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

[Purpose 5](#__RefHeading__282_532651186)

[Non-Functional Requirements 5](#__RefHeading__284_532651186)

[Design Outlines 5](#__RefHeading__286_532651186)

[Design Decisions and Components 5](#__RefHeading__288_532651186)

[Component Interaction 5](#__RefHeading__290_532651186)

[Design Issues 6](#__RefHeading__292_532651186)

[Scoring Algorithm 6](#__RefHeading__294_532651186)

[Offline Data Management 6](#__RefHeading__296_532651186)

[Fetching Data from Games 6](#__RefHeading__298_532651186)

[Design Details 6](#__RefHeading__300_532651186)

[Class Descriptions and Interactions 6](#__RefHeading__302_532651186)

[UML Diagram of Classes 6](#__RefHeading__304_532651186)

# Purpose

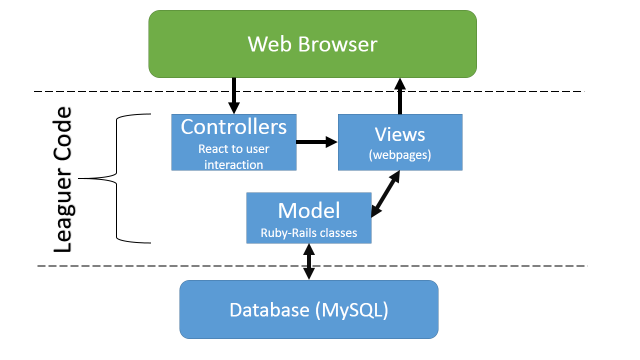
This document describes all components of the Leaguer Tournament management system. Leaguer is a software to be installed and run on a server. TODO. ANDREW COMPLETE THIS.

# 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 – The controllers will control any logic necessary to obtain the correct content for display. It then places the content in the request and decides which view it will be passed to.
* Models – The classes in the UML document below will reside in the model…
* Views – Views will be the HTML pages for Leaguer, and will display the users desired content inside of the web browser.

## Component Interaction

Controllers will be used to run all of the background work of Leaguer. They will fetch the necessary data and will tell the view what to do. We will be implementing seven controllers into Leaguer. Those will be:

I. PM & Alerts – This controller will be used for sending and receiving private messages to and from the host. Players will be able to message the host in order to inform him/her of anything during the tournament. This will also allow the host to post any notifications he or she desires that will be displayed for all to see.

II. Homepage – Used to handle the homepage. This will be the first web page seen by any user of the application.

III. Login – This controller will be used when a user attempts to sign in to their profile on Leaguer.

IV. Search – This controller will be used to search the web-base for on going tournaments, players and past tournaments.

V. Tournament – Used for setting up a tournament. This will be restricted to the host of the tournament.

VI. User – The controller that will take each user to their own profile.

VII. Match/Peer Review – used for gather game statistics and the separate player reviews.

# Each of these controllers will fetch the data specified by its separate section. The view will then be used to display all of this information, so Login will take the user to a login page, search will take the user to a search page and so on.

# 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

TODO – I will do this, but Andrew you will guide me through some of the ideas

Note – All of these classes are represented in the “Model” part of the Model 2 software pattern.

**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