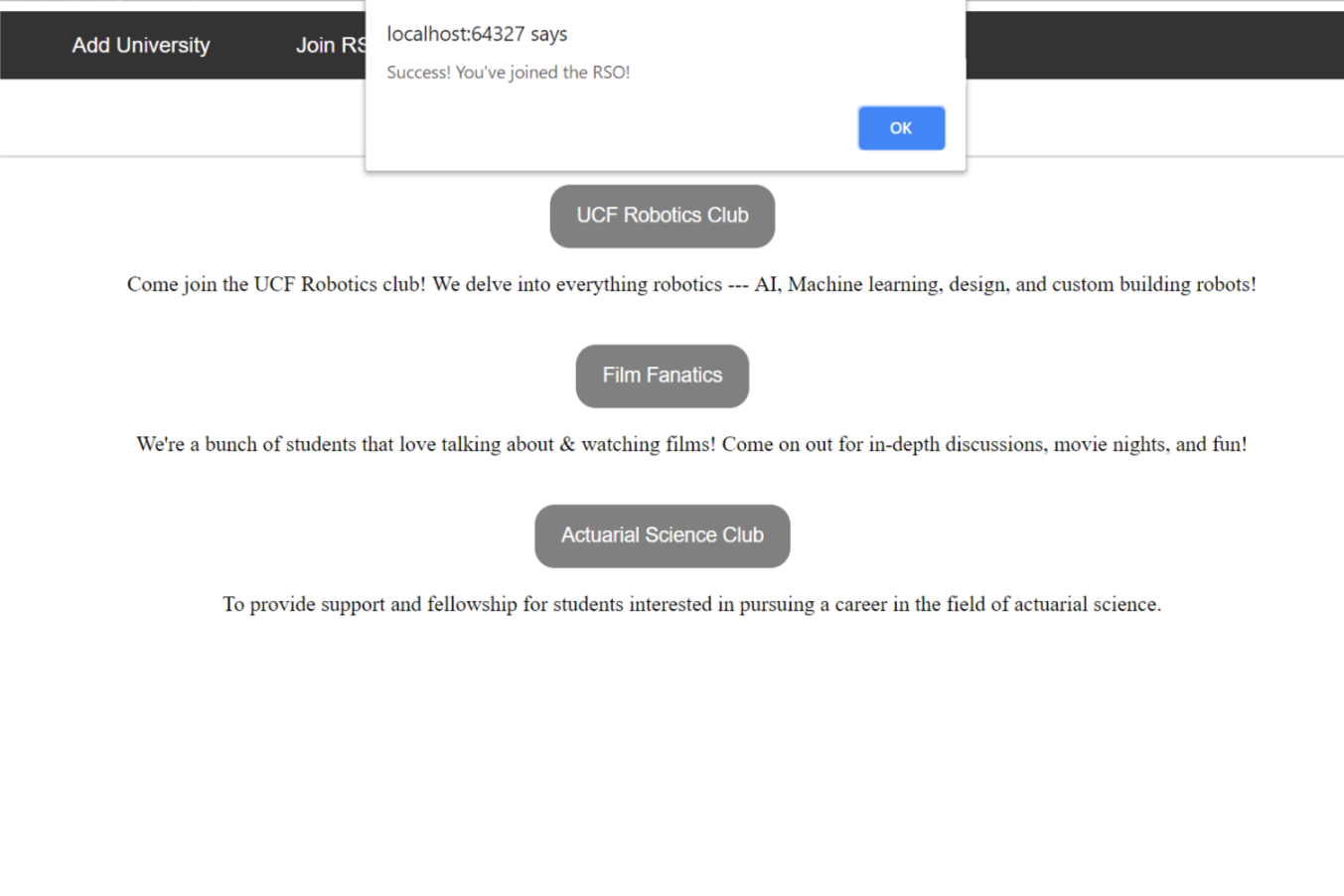
**The Add Events with Geolocation working**

