Project Kaisel

Web Application Displaying

Riot-Games and Twitch Statistics and Streams

Martin Wong, Lucas Lee, Leo Chung, Cesar Figueroa Socarras

Github: Project Kaisel Github Repository

Website: Project Kaisel Website

Table of Contents

Project Overview

For our project, we designed and implemented a website that displays statistics for the popular multiplayer games *League of Legends*, *VALORANT and TFT*. In addition, we integrated the Twitch API to display the top streamer and popularity for both games. If there is any unfamiliarity with Twitch, it is the most popular streaming platform mainly targeted for gaming. Our project was divided into two main phases in case of not having enough time. We decided to have all our features reside in our web application before proceeding with a discord bot.

Phase 1: Web Application

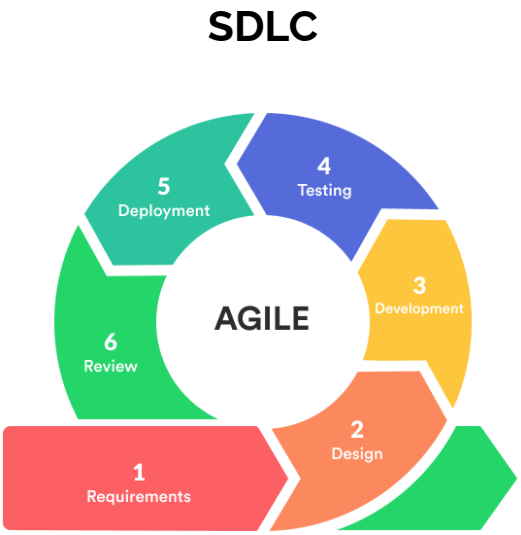
Our first phase of our project was to create a web application for games supported by the company Riot Games and display the top streamers of those games from a streaming platform called Twitch. We used the Riot Games API to display in game statistics of players searched by the user and it would include statistics like win-rate and kill/death ratios

. This was developed using the following:

- ReactJS framework that allows us to write html and css within a javascript environment. React made it easier to encapsulate the web application into its own components and state. As a result, React made it much easier to make global changes across multiple webpages and also made re-rendering data much easier.
- Within the website, we have the following:
    - A home page that overviews our website as a whole. Displaying the games we support, a search bar to search for a username for a game and a button that is hyperlinked to our twitch page.
    - A statistics page for users to search up statistics for their desired game and account
        - Ranked Solo/Duo stats (League of Legend)
        - Ranked Flex stats (League of Legend)
        - Top 5 Most Played Games/Champions (League of Legend)
        - Top number of players (Valorant)
    - A Twitch page that has the following
        - Top Categories
        - Featured Streamer
        - Top Streamer of the games we support
    - A contact page displaying the members of the project
    - A 404 page in case a user types the wrong link

Phase 2: Discord Bot

In our second phase, our original plan was to create a discord bot that will use both API's and have its own set of commands. However, mid-way through March, we unanimously decided on scraping the discord bot and focusing entirely on our web application. This was the best decision we made as we encountered more and more challenges with the web application itself even near the deadline of the project.



Chosen SDLC Overview

The Software Development Life Cycle (SDLC) we have chosen is Agile. Agile is the perfect fit for both the team and for our project plan for the following reasons.

First, as post-secondary students, we are constantly subjected to numerous changes regarding our education and plan our daily lives around that. As a result, while we may have a general idea of what to expect in each course, there could come a time where we are unable to achieve the goals we initially set out for ourselves. Therefore, preparing our plans based on a flexible SDLC will allow us to accommodate and adjust to changes in both the project and in our daily lives.

Secondly, our project is ambitious. We aim to build both a website and Discord in the span of a month and a half (Second half of February and all of March). Being able to incorporate a project such as this one in such a short amount of time, including our daily responsibilities, is time extensive and desires that we are familiar with the languages we are using and the interfaces that are applied to those languages. This requirement is one that only some of the members on our team possess, while others need to be trained. Taking into account the time needed to plan for the project, train individuals, and build both website and Discord bot, it is imperative that our SDLC centers on flexibility. As a result, Agile accommodates the time needed to implement all three requirements listed above.

