2016

Edinburgh Napier University

40110560

[LEAGUE OF STATS]

This document contains the report for second coursework - a web application for the Advanced Web Technologies module.

Contents

Introduction – League of Stats

As an avid fan of the online video game – League of Legends, I wanted to create a website which would help more advanced players find statistics and information about the game and to improve their play.

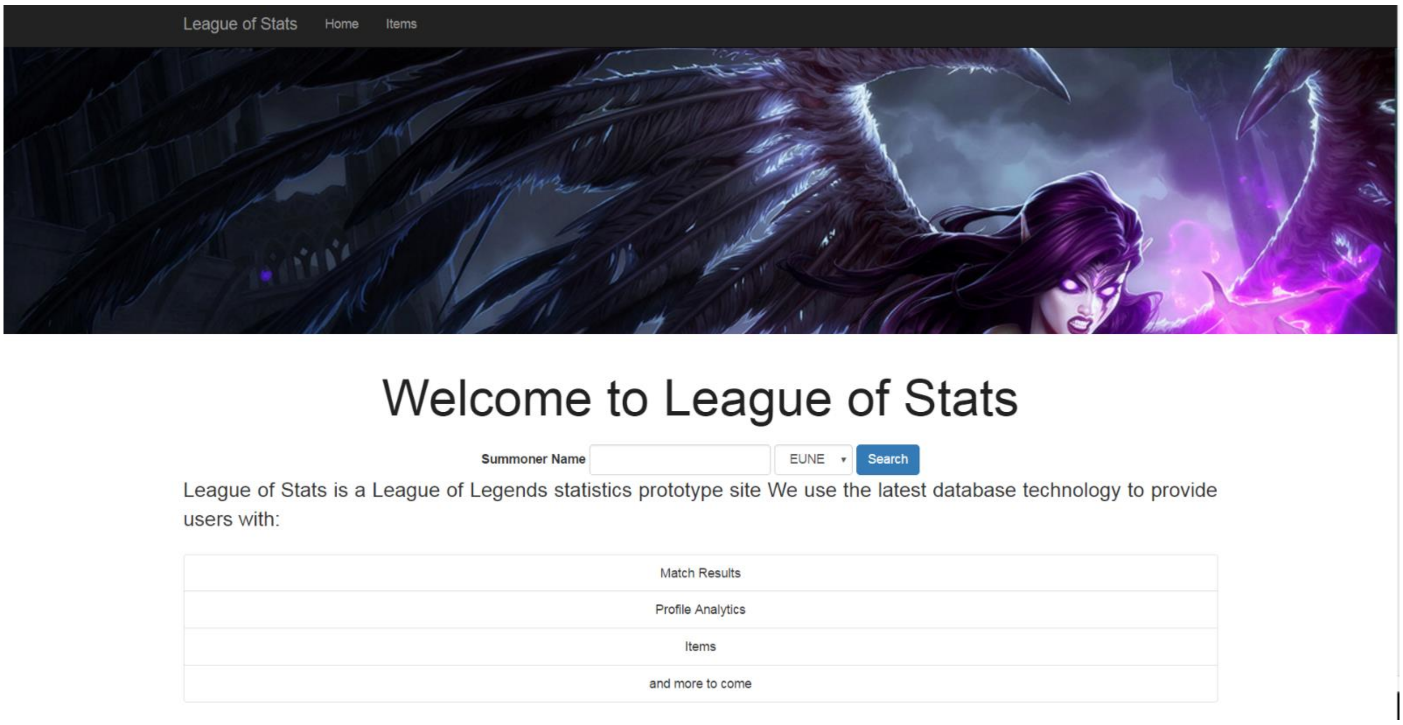
The League of Stats is a web application which utilises the API provided by the company that created the game – Riot Games. It also uses a python library over the API called Cassiopeia. At the moment the application supports the 3 main regions of the game – East Europe, West Europe and North America, more can be added in the future.

The app is written in Python utilising the Flask framework with the help of Jinja2 templating, HTML, JavaScript and Bootstrap framework. It also uses Flask-Alchemy database integration an extension of SQLAlchemy for Python. SQLite was chosen as the langue of the database.

Design

As suggested in the workbook, the Bootstrap Framework was a great starting point and provided most the functionality regarding the looks of the website.