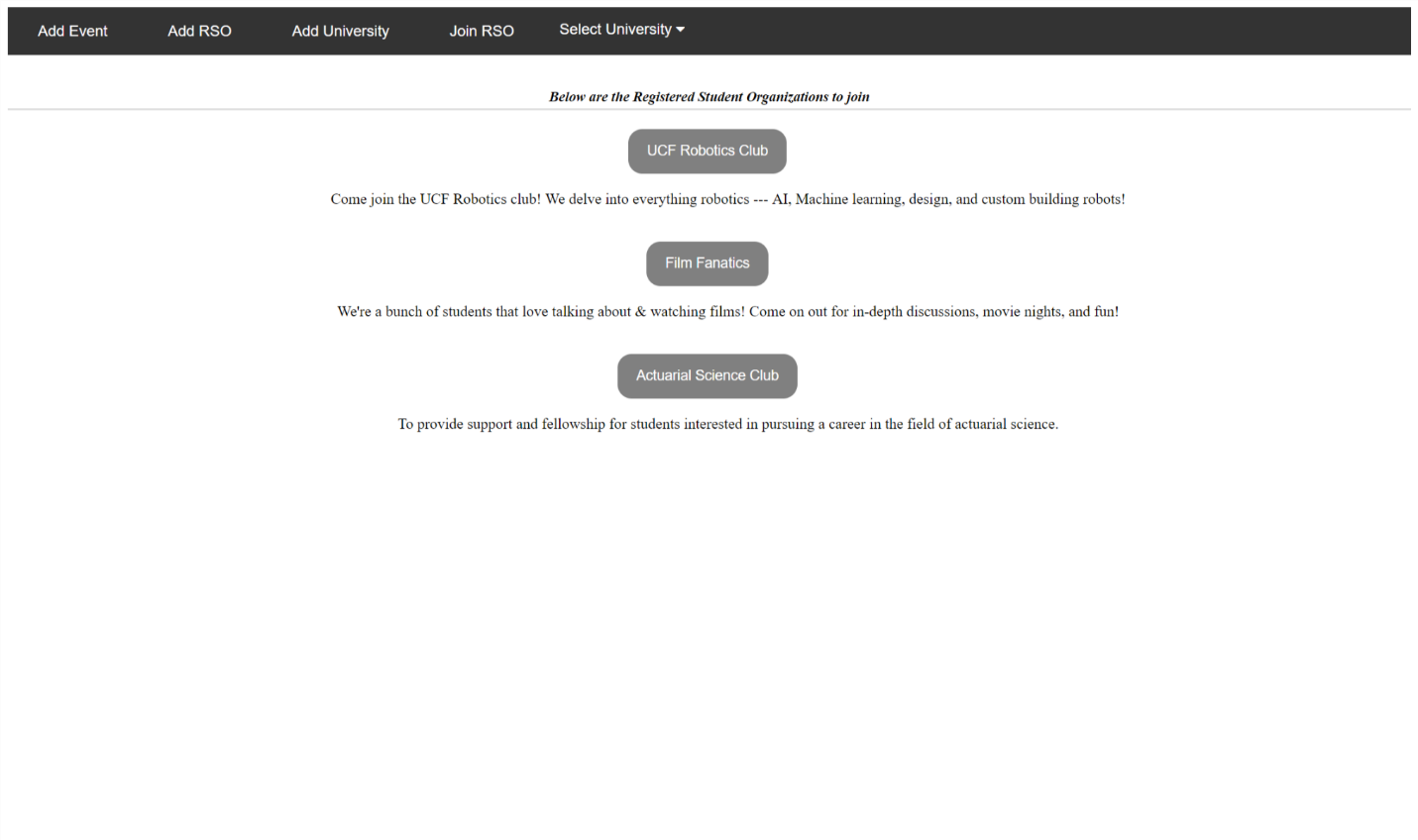
**Add RSO page**

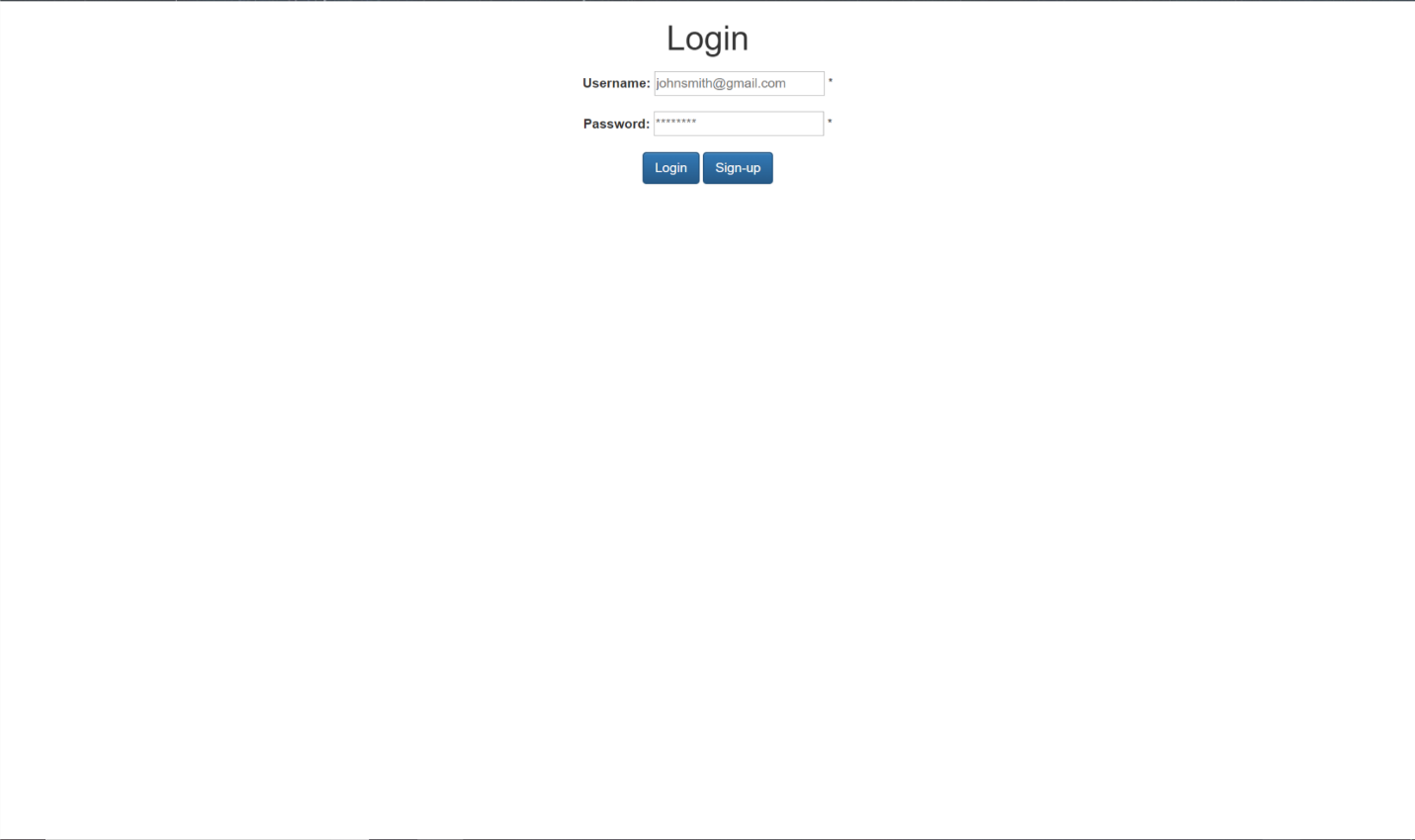
**Add University Page**