Based on the two reasons listed above, the team has concluded that an agile framework is the optimal solution to our situation.

What Went Well

With every project, teamwork is a key component that can make or break a project's success. It is no understatement to say our teamwork was superb throughout the entire semester. The group members were in constant communication with each other having weekly meetings and notifying the team on the tasks they have done with great detail. Furthermore, tasks were allocated such that it fit the person who was doing it and that each group member was not overburdened. For example, setting up the initial React project was left to Cesar as he had previous experience with React. As for the planning, we did have a good idea of what our project was going to look like and the features we were planning to have on it. However, our inexperience with a project such as this one left a few holes in it especially when it came to the testing and CI/CD parts. Nonetheless, teamwork dynamics and requirement gathering were the parts of this project that we excelled in.

What Can Be Improved On

As Agile is a SDLC that focuses on repeated iterations of a task, we could not fully accomplish that. There were parts of the project that we were unfamiliar with at the time or simply had not learned yet and therefore could not integrate these features until a later date. Some parts of the cycle include the testing and deployment phases as those were ones that we were unfamiliar with. Furthermore, our limited experience prevented us from fully utilizing the design phase and fully understanding our initial requirements. On the other hand, we made a good decision to choose Agile over our original plan to use the Waterfall model. The extra flexibility was really needed halfway through our project.

High-level Implemented Features

Since we are making both a website and Discord bot that displays League and Valorant stats plus any Twitch streamers that play the associated games, we have chosen the Riot Games API for League of Legends and Valorant, the Twitch, and Discord API. Each API has the following uses:

Riot Games API

- League of Legends
    - Username
    - Current rank
    - Kill/Death/Assist ratio (K/DA)

    - Favorite role/champion
- Valorant
    - Top 200 players per region in the current act displaying:
        - Username
        - Tagline
        - Competitive Rank

Twitch API

- Top (1-3) streamer(s) for each game
- Current views on Twitch for game
- Allow user to search for streamers/channels playing their chosen game

API Application Features

Riot Games

League of Legends Ranked Solo Duo 5x5/Flex Sr

This section provides users with the opportunity to quickly grab the statistics of an in-game player by searching for the player's League of Legends username and region where that user plays. When the user has found the player they're looking for, they will be provided with the following:

- Player's username
- Game rank
- Win rate by percentage

League of Legends Champions

This section is combined with the Solo Duo 5x5/Flex Sr functions as the top champions the player has selected can be seen alongside the player's statistics. Once a user

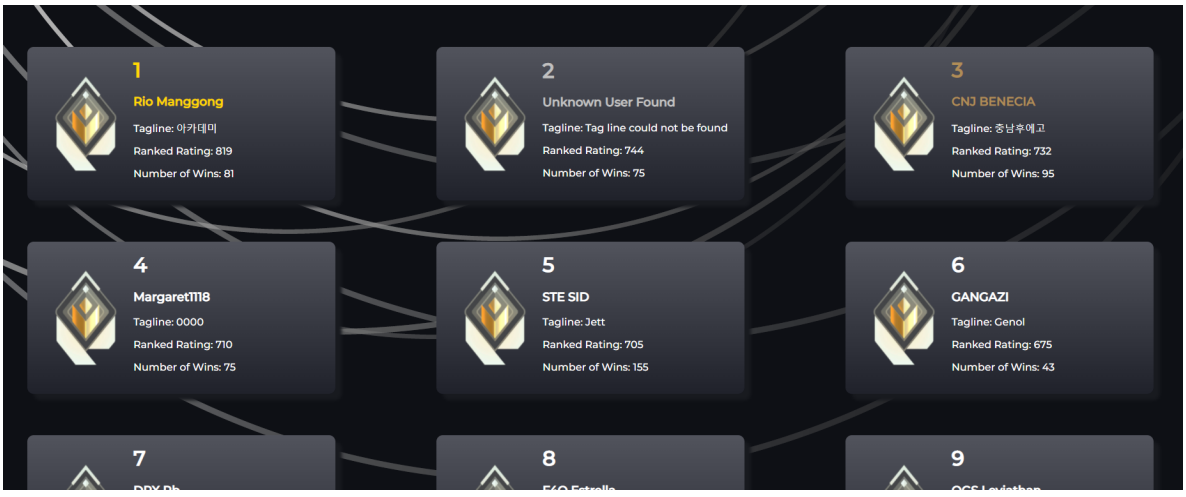
searches for their chosen player, the user will be supplied with the champion's following statistics:

- The champion's name
- Role of the champion in the game
- Average kills, deaths and assists per champion
- Lastly, the farming per minute

Champ	Role	Kills	Deaths	Assist	Farm/min
Rell	UTILITY	2.3	6.3	24.3	1.3 CS
Taliyah	UTILITY	8.5	12.0	17.0	1.0 CS
Gwen	TOP	7.0	7.0	4.0	6.6 CS
Camille	TOP	3.0	7.0	4.0	5.4 CS
Zac	UTILITY	3.0	7.0	22.0	1.3 CS
Rell	UTILITY	5.0	3.5	18.0	1.4 CS
Gwen	TOP	6.0	4.0	2.0	7.2 CS

Top Valorant Players by Region

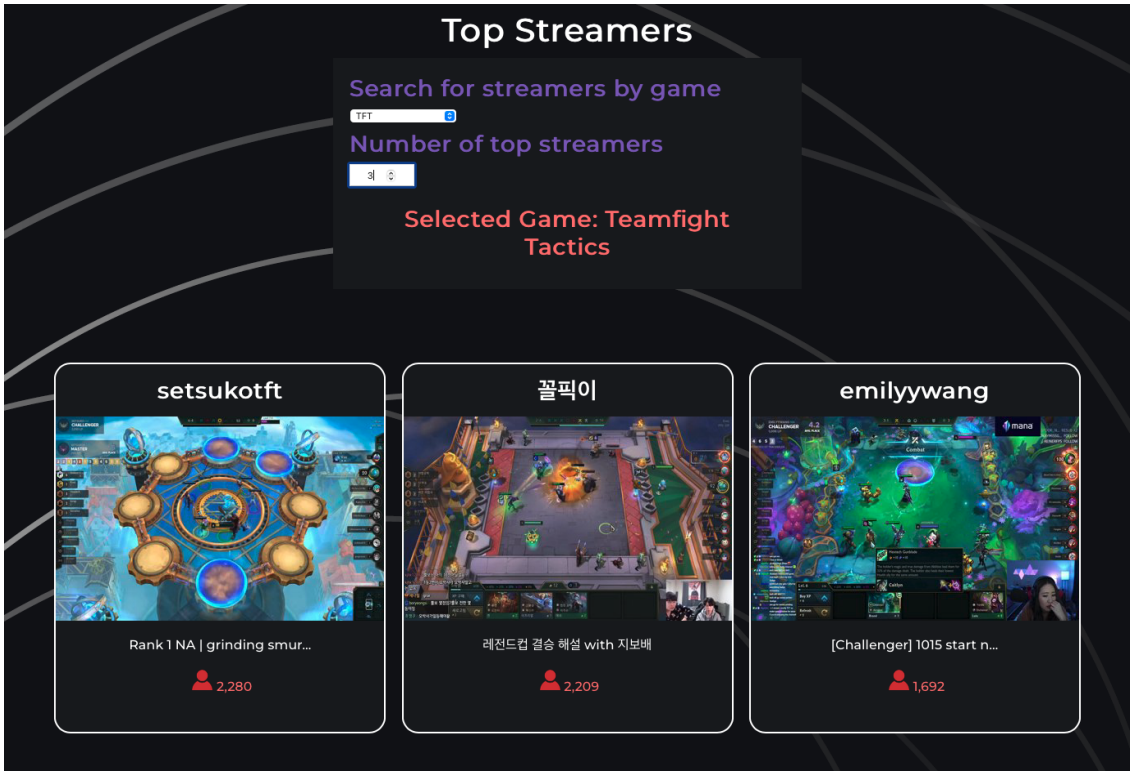
One of the features implemented using the Riot Games API is the usage of finding the top players in the current act by region. While more features can be implemented with Valorant, the restrictions on our API keys prevent us from developing a more cohesive search. Instead, we are limited to displaying a given number of top players based on how many the user wants to view at the time.



