



**UNIVERSITAT
JAUME·I**

AerosTracker.gg, a League of Legends Full-Stack Web Application

Javier Selma Rubio

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

May 25, 2022

Supervised by: Raül Montoliu Colás



To Javier and Ana

ACKNOWLEDGMENTS

First of all, I would like to thank to every member of my family who have helped me throughout the years of my Bachelor's Degree. My Father Javier and my Mother Ana, the best parents a person could wish for, who have helped me through the difficulties life has tried to put in our paths. My Grandmothers, Georgina and Consuelo, whose loving hands have taken care of me in difficult moments. My Godfather, Jesús, and my Godmother, Estefanía, for always keeping a space in their hearts for me. Finally, my beloved cousins, Daniel, Bárbara, Abril and Júlia, who, even as an only child, have made me feel what it is like to have a sibling.

Secondly, I would like to mention all those people who have accompanied me throughout my time as a university student. My friends from Alcoy and Castellón have made this journey easier for me in every single way they could. They have been for me in the dark and in the bright times, making me a better person. Thanks, Sam and George, you have made me realise what it is like to meet your soulmates.

Thirdly, I would like to thank my supervisor Raul Montoliu Colás, for accepting my project, even when it falls outside of his field of expertise.

Finally, I also would like to thank all the professors that I have met along my academic life. Some of them awoke in me the skeptical and scientific spirit that I possess. Thanks to Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

ABSTRACT

This document is a reflection of the research done to understand the operation of a web application consisting of a BackEnd and a FrontEnd, and reflecting this research work developing an application for a Final Degree Project whose purpose is retrieving information from an API and showcasing it in a web.

To sum up, the work has consisted of investigating how to make both parties communicate with each other, as well as knowing the ins and outs and delving into the information offered by the official API of the League of Legends game (provided by Riot Games). Finally, it has also been necessary to investigate how to show the information obtained to the end user, in an attractive, practical and concise way.

The project consisted of several different stages: first, it was necessary to investigate how web applications work. Later, after learning about the existence of the server side and the FrontEnd, it was necessary to learn first hand what information the Riot API provides, how it provides it, how to handle it and decide what information to use and what information not to use. For this, it has also been necessary to develop a database with PostgreSQL, making use of the knowledge acquired in the course to design a relational database with several entities. Once the server had been developed, it was necessary to develop a FrontEnd that would communicate with the server, obtain the data provided by the API created in the server, and display them to the user according to the information requested by the user.

The report will explain how every part of the project has been researched, designed, developed and deployed, starting from the logic of the server, going through the entities of the database and ending with the logic of the frontal part.

CONTENTS

Contents	iii
List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Work Motivation	1
1.2 Objectives	2
1.2.1 BackEnd	2
1.2.2 FrontEnd	2
1.3 Environment and Initial State	3
1.4 Expected Results	3
2 Planning and resources evaluation	4
2.1 Project Planning and Time Estimation	4
2.2 Resource Evaluation	7
2.3 Project Actual Time Cost	8
2.4 Tools	11
3 System Analysis and Design	13
3.1 OP.GG	13
3.2 FullStack Development	16
3.2.1 BackEnd	18
3.2.1.1 Jax	19
3.2.1.2 Database	20
3.2.2 FrontEnd	21
3.2.2.1 Landing Page	23
3.2.2.2 Summoner Page	24
4 Work Development	31
4.1 Work Development	31
4.1.1 BackEnd	32
4.1.2 FrontEnd	37

4.2	Project Technical Report Writing and Project Defense	39
5	Results	40
5.1	Results	40
5.2	Objectives Accomplished	43
5.3	Access to the Project	44
6	Conclusions and Future Work	45
6.1	Conclusions	45
6.2	Future work	45
	Bibliography	47
A	Riot API	50
A.1	Endpoints	50
B	JSON Parsing	53
C	Data Dragon	57

LIST OF FIGURES

2.1	Gantt chart of the Project (made with GanttProject)	9
3.1	Simple match shown in League of Legends client	14
3.2	Main screen of the web app OP.GG	15
3.3	Main screen when searching a Summoner in OP.GG	15
3.4	Data shown when clicking a match in OP.GG	16
3.5	Diagram showing the project's communication routes	17
3.6	Diagram showing the BackEnd workflow	18
3.7	Image showing the proper petition made in the Riot Games Developer Portal	19
3.8	Splash art of Jax Godfist	20
3.9	Scheme showing the real abstraction of the BackEnd	20
3.10	Relational scheme of the database used by the BackEnd	21
3.11	Diagram of SPA operation	22
3.12	Main screen of the web app OP.GG	23
3.13	Main screen of the web app U.GG	23
3.14	Main screen of the web app League of Graphs	24
3.15	Technical mock-up of AerosTracker.gg landing page	24
3.16	Matches screen of the web app OP.GG	25
3.17	Matches screen of the web app U.GG	25
3.18	Matches screen of the web app League of Graphs	26
3.19	Technical mock-up of AerosTracker.gg matches screen	26
3.20	Detailed match screen of the web app OP.GG	27
3.21	Detailed match screen of the web app U.GG	28
3.22	Detailed match screen of the web app League of Graphs	29
3.23	Technical mock-up of AerosTracker.gg detailed match screen	30
4.1	Console logs	33
4.2	Time diagram of non-pipelined vs. pipelined connection.	33
4.3	Several VMs working at the same time.	34
4.4	Docker standard environment.	35
4.5	Simple Wiki app developed in VJ1229 [1]	36
4.6	PgAdmin4 [2] dashboard image	37
4.7	AerosTracker.gg Logo	38

5.1	AerosTracker.gg main screen	42
5.2	AerosTracker.gg summoner overview	42
5.3	AerosTracker.gg detailed match view	43
A.1	Object returned by the Riot API when requesting a Summoner puuid	51

LIST OF TABLES

2.1	Documentation Phase's Estimation	5
2.2	Research Phase's Estimation	6
2.3	Development Phase's Estimation	6
2.4	Estimated duration of project's tasks summary	7
2.5	Economic costs of developing the project in an actual business environment	8
2.6	Documentation phase's actual duration	10
2.7	Research Phase's actual duration	10
2.8	Development Phase's actual duration	11
2.9	Real duration of project's phases summary	11

INTRODUCTION

Contents

1.1	Work Motivation	1
1.2	Objectives	2
	1.2.1 BackEnd	2
	1.2.2 FrontEnd	2
1.3	Environment and Initial State	3
1.4	Expected Results	3

This chapter will explain the different reasons why, in a career focused on Video Game Design and Development, I have chosen a web application as my final project, as well as explaining the objectives I aspire to when developing it and what minimum result I expect to obtain.

1.1 Work Motivation

After having had several subjects based on the development of a video game, either within engines such as Unity, or making use of libraries such as the one used in Web Development, and realizing that the development of applications makes use of skills that I have been acquiring throughout the degree, I thought off about the final project as a good opportunity to learn new languages, new skills and make use of skills already acquired that, put together, could serve a common purpose.

On the one hand, this project makes use of skills acquired in subjects such as Web

Development, Databases, Mobile Applications, and even the programming knowledge acquired in the basic subjects of the degree. These skills have made me know how to treat data, how to approach and face the development of a web application, and how to make use of such important elements in Information Technology (IT for now on) as APIs[3].

On the other hand, after having done extensive research on how to develop a complex web application that makes treatment of data from an API, in addition to having to make an extensive study of new frameworks and technologies that had not been used in the race, the project has also served as a gateway to a new world, the world of application development (specifically web applications).

That is why the main reason contemplated to do this kind of project was to continue learning in a field as vast as development and programming, specifically applications in this case. I'm not so much looking for perfection in the final application, but to learn along the way how a complex and complicated multi-part application works.

1.2 Objectives

The project, as previously stated, has two main parts, the server or BackEnd, and the user part or FrontEnd. The objectives to be achieved in each of them will be listed below.

1.2.1 BackEnd

The main objectives of the server-side or BackEnd are essentially the following:

- Retrieve information from the Riot API.
- Receive the information, select the desired items and discard the rest.
- Store the desired items in a relational database (PostgreSQL).
- Publish the items when required in an internal API to be consulted by the FrontEnd.

1.2.2 FrontEnd

The main objectives of the client-side or FrontEnd are essentially the following:

- Allow users to retrieve information on demand (introducing their League of Legends username).
- Retrieve the desired information from the BackEnd API.

- Receive the information and display it.
- Display the information in an effective and visual-appealing way.

1.3 Environment and Initial State

To put the reader in context, the project starts with a clear idea of what to do, but not how to do it, since this kind of application is well known in the League of Legends game universe. Any user who has played the game has used this kind of application or website at some point in their journey inside the game. Therefore, the interest of the project is to develop something widely known and find the way to do it efficiently and elegantly. In addition, it is vital to maintain the relationship between the project and the degree, thus being ideal the idea of doing the project on an API related to video games, which also I have worked on previously in other projects of the degree, so I know the information I can get from it.

1.4 Expected Results

As it has been mentioned before, the expected results are based on making a functional application that at least meets the minimum expectations of a user, which is to consult his games and their information on the web.

It is also of great interest to find a way to display the information making use of graphs, tables and visually appealing ways to allow full user interaction with the application.

This is why the problem faced when carrying out the project is to achieve a functional application both on the back and front end, in addition to design a user interface that allows, at a glance, to obtain the basic information present in a game.

PLANNING AND RESOURCES EVALUATION

Contents

2.1	Project Planning and Time Estimation	4
2.2	Resource Evaluation	7
2.3	Project Actual Time Cost	8
2.4	Tools	11

This chapter will explain the time and resource planning part of the project. Development work must at least follow a well-defined pattern by stages that allows the developer to advance step by step in the different functionalities that are desired in the application.

For this reason, first, a division into tasks will be made to develop the project correctly, and then an estimate of hours will be made for each of them to have an idea of the final theoretical time dedication of the project. Then, the real dedication of hours to each of the tasks will be compared to know if the planning, estimation, and division were done correctly

2.1 Project Planning and Time Estimation

For the planning of the project, it has been decided to separate it into three distinct stages. The first stage is the writing and preparation of the documentation (including the report and its defense). The second, the research of the various technologies necessary for the development of the application, and the third and last, the development itself.

The hours assigned are an arbitrary estimation taking into account the ideal target of hours per ECTS of the subject, therefore 300 hours will be the ideal total estimate of the project.

Firstly, the documentation stage (Table 2.1) includes the necessary hours for the drafting of the various documents required for the project, in addition to the preparation of the defense of the report.

Table 2.1: Documentation Phase's Estimation

Documentation Phase	
Tasks	Estimated Duration (in hours)
Technical Proposal	5
System Analysis and Design	5
Technical Report	40
Project Defense	10
Total	60

Secondly, in the stage of researching the necessary technologies (Table 2.2), it has been taken into account the need to learn about two new technologies: the one used for the server –Adonis framework [4]– and the one used for the client –Vue.JS framework [5]– in addition to having to dig into the depth of the information provided by the Riot API [6]. This stage is mainly focused on searching for information and acquiring the necessary knowledge to proceed with the complete development of the application.

Table 2.2: Research Phase's Estimation

Research Phase	
Tasks	Estimated Duration (in hours)
Learn about Adonis.JS	10
Learn about Vue.JS	10
Dive into Riot API	5
Research and Comprehend how real OP.GG works	5
Research optional JavaScript libraries to use	10
Total	40

Thirdly, Table 2.3 shows the tasks related to the development of the application, the design of the database and the user interface, and the development of the various services required.

Table 2.3: Development Phase's Estimation

Development Phase	
Tasks	Estimated Duration (in hours)
Design BackEnd	20
Design Database	15
Develop BackEnd	60
Design FrontEnd	20
Design UI	15
Develop FrontEnd	60
Test and Deploy	10
Total	200

Finally, this Table 2.4 serves as a summary to show the total estimate of the project

divided into its three stages.

Table 2.4: Estimated duration of project's tasks summary

Development Phase	
Tasks	Estimated Duration (in hours)
Documentation	60
Research	40
Development	200
Total	300

2.2 Resource Evaluation

When evaluating the resources required for the theoretical development of the project, it is necessary to take into account both the cost of the equipment –hardware and software– and the cost of the labor required to carry out both the design and the development.

By using free libraries and frameworks, a private but free API and free tools to design both the database and the front end, only the cost of maintaining the application on a server 24 hours a day and the cost of an average programmer's salary plus the computer he/she will use will have to be taken into account. Estimated costs are exposed in Table 2.5

Table 2.5: Economic costs of developing the project in an actual business environment

Cost Evaluation Summary			
Resource	Cost (in €)	Time spent (in hours)	Total cost (in €)
Hardware costs			
Computer	2000.00		2000.00
Internet Access	40.00		40.00
Server costs			
BackEnd hosting	500.00 / year	8760 (1 year)	500.00
FrontEnd hosting	200.00 / year	8760 (1 year)	200.00
Software license costs			
Windows 10 Pro	129.90		129.90
GitHub Enterprise	12.00 / month	8760 (12 months)	144
Human costs			
Full-Stack Developer	30.00 / hour	300	9000.00
Total			12013.90

2.3 Project Actual Time Cost

The objective of this section is to compare the times that were estimated at the beginning of the preparation of the technical report with the actual times that have been used in each of the stages and parts of each of them.

It can be seen how the times have varied slightly, since throughout the development of the project various unforeseen events have arisen that have made it necessary to invest more hours than planned.

On the one hand, one of the main problems was having to use two frameworks that were unknown at the beginning of the development. Having to learn from scratch two technologies that have not been used in the race has been a real challenge.

On the other hand, when it came to preparing the technical documentation for the project, there were several obstacles related to the complexity of using \LaTeX and the Overleaf editor.

The Figure 2.1 shows the actual timeline of the project, using a Gantt chart.