The landing page of the website consists of a short description of the website as well as form. The form has a field in which the user must input a Summoner Name (in game usernames) for which he wants to retrieve information. They also have to choose which server the Summoner is playing on, as the Summoner might not exist on other servers.

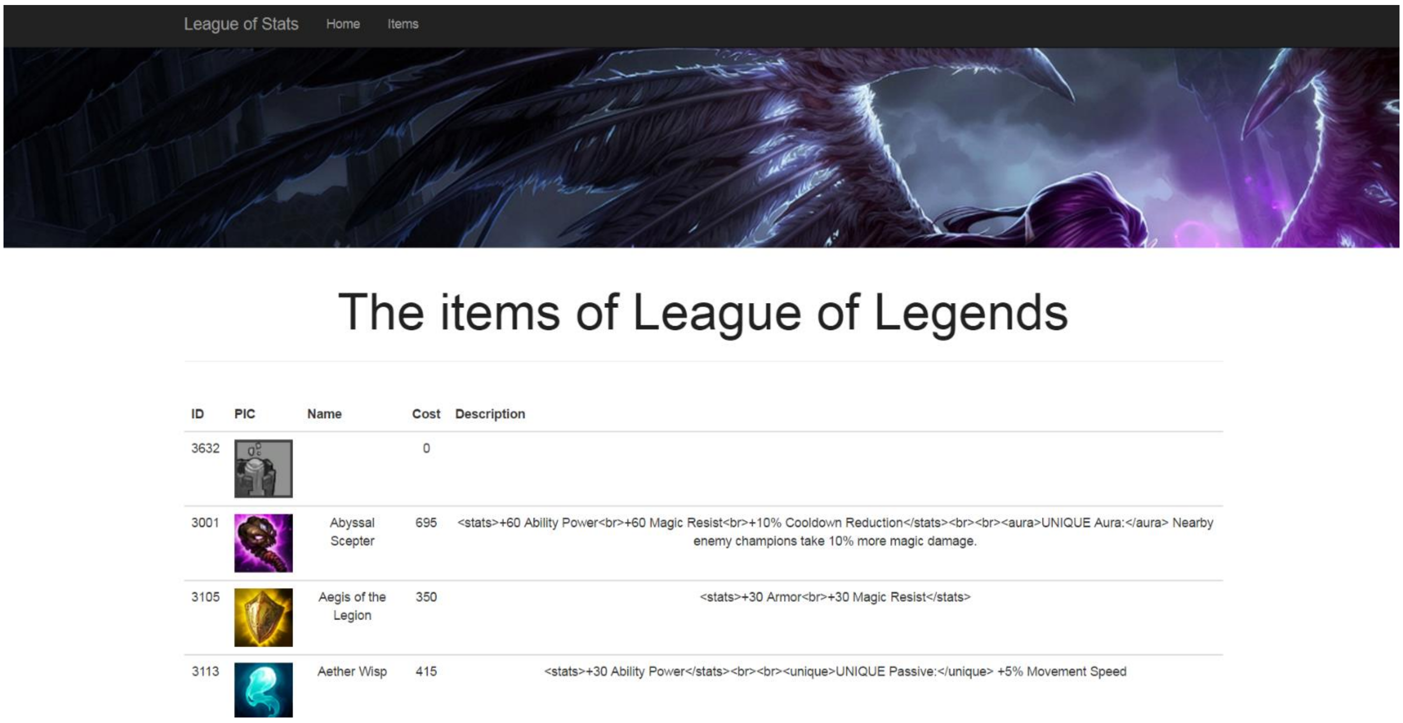


Welcome to League of Stats

League of Stats is a League of Legends statistics prototype site We use the latest database technology to provide users with:

Match Results
Profile Analytics
Items
and more to come

The items page has a big table in which all Item from the game can be looked up for information. The table has a column for each of the items' properties: ID, image, in-game cost and description.