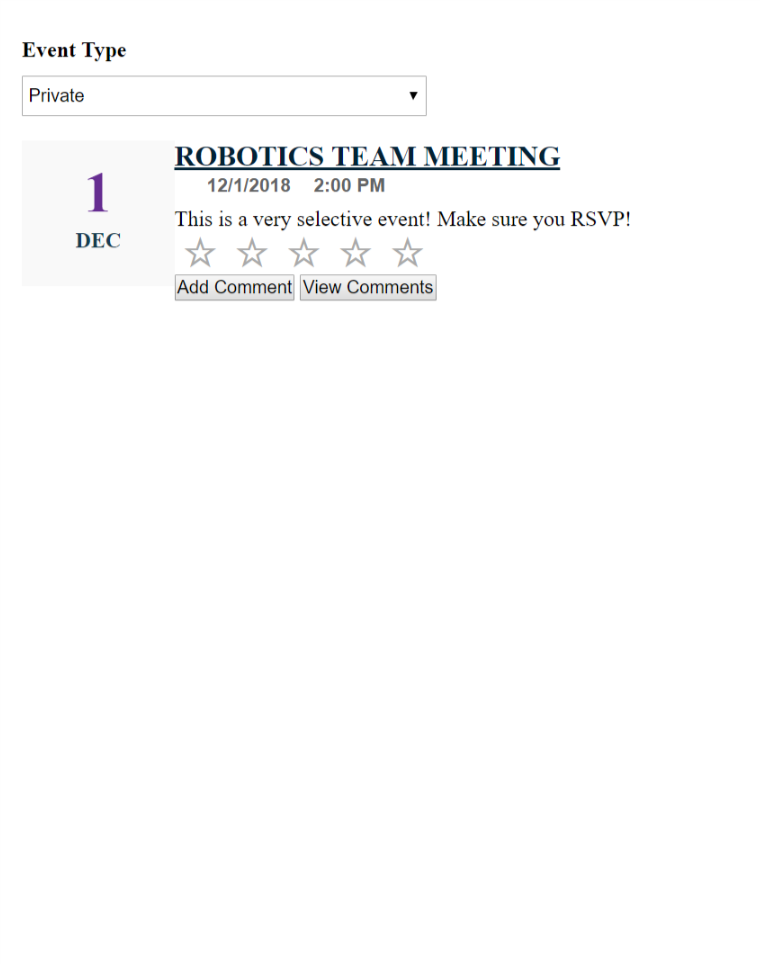
**Admin and Student Landing Page**

**Failure to Join an RSO**

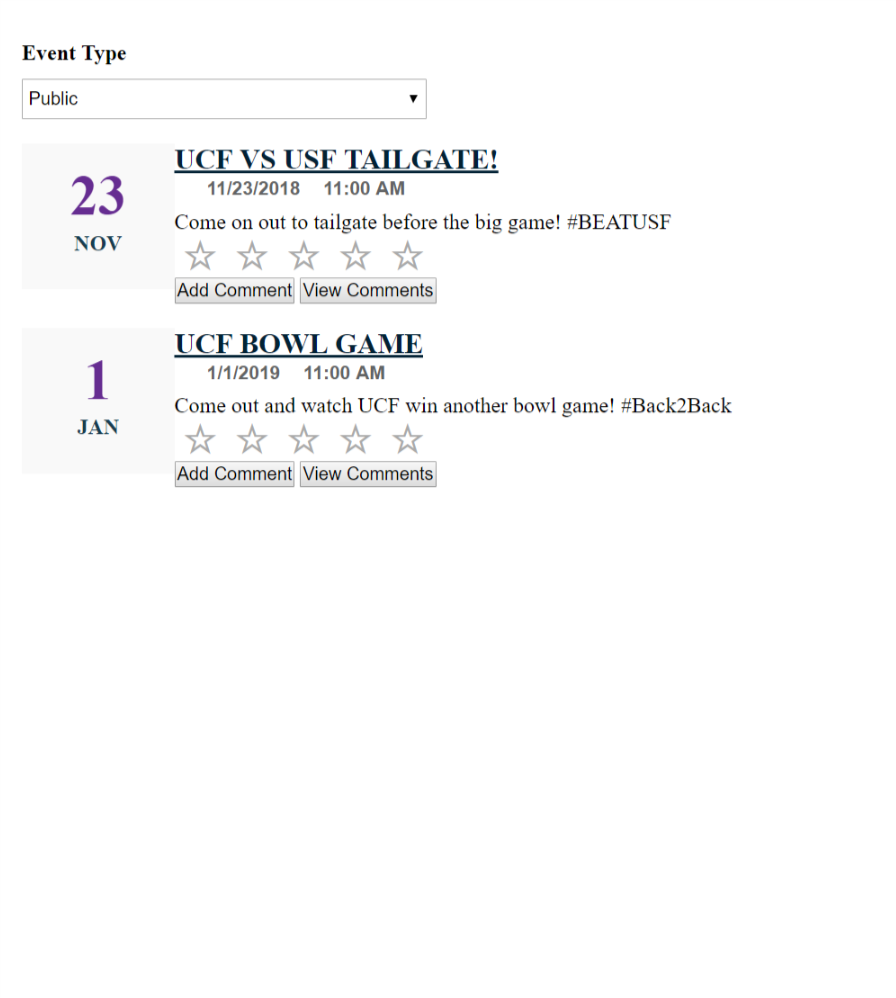
