The Community Service!

Helping you, help everyone!

David Douberley
Austin Macdade
Henry Salazar

Table of Contents

Executive Summary

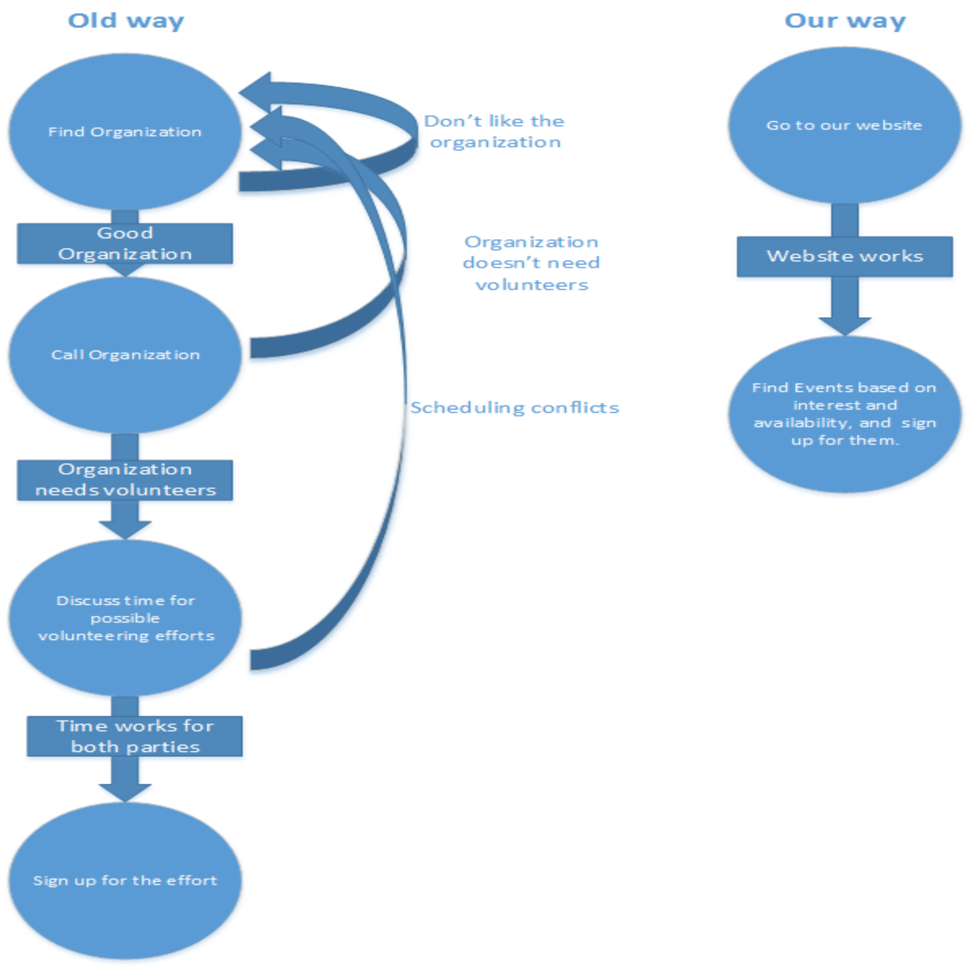
General Idea

The core idea was to create a website / mobile app that tells volunteers what charities and organizations need their help and when. For example, if a user likes animals, he can browse by category to see what organizations are looking for volunteers to help animals. Or, he may want to browse by time. Let us say he has no plans for Saturday. He can look to see if volunteers are needed this Saturday in his town. These examples could be combined for even more specific scenarios.
Also, a large portion of the functionality is tracking user volunteering and motivating users to volunteer more. For motivating we included leaderboards for hours worked, and for quality of work done, for users and groups, so that everyone can see the work a user is doing. At the end of the day our goal was to make volunteering easier for everyone and to encourage more people to do it.

Reason for Project

Social networking at this point has been primarily used among friends to send jokes and post funny pictures. But what if the power of social networking could be used to make a real difference? Everyone likes the idea of volunteering, but most do not really know where to start. Volunteering is not always as easy as making a phone call. In real life, you have to start by finding an organization that matters to you. Next you have to call the organization to find out if they even need volunteers, find out what day and time, and then see if it fits in your schedule. Repeat the process and you begin to see the problem. There is a ton of overhead just to help the community. Things should be easier to help!

Face it, most people are lazy. If volunteering were easier, more people would do it! The best way to make it easier is to unify the system, and make it easy for both parties to connect with each other.


Old way
Find Organization
Good Organization
Call Organization
Organization needs volunteers
Discuss time for possible volunteering efforts
Time works for both parties
Sign up for the effort
Don't like the organization
Organization doesn't need volunteers
Scheduling conflicts
Our way
Go to our website
Website works
Find Events based on interest and availability, and sign up for them.

Individual Motivations

David's Motivation

The most important motivation for choosing the project for me was for having something to show employers. I believe if this service is done correctly it could be a great tool for organizations, groups and users, which means it could be well received by the public and get used regularly. I think employers will really like the idea behind this project and the implementation. In general I am excited to have this on my resume.

Even though, I have done a database backed application before, this was a really great learning experience too. I had never done the entire backend of a project like this before, and I would also have never done a web service that is this intricate before. Learning new web technologies, which is something I am already fascinated in, was fun and makes me a better candidate for future jobs in the field.

Austin's Motivation

This type of project is something that I see myself having to deal with on a regular basis in the future. This project deals mostly with technologies that I have at least had to tinker with on existing projects, but few that I have had to implement from nothing. This project presented the opportunity to learn a great deal more about the technology underlying many web sites and services, so that I may be able to use the knowledge I gain during this project in future projects.

Secondary to that, this specific project is one that I have been tossing around in my head for a while. The idea is a tough nut to crack, but there is definitely a need. I have volunteered for many different groups and organizations, each with their own level of expertise in management and recruiting. Having a site like this to ease those issues will help simplify operations and make it easier for them to focus their efforts more on their goal and less on operations management.

Henry's Motivation

I wanted to learn new technology. I am a bit embarrassed to not know much about servers so that was a motivation for me. In the end, I learned a great deal. At one point, we had to set up the sever all over again because we had to switch Amazon accounts to a different team member, considering David's trial was about to expire. The process took hours but keeping in mind the number of steps (dozens) it was not that bad. It is nice to see how things work behind the scenes of a website, so to speak. I am sure I will continue to learn more as the project evolves.

During the two semesters, learned about Tomcat, Hibernate, Servlets, Web Services, etc. I really like Hibernate. Hibernate automates the process to a degree, acting as an agent that translates data into different formats, from the server to a Java class and back again. I learned all this by doing various tutorials I had found online. Luckily, the amount of resources for Java and Android is high, so the information and knowledge is definitely out there.

A big motivating factor for me was putting this project on my resume and being able to say, "yes" I know about servers and websites and such. Website work was what my current resume was missing. I had done years of C++ and some 3D in the past, but lacked web experience. Unfortunately, UCF generally does not teach you relevant technologies that employers are actually looking for (in my opinion). You really have to be aggressive and teach yourself. This class and project is a welcome change. Instead of learning about something from 1935 (such as the Turing Machine which I had the misfortune to study in several classes) I feel we are at the forefront of technology that employers seek out.

Yet another motivating factor is to make a difference and help people, which I hope this project will do. Even if it does not take off immediately, the potential is there. It is nice to be able to use job skills for the greater good instead of the usual website that focuses on trivial, mundane social aspects. The world does not need another Facebook. Nowadays, I think people are actively looking to volunteer in some way and hopefully our website and app will make that possible or at the very least, inspire.

Function of Web and Android Application

Web Component

The website provides and easy way to create an account and profile for a user so that he or she can jump right to the fun part of finding organizations and events. The user is provided the option to create an account with only the following information

- Email
- Phone Number
- Username
- Password
- First Name
- Last Name

We plan to continue work on the website. An upcoming feature is the ability of the user to create an account with Facebook, which makes it easier by pulling information from Facebook, and asking the user for the rest. This would also be useful for things like posting scores. We also would want to make this Facebook option very non intrusive. Many people do not like using Facebook connect because they are afraid it will bother them. We want to provide the services without being annoying.

The website is useful for finding events. This is accomplished with a calendar view and a list view. There is a way to constrict the parameters of the list/calendar so that you only see events that matter to you, as described in the General Idea section.

Each user also has a profile. This profile keeps track of the events the user has attended, along with the users overall statistics. The statistics include

- Hours worked
- % in different fields(Clean up, Service, Etc...)
- Organizations worked with
- Interests
- Number of events

Profiles for every user is viewable by all users, but detailed information such as phone number, and email, is only viewable to organizations the user has an event with. This gives the user full control over who sees their important information.

Organizations also have profiles that are set up by the creator of the organization. Organizations include many of the same statistics as users. Organizations also have information about the information, and all contact information would be available to all users.

Groups, the last main profile page type, is a collection of users. The groups are useful for groups such as the boy scouts. Groups are able to keep track of all of their members, and their statistics aggregated. Users can join a group, and the creator of the group has to allow them into the group.

From the users point of view, signing up for an event is easy. There is a way to sign up with one click (and a confirmation dialogue). This dialogue then sends that users main information to the administrator of the organization. The dialogue also allows the organization to see the users contact information until the event has passed.

From the organization point of view, it is easy to keep track of the users who have signed up. There is a page for managing all the signed up users for each event.

Some events are open enrollment, so that users could join without confirmation. This is useful for big events that do not need a number of people showing up, like a park clean up.

Web Component Goals/Constraints

The web component needed to be fast and simple. The design needed to work across browsers and look good. The average person should be able to navigate and sign up for the things they want with minimal clicking. The website was also designed in such a way to motivate people to volunteer more frequently. For instance the feed page was made so that users had events that might interest them on the front page when they log in. This gives them something quick and hopefully interesting to sign up for, motivating the user and simplifying the experience.

Android Component

The Android component of this application would be for easy mobile use. When a user is out of his house but wants to find out what time and where the event he or she signed up for is, it should not require him to find a laptop. Also importantly, if someone is out of his house with nothing to do for a number of hours, he might want to find events that he could be a part of. The Android component is what this is for.

The user has two main abilities on the Android component.

- Find Events
- See events that he or she previously registered for.

This is accomplished by the user signing in first. Once he or she is signed in, the user is able to see all of the information about his or her events. Also the user has the option in another tab to find events. The finding feature allows him or her to restrict the parameters to find the event that worked out.

Organizations also have two abilities

- Accept a user for the event
- Approve hours for the event

These two abilities are both good to have when the administrator is at the event and a user needs to be signed up on the fly, or the administrator wants to approve someone's hours as soon as the leave.

The Android component also has room for expanding. In the future it would have many if not all of the components of the web component.

Android Component Goals/Constraints

The Android component was designed to be lightweight, and only include the things people need to do on the go. The Android component needed to be so simple so a user does not need to focus intently on the app. This makes it easier for someone to pick up their phone and volunteer on a whim. We did not be including ads in the mobile application, at least until adoption is picked up. We might instead create a pro application in the future with more useful features. The application needed to be very low data usage, and lightweight so as to keep people from removing the application. No one likes a battery sucking, data wasting mobile application, and remove applications that are as such.

Organization Requirements:

Clients are organizations who post volunteer ads. There is no limit on the number of ads they can post or the number of volunteers needed. Right now, there is no cost to post an ad. However, in the future there may be a cost to have an ad appear first in the search results. In the future we would like to set up a system where an organization will be verified by a moderator when an organization requests they be verified. A client is required to provide basic information such as address, phone number and email address so users can contact them. This information is also needed so a user can do a basic search.

User Requirements:

Users are individuals who would like to volunteer. Users can be in groups or organizations. There will be no cost for them to register. Creating an account is free. Once a member, there is no requirement to respond to an ad, although a point system and leaderboard encourages them to do so.

Group Requirements:

Groups are for users to be part of. Unlike organizations, groups do not organize events, they just participate in them together. Groups are a way of working with friends toward a common goal of collecting the most community service hours or points. A good example of a group is a boy scout troup. A boy scout group, does not necessarily hold events for the public to do service (though the boy scout organization does), but they do participate in events. Whenever the group volunteers though, they probably want all of their volunteers to be accounted for in the registration which we would provide. Also this is a good way for them to compete with other groups, such as other boy scout groups, as we provide tracking for the hours of all the members in a group.

User Point Scoring Specifications:

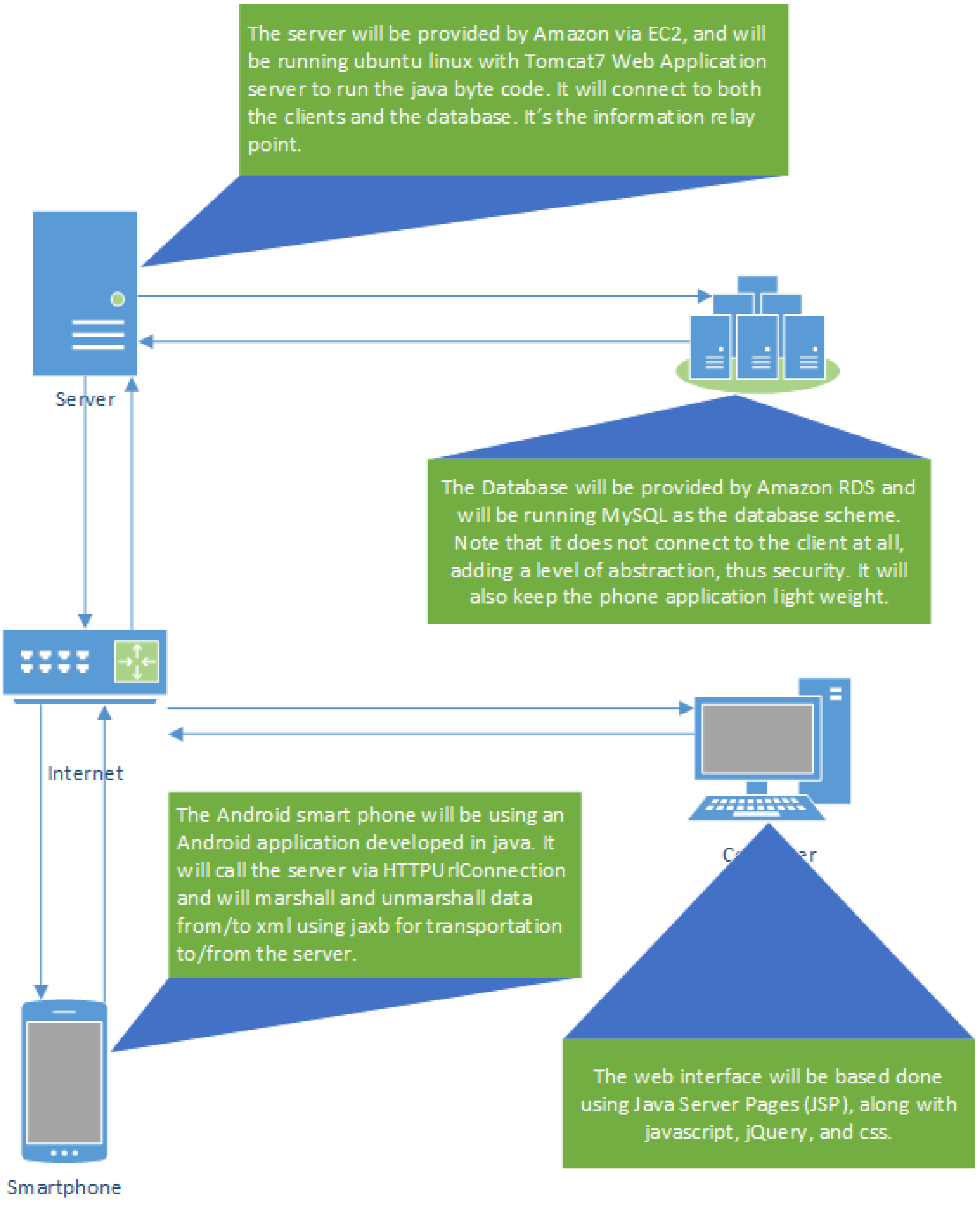
The user earns points as they volunteer. Points are given based on the number of hours volunteered times a multiplier that might be added for special events. Users can earn additional points if they volunteer at more than one organization. Other statistics such as the events worked are tracked so that users can provide the information to schools, jobs, or even the state

Matching a Volunteer with an Organization:

When a user conducts a search, parameters passed are used to return search results. Other specifications that affect a search include, but are not limited to user availability versus volunteer time needed, a volunteer's history of attended events, etc. A volunteer may also filter search results based on tags associated with skills required for tasks.

Diagrams for better understanding

Overview of Technology Diagram



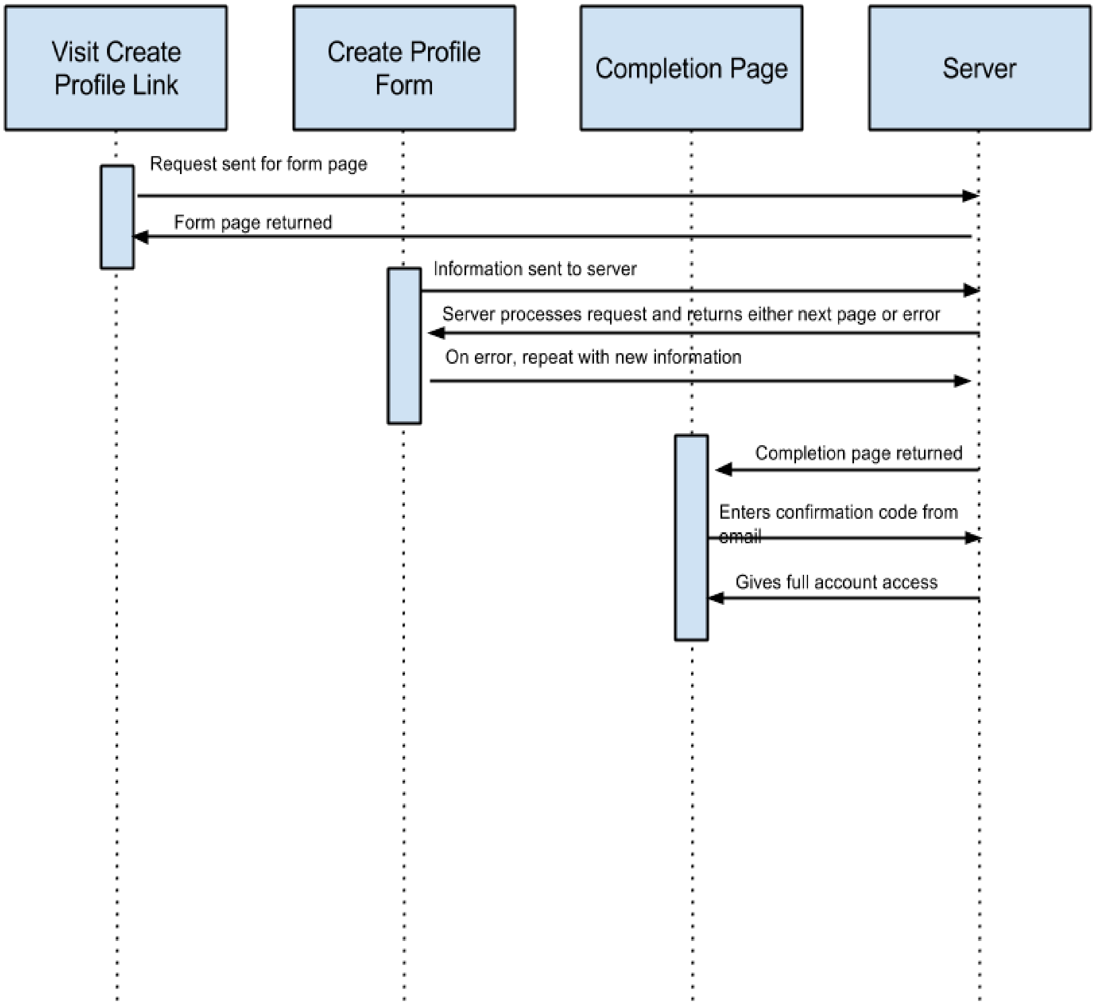
The server will be provided by Amazon via EC2, and will be running ubuntu linux with Tomcat7 Web Application server to run the java byte code. It will connect to both the clients and the database. It’s the information relay point.

The Database will be provided by Amazon RDS and will be running MySQL as the database scheme. Note that it does not connect to the client at all, adding a level of abstraction, thus security. It will also keep the phone application light weight.

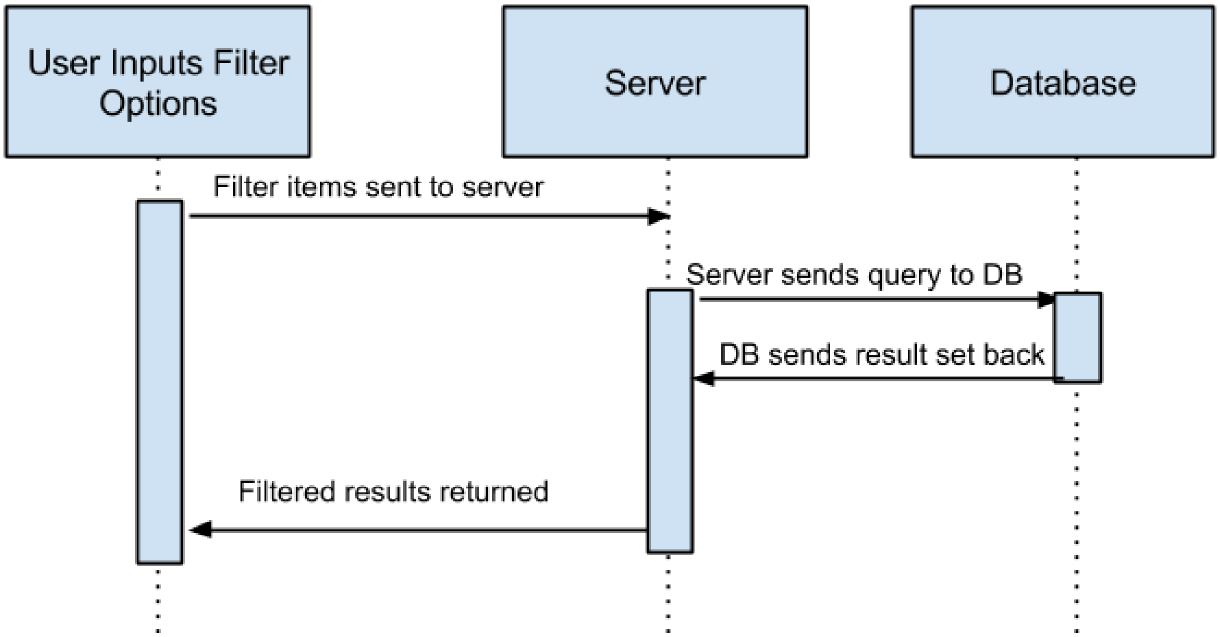
The Android smart phone will be using an Android application developed in java. It will call the server via HTTPUrlConnection and will marshall and unmarshall data from/to xml using jaxb for transportation to/from the server.

The web interface will be based done using Java Server Pages (JSP), along with javascript, jQuery, and css.

Sequence Diagram for creating profile


Visit Create Profile Link
Create Profile Form
Completion Page
Server
Request sent for form page
Form page returned
Information sent to server
Server processes request and returns either next page or error
On error, repeat with new information
Completion page returned
Enters confirmation code from email
Gives full account access

General Query Sequence Diagram Models Search Filter


User Inputs Filter Options
Server
Database
Filter items sent to server
Server sends query to DB
DB sends result set back
Filtered results returned

Web Application

Technical Components Of the Website

We decided the web application should be a java web application. The decision was made based off a couple of big case studies and what the group already knew. For instance, another technology some start ups use for web applications is known as Ruby on Rails. Rails is a web framework based on the Ruby language. Twitter decided to use Rails when it was created but eventually found that it was not scalable enough for their use. Twitter eventually switched to Scala, which is a language that compiles down to Java bytecode and then runs on the JVM. Meaning we could extend what we have with scala in the future if we need to.

Often it is mentioned how newer web technologies are the ones that are really popular right now for start ups and I have found this is for two major reasons. Most start ups want to get off the ground really quickly, to get their product out there. Java web applications take a little more time than other frameworks such as Rails, because you need to re-deploy for every code change. Adding libraries and jars even requires restarting the application server at times. This is sort of negated though with new application servers such as Tomcat, which we are using, that allow hot swapping of code, meaning the server does not need a restart.

The other big reason is that Java web applications are overkill for smaller sites. J2EE applications (Java Web Applications) are very scalable but are not really needed for sites that do not have a large user base. Since we had time though to make this project, we felt it was in our best interest to create a web application that will be able to expand with our clientele.

JAVA

For a java web application, using java seems like a pretty obvious choice, and it is. There are other options such a Scala though, which should be mentioned. Scala is rather new to the web application field, and has very little documentation compared to java. Java stresses the views and the java code being separated, while Scala puts them more often into one file. We prefer Java’s modularization, to Scala’s unifying theme. Also the group was familiar with Java making it easier to get working quickly.

Amazon EC2

Amazon Elastic Cloud computing is a virtual machine service in the cloud. EC2 is hosted by Amazon as part of their Amazon Web Services. EC2 is elastic in that you can change the amount of resources that your server has very easily, as well as the flexible payments that only charge you for the resources you use. We decided to run a Ubuntu virtual machine instance as part of the free tier of services they offer. We chose to use Amazon EC2 as opposed to hosting our own server, or using another service such as google app engineer for the following reasons:

- EC2 has very reliable infrastructure and strong security measures as compared to providing our own server.
- EC2 is quick to set up and scale.
- EC2 provides a free tier with a dedicated IP that we can map a domain to, while Google App engine requires payment to map to your own URL
- App engine does not support a full virtual machine, but instead runs applications that follow App engine specifications which makes the code less portable.