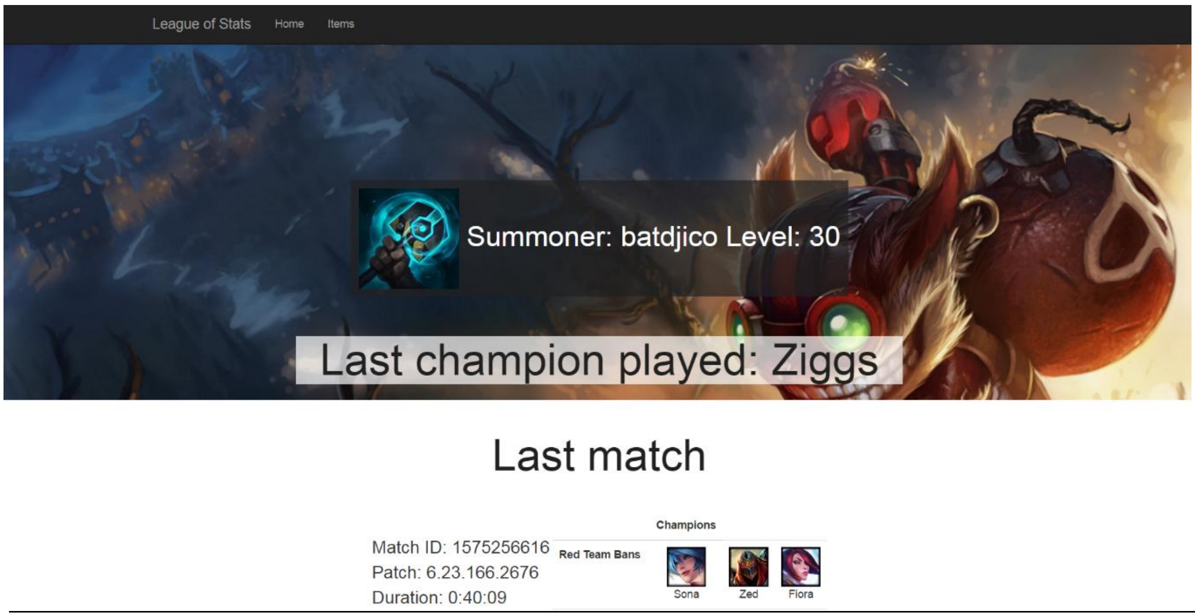
The items of League of Legends

ID	PIC	Name	Cost	Description
3632			0	
3001		Abyssal Scepter	695	<stats>+60 Ability Power +60 Magic Resist +10% Cooldown Reduction</stats> <aura>UNIQUE Aura:</aura> Nearby enemy champions take 10% more magic damage.
3105		Aegis of the Legion	350	<stats>+30 Armor +30 Magic Resist</stats>
3113		Aether Wisp	415	<stats>+30 Ability Power</stats> <unique>UNIQUE Passive:</unique> +5% Movement Speed

As the data provided by the API is in JSON, I wasn't able to create a proper REGEX (regular expression) which to use in order to filter out the "junk" tags, before storing the information in the database.

The summoner page is the main feature of the application. On it the user can view the basic information about the summoner name that he has searched for – The summoner name, the level of the account as well as the in-game icon of the profile. The page mentions the last champion that was played by that summoner and background of the page is the image of the last champion which the summoner has played.

Currently the application displays information about only the last match which was played by the searched summoner. The user can see on which Patch(version of the game) the match was played, on which map as well as the duration of the game.



