# Final Report for Web Tournaments Project

Group #7

Michael Davis

Mark Gollnick

Taehyun Park

Maxwell Peterson

Rob Sheehy

| | | | |
|---|---|---|---|
| 1 | 11/24/12 | MP | Initial Document |
| 2 | 12/06/12 | MD | Finishing Document |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Table of Contents

# 1 Introduction

## 1.1 PURPOSE

The purpose of this document is to enumerate and elaborate on the requirements and design of the Web Tournaments web application. This document should enable the reader to understand the thought process of the programmers as they put together the different components of the project.

## 1.2 SCOPE

The project aims to solve the problems presented when keeping track of tournaments on paper and white boards.  There is a lot of manual work and logic involved when doing tournaments the old fashioned way, such as calculating seeds after a round robin or trying to draw a double elimination bracket for 17 people.  The goal of this project is to create a platform on the web that can handle many different kinds of tournaments.  A platform that will eventually be expanded upon, but for the purposes of this project we will only implement a few kinds of tournaments due to time constraints. The plan for this project is to fully implement single and double elimination tournaments, as well as round robin tournaments.  If there is time we would like to be able to calculate seeds from round robin results to go into a single or double elimination bracket.

## 1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS

| Term | Description |
|---|---|
| Round Robin Tournament | A type of tournament in which each player or team plays every other player or team in the tournament or pool. |
| Single Elimination Tournament | A type of tournament in which a player or team can only lose once before they are out of the tournament. |
| Double Elimination Tournament | A type of tournament in which a player or team can lose twice before being out of the tournament. After losing once, they enter the consolation bracket, which at best can win third place. |

## 1.4 DESIGN GOALS (BASED ON DELIVERABLES, FUNCTIONAL, NON-FUNCTIONAL, USER-INTERFACE REQUIREMENTS)

1. Usability
    a. The end product should be easy to use, and intuitive. It should not require an instruction manual or help pages to use. The user should not be confused about what to do next, and it should have a kind of 'flow' of easiness.
2. Extensibility
    a. The project should be able to be expanded upon after the semester has ended. New tournament types and user interface interactions should easily be able to be added and modified.
3. Reliability
    a. The project should be available from any internet connected device at any time.
4. User should be able to create his or her first tournament easily and quickly.
    a. Creating a user's first tournament should not require logging in until after they have gone through the tournament creation wizard.
    b. Entering data to create a tournament should be easy and intuitive, and must not be confusing or tedious.

5. Response Time
    a. The service should respond quickly when an action is performed.  No page should take more than 3 full seconds to load (internet connection not being the limiting factor)
6. Users can update the status and score of matches, and the changes will be reflected to the rest of the users/viewers.
7. Users can create tournaments of different types and rule sets.
8. Users can share their tournament with other people via a URL or Facebook post.

## 2 References

**Backend**

Google App Engine – Free/Low cost hosting solution that is able to quickly scale with our app and has the power of Google behind it.

Flask – A slim web application framework for Python.  Our team discovered this as a replacement for webapp2 and thought it looked really nice.  This framework handles HTTP requests and routing based on URL patterns.

WTForms – The forms library.  It allows developers to create a form class with field attributes which end up being rendered into HTML forms easily.

Jinja Templates – The HTML templating language.  Create HTML templates with variables passed into them; ability to do loops and template subclassing/extension.

Flask extensions – Various flask extensions are used such as flask-login, flask-oauth, and flask-assets.

