Software Design Document

Dota Track

Michael Eaton, Hazen Johnson, Sarah Witty

September 24, 2014

Contents

1 Summary

Dota Track is a Dota statistics application distinguished it's ability to give historical data for a player's history. This will allow players to track how they have improved over time, and as a result, to become better players. The main client for this Software Engineering 2 project is the Letu Moba Club; although the application will be general enough for any Dota player to use. This document lays out the entire design of the **Dota Track** application.

1.1 References

This document is based on the **Dota Track** SRS. References to this document will be noted as (SRS X.X.X) indicating relevant sections in the SRS.

2 Front-end Design

2.1 Input

- Users
  - Steam login information
  - Farm priority tags
- Steam Guard
  - Login using this API for basic authentication
- Dota 2 Web API
  - Match information (SRS 2.1.1)

2.2 Outputs

- Display statistics to the user
  - Graphs (SRS 2.1.3 - 2.1.4)
  - Match information
  - Explanatory information/settings

2.3 Responses

- Users
  - If the user fails to supply valid login information according to Steam Guard, the application will notify the user that their login information is incorrect.
  - If the user does not enter a farm priority tag after visiting a match page, the application will disregard the tag and not force a default value.
- Steam Guard
  - The application will send login information to Steam Guard and will not proceed until verification returns from Steam Guard. A waiting indication will be displayed while the application waits for a response from Steam Guard.

- Dota 2 Web API
  - When the user logs in, the application will immediately request an update for the users matches. The application will not prevent the user from viewing statistics currently available. While the application is waiting for data from the Dota 2 Web API, the user will be notified that his/her data is being loaded. When data has been obtained, the user will likewise be notified that their data has been updated.

3 Architectural Design

3.1 Platforms

The server will run on an Apache2 web server backed by a MySQL database. The server-side programming language used will be PHP using the Kohana Framework.

The client will run on any HTML5 web browser, targeting specifically Chrome because of Chrome's cross platform capabilities. Graphing will be accomplished using the D3 javascript library and jQuery will be used to simplify client side scripting.

Json will be used as the main data exchange format between the server and the client.

3.2 Major Components/Layers

The **Dota Track** application consists of two main layers. The web server application which negotiates between the client and the database and the database and the web APIs. The client application runs on the client device taking information from the server and generating relevant graphs dynamically.

3.2.1 Server-Side Application

The server side application consists of a controlling application component and a database component.

The controlling application component follows the MVC model and is constructed of the following modules:

- Models
  - Match information - This model contains information about a single match (SRS 2.1.1).
  - Player information - This model contains information about a player during a single match.
  - Series information - This model contains a series of matches.
  - Model implementations - The models mentioned above will contain support for the following interactions:
    * This model will implement interaction with the Dota 2 Web API. (Obtains information.)
    * This model will implement interaction with the database. (Obtains information and sends information)
    * This model will implement interaction with the client application. (Obtains information and sends information)
- Views

  - Login screen
    - Failed to login message
    - Timeout notification message
    - Steam Guard code acceptance box
  - Matches screen
    - Selecting priority tags from the match table
    - Loading new matches
  - Statistics screen
    - Graph zoom
    - Filter selection box
  - Match screen
    - Graph zoom
    - Selecting priority tags
  - Settings
    - FAQ

- Controller - The controller operates as a REST (Representational State Transfer - or a stateless) API service. Requests are serviced immediately and the controller returns back to a waiting state.
  - Serves interfaces
    - Basic views are served in response to requests from the client application in the form of HTTP requests.
    - Authenticates users from the login screen with Steam Guard. Maintains a session for the user after authentication, until logout or timeout.
  - Serves information to the client application
    - Populates views served with data based on the session id and user requests from the client application.
    - Queries and returns matches based on criteria from/to the client application.
  - Receives information from the client application.
    - Supports adding farm priority tags to the database.
  - Regulates traffic to external API services.
    - Loads matches from Dota 2 Web API when it does not find relevant matches in the database.
    - Regulates, by means of the models, API calls to less than 100,000 queries per day (SRS 2.2.1).
    - Daily queries the Dota 2 Web API in order to catch new matches for each user of the application.