Summoner: batdjico Level: 30

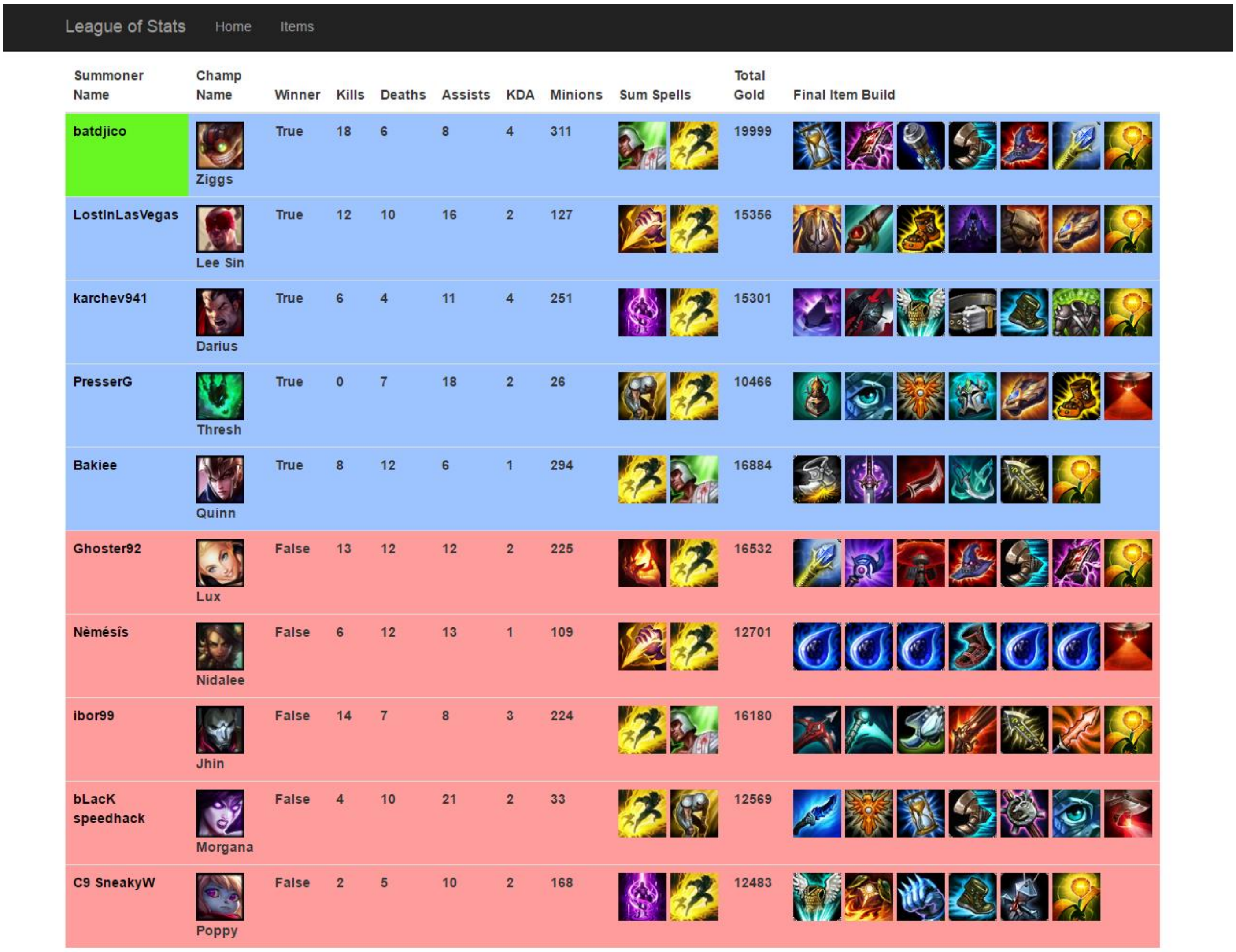
Last champion played: Ziggs

Last match

Match ID: 1575256616
Patch: 6.23.166.2676
Duration: 0:40:09

The user can see which champions were Banned (were not allowed to be picked by players).
In the participants table it is displayed the information about the state of each player and their end game results. Which include:

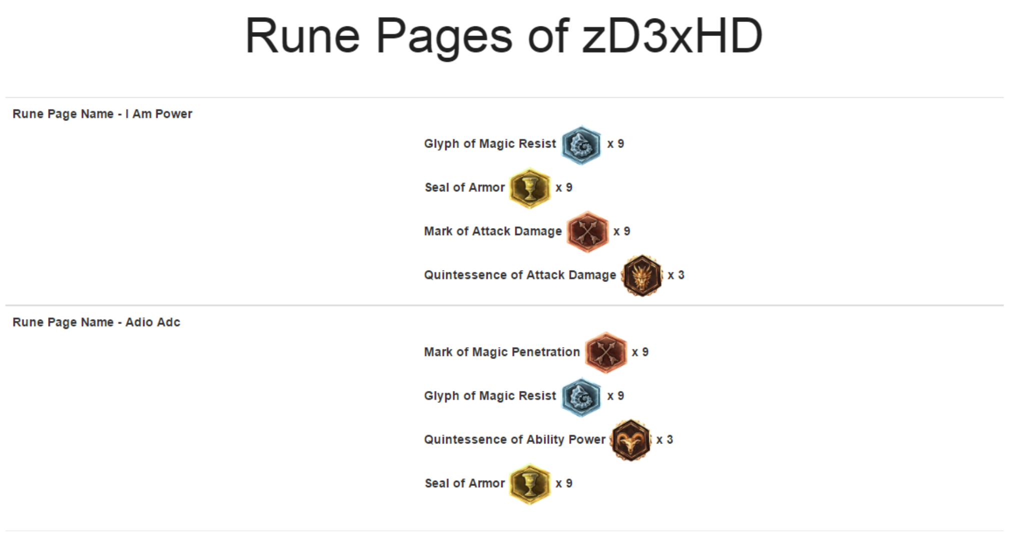
- Champion played name and image
- If they won or lost the game
- Kills/Deaths/Assists
- Minions slain
- Their Summoner's Spells (Abilities which aren't tied to specific champion and are chosen before the game starts)
- Total Gold
- Final Item build