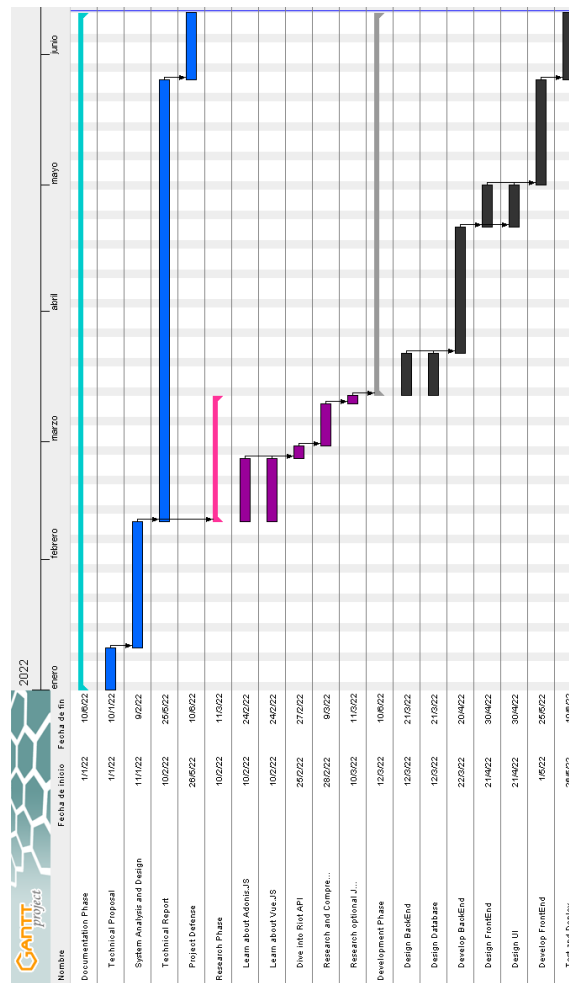


Figure 2.1: Gantt chart of the Project (made with GanttProject)

Tables 2.6 , 2.7 and 2.8 show the actual project duration in hours. There have been several modifications in the times dedicated to each of the sub-stages to ensure the final quality of the project.

Table 2.6: Documentation phase's actual duration

Documentation Phase			
Tasks	Estimated Duration (in hours)	Real Duration (in hours)	(in hours)
Technical Proposal	5	5	
System Analysis and Design	5	2	
Technical Report	40	70	
Project Defense	10	6	
Total	60	83	

Table 2.7: Research Phase's actual duration

Research Phase			
Tasks	Estimated Duration (in hours)	Real Duration (in hours)	(in hours)
Learn about Adonis.JS	10	15	
Learn about Vue.JS	10	15	
Dive into Riot API	5	2	
Research and Comprehend how real OP.GG works	5	5	
Research optional JavaScript libraries to use	10	8	
Total	40	45	

Table 2.8: Development Phase's actual duration

Development Phase			
Tasks	Estimated Duration (in hours)	Real Duration (in hours)	(in hours)
Design BackEnd	20	15	
Design Database	15	10	
Develop BackEnd	60	90	
Design FrontEnd	20	15	
Design UI	15	10	
Develop FrontEnd	60	80	
Test and Deploy	10	15	
Total	200	235	

Finally, to expose clearly the difference between the estimated hours and the actual ones, the different phases are showcased in Table 2.9.

Table 2.9: Real duration of project's phases summary

Development Phase		
Tasks	Estimated Duration (in hours)	Real Duration (in hours)
Documentation	60	83
Research	40	45
Development	200	235
Total	300	363

2.4 Tools

This section will list the necessary tools that will be used throughout the development of the project.

- **Documents**
 - Overleaf: tool used to elaborate scientific papers in \LaTeX .
 - LaTeX Table Generator: tool used to generate \LaTeX tables.
 - GanttProject: free open app to create Gantt charts.
 - Grammarly: free online app that corrects grammatical errors.
 - DeepL: free online translator.
- **Programming**
 - Visual Studio Code: Free, Open-Source IDE
 - Adonis.JS: Node.JS based framework, working with TypeScript, used to publish APIs
 - Vue.JS: JavaScript reactive framework
- **Version Control**
 - GitHub: a Git repository hosting service.

SYSTEM ANALYSIS AND DESIGN

Contents

3.1	OP.GG	13
3.2	FullStack Development	16
3.2.1	BackEnd	18
3.2.1.1	Jax	19
3.2.1.2	Database	20
3.2.2	FrontEnd	21
3.2.2.1	Landing Page	23
3.2.2.2	Summoner Page	24

In the previous chapters, it was stated that documentation, research and preparation would be necessary before embarking on the development of the project.

This chapter is going to explain what exactly an OP.GG [7] page is and why it is so widely used by all League of Legends [8] players.

In addition, it will also explain the decision of each of the frameworks for each of the parts of the application, their basic functioning, how they communicate with each other and the different design choices made for the project.

3.1 OP.GG

League of Legends is a game within the MOBA [9] genre. Each game pits 10 players, divided in two teams of 5 people, who fight against each other, and lasts between 30

and 45 minutes on average. In this kind of genre, basic statistics such as kills, assists or deaths are extremely important.

In addition, it is very important to note that each champion or character in the game has 4 skills, and can buy up to a maximum of 6 different items to choose from hundreds.

After knowing this, it is easy to assume that after each game, each player in the game will want to consult what kind of items he, his teammates and his enemies have bought, the basic statistics related to damage, gold or other aspects and the final scores of each of the players involved.

With this in mind, any user might think that the developer itself, Riot Games [10], would provide an option in the game client itself to be able to consult this kind of statistics, and the answer is ambiguous: yes and no.

Yes, because some statistics of the game are indeed shown, as well as a timeline of the game that tries to represent important events in the game such as the obtaining of multiple casualties, the purchase of key items or the death of neutral targets.

No, because this, as you might guess, is not enough for players who want to improve and better understand every existing aspect of the game. Riot Games' API [6] gives very extensive information about each of the matches, and the developer decided, when League of Legends became the most played game in the world [11], that they were going to give little information on the match history of each of the players in order to save on server maintenance costs, which are extremely high. Figure 3.1 gives an example of how little information the match history gives to a normal user.



Figure 3.1: Simple match shown in League of Legends client

This, combined with the large community that the game has, made people with sufficient knowledge and technical skills to get to work with the development of web applications that would allow consulting this information provided by the Riot Games API. This is where the first, the most popular and the most powerful game query page arose: OP.GG [7].

When delving into what kind of statistics can be obtained from this web application, it must be first known firsthand what a user needs to request information from the web server. And the answer is simple and concise, you only need to know your username and the server you play on, as shown in Figure 3.2.

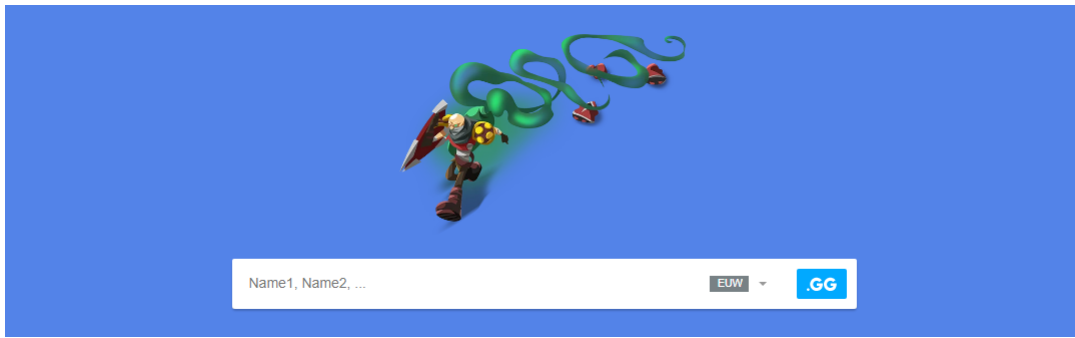


Figure 3.2: Main screen of the web app OP.GG

Once the request has been passed to the server, the BackEnd gets all the information that the page wants to show to the user, and gives it to the user showing all the public games that the user has played, as well as indicating other data such as his rank in the ranked queues, his statistics with various characters in the game, the winning and losing streak in his last games, and many other sections. The overall information obtained after searching your summoner name can be seen in Figure 3.3.

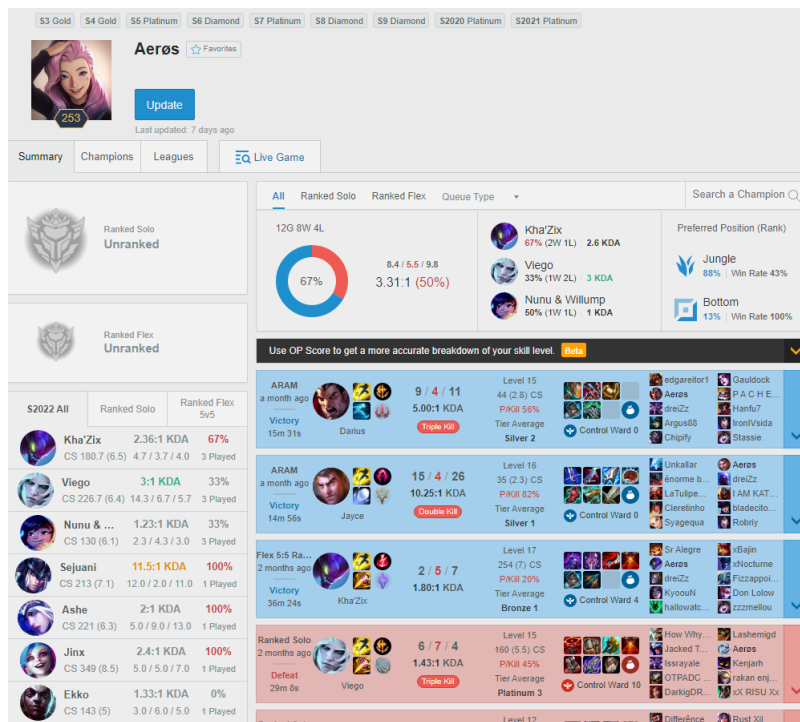


Figure 3.3: Main screen when searching a Summoner in OP.GG

This looks a bit like the interface provided by the developer itself in the official client

of the game, but the interesting part comes when the user wants to know the specific data of one of the games shown in the list of games. Once the user clicks on one of these games, an interface is displayed showing interesting data about each of the participants in the game. Items, levels, statistics —KDA [12]— and neutral and general objectives such as turrets. The match history shown in Figure 3.3 has several matches. Once the first one gets clicked, it shows the aforementioned data, as shown in Figure 3.4.

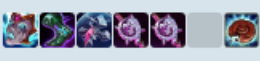

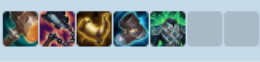

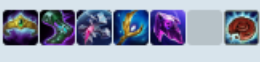
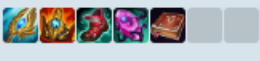
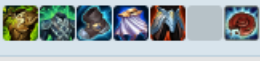


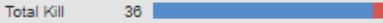



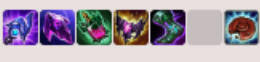

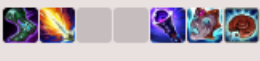


Overview		Team Analysis		Build			etc			
Victory (Blue Team)		Tier	KDA	Damage	Wards	CS	Item			
 15	edgareitor1	Level 196	3.86:1 6/7/21 (75%)	24,428	0 / 0	49 3.2/m				
 15	Aerøs	Level 253	5.00:1 9/4/11 (56%)	11,409	0 / 0	44 2.8/m				
 15	dreiZz	Silver 4	3.71:1 16/7/10 (72%)	17,622	0 / 0	20 1.3/m				
 15	Argus88	Level 292	5.80:1 0/5/29 (81%)	3,901	0 / 0	1 0.1/m				
 15	Chipify	Platinum 3	5.00:1 5/4/15 (56%)	11,123	0 / 0	30 1.9/m				
 0	 0	 4	Total Kill	36			27	 0	 0	 2
			Total Gold	55427			52531			
Defeat (Red Team)		Tier	KDA	Damage	Wards	CS	Item			
 14	Gauldock	Level 255	2.33:1 5/9/16 (78%)	14,081	0 / 0	25 1.6/m				
 14	PACHENCO	Level 535	3.33:1 4/6/16 (74%)	23,184	0 / 0	38 2.4/m				
 14	Hanfu7	Level 372	2.63:1 8/8/13 (78%)	25,908	0 / 0	44 2.8/m				
 14	IronIVsida	Bronze 2	2.50:1 10/6/5 (56%)	18,075	0 / 0	63 4.1/m				
 14	Stassie	Level 556	3.43:1 0/7/24 (89%)	2,511	0 / 0	4 0.3/m				

Figure 3.4: Data shown when clicking a match in OP.GG

3.2 FullStack Development

Web applications can have different complexities, depending on the purposes and functionalities that they must fulfill. It is the job of a developer to know what kind of application he wants to develop, what he needs to do it, and to know what parts to develop and how to make them interact with each other.

When deciding what kind of application and complexity the one developed in the project should have, it was necessary to investigate how the application used as an exam-

ple worked in order to develop our own: OP.GG. After finding out that the application made use of a FullStack [13] development, it was necessary to know what function and how each of the parts that make up this kind of development did. The back end, or server, and the front end, or client, are clearly delimited and fulfill a very specific function when interacting with the various external parts, such as the user or the Riot API[6].

In OP.GG, the BackEnd performs the function of checking if it has the user's data in the database, and if these do not exist or are not up to date, it also takes care of requesting them from the Riot Games API [6]. Once it has obtained this data, and after having saved it in the database if necessary, it is passed in the form of a JSON [14] file to the FrontEnd. This data is chosen and displayed in various sections on the FrontEnd, after decoding the file obtained from the server.

The same operating structure will be followed in the application developed in this project. The Adonis.JS [4] framework, a Node.JS-based framework, has been chosen for the BackEnd, in addition to communicating with a PostgreSQL-based database [15], while the Vue.JS [5] framework, which allows developing single-page applications [16], has been chosen for the front-end, in conjunction with the basic languages used to develop a web page, HTML5 [17], JavaScript [18] and CSS [19]. The following Figure 3.5 shows a diagram that explains the ways in which the parts of the project interact with.