Other big users of Amazon EC2 include Netflix, instagram, and Adobe.

Tomcat 7.0

Apache Tomcat is a very lightweight open source web server. Tomcat has has a Java Servlet 3.0 container known as Catalina, and a HTTP 1.1 protocol connector known as Coyote, that listens for connections on a chosen TCP port. The connector forwards the requests to the Tomcat engine for processing which returns a response. Tomcat was chosen for the following reasons:

- Very lightweight and able to run on a cheap Amazon EC2 instance
- Supports Servlets and JSP builtin, which opens the possibilities to add JSF and Hibernate support with added JARs.
- Very simple set up on linux

- We have no need for EJB container (JBoss)
- Tomcat has been favored by web application developers for many years (“Apache Tomcat”)

Amazon RDS

Amazon Relational Database Service is another Amazon web service. RDS is essentially a relational database provider that you can place any query language on. Relational Databases are easy to manage and have fast access to data interrelated data, making them a good choice for anything but the largest of databases. Amazon RDS specifically was chosen because it has a very easy set up, strong security, and great up time. Amazon RDS also comes with a free tier, which means there was no cost incurred for this component of the project this year. Most other relational database service providers do not provide a free tier, making this an easy choice (“"Amazon Relational Database Service").

MySQL - MySQL Workbench

My Structured Query Language is a relation database management system. MySQL is widely popular and supported on Amazon RDS making it a prime choice. Also with MySQL workbench it is more tempting because workbench simplifies the process. With workbench, one can create tables, columns, constraints and foreign keys, all from a very simple UI. You can even create data in the table without doing any queries. The program figures out everything for you. We decided on using it for all of these reasons. MySQL is used by Facebook, Twitter and LinkedIn, which are all sites that share many qualities as our final goal of this project (“MySQL Workbench 6.0”).

Hibernate

Hibernate is a library created by the makers of JBoss, another application server, that is used for Java Persistence. Hibernate maps objects to a relational scheme and can then persist them to a database, that is pre configured. Hibernate simplifies the process of putting objects into the database, and handles the connection pools, which makes a large project like ours more manageable, and probably more efficient than what we could do. Hibernate is licensed under the GNU Lesser General Public License, and therefore is also free to use. Also, the mapping of object classes to the database is made easier with the Hibernate Tools plugin for

Eclipse IDE which generates the classes from the database for you. Hibernate also helps, in that if an error occurs during a sessions with multiple transactions, such as deleting an entity with foreign keys, it can rollback the entire sessions, giving us peace of mind knowing our data will not be inconsistent ("Hibernate").

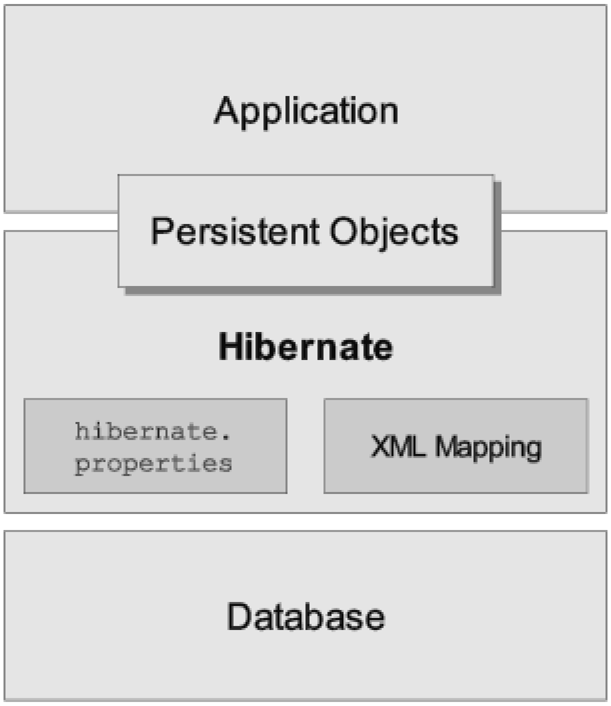


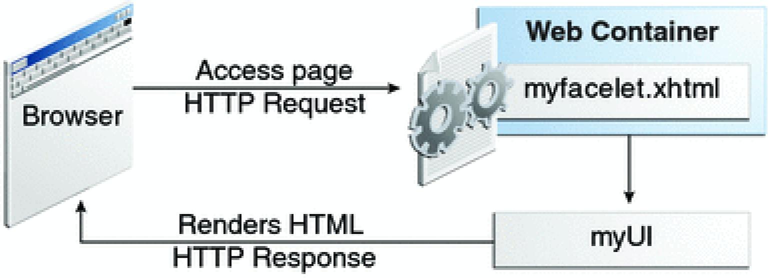
Image provided by http://viralpatel.net/blogs/introduction-to-hibernate-framework-architecture/

Gson

Gson is a an API to make mapping a Plain Old Java Object (POJO) to Javascript Object Notation(JSON). Doing this provides a convenient way to pass data through HyperText Transport Protocol (HTTP) connections. Gson is especially handy for passing information to the Android application because we are able to send objects to the application, rather than individual bits of data that would need to be parsed and then displayed. This speed ups development as objects are much easier to manage, especially when the information on the page will change regularly but should still persist through opening and closing the application ("GSON").

JSF

Java Server Faces is a framework for building java web applications. JSF is essentially a very robust API and tag library that makes it much easier to build a dynamic web project with java. With it we were be able to add components that are bound to server side data. The following picture shows the basic idea of how JSF works.



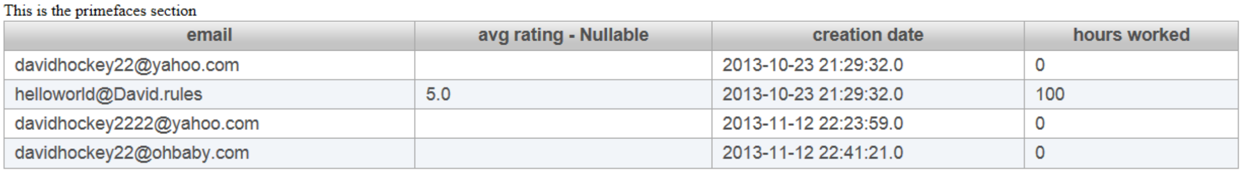
JSF is a good choice over JSP because JSF with Facelets is essentially a replacement for the JSP. JSF is far easier to use with managed beans than JSP is, since you really need to provide a framework such as struts on top of JSP to return pages from the server ("JavaServer Faces Technology").

AJAX

Asynchronous JavaScript and XML is not actually one thing but a style, and we are using implementations of it. AJAX is a way of getting information to and from a server without the UI being disturbed. This type of transaction is called Asynchronous because it is not in sync with the UI thread, instead running on its own thread. AJAX is implemented in JSF, so that partial processing of data can be done before sending the entirety of a form. AJAX is almost a must in web application development in recent years as users do not want to have to wait for a page to be submitted and reloaded just to find out they put in a username that already exists. AJAX made our web application more user friendly and more efficient, with very little programming overhead (“Ajax”).

PrimeFaces

PrimeFaces is a framework built on top of JSF that adds useful tags and components. Most of these components are widgets, but there are also some ease of use components such as a polling component to refresh certain information at a regular interval. An example of how primefaces can be implemented in our application is shown below. Primefaces also includes theming components so that we were easily be able to make the components fall into our scheme. Primefaces is used by big names such as Verizon, Dell, and NVidia (“Why Primefaces?”).



This is the primefaces section

email	avg rating - Nullable	creation date	hours worked
davidhockey22@yahoo.com		2013-10-23 21:29:32.0	0
helloworld@David.rules	5.0	2013-10-23 21:29:32.0	100
davidhockey2222@yahoo.com		2013-11-12 22:23:59.0	0
davidhockey22@ohbaby.com		2013-11-12 22:41:21.0	0

HTML5

HyperText Markup Language is the De Facto standard on the internet. Almost every single website uses HTML. Even with all of the other web technologies used in the project, most of the actual web site GUI was designed with HTML. With HTML5's release, some of the things that would have needed third party solutions like Adobe FlashPlayer, are not built in. Even web games are now being written in HTML, because it is becoming so powerful (“HTML5”).

CSS

Cascading Style Sheets, much like HTML, is used on almost all websites. CSS is used for styling elements. CSS can be written for a single tag, a class of tags, or even for re-defining a tag, on either the HTML page itself, or on a separate style sheet making it very good for theming a websites throughout all pages. Every other component mentioned so far makes the web application more functional, but CSS is entirely to make the website beatiful, which is incredibly important for acquiring and keeping users (“Cascading Style Sheets”).

Javascript

Javascript, while not quite as regularly used by the common website as CSS and HTML, is still widely used. Similar to HTML and CSS, it is built to work natively in most web browsers. Javascript is a lightweight, simple scripting language, that can be written for Functional, Imperative, or Object Oriented paradigms, and is dynamically typed. Javascript was used for some of the simple operations such as checking for password requirements.

Javascript being dynamically typed is both a blessing and a curse. The code is quite difficult to debug a javascript error because it can come from anywhere and it does not give much information, but at the same time javascript does allow for rapid development because you do not have to explicitly define line after line of information. Worth mention is the fact that that javascript is run client side, which is why it is useful for simple scripting on the page. This means that it has no access to server information unless it has already been loaded to the page, or retrieved with an AJAX request, but it also does not put any strain on the server (“JavaScript”).

JQuery

JQuery is a library built on top of javascript. JQuery is wildly popular as it simplifies development in Javascript. JQuery can turn many lines of javascript into a single concise line. JQuery is especially handy for AJAX, animations, DOM selection, and CSS manipulation. JQuery is used by Google, Microsoft and Netflix, along with hundreds of other big companies (“How JQuery Works”).

Security Components

In the last few years, security has become a huge concern for web applications. We want our users to feel that their information is safe with us so that they can trust us with it. With that in mind, we have taken a fairly close look at security and measure that we must perform. Some of the techniques we used are described below. More complete information on how the security was handled is in the Putting the Components together section of this document.

Hashing

Hashing is unidirectional function to change a string to a fixed length and entirely different string. While a password can be mapped to this hashed string with the hash function, the hashed string can not be mapped back to the original password string. A small change in a string makes the resultant string from a hash function entirely different. Essentially the function is used to make the users password different before placing it in the database. To check the login of a user, the password being sent to the server, is hashed and compared to the value in the database. If the two strings are the same, then the password is the same (with a VERY high probability.)

If someone were to break into the database, there is no function to return the passwords as plain text. This prevents users' passwords from being stolen, which is especially important if they use their password on multiple sites. To prevent the hashed password string, which can be stolen from the database, from being used in place of a user's password, hashing must be done on the server. For example, a hacker could stop the hashing from occurring on the client side by sending the hashed string that they stole from the database, which is then checked against the database with no modification, clearly giving access to the account ("Hash Function").

Example:

```
sha1("string") = ecb252044b5ea0f679ee78ec1a12904739e2904d
sha1("String") = 3df63b7acb0522da685dad5fe84b81fdd7b25264
```

Key-Stretching

Another thing to help prevent brute force hashing attacks to find passwords is called KeyStretching. Brute force hashing attacks consist of guessing and hashing every string combination until one matches the hash you are looking for. Multiple processors could be used to speed up these brute force methods, by hashing strings multiple times concurrently. Essentially KeyStreching is a method of slowing down the rate at which hashing can be done, which means hashing each guess takes longer.

Implementing an algorithm for KeyStreching by oneself is very difficult, but a javascript implementation exists in the Stanford JavaScript Crypto Library. Using this on the client side uses the processor of the client to add more hashing, without adding any strain to the server. Hashing a string multiple times at once becomes impossible with these key stretched algorithms, preventing multiple processors from being used to iterate through the hashes. One must still hash on the server though, for the same reason as mentioned in the above section (“Key Stretching”).

Salting Hash

Using Salt in a hash is a method used to break brute force cracking mechanisms, known as lookup tables or rainbow tables. A lookup table is a table of strings and their hash values that have been pre-generated. Salting a hash is as simple as prepending or appending a string to the password before using the hashing function, to change the hashed value to a different hashed value. Lookup tables can only be generated for so many string lengths, so adding salt to it helps prevent them from being useful. Salting should also be done with a unique string for each user, so that a table can not be generated with the salt added to all the strings. Also, the salt and the password together should be of a great enough length that it is impractical to store rainbow tables (a less memory-intensive lookup table) with all the salt and string combinations (“Salt”).

Tokens

Storing username and passwords in cookies or Android shared preferences is never a good idea. Although the shared preferences are semi-secure, they can be looked into, in certain circumstance, which makes storing critical information in them a very big security risk. Tokens are a good alternative. A token, which is just a randomly generated 256-character string, can be stored in the database in the user information, as a means of entering the site without a username or password. Tokens on servers works like getting a token at an arcade, and when you use the token, if it is valid, another one is spit out, and the old one is destroyed.

For instance, a token can be stored in a cookie in the browser, so when a user goes to the site, the token is sent to the server and verified against the user’s current tokens. This token then is destroyed and a new one is created and given to the user for access. Also, a token should be hashed in the database, with a salt that can be unique to the device, such as an IP address or, even better, a MAC address, since it essentially works as a username and password. This creates a more difficult challenge for an intruder who could take a cookie with the token to enter the account, but now would need to spoof their MAC or IP address. Tokens need to have an expiration date, as well, to prevent them from being given away with a device that has not been used in a long time. Tokens still are not a foolproof way of keeping intruders out, but at least, if they do get in, they do not have the users password, which is still very important as demonstrated with the next security measure.

Another important feature to include when tokens are used is the “sign me out of everything” option. If a user feels they may have had their login compromised, they should be able to change their password and/or sign out all current sessions, which means removing all tokens from the database for that user.

Also, tracking used token being used is another good practice. By tracking used tokens, a service can warn a user that their computer might be compromised. A used token being used should result in the user being logged out, all tokens being removed for the user, and an notification for the user on their next log in (“Security Token”).

Login on Write

There is no good way to keep a user logged in and still keep their accounts secured. There is one way to at least keep an attacker from making important changes to another user’s account, such as changing a password or a shipping address. When a change is being made to the account, the user should need to login with their username and password again. To make it easier on the user, this can be done only when the user is signed in automatically (ie remember me) or when the users session has timed out. The remember me buttons being off by default prevents a user from stealing a token and changing the user’s information, without too much overhead or overbearing security measures.

HTTPS using SSL

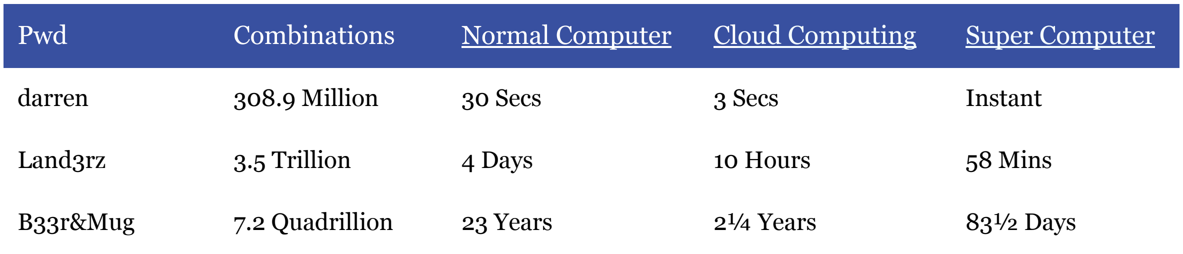
HTTPS describes a way of passing the information between the client and server securely. HTTPS is not a protocol itself but instead uses the standard HTTP protocol layered on the secure socket layer. HTTPS authenticates the server that a client is connecting to with certificates, which gives the user peace of mind, knowing they are connecting to the correct website, and not a forgery.

Also HTTPS encrypts communication from both sides, so that the information can not be read by a man in the middle attack. A man in the the middle attack is one in which a computer in between the client and server, intercepts the communication, and then sends the information, slightly altered usually, to the other side, so that it gains access to the server. With the communication encrypted, and both the client and server being authenticated before connections are made, it is almost impossible to perform a man in the middle attack ("What is HTTPS?").

Password Security

One of the weakest links of all secure systems is the user. A user will often use a short, simple, password that can be cracked with a brute force attack in a matter of hours, if preventative measures are not taken. If a users password is cracked, not only will their account be taken, but it becomes easier for a hacker to gain access to the users accounts on other websites.

One of the simplest ways to prevent a users password from being taken is to make password requirements. A longer password, with alternating cases, and symbols takes exponentially longer time than a simple password. The table below illustrates this point.



Pwd	Combinations	Normal Computer	Cloud Computing	Super Computer
darren	308.9 Million	30 Secs	3 Secs	Instant
Land3rz	3.5 Trillion	4 Days	10 Hours	58 Mins
B33r&Mug	7.2 Quadrillion	23 Years	2¼ Years	83½ Days

These password requirements do not need to be ridiculous to make a major impact.

Another preventative technique is to ask the user to change their password regularly. The interval can be chosen based on the password strength, but we did not use this method because users are more likely to write down or save their passwords in more places if they have to change their common password to a different one. This defeats the purpose of all the other password security measures ("Create Strong Passwords").

Technology Conclusions and Metrics

There are a number of ways to make a website, in the end we chose the technologies we believed would accomplish our task the best way possible. One such technology is Hibernate. We could have simply used MySQL, but that would have meant writing dozens of additional queries. Our site has not only one to one relationships, but one to many and even many to many, so in our case, Hibernate was a good choice. Finding a metric to prove this is difficult. Probably the time it saved along with the less likelihood of bugs is the best metric. Hibernate does not actually run faster than MySQL so the success of the technology can not really be measured in speed. However, Hibernate does have features that lead to better performance such as lazy loading, second level caching, and query caching.

Another big component we used was PrimeFaces. It just was not practical to build all our own form components. Although Primefaces can be buggy at times, it is probably the most popular solution used right now. All the technologies we chose are considered industry standard.

In the end, we probably would choose the same technologies again, especially for a database backed website that is query heavy. Research shows a decent number of jobs that require Hibernate which is possibly another metric showing the success of the technology.

Website Design

Here we cover in more detail some of the sections discussed above, in the context of designing the website and introduce new sections that only make sense in this context.

Purpose, Motivations, and Considerations

The design of the web front end will likely be the most visible aspect of the service. Being as it is, a great deal of thought needed to be put into the design to address the concerns of many types of users. We strove to make a highly usable interface for all users of the service. We did so by building on what we have experienced as users, doing research in designing user interfaces, and consulting with designers and potential users to help guide the initial designs into a final user interface. (The word final is used here to describe the finalized design for the initial launch of the product, and not connote permanency.)

Over the years we have seen many kinds of user interfaces rise and fall in popularity, research has been performed on the effectiveness of some of these, and others have only been reported on anecdotally, but, for the most part, there exists a big enough body of conflicting research to be able to defend almost any user interface design decision, both good or bad. For example, many designers are taught that your line lengths should be around 65 characters per line, and this is backed up by research and published in many popular typography books. The issue with this is that it has to do with text on paper, as opposed to text on a screen. Other studies show that longer line lengths on screen lead to better readability; some of these studies focus on physical line length and others characters per line, but displaying the information on screen instead of on paper. This, of course, is related to font size, context, and medium, so the issues here are complex.

As a group, we lack a formal education in designing user interfaces, so we were forced to rely on our experiences to conjure ideas of what we would like to see done, and what we would hate to include. As we are users with experience using many user interfaces, we have run into many issues over the years and some of us have experience providing support to people who need help using various interfaces, we have a large amount of experience to draw from. This can be both good and bad, as something that has become intuitive and expected to us may be alien and downright frustrating to other users. Also, our experience is limited to our personal exposure, so it is likely lacking in areas that would be otherwise useful for making design decisions. To tackle these issues, we focused our efforts on getting a working prototype as soon as possible so that we could use test the interface and work out any issues we did not foresee when designing the original interface, and consulted persons with more education and experience with designing and using things in peoplespace than we do throughout the design and testing process. We sought advice from people of varying skill levels and with different motivations for using the service, as our service provides multiple aspects of involvement, namely, group and organization management, event generation, and individual involvement. Since users roles may be mixed, e.g., they may be both organizers and volunteers, the user interface needed to be consistent across the web site, and the experience be continuous across all aspects of the service, something we accomplished by using the same component library and styling across all parts of the website.

Pitfalls

That being said, we did have some goals we wished to accomplish and anti-goals we wished to avoid that were specifically related to the website design. Our primary goal was to make something that works. When this is understated in projects, it tends to lead to scope creep and too much mucking around with details, which in turn leads to extended development times and a broken product. The next goal was that the site should work well and look good too. If a site does not work well, then users that suffer the poor design are likely to abandon the site. (Chapman, “10 Usability Tips Based on Research Studies”) This includes both performance issues and issues with UI design. If an action takes too long to complete or affect change, even just three seconds, some users are likely to give up using the site. This is important enough that Google has included page loading times into its search ranking algorithm. (Google, “Using site speed in web search ranking”) Page loading times, both perceived and real, can be affected by both backend and front end design. Lastly, an anti-goal was to create a website that is all things for all people, that is, make the user interface really thorough so that people can do every imaginable thing with the click of a (very tiny and hard to find and unintuitive) button (that only one developer knows, or possibly forgot, is there because they thought of it and put it there and did not give it any thought after the fact). Many designers get Keep It Simple Stupid pounded into their heads over and over, and with good reason. Sometimes, when motivated by a false assumption, such as “most users will not want to click more than three times to get something done,” the person designing the interface may wish to include more functionality into a view than is possible to do while maintaining a clean and easy to use interface. More on this later.

Related Websites and/or Services

When we were discussing the concrete goals of VolunteerMe along with what features the website should have, we did look at various related websites for inspiration of what we wanted and what we did not want. What we found was that many websites catered to one or two organizations only, such as the volunteer site for UCF students. While that is a great approach, we really wanted to consolidate the process by offering a single place users can go to be matched with a volunteer event.

Another site that captured our interest is VolunteerMatch.com. They really are the leader in this area and have a lot going for them. The user can search by interest, although not really by skill. Their website is colorful without going overboard and has a nice overall balance. Outdoing VolunteerMatch.com in only a few semesters would have been an enormous undertaking, so instead we decided to offer features the site largely ignored, such as the points system and leaderboards. We also encourage each organization that signs up to be specific about times and places. That was one thing we all noticed about VolunteerMatch.com. The information they gave was very vague, which made it hard to volunteer without doing quite a bit of additional research on your own.

General Methods, Tools, and Processes

The initial design process involved considering what it was that we wished users would be able to accomplish, and then branching out those ideas by carefully thinking about what those users may themselves wish to accomplish in addition to our primary focus. We did not wish to duplicate the functionality of other services that our users likely already use; we did not wish to create a facebook replica.

After considering what functionality we planned on including, we then started designing a user interface that addresses these goals. We started by roughly sketching out ideas, using physical representations such as pen and pencil on paper or marker on whiteboards to hash out ideas before moving on to creating electronic mockups.

Mockups were created using Balsamiq. Balsamiq is a very simple tool that is only capable of generating rough mockups of a user interface. This forces the user to decouple the design of the details from the design of the whole, so the higher level design decisions, such as element location, can be made before deciding on the detailed design, such as what style of corner to use on interface elements.

Once we had mockups, we moved on to prototyping. The reason that we went straight from mockups to prototyping without generating a detailed design first is that the detailed design is easier to test when a prototype is available to apply the design to, and the tools we used to power our web front end enabled us to use this approach without having to duplicate the work we did when we created the prototypes.

To come up with a detailed design, we considered elements such as color, typography, imagery, graphics, and logos. Methods, Tools, and Processes pertaining to the detailed design is be discussed in more detail in the Detailed Design section below.

As has been mentioned, we used PrimeFaces to accelerate the building of the website. PrimeFaces is a library of JSF components that uses jQuery and jQuery UI. As PrimeFaces uses jQuery UI, we were able to use a tool called ThemeRoller to create the CSS and scripting specific to jQuery UI. This alleviated the need to create the design elements and CSS from scratch and instead allowed us to quickly implement the intended design. Of course, we only used the generated CSS as a starting point, since it did not cover all of the design elements we needed.

Static vs. Dynamic Content and Compatibility with Browsers and Crawlers

PrimeFaces and jQuery include support for using Ajax, and some parts of it rely on Ajax to display information. This is good from the standpoint that these tools will make it easier to implement dynamic content, but dynamic content is not always a good thing, especially if implemented poorly. Dynamic content can make normal web navigation tools, such as the Back/Forward buttons and saving a URI as a shortcut that brings you to the view you were in, and make content invisible to search crawlers, such as those used by Google and Bing. A way to make navigation tools work with Ajax pages is to include a hash fragment after the URI. The benefit to using hash fragments is that the server code remains the same, and the browser will not refresh the webpage when the hash fragment is changed because hash fragments are processed on the client side and are not sent to the server during a request. The hash fragment can be easily changed by setting window.location.hash when a page is updated. This method of appending information to the URI has been supported since Netscape Navigator 2 and Internet Explorer 3.0, and has been included in browsers since. There are issues with older browsers not triggering the onhashchange event that is now commonly triggered when the hash fragment changes. When JavaScript listens for this event, it makes it easy to update content when this event occurs. For compatibility with older browsers, a timer can be set up to poll the URI for a changed hash. Since this issue is not present in modern web browsers, it can be safely ignored. Another issue with hash fragments is that this method still does not solve the issue with search engines being able to browse websites with dynamically loaded Ajax content.