Twitch

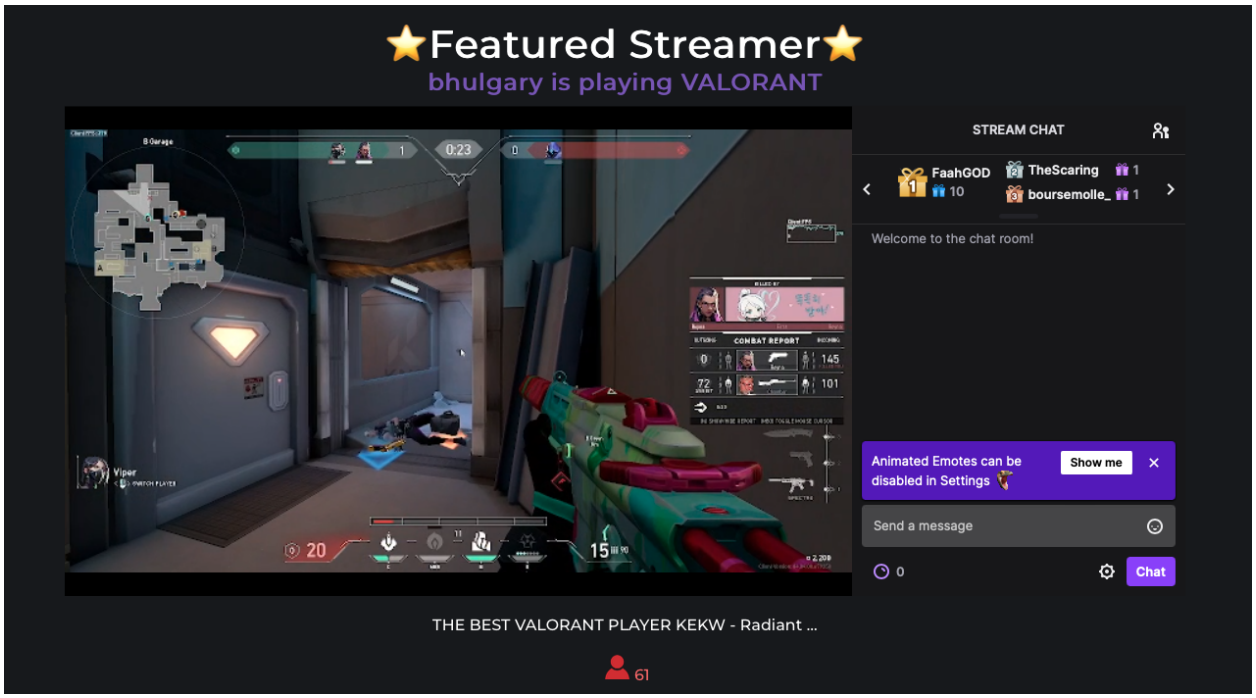
Search the Top Streamers By Chosen Game

Streamers are displayed from most viewed to least viewed in descending order based on the selected game. How many streamers are shown are also based on user input. The streamers are displayed with a thumbnail of their live stream, title of the stream, and live viewer count.



