LOL Helper

Software Architecture Documentation

Author: Yiming Liu

Version: 1.4

Last Edit: August 6th, 2022

Contents

Project Name

LOL Helper

Project Purpose

LOL Helper’s purpose is to help League of Legend players to know the things they want to know about the game. The game is changing with each patch, LOL Helper gives the information about the champion and items information up to date. LOL Helper helps new players to know the game easily.

Target Audience

LOL Players

Functionalities

- User can check Champion basic details
- User can get Items information
- User can get their friends’ statement
- User can see the up-to-date patch information
- User can know the history of summoner matches

Design Patterns

- Chain of responsibility
  - Checking the validation of the API key, version, language.
- Adapter pattern
  - Changing the information get from Riot API to user friendly format.
- Model-View-Controller pattern
  - User interface and control the data flow through database.

Technologies

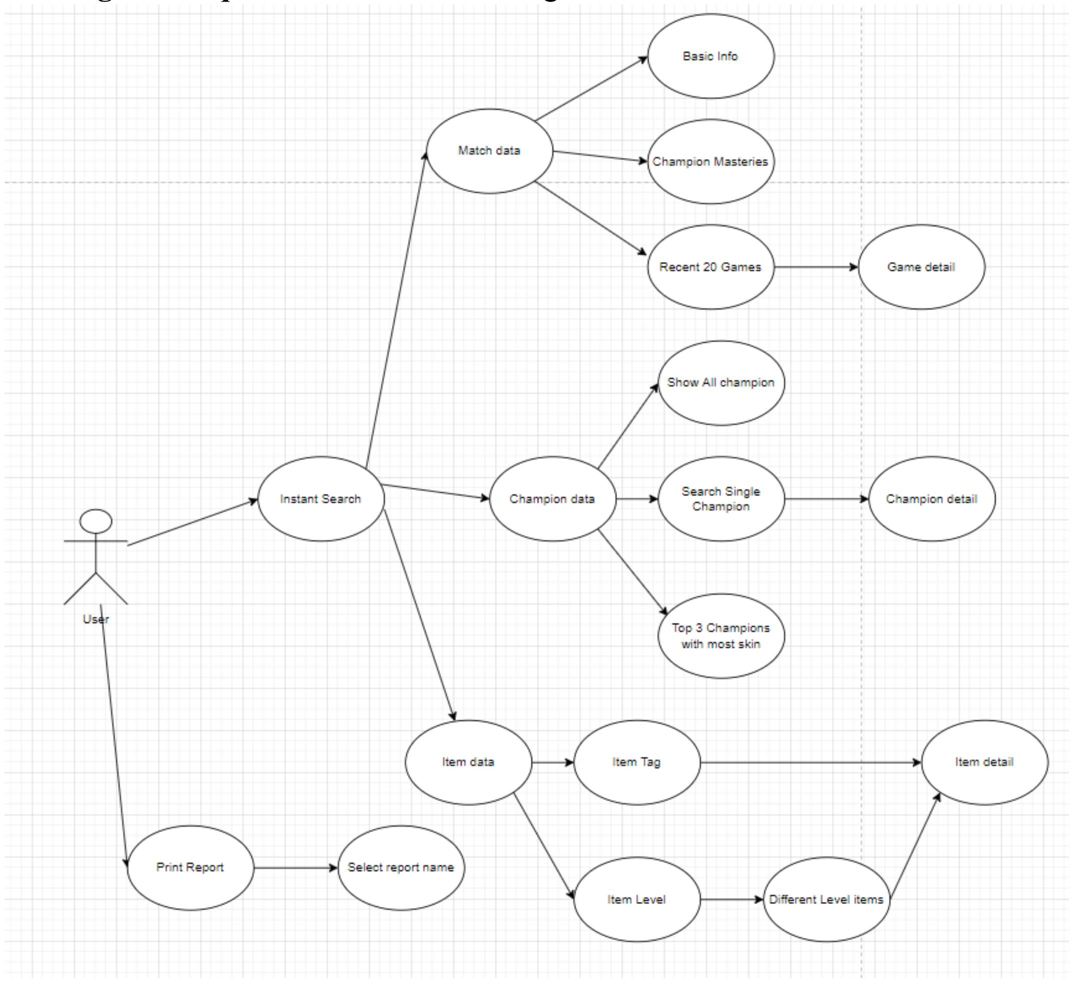
- Python
  - Implement the code
- Riot Developer Portal
  - https://developer.riotgames.com/docs/portal#_getting-started
  - For getting API information
- AWS Cloud SQL
  - https://docs.aws.amazon.com/ec2/?id=docs_gateway
  - Google cloud SQL for storing the information get from Riot

