WebApp Group 34 Milestone Report

Han, Qiao   Chabierski, Piotr   Smith, Bradley   Cingillioglu, Nuri

May 26, 2015

1 Group Structure and Work Division

We divided our group responsibilities based on our individual strengths. Han has done extensive development work on both server and client side applications in past internships and part time jobs and is hence the most suitable candidate for the role of group leader. The group leader is responsible for guiding the choices of technology together with ensuring a schedule for the timely completion of the project. In addition, he is also responsible for managing the team during evaluations of the project requirements and the suitability of different technology stacks in fulfilling our design goals.

More specifically, our team members are initially in charge of working on following areas.

1. Client-side presentation - Nuri

2. Client-side interaction - Nuri / Han

3. Server-side processing - Bradley / Piotr / Han

4. Database set-up - Bradley / Piotr

As we progress, we believe we will work on different parts of the project from this initial assignment so that everyone can get a holistic understanding of the full stack web application design.

2 Choice of Implementation Languages

Although the frontend has to be written in HTML, CSS, and Javascript, there exists a wide variety of web development frameworks that we have considered, ranging from full blown MVC models such as AngularJS, MeteorJS, Backbone.js, to minimalistic libraries such as jQuery, Modernizr, HTML5 Boilerplate. As the team is relatively new to web development, we decided to use frameworks that more closely represent the fundamental DOM layout to enhance our understanding. This means that HTML5 Boilerplate is a good choice because it modernises the standard HTML technologies without adding extra layers of abstractions. In addition, we have chosen to use the Bootstrap framework to enhance the visual presentation of our web app. Lastly, we have decided to separate our data models from the views by making AJAX calls to our backend using a RESTful API that retrieves JSON objects for updating the views. Using JSON as the data format is not only easy to parse and understand but also allows for rich data structures like nested objects and arrays.

We have picked Java as our backend implementation language, after considering the strengths and weaknesses of Python, node.js, and PHP in detail. Although it may increase the amount of time that writing our App will take, we believe this to be a good trade-off. Java code is much more habitable and allows for easier extension. We think it will keep our options open going forward as it is very portable and platform independent. Java also provides several frameworks we can use to unit test our code such as JUnit and Mockito along with many useful libraries to provide much of the functionality that we will need. We set up our web container using Apache Tomcat as it allows us to work in pure Java.

Lastly for the database we chose to use PostgreSQL due to convenience as CSG had already given us an already configured database to use.

3 Description of our App

Our project is a multi-user calendar App which allows people to, create/subscribe to calendars, create/join events and obtain feedback on who has signed up to your events. The App is targeted at volunteers for college events but can be easily extended for more general purposes.

It has three main entity types and several relations between them, these are the users, the calendars and the events, see appendix A for more details. The idea came from a specific problem that we found occurring very often, people using emails to organise events and work out specifically who can attend or to get a general idea of how many people to expect. This can lead to very inefficient communication and cause a verity of problems to occur. Such as events occurring so often that the people who are attending (volunteering) start to pay less attention to them, or if an event if so popular that it ends up being over subscribed and the organisers have to send a huge number of emails back to the people who didn't make it in time.

We are designing and making our App while having these problems in mind and aim to come up with a system which could eliminate or ease some of them.

4 User Interactions

The basic idea is that there will be two types of users (per calendar), the administrators and the users (subscribers). They will both interact with the calendars in different ways and have various typical uses. Note that any user can create a calendar or join others so these roles are not absolute.

For the subscribers of calendars that need to be able to see events in a clear manner and easily be able to gauge their interest about the event. They also need the ability to sign up for the event if they wish to attend (depending on the event this might be a strict attendance or a flexible one). In addition to this they should be able to change between the calendars they are subscribed to and create a view with the calendars that interest them at any time. Finally they should be able to create calendars and gain administrator privilege on them.

The administrations of a calendar need extra abilities in addition to those of a subscriber. They will be able to create, modify details of and delete any event on their calendars along with the ability to see the number of people who have signed up and their identities. In addition they can delete whole calendars and invite people to subscribe via an invite code which they can give out to anyone who may be interested. We are also looking into other ways that users may be invited to calendars.

