CSC468: Introduction to Cloud Computing

StatStrikeforce
Multiplayer Gaming Statistics Tracker and Prediction Generator

Team StatStrikeforce Members:

Katherine McCarthy

Maxwell Mendenhall

Ryan Sayre

Tobyn Sitar

Project Summary

Our project aims to revolutionize the statistics tracking service available for the popular competitive first-person shooter Rainbow Six Siege. While there are sites such asTracker.gg and U.gg, we plan to create a new application that provides statistics specifically for R6's unique mechanics. With our project, we want to revitalize the way users track their own or others' statistics. One of the major features we want to implement is comparison, in which you can have a side-by-side comparison of your stats compared to another player. With this feature, we plan to ramp up the competition between friends, or prove to an enemy that you are the better Rainbow 6 player. We also plan to have an accurate system, where you can see accurate statistics that automatically update the second you finish your match.

Furthermore, with our project, we want to implement a user-friendly hub for all your favorite topics regarding R6. We also plan to implement a machine learning algorithm to predict future performance statistics, based on the last batch of games played. Essentially, we want our model to factor in the most recent statistics into our algorithm to determine predictions for future matches. We want our project to be the best community website for Siege players, both competitive and casual.

We will do our best to utilize resources like CloudLab effectively and continuously assess the best options to continue making forward progress with the project. Our main source for assembling our project's components will be GitHub and Kubernetes, so we can remotely commit and add to the project from our separate spaces while working on the different components of our project collectively, and we plan on meeting regularly to work on the project together. With this, however, we need to consider some limitations. One of the most important limitations is the timeframe. With all the ideal features and components that we want to implement into our tracker, time will be critical to completion of our goals. The 5-month time limit given to us is ideal for a lot of work, but with other commitments and classes, providing the adequate time limit to complete our project will be difficult. Another factor that could potentially limit us is the lack of hardware. Fortunately, some of us have complete desktop computers fully equipped for the work, but differences in hardware among our team members may cause difficulties in the future. We will address project issues as they appear and work together to resolve them and continue progressing towards our final vision. We are excited to begin working on this project and hope we can achieve our goals at the course's end.

Chapter 1: Concept Vision

The vision of our project starts with the user, who will access our Web UI by URL. Our Web UI will have a welcome page where users input their own userID for the PC game

Valorant, made by Riot Games. The user can optionally create an account that will save relevant information into an account holders database. The account feature will be used for user convenience and to avoid the need for users to input redundant information. Once the userID is entered, a backend worker program will retrieve the necessary data from Riot Developer Portal, which contains the APIs for all user data and statistics. The necessary data includes a player's wins and losses, which can be filtered based on the timeframe of when the match was played, kills and deaths including the appropriate ratio, as well as statistics relating to objective completion performance.

The desired statistics and data will then be filtered and organized into a machine-readable format. Here we will apply a Machine-Learning Model which will use the data and statistics to perform a prediction on the outcome and statistics of a current match based on data trends. This prediction will use statistics that favor recency over a total average to be more accurate in real time, and ideally will also collect data about teammates and opponents as well to formulate predictions that are perfectly accurate and helpful. This can be achieved using the Riot Developer Portal, which provides historical statistics but also updates with data for active matches, which can be used to support our prediction model. We will brainstorm many different styles to convey our predictions, possibly through a percent chance to win, a prediction of a player's kills and deaths along with other relevant stats, and also maybe a grading system that tells a player how they have performed in a preset amount of time, providing letter grades on important game roles such as objective completion, support efficiency, gunfight success, and hero utility. The prediction model is something we are not completely certain of how it will function, but we are excited to work through its creation and implement features that we would find interesting and helpful as fans of the game.

Once the Machine Learning Model is applied to the data and the predictions are generated, the backend worker will retrieve the statistics and predictions from the model and display it to the user through the Web UI, which we want to be organized in an intuitive way. More than anything else, we want our project to provide insightful predictions that other statistic tracking services do not provide to make ours the most popular among the Valorant player base and give users an enjoyable and useful experience that players can share with their friends and expand the community to our site.