The way that search services such as Google and Bing build up their index is by using web crawlers to download copies of websites, gather information from the downloaded page, and use it to update the information in the index. The way the Google crawlers decides what pages to visit is by visiting the links in its index, finding new links from these pages and adding them to the index, removing dead links, and then repeat. If a web page has dynamic content, then the crawler will not be able to access all of the dynamic data simply by following a link, since the link alone will often render the page without all the dynamic data loaded. The page has to be loaded in a browser and JavaScript has to finish executing for the content to display. There are several methods for dealing with making Ajax content searchable. Google published methods for crawling Ajax content, (“Making AJAX Applications Crawlable”) which Bing later began using, (Fox, “Bing Now Supports Google’s Crawlable AJAX Standard?”) that, once implemented, allow website operators to continue update their websites with minimal manual intervention after changes to make the updated website crawlable by search engines.

The method that Google published involves a bit of work ahead of time. Firstly, if an Ajax page uses hash fragments, then unique pages that we wish to index must be accessible using a hash fragment that begins with an exclamation point; that is, the first character after the '#' must be a '!'. Next, since has fragments are not sent as part of the URI to web servers, crawlers access these pages using a special URI. This URI uses '?_escaped_fragment_=' in lieu of '#!' to access these pages. In addition, Google escapes some special characters in the fragment using the %XX notation. The web server must be set up to recognize these page requests and send the requesting crawler a web page rendered as it would show after JavaScript has finished loading the Ajax bits of the page. The Google search index translates these '?_escaped_fragment_=' pages back into '#!' pages for display in web results, when this method for accessing Ajax pages is properly implemented. This translation also has to do with specifying a sitemap, but more on that later. Once the web server is set up to properly recognize an Ajax page request from a crawler, then web server needs to send the crawler the HTML snapshot of the Ajax page. This can be done on the fly by creating an HTML snapshot with a headless browser like or using a third party rendering service, or by creating static versions of the web pages. Regardless of the method used to create the HTML snapshot of the Ajax page, the HTML snapshot is sent as the server response to the special URI, and the crawler then uses the snapshot to update the search engine's index.

This, however, is more of a band-aid that has to be carefully administered than an actual fix to the way dynamic content is searched and accessed. In developing HTML5, a new method for manipulating the URI and navigation history without reloading the web page: the history.pushState() and history.replaceState() functions and the popState event. pushState() changes the URI and pushes the current URI into the browser history, so navigating back will make the browser load the previous URI, replaceState() changes the URI without changing the browser history in any way, so navigating back will not work to “go back” in case of this event occurring, but bring you to whatever page was last navigated to or pushed into the history, if there was a prior pushState() call. If, however, there is an event listener that listens for a popState event, then forward and backward navigation can be intercepted by JavaScript, and the page can be updated without reloading. This method of manipulating the displayed URI and the search history is supported by all current browsers, with the exception of Opera Mini. However, different browsers support the history API in different manners. For example, webkit generates a popState event after a page finishes loading. Both Google and Bing support crawling pages that use pushState().

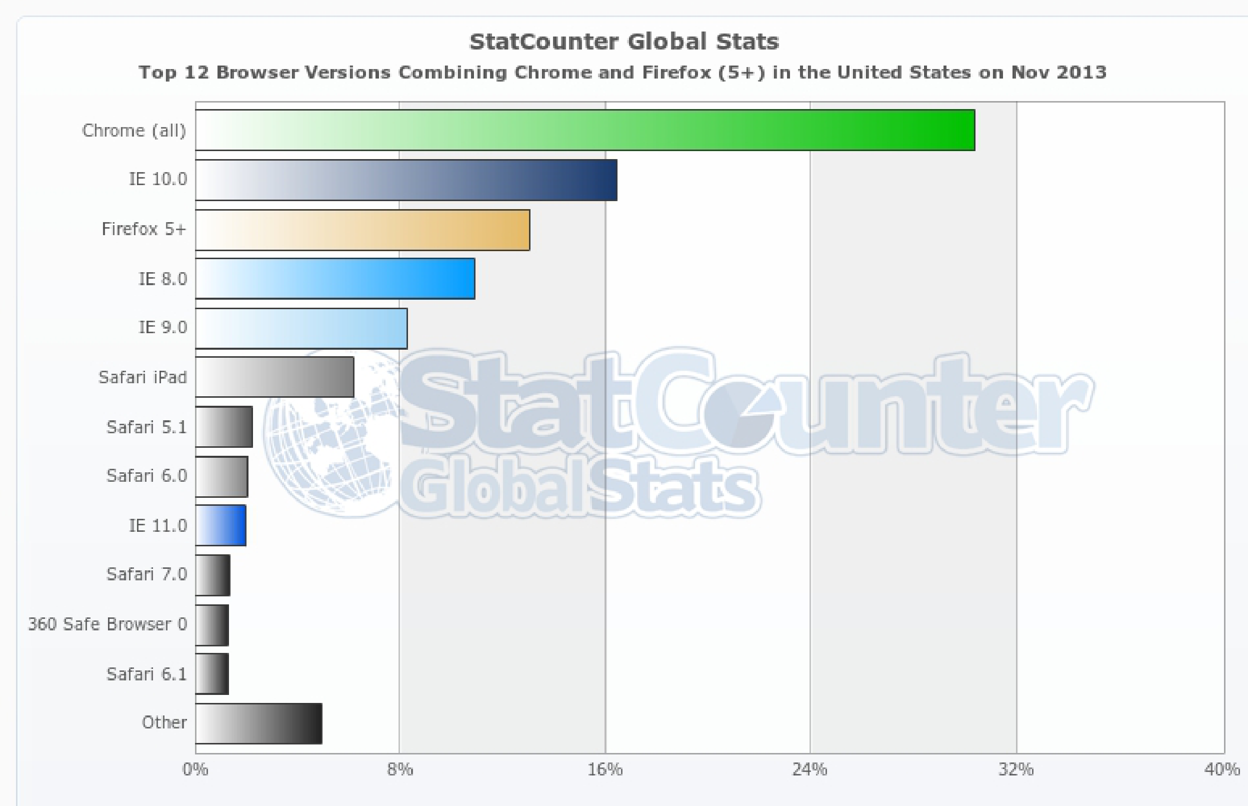
This tackles the issue of navigation on the front end, but only if the server supports requests to fetch pages at the URIs that are generated by the JavaScript. If a user manually refreshes the page or otherwise requests the page using the generated URI, the web server should send the user to the proper page. One way to do this is to have only a small set of static pages that the server loads, and then have JavaScript parse the URI and finish loading the page. This method involves the least amount of server-side code and relies on the browser to supply the view, instead of having the server generate the entire view each time a URL is fetched. An advantage to this approach is that the static pages are easy to cache, since they do not often change, along with the supporting scripts and CSS. This reduces the amount of web traffic required to use the website and the load on the server, since it will have fewer full page requests to process, allowing the server to serve more content, and thus serve more users without increasing the hardware requirements.

There are dangers involving using pushState() that need careful attention. One danger is creating an invalid URI. If a user navigates to the site, takes actions that change the URI, and then bookmarks that page, that breaks access to the site in the future for the user. Worse, if the user adds a link to the invalid URI on another site, that breaks the site for potential users and search engine crawlers. There is also the simpler issue of a user refreshing a page when the URI is invalid, or a web crawler trying to access an invalid URI that it indexed while visiting the site. Another danger is the proliferation of URIs that all point to the same page. Having too many different URIs for the same page dilute page rankings in search engines. This is because crawlers have limited bandwidth allocated to crawling sites, bandwidth that gets wasted crawling the same page over and over, and the page attribution getting split

among multiple URIs. Google and Bing recognize the canonical tag,
<link rel="canonical"
href="http://www.example.com/product.php?item=swedish-fish"/>
which can be used to collapse multiple URIs to one in the index. (“About rel="canonical"”) Bing also uses a search directive called normalization that makes the crawler ignore parameters in a URI, but this is something that is set up using Bing Webmaster Tools, not something that can be included in a file that the Bing crawler would specifically look for.

To tackle issues with older browsers and inconsistencies among current browsers, History.js was developed. History.js's purpose is to implement the HTML5 History API in all browsers, providing consistency in current browsers and fallback techniques in older ones. This library falls back to using hash fragments in older browsers, and transforms hash fragments back into non-hashed URIs should a link be passed from an HTML4 browser to an HTML5 browser. The HTML4 fallback can also be disabled, should it be deemed inappropriate. (Lupton, “history.js”)

One reason to disable the fallback support is to maintain consistency. If we have the web server set up to properly handle the URIs generated by the web page, then we can fall back to old school page loads when there is a URI changing event, making URIs consistent for all browsers, and thus parsing the URIs consistent for every page, simplifying the JavaScript required to read the URIs. Looking at the following chart, we see that at most about 25% of page requests in the United States in November 2013 were made with browsers that cannot handle the HTML5 History API, when the Internet Explorer versions 8 and 9 and Other categories are lumped together. Safari has partial support, according to caniuse.com, but using History.js would solve that discrepancy. (@Fyrd, “Session history management - Candidate Recommendation”) So, the benefits in decreasing server load with fewer requests for static pages would be negated by disabling fallback less than 25% of the time. (“Top 12 Browser Versions Combining Chrome and Firefox (5+) in the United States on Nov. 2013”)



Search engines, again, come into the reasons behind disabling fallback to hash fragments when using History.js. If the hash fragment form of a URI is used in a web page linking to the site, and we do not implement the hashbang workaround Google published, then the link will be incorrectly crawled and attributed.

Earlier, sitemaps was mentioned. Sitemap is a protocol that webmasters can use to help suggest pages to crawl to search engine web crawlers. Sitemaps are useful in pointing web crawlers at pages that are new or have few links pointing to them, and in telling a web crawler when a page last updated or how likely it is to update, among other things, such as the type of content on the page, the priority of web pages on your site relative to the rest of the pages on your site. (“What are Sitemaps?”) Sitemaps both reside on the website accessible as an XML file and can be submitted directly to search engines. The location of the sitemap XML file can be included in the site's robots.txt file. (“Submitting Sitemaps”) Due to the nature of the site we designed, however, this is not a good approach. The sitemap file would have to be very long in order to cover all of the user generated content, and would have to be dynamically generated in order to stay reasonably up-to-date, since the pages on the site are hard to predict, as they are user generated.

Robots.txt files are used to tell crawlers what they are and are not allowed to browse on a site. If there are pages or file types that we wish not to be crawled, or certain crawlers we want to exclude entirely, we can include those in the robots.txt file. (“About /robots.txt”) For example, if we did not want the Google Image Search crawler accessing our site, then we would include the following lines in our robots.txt file:

```
User-agent: Googlebot-Image
Disallow: /images/dogs.jpg
```

(“Block or remove pages using a robots.txt file”)
This can be useful to reduce the amount of traffic that the site receives from web crawlers and direct crawlers to more interesting parts of the site, so that the limited amount of bandwidth that crawlers allocate to crawling each site is used in a way that most benefits the site. We haven’t implemented a robots.txt file because the site we created provide different content to logged in users and logged out users that already provides the benefits associated with a robots.txt file.

The method we used the most to make URLs more easily usable, both from a readability standpoint and from a crawlability standpoint, was to use OCPSoft’s Rewrite. Rewrite allowed us to make parameter queries look like URL paths and change the underlying implementation easily without affecting the URL. The end result is a cleaner looking URL in the address bar and when linking. This makes it easier for event organizers to use a static-looking URL to point to their events and use whatever techniques they are already familiar with to promote their events.

Mockups, intended implementation

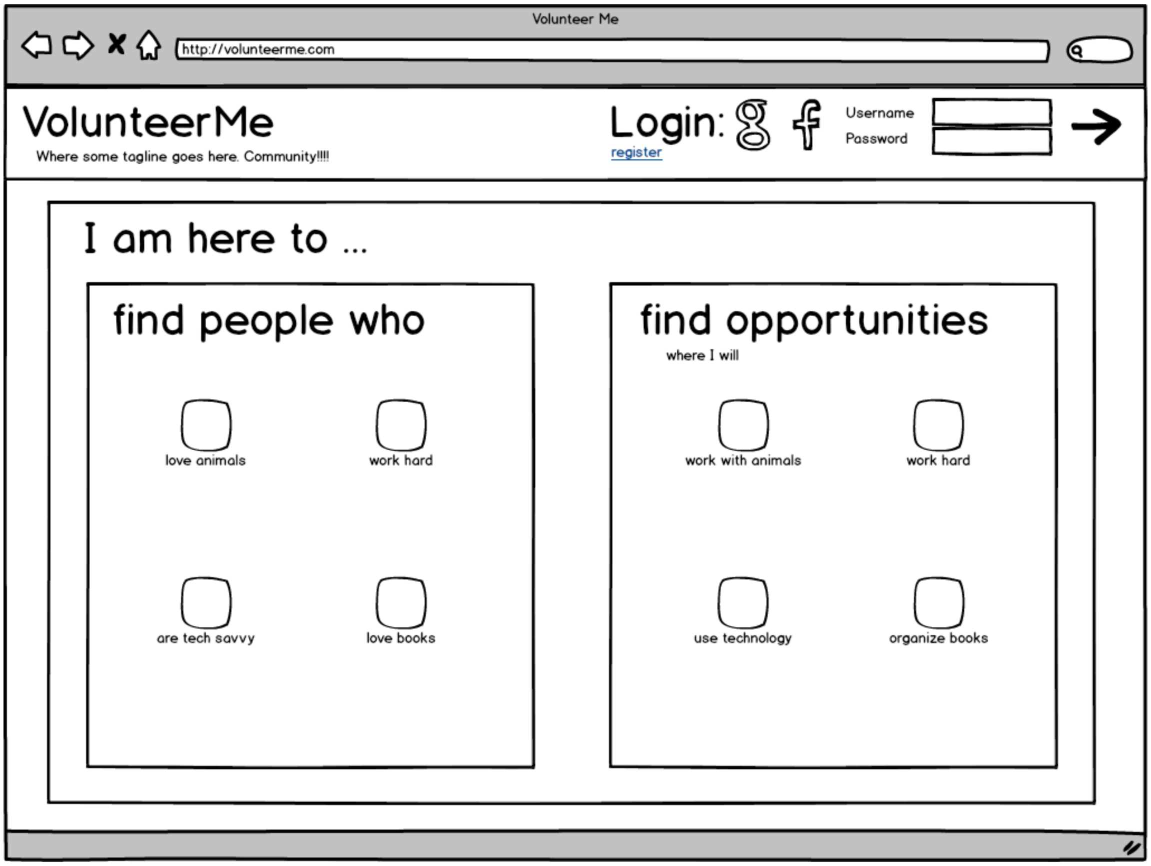
In this section we cover some of the designs for the website.

Home Page

The home page looks different depending on whether a user is logged in or not. This page is primarily a dynamic page, and so does not have much useful crawlable information.

Homepage when not logged in

When no user is logged in, the homepage focuses on getting new users to information that would be useful for them. The reason for this is that we want new users to immediately begin using the site, as much as can be done without logging on, without needing to register or sign on. This immediate access to functionality will hopefully keep new users on the site longer, so that they will be more driven to create an account. Actions that can only be completed by logging in or creating an account are clearly marked as such, or absent entirely. For example, it would not be possible to apply to volunteer for an event without first logging in, so the option is not present unless logged in. Searching for events, volunteers, groups, and organizations is all possible without having to log in either.

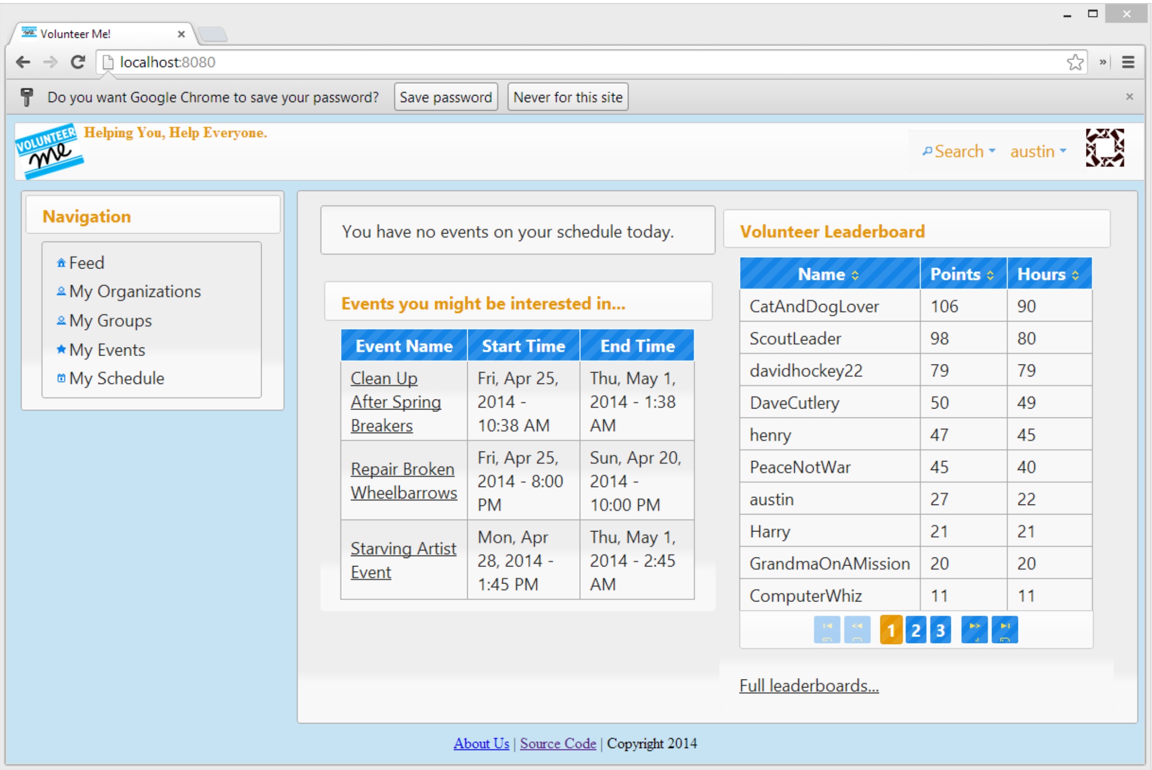


Early homepage design

Homepage when logged in

When a user is logged on, the homepage presents a very different view. The header changes to show information about the currently logged in user, and a menu for navigating to pages that a user is linked to is on the left.

In the center, there are upcoming events that a user has signed up for, along with events that will be occurring soon that a user may be interested in signing up for. on the right, there is a leaderboard that shows the points that different people in the community have accumulated.



Do you want Google Chrome to save your password?

Helping You, Help Everyone.

Navigation

- Feed
- My Organizations
- My Groups
- My Events
- My Schedule

You have no events on your schedule today.

Events you might be interested in...

Event Name	Start Time	End Time
Clean Up After Spring Breakers	Fri, Apr 25, 2014 - 10:38 AM	Thu, May 1, 2014 - 1:38 AM
Repair Broken Wheelbarrows	Fri, Apr 25, 2014 - 8:00 PM	Sun, Apr 20, 2014 - 10:00 PM
Starving Artist Event	Mon, Apr 28, 2014 - 1:45 PM	Thu, May 1, 2014 - 2:45 AM

Volunteer Leaderboard

Name	Points	Hours
CatAndDogLover	106	90
ScoutLeader	98	80
davidhockey22	79	79
DaveCutlery	50	49
henry	47	45
PeaceNotWar	45	40
austin	27	22
Harry	21	21
GrandmaOnAMission	20	20
ComputerWhiz	11	11

1 2 3

Full leaderboards...

About Us | Source Code | Copyright 2014

Implemented homepage when user is logged in

Event Page when logged in

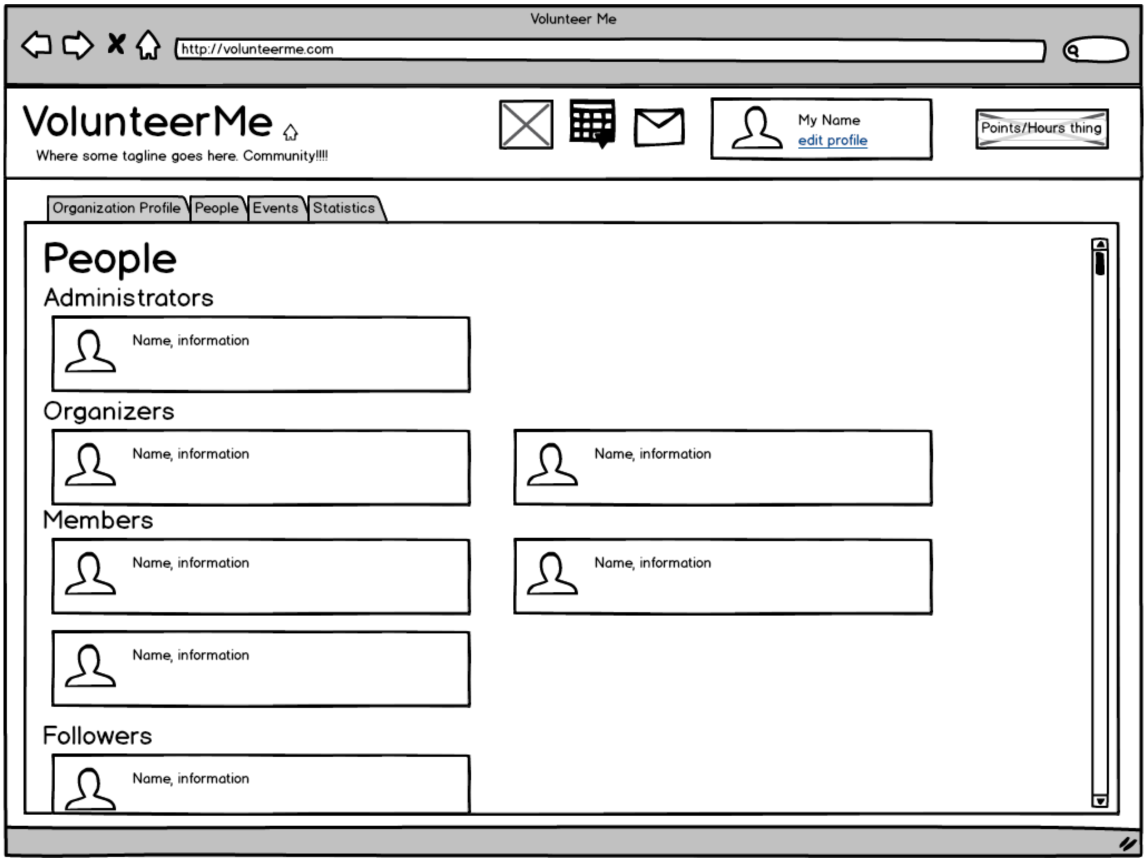
Whether a user is logged in or not, a page for an event looks the same, with the exception of a sign up button. This makes it easy for users visiting an event from an external link for the first time.

Event pages have their own URI, that way event organizers can easily share links to the event page, and search engines can crawl these pages easily.

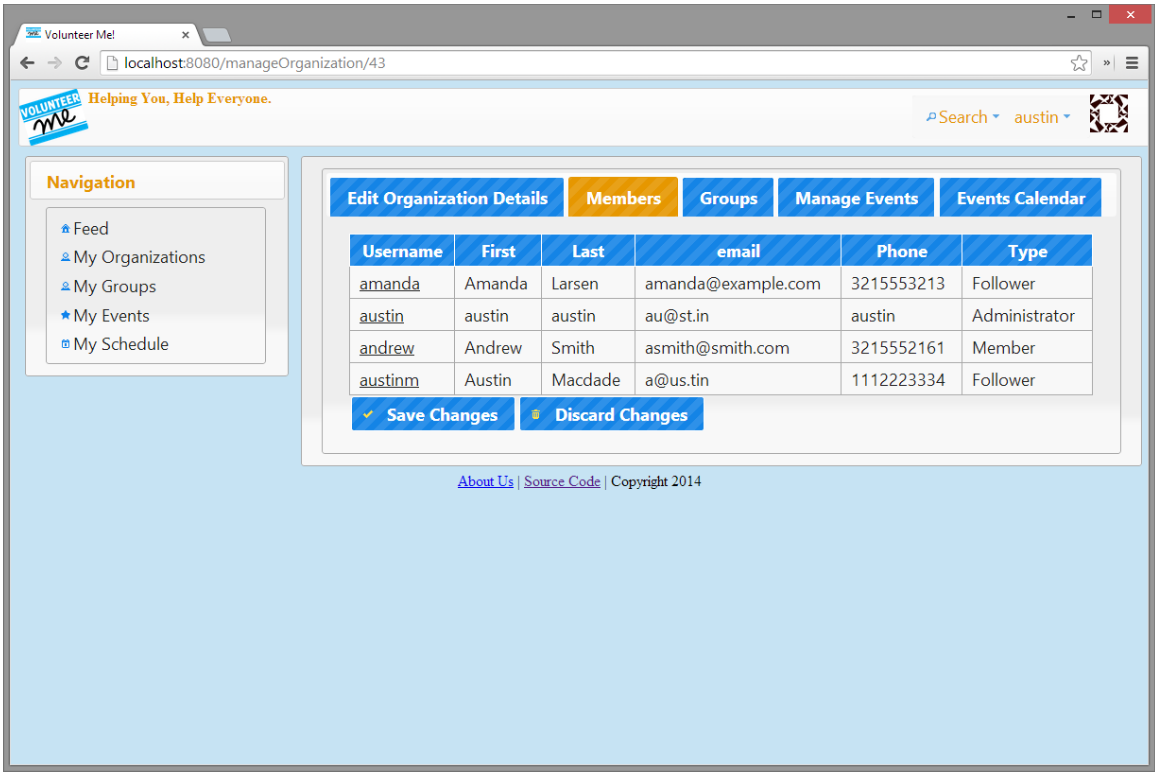
Group/Organization Administrator People Management Page

To offer a well rounded product, we designed and implemented an administrative view for groups and organizations. Options we have implemented are management of the organization or group's profile page, the people associated with the group, and the events owned or sponsored by the group. Primefaces provides easy to use tools for setting up administrative panels such as these, with support for dragging and dropping people between roles. Organizers have a similar panel, but it has fewer tabs and options specific to their role in the group or organization.