The database component will store match and player information. Access to the database will be regulated by the controller application component.

3.2.2 Client-Side Application

The client side application will consist of a graphing module, an Ajax module, and response/interactivity module.

The graphing module will attach the appropriate information to the relevant HTML elements to generate graphs.

- It will fill specified HTML elements with relevant data and format this into a graph
  - A general class of HTML elements will be used to identify graph elements.
  - The specific ID of each element will determine the information source and graph type.
- It will enable user interaction with the generated graphs.

The Ajax module will negotiate with the server layer.

- It will obtain relevant data for graphs.
- It will obtain sub-forms and popups for interactivity.

The response/interactivity module will make the client application interactive.

- It will display sub-forms and popups.
- It will display loading icons as required.
- Any other reactivity of the application will be handled by this module.

3.3 Component Interaction

3.3.1 Client-Side Interactions

The interactivity/responsive and graphing modules interact with the Ajax module.

They will:

- Interpret user input into requests for the Ajax module. These requests are driven by:
  - Query filters (such as search criteria)
  - View state (current screen of the application)
- Format the resulting data from the Ajax module into its final form.

3.3.2 Inter-Layer Interactions

The Ajax module on the client-side will interface with the controller application module on the server side.

It will:

- Receive requests from other client-side components.
- Send queries for requested data to the controller application.
- Receive results from the controller application, interpreting the results into a usable client-side object.

  - The controller application will send the data in the form of Json data files.
  - The Json files will be interpreted into Javascript objects to be used by the client-side application.
  - The resulting object will be immediately attachable to graph objects without further manipulation.

- Pass the results to the requesting portion of the application.

3.3.3 Server-Side Interactions

The controller application module will interface with the database component.
It will:

- Receive requests from other portions of the controller application and/or the client-side Ajax module.
- Resolve any queries against the database.
- Pass the results to the requesting portion of the application.

3.4 Interfaces

3.4.1 Steam Guard

The server-side controller application component will interface with the Steam Guard API to provide client authentication in every case when authentication is necessary as determined by the Steam Guard API.

3.4.2 Dota 2 Web API

The server-side controller application component will interface with the Dota 2 Web API, submitting requests to the API and storing the resulting information in the server-side database component. The controller will store only information laid out in the SRS into the database.
The Dota 2 Web API provides the following match structures in Json:

- Players
  - Account ID (32bit Steam ID)
  - Player Slots (8bit packed flags - dire/radiant and two 0-4 values representing player slot)
  - Hero ID (Numeric hero ID)
  - Item 0 (Numeric item ID top left slot)
  - Item X (etc...)
  - Item 5 (Numeric item ID bottom right slot)
  - Kills (Integer)
  - Deaths (Integer)
  - Assists (Integer)
  - Leaver Status (0-6, abandon reason)
  - Gold (Integer - gold left at end of match)

  - Last Hits (Integer)
  - Denies (Integer)
  - Gold per Minute (Integer?)
  - Experience per Minute (Integer?)
  - Gold Spent (Integer - total gold spent)
  - Hero Damage (Integer)
  - Tower Damage (Integer)
  - Hero Healing (Integer - damage healed on teammates)
  - Level (Integer)
  - Ability Upgrades
    * Ability (Numeric ability ID)
    * Time (Integer - seconds since beginning of match when point was spent)
    * Level (Integer - level of hero when ability was leveled
  - Additional units
    * Unit Name (String name of unit)
    * Items 0-5 (Numeric ID of items - see above for order)
- Season (?)
- Radiant win (Boolean)
- Duration (Integer - total time of match in seconds)
- Start time (Integer - Date - time in UTC seconds since Jan 1, 1970 - Unix Time Format)
- Match ID (Numeric match ID)
- Match Sequence Number (Integer - order in which matches are recorded)
- Tower Status Radiant (Integer - 11bit unsigned - bit flag)
- Tower Status Dire (Integer - 11bit unsigned - bit flag)
- Barracks Status Radiant (Integer - 6bit unsigned - bit flag)
- Barracks Status Dire (Integer - 6bit unsigned - bit flag)
- Cluster (Integer - replay statistic)
- First blood time (Integer - time in seconds at which first blood occurred)
- Replay salt (N/A)
- Lobby Type (?)
- Human Players (Integer - number of human players)
- League ID (0 - Tournament only)
- Positive Votes (Integer - number of thumbs up)