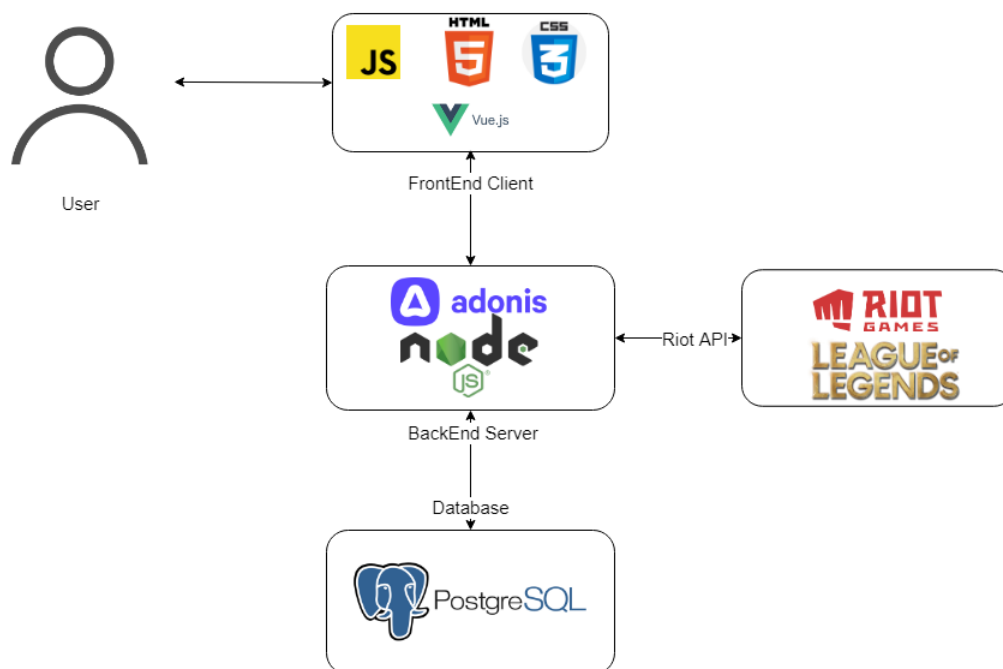


Figure 3.5: Diagram showing the project's communication routes

3.2.1 BackEnd

When talking about the BackEnd or server, it is first necessary to know what an API [3] is in order to know how the data will be obtained from Riot Games [10], and how the data will be exposed to the FrontEnd or client so that the user can enjoy the information they demand.

An API [3] or application programming interface is a set of definitions and protocols used to design a port for integrating application software together. It can be described as a listener that is asleep, waiting for a client that wants to demand information from it to access one of its endpoints [20] or routes.

For this reason, besides having to design the database that will store the data obtained from the Riot API [6], it is also necessary to define the routes that can be accessed by the FrontEnd. The following diagram, shown in Figure 3.6 , will show how the BackEnd will act when requesting, storing and displaying the information used by the project application.

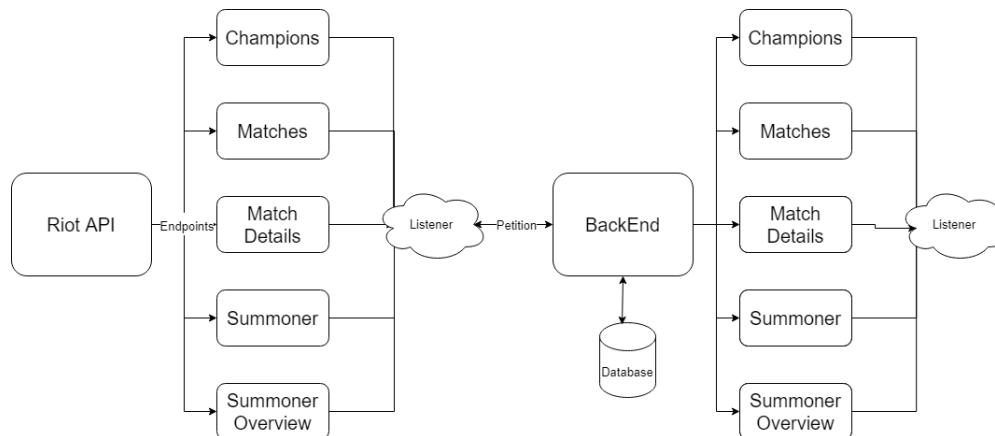


Figure 3.6: Diagram showing the BackEnd workflow

After looking at Figure 3.6 , it is easily conceived why to split the application into two parts, when the front end could communicate directly with the Riot API [6], since it shares the same endpoints and the same listener structure as the server. The problem is that, by making use of a proprietary API, the application is limited by the developer itself to be able to make a limit of requests per minute to the API, in addition to having its limitations in each of the endpoints that can be even more restrictive than the general ones.

Before starting to develop the project, it has been necessary to make an application request to Riot Games [10] on their developer portal in order to have a more lax limit. Thanks to having made use of the API in previous projects of other subjects of the

degree, the process was quick and the response was positive from the developer. The Figure 3.7 shows the request made on the development website.

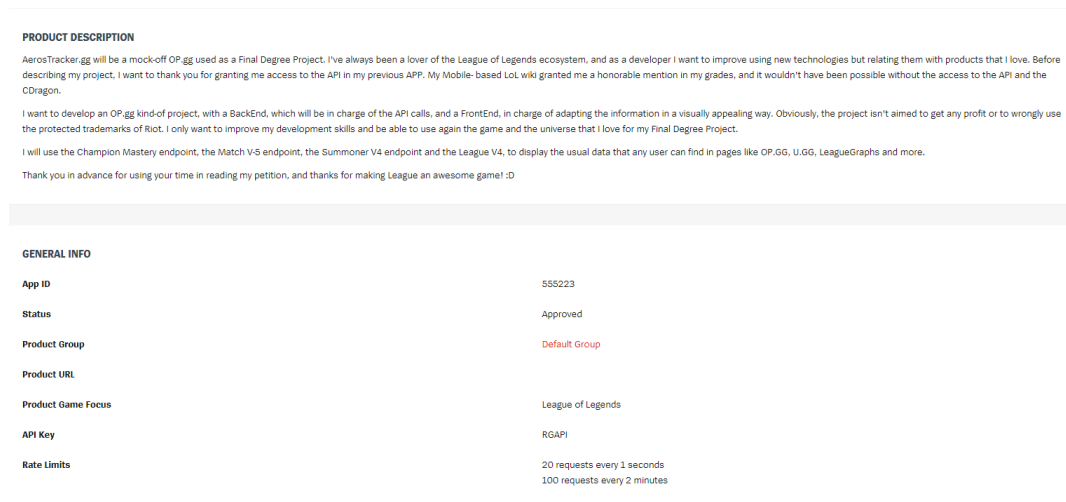


Figure 3.7: Image showing the proper petition made in the Riot Games Developer Portal

When the request is approved, an API Key is granted, which has been hidden for obvious reasons in Figure 3.7, as it is the cornerstone that allows access to the developer's server. Here it is also possible to see how a rate limit is granted to each approved application in the development portal. When making requests to the API, the limit indicated here cannot be exceeded at any time, nor the limit indicated in each of the endpoints to be accessed.

When making requests to the Riot API [6], a higher level of abstraction will be used than the one indicated in the diagram in the Figure 3.6. This diagram shows the schematic operation of the application and its communication with the Riot API [6], but in practice, there will be an intermediary in charge of centralizing the requests at a single point, and later differentiating which endpoint and exactly what information is to be obtained.

3.2.1.1 Jax

In honor of the favorite character of the author of this project, the intermediary that will act between the BackEnd and Riot's API [6] will be named Jax [21], one of the most famous characters in the League of Legends [8] universe. Figure 3.8 shows a splash art of the character whose name gets referenced in the intermediary.



Figure 3.8: Splash art of Jax Godfist

Source: https://aminoapps.com/c/league-of-legends-en-espanol/page/blog/teamodls-guia-sobre-jax/xpE5_qV4h2uwkRBKa4JVwE2ba7jXReZDqD4

The following schematic, shown in the Figure 3.9, shows the design chosen for the abstraction between the API and the BackEnd.

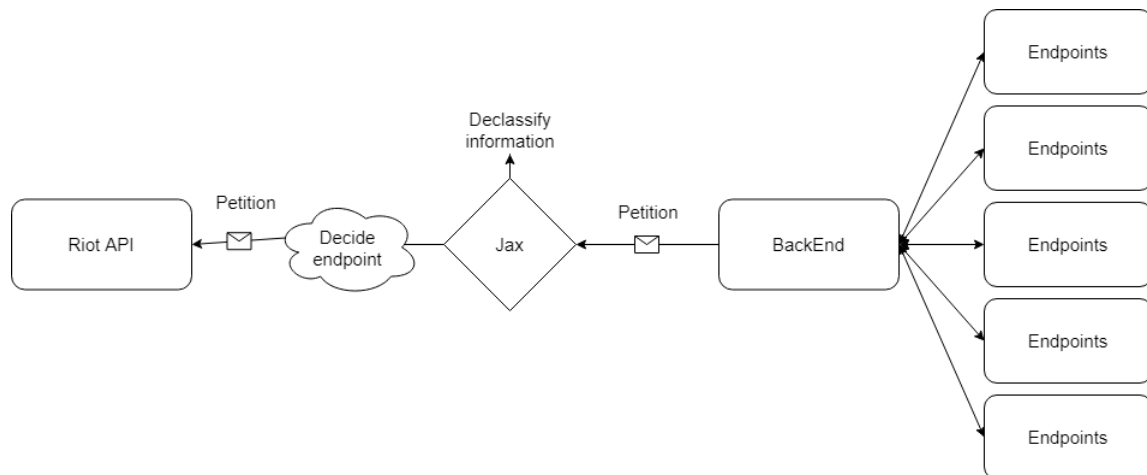


Figure 3.9: Scheme showing the real abstraction of the BackEnd

3.2.1.2 Database

When designing the database to be used for the server, it has been taken into account the endpoints previously mentioned in Figure 3.6. From them, and taking into account

what kind of information is going to be needed in the FrontEnd, a relational model in PostgreSQL [15] has been used, shown in Figure 3.10, designing the following models to host the information.

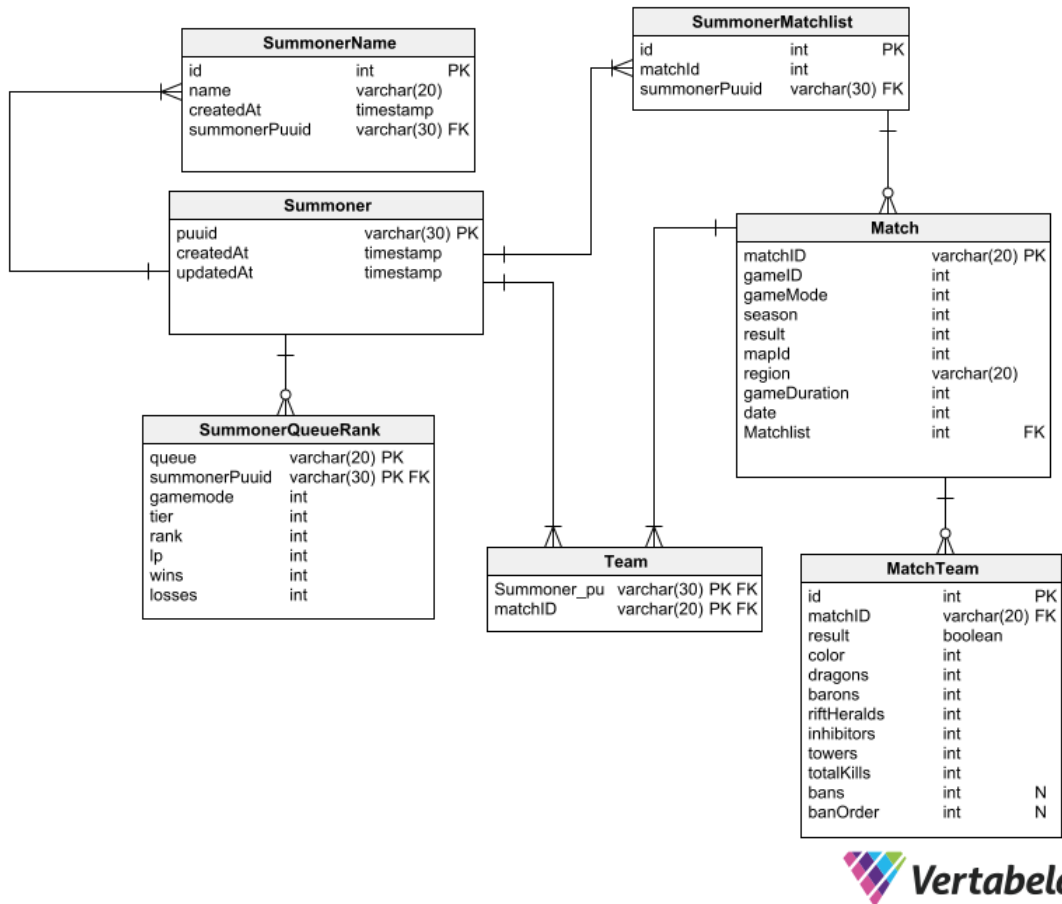


Figure 3.10: Relational scheme of the database used by the BackEnd

Source: Used Vertabelo Data Modeler, <https://vertabelo.com/>, to make the schematic

3.2.2 FrontEnd

The client or FrontEnd is an essential part of the application. Both when communicating with the BackEnd and displaying the information obtained from it, it plays a fundamental role in the basic operation of any request from the user.

As it has been observed in the screenshots made to the OP.GG [7] page, this kind of applications have two basic parts, the main screen where the server and the user name are introduced, and the display screen where all the information related to the user of the game is shown.