People who are members of the group or organization but have no administrative roles are simply listed as members and will not be able to access the administration panel for the group or organization for which they are listed as a member, but can view organization, member, and event details.



Early design of the organization administration page



Username	First	Last	email	Phone	Type
amanda	Amanda	Larsen	amanda@example.com	3215553213	Follower
austin	austin	austin	au@st.in	austin	Administrator
andrew	Andrew	Smith	asmith@smith.com	3215552161	Member
austinm	Austin	Macdade	a@us.tin	1112223334	Follower

Implementation of the organization administration page

Detailed design considerations

Web design covers a great deal of ground, there are many decisions to be made, and no clear cut way to make them. Many decisions came down to personal aesthetic, since that is the driving force of design. When asking the question, “will this drive more traffic?”, however, we do have tools that can help answer the question in a meaningful manner. Specifically, one such tool is A|B testing. Tools for evaluating the effectiveness of design decisions are written about in the general testing section below.

Shape and Layout of Design Elements

The feel of a site can be dominated by the shape of its elements, more rounded and the site can feel more friendly and inviting, more sharp and it can feel more trendy or more intimidating. Humans simply feel more at ease when dealing with “curved visual objects”. (Bar and Neta, “Humans Prefer Curved Visual Objects”) However, sharper objects can demand more attention and cause the brain to process them more deeply. (Lidwell, Holden, and Butler 62-63) So elements that need to be warm and inviting should be round, and elements that need more attention and to be well understood should be more sharp.

Element Size

Elements can be generally grouped into two categories: ones that can be interacted with and ones that can not. Things like buttons, links, scroll bars, etc. fit into the former, and blocks of formatted text, static borders, backgrounds, etc. fit into the latter. When considering how to size and position different elements, it is important to consider what class they are in. If the elements are in the interactive class, it is worth considering Fitts's Law when designing the web site. Fitts's Law models the act of pointing. The law can be used to estimate the amount of time required to point at an object on screen using a mouse or a finger. The following form of Fitts's Law is called the Shannon formulation:

$$T = a + b \log_2 \left(1 + \frac{D}{W}\right)$$

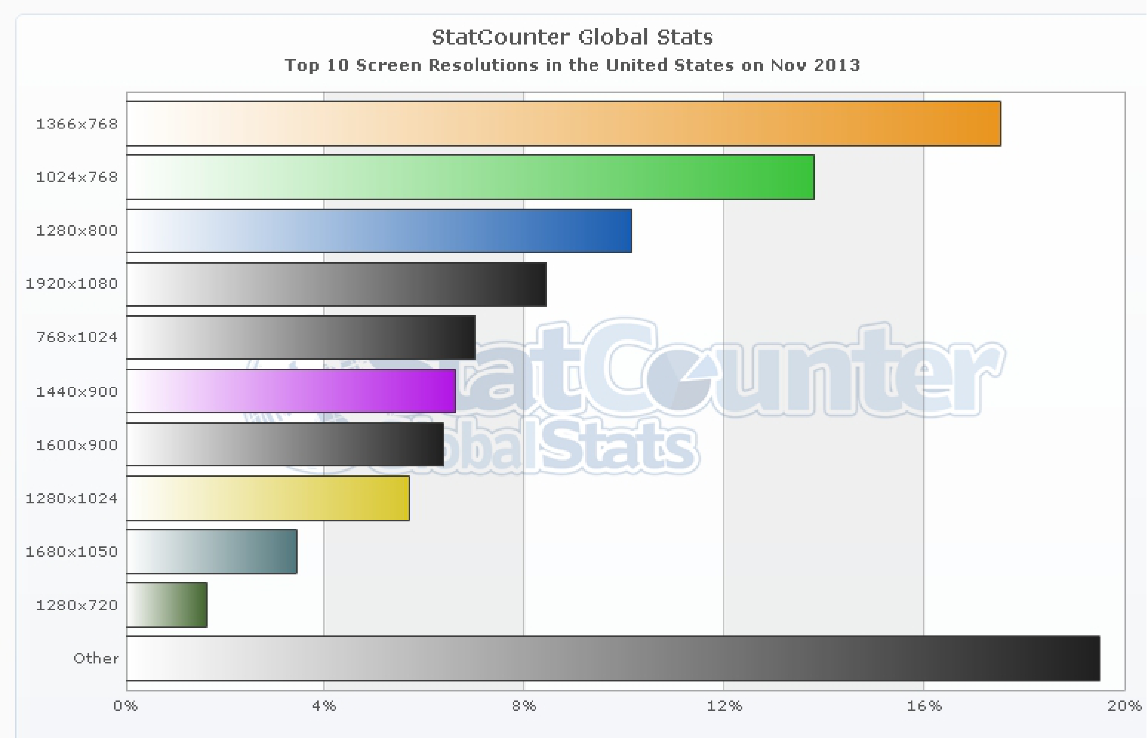
where:

- *T* is the average time taken to complete the movement.
- *a* represents the start/stop time of the device
- *b* stands for the inherent 1/speed of the device
- *D* is the distance from the starting point to the center of the target
- *W* is the width of thetarget measured along the axis of motion ("Fitts's Law")

Since *a* and *b* are constants that are device and user dependent, the only controls we have over this equation are *D* and *W*. So, if we know an element will be interacted with via pointing, to minimize the amount of time it takes to interact with said element, we should take care to minimize the distance to the element and maximize its size, within reason and good taste.

Resolution

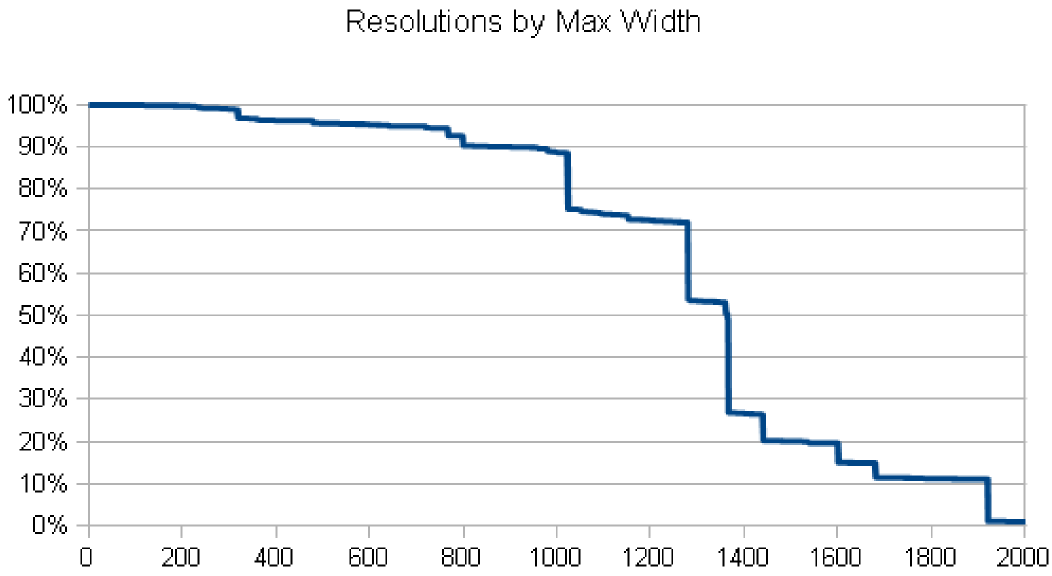
Another important decision is how to approach layout changes based on screen resolution. We could have a site that dynamically changes based on the amount of horizontal and vertical resolution is available, like how Google Plus adds more columns, or we could have a fixed width site like, well, most other social media and “feed based” sites with news feeds and articles. Either way, we needed to decide then what minimum and maximum resolutions we had to focus on. For this, we relied on data collected and published by others. The chart below, for example, gives good insight on what screen resolutions are common this year.



(“Top 10 Screen Resolutions in the United States on Nov 2013”)

From this, we can see that for the most common resolutions, there are quite a few pixels to play with. Most of them seem to be at least above 1000 pixels. Upon a cursory perusal of the web, many sites rendered the body container of their pages in a container with a width of about 1000 pixels. Note that there is one screen resolution in the above chart that would be affected by this thousandish pixel limit, the portrait 1024x768 display size.

The following illustration focuses on the percent of users that will be cut off and be forced to use horizontal scrolling for given minimum display area widths. This data is based on actual screen resolution, not the display window width. So, this data is only accurate for the scenario when the browser is maximized and has no chrome blocking the horizontal display of the page. Of course, this is not always the case, as some people with prefer to browse in a non-maximized window, and some browsers have chrome surrounding the whole display area when maximized, preventing websites from using the whole width of the screen.



(“Internet Users Screen Resolution Realtime Statistics for 2013”)

Looking at the data in this light, we can see how many users we would expect to exclude by supporting different minimum resolution widths. Something to also consider is mobile users. Since we are working on an app to interface with the web services, we are not too focused on mobile web development, but we still considered how people access the web. In the last year, about 13% of all web accesses in the U.S. measured by StatCounter were made on a moblie device. Of that, only around 12% of those accesses were made from devices that can display a super thousandish wide web container, or about 1.5%. So, of that 13%, we would be forcing 11.5% of them to use horizontal scrolling to have full access to the website.

Color

The colors used in a site set the mood for the user and can help guide the user's experience. Blue tends to be the color of choice in many sites, sometimes by chance, but usually not. Blue tends to be a good color to use because it is pleasant to most people, and only very few people have colorblindness that affects their seeing of blues, making blue a very safe color for designs. In fact, facebook is blue because Mark Zuckerberg is red-green colorblind, and chose a color he could see. Likewise, orange and brown tend to be liked least, but can be used successfully, nonetheless.

Depending on brand dominance, the brand colors may be used as the primary site colors. For instance, Coca Cola, Five Guys, Verizon, and Cabot use red themes, reflecting the color theme of their brand. Other sites may go with locality, ranging from nearby institutions, such as UCF, or something much broader such as national identity.

There are many resources available on the web for aiding in color choice. Web Design Ledger has a good list of tools that can aid in creating color palettes, including palette sharing tools such as Kuler and COLOURlovers.

The issue with looking at evidence for choosing colors for a website is that the evidence tends to be anecdotal, or at best case specific. In one case, designers ran a test on the button color of the “Get Started Now!” buttons, all else being equal, of the colors red and green. The designers found that the overall conversion, the rate of a website developer's goal being achieved per visit, was 21% higher when the site displayed a red button than when it displayed a green button.

All said, there are pages of conflicting advice on what to do, but there is also a great deal of advice on what to avoid, something that may be more useful, as it creates useful boundaries to work within. Color combinations can be fatiguing, when using colors that stand out against each other, it can create confusing visual elements that fight each other for focus. Likewise, if a color combination looks fine to most people, it might not look fine to people with colorblindness. We kept advice like this in mind when selecting the colors for the site.

To decide on the colors for the site, we started with a monochrome prototype and then added color to the elements. This helped in knowing how many colors we needed to choose, and how they were supposed to be laid out, so that we knew what colors would be adjacent, and how to use color to draw attention to the correct areas of the page, when attention needed to be grabbed.

Typography

There are many typographical decisions we made, ranging from what font faces to use to line height and kerning, whether to enable ligatures, etc, as well as decisions on how to implement those decisions.

Font faces

We used multiple font faces in the finished website, since using one font across the entire site would likely make it look dull. We did, however, aim for a consistent look that does not leave users with a jarring feeling when going from one part of the site to the next. One of the simplest decisions made was whether to use a serif font or a sans-serif font. Many designers suggest sans-serif fonts for body text and serif fonts for titles or headlines, things that are big. The reason for this suggestion is that, although the reverse is usually suggested in the print world, most screens do not have high enough pixel density to properly display a serif font, which leads to readability issues when they are used for displaying body text.

Likewise, the larger size of headings and titles lends to proper display of serif fonts on screen. The next thing to consider is the font family used in each case. Font families are numerous, though not all of them are freely available. We will restrict our search to freely available fonts, so as to not incur costs on something that could otherwise be easily avoided. However, even when restricting ourselves to this class, we have a large base to choose from. We will evaluate many fonts and decide on a case by case basis as an implementation detail.

Web Font Services

Should we ever decide to use any fonts later that are not commonly present on a user's machine, or even if a font is common but we want to be sure that our text is displayed as we designed it to, then we will want to use a web font service to serve the fonts. This will cut down on the amount of bandwidth and computing power our server must provide, although it does increase the number of different servers that a client will have to communicate with to finish loading a web page. Google Fonts and Adobe Edge are both free and easy to use. The site, http://web.appstorm.net/roundups/web-development/the-best-places-to-get-web-fonts- for-your-site/, provides more options for us to consider, should we find Google Fonts and Adobe Edge lacking.

Should we decide to use a web font service, we need to consider how our site will function, as fonts will need to be loaded for the page to display as intended. If the font is applied before the text is rendered, then the page will take longer to start loading, as the font has to fully load before the part of the page that comes after it renders, and if the font is applied after the page loads, then the site may suffer from a phenomenon called Flash Of Unformatted Text or FOUT, and depending on how long it takes the font to download and render, the page may linger with malformatted text for a bit before refreshing with the new typeface. There are methods for dealing with FOUT, Adobe and Google have published CSS and JavaScript to help eliminate the occurrence of FOUT or slower page loads because of font downloads, such as adding classes to the HTML element dependent on whether or not there is a font request and what state it is in and only sending the subset necessary for displaying our web pages properly to the client.

<u>Imagery</u>

Due to concerns about storage and bandwidth, we wish to keep images to a minimum. We may in the future include the ability to have one image per person, group, and event, to aid in identification of people, groups, and events. Should we include this feature, then we will have to address how to store, display, and fetch the images from the users.

There are a couple of different ways to store the images, especially since there is a one-to-one mapping of images and users/groups/events. We can store the images as blobs in the database or as files in the file system. If we store them in the file system, then we must decide whether to store the link to the image as a field in the database or if we want to have a computed mapping from database entry to image location in the file system.

Storing the images in the database would probably be the easiest thing to do, as far as additional code is concerned, but it would put more strain on the database, leading to more expensive database transactions and probably slower page loads. Storing images in the database also does not allow the web server to take advantage of using generated paths could work well if properly implemented, but would be more difficult to change once in production, since changing the logic could involve finding and relinking all the images. At the very least, this involve redirection, which could put a slight administrative burden on the website management side, as then the images would be in multiple locations, and could cause performance issues depending on the type of redirection, operating system and file system used to store the images, as Windows suffers a slight performance hit from reading the disk, calculating the link redirection and then reading again.

There is another option entirely for user and organization images, that we used for the short term at least. We used a third party service called Gravatar. Using this method eliminated the need to handle file upload, storage, and conversion/resizing, since the service handle all of those requirements. All we have to do is request the image from the appropriate URL that includes the requested account and size information. As an example on how to use the service, the following code retrieves the avatar image from gravatar:

```
function get_avatar_from_gravatar(userid, size) {
 // derived from https://gist.github.com/jcsrb/1081548
 var url = '';
 url = "http://www.gravatar.com/avatar/" + userid + "?s=" + size
 return url;
}
```

This, however, is only useful for volunteers and organizations.

Icons

Icons provide visual cues, hopefully that are usually widely understood, for users to aid in completing tasks, including but not limited to navigation, and recognizing elements. jQuery UI provides a wide array of common icons, and uses them in some of its widgets. As such, these icons are also usable in PrimeFaces. We might in the future need icons other than what is provided in jQuery UI, in which case we would have to either design the icons ourselves or else find a suitable icon from somewhere else and incorporate it into jQuery. Luckily, the hard part in this case would be making or finding and adapting an icon, since incorporating the icon into jQuery seems to be trivial, based on this stackoverflow Q&A:

Putting It All Together

Now that we have all the components defined it is time to put them in place. All of the technologies above are used in some way to create the webservice, but some of them are more important than others.

To start we are using JAVA in ECLIPSE to write the server side code. The server code will be split into multiple packages. The first package to mention is the org.CommunityService.EntitiesMapped. This Entity package was auto generated by hibernate against our MySQL database. The package includes annotations mapping the class to the MySQL database tables.

Example:

```
@Entity
@Table(name = "Volunteer", catalog = "dbAppData", uniqueConstraints = {
@UniqueConstraint(columnNames = "EmailAddress"),
@UniqueConstraint(columnNames = "PhoneNumber"),
@UniqueConstraint(columnNames = "VolunteerName") })
public class Volunteer implements java.io.Serializable {
```

Another package used is the hibernate tools package. This package includes all of the necessary code to access the database information using the hibernate libraries, including the listener to start the hibernate service when the package is deployed, and the necessary connection code, for persistence or querying. By keeping the code abstracted from the other classes, the results of the hibernate transactions will be consistent and uniform.

Example:

```
public static Object persist(Object object) throws Exception {
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        Session session = sessionFactory.openSession();
        session.beginTransaction();
        if (object != null) {
            session.persist(object);
        }
        session.getTransaction().commit();
        session.close();
        return null;
    }
```

The next package that we used is the managed beans package. The managed beans will include all of the JSF backing beans for our pages. As mentioned above JSF uses backing beans, which are java POJOs that are scoped for the page as needed, so that all of the necessary information on the page can be populated and the even persisted back to the database when the page is submitted.. An example backing bean as shown below, is for our login page.

Example (code simplified):

```
@ManagedBean
@RequestScoped
public class LoginBean {
    private String username;
    private String password;
    public String Login() {
        try {
            Volunteer v = VolunteerService.getVolunteerByName(username);
            if (v!=null && v.getVolunteerPassword().equals(password)) {
                return "LandingPage";
            } else {
                new FacesMessage(FacesMessage.SEVERITY_ERROR, "Sample error
message", "Login credentials didn't match.");
                return "Login";
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return "error";
        }
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

A subpackage of the managed beans package is for JSF validators. Validators are java backing scripts that are called by events on the JSF page to check input that can not be checked client side. For instance, we needed to check the username of a user when he or she tries to log in on the server because the client side does not know the names of all of the users. The username must be checked without even changing the page so that the user will not have to keep resubmitting the form to check for a name not taken. This example is demonstrated in the following code.

Subset of JSF Page Code for Registration

```
<h:outputLabel for="#{registrationBean.username}" value="UserName" />
<h:inputText id="username" value="#{registrationBean.username}"
label="UserName" required="true">
   <f:validateLength minimum="5" maximum="20" />
   <f:validator
validatorId="org.CommunityService.Validators.UserValidator" />
   <p:ajax update="userMessage" event="blur" />
</h:inputText>
```

Validator Code

```
public void validate(FacesContext context, UIComponent component, Object value)
throws ValidatorException {
        String username = value.toString();
        Volunteer v = null;
        try {
            v = VolunteerService.getVolunteerByName(username);
        } catch (Exception e) {
            FacesMessage msg = new FacesMessage("Name Validation Failed",
"Username field is invalid.");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(msg);
        }
        if(v!=null){
            FacesMessage msg = new FacesMessage("Name Validation Failed", "This
username has already been taken.");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(msg);
        }
    }
```

The penultimate package we will be using is the services package. The services package contains code to work with each entity. From a service you are be able to call of the useful and repeated codes for entities. For instance, you are able to get all of a certain instance, get a particular one by it is ID (or some other unique key), or add that entity back to the database. By moving these out of the managed beans, the code does not have to be rewritten, the beans look cleaner, and you have a more consistent result.

Example of a Service Function:

```java
public static Volunteer getVolunteerByName(String name) throws Exception {
        String hql = "from Volunteer as v where v.volunteerName=?";
        ArrayList<String> params = new ArrayList<String>();
        params.add(name);
        try{
            Volunteer v = (Volunteer) (((List<Volunteer>) DBConnection.query(hql, 
params)).get(0));
            return v;
        } catch(Exception e){
            return null;
        }
    }
```

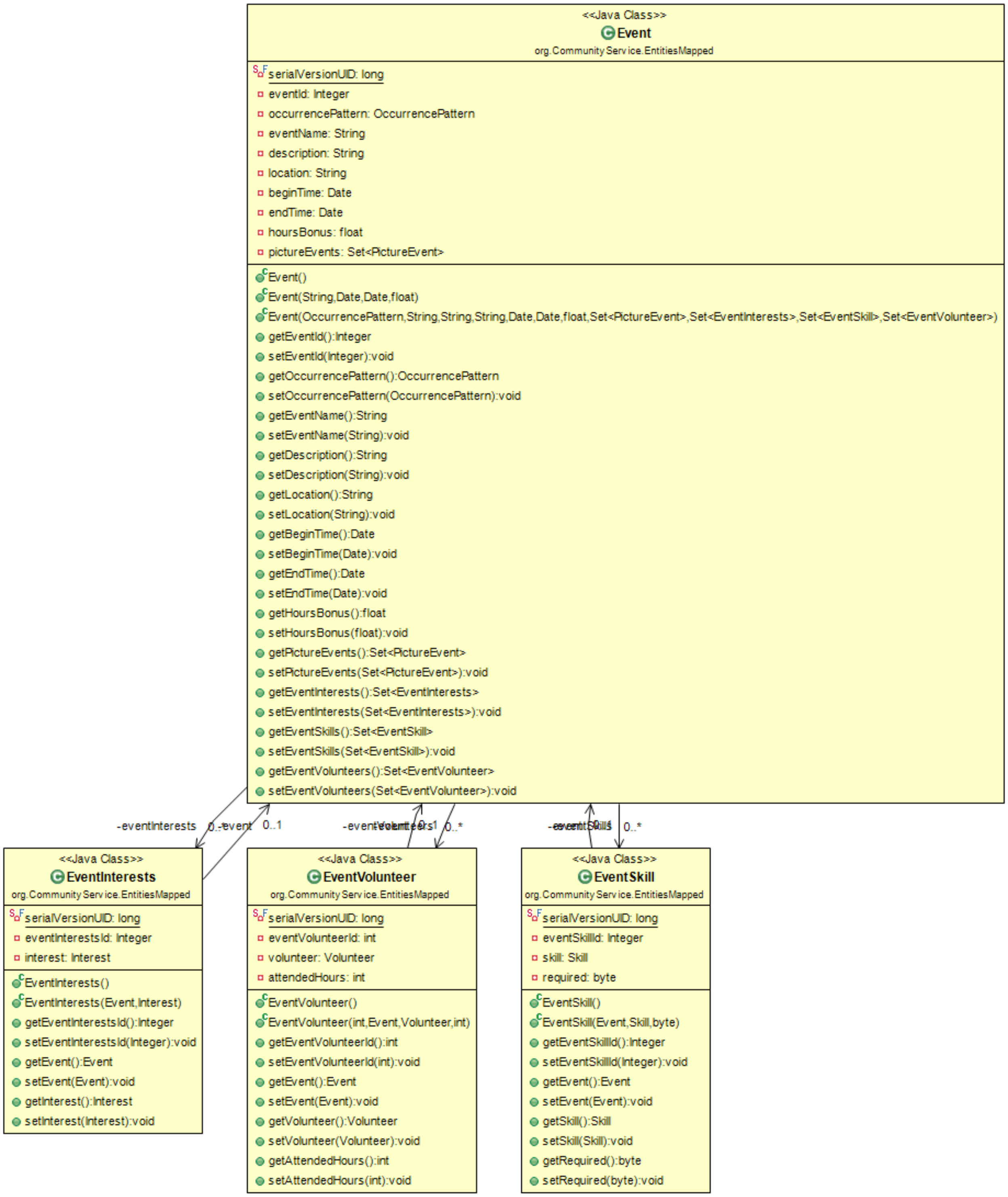
The last package includes the servlets for the Android application to transfer data to and from the database. The implementation in this package is discussed in more detail in the Android section.

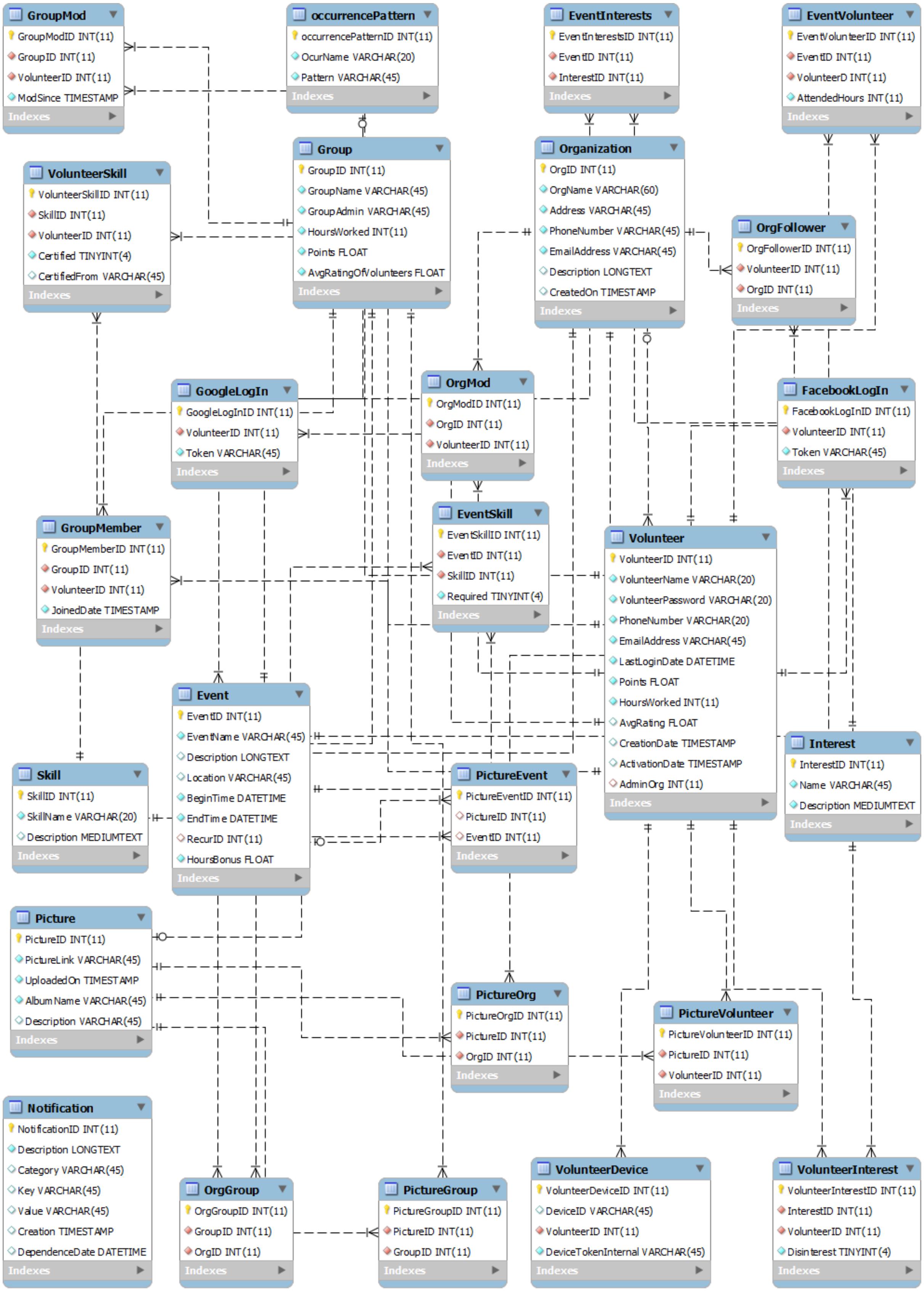
The package with all of these files is deployed on a Tomcat server, on our EC2 instance. The database connection, managed by hibernate is to the RDS database that we are using. The next page shows a subset of the entity classes to demonstrate the relationships and layout of the classes while the following page shows the structure of the entire database. The database was created below using MySQL workbench, and the diagram was generated from that as well.

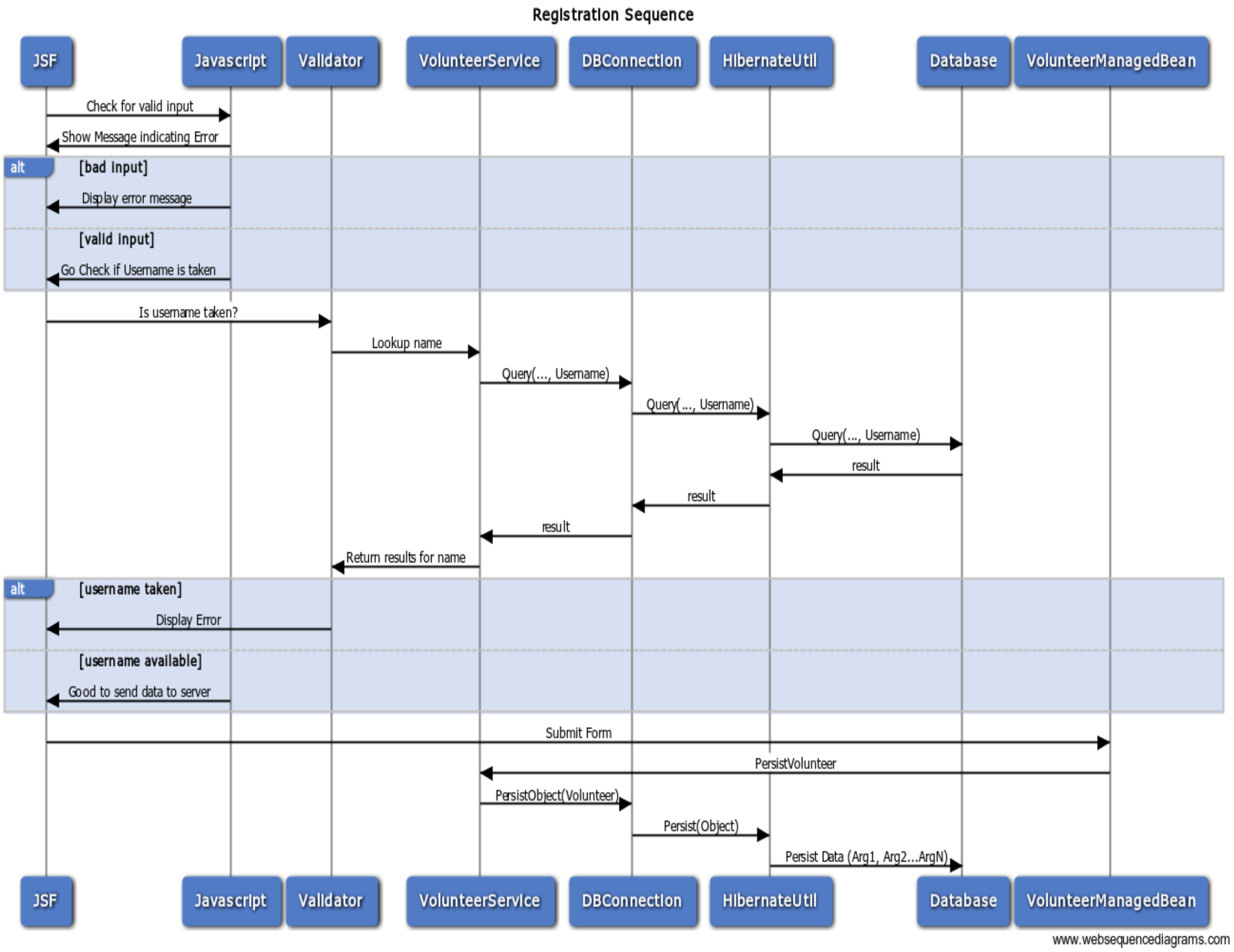
In our project javascript is used for many small things. For instance, before sending out any input javascript is used to check the inputs for correct parameters, and escape xss attacks as well. Also jquery is used for UI manipulation to make the site look and feel better for a user.

Primefaces is used for the UI elements and the ease of Ajax it provides us. Particularly data tables are used regularly from primefaces, and the ajax to lazy load extra information when clicking on table elements.

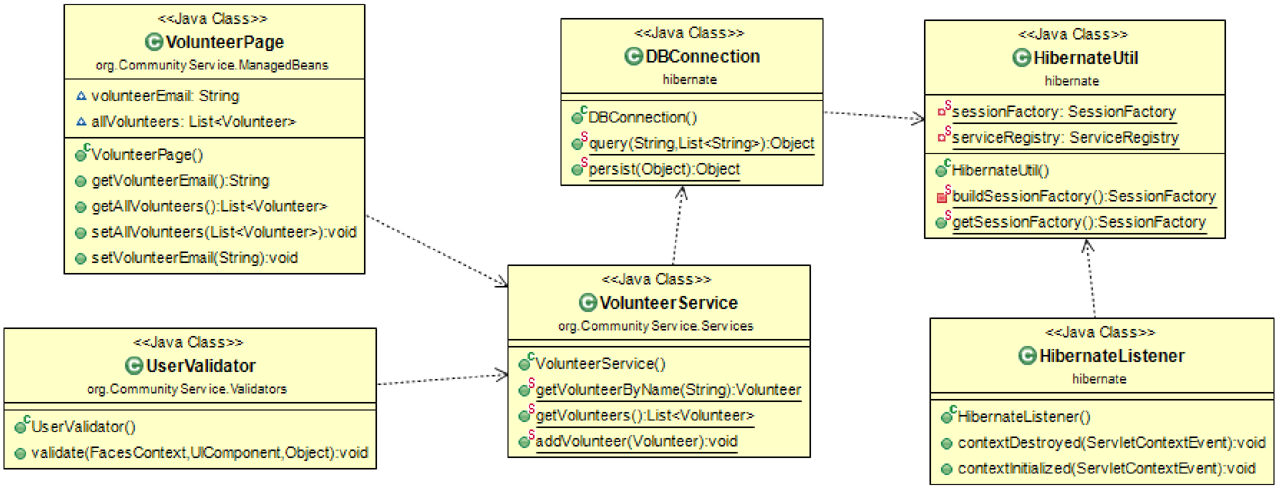
Examples of how the project back end works follow the next 2 diagrams:


<<Java Class>>
Event
org.CommunityService.EntitiesMapped
serialVersionUID: long
eventId: Integer
occurrencePattern: OccurrencePattern
eventName: String
description: String
location: String
beginTime: Date
endTime: Date
hoursBonus: float
pictureEvents: Set<PictureEvent>
Event()
Event(String,Date,Date,float)
Event(OccurrencePattern,String,String,String,Date,Date,float,Set<PictureEvent>,Set<EventInterests>,Set<EventSkill>,Set<EventVolunteer>)
getEventId():Integer
setEventId(Integer):void
getOccurrencePattern():OccurrencePattern
setOccurrencePattern(OccurrencePattern):void
getEventName():String
setEventName(String):void
getDescription():String
setDescription(String):void
getLocation():String
setLocation(String):void
getBeginTime():Date
setBeginTime(Date):void
getEndTime():Date
setEndTime(Date):void
getHoursBonus():float
setHoursBonus(float):void
getPictureEvents():Set<PictureEvent>
setPictureEvents(Set<PictureEvent>):void
getEventInterests():Set<EventInterests>
setEventInterests(Set<EventInterests>):void
getEventSkills():Set<EventSkill>
setEventSkills(Set<EventSkill>):void
getEventVolunteers():Set<EventVolunteer>
setEventVolunteers(Set<EventVolunteer>):void
-eventInterests
0..*
-event
0..1
-eventVolunteers
-event
0..1
0..*
-eventSkills
-event
0..1
0..*
<<Java Class>>
EventInterests
org.CommunityService.EntitiesMapped
serialVersionUID: long
eventInterestsId: Integer
interest: Interest
EventInterests()
EventInterests(Event,Interest)
getEventInterestsId():Integer
setEventInterestsId(Integer):void
getEvent():Event
setEvent(Event):void
getInterest():Interest
setInterest(Interest):void
<<Java Class>>
EventVolunteer
org.CommunityService.EntitiesMapped
serialVersionUID: long
eventVolunteerId: int
volunteer: Volunteer
attendedHours: int
EventVolunteer()
EventVolunteer(int,Event,Volunteer,int)
getEventVolunteerId():int
setEventVolunteerId(int):void
getEvent():Event
setEvent(Event):void
getVolunteer():Volunteer
setVolunteer(Volunteer):void
getAttendedHours():int
setAttendedHours(int):void
<<Java Class>>
EventSkill
org.CommunityService.EntitiesMapped
serialVersionUID: long
eventSkillId: Integer
skill: Skill
required: byte
EventSkill()
EventSkill(Event,Skill,byte)
getEventSkillId():Integer
setEventSkillId(Integer):void
getEvent():Event
setEvent(Event):void
getSkill():Skill
setSkill(Skill):void
getRequired():byte
setRequired(byte):void


GroupMod
GroupModID INT(11)
GroupID INT(11)
VolunteerID INT(11)
ModSince TIMESTAMP
Indexes
occurrencePattern
occurrencePatternID INT(11)
OcurName VARCHAR(20)
Pattern VARCHAR(45)
Indexes
EventInterests
EventInterestsID INT(11)
EventID INT(11)
InterestID INT(11)
Indexes
EventVolunteer
EventVolunteerID INT(11)
EventID INT(11)
VolunteerD INT(11)
AttendedHours INT(11)
Indexes
VolunteerSkill
VolunteerSkillID INT(11)
SkillID INT(11)
VolunteerID INT(11)
Certified TINYINT(4)
CertifiedFrom VARCHAR(45)
Indexes
Group
GroupID INT(11)
GroupName VARCHAR(45)
GroupAdmin VARCHAR(45)
HoursWorked INT(11)
Points FLOAT
AvgRatingOfVolunteers FLOAT
Indexes
Organization
OrgID INT(11)
OrgName VARCHAR(60)
Address VARCHAR(45)
PhoneNumber VARCHAR(45)
EmailAddress VARCHAR(45)
Description LONGTEXT
CreatedOn TIMESTAMP
Indexes
OrgFollower
OrgFollowerID INT(11)
VolunteerID INT(11)
OrgID INT(11)
Indexes
GoogleLogIn
GoogleLogInID INT(11)
VolunteerID INT(11)
Token VARCHAR(45)
Indexes
OrgMod
OrgModID INT(11)
OrgID INT(11)
VolunteerID INT(11)
Indexes
FacebookLogIn
FacebookLogInID INT(11)
VolunteerID INT(11)
Token VARCHAR(45)
Indexes
GroupMember
GroupMemberID INT(11)
GroupID INT(11)
VolunteerID INT(11)
JoinedDate TIMESTAMP
Indexes
EventSkill
EventSkillID INT(11)
EventID INT(11)
SkillID INT(11)
Required TINYINT(4)
Indexes
Volunteer
VolunteerID INT(11)
VolunteerName VARCHAR(20)
VolunteerPassword VARCHAR(20)
PhoneNumber VARCHAR(20)
EmailAddress VARCHAR(45)
LastLoginDate DATETIME
Points FLOAT
HoursWorked INT(11)
AvgRating FLOAT
CreationDate TIMESTAMP
ActivationDate TIMESTAMP
AdminOrg INT(11)
Indexes
Event
EventID INT(11)
EventName VARCHAR(45)
Description LONGTEXT
Location VARCHAR(45)
BeginTime DATETIME
EndTime DATETIME
RecurID INT(11)
HoursBonus FLOAT
Indexes
Skill
SkillID INT(11)
SkillName VARCHAR(20)
Description MEDIUMTEXT
Indexes
PictureEvent
PictureEventID INT(11)
PictureID INT(11)
EventID INT(11)
Indexes
Interest
InterestID INT(11)
Name VARCHAR(45)
Description MEDIUMTEXT
Indexes
Picture
PictureID INT(11)
PictureLink VARCHAR(45)
UploadedOn TIMESTAMP
AlbumName VARCHAR(45)
Description VARCHAR(45)
Indexes
PictureOrg
PictureOrgID INT(11)
PictureID INT(11)
OrgID INT(11)
Indexes
PictureVolunteer
PictureVolunteerID INT(11)
PictureID INT(11)
VolunteerID INT(11)
Indexes
Notification
NotificationID INT(11)
Description LONGTEXT
Category VARCHAR(45)
Key VARCHAR(45)
Value VARCHAR(45)
Creation TIMESTAMP
DependenceDate DATETIME
Indexes
OrgGroup
OrgGroupID INT(11)
GroupID INT(11)
OrgID INT(11)
Indexes
PictureGroup
PictureGroupID INT(11)
PictureID INT(11)
GroupID INT(11)
Indexes
VolunteerDevice
VolunteerDeviceID INT(11)
DeviceID VARCHAR(45)
VolunteerID INT(11)
DeviceTokenInternal VARCHAR(45)
Indexes
VolunteerInterest
VolunteerInterestID INT(11)
InterestID INT(11)
VolunteerID INT(11)
Disinterest TINYINT(4)
Indexes



In this scenario a user is registering. As you can see there are many steps to register including many classes and technologies.


<<Java Class>>
VolunteerPage
org.CommunityService.ManagedBeans
volunteerEmail: String
allVolunteers: List<Volunteer>
VolunteerPage()
getVolunteerEmail():String
getAllVolunteers():List<Volunteer>
setAllVolunteers(List<Volunteer>):void
setVolunteerEmail(String):void
<<Java Class>>
DBConnection
hibernate
DBConnection()
query(String,List<String>):Object
persist(Object):Object
<<Java Class>>
HibernateUtil
hibernate
sessionFactory: SessionFactory
serviceRegistry: ServiceRegistry
HibernateUtil()
buildSessionFactory():SessionFactory
getSessionFactory():SessionFactory
<<Java Class>>
VolunteerService
org.CommunityService.Services
VolunteerService()
getVolunteerByName(String):Volunteer
getVolunteers():List<Volunteer>
addVolunteer(Volunteer):void
<<Java Class>>
UserValidator
org.CommunityService.Validators
UserValidator()
validate(FacesContext,UIComponent,Object):void
<<Java Class>>
HibernateListener
hibernate
HibernateListener()
contextDestroyed(ServletContextEvent):void
contextInitialized(ServletContextEvent):void

Security Measures

The security of our website is based on using a mixture of most of the measure mentioned previously as well.

The plan for password protection was to use both client and server side hashing, but ended up entirely on the server's shoulders due to some constraints of javascript we did not know about. The password is sent to the server where it is then be hashed for security reasons mentioned earlier. The salt we used on the hash is only used on the server and consists of a separate field that was added to the database user table. The field was a randomly generated number of a length equal to the 256 bit hash string length. A long and random string for salt is the most secure. We also iterated (keystreched) over the algorithm 1000 times for more difficult to crack hashes.

None of the previous measures affect the user, but some of the other ones do. The user is required to have a certain length password, with a certain level of complexity. The user also needs to have a password that is at least 8 characters long, and less than 15. The password needs to include a letter of each case, a number, and a symbol.

SSL is still in progress due to problems with obtaining a Domain.

Android

Android Design and Research Decisions

Android Goals and Objectives

To allow the user easy access to their volunteer plans when they are not in front of a computer. The app does not have all the features as the website but has the vital features such as finding a volunteer event or seeing what the user already signed up for. As mobile apps are becoming increasingly popular, we decided to make it a mandatory instead of optional objective. Leaving it out would leave the project feeling incomplete.

Significance

The Android app is important and plays a significant role in the overall project simply because most people browse the internet via their phones nowadays instead of at a desktop or laptop computer. As this trend continues into the future (which we predict it will) the Android app may become even more significant and it is role expanded not just to find volunteer events already signed up for and search the database, but to edit profile information and create new volunteer events.

Although there is no iPhone version planned at this point, we may in the future. The primary obstacle is cost, since development would require a Mac laptop and an iPhone, neither of which anyone in our group has at the moment.

We may also need to consider support for tablets. Designing for a phone gives automatic tablet support. The app should simply run on both platforms without changes, but we may decide to reconfigure the user interface so the app looks better on a tablet in the future.

Research and Investigation

Before working on the Android app we had to do quite a bit of research. Google Images provides a generous number of screenshots of Android apps that use the Holo Theme, so we were able to fairly easily compare various themes. Since the user interface looks consistent across apps, the screenshots also gave us an idea of what our app might look like.

Another advantage is seeing how other people and companies design their apps. We took inspiration from some of the larger more popular apps such as Facebook which uses more of the recent Android SDK features such as the navigation drawer.

Overall, the research involving the Android portion really was not that difficult or extensive. Luckily, Google already provides a theme that looks slick and modern, something we all could agree on easily.

The web service portion took more research and actually turned out to be significantly harder than the user interface. One would think something as simple as connecting to the internet would be trivial, but due to security issues on Android, it is not. This is one area Google could improve on. The current method of extracting data from a database over the internet is clumsy and more involved than necessary. JBoss has not announced a Hibernate version for Android but as the platform becomes more popular, we hope to use better tools.

We considered using the Simple library for XML serialization but after research found only a minimal amount of information and tutorials, seeing how the technology is still new. So that was the deciding factor against it. Instead, we went with a more classic approach involving servlets and GSON which has more documentation available online.

Design Approach and Implementation

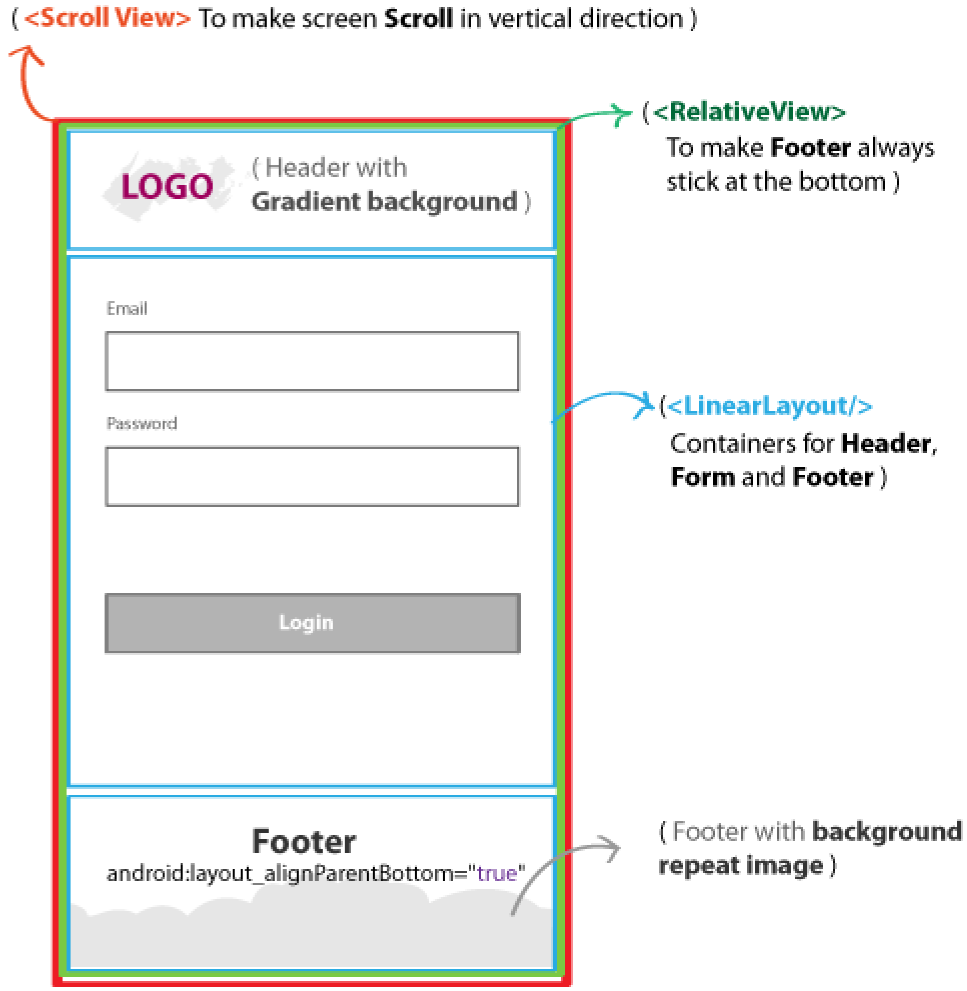
Our approach for the Android portion is simple, easy to use and has a graphical theme that complements Android’s native interface. That is why we decided on the Holo Light theme which is the newest offered by Google!

Keeping in mind this is an Android app and not a website, we did not want to just copy the exact look and feel of the website. Instead, we wanted to combine the style of Android with design aspects of the website such as color scheme. We did this by using Android classes to implement the features, such as the ListView class and the Navigation Drawer. This gives the app a different feel from the website, which is not necessarily a bad thing. If users prefer one over the other, it gives them the option of which to use.

Design

Here are mock ups of the Android app we used to plan the user interface and to help make implementation decisions.

Merging Multiple Layouts



(<Scroll View> To make screen **Scroll** in vertical direction)

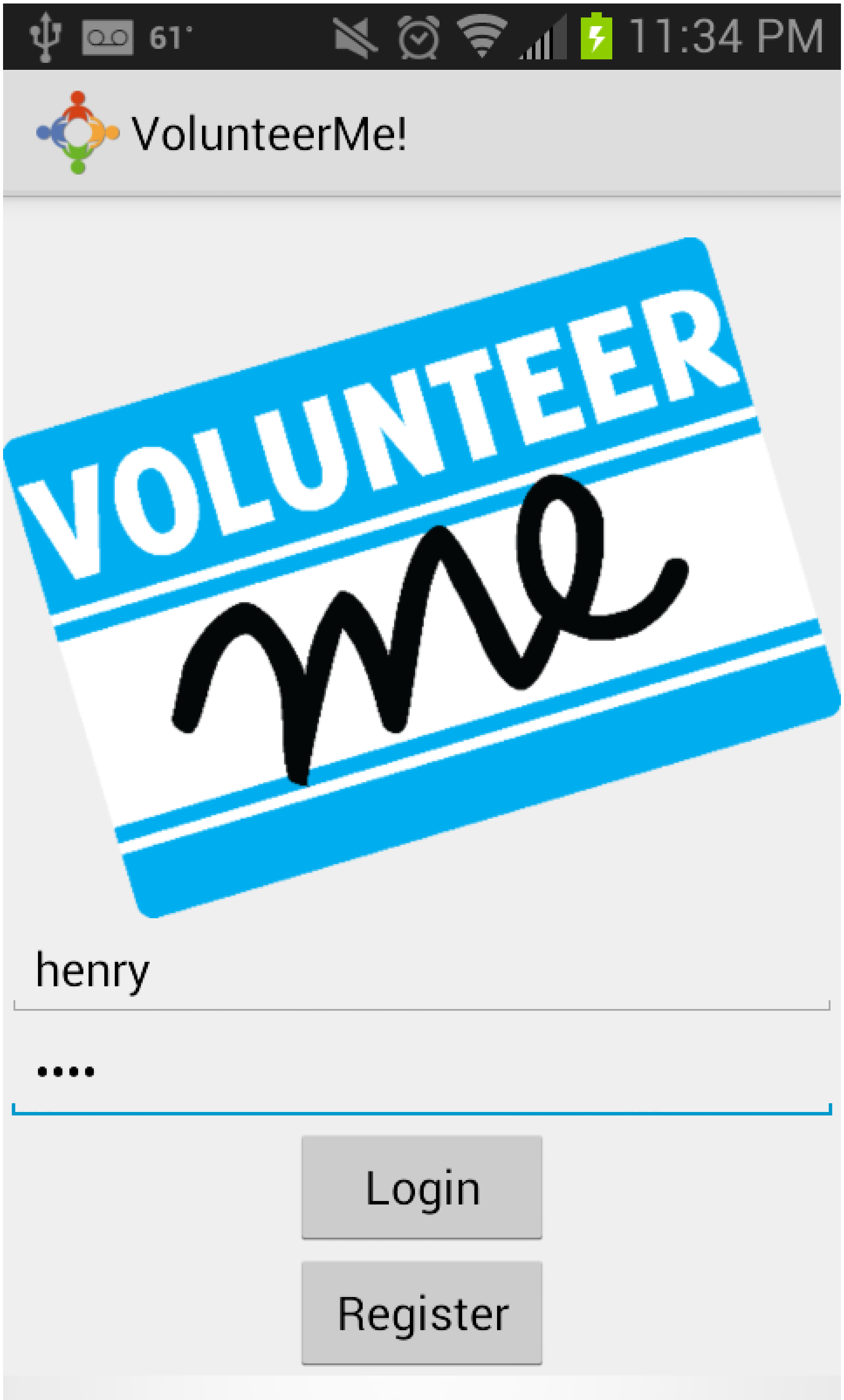
(<RelativeView> To make **Footer** always stick at the bottom)

(<LinearLayout/> Containers for **Header**, **Form** and **Footer**)

(Footer with **background repeat image**)

LOGO
(Header with **Gradient background**)
Email
Password
Login
Footer
android:layout_alignParentBottom="true"

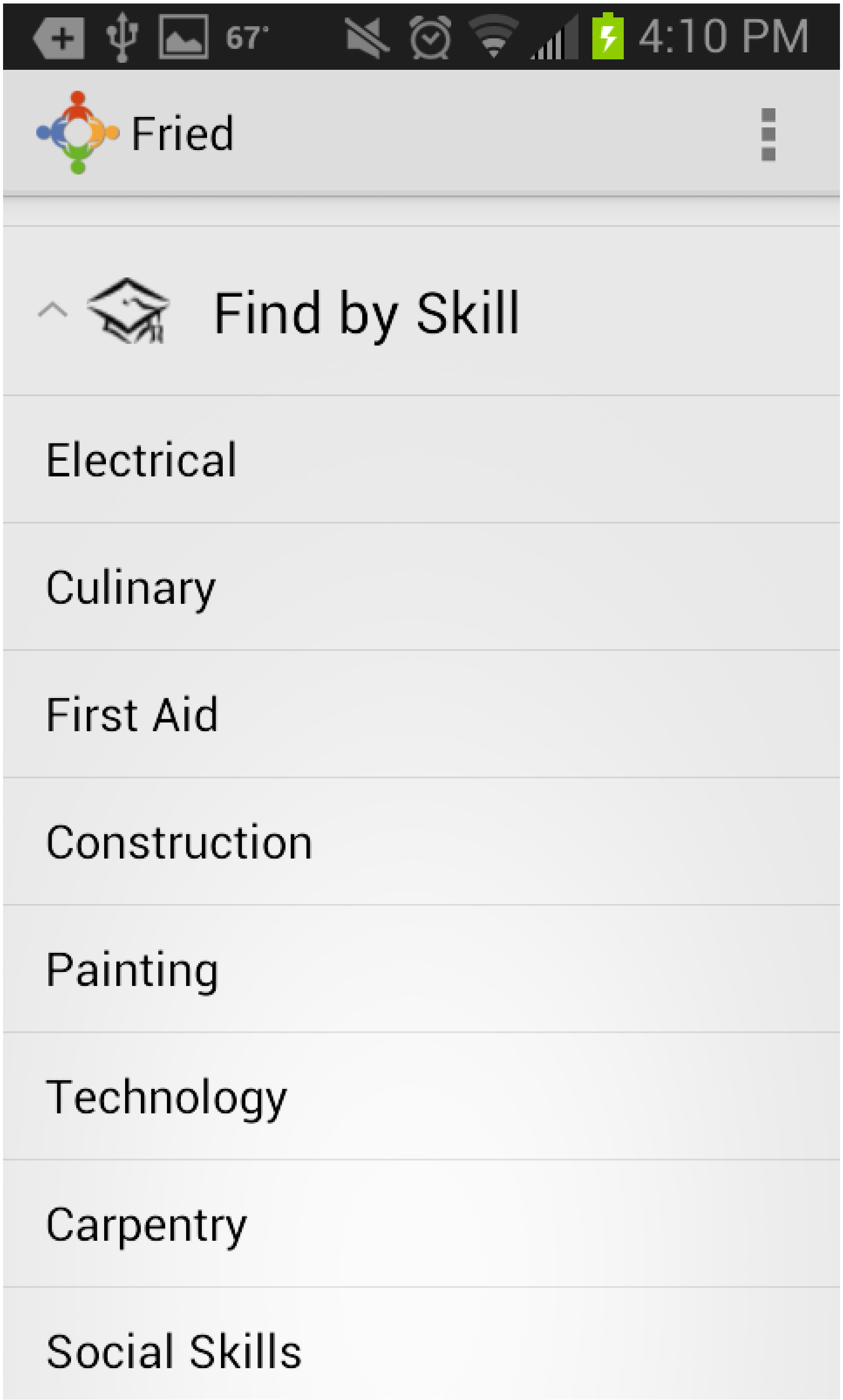
androidhive



henry

••••

Actual screenshot!! Awesomeness!!



Find by Skill

Electrical

Culinary

First Aid

Construction

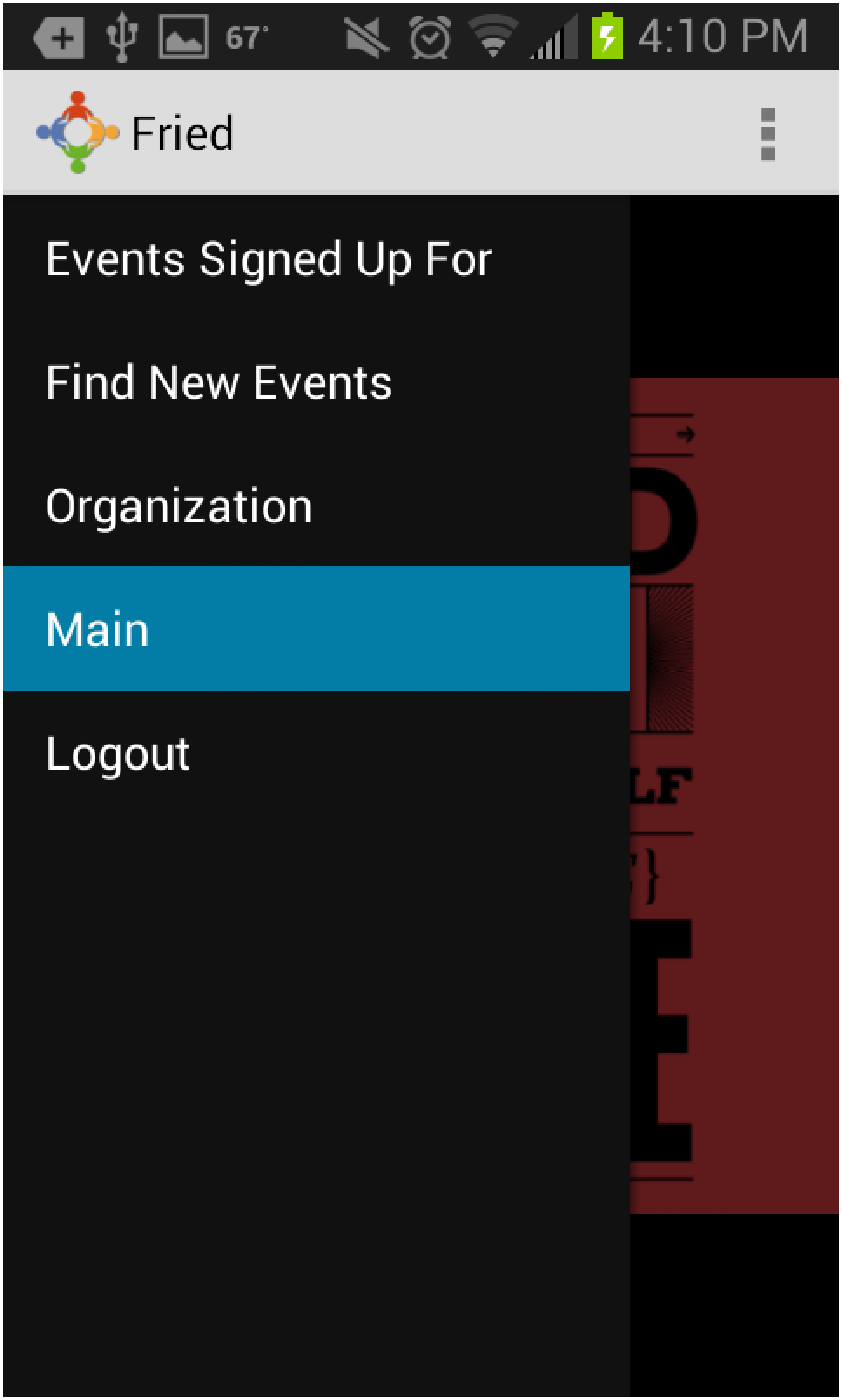
Painting

Technology

Carpentry

Social Skills

An example of the Find feature that lets the user find volunteer organizations. The feature will also let the user find events they already signed up for. The programming code uses a ListView object.



67°

4:10 PM

Fried

Events Signed Up For

Find New Events

Organization

Main

Logout

The user can navigate by swiping a side menu that slides out. This style is found in newer apps such as Google Maps and Facebook. The object in code is called a "navigation drawer".

Android UI Decisions

There are a few popular android themes, but we chose the Holo light instead of the dark because we feel volunteering and community service has a positive energy about it and would not go well with a dark theme. We used a teal / light blue color as a complement to the white. Icons also provide pops of color.

Android Components & Specifications

The app uses the following technologies:

- Java
- Eclipse
- XML
- Holo Light Android Theme
- JAXB or SimpleXML for parsing Java objects from XML stored on the server. Unfortunately, we discovered Android has no built in support for JAXB and using it (if we could get it working at all), would add 9 MB to the app size, which is due to the large size of the JAXB library. Our research shows SimpleXML is commonly used as an alternative on Android and has features we need such as:
    - XML serialization.
    - Read and write objects to and from XML.
    - Serialization that supports recursion.
    - XML templates

Activity

An activity is like a webpage on a website but in this case a “page” or “screen” in an Android application. Just about all activities interact with the user. The activity handles window placement and the user interface that goes inside of the window. Our activities are full screen, but they can also be presented in different styles like floating windows. Most Android apps start with the onCreate() method which handles initialization of variables, etc. The view is then set, by passing an xml layout. Activities are a vital part of the lifecycle of an Android application. Activities define the structure and ultimately dictates the behavior of the app (“Activity”).

Fragments

A Fragment is a class that defines the behavior of part of the user interface of an activity. More than one fragment makes up a multi pane user interface. The big advantage of Fragments is they can be reused across activities. Fragments are modular. They have their own input events, lifecycle and can be added or removed while an activity is running.

Fragments are embedded in an activity and take cues from the activity. For example, when the activity is paused, so are all the fragments. The back button on the phone can be used to reverse a fragment’s navigation.

Fragments live in an object called a ViewGroup and is defined in an xml file that describes the layout (“Fragments”).

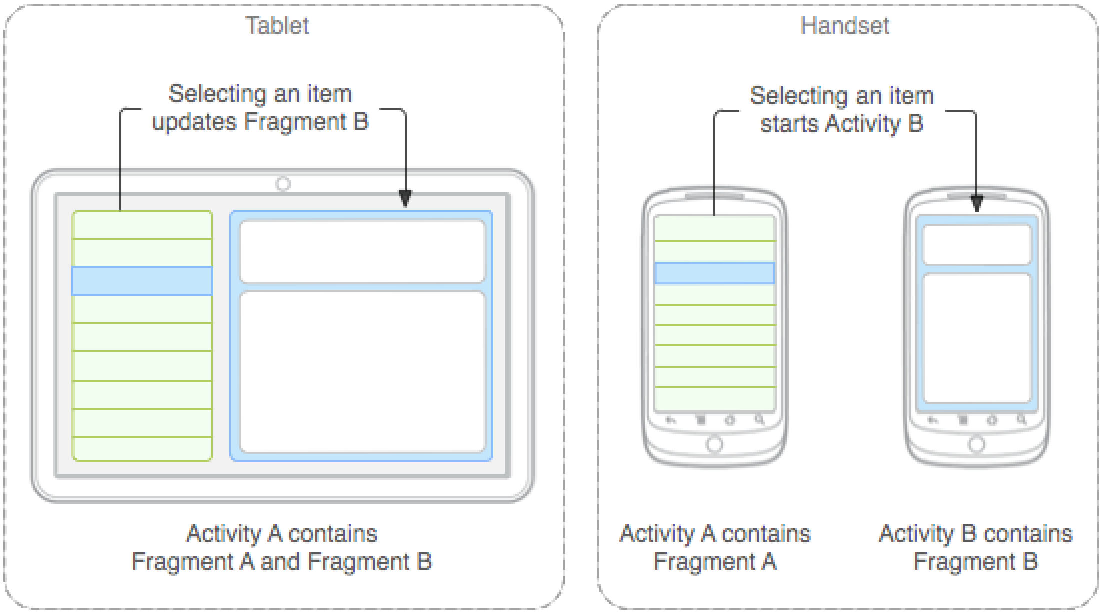
Fragment Design Philosophy

Fragments are part of Android API level 11 which is below the minimum we are supporting, so no problem there. Fragments were introduced to make user interfaces more dynamic and flexible, especially on larger screens such as tablets. Although we are not developing for tablets, our app is made up of fragments:

- Find Volunteer Events Fragment
- Current Events Fragment

Although the navigation drawer is not a fragment, it behaves in a similar way, remaining on the screen statically across different activities. Our aim was to make each fragment as modular and reusable as possible. The benefit is the same fragment can be used in multiple activities without changing the fragment code. This is important when designing for different screen sizes, such as different phones.

(“http://developer.android.com/guide/components/fragments.html”)


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Shared Preferences

This Android feature is used to save data despite the misleading name, for example, username and password data.

How secure is it?
The file is by default, stored within the app's directory with permissions set to only allow the UID (Android app ID) access to the file. A UID on Android has permissions similar to Linux.

Anyone with root level access will be able to see the file. Also, if someone manages to mount the device without using the installed Android OS, they could bypass the permissions and access the file.

We decided not to encrypt the save file because it is already private. Anyone rooting the phone is violating Android terms and conditions, which is usually the user anyway. Even if a person manages to see the data, we are not storing anything sensitive such as credit card information. The most personal information would be phone and email address ("Shared Preferences").

HttpURLConnection

The HttpURLConnection Android class allows data to be sent and received over the internet. The class can be of any type and length, as well as streaming data. It is also possible to establish a secure connection by using a X509TrustManager for confirming certificates and X509KeyManager for supplying client certificates. It is possible this class may not work as well on older phones due to the proxy not being configured correctly. We do not support the older phones (2.X) anyway due to limitations on user interface, so this is not a big concern for us. This class is Google's "preferred" way of accessing the internet and also a common one ("HttpURLConnection").

AsyncTask

This important class lets the app perform background operations on a separate thread without freezing the user interface. For example, we may use it to load data from the server while the user navigates a menu. AsyncTask is usually tied to an activity (which is like a webpage on a website). We may use the Service class for anything that takes a significant amount of time or global in scope. Other possible uses of AsyncTask in our app (“AsyncTask”):

- Updating a progress bar
- Downloading from the internet
- Loading a webpage

JSON

JSON stands for JavaScript Object Notation. JSON is lightweight and the format handles data exchange. JSON also is supposed to be easy for humans to write and easy for computers to parse. The format is based on Javascript standard ECMA-262 Edition. Although JSON is language independent, it uses coding conventions not unlike C family languages. JSON is built on two concepts.

- Name / value pairs similar to what is used in a hash table.
- And ordered list similar to an array.

JSON uses simple data structures to make sure all modern programming languages support them. We do not really have a choice and have to use JSON since we are using GSON as well (“Introducing JSON”).

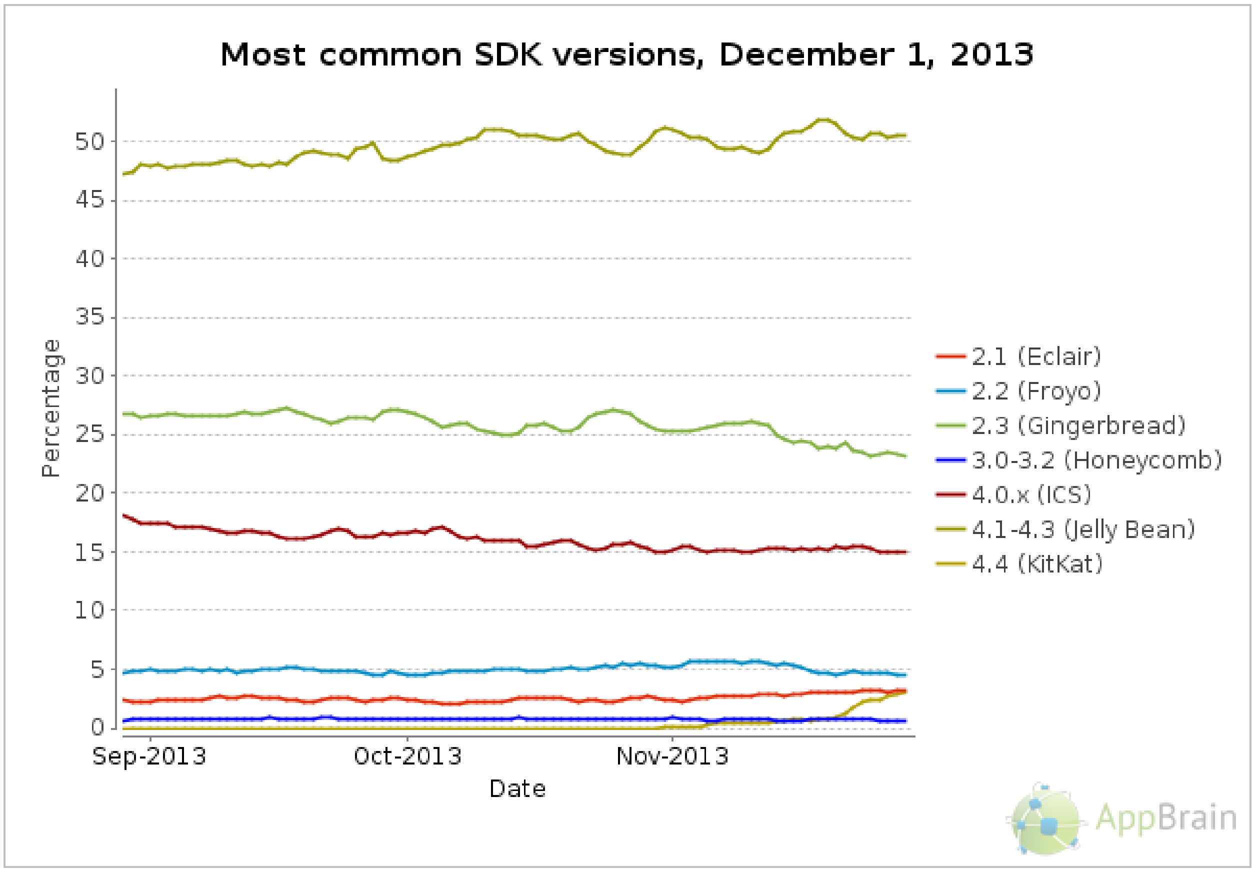
Gson

A completely different technology we are researching and considering instead of JAXB or SimpleXML is Gson. GSON was developed by Google for internal use. Instead of mapping to XML, it maps Java Objects to Javascript Object Notation, thus providing an alternate way of passing data to Android, which is probably the easiest and simplest option and does not involve tedious parsing XML like the above two technologies (“GSON”).

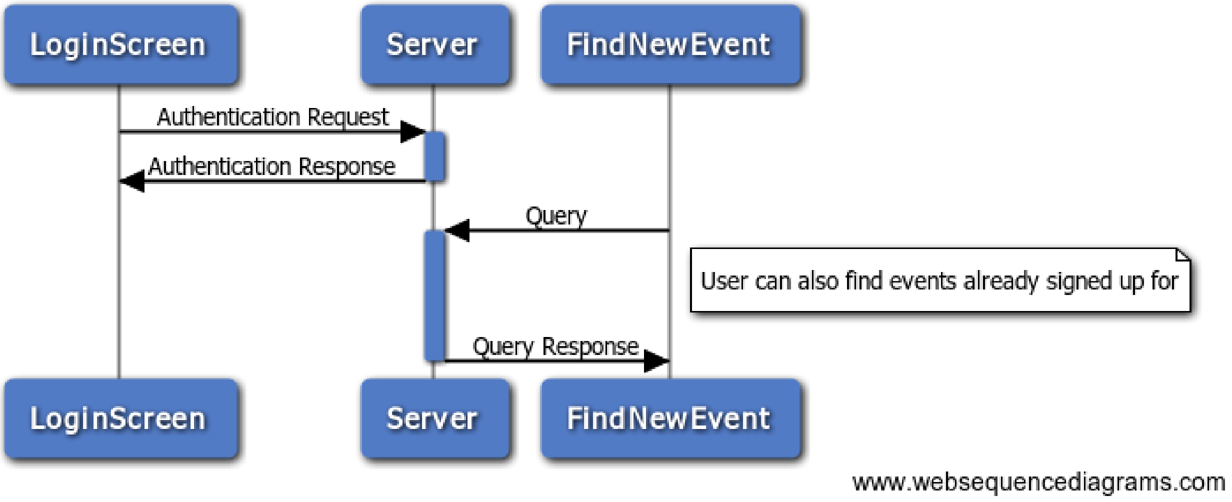
Android Requirements

The minimum Android API level required to run the app is API level 11, which is Honeycomb (Designed primarily for tablets). The reason for this is the Holo Light Theme is not compatible with older phones. The app's target API level is 18, Jellybean. Recent statistics show ~50% of Android phones run Jellybean and ~15% run Ice Cream Sandwich.

("http://www.appbrain.com/stats/top-android-sdk-versions")



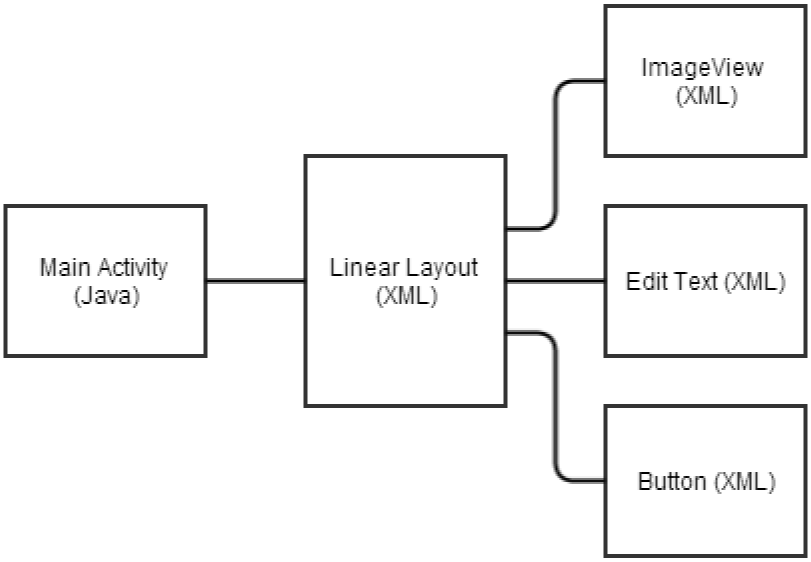
Android Login and Find Sequence Diagram



User can also find events already signed up for

www.websequencediagrams.com

Login Screen Class Organization



Find Feature

The user will be able to find a volunteer organization by:

- Interest
- Skills
- Event Date/Time

The results is a list containing the name of the organization and a short description which will be expanded on when the user selects it.

The second find feature are events the user already signed up for. The feature is displayed in a similar way as the first find feature, except it is sorted by time, with the most recent event listed first. Basic information is displayed, such as:

- organization name
- event time
- description.

Android Server Goals and Objectives

Android can not use Hibernate or get data off the server directly, so another project must be created which acts as a web service. Android communicates with the web service, passing an id and key to a servlet, something like a hash table entry. The web service confirms the key and uses the id to query the MySQL database. The results are passed to data members in the Event class. The class instance is then converted to a JSON string. All the data members are together in a single string which is returned to the original Android App. The app has the job of parsing the string to extract the data members and put them back in the Event class. This is where GSON comes in. The process is surprisingly complicated for such a simple concept as connecting to the internet and pulling data off a server via Android.

Key Points:

- Create a new project that includes a servlet
- Run a MySQL query
- The web service returns the MySQL table rows in a single string
- This string format is called JSON
- Use GSON to convert to a Java object

Android Server Design Decisions

We had to do hours of research to figure out how to implement the web service. Most projects use a PHP script to query MySQL but since our Tomcat server does not support PHP we decided to do something similar with a servlet. We had to consider security. The problem is the servlet could potentially be run outside of the Android app but since it only returns public information, such as volunteer event information, it is not really a concern. The servlet requires a volunteer ID anyway, which is not available off the website.

Java Servlet For Android

The servlet is a Java class that adds capabilities to the server. In regular use, servlets extend the applications hosted on the server. In a way, servlets are similar to Java Applets, but on a server instead of a web browser. Comparing technologies, a servlet is comparable to a PHP and ASP.NET script.
Servlets are commonly used to:

- Store or process data from an HTML form.
- Provide content that is dynamic, for example database query results.
- Manage state information that can not be found in the HTTP protocol. An example would be filling items in a customer’s shopping cart.

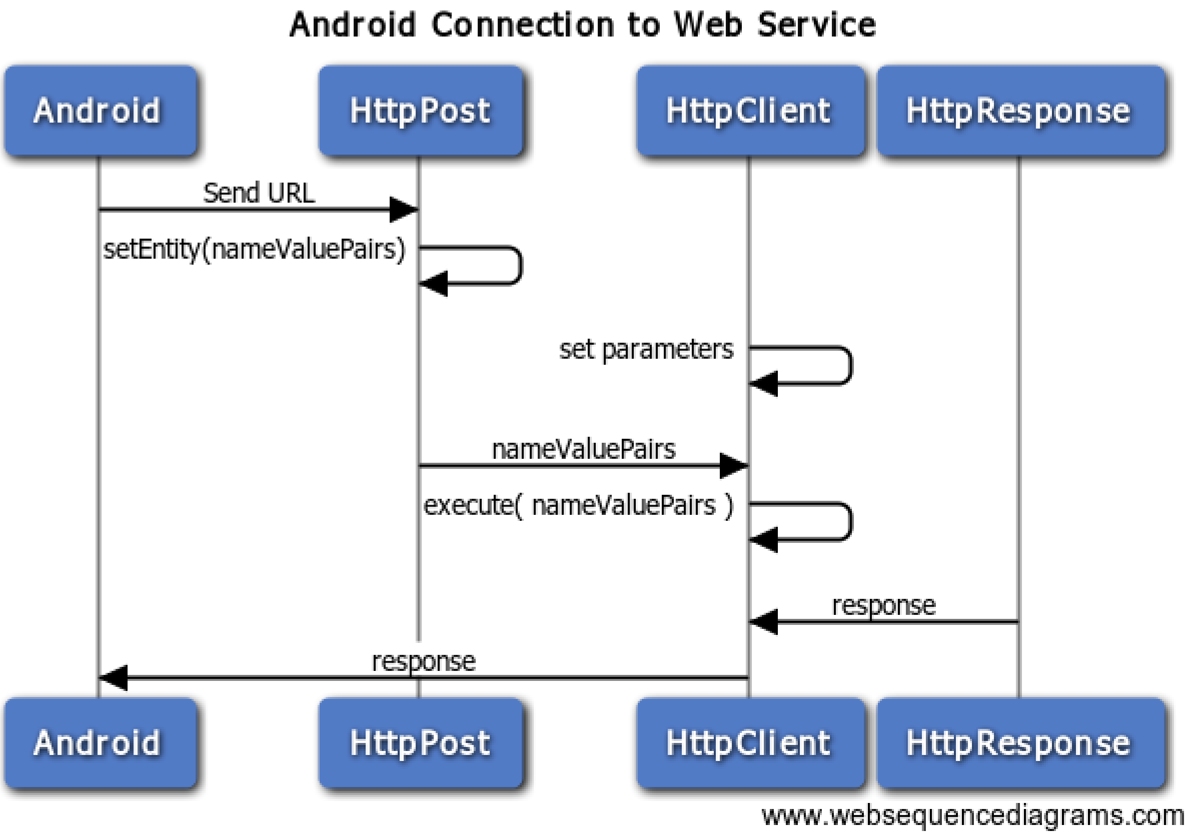
The Java Servlet API is a standard for implementing classes that handle requests. Servlets are commonly used with the HTTP protocol, allowing a developer to add content that is dynamic to a server using Java. The content generated is usually HTML, but could also be XML. Servlets maintain the session state in variables by using cookies or URL rewriting (“Java Servlet”).

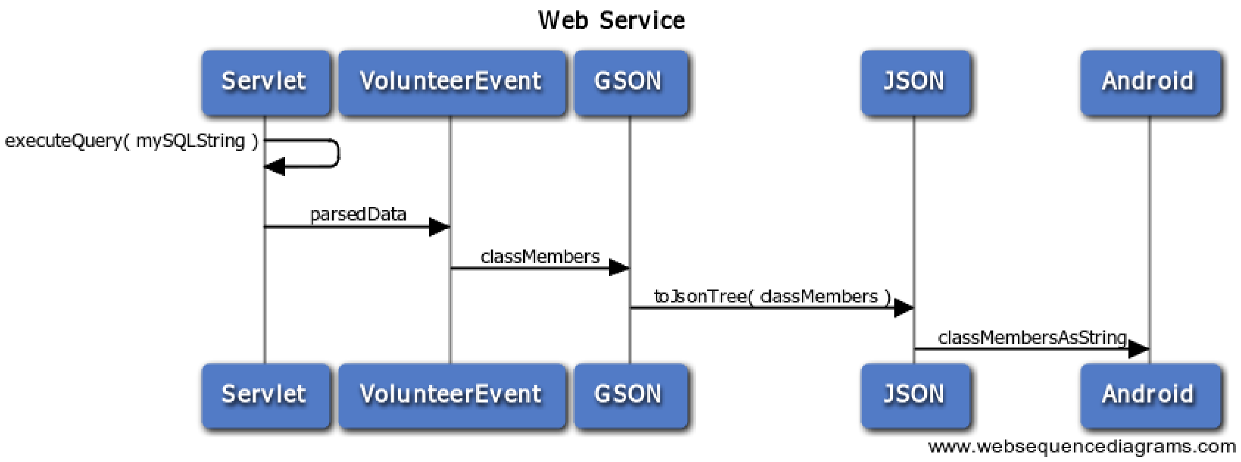
Sample Code

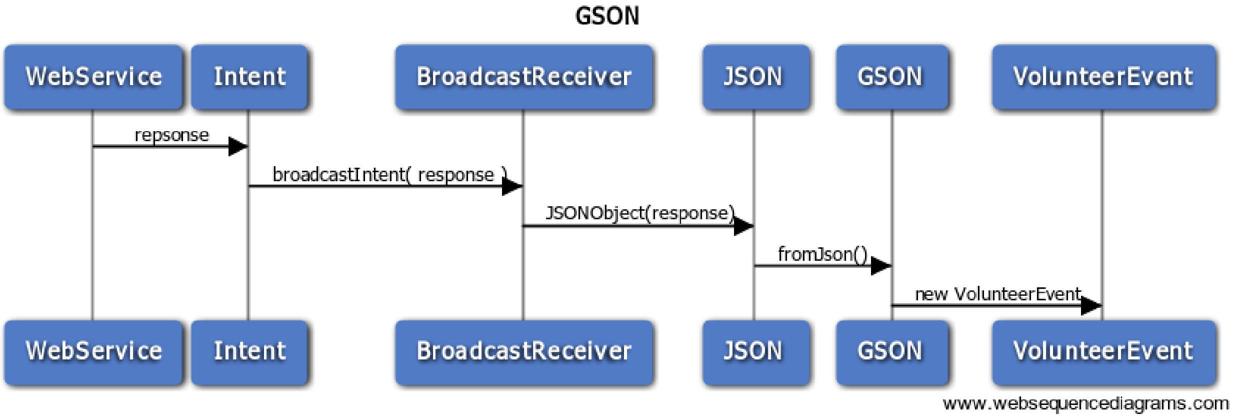
```
public class EventDataServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
     public EventDataServlet() {
        super();
    }
     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request,response); //request response is a data set similar to hash
key
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
         PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        response.setHeader("Cache-control", "no-cache, no-store");
        response.setHeader("Pragma", "no-cache");
        response.setHeader("Expires", "-1");
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "GET,POST");
        response.setHeader("Access-Control-Allow-Headers", "Content-Type");
        response.setHeader("Access-Control-Max-Age", "86400");
        //get information based on id
        EventDataGetter getter = new EventDataGetter();
        EventData event = getter.getInfo(eventID);
         //if invalid id
        if(event.getEventId() == null){
            JsonObject myObj = new JsonObject();
            myObj.addProperty("success", false);
            out.println(myObj.toString());
        }
         //if a valid id was sent
        else {
            Gson gson = new Gson();
            //creates json from EventData object
            JsonElement eventObj = gson.toJsonTree(event);
            //create a new JSON object
            JsonObject myObj = new JsonObject();
            //add property as success
            myObj.addProperty("success", true);
            //add the event object
            myObj.add("eventInfo", eventObj);
            //convert the JSON to string and send back
            out.println(myObj.toString());
        }
        out.close();
    }
}
```

The next code segment includes the MySQL query