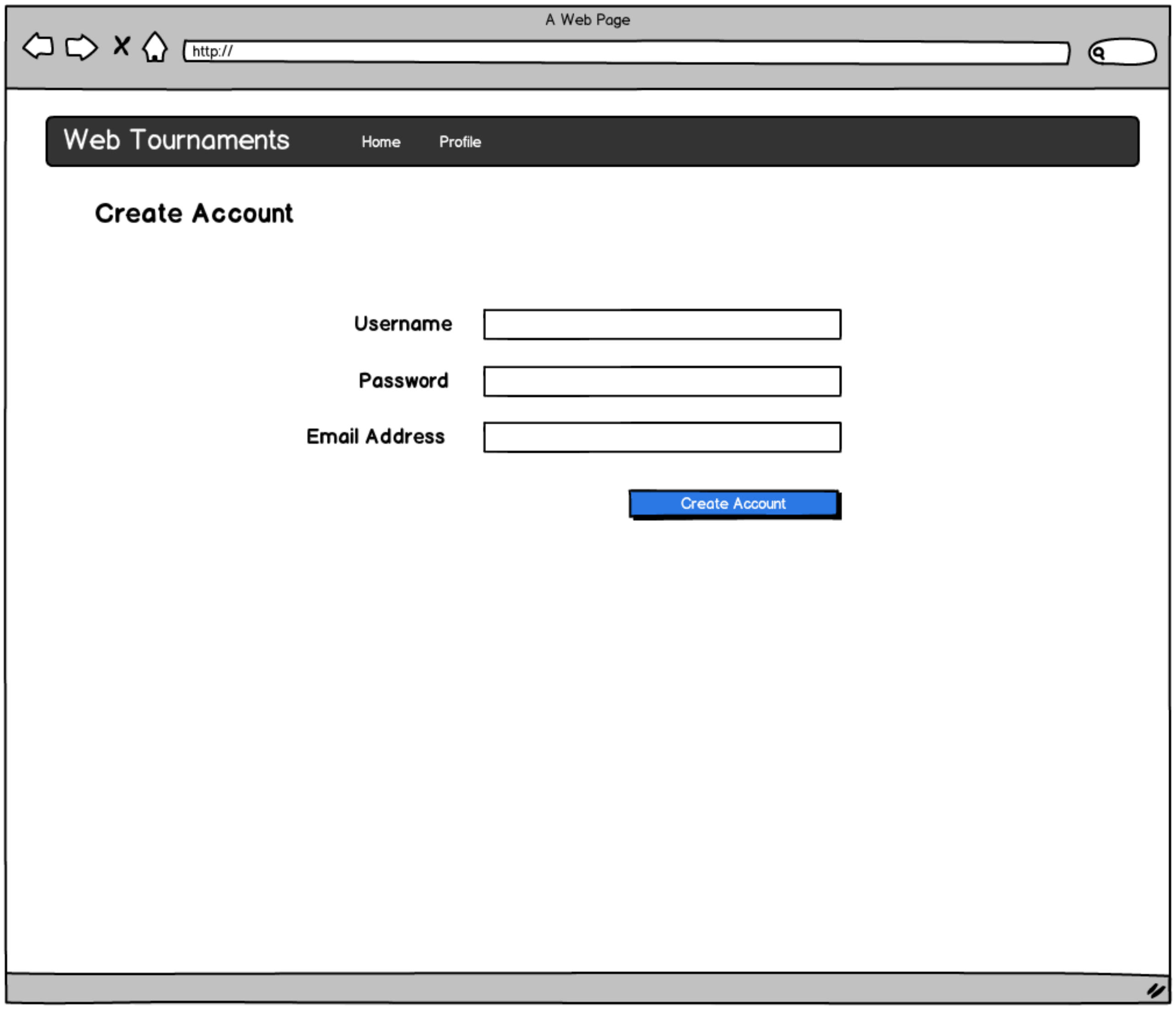
**Frontend**

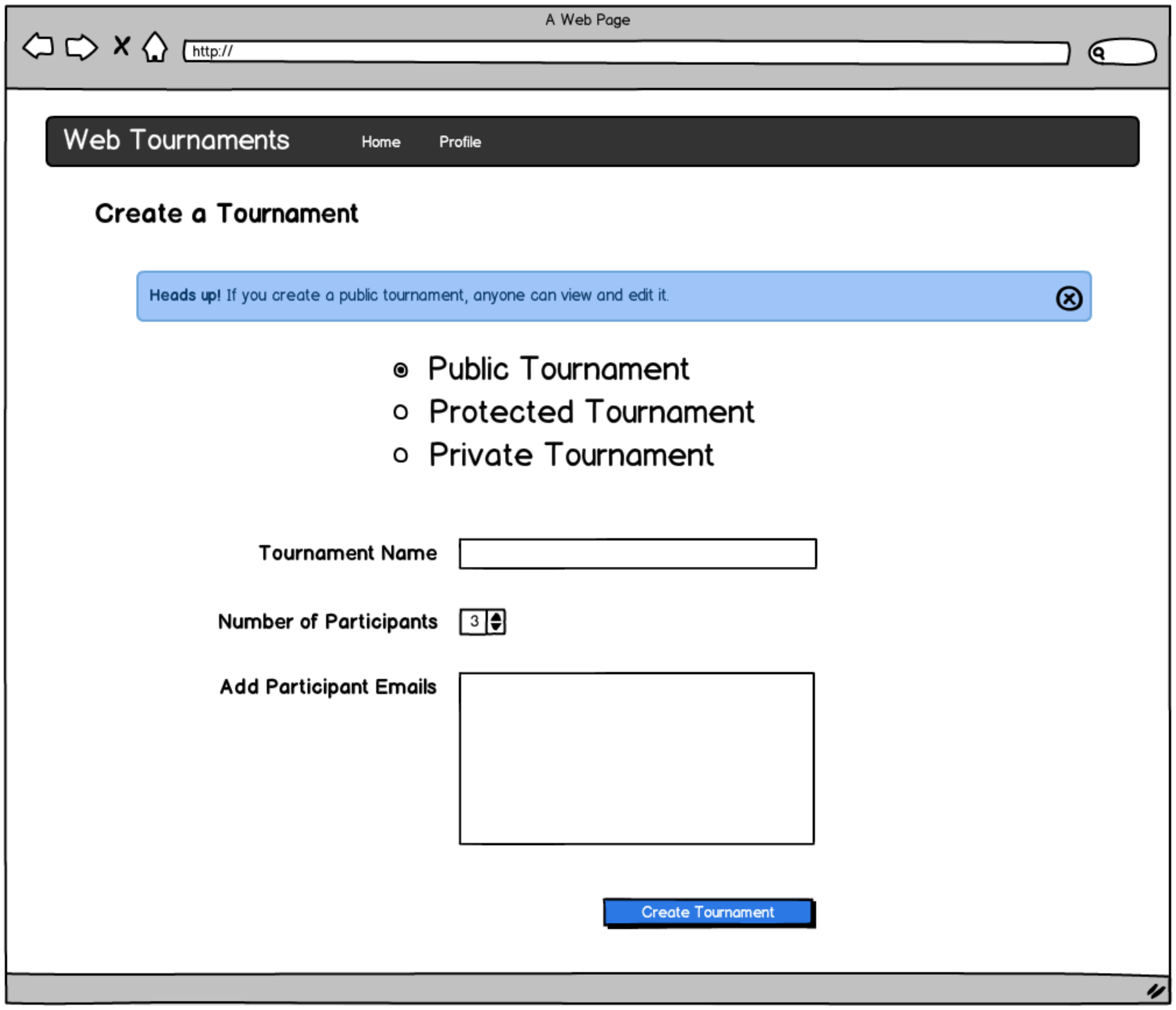
Twitter Bootstrap – An awesome frontend framework that provides us with a slick look and feel for the web application, without having a graphic designer on the team.