Lastly everyone will also be able to interact by registering and logging in to our App, see appendix B for user flow diagrams.
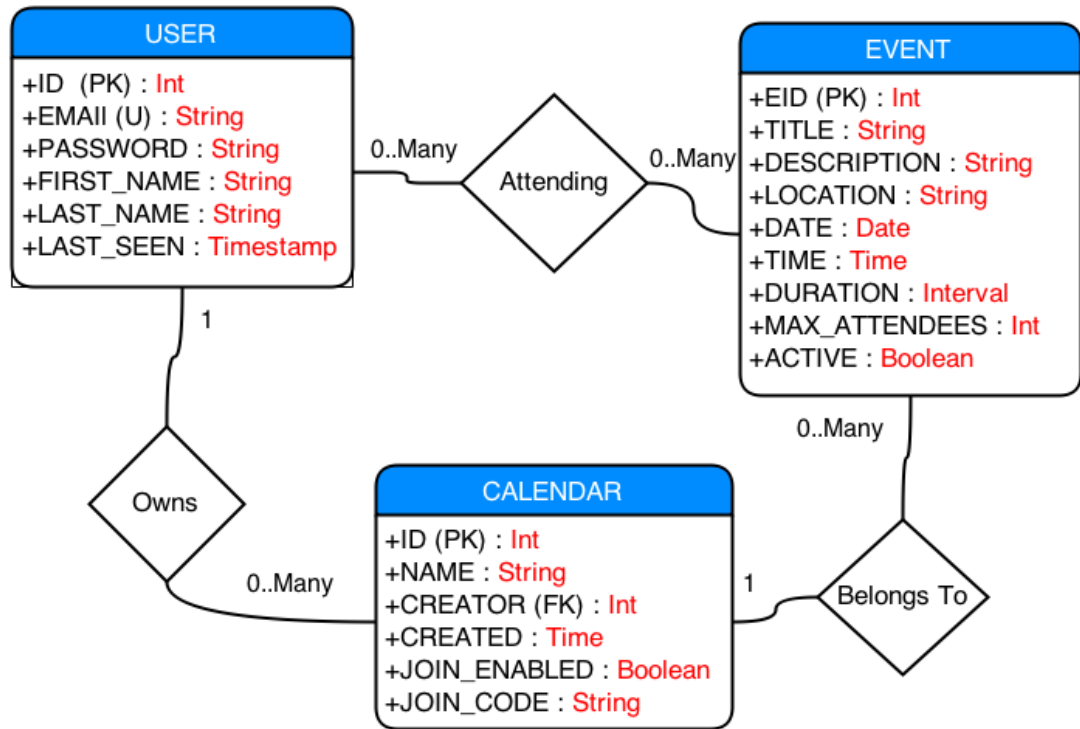
A   System Entities and Relations

Figure 1: Entity Relation diagram with possible database layout

PK, FK and U stand for Primary Key, Foreign Key and Unique respectively. PK implies U.

There is a relationship missing from this diagram, this is the subscriber relationship. Each of the relationships will be represented in the database as a table with ID to ID mappings with the 2 participants. The cardinality constraints can also be very easily added into the database to ensure that the data remains consistent. For example the **belongs to** relationship will be realised with a table with one column being the CID (Calendar Id) and the other being the EID (Event Id). The (CID,EID) pair will be the primary key.
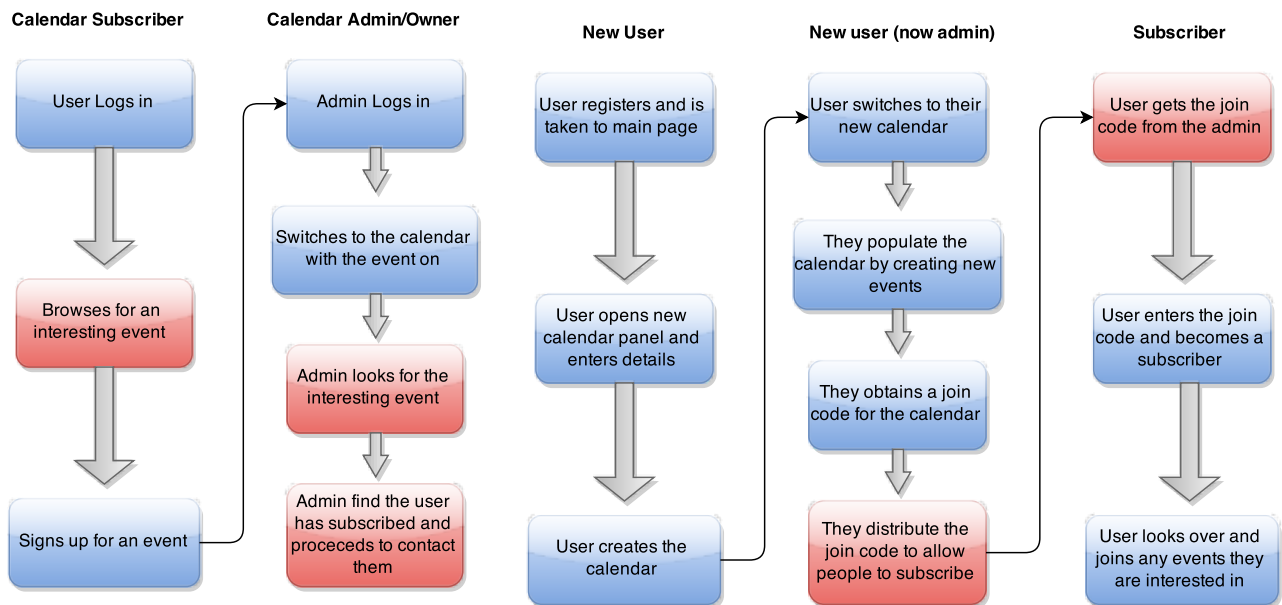
B   User Interaction Diagrams

Calendar Subscriber   **Calendar Admin/Owner**   **New User**   **New user (now admin)**   **Subscriber**

Figure 2: Use flow of a user signing up

Figure 3: Use flow of a new user creating a calendar

These diagrams show two typical use situations it which the user would interact with our WebApp. The blue boxes are explicit interactions with our App, where as the red boxes are events which require no explicit (but most like some) interaction.