```
public class EventDataGetter {
     Connection conn = null;
    PreparedStatement stmt = null;
    String sql = null;
        public EventData getInfo(String id) {
        EventData dat = new EventData();
         try {
            Context ctx = (Context) new InitialContext().lookup("java:comp/env");
            conn = ((DataSource) ctx.lookup("jdbc/mysql")).getConnection();
            sql = "SELECT * FROM mydatabase.event WHERE eventID = ?";
            stmt = conn.prepareStatement(sql);
            stmt.setString(1,id);
            ResultSet rs = stmt.executeQuery();
    //put data from server in an Event class instance
            while(rs.next()){
                dat.setEventId( id );
                dat.setEventName(rs.getString("eventName").trim());
                dat.setBeginTime(rs.getString("beginTime").trim());
                dat.setLocation(rs.getString("location").trim());
                dat.setDescription(rs.getString("description").trim());
            }
            rs.close();
            stmt.close();
            stmt = null;
            conn.close();
            conn = null;
        }
        catch(Exception e){System.out.println(e);}
        finally {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlex) {
                    // ignore -- as we can't do anything about it here
                }
                stmt = null;
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException sqlex) {
                    // ignore -- as we can't do anything about it here
                }
                conn = null;
            }
        }
        return dat;
    }
}
```


Android Connection to Web Service
Android
HttpPost
HttpClient
HttpResponse
Send URL
setEntity(nameValuePairs)
set parameters
nameValuePairs
execute(nameValuePairs)
response
response
Android
HttpPost
HttpClient
HttpResponse
www.websequencediagrams.com


Web Service
Servlet
VolunteerEvent
GSON
JSON
Android
executeQuery(mySQLString)
parsedData
classMembers
toJsonTree(classMembers)
classMembersAsString
Servlet
VolunteerEvent
GSON
JSON
Android
www.websequencediagrams.com



IntentService

The Android app uses the IntentService class, which handles asynchronous requests. A request is sent through the function, startService(Intent). Each Intent is handled by a worker thread, and automatically stops once completed (no work left to do). This programming pattern is called “work queue processor” and is used to offload tasks from the main thread of an application. IntentService makes all this simpler and easier, by taking care of much of the mechanics. The android app extends this class and implements the function, onHandleIntent(Intent). Only one request is processed at a time and may take as much time as needed. This way, it does not block the main thread’s loop (“Intent Service”).

BroadcastReceiver

This class is used by Android and receives intents that are sent by the function, sendBroadcast(), allowing for cross process communication. Main classes of broadcasts are:

- Normal broadcasts: These are asynchronous. Often all receivers of the broadcast are run at the same time. However the receiver can not use the result or abort the API.
- Ordered broadcasts: These are sent to one receiver at a time. When received, it is executed in turn, possibly sending the result to the next receiver. The order the receivers run is specified in android:priority. If the priority is the same, they are run in an arbitrary order.

If the receiver creates a process, only one process runs at any given time to avoid overloading the Android operating system. Launching an activity with an intent is

done in the foreground. Permission access is enforced by either the sender or receiver. If done when sending, a permission argument is entered in sendBroadcast(). If done when receiving, a permission argument is passed when registering the class.
For long running operations, a service is normally used in conjunction with a BroadcastReceiver so the process remains active during the life of the application (“BroadcastReceiver”).

BaseExpandableListAdapter

This class is used to give data to an expandable list view object. In other words, it populates the list of Events found when the user does a search. In our project, the BaseExpandableListAdapter is inherited and built upon. A number of functions have to be overloaded (“BaseExpandableListAdapter”):

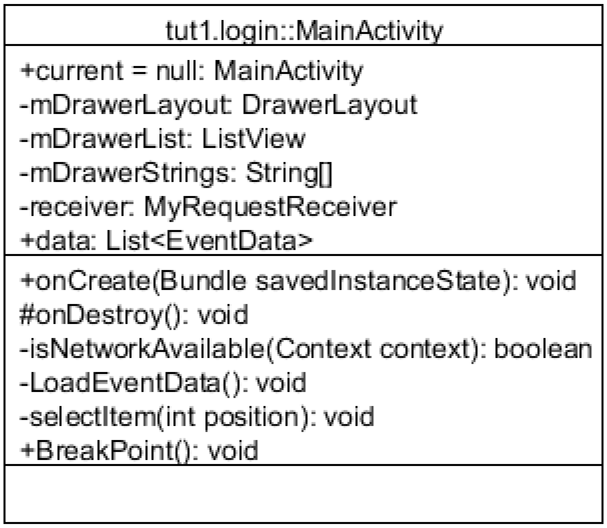
- getGroup()
- getChild()
- getGroupView()
- getChildView()

XML

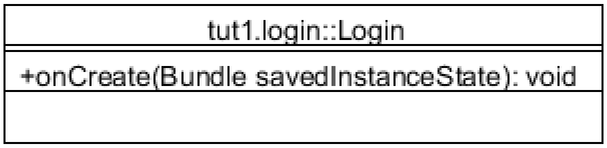
Sample xml code that defines the navigation drawer which is the menu that slides out when the user swipes across the screen:

```
<!-- A DrawerLayout is intended to be used as the top-level content view using
match_parent for both width and height to consume the full space available. -->
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- As the main content view, the view below consumes the entire
         space available using match_parent in both dimensions. -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- android:layout_gravity="start" tells DrawerLayout to treat
         this as a sliding drawer on the left side for left-to-right
         languages and on the right side for right-to-left languages.
         The drawer is given a fixed width in dp and extends the full height of
         the container. A solid background is used for contrast
         with the content view. -->
    <ListView
        android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

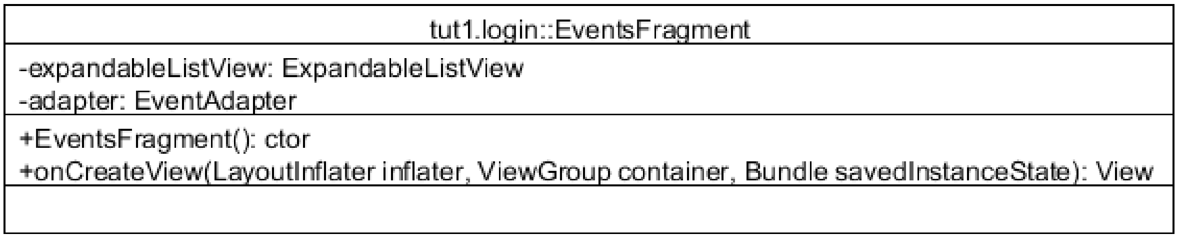
Android Class Diagrams



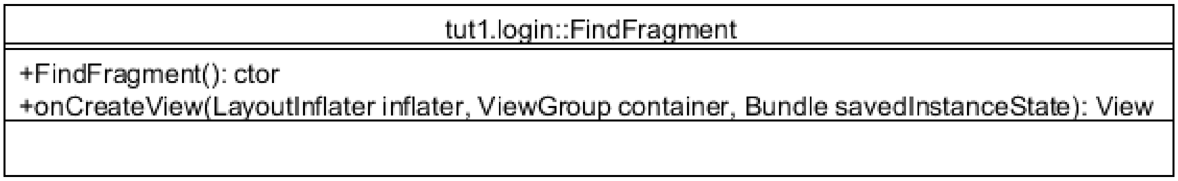
```
tut1.login::MainActivity
+current = null: MainActivity
-mDrawerLayout: DrawerLayout
-mDrawerList: ListView
-mDrawerStrings: String[]
-receiver: MyRequestReceiver
+data: List<EventData>
+onCreate(Bundle savedInstanceState): void
#onDestroy(): void
-isNetworkAvailable(Context context): boolean
-LoadEventData(): void
-selectItem(int position): void
+BreakPoint(): void
```



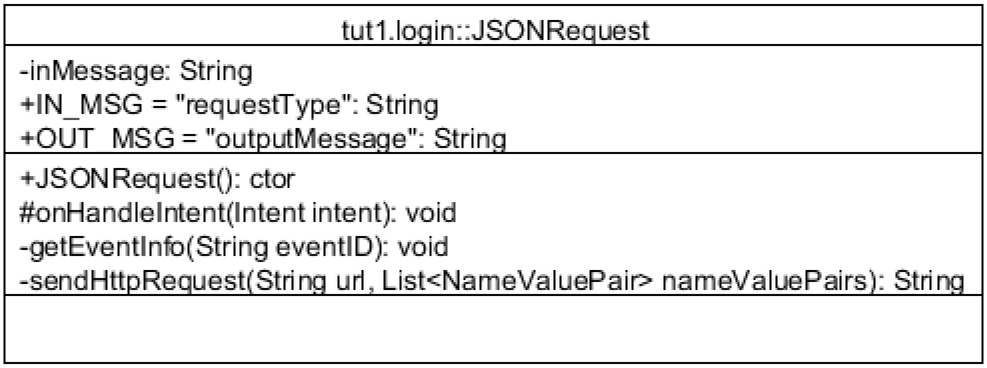
```
tut1.login::Login
+onCreate(Bundle savedInstanceState): void
```



```
tut1.login::EventsFragment
-expandableListView: ExpandableListView
-adapter: EventAdapter
+EventsFragment(): ctor
+onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View
```

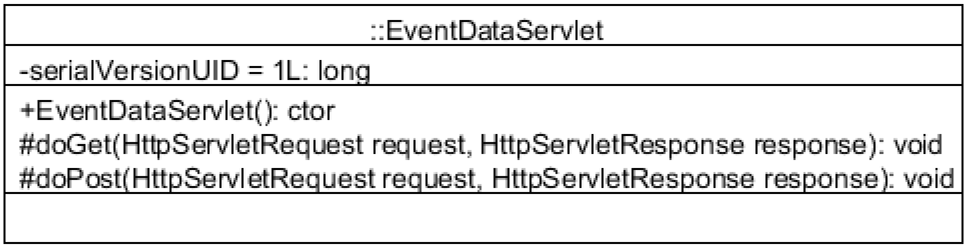


```
tut1.login::FindFragment
+FindFragment(): ctor
+onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View
```

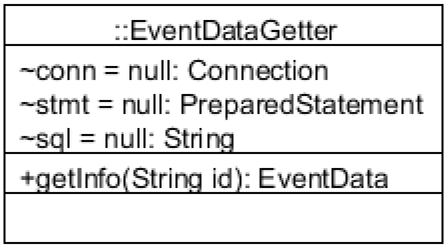


```
tut1.login::JSONRequest
-inMessage: String
+IN_MSG = "requestType": String
+OUT_MSG = "outputMessage": String
+JSONRequest(): ctor
#onHandleIntent(Intent intent): void
-getEventInfo(String eventID): void
-sendHttpRequest(String url, List<NameValuePair> nameValuePairs): String
```

Web Service Class Diagrams



```
::EventDataServlet
-serialVersionUID = 1L: long
+EventDataServlet(): ctor
#doGet(HttpServletRequest request, HttpServletResponse response): void
#doPost(HttpServletRequest request, HttpServletResponse response): void
```



```
::EventDataGetter
~conn = null: Connection
~stmt = null: PreparedStatement
~sql = null: String
+getInfo(String id): EventData
```

Administration And Overall Choices

Software and Web Applications/Service Used

- Eclipse - Used for Java, JSF, XML, Javascript, Android, and CSS programming.
- Android Virtual Device Manager - Used to test the Android code on a virtual Android device.
- Vizio - Used for diagrams for this document and for internal discussion use.
- Chrome - Used for testing the deployed website. Also used for the developer tools built in to help debug problems, and test css changes.
- Internet Explorer - Used for testing deployed website.
- FireFox - Used for testing deployed website
- Github - Used for revision control of the project. Also being used for the issue tracking system built in.
- Git for Window - Used to make syncing and such with github easier on windows.
- MySQL WorkBench - Used for SQL database management
- Hibernate Utils Eclipse Plug In - Used for generating hibernate entities and mappings.
- Putty - Used for SSHing into the Amazon EC2 instance for management
- FileZilla - Used for transferring files to the server.
- Winscp - Used for SSHing and transferring files to server.
- Illustrator/Photoshop - Used for making/editing images to be used on the website.
- Windows Operating System - Used for work environment OS
- Linux(Ubuntu) Operating System - Used as the server OS.

Logos

Here are some possible logos we took inspiration from.

(public domain image)

(public domain image)

(public domain image)

Progress and Plans

Web Application

The web application made good progress early in the development cycle. Here is a list of major milestones.

- EC2 instance created
- Tomcat installed and running on EC2 instance
- RDS instance created
- Tables created in the RDS database
- Entities classes created
- Hibernate configured to the database
- Hibernate mappings created between entities and database
- Registration pages created with JSF and primefaces configured
- Login page created
- DB connections established and tested with entities persisted from pages
- Landing page created

Android

The Android app was up and running, with the XML and Java code for the log in screen complete during the first semester. However at that point, it did not actually connect to the server yet.

The app displays a navigation drawer. The side menu expands when the user swipes the screen. For awhile, there were only a handful of other options such as the Events and Find features.

When the user selects the Events option, the activity loads test data. In the beginning it did not use the database data, it just used test data, to make sure the user interface worked properly. At present, of course, it does in fact load all data from the server. This was accomplished early in the second semester.

The Find feature displays a list of options such as find by date or interest. This was a little more complicated because events are likely to change during the app's use, while the events already signed up for are static (unless the user signs up for another event during app use). So in other words, new queries had to be run. Events can not really be cached.

There was generous testing involved. One thing I hate about using Eclipse for Android programming is the generic error messages the programmer receives. For example, it might crash due to a null pointer exception, yet does not point to the line the error occured on because half the time it occurred in some class buried in the API, or even if that is not the case, it still will not point to the line. So testing is a little tedious.

Android Web Service

The web application successfully deploys and returns a string which represents the data from a row in the MySQL database. Tesing for the web service was done on the main server.

JSON was not too difficult to set up. The code follows a predictable pattern when using, so much of it is rinse and repeat. JSON takes database row information and uses GSON to do the appropriate conversion.

Main development points for the web service included:

- Converting from GSON back to an Event object
- Feeding the object data to the listview object
- Making sure it actually displays the results in the menu
- Making sure it displays all results (such as multiple events)

Project Expected Difficulties

Android Component Challenges

Expected

What made it difficult is the number of components and technologies involved. There is the main Android App itself, the web service project, JSON, GSON, MYSQL, Tomcat 7 server, among other things. So although there are samples and tutorials online, most are missing one or two components. The few that are not do not provide source code to download and test because there’s just too much going on. Even if the source code was provided, everything would have to be configured correctly to get it to run.

Luckily, Android has a robust community of developers, and that means easy access to tutorials and support. So the coding for the user interface was fairly easy and a definite success. Since not all of us have experience writing code that communicates with servers, that was perhaps the hardest part, as the technology is quite complex and is not just one technology but a host of technologies working together. Another difficulty was deciding and agreeing on what features should be added at a later time, which was very limited. None of us have used Gson, but it turned out to not be too time consuming.

Difficulties Faced

While testing, we had trouble connecting Android to the server, then connecting to the web service. After investigation, the problem turned out to be the keyword “localhost” which is used to access a server on the development machine. While this works for a Java application, it does not work for Android because localhost ends up referring to the Android phone and not the development computer as expected. So instead of looking for a server on the development computer, the code looked for one on the phone.

The problem we faced in the Android web service was the number of technologies that play a role in doing ultimately a simple task: running a MySQL query and returning the results. The technologies include JSON, GSON, Java Servlet, MySQL among others. Finding a tutorial online that covered all of these hurdles was challenging, so we had to find various tutorials and piece them all together. This was a learning experience as well as a test of patience. Of course, doing a MySQL query is not exactly a “new” problem so there was decent information regarding the subject online.

The biggest challenge was stepping through the program using the debugger and figuring out what various errors meant, many of which indicated not one, but various possible problems, so it was a game of trial and error when finding a solution.

After using Eclipse extensively to develop for Android, we have to say some of us do not like it. Eclipse tends to crash at least once during every session. Sometimes Eclipse loses contact with the phone that is connected via USB (and used to run the actual app). Every time it loses contact, a restart of the entire computer is required. Surprisingly, the error messages are more generic than other languages, such as C++. This may not be true when programming a traditional Java application. But it is for an Android application because most of the code is part of the Android API.

Web Application Challenges

Expected

We expect that one of the hardest parts of this project will be error catching. Error catching is major part of big websites and it is a rarely seen part so it is not very gratifying. Error catching means we had to make sure that we cover a vast array of possibilities, including data inconsistencies, bad form data, and even server problems.

Another problem was Javascript programming. Javascript makes it difficult o find errors due to the fact it is not statically typed, and has very few debugging tools. When you get a javascript error, you often spend a long time looking for something very small, like a missing semicolon.