- Negative Votes (Integer - number of thumbs down)
- Game Mode (Integer - representative of game type)

API details may be found at http://dev.dota2.com/showthread.php?t=58317 or https://www.mashape.com/community/dota-2-steam-web.

3.4.3 User Interface Design

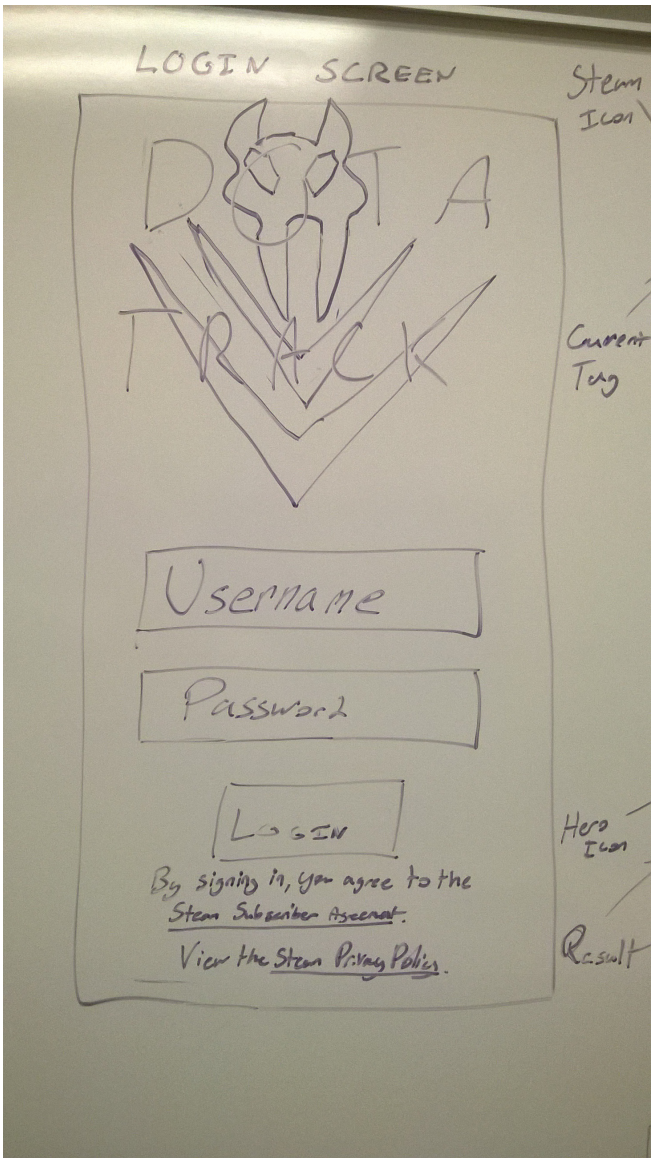

