SLO Dancing Final Report

Senior Project of Ryan Moelter, Audrey Bruscia, and Matthew Smith

Advisor: Dr. Franz Kurfess

June 2016

Introduction

Our project is a website (slodancing.com) that displays all social dance events in the SLO area.

Relevance

Currently there is no centralized place to find information for all dances. Most dance communities have their own websites or events on Facebook, but that information can be difficult to find. Oftentimes, people have to sign up for email lists or walk around town or Cal Poly looking for posters advertising the events. Our website consolidates all of this information into one convenient place.

Intended Users

SLO residents, Cal Poly students, and SLO visitors are the intended users. The project framework is generic and can be expanded to other cities in the future if someone in that city wants to manage it.

Goal and Objectives

The goal was to make the website into one consolidated place where a person can retrieve dance info from all the organizations in the area. A search box at the top allows users to filter the data how they want, including by dance style or by venue. The website also allows trusted users to have accounts and submit or edit dance events for the website, allowing the info on the website to be regulated by trusted organizations.

Progress

We have completed most of the website we envisioned when we set out on this project. We have our list of dance events connected to our backend and database, methods of posting and editing events, accounts with associated organizations and authentication through google, and management of said organizations and accounts. We've fully implemented the frontend, backend, and database for the features we've launched.

Team Structure

We worked as a team, which means that we all ended up getting our hands in everything. During the first quarter, we structured role assignments by the categories of Frontend, Backend, and Database so that we could have a solid understanding of how each section would work and how the components would communicate. During the second quarter, we taught each other what we had learned so that we could make roll assignments based on feature and have a full stack understanding when implementing them. Below are the role descriptions and primary features we worked on.

Ryan Moelter

Ryan is responsible for much of the general architectural design, project management, and initial frontend development. He laid out the general design of the full stack of the project,

Audrey Bruscia

Audrey is responsible for the design of database tables and integration between backend and database. She designed and created the MySQL database, and worked with Matt to integrate it with the backend. The other key feature she worked on was authentication to make sure that only trusted users that are added into the database are able to sign in and access the pages to add or edit events.

Matthew Smith

Matt was responsible for the design of the backend API and object relational models in backend. He designed, created, and deployed the Play Framework application, in addition to working with Audrey to get it integrated with the database and working with Ryan to hammer down the client API. After the project's skeleton was developed, Matt went on to work on features like event management (adding the ability to add/edit events), and did some work with organization management and the ability to invite new users.

Background & Inspiration

Existing Systems

Our project was inspired mostly by a site with the domain www.portlanddancing.com, a website where Portlandians can check the dates and locations of upcoming dance events in Portland. A website like this did not exist in San Luis Obispo, but there were some other existing technologies that had similar functions to our project, such as Facebook Events. However, finding dance events on Facebook can be difficult since its search engine isn't specifically designed for it.

Technology Overview

This project consists of a webapp that SLO residents and visitors will use to find dance events that they would like to attend. We decided on a webapp as opposed to a mobile app because anyone can access a website from almost any device, and because we guessed that most people will be accessing this from a desktop computer. Additionally, the website was designed for scalability in different browser sizes, which makes it mobile friendly. We also wanted to learn web development, since none of us had previous experience in it.