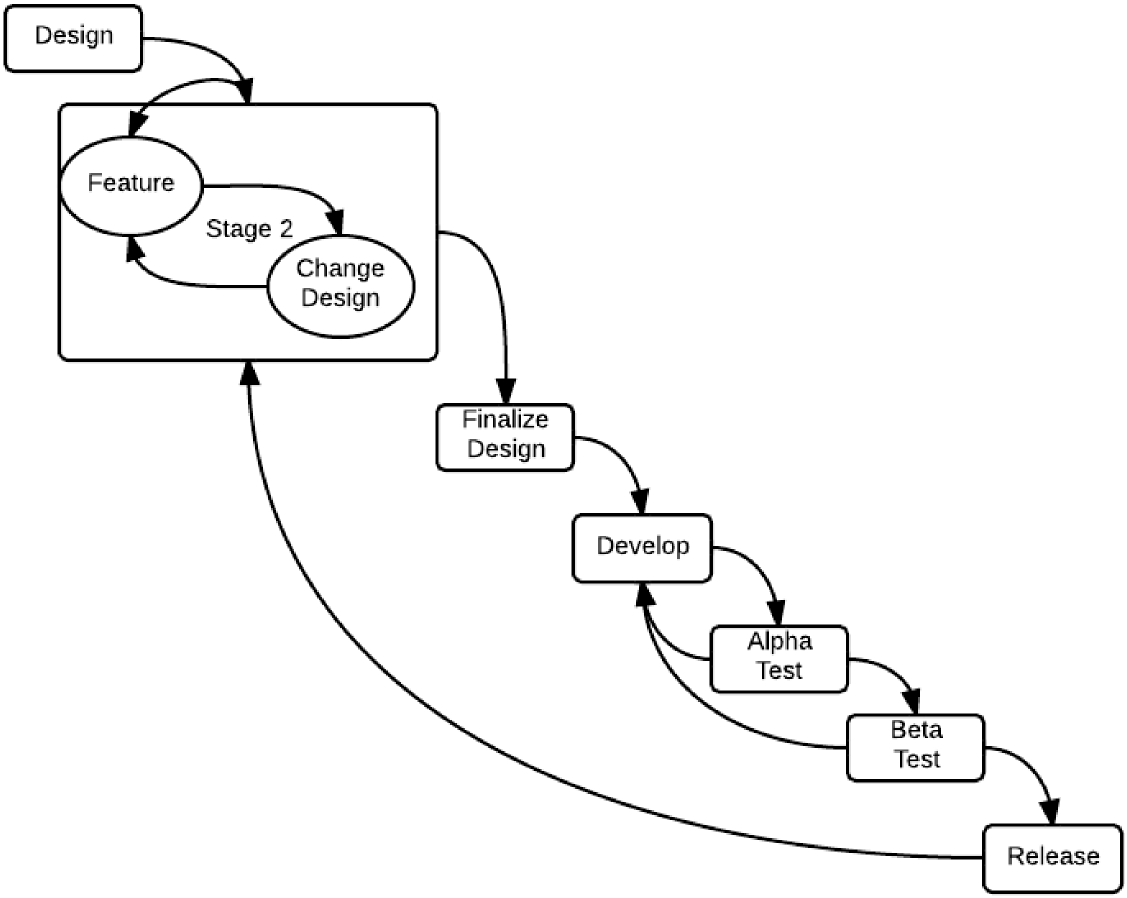
Difficulties Faced

The web application had very different challenges than Android. One of the hardest things for the web application was simply set up. Setting up a Tomcat server on your computer for Eclipse to connect to seems like it would be very simple. Unfortunately some things went wrong and we ended up spending almost 6 hours just trying to get this connection to happen.

From there we ran into issues setting up the Hibernate mappings. When Hibernate Utils, which auto generates the mappings to the database, was first installed on a development computer, it actually caused Eclipse to crash on startup. We had to redownload Eclipse on the machine to be able to generate the entity database relationships.

Software Cycle

We as a group decided the best course of action was using a slight variation of the waterfall model. The goal of this project was to have a finished project that was ready for release by the end of the semester. At the same time we also needed have this project done slightly early so we could get a few beta test runs out in action. With this in mind we have stages in mind.



Stage 1: Design the project - In this stage we came up with a complete plan of what needed to be done by the end of our project and how we planned to accomplish these goals, starting by researching technologies that we can use.

Stage 2: Begin work - This stage is just the beginning of development. This stage is broken into # substages.

- Develop components - Develop a feature, such as logging in.
- Assess changes needed to be made to the general project with problems faces, and new ideas in mind. Repeat step 2 until general outline of project is entirely complete.

Stage 2 was done this way because not all of the technologies researched had been used by all the developers in the group, so some things were more complicated than originally thought, or needed slight tweaks to other components to work properly.

Stage 3: Finalize Design - In this stage the design document was finalized to make sure everything was done according to specification, unless it was absolutely necessary to make a change. Stage 3 allowed us to finish our design with some previous work in mind to make sure the entire project was doable to specification.

Stage 4: Continue development - Each week new tasks were given to the developers to complete, keeping everyone on task, and keeping the project goals attainable. Each developer was expected to test their work, before calling it complete.

Stage 5: Alpha Test - In this stage tested the website ourselves with some mock data. Repeat from Stage 4 when we encountered major problems.

Stage 6: Beta Test - In this stage we looked for outside input for involvement and testing. Repeat from stage 4 when problems were found.

Stage 7 - Release to be evaluated by professor.

We ended up following this model pretty well in the end. We step 5 we follow to the letter without even realizing it about a week before the presentation. This model turned out to be the way we developed naturally so it was not forced.

General Testing

Self Testing

As mentioned above, no feature was considered completed until tested thoroughly. When testing a feature we followed a general test plan.

Step 1: Input correct information and check for expected output, and data persistence in the database if necessary. Repeat this at least 3 times.

Step 2: Input ‘bad’ data and check for error catching. Make sure that errors do not throw exceptions to the user, but information that he or she can actually understand. Also make sure that the exceptions are logged on the server so that we could look back and fix problems.

Step 3: Place in malicious data, such as would be used during an SQL injection or XSS attack. Make sure that the malicious data was handled in a way such that it prevented the attack, but did not bring attention to it, so that a normal user would never run into a situation where they think they had broken the website.

If any of the steps had problems, we fixed them before checking in the code.

Alpha Testing

During the alpha testing we needed to quickly check the entire website for major problems. To do this we set up mock data and run through normal user scenarios to make sure that during the beta tests the normal users did not run into big problems. All bugs were placed in tickets and ordered by importance to fix, by the leader. When the most serious cases were fixed, we proceed to the Beta Test.

Beta Testing

This stage of testing is the most important. Although we did not find actual organizations to use during testing, we created detailed scenarios an organization or user would encounter during actual use. In some ways, this stage was similar to Alpha Testing but more detailed and complex. We had concrete goals of what had to work by our deadline and worked to make sure it happened.

Web Development Testing

Something that is important to all kinds of development is testing. Web development has special hurdles to leap when it comes to testing sites not bound for a homogenized enterprise environment. There are many browsers, operating systems, and hardware combinations that need to be considered in testing just the way the website renders, much less how it is used. Microsoft provides Virtual Machine images for testing various combinations of Internet Explorer with different versions of Windows. Since Primefaces is based on the latest jQuery 1.x, jQuery at least should run on IE6, the latest two versions of Chrome, Firefox, and Opera, and Safari 5.1+ as of this writing. We may end up using CSS that breaks in versions of Internet Explorer older than 9 or 10, so we had to take into consideration browser market share when deciding what version to cut off support at. Linux has a very low install base, but it was worth testing at least Firefox and Chrome in Linux, as we have personally seen local non-profits using Linux to help cut licensing costs for their software solutions. Selenium is a tool that automates web browsers that we used for testing. Microsoft also has a suite of VMs that they distribute through modern.ie to aid testing browser compatibility.

In practice, we found Internet Explorer just did not work as well with our website as Chrome and Firefox. Although Internet Explorer used to be considered the standard, that may be changing. Internet Explorer has legacy code that causes it to work slightly different than Chrome and Firefox, which causes problems even today.

Testing does not just involve visual verification, however. As was mentioned earlier, speed is an important aspect of the user experience. Profiling different parts of the website will provide necessary information to feed back into development to make sure that the website performs well for all users, so that users are not deterred from use by the speed of the site. Tools like YSlow that aid in diagnosing what the bottlenecks are and how to improve them are indispensable in this process.

Colorblindness was mentioned earlier also. We know people that are red-green colorblind that could help us test the site to make sure that it remains accessible once we choose a color palette.

Once the site was solidified, then the testing became more nuanced. We can use A|B testing to test the effects of design changes on actual use, and we can collect information about users using analytics tools like StatCounter or Google Analytics to be able to see what parts of the site get the most use or may need to be tweaked or redesigned altogether. (Chapman, “10 Usability Tips Based on Research Studies”) This, however, is something we plan to do in the future as real people use our site and the number of users grow.

Android Prototype and Test

We tested the application connecting to the internet on a Samsung Galaxy S2. Testing for the web service was done using a local MySQL database on the same computer as the app project. So in other words, a prototype database was built because the main one on our server was mostly complete, but had only limited test data in it.

Another reason is because it is easier to use a simple test database than one not all the team members have access to (at least not initially). When the prototype database was built, we mimicked data and tables the real one contained.

A third reason for using the prototype is it provided a great learning experience for those team members who are not familiar with MySQL and performing MySQL queries. The web service had to perform actually MySQL queries, so it was vital the code be correct and all members understand it. In effect, we created a local server on the development machine to do initial testing.

At that stage, we also used sample logos and color schemes as that had not been entirely finalized yet.

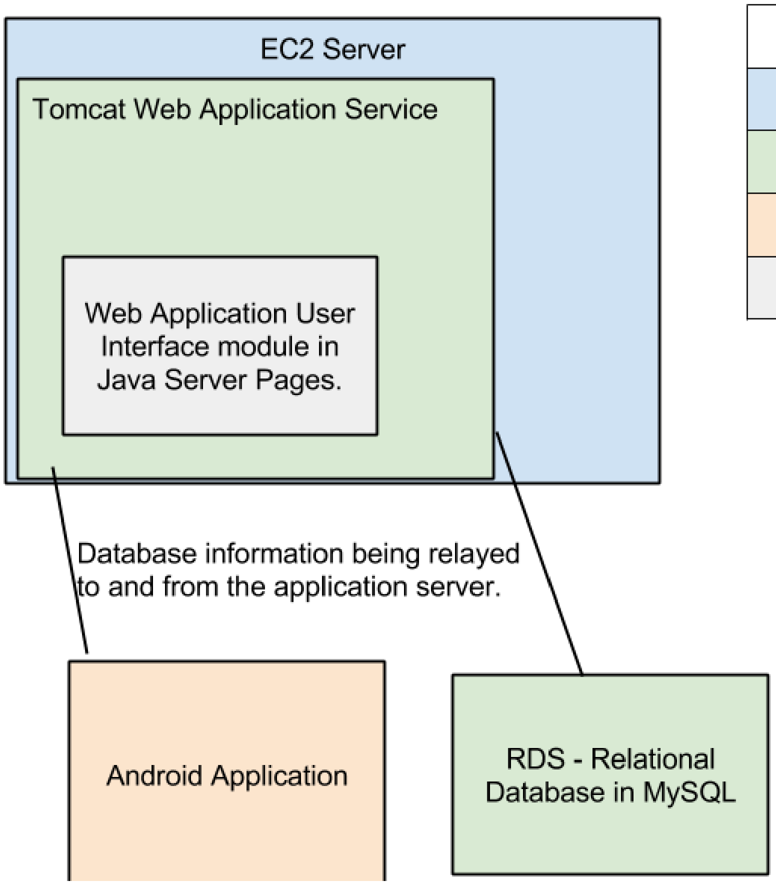
Testing was mostly done in Eclipse by running the project in debug mode and stepping through the code. We also set breakpoints at various locations to jump to specific code. Testing was really a huge component of the project and a big time drain in some cases. For the Android app, large portions of code were written then tested, then we tested it all together. Testing tiny pieces of code can be difficult, because so much goes into doing something simple as connecting to the database. There is a variety of technologies involved and they had to be in place before significant testing could be done.

For testing purposes, we needed to fill the database with the following:

- Volunteer class data
- Group Data
- Event class data

The database had to be queried using the volunteer id. The id pulled all the events with a matching id from the database. An event search worked in a similar way, except instead of using a volunteer id, the query was based on something such as interests, skills, etc.

All blocks are currently in the research phase.


EC2 Server
Tomcat Web Application Service
Web Application User Interface module in Java Server Pages.
Database information being relayed to and from the application server.
Android Application
RDS - Relational Database in MySQL

Color	Owned by
🟦	Everyone
🟩	David
🟧	Henry
⬜	Austin

Facilities and Equipment

We often worked from home so our main facilities were our houses. We did have Harris Engineering Center room 208 on UCF's campus. The equipment we used was as follows:

- 3 Laptop Computers
- 3 Cellphones
- 2 Desktop Computers
- 1 Server
- 1 Database Server

Budgets

Samsung Galaxy S2	Lent by group member for testing
Samsung Galaxy S3	Lent by group member for testing
HTC One	Lent by group member for testing
2x Home Built High End Computer	Lent by group member for testing
Toshiba laptop	Lent by group member for testing
Acer Laptop	Lent by group member for testing
AWS EC2 free tier	Free, provided by Amazon
AWS RDS free tier	Free, provided by Amazon

Project milestones for both semesters.

Semester 1:

All due dates were 3:00pm on the date listed. No task was considered done until tested thoroughly.

Oct 3 -

Set up server - Get a generic restful server up and running on EC2 instance that we could build off of. This includes setting up a MySQL database which initially was empty. - David

Android Tutorials - Henry

HTML5/CSS3 tutorials - Austin

Look into RestFul Architectures. - Everyone

Oct 7 -

Start designing the user interface for the website - Decide the layout of each webpage of the website. Mock up the basic look. - Austin

Begin Android designing - Get the design of the Android Application figured out and mocked up. - Henry

For these goals we used various implementations for the mock up. Austin will draw the design by hand or use software. Henry will use screenshots of the Holo theme for inspiration and then a paint program to create the mock up.

Oct 14 -

Acquire resources such as a logo and graphics so the website looked polished - Items such as backgrounds, buttons, images, etc. We used public domain logos and resources initially then replaced them with custom artwork as time dictated. - Austin

Oct 21 -

Begin to code the website GUI - Get the basic GUI started - Austin

Begin work on the Restful architecture - Start the server code - David (Austin look at server code to learn for future)

Begin sprint to finish design document - With all of the work done at this point, we had a better idea of how the design was supposed to be - Everyone

Setup hibernate - map the components though hibernate from the server to the DB.- David

Since David initially set up the server and wrote Hibernate code, it was a good idea for other team members to look at what he had done or engage in Hibernate tutorials to fully understand the process.

Nov 25 - *Have simple prototype of website -* This was not fully functional but included log in, and some of the easier pages. The pages did not necessarily contain all the artwork, just basic functionality. - Austin/David

Dec 2 - *Finish design document first draft* - With the draft in hand we later got feedback from the professor to get a better idea of what needed to be added or amended. We also voted on any unresolved design decisions, such as website name, for resolution. - David

Dec 21st -

Code the profile feature for three classes: user, organization and groups - Add the required information to the servlets that will return the page. - David

Design the UI of the added components - the actual page with the servlet components added - Austin

Have simple Android Prototype - Same as website prototype. Get a few activities up and running with hopefully some artwork in place. - Henry

Semester 2:

Jan 13 - *Code the filter feature* - Allow the user to search by various criteria. All three components needed to be done. Additional components were done at a later time. - Everyone

Jan 27 - *Add signing up for events* - Add the relationships on the server, the website and the android components. - Everyone

Add notifications - this was considered for both Android and the Web Application, so the user knows what events are approaching.

Feb 10 - *Finish the system for connecting organization and people* - This needed to be all three components. The bulk of the code was written with only minor details remaining.

March 17 - *Alpha Testing* - Major features such as Search, Calendar, Profile, etc. Everything was together to ensure inoperability.

March 31 - *Plan extra features* - Taking into consideration what was possible with the remaining time. Possibilities include:

- Phone verification
- Organization verification
- Third party calendar integration (google calendar, icalendar,
- Android facebook sign in
- Google sign in
- Google maps on both components
- iPhone App
- “share this event” social buttons like for fb/twitter/g+/reddit/email/print flyer/etc.
- connect already made account to facebook
- list of ‘nots, like “do not have these skills” or “will not do these tasks”, such as if someone is allergic to animals, disabled, unskilled at certain tasks, or simply dislikes a task/skill

April 14 - *Beta Testing* - Feedback for new features was considered. We hoped to have friends test our website and app.

April 28 - *Beta Testing* - This included any additional features added. Any remaining bugs were found because everyone in our group participated in this final stage of testing.

May ? - *Celebrate with drinks!* - Paid for by Austin and Henry! Professor, you are invited as well!

In the end, we tried to stick to this schedule as much as possible. But of course, real life got in the way. For example, we were attending career fairs and interviewing which took up weeks worth of time. Some weeks some of us had exams to focus on.

So this served as a rough guideline. Although not every deadline was met on time, it still helped a great deal by providing a way to look back and see exactly what needed to be done. And like every group, we worked more toward the end of the semester to finish any remaining issues. But overall, we are very proud of what we accomplished in relatively a short time considering the ambitious goals set for ourselves.

General Responsibilities

David - as manager of the project, David was responsible for setting up meetings, helping the others meet deadlines, re-evaluating project components, and making executive decisions when needed. He mainly worked on the Database and Server code, but also looked over the rest of the project and helped as needed.

Austin - Austin took a lead role on the Web interface and of course helped on the server since they are intertwined tightly.

Henry - Henry worked on the Android Application, but also took a significant development role programming the main website.

Project Summary and Conclusions

Our overall goal was to create a website that helps people quickly and easily find a volunteer event they can participate in. We wanted to do something that has a positive impact on the community.

There are several types of users. One is a volunteer, who creates a profile, does a search for volunteer events and can look up events already signed up for. Another is a group, which consists of volunteers. A user can join a group to be part of something greater than just one person, for example, a cleanup event. The last kind is an organization such as Goodwill that posts ads looking for volunteers. Organizations are also responsible for approving volunteers.

There are three main components of our project: the server backend, the website and the Android app. The server hosts our MySQL database. The website is the main tool people use to participate in volunteer events. While the Android app provides basic functionality for users on the go to quickly look up recent volunteer events.

One of our ongoing challenges was to find graphics such as logos that we all agree on. None of us are artists so we used a few public domain images while testing. One of the aspects we all enjoyed was visiting different real world websites to find a style we all liked.

Working together as a team, at times, has also been a challenge because of conflicting schedules. And we all work at different paces and have different styles of coding. In conclusion, we have found that being tightly organized, especially at the beginning, and having a solid plan has helped a great deal.

We are hoping in the near future to get a real organization to post something on our website. Our team members could be the first volunteers. This would be the exciting part, to use it in the real world and make a real difference.

Another possibility is to donate our website to a professor at UCF who has shown interest using it not only as a database teaching aid, but also something his students can expand on in the future.

In conclusion, we have all learned a tremendous amount. Just putting a project together of this size, with so many components has been a challenge, but one that was welcome because in the real world, at a real company, projects like this are commonplace. So we feel we’ve gotten real experience that would look great on a resume.

References

“About rel="canonical"” *Google Webmaster Tools*. Google, Inc., n.d. 2 Dec 2013.

<https://support.google.com/webmasters/answer/139394?hl=en>

“About /robots.txt” *GET /ROBOTS.TXT*. robotstxt.org, n.d. 2 Dec 2013. <http://www.robotstxt.org/robotstxt.html>

“Activity”. *Android Developers.* 2013. 02 Dec 2013. <http://developer.android.com/reference/android/app/Activity.html>

"Ajax". *Wikipedia*. 27 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/Ajax_(programming)>

"Amazon Elastic Compute Cloud". *Amazon web services*. 2013. 2 Dec 2013. <http://aws.amazon.com/ec2/>

"Amazon Relational Database Services". *Amazon web services*. 2013. 2 Dec 2013., <http://aws.amazon.com/rds/>

"Apache Tomcat". 2013. 2 Dec 2013. <http://tomcat.apache.org/>

“AsyncTask”. *Android Developers.* 2013. 02 Dec 2013. <http://developer.android.com/reference/android/os/AsyncTask.html>

“BaseExpandableListAdapter”. *Android Developers.* 2013. 02 Dec 2013. <http://developer.android.com/reference/android/widget/BaseExpandableListAdapter.html>

“Block or remove pages using a robots.txt file” *Google Webmaster Tools*. Google, Inc., n.d. 2 Dec 2013. <https://support.google.com/webmasters/answer/156449?hl=en>

“BroadcastReceiver”. *Android Developers*. 2013. 02 Dec 2013. <http://developer.android.com/reference/android/content/BroadcastReceiver.html>

"Cascading Style Sheets". *Wikipedia*. 22 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/Cascading_Style_Sheets>

Chapman, Cameron. “10 Usability Tips Based on Research Studies” *Six revisions.* Six Revisions, 15 Sep 2010. 2 Dec 2013. <http://sixrevisions.com/usabilityaccessibility/10-usability-tips-based-on-research-st

udies/>

“Create Strong Passwords”. *Microsoft Safety and Security Center.* 2013. 2 Dec 2013. <http://www.microsoft.com/security/online-privacy/passwords-create.aspx>

Fox, Vanessa. “Bing Now Supports Google’s Crawlable AJAX Standard?” *Search Engine Land*. Third Door Media, Inc., 4 Jul 2011. 2 Dec 2013. <http://searchengineland.com/bing-now-supports-googles-crawlable-ajax-standard-84149>

“Fragments”. *Android Developers*. 2013. 02 Dec 2013. <http://developer.android.com/guide/components/fragments.html>

“GSON”. *Wikipedia.* 27 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/GSON>

“Hash Function”. *Wikipedia.* 29 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/Hash_function>

"Hibernate". *JBossCommunity*. 2013. 2 Dec 2013. <http://www.hibernate.org/>

“How JQuery Works”. 2013. 2 Dec 2013. <http://learn.jquery.com/about-jquery/how-jquery-works/>

"HTML5" *Wikipedia*. 21 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/HTML5>

“HttpURLConnection”. *Android Developers*. 2013. 02 Dec 2013. <http://developer.android.com/reference/java/net/HttpURLConnection.html>

“IntentService”. *Android Developers*. 2013. 02 Dec 2013. <http://developer.android.com/reference/android/app/IntentService.html>

“Internet Users Screen Resolution Realtime Statistics for 2013” *Screen Resolution Statistics*. ScreenResolution.org, n.d. 30 Nov, 2013 <http://www.screenresolution.org/year-2013/>

“Introducing JSON”. *json.org.* 2013. 2 Dec 2013. <http://www.json.org/>

“Java Servlet”. *Wikipedia.* 1 Dec 2013. 02 Dec 2013. <http://en.wikipedia.org/wiki/Java_Servlet>

"JavaScript". *Wikipedia*. 30 Nov 2013. 2 Dec 2013. <http://en.wikipedia.org/wiki/JavaScript>

"JavaServer Faces Technology". *Oracle*. 2013. 2 Dec 2013. <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

“Key Stretching”. *Wikipedia.* 4 Oct 2013. 02 Dec 2013. <http://en.wikipedia.org/wiki/Key_stretching>

Lidwell, William, Kritina Holden, and Jill Butler. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design*. Rockport Publishers, 2010. Print.

Lupton, Benjamin Arthur, et. al. “history.js” *history.js*. Github, Inc., 22 June 2013. 2 Dec 2013. <https://github.com/browserstate/history.js>

“Making AJAX Applications Crawlable” *Google Developers Webmasters*. Google, Inc., 17 Feb 2012. 2 Dec 2013. <https://developers.google.com/webmasters/ajax-crawling/>

"MySQL Workbench 6.0". *MySQL*. 2013. 2 Dec 2013. <http://www.mysql.com/products/workbench/>

“Salt”. *Wikipedia.* 18 Nov 2013. 02 Dec 2013. <http://en.wikipedia.org/wiki/Salt_(cryptography)>

“Security Tokens”. *Wikipedia.* 20 Nov 2013. 02 Dec 2013. <http://en.wikipedia.org/wiki/Security_token>

“Shared Preferences”. *Android Developers*. 2013. 02 Dec 2013.<http://developer.android.com/reference/android/content/SharedPreferences.html>

“Submitting Sitemaps” *Google Webmaster Tools*. Google, Inc., n.d. 2 Dec 2013. <https://support.google.com/webmasters/answer/183669?hl=en>

“Top 12 Browser Versions Combining Chrome and Firefox (5+) in the United States on Nov. 2013” *StatCounter GlobalStats*. StatCounter, n.d. 2 Dec 2013. <http://gs.statcounter.com/#browser_version_partially_combined-US-monthly-201311-201311-bar>

“Using site speed in web search ranking.” *Official Google Webmaster Central Blog: Using site speed in web search ranking*. Google, Inc., 9 April 2010. <http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html>

“What are Sitemaps?” *sitemaps.org*. Google, Inc., Yahoo, Inc., and Microsoft Corporation, 27 Feb 2008. 2 Dec 2013. <http://www.sitemaps.org/>

“What is HTTPS?”. *Instant SSL.* 2013. 2 Dec 2013. <http://www.instantssl.com/ssl-certificate-products/https.html>

"Why Primefaces?" *Primefaces*. 2011. 2 Dec 2013. <http://www.primefaces.org/whyprimefaces.html>

@Fyrd. “Session history management - Candidate Recommendation” *Can I use...* @Fyrd, n.d. 2 Dec 2013. <http://caniuse.com/#search=pushState>