Summoner Name	Champ Name	Winner	Kills	Deaths	Assists	KDA	Minions	Sum Spells	Total Gold	Final Item Build
batdjico	Ziggs	True	18	6	8	4	311		19999	
LostInLasVegas	Lee Sin	True	12	10	16	2	127		15356	
karchev941	Darius	True	6	4	11	4	251		15301	
PresserG	Thresh	True	0	7	18	2	26		10466	
Bakiee	Quinn	True	8	12	6	1	294		16884	
Ghoster92	Lux	False	13	12	12	2	225		16532	
Nèmésîs	Nidalee	False	6	12	13	1	109		12701	
ibor99	Jhin	False	14	7	8	3	224		16180	
bLacK speedhack	Morgana	False	4	10	21	2	33		12569	
C9 SneakyW	Poppy	False	2	5	10	2	168		12483	

The current summoner is also highlighted in green for easier spotting. Each of the summoners' names in the table is also a link which will lead to their summoner page information.

At the bottom of the summoner's page the information about their Runes (Additional enhancements for the champions which they have chosen). Each summoner can have up to 20 different Rune pages which they can customise to suite their champion's in-game needs.



Rune Pages of zD3xHD

Rune Page Name - I Am Power

Glyph of Magic Resist x 9

Seal of Armor x 9

Mark of Attack Damage x 9

Quintessence of Attack Damage x 3

Rune Page Name - Adio Adc

Mark of Magic Penetration x 9

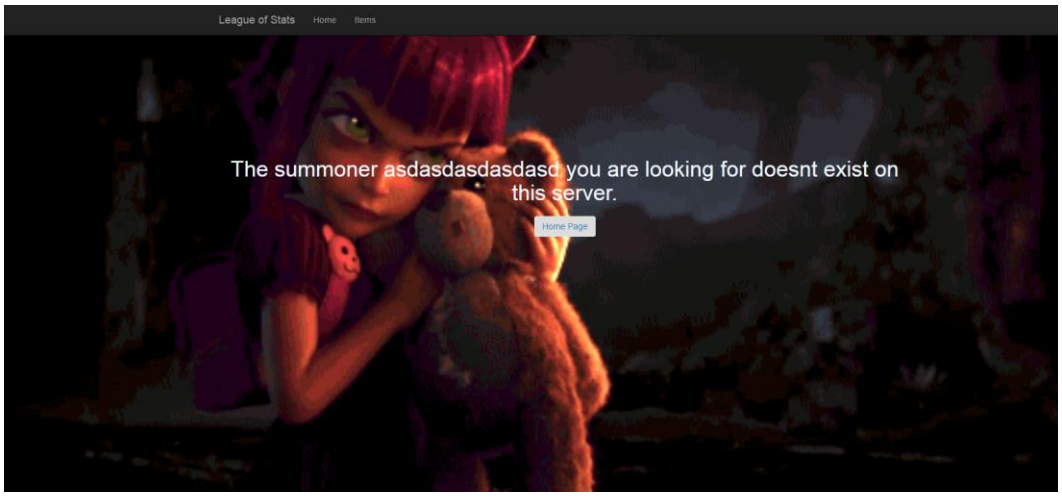
Glyph of Magic Resist x 9

Quintessence of Ability Power x 3

Seal of Armor x 9

Implementation

The landing page of the application uses the input from the form in order to fetch the data for the requested summoner. If the form is left blank and error will be displayed. The application checks the existing database for summoner name and if it is stored it returns the data associated with it. If not, the API is called and the data is drawn from it and saved to the database. Currently the database only saves basic information about the summoners: id, profile icon id, level, summoner name. The database also has a table for all items which takes its information from a JSON file provided by Riot Games for "Static" data (data which changes very rarely). In the future more "Static" data can be added to the database e.g. champion information (abilities, lore). The application tries to catch any 404 errors returned by the API. If a summoner name which doesn't exist is entered in the form, the user will be redirected to a 404 page. The background of the 404 is Annie with her bear Tibbers, one of the most beloved characters in the game.