System Design

Overview

***Figure 1**: Diagram of general system architecture*

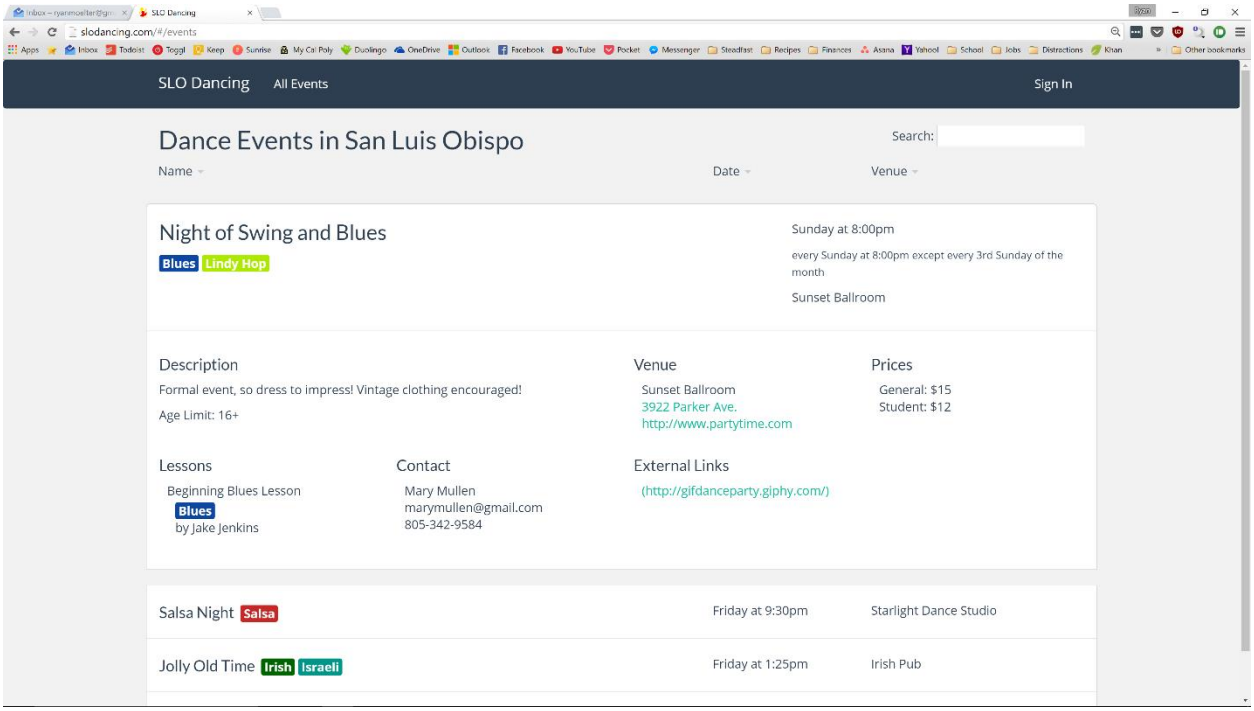

***Figure 2**: Screenshots of finished product for this project, showcasing the responsive design*

Our website contains three parts: the frontend, which contains everything that runs in the client's browser; the backend, which serves the frontend to the client and provides an API to allow the frontend to get data; and the database, which provides long-term storage for the dance events list and contains tables for organiazations and their associated members. The frontend leverages AngularJS and is written in HTML, CSS, and Javascript, like most webapps. The backend is built on Play Framework and is written in Java. The database is a MySQL database.

We spent a large portion of the first quarter deciding on the technologies to use. The frontend uses Angular JS because it provides the developer with many tools to make development easier, such as data binding and animation, and it has the clout of Google behind it. We chose to use the Play framework because it has extensive functionality and is written in Java, which we're all comfortable

with. It is simple to learn, fast, stateless, and non-blocking, and it easily supports the REST API we use to communicate with the frontend. The database uses MySQL due to its read-after-write reliability and its common use in larger applications.

Frontend [RM]

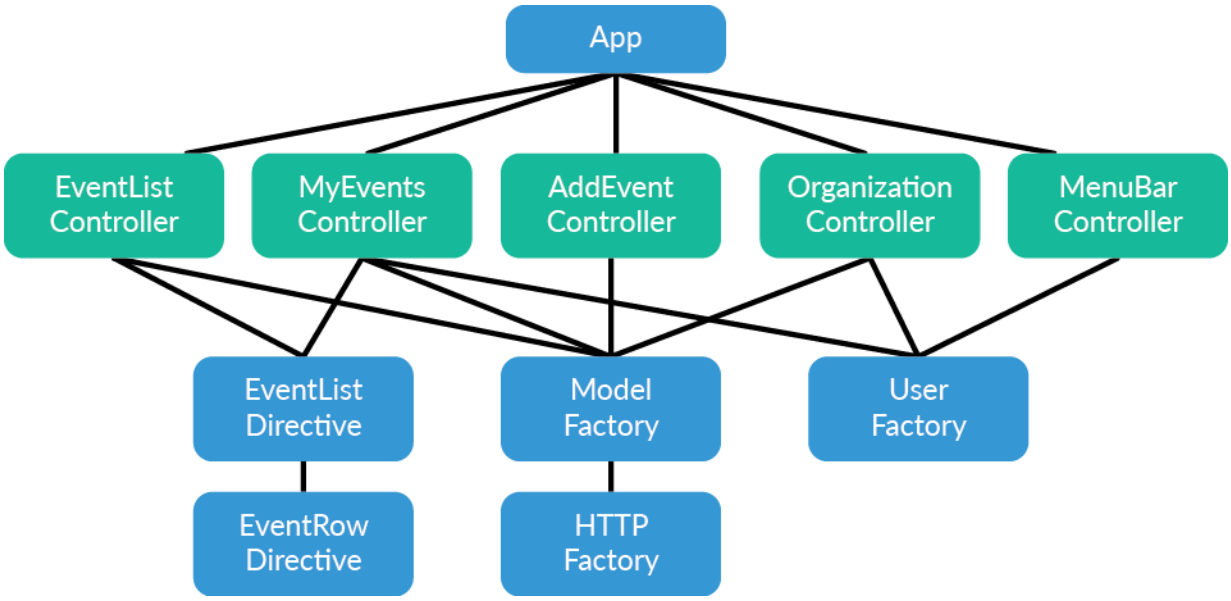

***Figure 3**: Diagram of most of the AngularJS app*

The frontend is a fairly typical AngularJS webapp; it consists of a main app module (Javascript) with controllers (Javascript) bound to specific views (HTML/CSS) which sometimes use directives (reusable Javascript with HTML/CSS templates) and factories (shared Javascript objects). A diagram can be found in **Figure 3** above.

Generally, each controller is bound to one view, and each controller-view pair represents a page on our site. The one exception to this is the navbar with the `signinController`, which is on every page.

Any code that is shared between multiple pages goes into a factory or directive. We put all of our API-accessing logic into factories (`modelFactory` and `httpFactory`), and same with our user authentication logic (`userFactory`). We put reused UI elements into directives, as is standard with AngularJS.