**Successful RSO Join**

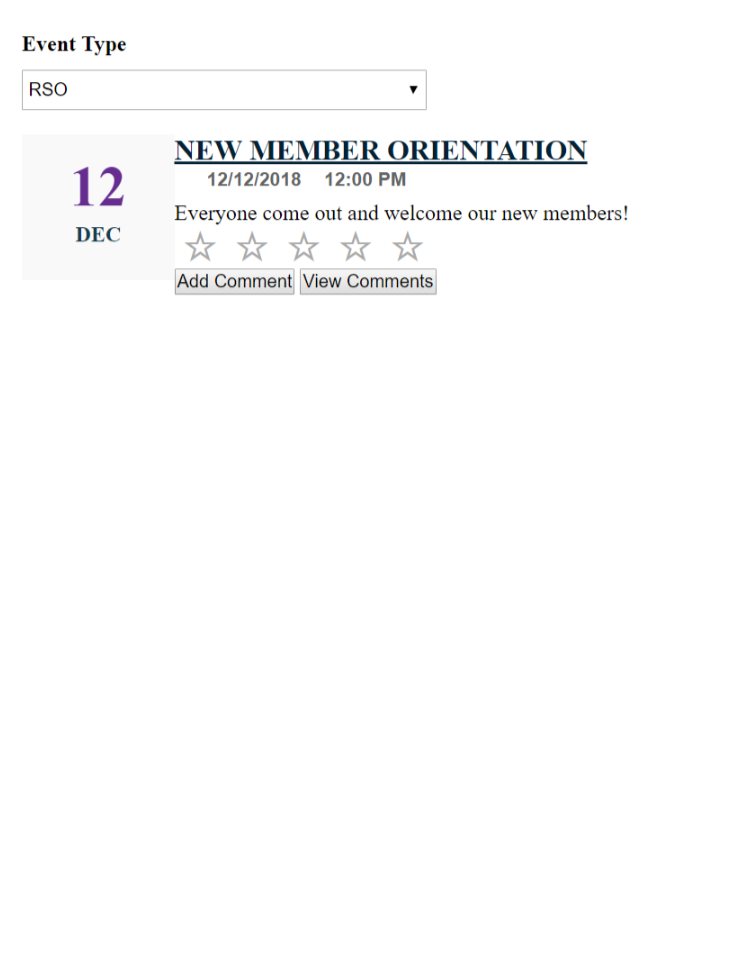
**Join an RSO page**

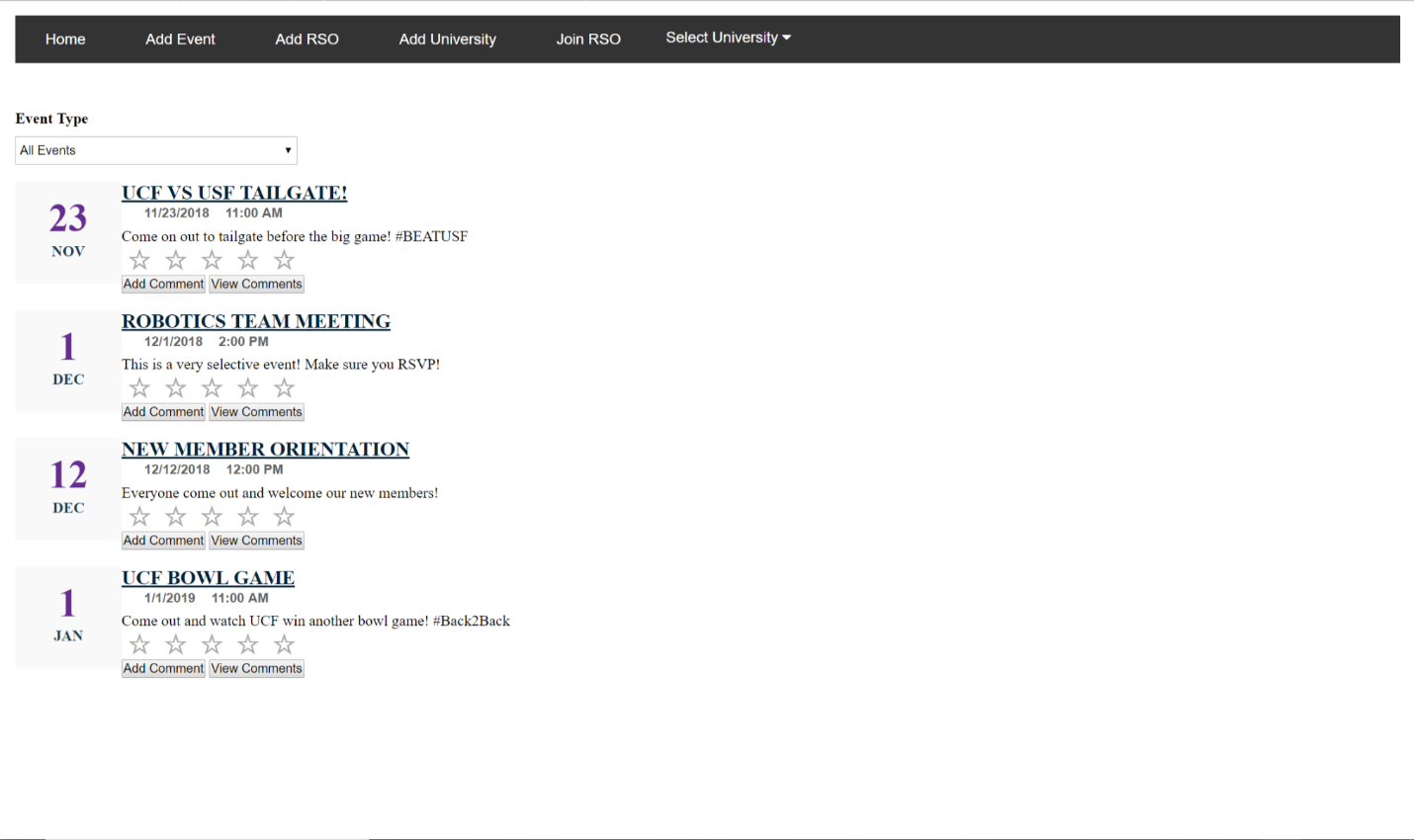