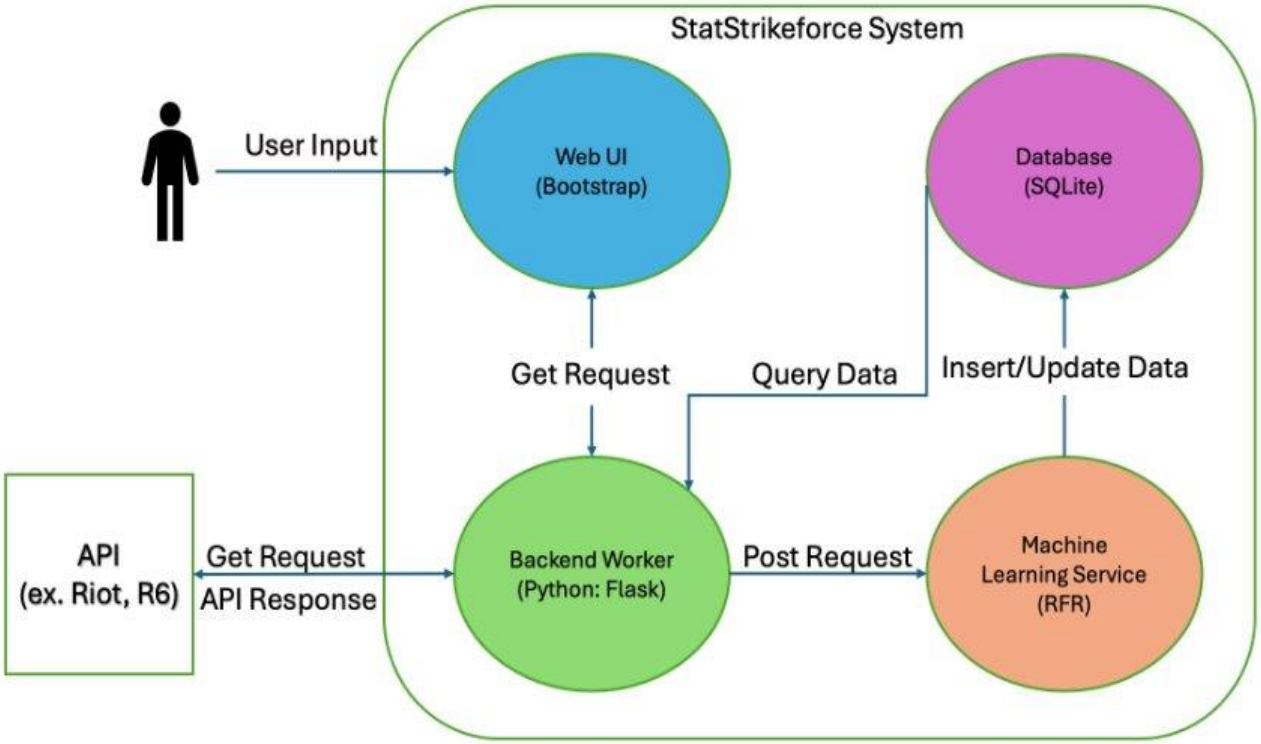

Figure 1: Design Document

Full Stack:

Front end: Web UI
Back-end: Worker, Machine Learning Model
Database: SQLite

Chapter 2: Design Plan

Web UI

For this program, we want a clean, user-friendly UI. Ideally, we want to utilize a tab system to direct users to their preferred method of tracking, whether that be recent games, aim statistics, the ability to see recently played with or most played with users, etc. Essentially, we want the program to run the same way as other statistics trackers, but stylized to Valorant's unique playstyle, and with more features not commonly found on other sites. Essentially, we would like to have separate tabs for community updates, player statistics reports, chatting, and feedback on how to improve the site as well. The website will be clean, with no outside influences (ADs), and have an aesthetic appearance to it that uses coloring and imagery that is present in the game, which will hopefully draw users to frequent the site. Furthermore, if time permits, we plan to take our application to the IOS and Android stores for users to take with them on the go.

We also want to implement a system for users to login, connect their RIOT account, and track their personal stats. Our priority is making sure the system works for everyone, not just a specified group of players that know their way around complex online gaming trackers. We plan to use a variety of Web UI frameworks to discover which ones are the most straightforward to create with regards to our desired features. We are not 100% certain how these will interact, but plan to have the program work much like other statistics trackers in its finished stage. Our front-end developer will brainstorm several templates for our site, and we will convene to discuss which model we believe fits our vision closest and proceed with development. The brainstorming phase will include a discussion of which frameworks we believe will be best to achieve our vision, whether that be HTML, CSS, or JavaScript. We also plan to use Bootstrap as a supplemental tool to help us with creating our website, which we will begin to work with and inform ourselves about its functionality. We will continue to work together to implement our desired features and offer our users a vast experience using the web interface.

