Project Echo Final Report
EECS 214 - Data Structures
June 6th, 2014

Intramural Sports Scheduler

Overview:

The Intramural Sports Program at Northwestern University strives to offer its students, faculty, and staff opportunities to recreate in a fun and enjoyable way. The program provides a chance for participation in sports in an organized, competitive environment while structuring the competition along various levels of skill. In its current state, the registration, scheduling and processing of intramural sports at Northwestern is not handled digitally. Our project aims to provide the Department of Athletics and Recreation, particularly the Director of Intramural Sports, with a tool to facilitate future tournaments at Northwestern.

Given that on any given quarter, there are hundreds of teams playing different sports in different leagues, matching teams is a cumbersome process. Our project's main goal is to provide a registration and scheduling system that would automatically process team registration, and subsequently assign teams in different leagues and sports to the relevant time slots and courts for their games.

Design Process:

At the beginning of the quarter, when we decided we would build a scheduling tool for intramural sports, two of our members met with the Director of Intramural Sports to discuss what features he would like to see in such a tool. We used an agile methodology to manage our project, maintaining a backlog of features and user stories that needed to be built in order for our application to be successful. As a team, we maintained this to-do list after every meeting, and we discussed who would tackle each individual task according to our strengths and weaknesses.

Before we started building, we began by brainstorming potential classes and evaluated how these classes would need to fit the requirements. When we started building the application, each team member came up with a potential schema and architecture for the application. The team met to discuss the strengths and weaknesses of each schema, and decided to implement the architecture shown in appendix I. We utilized a GitHub repository[1] to manage version control, and to allow different members in our team to work in parallel on various features of the system.

We met on Fridays to discuss our individual progress on tasks and then generated a list of todos and determined their priority. We then coordinated a meeting for the weekend during which we would program as a team for a few hours and then divide up tasks to complete before our next friday meeting. Throughout the week, we would email the group about our progress, issues and explanations of how our additions worked.

Since our application relies heavily on CRUD actions (create, read, update, delete), we decided that having a database would help streamline our workflow. Although we had initially considered using a PHP wrapper to access a MySQL database, and worked with it for a couple of weeks, we eventually decided to instead use Parse.com as a backend. This allowed us to maintain the entirety of our application logic written in Javascript, by passing JSON objects from Parse to the client side scripts. More information about our database implementation can be found in appendix II.

With an overarching structure defined and a database running, the implementation of the various classes was relatively straightforward, and the seed_data.js file was written to create mock objects for testing. These objects are meant to represent the data that would be generated if users were actually utilizing the site, allowing us to test that our various methods did what we expected them to do.

Lastly, our documentation was generated using JSDoc, which is equivalent to the JavaDoc generated by Netbeans, but aimed at web projects built on javascript.

Features:

Since the user base of our tool is mostly comprised of non-technical students and administrators in the Northwestern Fitness and Recreation Department, we decided that including a graphical user interface (GUI) was imperative. We utilized Twitter Bootstrap and

[1] See https://github.com/selberts/ProjectEcho/tree/gh-pages

jQuery to build our user interface, and hosted the front-end on github.io[2]. We also leveraged the Google Calendar API to show the scheduled games in a format familiar to the users, as seen in the calendar page of our website.

As our application set out to solve a problem with two sides, administrators and students, we divided most of the application logic into a page that would serve each group of users. The administration logic is contained in the "admin" page, while the student logic is contained in the "register" page.

The administration page allows the system administrator to create new timeslots, and run the scheduling function once those timeslots have been filled with student groups. The scheduling itself works under a FIFO (first in, first out) method, where the first teams who submit their preferences get assigned timeslots first.This page requires authentication, since only the administrator should create timeslots or schedule games. The students on the other hand can only submit their time preferences from the available timeslots. Once both groups have done their part, the games for the quarter are scheduled and shown in the "calendar" page.

We created a Google account, northwesternintramurals@gmail.com, to store the calendar representations of all league schedules. To push the schedule output by our application to the calendar, an administrator must simply log into this account, go to the Admin Page on our website, select the sport to be scheduled, then hit the “Assign Teams” button. Since the application authorizes administrators with the OAuth 2.0 protocol via the Google APIs client library, we ensure that only admins can edit the calendar, but any user can see the result. Users then get access to all the features of public Google Calendars, like adding the league calendar to their personal calendars, setting up alerts for game times, or viewing the location of the game in Google Maps.

Future Plans

Despite being able to complete the main goal of our project, there is still a significant amount of room for improvement. On the administration page, the there is also an issue to be addressed with authentication. If a person were to simply read the javascript, they would be able to access the database and read the password. A final product would require better data security