Figure 1: Login Screen

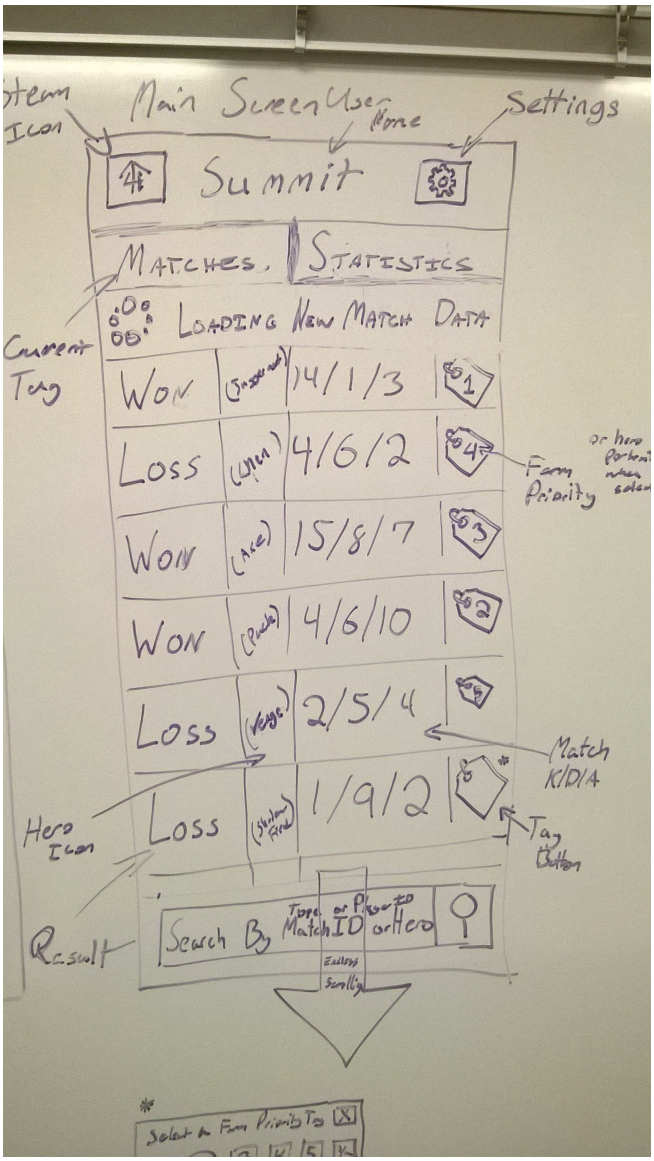

Figure 2: Main Matches Screen

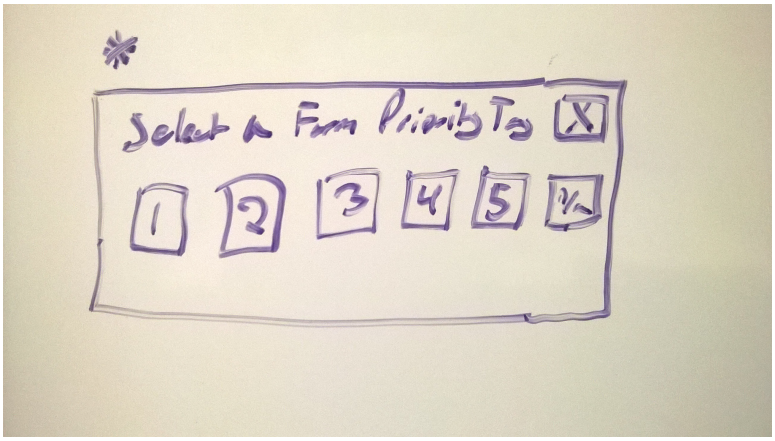

Figure 3: Farm Priority Tag Popup

Figure 4: Main Statistics Page

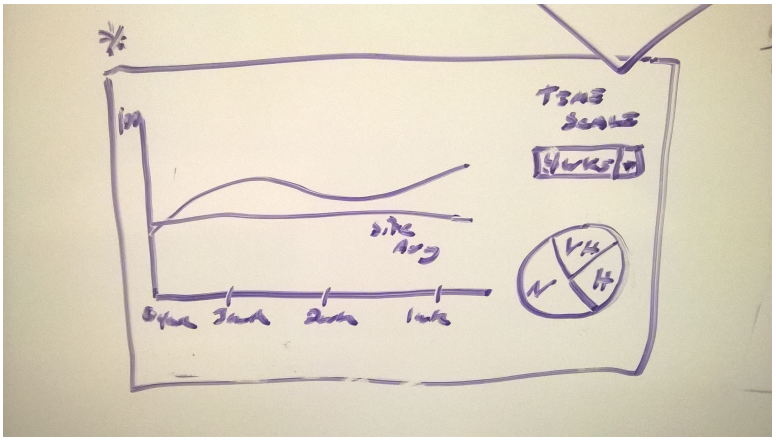

Figure 5: Graph Popup

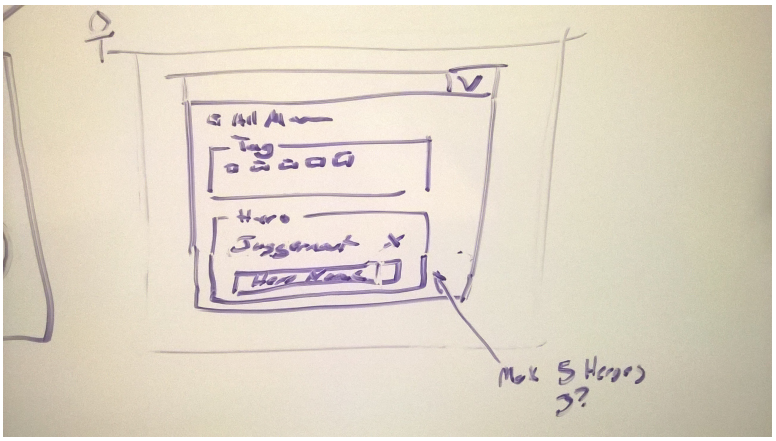