Backend

The backend, written in Java with the Play Framework, was created as a RESTful API to allow GET and POST requests to retrieve data from and manipulate the database that it was connected to. This API included endpoints that allowed for adding and retrieving events, managing users, and authentication (through session in POST data). The Play Framework allowed for an easy template to create endpoints within the API for the frontend to connect to. It also allowed for simple connection to the database and an easy framework for object relational model creating and manipulation. The backend also incorporated Jackson, a Java library allowing for simple parsing and modeling of JSON objects. Jackson annotations on Java objects allowed for translation between

the database models on the backend and the JSON objects that got sent to and received from the frontend.

Database

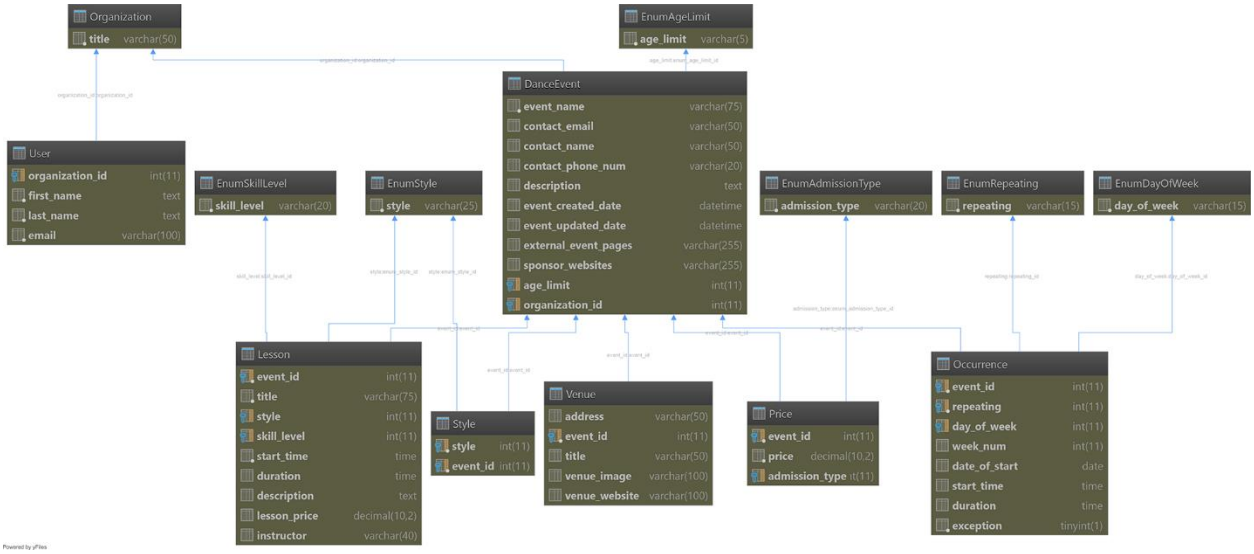

***Figure 4**: Diagram of the database schema*

The complete MySQL database schema is included in **Appendix 1**, but a visual representation can be found in **Figure 4** above. As part of the design, database tables are broken up into distinct representations of data. The table `DanceEvent` is the primary table that holds a single reference to every existing event. All other tables have foreign keys that reference the `DanceEvent.event_id`. This structure allows all of the details associated with an event to be linked back to that event, including the style, venue information, price, occurrence, and lesson information. Additionally, all enum possibilities are in their own tables where the table name preface is `Enum` and the enums are referenced with foreign keys. For example, the table `Style` has a foreign key into the `EnumStyle` table to be able to retrieve the text string of the dance style for that event. We followed the naming convention of capitalizing table names and using lower case with underscores for table column names.