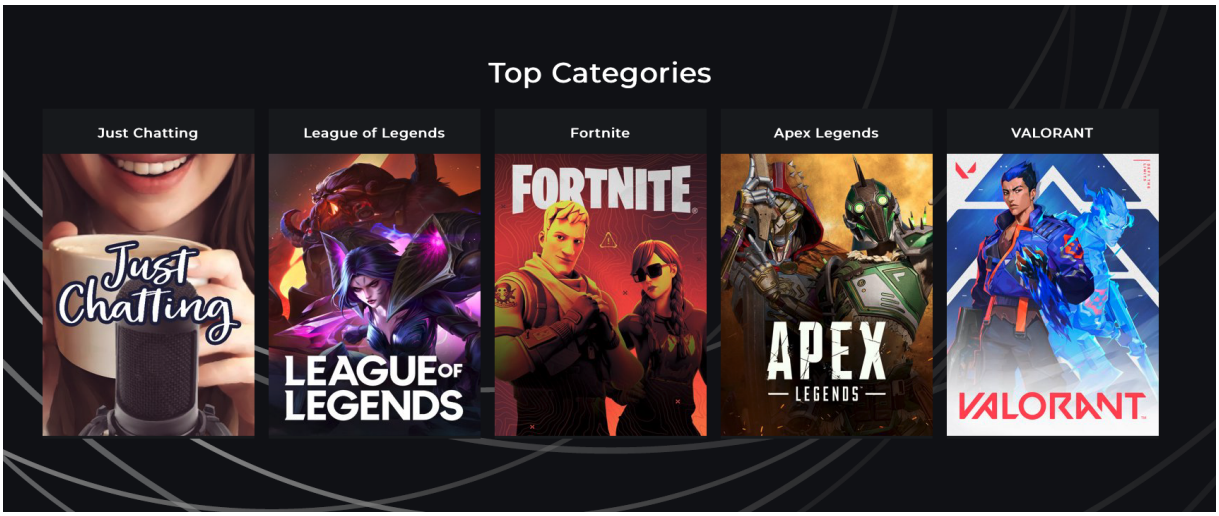
Watch a Featured Streamer's Stream Without Having to Navigate to Twitch

A streamer was randomly chosen out of 100 streams of a randomly picked game (League, Valorant, TFT). The streamer is then rendered using a special react library that embed the stream into the website. The viewer count also updates every 20 seconds as the Twitch API is slow for live viewer changes.



The Top 5 Most Viewed Categories on Twitch

The top 5 most viewed game categories are displayed from a get request to a specific endpoint. A user can also click on the category to be redirected to Twitch.



Unit and Integration Tests

Unit Tests

For the unit tests, we have seven files that test various parts of our application and check if we have the required information we want to display. The tools used in this step include the React testing library as we are using React JS for our application and Jest to describe and test each feature's part. We have a total of 31 tests among 7 testing files. There were no issues regarding creating the tests.

- GameCard Test
- Features Test
- Footer Test
- Header Test
- Information Test
- Project Member Test
- Twitch Intro Test

Integration Tests

For the integration tests, we focused on three features of each API. Just like the unit tests, we used React JS and Jest to test the features. In this area, we encounter numerous problems. The first problem was figuring out how to connect the tests to send out data just like a user would. We found that we had to create an object that had the same variables as the ones found in our features. The second issue was regarding understanding the differences between the two APIs in testing. The features used in the Riot Games API had constructor objects that we could make sense of to send the data. However, when it came to the Twitch API, the format was different and took a lot of time to get done.

- Riot Games League of Legends Champions Statistics Test
- Riot Games League of Legends Player Statistics
- Riot Games Valorant Current Act Players Test
- Twitch Featured Streamer Test
- Twitch Top Categories Test
- Twitch Top Streamers Test

CI/CD Infrastructure

Test and Build

The first step to developing our CI/CD infrastructure was to choose a service to test our application. Since the repository is located in Github, it was deemed appropriate to use GitHub Actions as our continuous integration infrastructure.