Database

For our database, we are planning to use SQLite, as it is lightweight and already built into Python. We have done research on this, and SQLite fits all we want to build into our tracker, without difficult aspects of using MySQL. Because our team is most comfortable using Python, and SQLite has a python package built into it, we have decided that currently we will work with that to complete the database related functions of our project, which will mostly consist of storing userIDs and code-generated statistics such as K/D ratio, win rate, hero choice, score, etc. as well as our prediction calculations. SQLite also offers more data filtering commands than other database services, such as Select Distinct, Limit, Between, pattern matching, sub-querying, data transformation commands, and several unique table joining operations such as Self Join and Cross Join which will be very helpful for aggregating our necessary data and filtering that to a machine-readable format for our prediction model. This is much easier to implement for a project of our scale, which will give us more time to focus on the more challenging components of our project. SQLite also offers an online tutorial for inexperienced users to teach beginners how to effectively use the wide variety of commands that the platform offers, which we plan to take advantage of to help us complete this component of our project smoothly. The alternative to this would be using MySQL, but for a project of our scale we have currently deemed it not necessary, however we will re-evaluate our progress regularly and assess whether software changes must be made to meet the criteria of our vision.

Backend

For the backend development, we are leaning towards the use of Python, because of Python's wide spread of support for different databases and other software. It will also be

compatible with SQL based databases and have good integration with SQLite which we plan to use. Due to Python's versatility with our machine learning model and its API capabilities, as well as our overall comfort with the language, we decided that it is the language we will use to create our worker in the backend. Research into Riot Developer Portal's functionality has shown that Python is the preferred language for the community to request data from their APIs, so we believe that because of these factors it is our best course of action. Riot Developer Portal offers many different APIs each with a different purpose, for example VAL-MATCH-VI specialized in match data based on an ID or recency, VAL-RANKED-VI is specifically for ranked leaderboards and associated statistics, and finally VAL-STATUS-VI is used to find the status of a player account in the present. This data can be transferred using an index of .Json files, which we can use to retrieve relevant data from each API and format it into appropriate data tables. While working with API is something that is new to us, we have researched and found valuable tutorials from Riot Games users on how to effectively use their Developer Portal and its many creative features, which we will use to learn how we can apply the concepts of data requesting for the needs of our own project. We will monitor our success with this approach and assess its effectiveness regularly to ensure positive progress is made with the project.

Performance Prediction Through Machine Learning

This prediction aspect is the primary feature of our project vision. Competitive players often want to see how they will perform in a match and often look up the stats of other opponents to see how they will play. Making it easy for competitive players to use external software to predict this is the primary goal of our project. However, this prediction heavily relies on the fact that the user has a type of consistent performance throughout their games, otherwise results may be inaccurate.

Many ML models will be tested on the data before we decide on the final model. The models and not limited to are RFR (random forest regressor) and Basic Feedforward Neural Network. Stacking multiple models also to see the cost outcome will be considered.

A large part of machine learning and the part that usually takes up the most time is feature engineering. We will take the data given by the APIs and engineer the data to come up with more features that we think will be relevant in helping predictions. This step-in machine learning is usually a process of trial and error so an exact number of features we will not be able to provide until the end.

The language this feature will be built on top of is python. Using python there are many libraries that aid in the development of machine learning like Scikit-learn and TensorFlow. We can build an API on top of flask (python web framework) to expose the model endpoints for the other microservices. The features will then be transferred over web protocols and the data will be passed into the model via query parameters with the web server responding with the prediction. We will evaluate the success of this method regularly and update on our progress as the project continues.

Mathematics behind RFR

RFR is a supervised machine learning model which means it learns from mapping X inputs to Y outputs. The X inputs must be labeled data so the model can learn from it. Unsupervised machine learning is when the X inputs are not labeled so the X data cannot be distinguishable from other X data within the dataset.

The idea of how RFR works is finding the best possible mean squared error (MSE) for the features provided. It uses the ensemble method of finding the MSE which operates on developing different decision trees at training time and then outputting the MSE. During the process of growing each tree it randomly selects a subset of the features at each split point, this leads to increased diversity among the trees, which in the long term helps prevent overfitting for

data tremendously. It chooses the final value (or predicted value) by comparing the outcome of each tree, it then makes a prediction on the outcome data.

