Design Document

Version 1.0 – 2014.02.09

*Created 2014.02.09*

Leaguer

Nathaniel Foy

Guntas Grewal

Tomer Kimia

Andrew Murell

Luke Shumaker

Davis Webb

Contents

[1 Purpose 3](#_Toc379754535)

[2 Non-Functional Requirements 3](#_Toc379754536)

[3 Design Outlines 3](#_Toc379754537)

[3.1 Design Decisions and Components 3](#_Toc379754538)

[3.2 Component Interaction 3](#_Toc379754539)

[4 Design Issues 3](#_Toc379754540)

[4.1 Scoring Algorithm 3](#_Toc379754541)

[4.2 Offline Data Management 4](#_Toc379754542)

[4.3 Fetching Data from Games 4](#_Toc379754543)

[5 Design Details 4](#_Toc379754544)

[5.1 Class Descriptions and Interactions 4](#_Toc379754545)

[5.2 UML Diagram of Classes 4](#_Toc379754546)

# Purpose

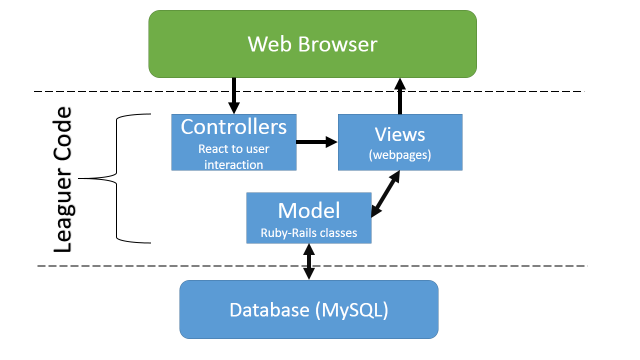
This document describes all components of the Leaguer Tournament management system. Leaguer is a software to be installed and run on a server. TODO. ANDREW COMPLETE THIS.

# Non-Functional Requirements

TODO Guntas. Email dunsmore and marco about this, then fill it out.

# Design Outlines

## Design Decisions and Components

Our system will on the [Model 2](http://en.wikipedia.org/wiki/Model_2) design pattern/architecture. TODO: Davis – add the purpose of EACH component as a list.

* Controllers – These will be….
* Models – The classes in the UML document below will residee in the model…
* Views – Views will be the HTML pages for Leaguer, and will

## Component Interaction

TODO – Luke you know more about Model 2. Help Davis with his section.

# Design Issues

## Scoring Algorithm

In an effort to keep our system broad, one of our requirements is that Leaguer is adaptable to many competitions, not just League of Legends. How do we assure that the different scoring systems of different sports are represented in Leaguer?

**Option 1:** One of our interfaces could be “Scoring System” which will be implemented by many classes with common scoring systems. For example there would be a implementing class in which the highest score wins, and one in which the lowest score wins. This is likely to be the winning option, as there are not too many obscure scoring systems that we could not think of.

**Option 2:** We could design an API in which the host writes a method to update the scoring. This is pretty complex, and while it would allow more customization, it is hard to imagine completing this task without first completing option 1.

## Offline Data Management

TODO – Nathniel write this

## Fetching Data from Games

TODO – Nathaniel write this.

# Design Details

## Class Descriptions and Interactions

**VIEWS**

**Webpage:** An abstract HTML file, all entries below are webpages (we represent them as subclasses of the abstract “Webpage” class. All webpages will send HTTP requests to the server. Most of the visual effects and update the display with Javascript methods. Each page will have a link to either the login or the logged in user’s page.

**Homepage**: This page has 3 basic options. Visually simple – two large buttons on a white screen, and a search bar above them. The search bar will allow you to search upcoming or current searchable tournaments. Log in (which will take you to the login page) and “Go to Tournament” in which you enter a tournament title. This interacts with the Homepage Controller.

**Login**: Page with form entries for username, password. If user clicks “new user” more forms entries will appear. One for repeating the password, and one for email. This interacts with the Login controller.

**Tournament:** A tree-like display of pairs of matches, where each match consists of a pair of teams. All users can click on a match to go to that match’s page. Host can see a gear on top left corner that represents tournament settings. This will open up more options for the host to change. This interacts with the tournament controller.

**Match:** A display of both teams.

**Server:** Rails’ Server class handles all HTTP events. Our Server class is the class that is the main program. It instantiates other classes, manages requests from Views, and runs static methods.

**User:** A class that represents someone using the Views (HTML, javascript) the user is in competitions and

## UML Diagram of Classes

TODO – I’m working on this – see images.pptx