In modern front-end applications, a specific kind of frameworks are used that allow the page to function in Single-Page Application [16] mode. This means that, when clicking on a page element, a new web page will not be loaded, with all that this implies in terms of server load, waiting time and updating and use of the various data by the page. This is shown in Figure 3.11

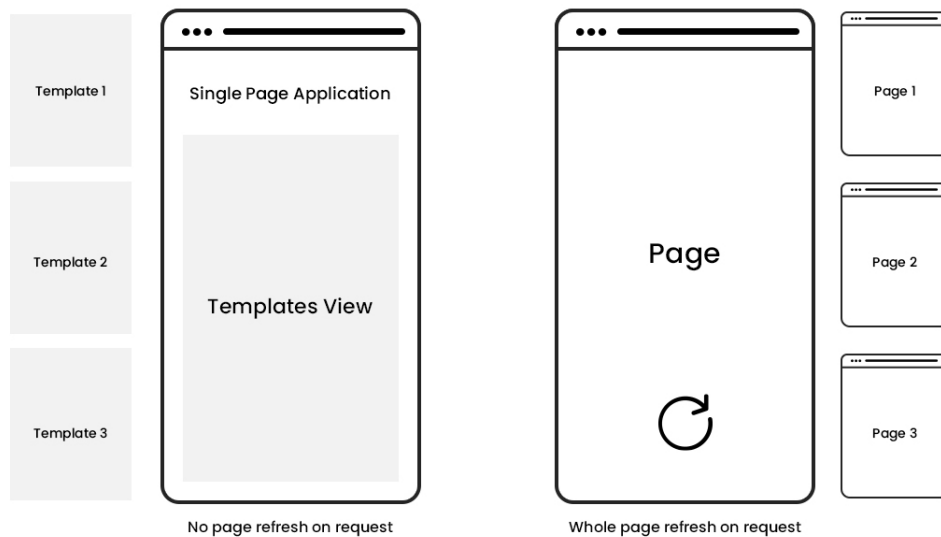


Figure 3.11: Diagram of SPA operation

Source: [What is a SPA?](#)

Seeing how this kind of applications work, it is easy to imagine which parts of the developed project will depend on a template structure. The main page where the user will enter his summoner name and his server will be an independent page, which once it sends the request to the server, will change to the user's profile page, where everything will be handled by components [22], the basic unit of the Vue.JS [5] framework.

Just as the operation of the server part of the application has been described by means of several schemes and diagrams, it is also pertinent to look for several references in webs of the same style, in order to be able to work a mock-up that has the maximum resemblance to the final product that is desired to obtain.

For this reason, both pages that are going to form the client are going to be referenced making use of several images and diagrams of other pages that perform the same functions. It is not necessary to reinvent the wheel to make a new application that can work better than the previous ones, simply by putting together ideas that are good but that together would work better, you can get an excellent final result.

In order to have enough references, and to compare the performance of different pages with each other, thus being able to choose the desired elements of each one of them, three applications with wide support that are well known by the community and that perform thousands of searches per day will be used: OP.GG [7], League of Graphs [23] and U.GG [24].

3.2.2.1 Landing Page

This part of the application is the simplest to prototype, since it simply has to fulfill one function, that of allowing the user to enter his player name and the server where he plays, thus allowing the application to request the necessary information from Riot's API [6].

In the following Figures —Figure 3.12, Figure 3.13 and Figure 3.14—, it is possible to see how, except for small visual details that allow you to see more or less information about the current situation of the game, news that may have arisen during the week about game updates or the release of new champions, players only have to enter their user in a search bar that will allow them to move to the next screen.

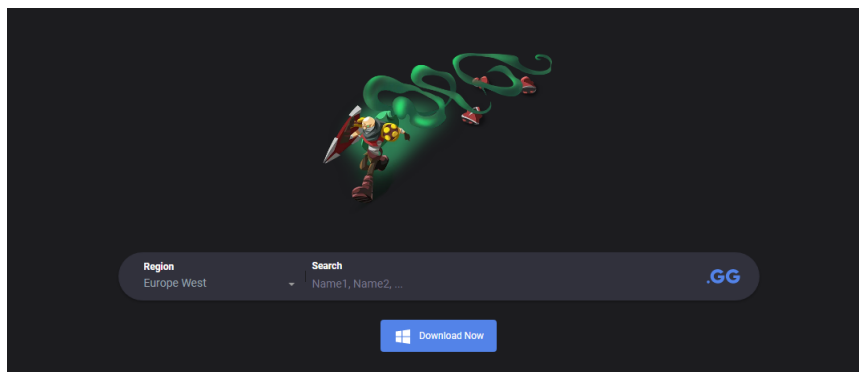


Figure 3.12: Main screen of the web app OP.GG

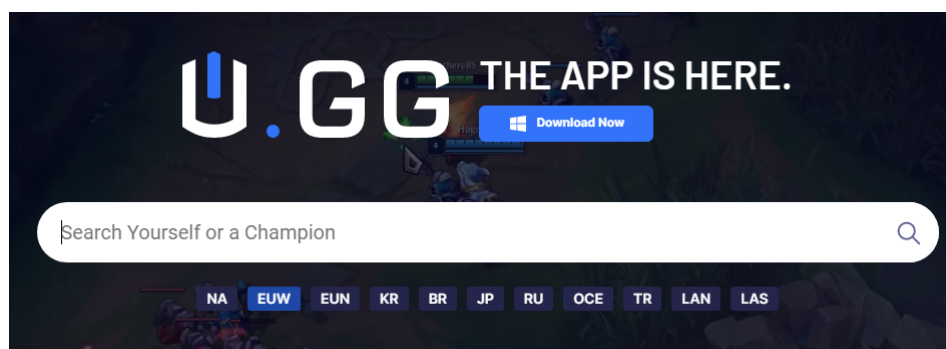


Figure 3.13: Main screen of the web app U.GG

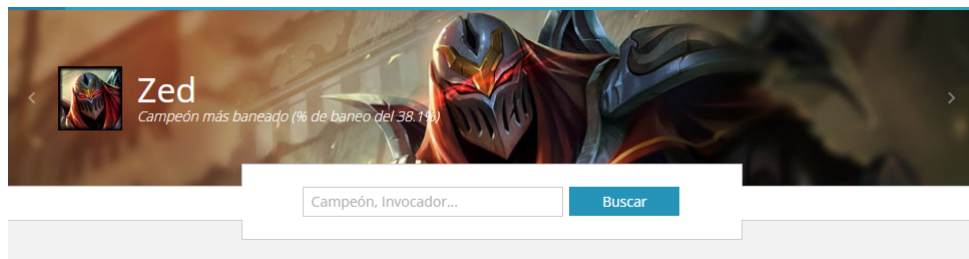


Figure 3.14: Main screen of the web app League of Graphs

After observing that practically all applications with this kind of purpose share the same type of home page, it is easy to imagine and design how the client's home page of the project is going to look like. The Figure 3.15 shows a mock-up of the technical design of the page.

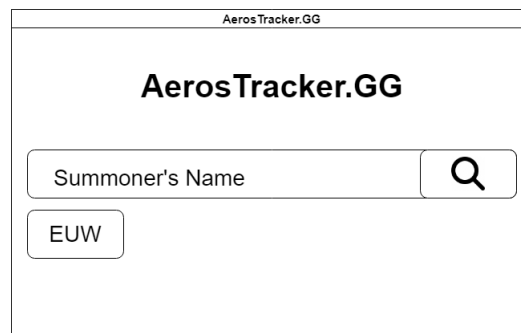


Figure 3.15: Technical mock-up of AerosTracker.gg landing page

3.2.2.2 Summoner Page

Once again, taking into account the features present in the previous web applications, the second and most important page will be designed from the elements that can be found in them.

It is common to have a brief description of the user's basic statistics, such as his own Summoner name, the level he has, his icon, statistics related to ranked games such as his own rank, the use of certain champions and the games played with friends or with other users repeatedly.

Finally, the most important part includes a list of the last 20 games played by the player, being able to see more on demand, thus saving resources and requests to the Riot server. Once the user clicks on any of these games, he will see detailed information about that game in question, seeing the information arranged in different ways depending on what the web developer considers most important. The interface of the webs can be seen in Figure 3.16, Figure 3.17 and Figure 3.18.



Figure 3.16: Matches screen of the web app OP.GG

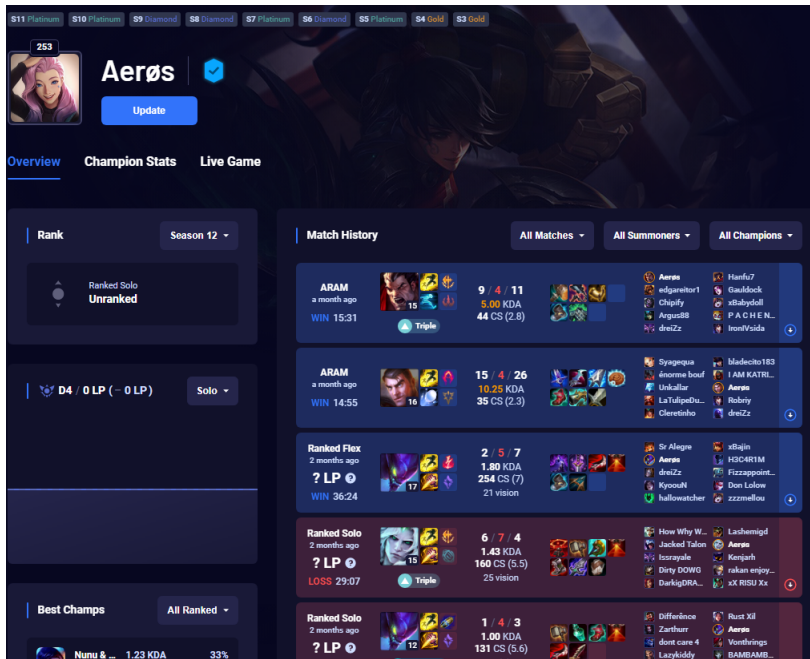


Figure 3.17: Matches screen of the web app U.GG

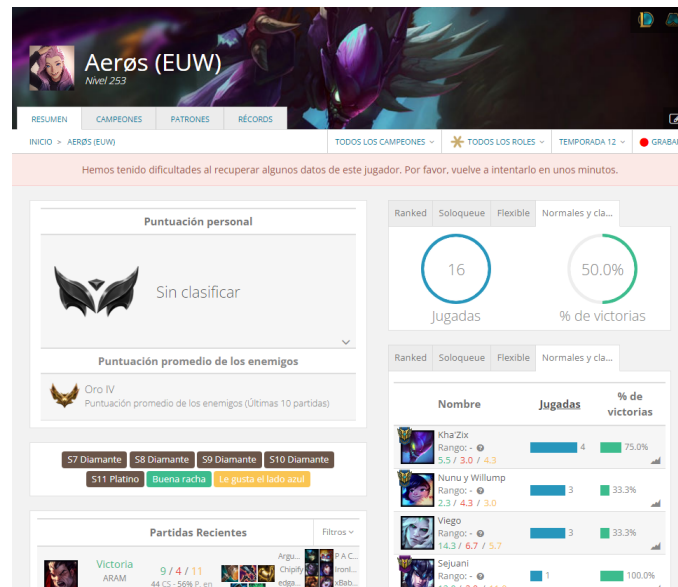


Figure 3.18: Matches screen of the web app League of Graphs

As it happened in the landing page, the designs are practically the same, therefore, at the time of making the technical mock-up of the client's project, it will follow a similar form to the one seen in the images. It will make use of various components to be able to fragment the code into smaller pieces, and thus be able to reuse and integrate efficiently the various parts of the FrontEnd. This can be seen in Figure 3.19.

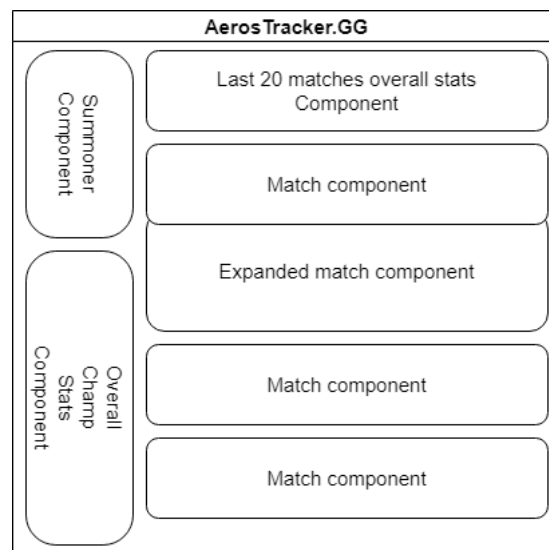


Figure 3.19: Technical mock-up of AerosTracker.gg matches screen

Finally, an important part of the application is to be able to obtain the details of each of the games. As shown in Figure 3.19, clicking on a game displays an extended view of the various details present in the game. Therefore, as in the other cases, the webs will be taken into account in order to obtain a satisfactory final design that is in tune with the data that will be stored in the database. Detailed match views can be seen in Figure 3.20, Figure 3.21 and Figure 3.22.