Features

Event List

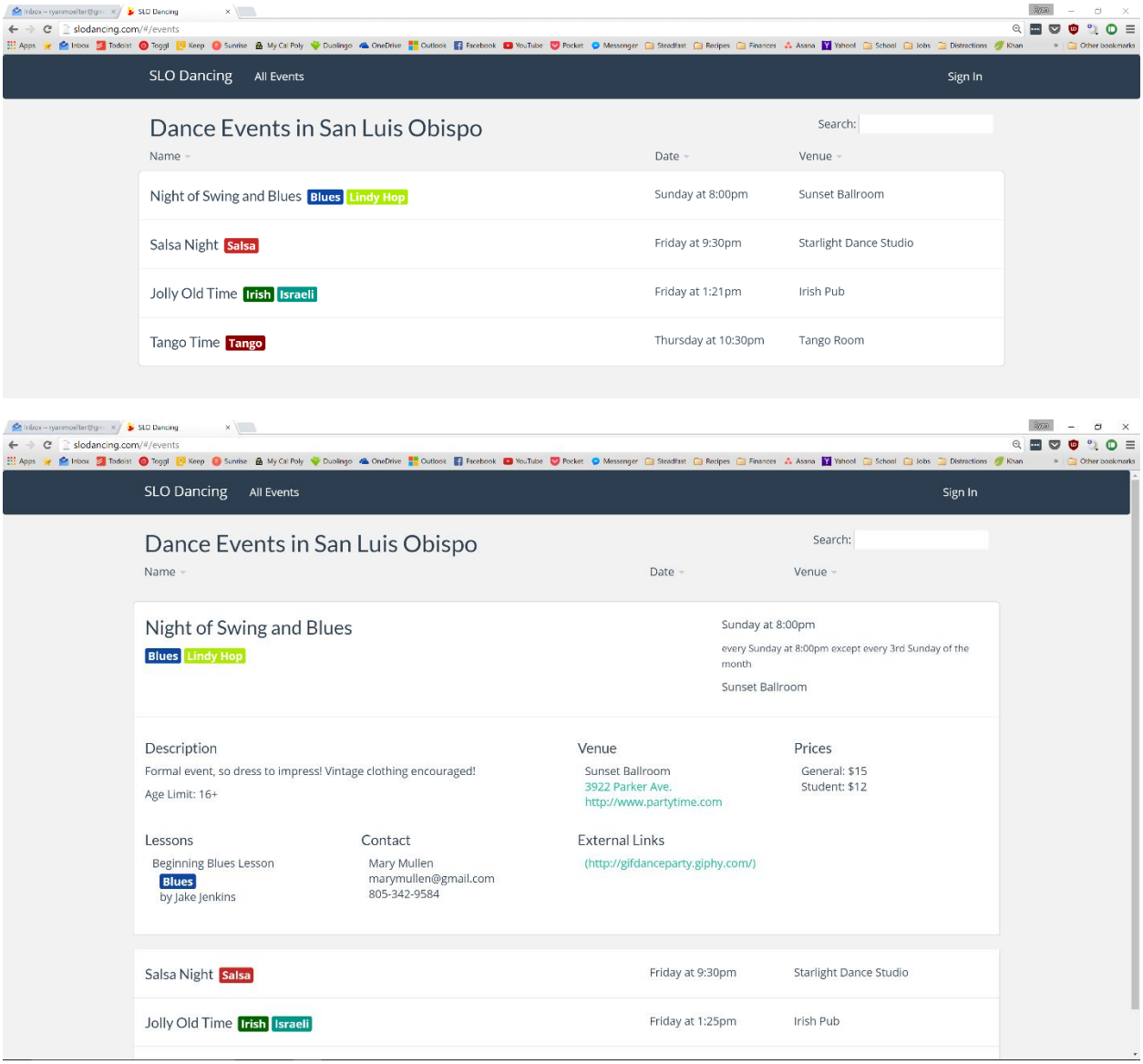

***Figure 5**: Screenshots of the main list of events*

The event list is the most important feature of the SLO Dancing project. This list is the first thing that the user sees when navigating to slodancing.com. The front page list is populated by querying the backend for all dance events that exist and listing them in the order of the next occurring event. The event list was implemented as a directive so that it could be incorporated into multiple views (in this case, the home page and the My Events page). Creating the event list as a directive also allowed us to filter the list depending on which view it appeared. In the case of the My Events page's view, the list was filtered by the organization of the logged-in user.

The event list appears to the user as a list of rows which show details about a specific event. When the user first visits the page, these events appear as a compact view and only show the essential details about the event (title, styles, date of next occurrence, and venue name). The styles are shown in bright colors that make the event visually appealing and make it easier for the user to identify specific events when browsing. When a specific event is clicked, the event is expanded and more details are shown. The expanded characteristics include a description, lesson details, contact information, and prices.