**Login Page**

**Private Events Filter on the Main Events Page**

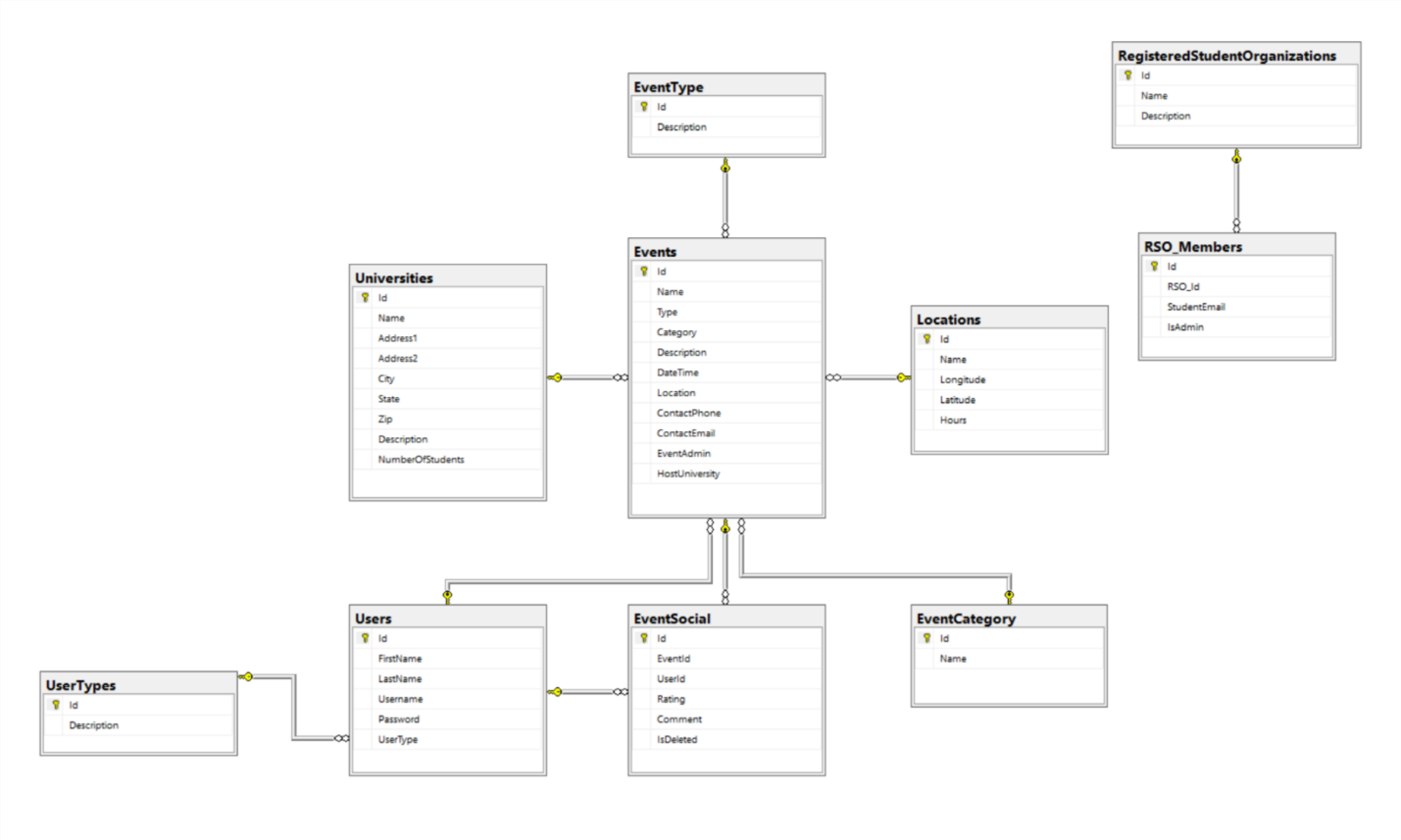
**Public Events Filter on the Main Events Page**