Figure 3.20: Detailed match screen of the web app OP.GG



Figure 3.21: Detailed match screen of the web app U.GG

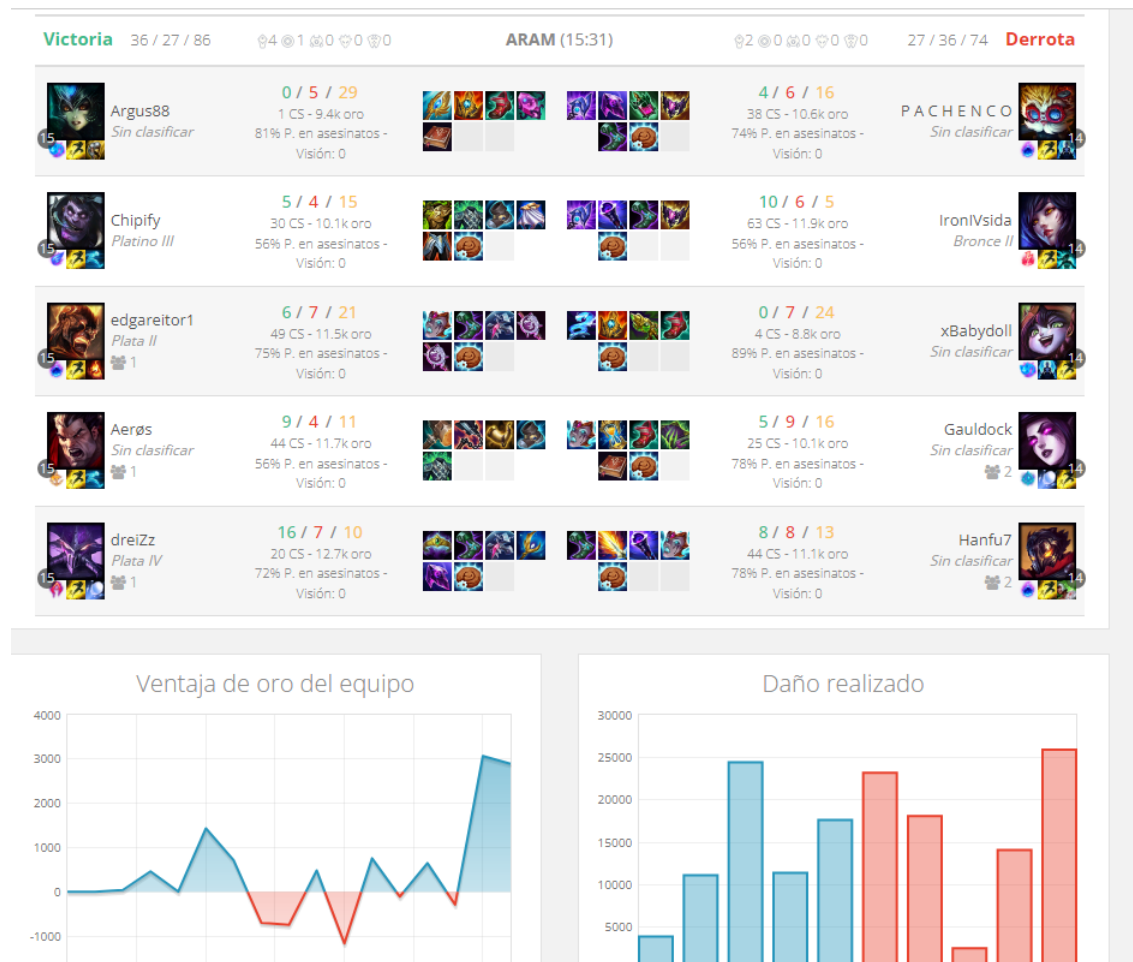


Figure 3.22: Detailed match screen of the web app League of Graphs

Taking into account the user interfaces shown in the previous figures, the most detailed and the one that provides the most information is the one on the League of Graphs website. Therefore, the technical mock-up elaborated for the detailed part of the games will be closely related to the aforementioned website. This can be seen in Figure 3.23.

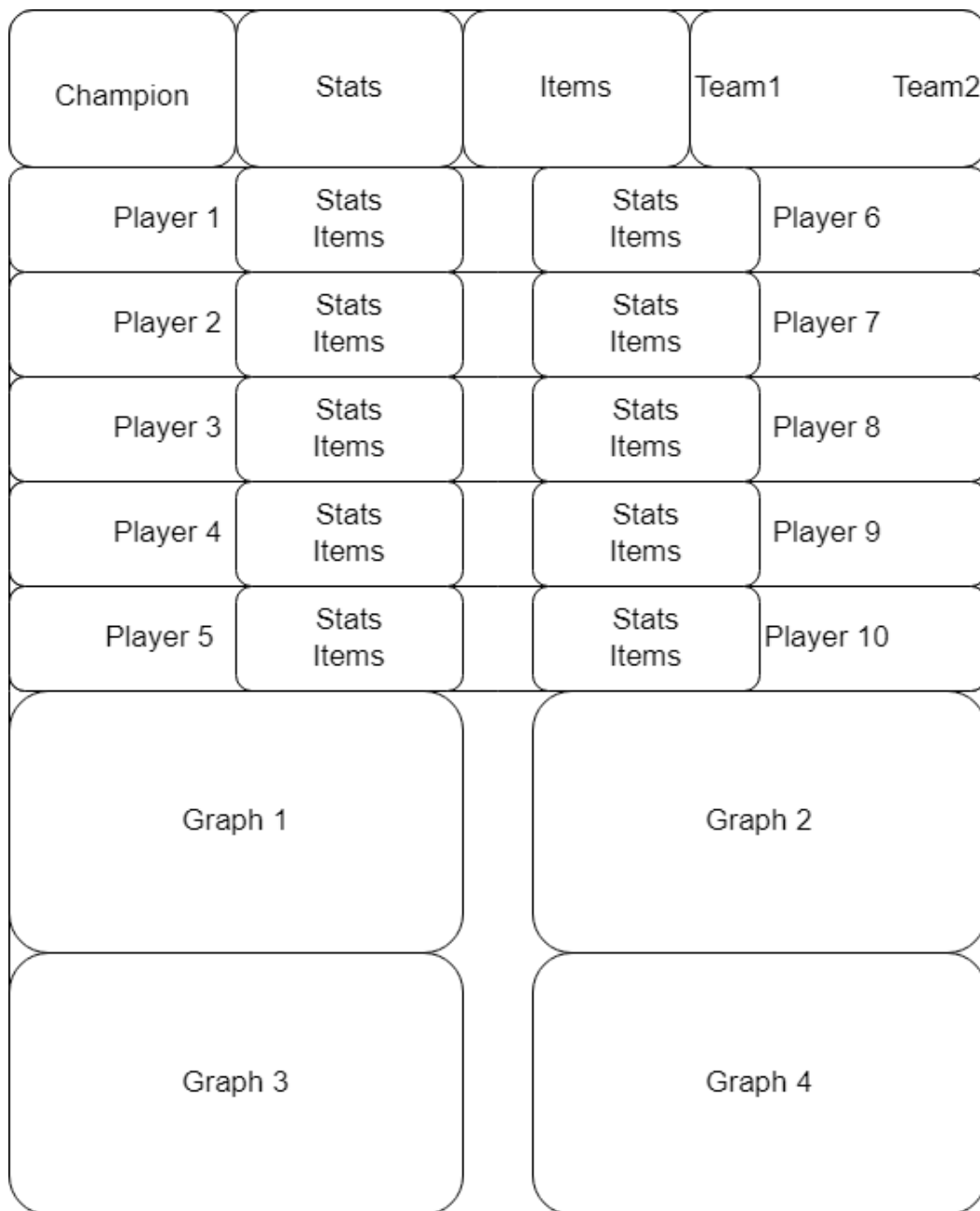


Figure 3.23: Technical mock-up of AerosTracker.gg detailed match screen

WORK DEVELOPMENT

Contents

4.1	Work Development	31
4.1.1	BackEnd	32
4.1.2	FrontEnd	37
4.2	Project Technical Report Writing and Project Defense	39

In this chapter the obstacles encountered during the development of the project will be discussed, as well as the possible doubts that arose throughout the development, the solution found to them, and several important technical aspects that, although not explained in this chapter but in the appendices attached to the report, deserve to be mentioned since they were an obstacle when creating a functional final product.

4.1 Work Development

To be consistent with the chronology followed with the project, it is possible to consult the Gantt Chart Figure 2.1, since the development of the project has followed the steps one after the other for a simple reason.

The FrontEnd is dependent on the BackEnd, and at the same time, the BackEnd is dependent on the database and the Riot API queries. Once the essential parts of the project were developed, parts like the FrontEnd simply required a simple development that made use of all the previously developed elements.

4.1.1 BackEnd

While developing the BackEnd, it was necessary to face several problems to ensure the correct functioning of an interface that would allow quick querying of data to the Riot API.

One of the first problems that appeared during the development of the project was to fix the long API call times. Whenever it was necessary to get games from the API, often the response time was too long because of the initial approach in the logic of the calls to the Riot interface, due to the existence of data of 10 different players, each with his champion, stats, rank, and a long etcetera of data.

One of the solutions was to find a helper to maximize the optimization of each API call and its subsequent storage in the application's existing database.

This solution was found by making use of an open-source technology –Redis [25]– in charge of optimizing to the maximum the existing calls to both APIs and databases.

At the same time as having developed a database based on PostgreSQL [15], making use of knowledge already acquired in the degree, it is also put in common with the use of Redis, an application that allows the storage in memory of certain information until it is needed, thus avoiding unnecessary queries to the database and the Riot Games API.

By using this technology, it is possible to improve the performance of the application [26] when querying certain data. In a scenario in which a new Summoner is searched, it could be possible that, in one of its matches, an already existing Summoner that has already been previously searched appears. The application will instantly detect, thanks to having stored the game momentarily in memory, that it is not necessary to make this call neither to the API nor to the database.

The application, once it obtains for the first time the data of a Summoner, a match, or another element that can be repeated between calls, will make sure not to repeat information stored in the database, thus allowing a quick search in the database using Redis, optimizing the application by several orders of magnitude.

Finally, to know if there has been any problem when obtaining any of the data present in the relational model, a series of console logs have been implemented throughout the process of API calls, to be able to identify the time it took to obtain the data. This is why if there has been an error getting a single item, such as the players' rank, or the image of one of the specific items of one of the players, you can quickly find out which part has failed, why, and what to do to fix it.

Figure 4.1: Console logs

```

playing: 333.818ms
ranked: 234.27ms
RecentActivity: 44.477ms
BASIC_REQUEST: 2.074s
getMatches: 219.051ms
GLOBAL: 4.35ms
GAMEMODE: 3.016ms
ROLE: 2.733ms
CHAMPION: 3.144ms
CHAMPION-CLASS: 2.338ms
MATES: 12.732ms
RECENT_ACTIVITY: 5.326ms
STATS: 40.335ms
OVERVIEW_REQUEST: 266.877ms
--> CALL TO RIOT MATCHLIST
RiotMatchList: 369.876ms
matchList: 381.679ms
championsRequest: 7.556ms
MatchDetails: 38.584ms
MatchDetails: 30.315ms
MatchDetails: 12.691ms
Ranks: 9.000s
MatchDetails: 13.851ms

```

It is observable in Figure 4.1 how a basic call to retrieve Summoner information is made, including several calls inside it as champions used by the Summoner, its rank, and so on. Later, the match list of the Summoner is retrieved from the API, as well as the details of 4 of the matches present in the list.

Another use of Redis within the project is to allow pipelining [27]. This allows several calls that are different from each other to be executed at the same time. A visual example of how pipelining works can be seen in Figure 4.2. It can be seen how, in an HTTP communication between a client and a server, it is possible to optimize the response and data retrieval time by making simultaneous calls.

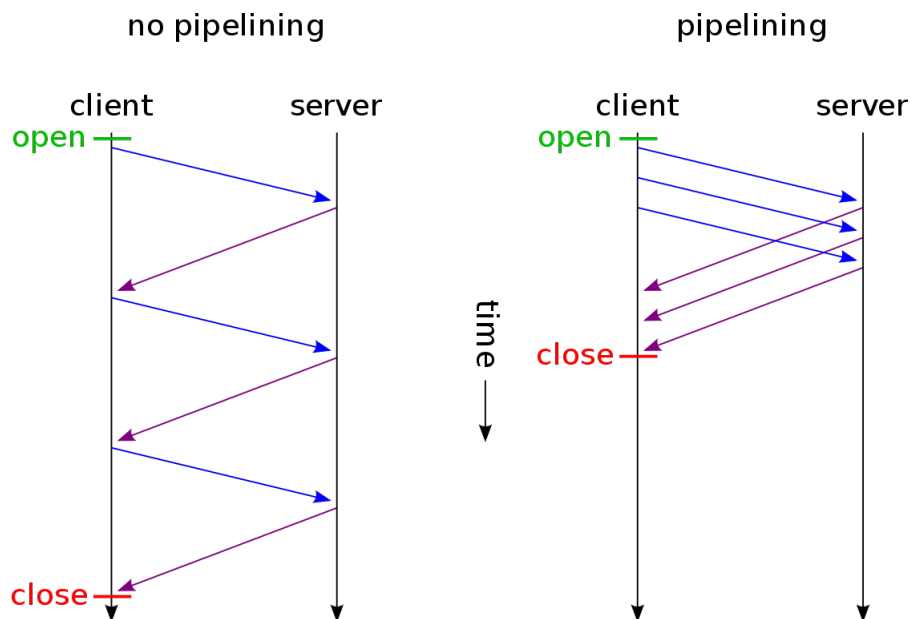


Figure 4.2: Time diagram of non-pipelined vs. pipelined connection.

Source: https://en.wikipedia.org/wiki/HTTP_pipelining

Within the same game, it is possible to make simultaneous calls to obtain items, statistics, Summoner ranks, Summoner names, and so on. Making these calls simulta-

neously will not cause any problems between them, therefore making use of the logic present within Redis saves extraordinary time that could not have been saved without the application.