jQuery – Making javascript not horrible since 2006.  It helped with making AJAX calls and refreshing things on the page.

jQuery Datatables – Easily making searchable and sortable tables.

## 3 User Interface Description




A Web Page

http://

Web Tournaments    Home    Profile

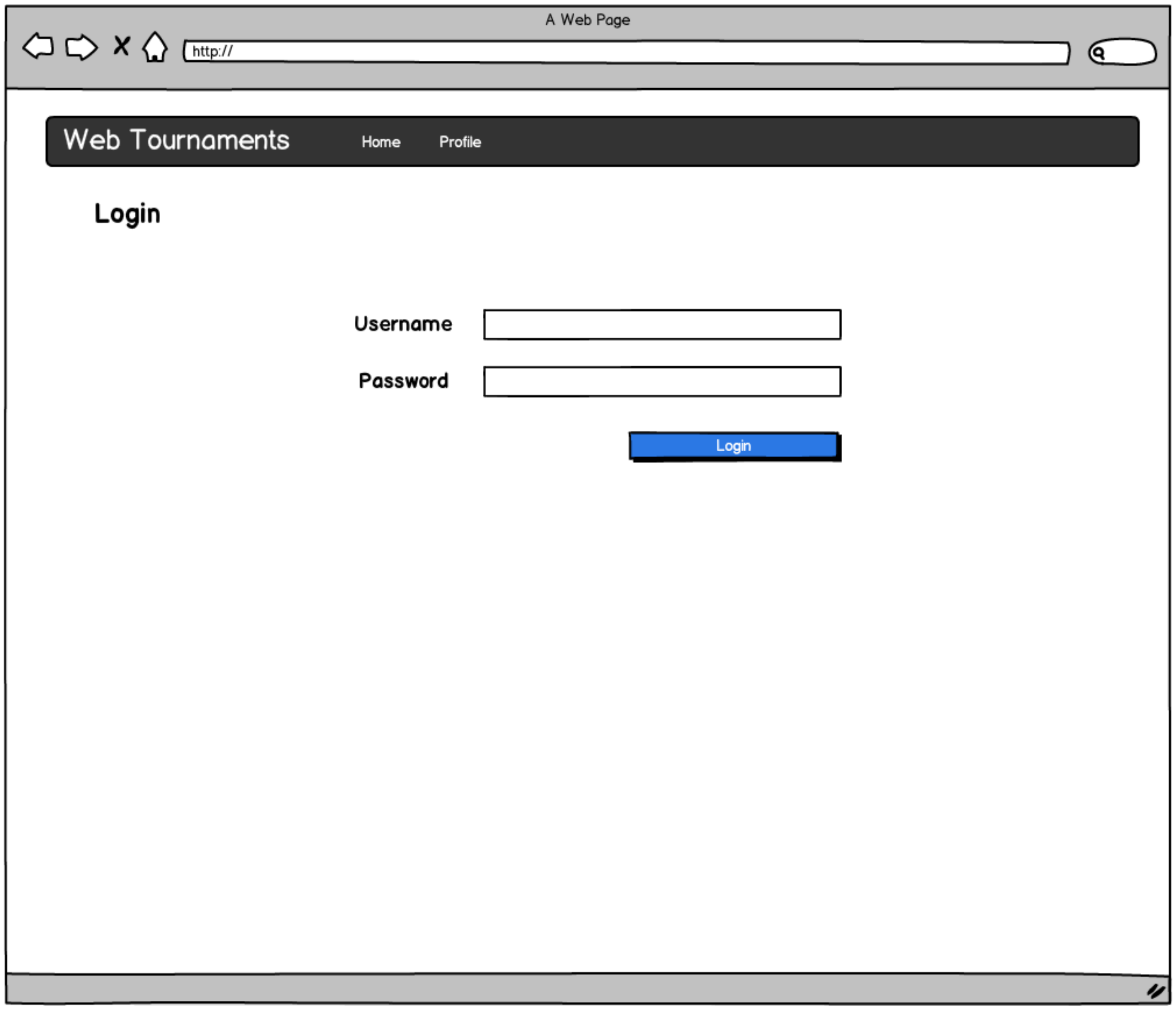
Create a Tournament

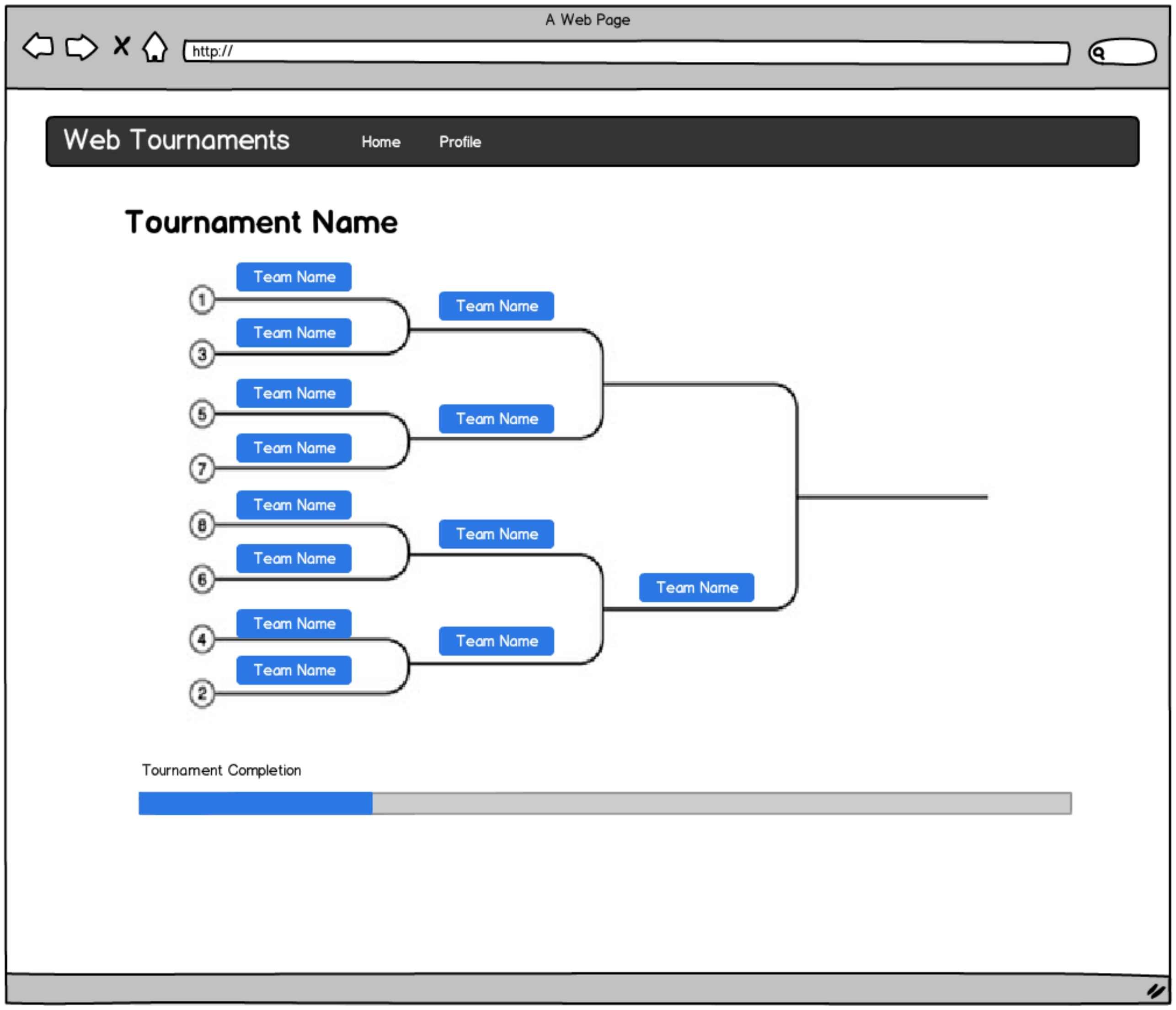
**Heads up!** If you create a public tournament, anyone can view and edit it.