Event Management

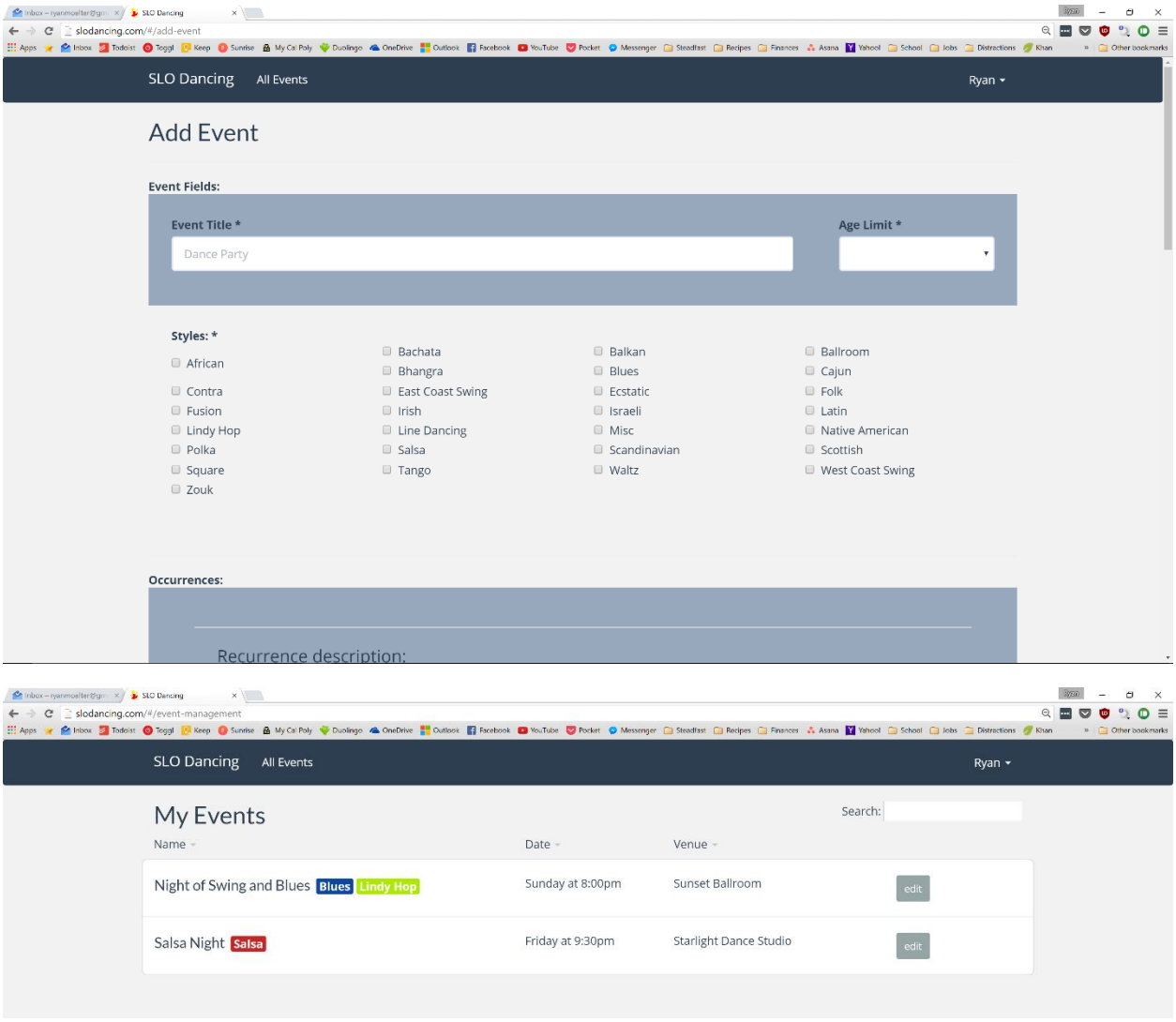

***Figure 6**: Screenshots of the Add Event and My Events pages*

Another key feature of the SLO Dancing website is the ability to create and manage events. When a user is logged in, they have the ability to directly add an event or navigate to the Event Management page. A directive was created to represent a form that reflects an event's details. The reason for creating this directive was so that it could be utilized in both the Add Event and Edit Event page and eliminate the need for duplicate code.

Add

In order to add an event, a user simply selects the "Add Event" option from the drop down menu below their name. This redirects the app to the add event page of the app, which includes the directive representing a large form that allows users to insert details about the event that they wish to create. This page utilizes the formly-angular library for Angular which allows for JavaScript objects to be automatically updated to reflect the state of the form in the Angular UI. Formly also allows for simple input validation (input fields can be forced to match a regex expression or conform to other requirements).

Edit

The ability to edit an event is available in the My Events page, which can also be navigated to from the drop down menu below the user's name. This page utilizes the Event List directive. In this case, a filter is bound to the directive to hide any events that are not owned by the organization that the logged-on user is a part of. The event list also shows an edit button on each event that, when clicked, redirects the user to the Edit Event page for the specific event. This page utilizes the same directive as the Add Event page to display the form, which allows the app to bind the Javascript object that represents this event to the directive. By binding this object to the directive, the form is partially filled out for the user automatically. This page also allows the user to delete the specific event that they are editing. Adding, editing, and deleting of events is done by the frontend POSTing to the corresponding endpoints on the Java Play backend.