Figure 6: Match Filter Popup

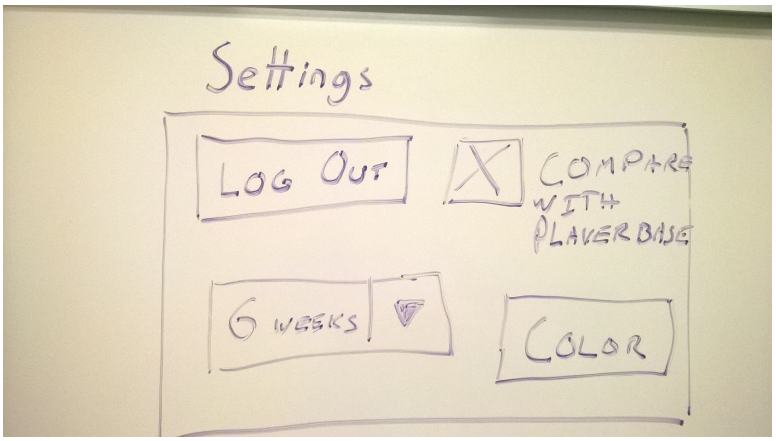

Figure 7: Settings Page

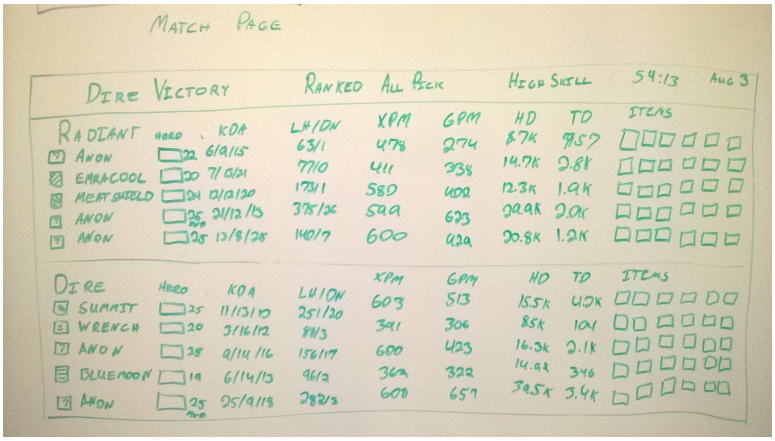

Figure 8: Match Page

4 Detailed Component Design

4.1 Controller Application Module

This component is designed according to the MVC (Model View Controller) architecture.