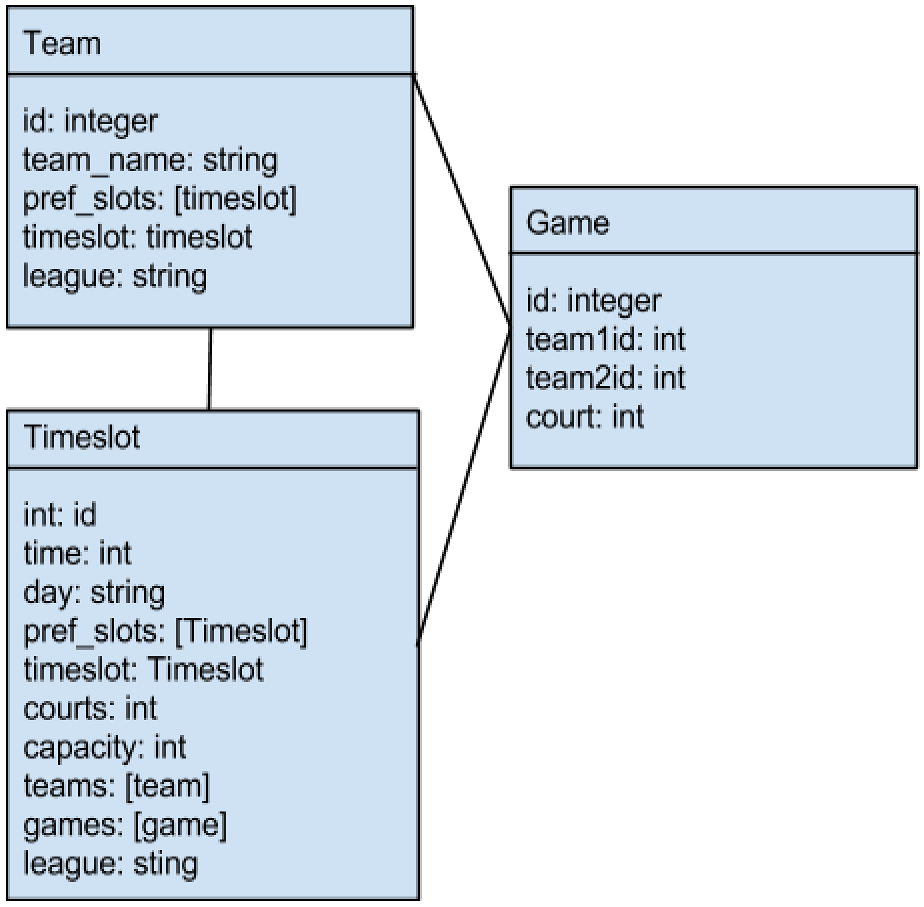
[2] See http://selberts.github.io/ProjectEcho/

and true authentication. Next, there should be a feature allowing the admin to remove timeslots already on the database. This ability should be removed after registration has opened, but the admin should still be able to add timeslots from the GUI (currently, this is only possible directly from Parse). Furthermore, the administrator should also be able to create new leagues, an important feature we are still lacking.

The next step would be a safety evaluation of our project. Currently, there are several issues in regards to safety and authentication that would need to be fixed in a final product. First, registration is completely unsecured. A future implementation would require students to login with their Net-IDs and passwords. Furthermore, this system would also need to know if a team has paid its deposit before being allowed to register. An unnecessary, but useful feature would be a payment portal to allow this deposit to be paid electronically.

Appendix I: Schema Diagram

The following diagram represents the classes used in our project as defined in classes.js:



Appendix II: Database implementation: Parse

1. Add keys to application

In the <head> of the page, the source of Parse need to be imported

```
<script src="//www.parsecdn.com/js/parse-1.2.18.min.js"></script>
```

Then add initialize the Parse connection by adding application ID and JavaScript key

```
Parse.initialize("r3WndIFb85R0lx1qhchN4nquvAQVeKVrkA3TBnpI","Wui7puCTZpnTmA5ZLvJmlj5R044v
AyDerOBXhYzq");
```

2. Select the table in the database

```
// Select the table Teamdata in the database

var Teamdata = Parse.Object.extend("Teamdata");
```

3. Push data into the selected table

```
// Create a new tuple of the selected table

var teamdata = new Teamdata();

teamdata.save(

{team_name:"echo",pref_days:pref_days,pref_times:pref_times}

).then(function(object) {

//succeed in saving, do whatever you want with the saved data object

});
```

4. Pull the data out with a query

```
// Prepare a query for a certain table

var query = new Parse.Query(Teamdata);

// Constrain the query by need

// select the tuples whose type equals team

query.equalTo("type", "team");

// make the table to be FIFO
```

```
query.ascending("createdAt");

query.find().then(function(results) {

//the results set can only be accessed in this function!!!

//results is an array consisting of tuples

// get the first tuple

var object = results[0];

// get some attribute of the tuple

var team_name = object.get["team_name"];

});
```

Advantages and Disadvantages of Parse

Advantages:

The advantages of using Parse are obvious. It is small, convenient and fast. The most important thing is we do not need to bother building a server which makes this project actually work. What's more, with Parse, every team member can test his/her part of the project with the data supported in Parse Data Browser. Otherwise, we definitely need to build an external environment and copy the database source from computer to computer, as we did at first when using PHP and MySQL.

Disadvantages:

The disadvantages of Parse are obvious too, in that comparing to MySQL, SQL Server or Microsoft Access, Parse is too small to handle some middle-size data. The documentation says using a query to count a few thousand of tuples may cause a timeout error. However, considering the needs of our project, a micro-database is enough because we will not have too many teams. Also, the query complexity is limited. We cannot execute some complex queries in Parse. But again, it is still enough to solve the problem we have, which is the FIFO requirement.