**System Design for Local RiotAPI**

# Table of Contents

[Table of Contents 1](#_Toc165493404)

[1 Introduction to System Design [TBD] 2](#_Toc165493405)

[1.1 What Tools to be Used? 2](#_Toc165493406)

[1.2 Architecture Design 2](#_Toc165493407)

[1.3 Key Features and Limitations 3](#_Toc165493408)

# Introduction [TBD]

The following document is intended to do a fully system analysis and design to build the FastAPI application for our purpose. The application is fully inspired by this [article](https://medium.com/data-engineer-things/exploring-starcraft-2-data-with-airflow-duckdb-and-streamlit-7c0ad79f9ca6), but I want to make a twist here. Our application is similar to the work above, but the objection is to build a similar work for the League of Legends here. So I think it is a local development of the application Blitz or op.gg. The whole application is my dedicated devotion to Python and LOL game together, and a bit of self-learning for a tool I want to achieve.

So why am I doing this? Well, just for learning actually, but probably let’s try how I can get a real-time data statistics during the match rather than relying on third-party applications. It is not because I don’t trust them but their update is not typically fast as they are having a lot of users around the world, and normal statistics can be updated frequently by batch instead of real-time match data

## What Tools to be Used?

I don’t plan to produce some production-grade products or doing things meaningful here, so the feature request and infrastructure for application (load balancer, Redis, …) would be ignored, but more focused on the technical design here. As I need to build a custom application that support querying the Riot API, an API web-based application and a small database seems enough. Since I am learning MongoDB, this would be a great opportunity to start off, but for the better design I recommend to use DuckDB or SQLite if necessary. The data inside are not critical, so probably after the first try, I would backup the database for initial seeding.

## Architecture Design

Our architecture design are perform as follows: The backend would keep querying the custom wrapper of Riot API (that is specifically designed to suit our need), and store the data into the MongoDB database server. The backend service would keep running forever to collect the data. So whenever the frontend calling the data, probably we don’t need to request the backend to provide the data, or at least that what we hopes for.

A diagram of a software application

Description automatically generated

To be

# RiotAPI: Wrapper Design

## What Tools to be Used?

In here want to

I don’t plan to produce some production-grade products or doing things meaningful here, so the feature request and infrastructure for application (load balancer, Redis, …) would be ignored, but more focused on the technical design here. As I need to build a custom

The following document is intended to do a fully system analysis and design to build the FastAPI application for our purpose. The application is fully inspired by this [article](https://medium.com/data-engineer-things/exploring-starcraft-2-data-with-airflow-duckdb-and-streamlit-7c0ad79f9ca6), but I want to make a twist here. Our application is similar to the work above, but the objection is to build a similar work for the League of Legends here. So I think it is a local development of the application Blitz or op.gg. The whole application is my dedicated devotion to Python and LOL game together, and a bit of self-learning for a tool I want to achieve.

So why am I doing this? Well, just for learning actually, but probably let’s try how I can get a real-time data statistics during the match rather than relying on third-party applications. It is not because I don’t trust them but their update is not typically fast as they are having a lot of users around the world, and normal statistics can be updated frequently by batch instead of real-time match data

# Introduction [TBD]

The following document is intended to do a fully system analysis and design to build the FastAPI application for our purpose. The application is fully inspired by this [article](https://medium.com/data-engineer-things/exploring-starcraft-2-data-with-airflow-duckdb-and-streamlit-7c0ad79f9ca6), but I want to make a twist here. Our application is similar to the work above, but the objection is to build a similar work for the League of Legends here. So I think it is a local development of the application Blitz or op.gg. The whole application is my dedicated devotion to Python and LOL game together, and a bit of self-learning for a tool I want to achieve.

So why am I doing this? Well, just for learning actually, but probably let’s try how I can get a real-time data statistics during the match rather than relying on third-party applications. It is not because I don’t trust them but their update is not typically fast as they are having a lot of users around the world, and normal statistics can be updated frequently by batch instead of real-time match data

# Introduction [TBD]

The following document is intended to do a fully system analysis and design to build the FastAPI application for our purpose. The application is fully inspired by this [article](https://medium.com/data-engineer-things/exploring-starcraft-2-data-with-airflow-duckdb-and-streamlit-7c0ad79f9ca6), but I want to make a twist here. Our application is similar to the work above, but the objection is to build a similar work for the League of Legends here. So I think it is a local development of the application Blitz or op.gg. The whole application is my dedicated devotion to Python and LOL game together, and a bit of self-learning for a tool I want to achieve.

So why am I doing this? Well, just for learning actually, but probably let’s try how I can get a real-time data statistics during the match rather than relying on third-party applications. It is not because I don’t trust them but their update is not typically fast as they are having a lot of users around the world, and normal statistics can be updated frequently by batch instead of real-time match data