****

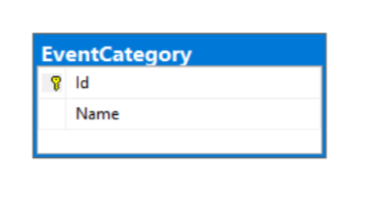
**RSO Events Filter on the Main Events Page**

**Super Admin Landing Page**

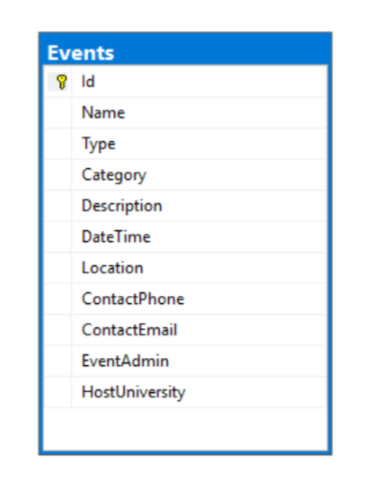
# ER-Model

**Full Database**

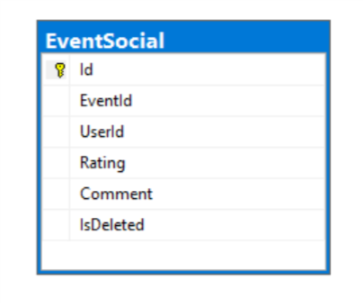
**EventCategory**

****

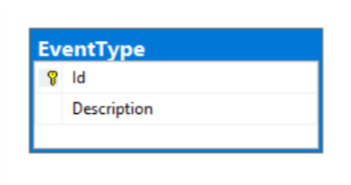
**Events**

****

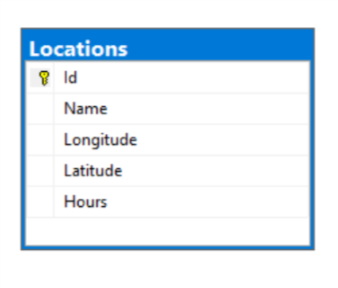
**EventSocial**

****

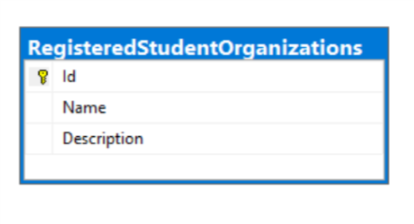
**EventType**



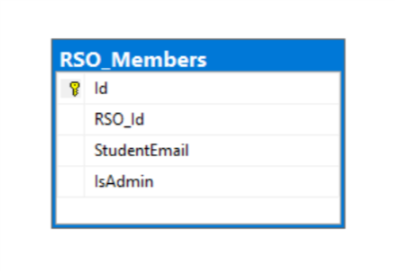
**Locations**

****

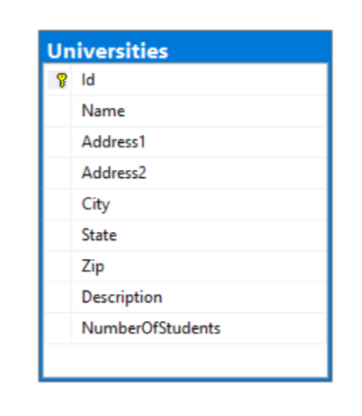
**RegisteredStudentOrganizations**

****

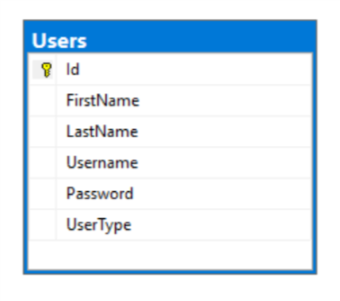
**RSO\_Members**

****

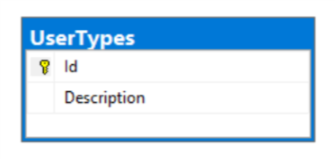
**Universities**



**Users**

****

**UserTypes**

****

# The Relational Data Model

This can be seen in the following script:

Publish/Database/Database\_TablePopulationScript.sql

# Table Populations

The table population scripts can be found in the following directory:

Publish/Database/Database\_TablePopulationScript.sql

This script adheres to all Foreign Key Restraints.