In the following code snippet, in each of the calls to the API or to the database, the Redis [25] application is used, making use of a function called "task queue" [28]. A task will be added to the pipelining present in the application logic, to be able to make different calls in the logical form of "try-catch" [29] to obtain errors from each one of them.

```
1 try {  
2     const response = await got(url)  
3  
4     await Redis.set(url, response.body, 'EX', this.cacheTime)  
5     return JSON.parse(response.body)  
6 }
```

One of the curiosities of Redis[25] is that, in order to allow its great optimization, it is not developed for Windows environments. Therefore, it is necessary to make use of another tool called Docker [30]. This tool allows running images of programs such as Redis in optimal environments for its execution. This is similar to what was seen in subjects such as **VJ1225 - Operating Systems** [31], in which a Virtual Machine [32] was used to carry out the development in other systems such as Unix [33], Linux [34] or Ubuntu [35]. It is observable in Figure 4.3

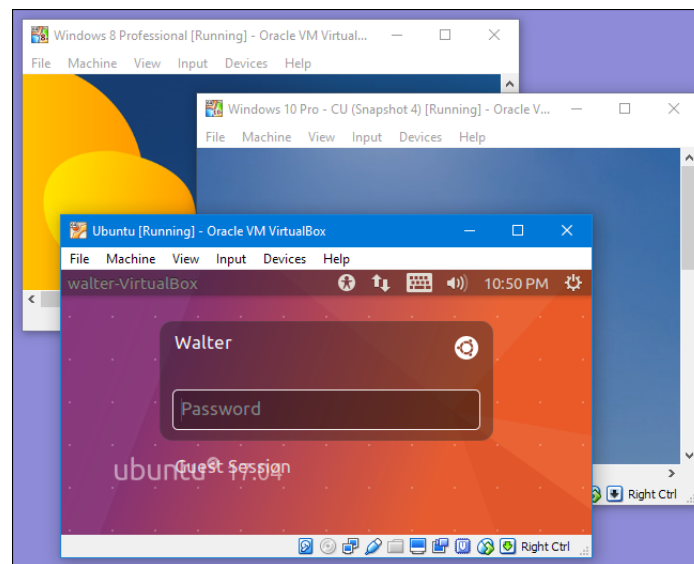


Figure 4.3: Several VMs working at the same time.

Source: <https://howpedia.net/es/beginner-geek-how-to-create-and-use-virtual-machines-como-crear-y-usar-maquinas-virtuales>

Once a Docker [30] environment has been set up, it simply uses an official Redis [25] image to launch a local server that allows the application to make queries through Redis [25]. A simple process thanks to Docker that allows to avoid emulating a complete Linux-based [34] operating system by simply running an image in real-time in a secure environment for its execution. It is observable in Figure 4.4 how a Docker environment looks like.

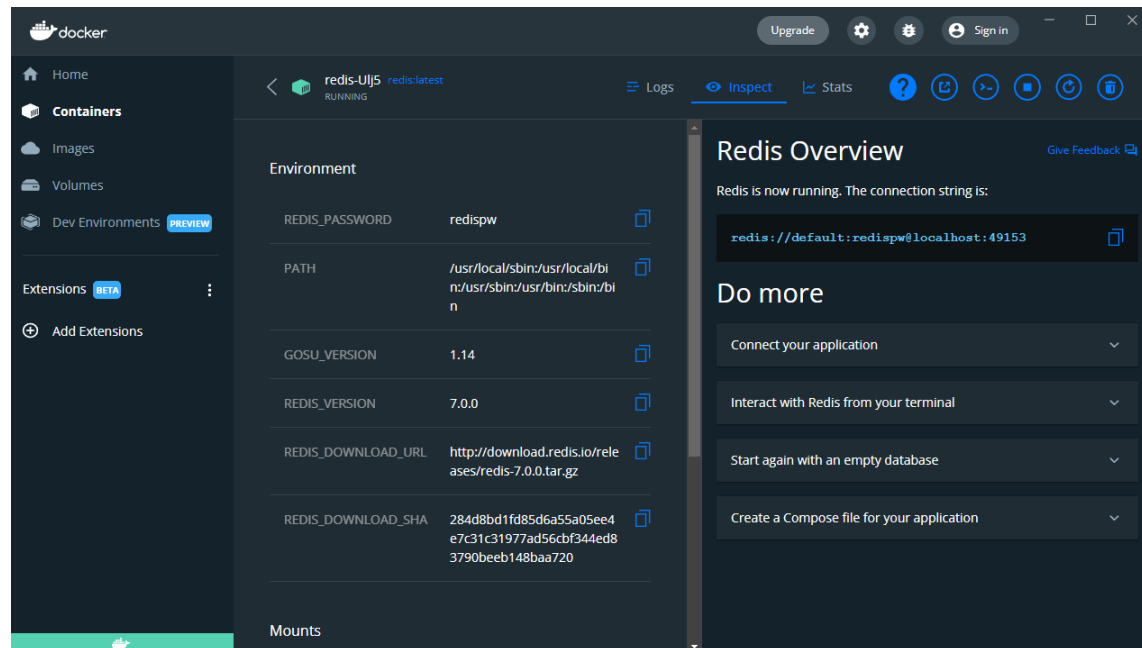


Figure 4.4: Docker standard environment.

Following the Gantt chart, the next thing to do within the project development was to dive in and learn the existence of the various endpoints within the Riot API. As was also mentioned in other chapters of the technical report, the API has been used by me in other projects in the degree, so I was well aware of its virtues and limitations.

Previously, in the subject **VJ1229 - Mobile App Development** [1], a final project app was developed using the API. The project consisted in creating a small wiki containing information about all the existing characters in the League of Legends universe. The project obtained a final grade of 10, therefore the correct use of Riot's API was demonstrated. It is possible to observe in Figure 4.5 that a simple app, using a JSON parser like this project, displays a list of champions in a similar way that the web application will display the several matches a summoner has played.

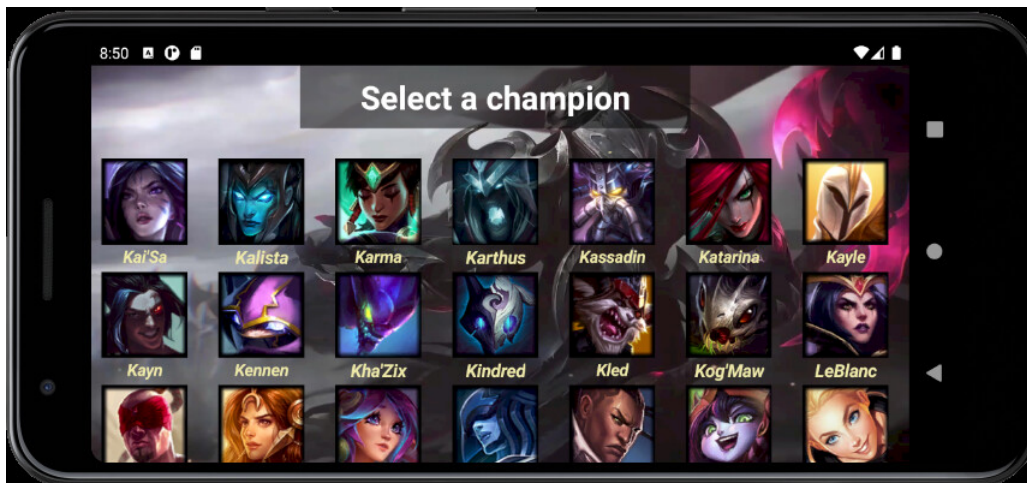


Figure 4.5: Simple Wiki app developed in VJ1229 [1]

Even so, the Riot API hasn't been used in a project of such a great magnitude as the present one, therefore it was necessary an extra study that has given elegant solutions as the one explained in the Appendix A to handle all the requests on the part of the server.

Furthermore, even having had to develop several parsers in these other projects, each project is a world of its own and requires specific information to be collected for the final purpose of the applications. Therefore, for a project that required to obtain information from so many endpoints, and to show everything related to the summoners, their games, their ranks and other data, it has been necessary to make new parsers. This solution is also explained in the Appendix B.

Another of the pitfalls encountered in the development of the project was, at the same time as the development and design of the parsers, to design and develop the database in PostgreSQL [15]. Although it is the language used in the degree's subject **VJ1220 - Databases** [36], it was taken in the second year of the degree, and it has been necessary to refresh the knowledge about relational models, in order to make a functional design that would allow storing the information obtained from the JSON [14] of the API, as shown in the Figure 3.10.

Thanks to the existence of a program that was previously studied in the subject **VJ1220 - Databases** [36], whose name is **pgAdmin4** [2], it has also been possible to create a local PostgreSQL [15] server, and at the same time to monitor the database and the schema created to run the application. The relational schema seen in the Figure 3.10 has been created on the server thanks to the so-called migrations present in Adonis.JS [4]. With TypeScript [37] code, it has not been necessary to redo the work twice, creating consequently the tables in the database schema. The tables previously mentioned can

be observed in Figure 4.6.

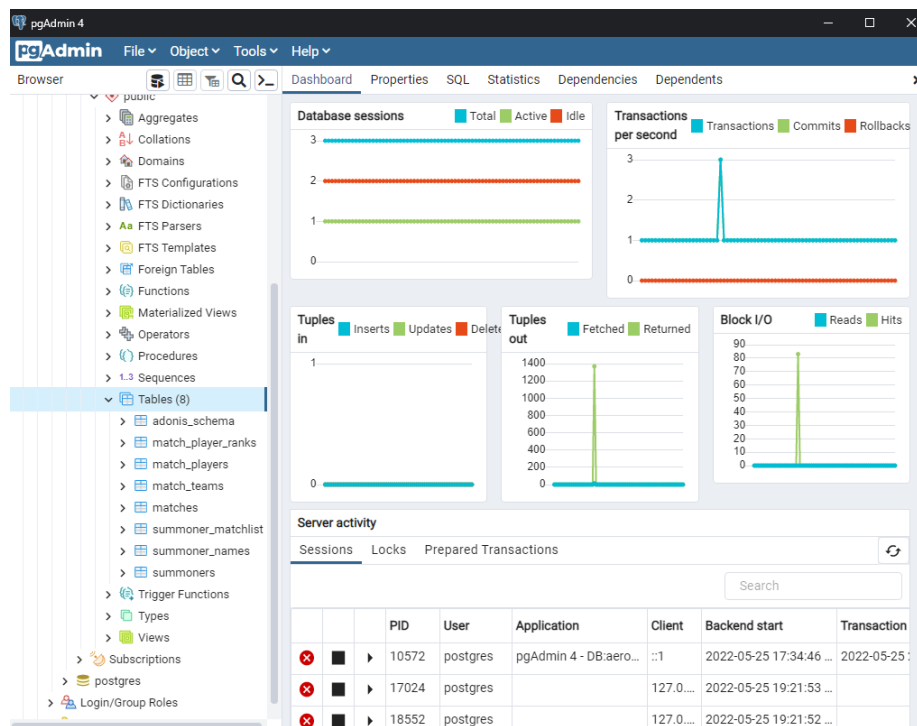


Figure 4.6: PgAdmin4 [2] dashboard image

4.1.2 FrontEnd

Learning about Single-Page Applications [16] or APIS was not difficult, thanks to knowledge acquired in previous projects during the degree. The documentation of Adonis.JS [4] and Vue.JS [5] is clear and concise, and there are thousands of examples on the web that allow a great reusability and a quick implementation of fast methods such as the POST [38] of information on the BackEnd side, or the design of components [22] on the FrontEnd side.

It is also important to mention the internship at the company CloudAppi.SL [39], which has greatly streamlined the learning process both in the development of the Back-End, and especially the development of the FrontEnd, since the internship consisted of the complete refactoring of a web application based on Vue.JS.

Talking about one of the first obstacles encountered in the development of the FrontEnd, was to obtain the necessary images for each of the elements that exist in the game. Riot Games found an elegant solution to locate all these unique elements in a concentrated and accessible place that would allow all developers, both in-house and independent, to access the existing assets in each of the different patches of the game.

The solution that Riot Games considered is creating Data Dragon.

Data Dragon is Riot Game's way of centralizing League of Legends game data and assets, including champions, items, runes, summoner spells, and profile icons. All of which can be used by third-party developers. It is possible to download a compressed tarball(.tgz) [40] for each patch which will contain all assets for that patch. The precise operation of this technology will be explained in a third Appendix C.

Once all the necessary assets have been obtained, the information has been requested to the BackEnd and it has been successfully obtained, it only remains to display it in the places designed for each of the types of data the database returns.

The information will be divided into a basic section where the Summoner's data will be shown, a list containing each of the Summoner's games, and an area where the statistics of the last champions and roles used can be observed.

When the design of each of the parts was approached, it was believed that the design of the user interface was going to be a problem, since there was no knowledge of interface design. But surprisingly, thanks to the great similarity of all the applications, the interface could be designed without hindrance, as demonstrated in the technical mock-ups shown in Figure 3.15, Figure 3.19 and Figure 3.23.

Finally, to finish designing the FrontEnd, it was necessary to design a logo that would serve as an identifier on the front page. The logo was designed using Illustrator [41], a vectorial design program that allows creating professional logos. It can be seen in Figure 4.7.



Figure 4.7: AerosTracker.gg Logo

4.2 Project Technical Report Writing and Project Defense

A very important part of the project has been the writing of the technical report. It is therefore important to underline the time spent in learning a new language for writing scientific papers: L^AT_EX.

In addition, in order to achieve a correct and formal English, besides making use of my own knowledge, which I consider to have been sufficient to defend myself when it came to writing the report, I have made use of DeepL [42] to ensure that my translations and my writing were as formal as possible.

In addition, in order to ensure this grammatical correctness, Grammarly [43] has been used to ensure the cohesion and coherence of the translations.

Finally, it was also necessary to transform the information shown in the technical report into a more visual and concise form in order to be able to defend the project in the final presentation of the final degree project.

CHAPTER **5**

RESULTS

Contents

5.1	Results	40
5.2	Objectives Accomplished	43
5.3	Access to the Project	44

5.1 Results

As a final part of the development of the project, it is necessary to analyze the results that have been obtained, taking into account what has been learned throughout the development, as well as the knowledge acquired when solving the various complications that have arisen along the way.

So, the question to answer is, what have I achieved with this project?

- I have learned how a Full-Stack web application is developed, including the Back-End and FrontEnd.
- I have been able to design a full web application, taking into account every important aspect that has allowed the correct operation
- I have learned to contact a big developer such as Riot Games, designing a mock-up project to get access to their private API and get granted a private API Key that allowed me to access several endpoints.