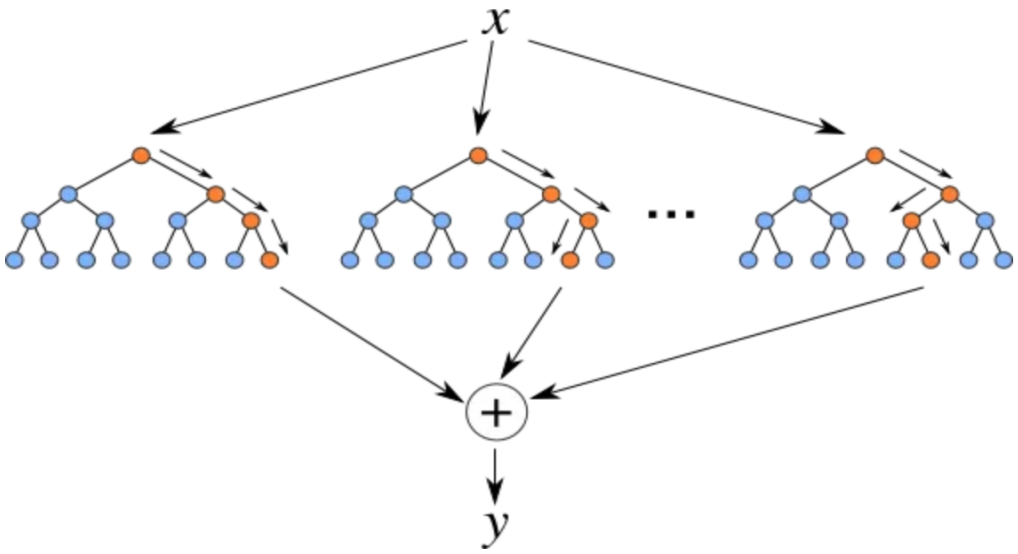

Figure 2: RFR Diagram via https://dsc-spidal.github.io/harp/docs/examples/rf/

Since RFR works on knowing the MSE, understanding how the formula works is crucial for the best output from the best features.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Where *n* is the number of data points.
- y_i is the value returned by the model.
- $\hat{y}_i$ is the actual value for data point *i*.

Deployment

The last aspect of our project is how to effectively deploy our project using a web server. We have experience creating default web servers using nginx and Apache, but this is something we will need to research and determine once our project components become more developed. To containerize the components of our project for portability, application efficiency, and security in the deployment phase, we plan to use Kubernetes. Containerization is a new concept for our team but one that we are excited to learn how to use and apply that knowledge to our project's needs. For now, we will focus on developing our components, but will take some time in the future to learn about the best strategies for web deployment to https, and how to configure our domain name and continuously run our web server for extended periods of time. We will utilize our Cloud Lab docker profiles to learn more about deployment and come to an informed decision on how to continue, along with providing updates along the way. We are pleased with the conceptualization of our project at this stage and are overly excited to begin technical development of the core components of StatStrikeforce.

Chapter 3: Intermediate Milestones

Current Progress

While working on our project since our conceptualization phase, we have made notable progress in several aspects. However, there has been one key developmental change to our

project concept we made during this time. After significant research into the APIs that many different multiplayer games offer, their strengths and weaknesses, and their overall compatibility with the scope, resources, and timeframe of our project, we have decided to pursue development of StatStrikeforce with the popular multiplayer shooting game Rainbow Six Siege (R6) as opposed to our original plan to use Valorant. This is due to many reasons, which we will discuss in detail in our Technical Challenges section. This realization also prompted us to change the concept of our idea to be not solely focused on Valorant, but on multiplayer gaming as a whole. We want to develop our project in a way that leaves room open to develop statistics tracking and prediction services for other games should we decide to pursue the project after the conclusion of the course, and because of this we changed our project goal statement to use more inclusive language to online gaming. However, this change does not affect our goal to create a working cloud-based application that displays a Web UI with recent statistics from an API and machine learning generated predictions for a multiplayer game, we have only decided that the best course of action to delivering our goal on time is to pursue using R6 as our targeted first game to be functional with StatStrikeforce.

With this change realized, we have taken action to complete the necessary components needed to create a working application to be deployed by our deliverable date, which has included development of a functional Web UI modeled in Bootstrap, a working backend that can successfully pull statistics from the R6 API and store them into a SQLite database that we have designed, research into our machine learning service and how it will function with what we would find interesting/helpful as players of R6, and lastly the development of Docker images and necessary files to deploy our project components into our Kubernetes cluster. Once we have a functional application deployed within our CloudLab experiment's Kubernetes cluster, we plan to test functionality and get feedback from users on how to improve for the final build of the project displayed in our last deliverable. At this point, we are confident that we can achieve this goal in the time we have remaining, and next we will discuss in more detail the progress of each part we have developed for StatStrikeforce.