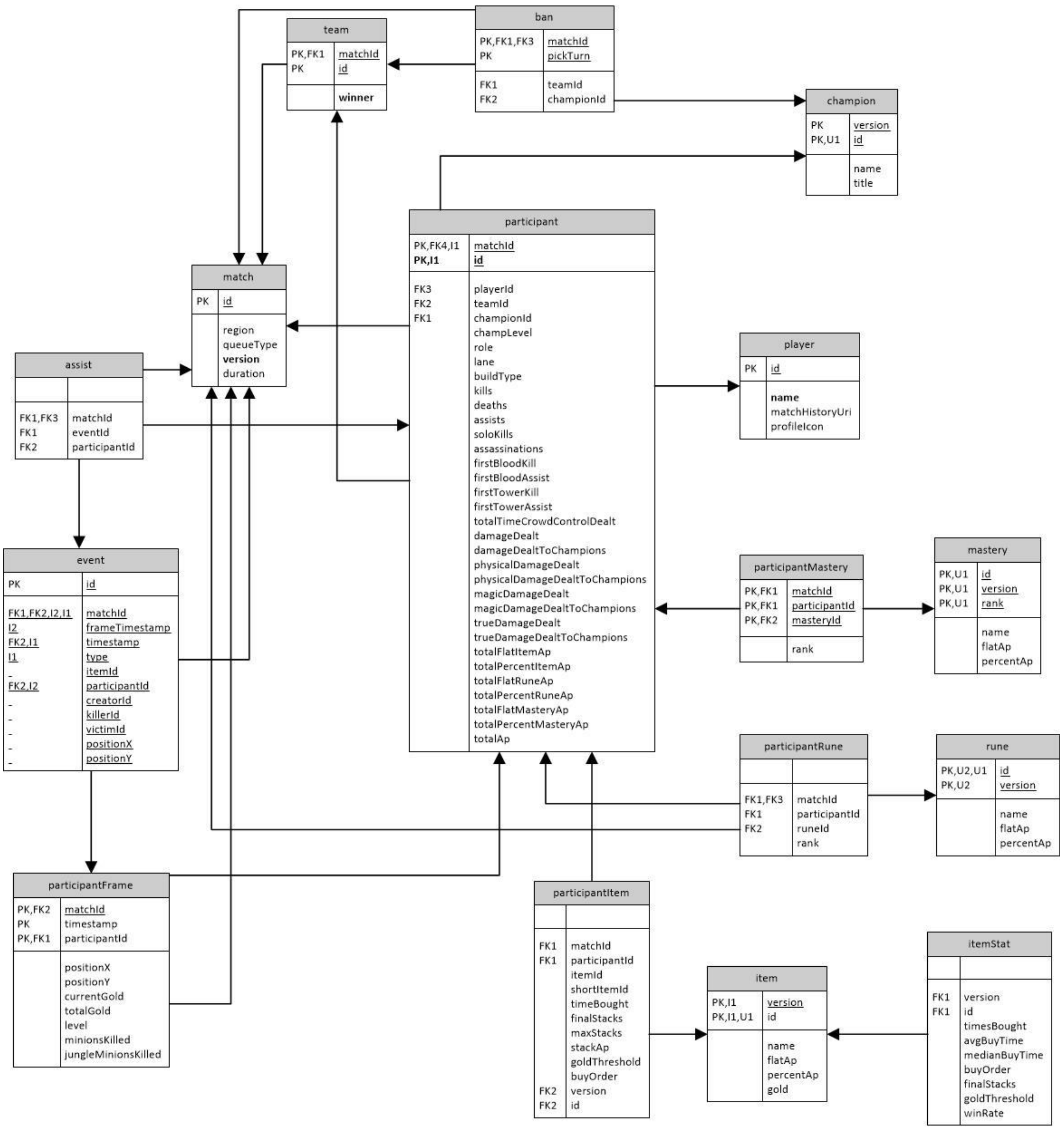
The summoner asdasdasdasdasd you are looking for doesnt exist on this server.