4.1.1 Model

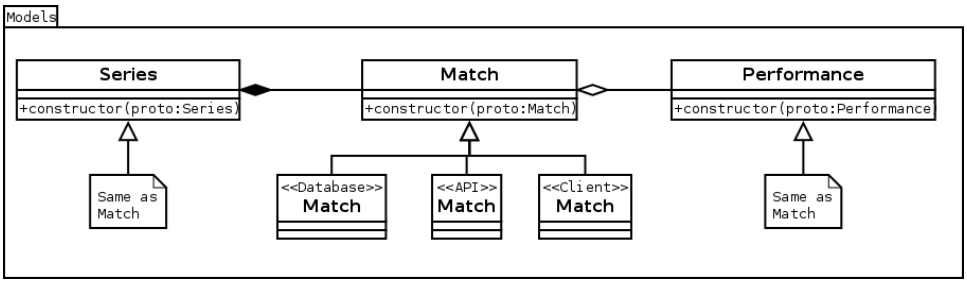

Figure 9: Model UML

Models encapsulate the information from various data sources for interchange around the controller application.

Each generic interface (Matches, Performance, and Series) is implemented for each data source/interface (Database, API, Client).

Models may be copied from any other form of the same generic model so that data can be obtained from any source and then written back to any other source.

Different data sources have different read/write capabilities:

- Database - Read and write
- API - Read only
- Client - Read and write (send)

4.1.2 Controller

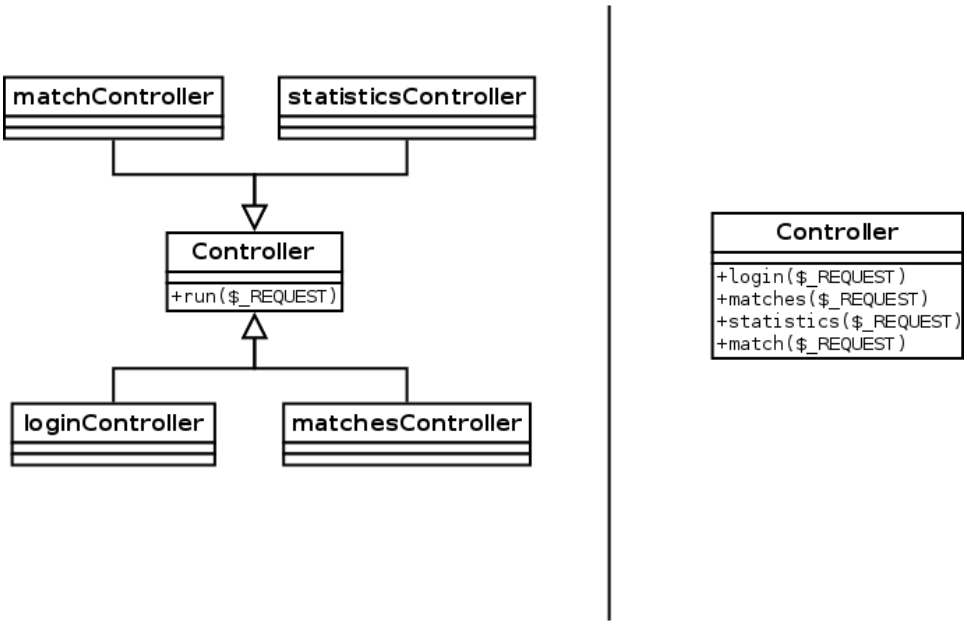

Figure 10: Controller UML

The controller accepts request URLs in the form http://www.dotatrack.com/login such that the loginController or login() function are called in response.

The exact design pattern used in the context will be determined by the needs of the customer and the Kohana framework. Gathering of this information is in progress.

4.1.3 Views

Views will be called and populated by the appropriate controller.

See the User Interface Design (3.4.3) for diagrams of the view design.