Requirements

	Functional Requirements
R01	Users shall be able to check champion's basic data by providing the champion's name.
R02	Users shall be able to check summoner's information by summoner name.
R03	Users shall be able to check item's basic information by providing the item's name.
R04	Users shall be able to save the items they select and the champion they select.
R05	Users shall be able to get the champion's skins by selecting the actual name of the skin or user can select the show all and user will see all the skins of the champion with the names of the skins.
R06	Users' saved selection will be stored and retrieved from a MySQL database.
R07	The application will remember the user choice history and stored on the cloud database.
R08	A configuration file will be used to store the information of log, database and API settings.
	Performance Requirements
R09	Single information of a champion or an item that selected by the user should be given the feedback in 1 second (The worst situation due to the internet). The combination (3 champions and 6 of their items combined) saved by the user should be given the feedback in 5 second.
	Traceability Requirements
R10	At least 75% of all application process should be stored in the history. Measured by (log methods/ total methods *100)

Scenario View

Concerns: Understanding the functions that User can use in the application.
Stakeholders: All stakeholders, especially the user.
Modeling Techniques: UML Use Case Diagram

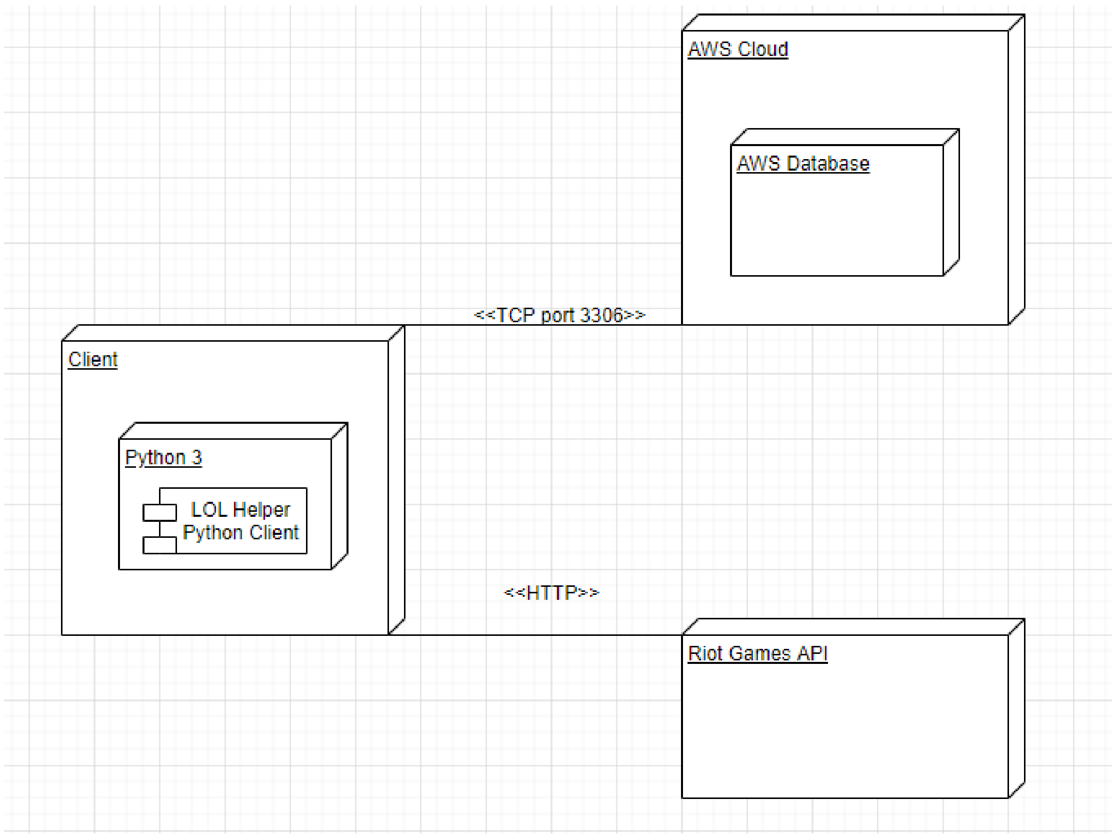


Users have two options. The first one is to just check the data instantly. The other one is to save the selected champion and items to combined.