Organizations

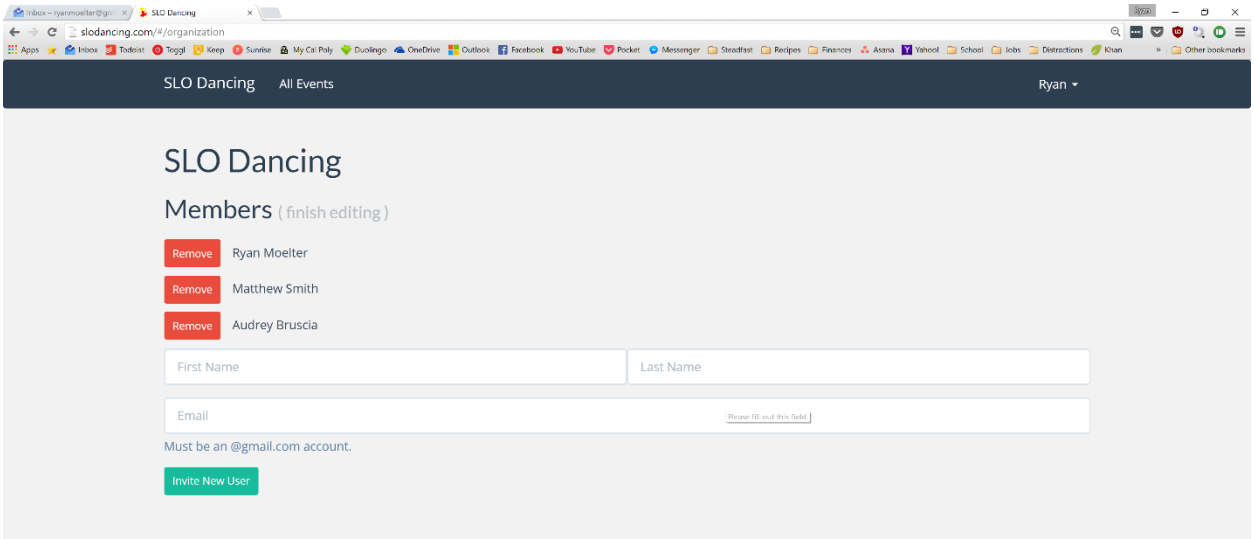

***Figure 7**: Screenshot of organization management page*

General structure

The purpose of Organizations is that multiple individuals can be associated with an organization and can post on behalf of the organization. With this structure, we ensure that only trusted users gain access to adding, editing, and deleting events. Initially, a president or leader of the organization is added to the database with their organization name and their Gmail account. Once one user in an organization has authorized access, they can sign in and navigate to the My Organization page where they will see their organization name. They can then invite other trusted users to be under their organization.

Inviting and Removing

When an authorized user is signed in and is on the My Organization page, they can invite new trusted users by entering the user's name and Gmail. The new information is then submitted to the database User table and the newly added user appears on the list of organization members, meaning that the user now has access to sign in. The list of organization members can also be edited by selecting the edit button and then selecting the remove button next to the user's name.

Authentication and Security

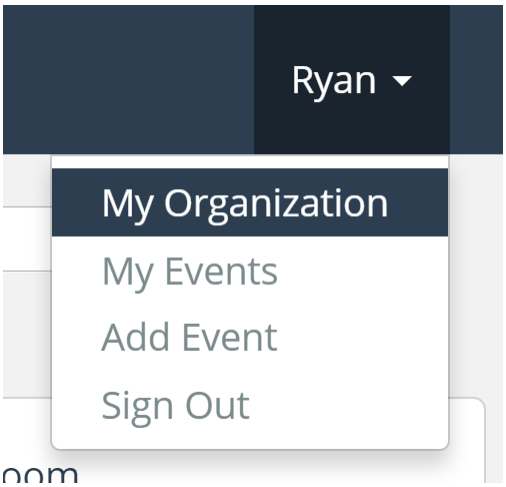

***Figure 8**: Screenshot of the user's menu when signed in*

Authentication is an important part of our website because we want the website to be self-regulating and managed by representatives from trusted organizations. Therefore, it is important that only these people have access to adding, editing, and deleting events so that people don't post irrelevant events that ruin the integrity of the website.

Google API

We decided to take advantage of Google authentication API because many people have Gmail accounts and would be inconvenienced by creating a new account and password just to post events on our website. Another advantage of using the API is that we do not need to worry about handling extremely sensitive data when authenticating users, as we would have to do if we did the authentication process without the help of Google.

Integrating the Google API with Angular proved more of a challenge than we first anticipated because they do not integrate seamlessly. We were unable to just follow the Google sign in documentation on Google's website and had to supplement the process with a wrapper which can be found at https://github.com/canemacchina/angular-google-client. This wrapper then allowed us to respond to the click of the sign in button and to initiate Google's authentication process.

Once a user signed in, Google gave us access to a `GoogleIdToken` which we could use for authentication purposes. Google also provided the user's first name, last name, email, google id, and photo. We took advantage of having access to the user's first name by placing their name on the navbar menu to indicate that the user was signed in.

Structure

To handle the authentication process, we created a sign in page supported by `UserFactory` to store user data. `UserFactory` holds a `User` object that populates user information when someone is signed in. When signed in on the frontend, we send the google id token to the backend for verification. Upon a success result, the user is signed in and their status on the website changes to reflect that and give them menu options to access the other pages of the website and redirects to the home page. Upon unsuccessful sign in, an error message appears and they are redirected to the home page.

Backend and Sessions

Once a user authenticated, we had to implement security to verify that the user was still validly signed in. We protected all endpoints and page routes that should not be publicly accessible. To protect the endpoints, we have `Auth.java` in the backend that receives the google token from the frontend and calls the Google methods to perform the verification. A JSON result is sent back to the frontend indicating success or failure and a message. If successful, a cookie session is started that contains the user's google id token and the organization id that they belong to, based on the database information. For every java method that is called when an endpoint is hit, we retrieve the google token from the session cookies and call the google verification to make sure the user is still valid, otherwise we return. To protect unauthorized users from accessing the frontend pages, the controller for that page calls `modelFactory.checkUserAuth()`, which will reroute to the home page if the user is not logged in, otherwise they can continue.

Future work

New features

We came up with some ideas to improve upon our final product that we either ran out of time to implement or decided were out of scope for this iteration.

Notes

We would like to implement the ability to add notes to events. The purpose of notes would be for when something happens that breaks the normal pattern. For example, if an event that happens every Sunday is canceled one Sunday because of rain, then a note would be able to explain that.

Animations

Adding animations to the project would help to improve the overall user experience. One of the main areas we would like to add animations is when the user clicks the compact event row and it expands to the detail view. Including an animation here would help improve the look and would make it more exciting and interactive.

Logo & Branding

We would like to add a distinct, visually appealing logo to our site, which would reside on the navbar and on the favicon, and would allow us to have a marketable branding that can appear any time we are promoting the website. Having a logo would make the website recognizable and would add more of a sense of legitimacy to the website and its content.