4.2 Database Module

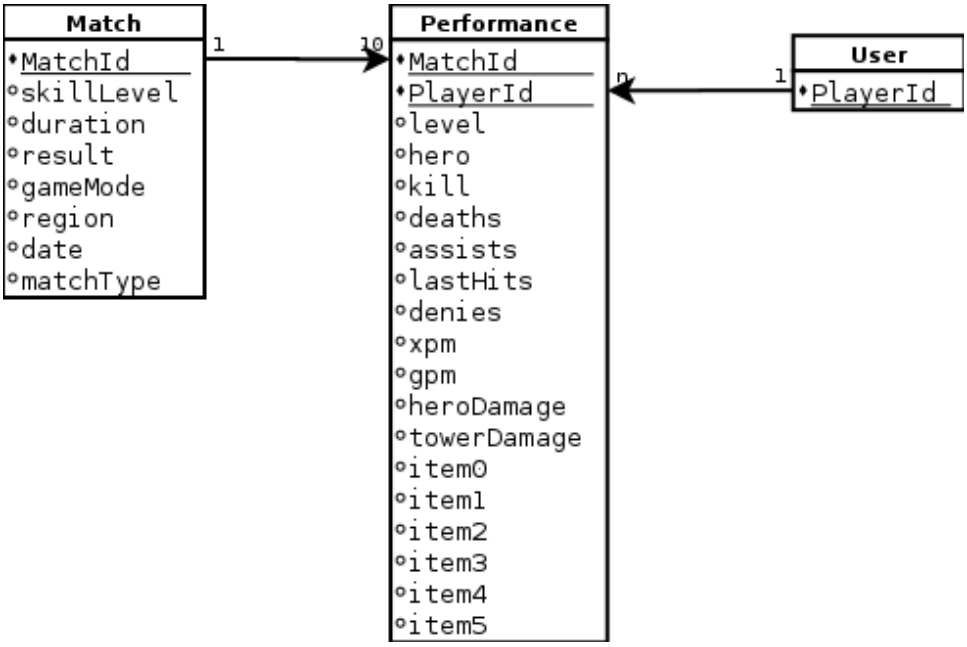

Figure 11: Database UML

The user table is put here to store possible user specific information in the database, but this is not imperative for the current design.

4.3 Ajax Module

The Ajax module contains functions to perform the following operations.

- getMatches(selectFilter, projectFilter) - This function gets all matches fulfilling the criteria in the selectFilter returning all information specified in the projectFilter.
- getSubView() - This function gets the requested HTML sub-form (pop-up or other interactive element) to insert into the current view.
- updateMatch(selectFilter, updateFilter) - This function updates the server database for all matches fulfilling criteria in the selectFilter with information stored in the updateFilter.

4.4 Graphing Module

On document load and on any change to criteria, the graphing module will:

- Scan the view for any elements tagged with the “graph” class attribute.
- Query for data according to the id attribute and any refining criteria stored by the interactivity/responsiveness module.
- Attach data from the Ajax module to the graph element in the view.

- Format the graph using D3 inside the graph element according to its type as defined by the following class attributes:
  - "line"
  - "pie"
  - "zoomLine"
  - "zoomPie"
- Register each graph element to listen for updates to the search criteria.

Each graph type will be formatted using a specialized function that takes the data associated with a graph element and generates the correct type of graph.

4.5 Interactivity/Responsiveness Module

The interactivity/responsiveness module will contain a working set of search criteria. This will be stored as a javascript object.

This working set will also be accompanied by a registry of listening objects so that when the search criteria is updated, all listeners can be easily update.

On document load, the interactivity/responsiveness module will:

- Register view elements with appropriate CSS selectors to the appropriate event handlers.
  - "graphHelp" - will be registered with a function which gets a zoomed view of the graph and help text in a pop-up.
  - "popUp" - will be registered with a function which reads the element id to determine which pop-up should be obtained from the server and loads this as the next sibling element of the pop-up element.
  - "loading" - will be registered with a function which checks regularly on the status of the process indicated by the id until the process is complete.
  - "filter" - will be registered with a function that updates the search criteria and causes all listening objects to update.
- Register any other elements of interactivity such as
  - Folding menus
  - Any other client-side interactivity (to be added as time permits).