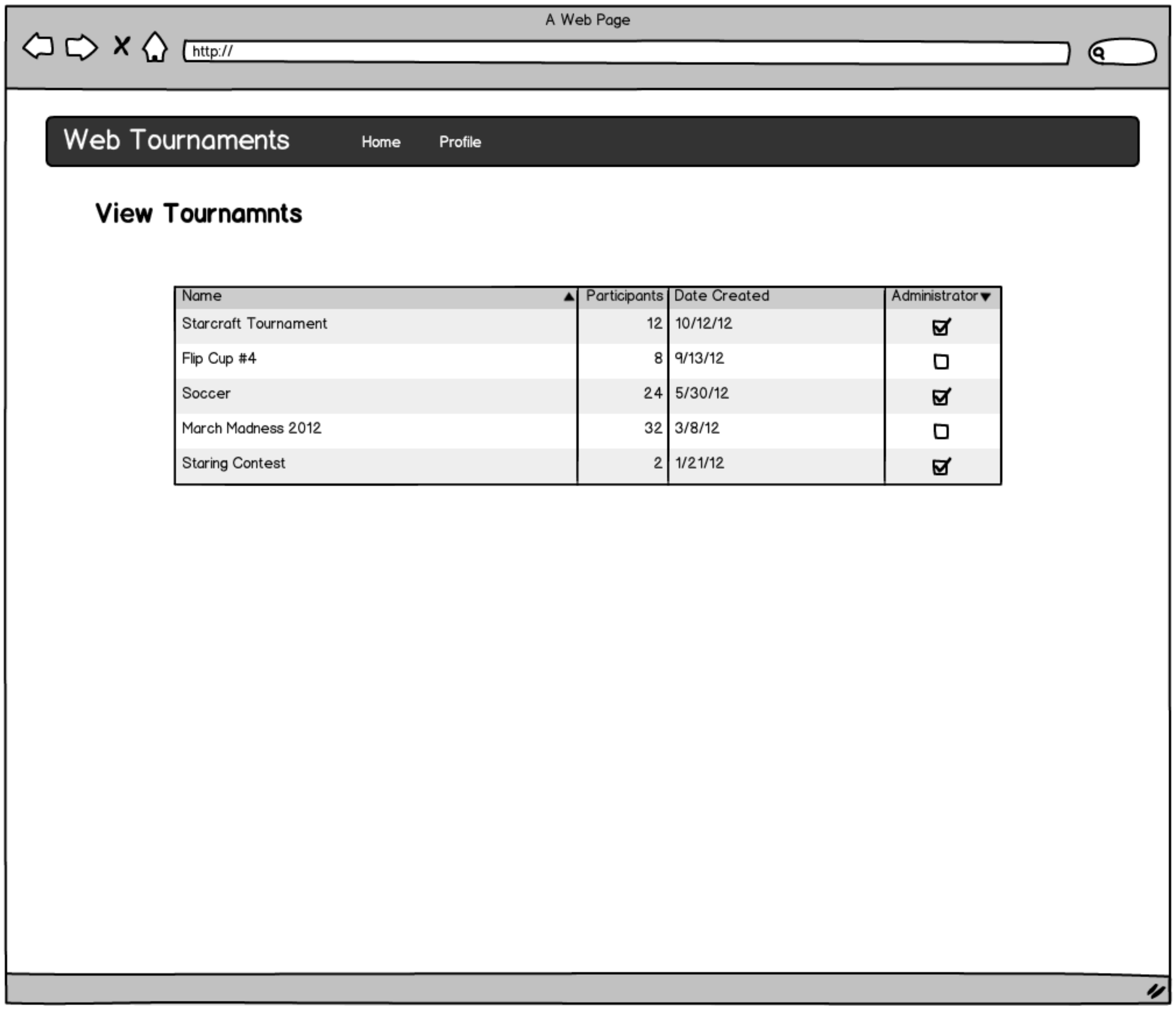
[✓ Public Tournament]
Protected Tournament
Private Tournament

| | |
|---|---|
| Tournament Name | |
| Number of Participants | 3 |
| Add Participant Emails | |

Create Tournament


A Web Page
http://
Web Tournaments
Home
Profile
Login
Username
Password
Login


A Web Page
http://
Web Tournaments
Home
Profile
Tournament Name
Team Name
1
3
5
7
8
6
4
2
Tournament Completion



View Tournamnts

| Name ▲ | Participants | Date Created | Administrator ▼ |
|---|---|---|---|
| Starcraft Tournament | 12 | 10/12/12 | ☑ |
| Flip Cup #4 | 8 | 9/13/12 | ☐ |
| Soccer | 24 | 5/30/12 | ☑ |
| March Madness 2012 | 32 | 3/8/12 | ☐ |
| Staring Contest | 2 | 1/21/12 | ☑ |

# 4 Decomposition Description

## 4.1 MODULE DECOMPOSITION

Modules are broken up by topic area. Each module contains classes that describe anything needed for that topic area. There are classes for forms, models, views, utilities, and different database actions that pertain to each module. For example, checking a password as valid would be found in the utilities section of the auth module.

### 4.1.1 Auth

4.1.1.1 The auth module includes a number of methods for dealing with user creation, authentication, and management. It includes the model descriptions for users, form descriptions for the register and login forms, and utilities for generating password hashes, generating secure tokens, validating tokens, and other authentication related tasks. All of the logic for validating a login and handling a user session is also included.

### 4.1.2 Base

4.1.2.1 The base module includes the methods and utilities that didn’t fit in any of the other modules. It includes mail functionality and form widgets that are used across the entire app.

### 4.1.3 Lib

4.1.3.1 Lib houses all of our external libraries needed to run the application.

### 4.1.4 Tournament

4.1.4.1 The tournament module houses all of the forms, models, utilities, database actions, and views for all of the tournament functionality. Actions houses all of the database interaction such as creating a tournament, updating a tournament, and getting tournament information out of the database. Forms contains all of the forms needed for dealing with tournaments, including the forms needed to create, edit and delete a tournament and the forms needed to update a tournament from the tournament view page. Models contains all of the model descriptions for our tournaments, including the models for Tournaments, Matches, and Tournaments.

This module has grown to be quite large, and should really be split into multiple modules at this point.

## 4.2 CONCURRENT PROCESS

Our application does not currently use multiple threads. Google App Engine provides minimal support for multi-threaded applications.

In the future there are plans for using task queues for creating large tournaments, but it has not been implemented at this point.

## 4.3 DATA DECOMPOSITION

### 4.3.1 WTUser

4.3.1.1 The WTUser module stores the information about different users. It includes properties that include username, email, salt, password, and Facebook or Google identifiers used in OAuth authentication.