In the testing and building phase of our repository, we first used a template implementing the Node.js framework into our project. As our project uses the React (JS) framework, we constantly need to install and start any dependencies using Node and thus need to call any "npm" commands that enable the website to function. The Node version we use for our CI is 16.x. Our "build-deploy.yml" must go through a few steps to test and build our website's features:

1. Choose a repository to checkout
    - We must use the actions keyword to checkout our repository
2. Set up Node.js in our workflow
    - Node.js supports a variety of versions that each have their own release timeline. When we're setting up Node.js, we need to specify which version we're using. As a result, we can apply the matrix to this step by stating which Node version is used from the matrix.
3. Install our Node packages
    - Usually, we would use "npm ci" for this step as the command is like "npm install" but used specifically for testing. Unfortunately, we ran into an unknown error regarding "npm ci". The error mentioned that "fsevents" was not accessible from the jest-haste-map. To work around this, we changed the command to "npm i fsevents@latest -f --save-optional" which worked on all 3 build versions (12.x, 14.x, and 16.x). However, we found 12.x and 14.x were not fully compatible with this command. As a result, we removed versions 12.x and 14.x from the test file leaving only 16.x.
4. Build our project
    - The "npm run build" command was used followed by "--if-present". There is nothing big to note here other than this worked as expected and caught any unforeseen parts of the application that were overlooked such as unused variables.
5. Test our files
    - The basic command "npm run test" is used for this step. Initial implementation of the CI file found that this step would result in an error for the reason that we had no tests written at the time.
6. Finally, upload our files such that they be built and deployed

- Additional information about Node.js releases

- Additional Information about Github Actions
- Refer to .github/workflows/build-deploy.yml for further technical details.

Deploy

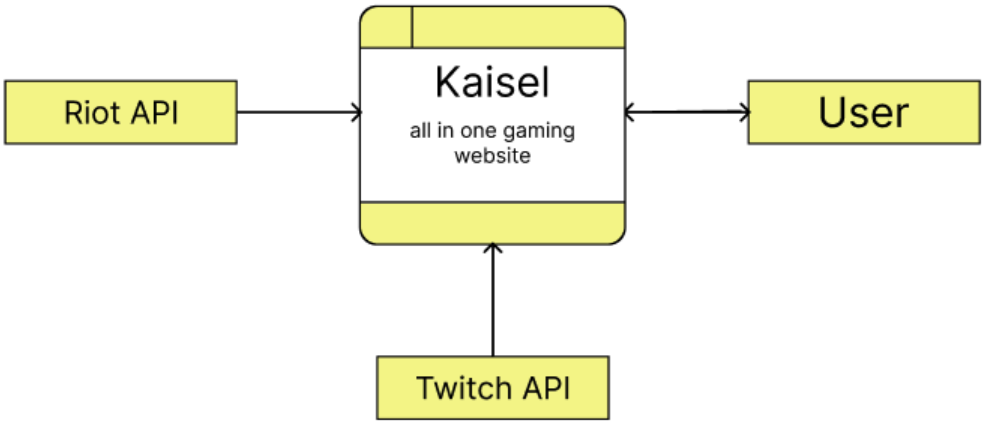
As Github is where the repository is located and Github Actions provides us with a continuous integration environment, we have decided to host our website through Github Pages. Our "build-deploy.yml" must go through a few steps to deploy our website once its done testing and building:

1. Download the artifacts
    - This simply gets and downloads the files we are working with.
2. Deploy to Github Pages
    - Our application reroutes our working files to a branch called "gh-pages" which we will use to deploy our website.
3. Validate that everything is working in Github Pages
    - This part gave us a lot of trouble. First, we noticed a few images and links were not working with Github pages despite working on our local environments. To fix this, we needed to make sure that those images and links were compatible on both Github pages and our local workplace. The second problem occurred when we tried to search for a player with our Riot Games search bar (see the "Stats" page for further details). When a user would search for a player in Github Pages, the link would lead us to a 404 page rather than display a player's statistics. Our hypothesis was that the React routing was not working as we wanted it to. In the end, Cesar and Martin managed to get it working so that the user could search for a Riot Game's username through Github Pages.