Calendar, Events tracking, & Notifications

Another feature we would like to add is a calendar system. There are many ways of implementing such a system. Some thoughts we had were to be able to add events to a calendar, which additionally could sync with Google calendar. We also thought it would be nice to have a way to subscribe to a recurring or one-time event and get a notification when the event is coming up.

Minor Improvements & Bugs

There were a few minor improvements that we would like to make to our project in the future that didn't quite make it into the final product of this project.

Update EventList UI

One of the improvements that we would like to make to the website is an improvement to the event list GUI. Right now, the expanded view of an event looks a little bit disorganized and messy. The improved format of the events list would ideally look like the following image:

[RM: Image of updated Event Row GUI here]

Add Preview to Add Event Page

Once this format is created, we would also like to eventually have this event row appear at the bottom of the page for Add Event and Edit Event. This would allow the user to see how their event will appear to visitors of the website.

Send Emails with Invites

Another improvement that could be made is verification emails that get sent to invited users. Currently, when a user invites a new user to an organization, the new user's email gets added to the table of authorized users but the new user is not notified by email. The new user simply has to be told by the original user that they have been added so that they can log in.

Fix EditEvent Form Population

A bug that currently exists in our app is that the Edit Event page does not completely fill all forms on the page. By binding the dance event object (that the user wants to edit) to the formly field on the Edit Event page, a lot of the forms are automatically filled out for the user with existing values. This requires some extra processing on the event object since the form object and event object are not identical. However, some of the formly fields do not appear in the GUI for the user to see. For a few of the fields, this could be solved with extra processing of the dance event to make the object match the format expected by formly. For certain fields, such as the date-picker, even after processing the field still does not display on the GUI.

Properly Implement Sessions

We have a few session-related bugs to fix. Currently, the user does not stay logged in when the page refreshes. Additionally, when the user stays logged in for a while, either the session cookies or the google id token expires; once it expires, the javascript still thinks the user is logged in, but the actual authentication process fails.

Dev Environment

Along with minor improvements and bug fixes to the project, the environment that we developed in had room for improvement. Efficient dev environments allow projects to run more efficiently and allow the project to be tested more thoroughly.

Automated testing

Java Play allows for integration with JUnit for unit testing. Testing files can be created in the /test directory of the Play project and the tests can be automatically run with the command:

```
play auto-test slodancing
```

This command automatically launches the browser, runs the tests, and closes the browser. The most popular option for Angular Unit testing is Jasmine. Jasmine allows for structured organization of tests under the `describe()` function, where individual `it()` functions define each individual test. Jasmine can be combined with Karma to run the tests automatically.

Automated deployment / Continuous integration

Continuous Integration for Java Play projects is usually done through Jenkins jobs. Jenkins allows for automation of any process including testing and deployment. Jenkins jobs include a build script that performs any testing/deployment commands that need to be automated. There are also many JetBrains tools for Continuous Integration and automated deployment which would be convenient to utilize since our development was mostly done on JetBrain's IntelliJ IDE.

Local MySQL instances

Currently, when we're developing, we need to establish an SSH tunnel so that our database can run on the server and we can connect to the database while we are working. Having a local database

instance would improve the development process because we would not have the hassle of establishing the SSH tunnel.

Conclusion

Going into this project, none of us had any experience building a website from scratch. Because of this, there was a steep learning curve involved in getting this project started since we needed to learn about all of the aspects that go into creating a website. Along with the software challenges that came with building a website (like building a MySQL database and learning new frameworks like PlayFramework and AngularJS) there were also many technical/logistical hurdles that we had to overcome (like buying a domain, server space, and an SSL certificate). Along with the many challenges that we had to overcome as a team, each individual in the group had different learning experiences because of the division of work and diversity of previous experience.

Personal Reflections

Ryan

Prior to this project, I had never created a website or completed a backend to an app; my primary interest in this project was to get this experience and see if web development is something I want to include in my career. Secondarily, I have been drifting and/or leaping into leadership roles in my group projects, so I wanted to further explore the art of leading a team. Somewhere in the middle of this project, these interests balanced out to be equally motivating my work on this project.

To these extents, this project was a resounding success.

As far as code goes, I was mostly in charge of the general architectural design and frontend development. I got to learn Javascript, or more specifically the AngularJS framework, and all of its oddities. I also got to experience designing the entire stack, though I'd had some minimal exposure to backend architecture from previous attempts to create a backend for another project that just never worked out. I plan to take this experience I wouldn't have gotten otherwise and use it to make more great projects for fun and profit.

Team management was an adventure for this project. I wanted to run the team right, so once I had drifted into the position of project lead I proposed that we use Todoist to track tasks and encouraged the other two to learn and use git extensively. Overall, these measures worked out well, though our team was easy to deal with because we all get along and are motivated to do this project. I'm looking forward to leading more teams whenever I get the chance; I'm beginning to realize that I like leading teams as much as being a part of them.

The actual product of our project definitely still needs work, and we plan to continue working on it over the summer to further improve it. But we've all learned more than we expected to, and we've structured the project in such a way that we can easily continue to collaborate despite parting ways soon.

This has been one of the most influential projects in my college career. I'm looking forward to polishing it enough for it to get popular.