### 4.3.2 Tournament

4.3.2.1 The Tournament model contains information about a complete tournament. This includes information such as the name, type, date, and location of the tournament, along with permissions on the tournament.

### 4.3.3 Match

4.3.3.1 The Match model contains information about a specific match in a tournament. It includes information such as the round of the tournament it is in, if it has been played, and contains a reference to the next match that the winner should advance to in a bracket tournament.

### 4.3.4 Participant

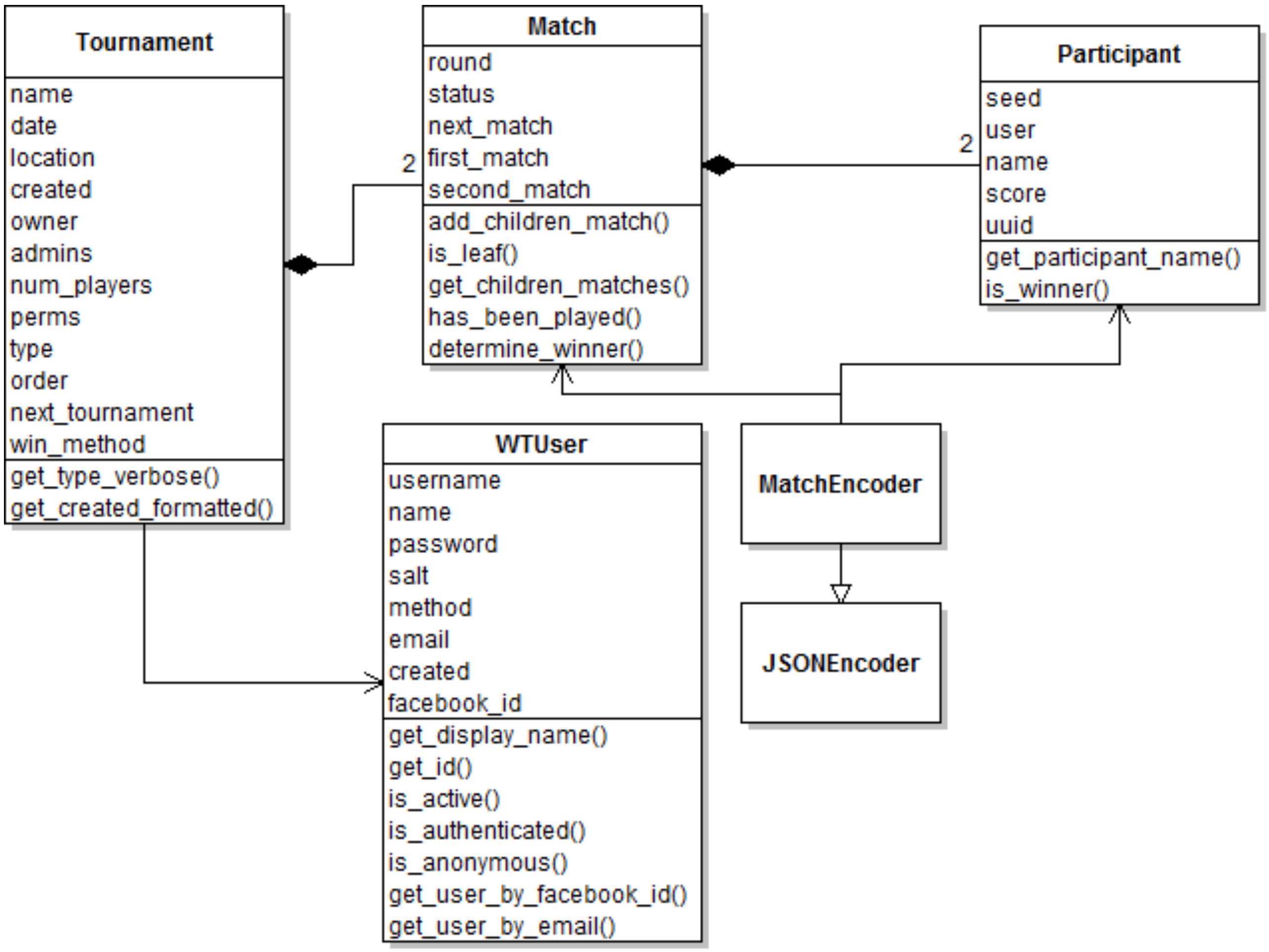
4.3.4.1 The Participant model contains information about one player in a specific match. It includes information about the player such as seed, a reference to a user, and a name field.