The URL hierarchy was built to be consistent. The root URL is '/' and it redirects the user to landing page of the app. Once the user has entered the correct summoner name and region, both are sent as variables to the app in order to create the hierarchy – region/summonerName. After a search has been made the user is redirected to the summoner page. If the user enters invalid URL name, the web-app will open a custom error page for error 404.

Improvements

Improvements to the structure of the database will allow for faster functionality as well as additional features. For instance, the API allows the developers to pull the whole match history of the summoner which can be used to display not only one but up to the last 20 games and all information about them to the users.

I created the scheme needed to achieve this but was unable to develop it in practice due to time constraints.



More so it would be possible to write a crawler which takes the top 50 players of the ranked ladder in the game and go through their match history and for every game store the information about each summoner, and for each of those summoners do the same. This would allow for a very quick database creation with relevant information and would boost the performance of the app significantly.

Another feature would be to check if the summoner is currently in a game and if so pull of the information – champions at the moment and summoner statistics like ranked placement and how good this person is with their current champion by displaying their Win/Loss ratio in percentages. It is possible to create relations between the summoners and for instance if you can track the win/loss ratio between summoners who have played together.

Displaying the win/loss ratio for each champion that the summoner has played in the past 30 days is another feature that could be implemented and would give useful information.

Critical Evaluation

Overall, the website covers most of the topics which were discussed in the workbook provided in the module. I feel like there is a lot of work to be done on application but for the scope of the module it currently has enough functionality to provide the user with. It runs on Python using the Flask framework and Flask-Alchemy as database with SQLite. The app uses two variables in its URL hierarchy in order to display the correct information and Jinja2 template. Currently the app is not mobile friendly but looks decent on the desktop. I think it turned out pretty good as this was my first using an API.

At the moment the application handles 404 errors by redirecting the user to a custom page. It doesn't handle any other errors.

Personal Evaluation

Using python and jinja2 during these courseworks was very fun and I know that in the future these skills will definitely come in handy. When I started working on this assignment I wasn't familiar with APIs at all and had to find a way to learn how to use the Riot Games API. I found several Python libraries which sadly weren't being update very frequently and I chose Cassiopeia as it had the best documentation available. Luckily Cassiopeia had built in SQLAlchemy support which after some configuration and tinkering was ready to go for Flask-Alchemy. I didn't think I would need a database at first but the Riot API allows only for a certain amount of calls to be made to it in a period of time (Rate Limit(s): 500 requests every 10 minutes,10 requests every 10 seconds). This meant that if I wanted to make this large scale I had to store information the app has already found for later usage. However, the database creation was too complicated and the app will probably stop working if a lot of people are using it at the same time.

Stack overflow gave me the answers to many questions: How to import JSON data and store it in SQLite DB, how to traverse said data and store only the relevant parts that I needed, how to convert an sqlalchemy query to a python dictionary and a lot more. Web development can be very frustrating at times, but when everything works it is amazing!

Installation of Cassiopeia:

Cassiopeia can be found in PIP however, when I tried installing it this way it came up with a bunch of errors. In the Git repository I have included the installation files for both Cassiopeia as well as SQLAlchemy(Cassiopeia requires).

First install SQLAlchem then Cassiopeia by starting the virtual env and going in the folder of each packed and running -python setup.py

References:

Bootstrap:
https://bootstrapdocs.com/v3.3.5/docs/

Jinja2:
http://jinja.pocoo.org/docs/dev/

APIs Used:

https://developer.riotgames.com/api/methods#!/1208/4684
https://developer.riotgames.com/

Flask Alchemy tutorials:
https://www.youtube.com/watch?v=Tu4vRU4lt6k&list=PLXmMXHVSvS-BlLA5beNJojJLlpE0PJgCW

API libraries:
http://cassiopeia.readthedocs.io/en/latest/index.html
https://developer.riotgames.com/discussion/tutorials-libraries/show/q6pnzTpo

Custom Error pages:
http://flask.pocoo.org/docs/0.11/patterns/errorpages/