Physical View

Concerns: Software to hardware connection, communication protocols and modules communication.
Stakeholders: Software developers.
Modeling Techniques: UML Deployment Diagram
The application communicates with the AWS database over TCP port 3306.

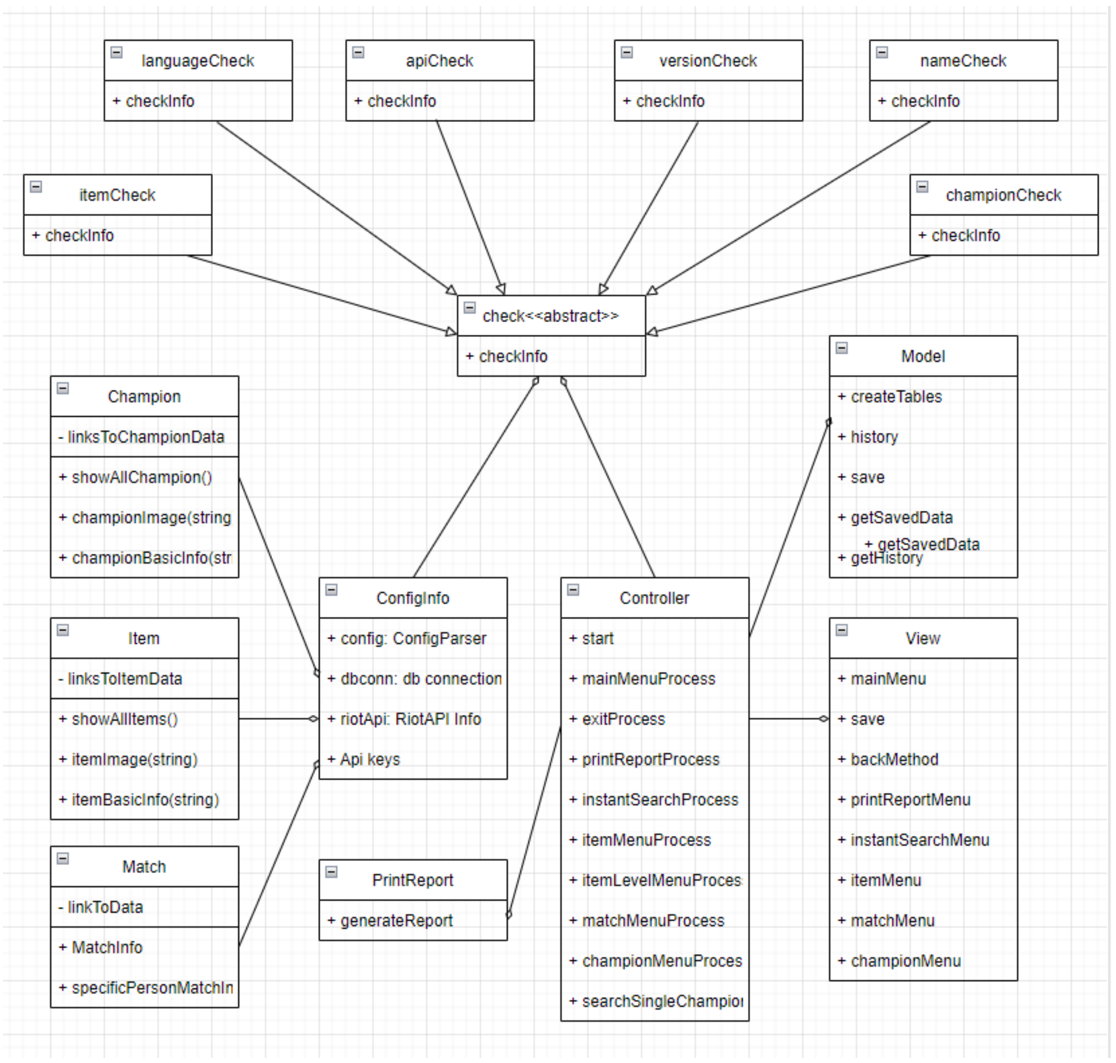
The Riot Games API is a web service and communicates with application through HTTP.