Audrey

This project was an excellent opportunity to work on a product that I have wanted and that many other people have wanted. I have been very involved in the dance community in SLO, so my motivation for undertaking this project was to contribute to the community and make it easier for dancers to find events and become more involved. Prior to this project, I had no experience with web dev or databases, so I learned a tremendous amount this year. I read a lot of articles about database design, writing SQL statements, and SQL injection. I also learned about POST and GET calls and JSON and how the frontend can communicate with the backend. During the process I learned about the general folder structure and components that are required to build a website. It took a while to learn the whole project structure because there are many folders containing files that are all important and that interact with each other. Additionally, I learned about project management and planning. We used slack to communicate effectively, had weekly meetings to stay in touch and on the same page, and used Todoist to track our planned tasks and who was covering each one. It was excellent to receive great feedback from our peers at the Senior Project Expo when we

presented our website with dummy data. Many people said they are looking forward to when the website is up and fully running, and I am excited along with them.

Matt

I wanted to be a part of this project because I had never worked on a project that involved creating a website before. The idea for the product which Audrey pitched sounded like a very useful resource for San Luis Obispo residents that are looking for venues to dance. The fact that this senior project could be something that can continue to be used event after I graduate made the project seem even more interesting. I would love for this project to actually leave a mark on the San Luis Obispo community and have my name be a part of that. The aspect of software that I improved the most at during this project was version control. My use of Git before this project had been fairly minimal, but because of the complex branching system that Ryan formulated for our project, I was forced to learn much more about Git commands and processes. By using a system of feature branching and rebasing, we were able to collaborate much easier on the code base and isolate work. This project also gave me more experience working on a larger system with many components that worked together (frontend, backend, database).

Appendix

1. MySQL Database Events Schema

DanceEvent

```
event_id INT(11)  not null, primary key
event_name VARCHAR(75) not null
organization_id INT(11) not null
contact_email VARCHAR(50)
contact_name VARCHAR(50)
contact_phone_num VARCHAR(20)
description TEXT
event_created_date DATETIME not null
event_updated_date DATETIME
external_event_pages VARCHAR(255)
sponsor_websites VARCHAR(255)
age_limit INT(11) foreign key
```

Style

```
styles_id INT(11) not null, primary key
event_id INT(11) not null, foreign key
style INT(11) not null, foreign key
```

Venue

```
venue_id INT(11) not null, primary key
event_id INT(11) not null, foreign key
address VARCHAR(50) not null
title VARCHAR(50)
venue_image VARCHAR(100)  (URL to static image)
venue_website VARCHAR(100)  (URL)
```

Price

```
price_id INT(11) not null, primary key
event_id INT(11) not null, foreign key
price DECIMAL(10,2) not null (price and admission_type are an
associated pair, ex. General $5, Student $0)
admission_type INT(11) not null, foreign key
```

Occurrence

```
occur_id INT(11) not null, primary key
event_id INT(11) not null, foreign key
repeating INT(11) not null, foreign key
day_of_week INT(11) foreign key
week_num INT(11)
date_of_start DATE  (inclusive)
start_time TIME
duration TIME (stored as hours:minutes:seconds, but seconds won't be
used)
exception TINYINT(1) not null
```

Lesson

```
lesson_id INT(11) not null, primary key
event_id INT(11) not null, foreign key
title VARCHAR(75) not null
style INT(11) foreign key
skill_level INT(11) foreign key
start_time TIME not null
duration TIME (stored as hours:minutes:seconds,  but seconds won't be
used)
description TEXT
lesson_price  DECIMAL(10,2)
instructor VARCHAR(40)
```

Organization

```
organization_id INT(11) not null, primary key
title VARCHAR(50) not null, foreign key
```

User

```
user_id INT(11) not null, primary key
organization_id INT(11) not null, foreign key
first_name TEXT
last_name TEXT
email VARCHAR(110)
```

EnumAdmissionType

```
admission_type_id INT(11) not null, primary key
admission_type VARCHAR(20)  not null options for this: General,
Student, Senior
```

EnumDayOfWeek

```
day_of_week_id INT(11) not null, primary key
day_of_week VARCHAR(15) not null Mondays, Tuesdays, Wednesdays,
Thursdays, Fridays, Saturdays, Sundays
```

EnumSkillLevel

```
skill_level_id INT(11) not null, primary key
skill_level VARCHAR(20) not null   options are Beginner, Novice,
Intermediate, Advanced, Champion
```

EnumRepeating

```
repeating_id INT(11) not null, primary key
repeating VARCHAR(15) not null  options are: Weekly, Monthly, One-time
```

EnumAgeLimit

```
enum_age_limit_id INT(11) not null, primary key
age_limit VARCHAR(5) not null options are: 16+, 18+, 21+, None
```

EnumStyle

```
enum_style_id INT(11) not null, primary key
style VARCHAR(25) not null
```

Style Options

```
Bachata
Balkan
Ballroom
Bhangra
Blues
Cajun
Contra
East Coast Swing
Ecstatic
Folk
Fusion
Irish
Israeli
Lindy Hop
Line Dancing
Misc
Polka
Salsa
Scandinavian
Scottish
Square
Tango
Waltz
West Coast Swing
Zouk
African
Native American
Latin
```