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 3](#__RefHeading__282_532651186)

[Non-Functional Requirements 3](#__RefHeading__284_532651186)

[Design Outlines 3](#__RefHeading__286_532651186)

[Design Decisions and Components 3](#__RefHeading__288_532651186)

[Component Interaction 3](#__RefHeading__290_532651186)

[Design Issues 3](#__RefHeading__292_532651186)

[Scoring Algorithm 3](#__RefHeading__294_532651186)

[Offline Data Management 3](#__RefHeading__296_532651186)

[Fetching Data from Games 3](#__RefHeading__298_532651186)

[Design Details 4](#__RefHeading__300_532651186)

[Class Descriptions and Interactions 4](#__RefHeading__302_532651186)

[UML Diagram of Classes 4](#__RefHeading__304_532651186)

# Purpose

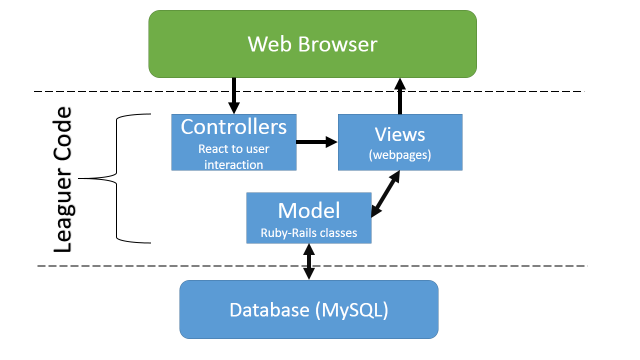
This document describes all components of the Leaguer Tournament management system. Leaguer is a software to be installed and run on a server. TODO. ANDREW COMPLETE THIS.

# 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 – 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.

The basic set up of our project will be the Model 2 design pattern for web applications. This is broken up into five distinct parts:

1. Controllers

The controller sections of the Model 2 design performs any logic necessary to obtain the correct content for display. It then places the content in the request and decides which view it will pass the request to. We will be implementing seven controllers into Leaguer. Those will be:

I. PM & Alerts – This controller will be used to handle private messages to the host of the tournament and alerts created by the host for the players/spectators to see.

II. Homepage – Used to handle the front page interactions with the website.

III. Login – When the users prompts for this, they will be taken to a separate login page

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.

2. View

The view is used to render the content passed by the controller. In Leaguer, this will be the separate web pages of the tournament website.

3. Model

4. Database

This will be the leaguer server which will be used to back up and store tournament information. This will be built by the host of the tournaments.

5. Web Browser

I really am not sure what to put here.

# 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