- I have learned to compare several working products to elaborate a final product that exceeded my own expectations
- I have been able to refresh the knowledge I have acquired throughout the degree, such as database design, web development, and JSON parsing.
- I have learned to use a Node.JS framework, Adonis.JS, and its workflow.
- I have learned to use a JavaScript FrontEnd framework, Vue.JS, and its workflow.
- I have parsed JSON files and I have successfully extracted the desired data from them

Continuing with the results of the project, it is difficult to show visually the final result of the BackEnd. Therefore, a summary of the various figures showing the interior and technical design of the BackEnd will be made, since the diagrams, guidelines, and technologies described in each of the chapters dedicated to the design of the application server have been followed one by one.

- Technical Design:
 - Communication routes in Figure 3.5
 - BackEnd workflow in Figure 3.6
 - Jax intermediary in Section 3.2.1.1
 - Abstraction of the BackEnd in Figure 3.9
 - Database relational scheme in Figure 3.10
- Tools Developed:
 - Console logs in Figure 4.1
 - Pipelined HTTP petitions in Figure 4.2
 - Redis petition code
 - Docker standard environment in Figure 4.4
 - Tables created in application's database in Figure 4.6
- Technologies Developed:
 - Riot API information retrieving in Appendix A
 - JSON Parsing in Appendix B

When talking about the FrontEnd, it is easy to observe the final result of the whole development. Following the structure of the different pages described in chapter 3 – OP.gg [7], U.gg [24] and League of Graphs [23] –, where the main pages, the general page of a specific Summoner, and the detailed items were exposed, an specific order will be followed in which the Figures 5.1, 5.2 and 5.3 will show the final result of the project in each of these aspects.

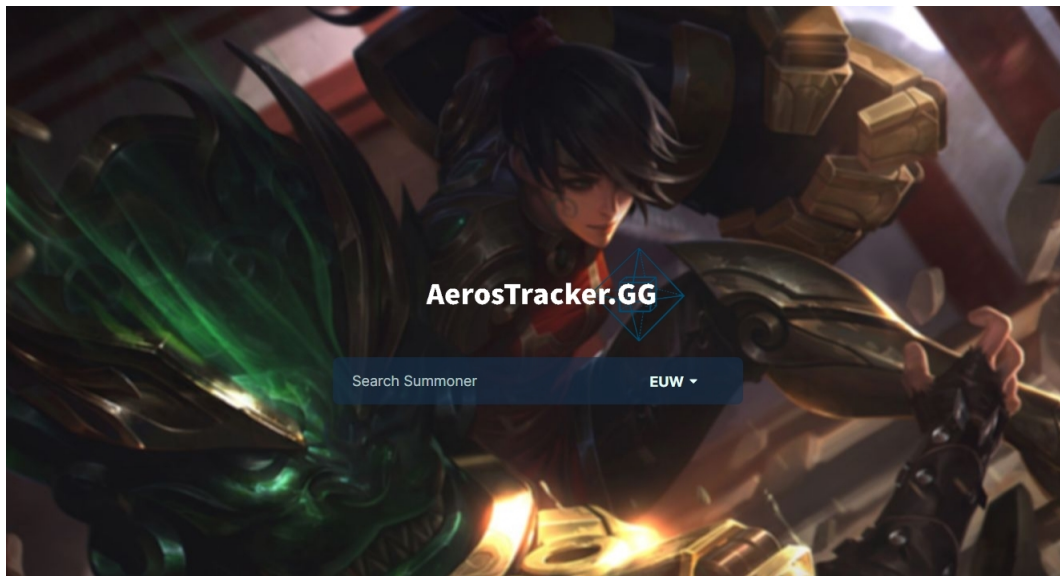


Figure 5.1: AerosTracker.gg main screen



Figure 5.2: AerosTracker.gg summoner overview

Ally Team		K	D	A	Cs	Vs	Gold	Dmg champ	Dmg obj	Dmg taken	KP
Jotichah Lee Sin	9	7	11	161	19	12.8k	18.9k	2.8k	26.4k	71.4%	
Aeros Nocturne	7	7	12	196	12	12.5k	11.5k	20.5k	30k	67.9%	
Monino Dard... Zed	8	7	6	154	16	11.3k	19k	4.1k	18.1k	50.0%	
csdarker93 Kalista	3	9	9	157	23	10.6k	12.7k	0.3k	22k	42.9%	
TheSaMMu Blitzcrank	1	11	6	31	6	6.4k	6.7k	0k	24.7k	25.0%	
Total	28/41/44	53.6k	68.8k	2 Towers	1 Dragons	0 Barons					
Enemy Team		K	D	A	Cs	Vs	Gold	Dmg champ	Dmg obj	Dmg taken	KP
zero ten zero Akali	6	8	11	145	15	11.8k	19.9k	5.5k	24.9k	41.5%	
Ehgenevi Master Yi	20	10	4	166	23	17.6k	34.8k	60.6k	33.7k	58.5%	
vodos Ahi	4	6	6	153	14	10.3k	13.5k	7.7k	17.4k	24.4%	
BloodReinā Rakan	6	0	17	112	11	11.5k	12.3k	9.6k	10.7k	56.1%	
Linch Seraphine	5	4	14	27	75	10.9k	10.4k	9.1k	10.3k	46.3%	
Total	9/28/52	62.2k	90.8k	9 Towers	4 Dragons	1 Barons					

Figure 5.3: AerosTracker.gg detailed match view

5.2 Objectives Accomplished

When beginning the redaction of this technical report, the main objectives of the project were the following:

- BackEnd: the main objectives of the server-side or BackEnd were essentially the following:
 - Retrieve information from the Riot API. ✓
 - Receive the information, select the desired items and discard the rest. ✓
 - Store the desired items in a relational database (PostgreSQL). ✓
 - Publish the items when required in an internal API to be consulted by the FrontEnd. ✓

- FrontEnd: the main objectives of the client-side or FrontEnd were essentially the following:
 - Allow users to retrieve information on demand (introducing their League of Legends username). ✓
 - Retrieve the desired information from the BackEnd API. ✓
 - Receive the information and display it. ✓
 - Display the information in an effective and visual-appealing way. ✓

All the objectives that were proposed in the beginning, have been achieved, so it can be said that the project has been a success.

5.3 Access to the Project

The source code of the project can be found in a public repository on GitHub, where both parts of the project can be found and used.

<https://github.com/JavierSelma/AerosTracker.gg>

CONCLUSIONS AND FUTURE WORK

Contents

6.1	Conclusions	45
6.2	Future work	45

In this chapter, the conclusions of the work, as well as its future extensions are shown.

6.1 Conclusions

Overall, the experience in developing the project has been very satisfactory. At the beginning, when I had to choose the theme of the project, I was very hesitant, and I did not know if finally the development of a website was going to be to my liking. Even so, having made use of tools and languages that I knew thanks to subjects of the degree, has allowed me to enjoy the challenge of learning new frameworks.

I think it is also essential that a developer, whether of general software or video games, experience developing a project of such magnitude as a final project, since it is necessary a commitment, an anticipation and a willingness to meet self-imposed goals, proposing a challenge that surely before I would not have faced satisfactorily.

6.2 Future work

Talking about the future of the project and the application developed in it, I believe that after finishing the final degree work the application support will be finished.

Although it has been a satisfactory path, hundreds of applications with the same purpose exist in the market, and companies with a lot of computing power and data storage can beat any independent developer who wants to compete.

I think it has been a satisfactory experience for my development as a programmer, and more than the product itself, I keep the path I have followed when doing the project, which has made me improve as a programmer and as a professional.

BIBLIOGRAPHY

- [1] UJI, “Vj1229.” <https://ujiapps.uji.es/sia/rest/publicacion/2021/estudio/231/ asignatura/VJ1229>.
- [2] PgAdmin4, “Pgadmin homepage.” <https://www.pgadmin.org/>.
- [3] Redhat, “What is an api?.” <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces#:~:text=Una%20API%20o%20interfaz%20de,el%20software%20de%20las%20aplicaciones>.
- [4] H. Virk, “Typescript backend framework adonis.js.” <https://docs.adonisjs.com/guides/introduction>.
- [5] E. You, “Javascript frontend framework vue.js.” <https://vuejs.org/guide/introduction.html>.
- [6] Riot Games, “League of legends api.” <https://developer.riotgames.com/>.
- [7] OP.GG, “Op.gg.” <https://euw.op.gg/>.
- [8] Riot Games, “League of legends game.” <https://www.leagueoflegends.com/es-es/>.
- [9] Wikipedia, “Moba genre.” https://es.wikipedia.org/wiki/Videojuego_multijugador_de_arena_de_batalla_en_l%C3%ADnea.
- [10] Riot Games, “Riot games developer.” <https://www.riotgames.com/es>.
- [11] Vandal, “League of legends, most popular game in the world.” <https://vandal.lespanol.com/noticia/1350727096/league-of-legends-es-el-juego-mas-popular-con-mas-de-8-millones-de-jugadores-diarios/>.
- [12] League of Legends Wiki, “Kill/death/assist ratio.” https://leagueoflegends.fandom.com/wiki/Kill_to_Death_Ratio.
- [13] Geeks for Geeks, “Fullstack development.” <https://www.geeksforgeeks.org/what-is-full-stack-development/>.
- [14] Wikipedia, “Json.” <https://es.wikipedia.org/wiki/JSON>.

-
- [15] PostgreSQL, “Postgresql.” <https://www.postgresql.org/>.
- [16] Wikipedia, “Single-page application.” https://es.wikipedia.org/wiki/Single-page_application.
- [17] Wikipedia, “Html5.” <https://es.wikipedia.org/wiki/HTML5>.
- [18] Mozilla Firefox, “Javascript.” <https://developer.mozilla.org/es/docs/Web/JavaScript>.
- [19] Wikipedia, “Css.” <https://es.wikipedia.org/wiki/CSS>.
- [20] SearchAppArchitecture, “What is an endpoint?” <https://www.techtarget.com/searchapparchitecture/definition/API-endpoint#:~:text=An%20API%20endpoint%20is%20a,server%20and%20receiving%20a%20response>.
- [21] Riot Games, “Jax champion.” <https://www.leagueoflegends.com/es-es/champions/jax/>.
- [22] Vue.JS, “Components, basic unit of vue.js.” <https://vuejs.org/guide/essentials/component-basics.html>.
- [23] League of Graphs, “League of graphs.” <https://www.leagueofgraphs.com/es/>.
- [24] U.GG, “U.gg.” <https://u.gg/>.
- [25] Redis, “In-memory data store.” <https://redis.io/>.
- [26] Redis, “In-memory data store performance study.” <https://www.datadoghq.com/pdf/Understanding-the-Top-5-Redis-Performance-Metrics.pdf>.
- [27] Wikipedia, “What is data pipeline?” [https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_\(electr%C3%B3nica\)#:~:text=La%20segmentaci%C3%B3n%20\(en%20ingl%C3%A9s%20pipelining,usa%20principalmente%20en%20los%20microprocesadores](https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(electr%C3%B3nica)#:~:text=La%20segmentaci%C3%B3n%20(en%20ingl%C3%A9s%20pipelining,usa%20principalmente%20en%20los%20microprocesadores).
- [28] Redis, “Task queuing.” <https://redis.com/ebook/part-2-core-concepts/chapter-6-application-components-in-redis/6-4-task-queues/>.
- [29] Redis, “Task queuing.” <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/try...catch>.
- [30] Docker, “What is docker?” <https://www.docker.com/>.
- [31] UJI, “Vj1225.” <https://ujiapps.uji.es/sia/rest/publicacion/2021/estudio/231/assignatura/VJ1225>.
- [32] Citrix, “What is a vm?” <https://www.citrix.com/es-es/solutions/vdi-and-daas/what-is-a-virtual-machine.html>.
- [33] Wikipedia, “What is unix?” <https://es.wikipedia.org/wiki/Unix>.

-
- [34] Wikipedia, “What is linux?.” <https://es.wikipedia.org/wiki/Linux>.
- [35] Wikipedia, “What is ubuntu?.” <https://es.wikipedia.org/wiki/Ubuntu>.
- [36] UJI, “Vj1220.” <https://ujiapps.uji.es/sia/rest/publicacion/2021/estudio/231/asignatura/VJ1220>.
- [37] TypeScript, “Typescript documentation.” <https://www.typescriptlang.org/>.
- [38] Geeks for Geeks, “What is a post method?.” [https://www.geeksforgeeks.org/http-get-post-methods-php/#:~:text=POST%20Method%3A%20In%20the%20POST,be%20visible%20in%20the%20URL.&text=The%20query%20string%20\(name%2Fweight,body%20of%20a%20POST%20request](https://www.geeksforgeeks.org/http-get-post-methods-php/#:~:text=POST%20Method%3A%20In%20the%20POST,be%20visible%20in%20the%20URL.&text=The%20query%20string%20(name%2Fweight,body%20of%20a%20POST%20request).
- [39] CloudAppi, “Cloudappi s.l.” <https://cloudappi.net/>.
- [40] Wikipedia, “What is tarball?.” [https://en.wikipedia.org/wiki/Tar_\(computing\)](https://en.wikipedia.org/wiki/Tar_(computing)).
- [41] Adobe, “Illustrator.” https://www.adobe.com/es/products/illustrator.html?mv=search&mv=search&sdid=KCJMLF6&ef_id=EAIAIQobChMI8byYo8n79wIVFdtRCh3_oQNSEAAAYASAAEgLJK_D_BwE:G:s&s_kwid=AL!3085!3!441886954705!e!g!illustrator!1479761001!62724397092&gclid=EAIAIQobChMI8byYo8n79wIVFdtRCh3_oQNSEAAAYASAAEgLJK_D_BwE.
- [42] DeepL, “English translator.” <https://www.deepl.com/translator>.
- [43] Grammarly, “English cohesion and coherence corrector.” <https://www.grammarly.com/>.
- [44] Riot Rate Limiter, “Limiter that ensures the compliance of the riot api limits.” <https://github.com/fightmegg/riot-rate-limiter>.
- [45] Wikipedia, “What is parsing?.” <https://en.wikipedia.org/wiki/Parsing>.