# SQL Examples and Results

**Insert a new RSO**

This uses the CreateRso Stored Procedure, seen below:

CREATE PROCEDURE [dbo].[CreateRso]

@Name nvarchar(128),

@Description nvarchar(128)

AS

INSERT INTO RegisteredStudentOrganizations (Name, [Description])

VALUES (@Name, @Description)

This Stored Procedure can be executed with the following command:

**exec CreateRso ‘RsoName’, ‘SampleDescription’**

Where “RsoName” and “SampleDescription” would be replaced with actual values.

Before firing the above stored procedure, the code first validates that all the members added to the RSO are within the same domain. This happens in the C# controller level. Additionally, the following Stored Procedure is used:

CREATE PROCEDURE [dbo].[AddMemberToRso]

@RsoId int,

@StudentEmail nvarchar(128),

@IsAdmin bit = 0

AS

BEGIN

INSERT INTO RSO\_Members(RSO\_Id, StudentEmail, IsAdmin)

VALUES (@RsoId, @StudentEmail, @IsAdmin)

END

To execute this, issue the following command:

**exec AddMemberToRso ‘x’, ‘example@email.com’, y**

Where “x” is the RsoId (an integer), [example@email.com](mailto:example@email.com) is an example email, and “y” is a flag for if the user is an admin.

These insert into their respective tables successfully. There is a constraint between the RSO\_Members table and the Registered Student Organizations table, so if you try and insert a member into the RSO\_Members table with an invalid RSO Id, an error occurs.

**Insert a new Student into an RSO**

The stored procedure to join an RSO (after the RSO has already been created) is the following:

CREATE PROCEDURE [dbo].[JoinRso]

@UserId int,

@RsoId int

AS

BEGIN

DECLARE @Username NVARCHAR(128)

SET @Username = (SELECT Username from Users where Id = @UserId)

INSERT INTO RSO\_Members(RSO\_Id, StudentEmail)

VALUES (@RsoId, @Username)

END

This stored procedure can be executed using the following command:

**exec JoinRso ‘x’, ‘y’**

Where “x” is the User’s logged in Id and “y” is the Id of the RSO.

Additionally, validation is done in the code to ensure the user can’t join an RSO they’re already in. This is done using the following:

CREATE PROCEDURE [dbo].[CheckIfUserIsInRso]

@UserId int,

@RsoId int

AS

BEGIN

SELECT Username FROM Users u

INNER JOIN

RSO\_Members rso

ON rso.StudentEmail = u.Username

WHERE u.Id = @UserId AND rso.RSO\_Id = @RsoId

END

This stored procedure can be executed using the following command:

**exec JoinRso ‘x’, ‘y’**

Where “x” is the User’s logged in Id and “y” is the Id of the RSO.

**Create a new Event**

The following stored procedure is used to create a new event:

CREATE PROCEDURE [dbo].[SaveEvent]

@Name nvarchar(128),

@Type int,

@Category int,

@Description nvarchar(128),

@Date datetime,

@Location int,

@ContactPhone nvarchar(128),

@ContactEmail nvarchar(128),

@EventAdmin int,

@HostUniversity int

AS

BEGIN

INSERT INTO [Events] (Name, [Type], Category, [Description], [DateTime], [Location], ContactPhone, ContactEmail, EventAdmin, HostUniversity)

VALUES(@Name, @Type, @Category, @Description, @Date, @Location, @ContactPhone, @ContactEmail, @EventAdmin, @HostUniversity)

END

This can be executed using the following command:

**exec SaveEvent ‘Name’, ‘x’, ‘y’, ‘Description’, ’11/24/18 11:00 PM’, ‘z’, ‘Phone’, ‘Email’, ‘i’ ‘j’**

Where “Name” would be replaced with the name of the event, “x” is the type of Event (a foreign key with the EventTypes table), “y” is the Category of the event (a foreign key with EventCategories table), “Description” is a description of the event, “11/24/18 11:00 PM” is the date and time of the event, “z” is the Id of the location of the event (a foreign key with the Locations table), “Phone” would be replaced with the contact’s phone, “Email” would be replaced with the contact’s email, “i" is the id of the admin of the event (a foreign key with the Users table) and “j” would be replaced with the Id of the University the event is hosted at (a foreign key with the University table).

**Insert Comment**

This is handled using the following Stored Procedure

CREATE PROCEDURE [dbo].[AddComment]

@UserId int,

@EventId int,

@Comment nvarchar(1024)

AS

BEGIN

INSERT INTO EventSocial(EventId, UserId, Comment)

VALUES (@EventId, @UserId, @Comment)

END

This can be executed with the following command:

**exec AddComment ‘x’, ‘y’, ‘Comment’**

Where “x” is the User Id (foreign key with the Users table), “y” is the Event Id (foreign key with the Events table), and “Comment” is the comment the user would want to leave.

**Edit Comment**

Editing comments is handled using the following stored procedure:

CREATE PROCEDURE [dbo].[EditComment]

@Id int,

@EditedComment nvarchar(1024)

AS

BEGIN

UPDATE EventSocial

SET Comment = @EditedComment

WHERE Id = @Id

END

This can be executed using the following command

**exec EditComment ‘x’, ‘NewComment’**

Where “x” is the Id of the Comment in the EventSocial table, and “NewComment” is the new comment.