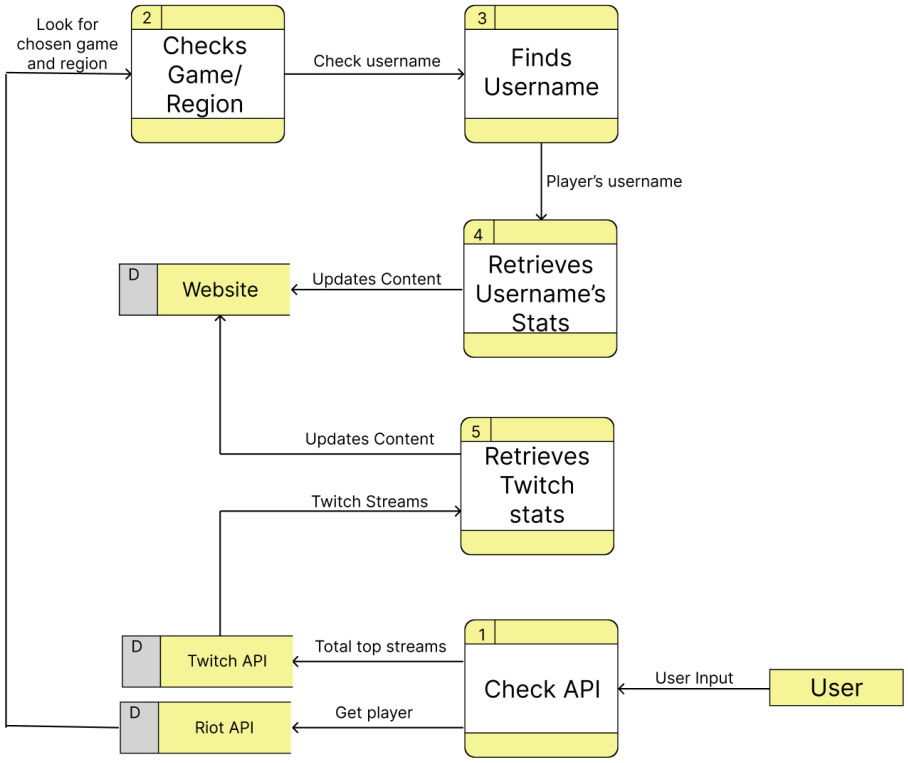
- Additional Information about Github Pages
- Github - What are Artifacts?

High-Level Data Flow Diagram

Level 0



Level 1



Project Takeaway

Lessons Learnt

- Practicality: Our group was way too ambitious in the fact that we wanted a backend with a database and a Discord bot. We also had a list of features to implement with the API's. However, we had no idea about the challenges that would come up until we started implementation.
- Task Allocation: While we had a general overview of what we wanted to do, we could not come up with a concrete breakdown of the task allocation each member should do. As a result, our task allocation was a bit messy and did not have a set structure when we first planned everything out.

Project Challenges

- Integration Testing: This part proved especially difficult as there were vast differences between the two APIs when it came to getting and setting data.
- Riot Games API: The API proved to be more challenging than initially thought. There were a lot more complexities involved in retrieving and displaying a user's data. In addition, we needed to ensure that a player had all the correct statistics such as their win ratio, favorite champion, and other necessities.
- Learning React in a short time frame: React itself is not an extensively hard framework. The hard part was having to learn and become familiar with it in such a short time frame as well as being familiar with how to call APIs, display data to the page, and connect components together.

Task Allocation

Each member of the team played an important role in the planning phase of the project. Listed below are the tasks that everyone was assigned and/or contributed to:

Cesar

- API researcher
- Decided on features
- Decided on the tech stack
- Developed Riot Games: League of Legends features
- Wrote unit and integration tests

Leo

- Assisted on wireframe sketching
- Built CSS stylings for mobile devices

- Created data flow diagrams
- Wrote user stories

Lucas

- Assisted on the powerpoint (presentation)
- Developed Riot Games: Valorant features
- Report #1 co-writer and editor and report #2 writer
- Wireframe designer (not sketcher)
- Wrote integration tests

Martin

- Assisted wireframe sketching
- Created powerpoint presentations #1 and #2
- Decided on the tech stack
- Decided on website structure
- Developed Twitch API features
- Original writer of report #1

Additional Links to Our Work:

Github Repository: Frontend Kaisel Bot

Website: Project Kaisel

Figma Wireframe: Kaisel Bot