RIOT API

This appendix will be used to describe the technical sections related to one of the most important parts of the project: the Riot Games API [6]. It is of utmost importance to describe the various problems that have arisen when interacting with the free but limited tool provided by the League of Legends [8] developer.

Once I started working with it when developing the BackEnd, many doubts and problems arose due to the limitations that Riot imposes on all developers who want to interact with its API and who want to make use of the data stored in the multiple databases that its servers keep.

A.1 Endpoints

It is time to talk about the endpoints present in the API. There are multiple endpoints, each one serving a purpose within the same, besides being able to make use of endpoints that belong to other games of the developer. Therefore, the endpoints that have been used are going to be listed, with a brief description of the items that each one returns.

- **Match-V5:** This endpoint allows to obtain information about the games from the puuid (unique id) of a summoner. A list of summoner's games can be obtained, and at the same time, from the id of the games in this list, the details of each one of them.
- **Summoner-V4:** This endpoint allows to obtain the puuid (unique id) of a summoner, using their name. This endpoint is crucial to allow the application to

transform the name given by a user in a unique id that will be used in every other endpoint.

- **League-V4:** This endpoint allows to obtain information about the ranks of a summoner in a given queue.

Combining these three endpoints, plus sub-endpoints present within them, which were specified in the Figure 3.6 in the BackEnd design, absolutely all the necessary data for the application is obtained.

An example of the use of one of the endpoints could be the following. Imagine that it is necessary to obtain for the first time the puuid of a summoner that has never made a search in the application, and therefore the database does not have any indication of the existence of the same one. It will be necessary to make a request to the **Summoner-V4** endpoint, sending the name as a string, in order to obtain this unique id from the API.

Once a name is passed to the endpoint, if it exists, it will return an object class of its own called SummonerDTO, which contains the sub-objects described in the Figure A.1.

SummonerDTO - represents a summoner

NAME	DATA TYPE	DESCRIPTION
accountId	string	Encrypted account ID. Max length 56 characters.
profileIconId	int	ID of the summoner icon associated with the summoner.
revisionDate	long	Date summoner was last modified specified as epoch milliseconds. The following events will update this timestamp: summoner name change, summoner level change, or profile icon change.
name	string	Summoner name.
id	string	Encrypted summoner ID. Max length 63 characters.
puuid	string	Encrypted PUUID. Exact length of 78 characters.
summonerLevel	long	Summoner level associated with the summoner.

Figure A.1: Object returned by the Riot API when requesting a Summoner puuid

Knowing how a basic but crucial endpoint such as the **Summoner-V4** works, it is easy to imagine how to obtain information from the other endpoints. This is where

Jax 3.2.1.1, the intermediary that was described in the BackEnd design, comes into play. By designing a generalized interface for API calls, Jax 3.2.1.1 simply has to take care of completing a path that shares common points, and is simply differentiated by the name of each of the endpoints.

This is how, thanks to the use of an intermediary, an elegant solution is achieved instead of having to program the calls one by one depending on the information to be obtained. All the endpoints related to the BackEnd database have their own name, which translates into an existing name in the Riot API.

```
1 export default class JaxRequest {
2   private region: string
3   private config: JaxConfig
4   private endpoint: string
5   private limiter: RiotRateLimiter}
6
7   public async execute() {
8     const url = `https://${this.region}.api.riotgames.com/lol/${this.endpoint}`
9
10    try {
11      const resp: any = await this.limiter.executing({
12        url,
13        token: this.config.key,
14      })
15      return JSON.parse(resp)
16    } catch ({ statusCode, ...rest })
```

As can be seen in the code, Jax 3.2.1.1 obtains the region, the endpoint to be accessed, and a limiter object that will be in charge of complying with the demanding limitations imposed in the API. Simply with these parameters, and by completing the url present in the request execution method, Jax 3.2.1.1 will quickly find the desired information.

This limiter, a crucial part of the project to allow the correct execution of requests, was obtained from a free repository on GitHub, developed by a fan who also had to deal with Riot's API. The JavaScript library is called Riot-Rate-Limiter [44].

It is also important to know what kind of information and in what form it is obtained from the API. The class of each type of object obtained is given by the API, and can be seen in the Data Type section in Figure A.1. But the information is given in a JSON [14], and therefore, it is necessary to treat it before it can be used in the database and passed to the FrontEnd. This is called JSON [14] Parsing [45], and will be discussed in Appendix B.



JSON PARSING

Parsing [45] is a technique used for obtaining and classifying each of the objects that make up a JSON [14] file. For example, when requesting a game from the API, it returns a JSON [14] with an infinite number of variables, objects and statistics. It is the developer's job to choose, discard and classify this information, since most of the data is used by Riot for various purposes but is not necessarily useful for applications such as the one developed in the project. An output example of a call to the **Match-V5** endpoint could be the following:

```
1 "metadata": {
2   "dataVersion": "2",
3   "matchId": "EUW1_5805593624",
4   "participants": [
5     "UtQQvGUFnuqcDyBGU0Q00zIcIjAX2_DmMwG_5P5mtAq3RmTuIn8bWczxVQut0NM4eu_a_9InAKtdQA",
6     "uevndakk6HP7E1gLn2Jr44hJyTDYUQC3ngd9VNHiPAusqWAMbqBbx7P70ZKIafgWv0aiIRxLxsFLRw",
7     "oJpwbms8Ra8ny9M5NU-GgVmUS7wsVS9sffkcU__0AWeUI3fu-w1pyePqBa3KuB6F7pwQT2cPtqI7-A",
8     "-ov4EyI701d09Z3IqkRa_D6xjtg9wdElR0VvkQdqS1WaTkKEmw6zxmi0ydQvmWBmMR7HQpml265SmQ",
9     "pGQANpwdJ_bG3xTCAC46SkvLM6-Qf4TioalvezHtnAkJkYRQ1IYBu10k0Pz1eRjV2kDDAt0GU4CB4A",
10    "mXlXnDvq1w10oL2rCS8G6fq0xUNJsB6N3zCe2FoIqiBR8aMKfLc1uZimbVRRmLSwvBVIkv0JRGnsyQ",
11    "tXcvCECbdVe0xhEigREQjIjjjobM7LTL0CF04B0zsII-Bkpl00MpMx3gzIMEt4MkwtBgyaxHwjSxFHQ",
12    "wcwAW7ybxR2xP78YMr1Lk1KDXLs__qslvavXLF16o0sEWRPuhgdwW5PgyED3TgxEKxftV6M8UX_L1g",
13    "-vfjEHTxQ2LHt8ksQsS4WiPa2p_GpLkhPlgoLTfb4-NMjzlcwECpJo8ZnB7_puiY7JE14uxRGj2Pig",
14    "dWCCVPZuRu2wy4aMzF58qeRHyZrP6oXnee2NlfXiBy08uJBFQs-WMdv-ETFcfhns--LDprVDqe0IVw"
15  ]
16 },
17 "info": {
18   "gameCreation": 1648924674000,
19   "gameDuration": 931,
20   "gameEndTimestamp": 1648925654289,
21   "gameId": 5805593624,
22   "gameMode": "ARAM",
```



```
23   "gameName": "teambuilder-match-5805593624",
24   "gameStartTimestamp": 1648924723175,
25   "gameType": "MATCHED_GAME",
26   "gameVersion": "12.6.432.1258",
27   "mapId": 12,
28   "participants": [
29     {
30       "assists": 29,
31       "baronKills": 0,
32       "bountyLevel": 0,
33       .
34       .
35       .
36       .
```

This file has a total of 2892 lines, which for obvious reasons have been omitted, since getting a general idea of how a JSON [14] object works is more than enough for the purpose of this appendix. It is visible how a game JSON [14] has the puuid of each of the players in it, information such as the time of creation, the game mode, the id within the server, and an object containing the statistics of each of the participants, in addition to many other data.

Here lies an example of how to parse a complete game, including each of the players in the game and their statistics. The elements seen in the code are those shown in the relational database schema in Figure 3.10.

```
1 public async parseOneMatch(match: MatchDto) {
2   let parsedMatch: Match | null = null
3   let trx: TransactionClientContract | undefined
4
5   try {
6     // Start transaction
7     trx = await Database.transaction()
8
9     const gameDuration =
10      match.info.gameDuration > 100_000
11      ? Math.round(match.info.gameDuration / 1000)
12      : match.info.gameDuration
13
14     const isRemake = gameDuration < 300
15
16     // - 1x Match
17     parsedMatch = await Match.create(
18       {
19         id: match.metadata.matchId,
20         gameId: match.info.gameId,
21         map: match.info.mapId,
22         gamemode: match.info.queueId,
23         date: match.info.gameCreation,
24         region: match.info.platformId.toLowerCase(),
25         result: match.info.teams[0].win ? match.info.teams[0].teamId : match.info.teams[1].teamId,
```

```
26     season: getSeasonNumber(match.info.gameCreation),
27     gameDuration,
28   },
29   { client: trx }
30 )
31
32 // - 2x MatchTeam : Red and Blue
33 for (const team of match.info.teams) {
34   let result = team.win ? 'Win' : 'Fail'
35   if (isRemake) {
36     result = 'Remake'
37   }
38   await parsedMatch.related('teams').create({
39     matchId: match.metadata.matchId,
40     color: team.teamId,
41     result: result,
42     barons: team.objectives.baron.kills,
43     dragons: team.objectives.dragon.kills,
44     inhibitors: team.objectives.inhibitor.kills,
45     riftHeralds: team.objectives.riftHerald.kills,
46     towers: team.objectives.tower.kills,
47     bans: team.bans.length ? team.bans.map((ban) => ban.championId) : undefined,
48     banOrders: team.bans.length ? team.bans.map((ban) => ban.pickTurn) : undefined,
49   })
50 }
51
52 matchPlayers.push({
53   match_id: match.metadata.matchId,
54   participant_id: player.participantId,
55   summoner_id: player.summonerId,
56   summoner_puuid: player.puuid,
57   summoner_name: player.summonerName,
58   win: team.win ? 1 : 0,
59   loss: team.win ? 0 : 1,
60   remake: isRemake ? 1 : 0,
61   team: player.teamId,
62   team_position:
63     player.teamPosition.length && queuesWithRole.includes(match.info.queueId)
64       ? TeamPosition[player.teamPosition]
65       : TeamPosition.NONE,
66   kills: player.kills,
67   deaths: player.deaths,
68   assists: player.assists,
69   kda: kda,
70   kp: kp,
71   champ_level: player.champLevel,
72   champion_id: player.championId,
73   champion_role: ChampionRoles[champRoles[0]],
74   double_kills: player.doubleKills,
75   triple_kills: player.tripleKills,
76   quadra_kills: player.quadraKills,
77   penta_kills: player.pentaKills,
78   baron_kills: player.baronKills,
79   dragon_kills: player.dragonKills,
```

```
80     turret_kills: player.turretKills,
81     vision_score: player.visionScore,
82     gold: player.goldEarned,
83     summoner1_id: player.summoner1Id,
84     summoner2_id: player.summoner2Id,
85     item0: player.item0,
86     item1: player.item1,
87     item2: player.item2,
88     item3: player.item3,
89     item4: player.item4,
90     item5: player.item5,
91     item6: player.item6,
92     damage_dealt_objectives: player.damageDealtToObjectives,
93     damage_dealt_champions: player.totalDamageDealtToChampions,
94     damage_taken: player.totalDamageTaken,
95     heal: player.totalHeal,
96     minions: player.totalMinionsKilled + player.neutralMinionsKilled,
97     critical_strike: player.largestCriticalStrike,
98     killing_spree: player.killingSprees,
99     time_spent_living: player.longestTimeSpentLiving,
100   })
101 }
102 await Database.table('match_players').multiInsert(matchPlayers)
103
104 return parsedMatch
105 }
```



DATA DRAGON

Data Dragon, or DDragon for short, is a set of static data files that provides images and info about champions, runes, and items. This includes info to translate champion IDs to names.

Because the data in DDragon only changes when new patches come out, it is possible to cache the data by saving it into a computer. An app can then load the data from disk rather than requesting it over the Internet. This will speed up the app and reduce the load on Riot's servers, which ensures the servers don't go down due to abnormally high usage. In general, it's a good idea to cache data that will be used often and that does not change often at the same time.

DDragon provides two kinds of static data; data files and game assets. The data files provide raw static data on various components of the game such as summoner spells, champions, and items. The assets are images of the components described in the data files.

There are two kinds of data files for champions. The `champion.json` data file returns a list of champions with a brief summary. The `individual champion JSON` files contain additional data for each champion.

It is possible to obtain too assets related to a specific champion. For example, using the following URL it is possible to obtain a specific skin that the desired champion possesses.

DDragon also provides the same level of detail for every item in the game. Within

DDragon it is possible to find info such as the item's description, purchase value, sell value, items it builds from, items it builds into, and stats granted from the item.

Thanks to DDragon it is possible to make a basic structure of URLs to call to obtain the different assets of the game. The structure followed in the project is explained in the following lines of code.

```
1 public async obtainGeneralChampionData(){
2     url = "http://ddragon.leagueoflegends.com/cdn/12.10.1/data/en_US/champion.json";
3     const requestCached = await Redis.get(url);
4 }
5
6 public async obtainSpecificChampionData(championName : string){
7     url = "http://ddragon.leagueoflegends.com/cdn/12.10.1/data/en_US/champion/${championName}.json";
8     const requestCached = await Redis.get(url);
9 }
10
11 public async obtainSpecificChampionSkin(championName : string, skinNumber : number){
12     url = "http://ddragon.leagueoflegends.com/cdn/12.10.1/data/en_US/champion/${championName}_${skinNumber}.json";
13     const requestCached = await Redis.get(url);
14 }
15
16 public async obtainSpecificItemData(itemName : string){
17     url = "http://ddragon.leagueoflegends.com/cdn/12.10.1/data/en_US/${itemName}.json";
18     const requestCached = await Redis.get(url);
19 }
```