## 4.4 OVERALL SYSTEM SPECIFICATION

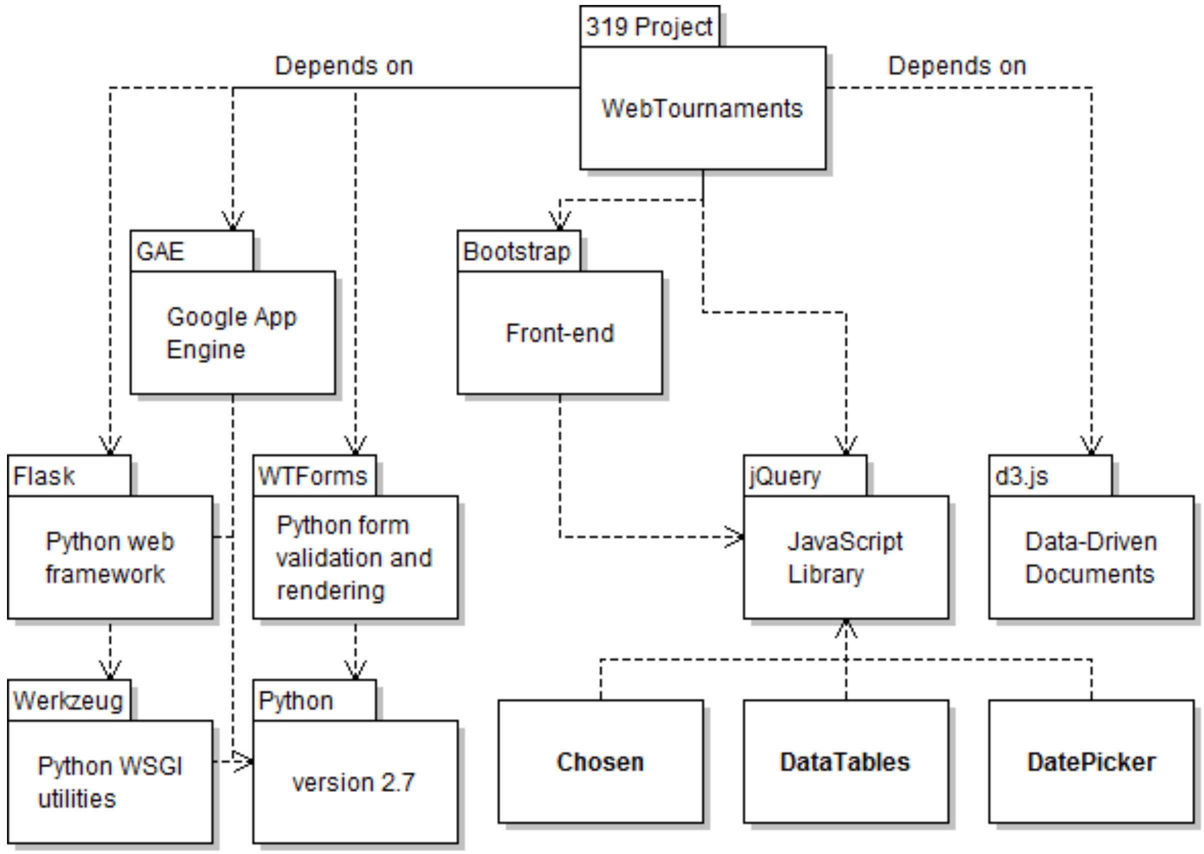
4.4.1 N/A – Our application does not live on a typical server system, but instead on the Google App Engine platform.

## 5 Dependency Description

### 5.1 INTERMODULE DEPENDENCIES



## 5.2 INTERPROCESS DEPENDENCIES



## 5.3 DATA DEPENDENCIES

5.3.1 The user and tournament data is stored in the datastore, which is a non-relational database managed and scaled by Google App Engine. When Displaying a tournament bracket, the data is requested at a designated endpoint using the tournament's ID key, where it is then translated from Python objects into a JSON representation.

# 6 Interface Description

## 6.1 MODULE INTERFACE

6.1.1 Views Module( Included in Auth and Tournament modules ) controller for views

6.1.2 Register(Request) : Gets page to register a new user account.

6.1.3 Login(Request) : Gets page to login a user.

6.1.4 New_Tournament(Request): Gets page with tournament creation wizard.

6.1.5 Tournament_Edit(Request, tournament_key): Gets page to edit a tournament. Takes in the tournament key to find the tournament to edit.

6.1.6 Tournament_View(Request, tournament_key): Gets page to view a tournament. Also Take in the tournament key to find the tournament to view.

6.1.7 Tournament_Search: Gets the page to search for tournaments.

6.1.8 Actions Module(Included in Auth and Tournament modules) calls to database

## 6.2 PROCESS INTERFACE

6.2.1 N/A

# 7 Design Rationale

## 7.1 DESIGN ISSUES (CHALLENGES YOU FACED)

## 7.2 TOURNAMENT VISUALIZATION WITH D3

### 7.2.1 Description

Displaying tournament brackets of single elimination was difficult. We hoped to find a framework that it uses in order to display nodes.

### 7.2.2 Factors affecting Issue

Until we figured out how to use D3 properly, we could not display tournaments. It affected us by not allowing use to demo the project significantly during the checkpoint.

### 7.2.3 Alternatives and their pros and cons

We considered using gRapheal before we discovered D3, but it proved to be too limited for our uses. gRapheal is geared more for the creation of charts rather than graphs. Aside from this limitation, D3 and gRapheal have similar jQuery-like interaction, which is a positive.

### 7.2.4 Resolution of Issue

We discovered D3.js and it allowed us to have the functionality that we needed in order draw brackets on the screen in an acceptable way.