**Delete Comment**

This can be executed with the following Stored Procedure:

CREATE PROCEDURE [dbo].[DeleteComment]

@Id int

AS

BEGIN

UPDATE EventSocial

SET IsDeleted = 1

WHERE Id = @Id

END

This can be executed using the following command:

**exec DeleteComment ‘x’**

Where “x” is the Id of the entry in the EventSocial table. Please note: deleting comments doesn’t delete them from the database, rather, they are flagged as deleted. This is considered good practice and prevents data-loss.

**View Events based on type**

This was accomplished using one stored procedure, as can be seen below:

CREATE PROCEDURE [dbo].[GetEvents]

@Type int

AS

BEGIN

IF(@Type = 0)

BEGIN

SELECT e.Id,

e.Name,

e.[DateTime],

e.[Description],

ISNULL(

(SUM(es.Rating) / COUNT(es.Rating))

, 0)

as EventRating

FROM [Events] e

LEFT JOIN EventSocial es

ON e.Id = es.EventId

GROUP BY e.Id, e.Name, e.[DateTime], e.[Description]

END

ELSE

BEGIN

SELECT e.Id,

e.Name,

e.[DateTime],

e.[Description],

ISNULL(

(SUM(es.Rating) / COUNT(es.Rating))

, 0)

as EventRating

FROM [Events] e

LEFT JOIN EventSocial es

ON e.Id = es.EventId

WHERE e.[Type] = @Type

GROUP BY e.Id, e.Name, e.[DateTime], e.[Description]

END

END

To execute the command, do the following:

**exec GetEvents ‘x’**

Where “x” is the Id of the type of event. If x = 0, then it grabs all events regardless of their type.

Additional Stored Procedures can be seen at the following directory:

Publish/Database/Database\_StoredProcedures.sql

# Constraint Enforcements

**An admin who is not the Admin of the RSO attempts to create an event for that RSO**

We’ve accounted for this in our code, rather than our database. On login we capture the user’s type and give them an access level based on their type. Because of this, non-admins are not given the rights to create the RSO Events.

**An INSERT of a member of an RSO with 4 members: Show the status of the RSO changing to ‘Active.’ A DELETE of a member of an RSO with 5 members: Show the status of the RSO changing to ‘Inactive.’**

This is also handled by our code. We have a private member called “ValidateMembers” which prevents an RSO from being created without the required number of members.

# Software Installation instructions

The website requires a few pre-requisites in order to install on a Windows Machine / Server (which the website was built for). These pre-requirements are as follows:

1. Internet Information and Services is enabled on your Windows PC / Server. For more information on how to do this, please look here: <https://docs.microsoft.com/en-us/previous-versions/ms181052(v=vs.80)>
2. SQL Server and SQL Server Management Studio is installed on your machine.

Once the above pre-requisites are done, you must do the following in order to run the website on your machine.

1. Host the website (located in Publish/Web of this directory) using IIS. If you are unaware how to do this, please follow the guide located here: <https://support.microsoft.com/en-us/help/323972/how-to-set-up-your-first-iis-web-site>
2. Create your database in SSMS.
3. Create the tables in your new database by running the following scripts against the database you created in step 2:
   1. Table Schema Script (located in Publish/Database)
   2. Stored Procedures Script (located in Publish/Database)
   3. Data Population Script (located in Publish/Database). Please note, this is optional to get the website running.
4. Once you have completed steps 1-4, make sure you change the DatabaseConnectionString key located in the Web.Config file of the Published Website (This is located at Publish/Web/Web.config)

Once this is done, you can navigate to the localhost of your machine/server and use the website as intended. In order to access it remotely, you must port-forward the IP Address of your server. More information about this can be found online.

# Advanced Features

The advanced features that were implemented in our program is the geolocation when creating an event to allow for quick location adds based off a current position. This relies on google maps integration. For this feature a specific key was needed to connect to Google Maps by creating a business account and this key is good for a year of use for free. The code was modified from Google Cloud services at: <https://developers.google.com/maps/documentation/javascript/geolocation>.

# Conclusions and Observations

## Database Performance

The database schema we aimed for is closely related to that of the StarSchema design. By indexing on int’s, we’re able to query faster, and by separating into a multitude of different tables we get nano/milliseconds of performance. There is currently no operation on our database that takes longer than .25 seconds.

## Desired Features and Functionality

The login page allows users to login or create an account. The page is then loaded to the home page where users only see events and the ability to see other create pages based on their user level. We were in the process of adding the functionality to share an event to Facebook but had to abandon that due to time constraints.

## Problems Encountered and Lessons Learned

Many problems came up in the creation of this website. Chandler Douglas had database experience, but Jesse Alsing did not have much experience, so it was a large learning curve for Jesse. Neither members were very proficient in front end development, so we kept it simple to make it easier to digest for the developers. The developers would like to work on their front-end web development to make the UI more appealing and easier for the user to navigate. The backend was a unique task, but we are proud of our developments with it but believe that we could better improve on the security and maintenance plan of the database for future projects. Overall, we are extremely proud of our project and have learned a lot about modern day web development.