Development View

Concerns: Organization of software modules.
Stakeholders: Software developers, manager.
Modeling Techniques: UML Class Diagram

The ConfigInfo object is used to read the information from the configuration files. The chain-of-responsibility pattern is used to check the apikey; version and language is right for the Riot API to process at first.
The name, item, champion checks are used in the process of the program to make sure the user entry are correct.
The façade pattern is used to simplify the information that gets from the API and can be used easily by other functions in the program.
The MVC pattern is used to control the process of this project and give the interface to the user by the view. Connection to the database is through Model.

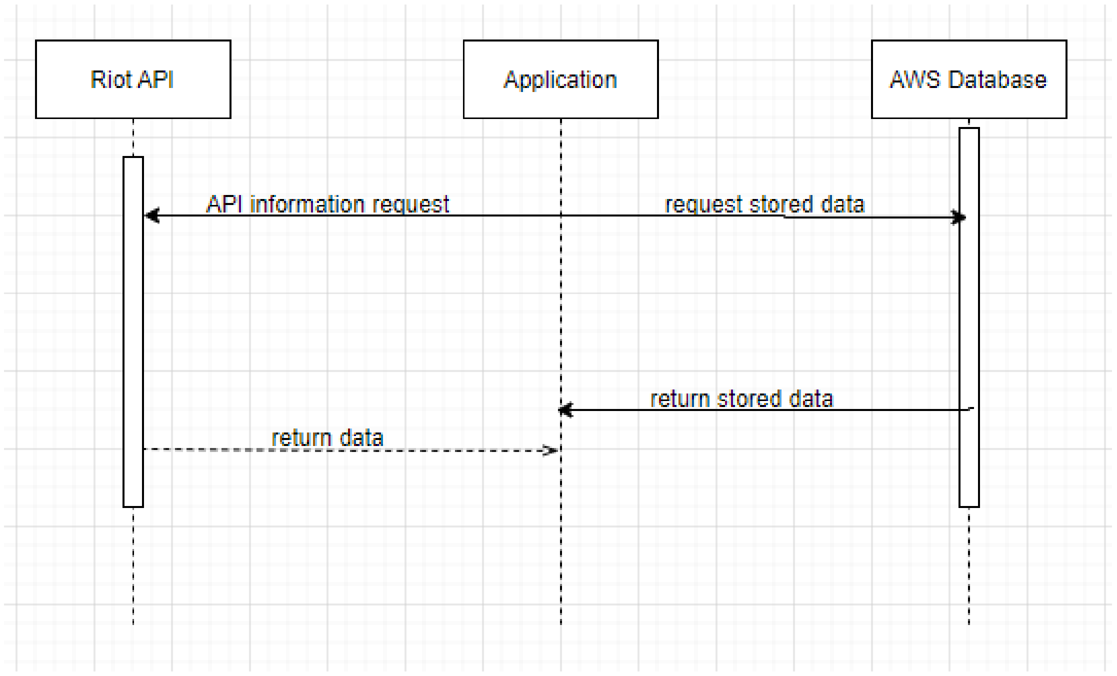


Design Pattern Summary

Pattern	Role	Classes
MVC	Control the project to run and communicate to the database and.	ProcessMVC
Chain-of-responsibility	Check the data entered is right for functions to work.	Check, Apicheck, Versioncheck, Languagecheck, Namecheck, Itemcheck, championcheck
Façade	Get information from API and easy apply to different functions	Champion, Item, Match