## 7.3 TOURNAMENT GENERATION ALGORITHM

### 7.3.1 Description

7.3.1.1 Creating a tournament with an odd number of participants does not work properly.

### 7.3.2 Factors affecting Issue

The complexity of the algorithm took us a while to figure out.  Our first approach towards creating a single elimination tournament was to create matches from the bottom to top recursively.  Since there was no way to travers or get the children of a parent node with the initial match model, we revised it to have children nodes, left_match and right_match.  With the revised match model, we could create matches and connect them from top to bottom.

### 7.3.3 Alternatives and their pros and cons

Running time of the algoright did not make a difference because bot algorithm occur in O(n) time. The only downside of the new algorithm is that it uses more space in the database to store the keys of the left and right matches, but that is very minimal.

### 7.3.4 Resolution of Issue

We revised, and continue to revise, the model and algorithm.

## 8 Tracability

| No | Use Case/ Non-functional Description | Subsystem/Module/classes that handles it |
|---|---|---|
| 1 | User Creation | Auth |
| 2 | User Login | Auth |
| 3 | Tournament Creation | Tournament |
| 4 | Tournament Updating | Tournament |
| 5 | Link User to Tournament | Tournament |

## 9 Language/Framework/Tools Used

**Python** – We love Python. It is especially good for writing web applications due to its simple syntax and high-level approach at programming. When writing code for a web application we are not concerned with the extreme performance or the low level access of memory (pointers, bit-shifting) of a language like C. Instead Python lets us focus on the logic of the application and helps us complete the task in fewer lines of code.

**Flask** – This was the first time that any of us had used the flask micro-framework. We had originally planned on using webapp2, and even started coding the skeleton of the application on it. But then we discovered Flask. Flask is similar to webapp2, but has far better documentation and community involvement. We were able to find some useful flask plugins (like flask-oauth) that simply do not exist for webapp2. Some of us had a lot of prior experience with Django, a larger Python web-application framework. But Django is big and sluggish compared to the slim flask framework. We enjoyed learning how to leverage flask and many of us will probably use flask again for our next Python web application.

**Google AppEngine** – Our application is served by a scalable cloud-based web application hosting service called Google AppEngine. AppEngine runs web applications written in Java, Python, or Go languages. It supplies a scalable non-relational database called BigTable, where all of our data gets stored. AppEngine applications have a free daily quota, which is perfect for school projects like this one. As long as your application doesn't use up a lot of instance hours or database storage, it won't cost a dime. Of course if it ever begins to be widely used, we can begin paying for resources.

**Javascript/jQuery/d3** – A lot of the frontend interactions are written in javascript and jQuery. We make use of AJAX calls to send updated match scores using JSON. jQuery makes constructing AJAX calls as easy as possible: simply provide a URL endpoint and some JSON to send, and a function to call when it gets a response. We also make use of a javascript library called d3. D3 allows us to draw SVG bracket lines and position the match divs in the correct locations.  All of that code is also written in javascript.

**Twitter Bootstrap** – Twitter released a web frontend as an open source project called Twitter Bootstrap.  It has helped us create a more professional looking web application and has taken away some of the pain of HTML and CSS design.  It gives us components to us like good looking buttons, icons, form fields, and dialogs.  Despite supplying us with a lot of the base CSS, we still customized the look and feel a lot to be unique to our application.

# 10 Individual Responsibilities

## 10.1 MICHAEL DAVIS

10.1.1 I contributed the entirety of the authentication module, allowing for user creation, management, and editing. It also covers a number of areas including user sessions and OAuth authentication with both Google and Facebook. I contributed with the front end of the bracket tournaments with the AJAX requests in order to update the tournaments on the server. I handled a great deal of the overall styling of the application as well. I helped design the application framework, everywhere from the switch from Webapp2 to Flask. The database models were designed with Maxwell Peterson based on our original thoughts on keeping track of the information we need and our experience with Google's BigTable.

## 10.2 MAXWELL PETERSON

10.2.1 In the beginning I contributed a lot of the framework code and model design. Michael Davis and I white-boarded out the original model design and implemented it in the model code. I drove the transition from webapp2 to flask, which included creating some framework functions to help make our views easier to write. After that I worked on the tournament creation wizard, passing the POST variable from view to view until the wizard was finished. Towards the end I helped with a lot of the front-end including handling some AJAX requests, javascript animations, and CSS tweaks.

## 10.3 ROB SHEEHY

10.3.1 I contributed the entirety of the Tournament back end logic , allowing for creation, editing, and deletion of tournaments. Most of the time I worked on the back end, however I worked on both sides of the tournament edit page. I contributed with the tournament search page and helping with setting up the data tables. I worked on any small pieces that needed to be worked on. Most of the time they would be small bugs helping ensure quality.

## 10.4 MARK GOLLNICK

10.4.1 I worked on displaying the tournament brackets using SVG and various Javascript rendering libraries including HighchartsJS and gRapheal. Before Max discovered D3.js (Data Driven Documents, formerly Protovis, a JavaScript rendering library) which proved to be much more flexible in the end. Mark also wrote the initial algorithm for turning the Python Tournament objects into a JSON format that could be queried and used by D3.

### 10.5 TAEHYUN PARK

10.5.1 I worked on the algorithm for creating tournament and printing models in JSON. For printing out models in JSON, I basically implemented a JSON encoder in Python. In addition to coding, I made the PowerPoints for the jump start presentation and the final presentation.