Process View

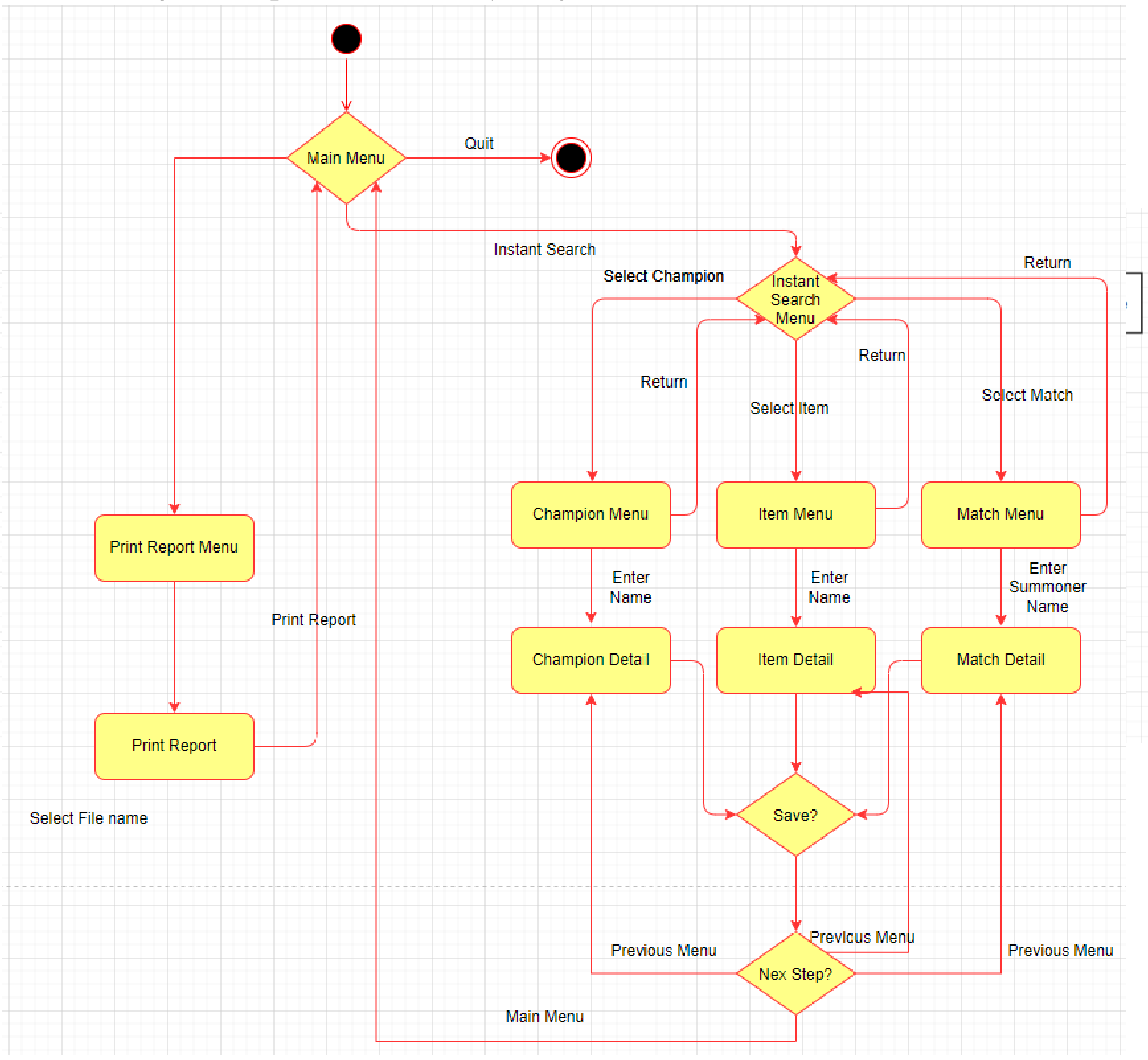
Concerns: Runtime communication
Stakeholders: Software architect
Modeling Techniques: UML Sequence Diagram
The important runtime communication is between application and the Riot API, Application and the AWS Database cloud database. The application sends requests to the Riot API and get the information back and stored to AWS Database cloud database. Application gets the stored data back from the AWS Database to create a selected report.



In the program, application will ask user to select what they want to search (champion or item or match). Then the application will call the functions in the champion/item/match classes to get the information from the Riot API. Riot API will give the raw data to the champion/item/match class to deal with the raw data to give out what user needs. User has another option to save the data they want and then combined them together to make a report. If user selects to make a combined report, the application will store the data user wanted to AWS Database and then at the end of the selection, the application will get the stored data back from the AWS Database and then generate a report for the user.

Logical View

Concerns: Functional requirements
Stakeholders: End user, software architect
Modeling Techniques: UML Activity Diagram



The use cases show a series of menus that allows users to pick between Instant Search Menu and Save Data Menu. In both menus, there are champion, item, and match menus for users to select for information to search about. After saving the data, user can select to print the combined report.

Database Schema

There are three tables in the AWS database.
CheckConnection Table is for checking the connection with the database is clear.
History Table is for saving the user interactions with the application.
SavedData Table is for saving the data that user chooses to save in the database and used for printing the report later.
All the data saved in the table will be erased every time the application start.

Table	Columns
checkConnection	Id, Data
History	Id, timeline, userAction
savedData	Id, dataInformation