Web UI

As of current, the Web UI is a functional website, with the choice to change tabs, access different information, and has a sleek, dynamic look utilizing Bootstrap. With the webpage, there are three tabs in the navigation bar: "Homebase" - a homepage for the user to navigate through the different features of the website, "About Us" - an informational page speaking on our group's passion for our website, the backgrounds of each member, and ways to contact each member regarding questions or concerns, and "Track Here" - which brings the user to a landing page in order to track a player's statistics. The footer of the webpage features a GitHub link, which takes the user directly to the project's repository. Furthermore, the website features a clean, slick, uncomplicated design, allowing readers to easily find what they need in an organized manner. The website features a background image of Rainbow Six Siege, to keep users mindful of the game they want to track, with contrasting blue and orange accents to peak the user's interest.

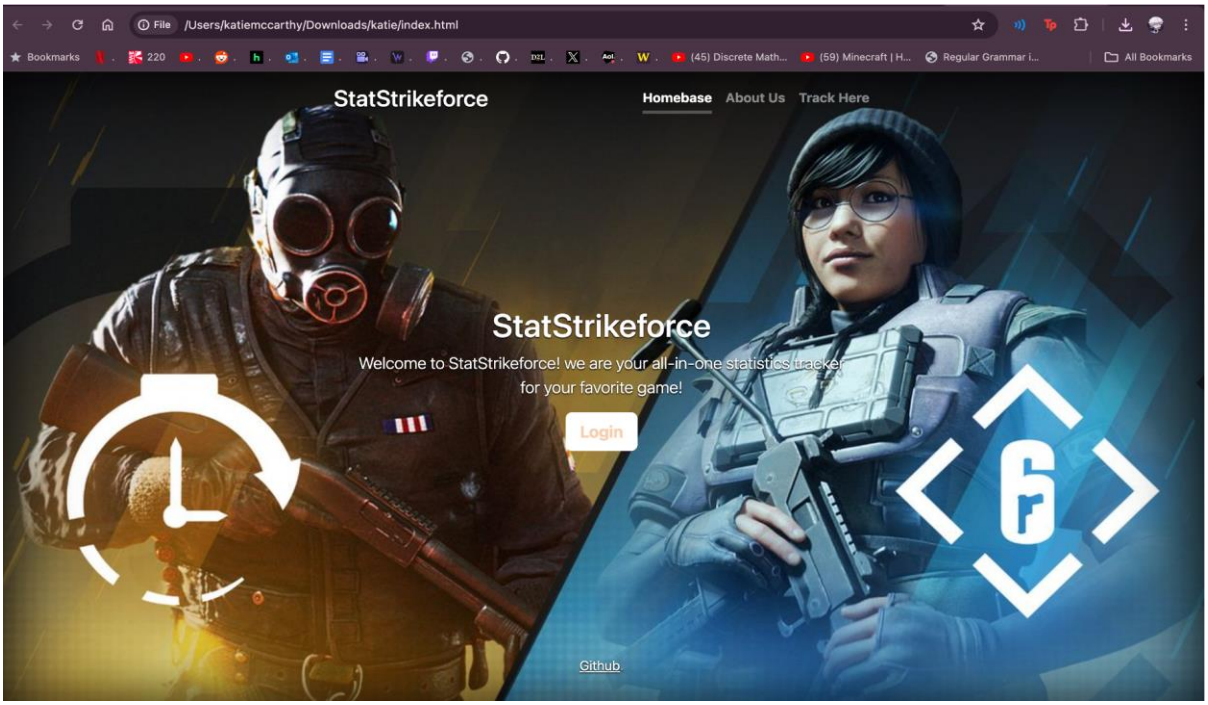

Figure 3: Web UI Design

Bootstrap

- Cover template with a functional Navigation Bar, three tabs, and buttons
- HTML file including the template, with all its references to external files
- CSS file that includes the aesthetics of the website including:
    - Font
    - Color
    - Sizing
    - Background image
    - Buttons
    - Headers
    - Scrolling bar
    - Menu bar
    - Footer
- A java file that includes references to menu swap pages dynamically, to keep users on the same webpage without refreshing, all while changing the content the user is viewing.

Backend

The application serves as a backend service that provides endpoints for web clients. It uses environmental variables for configuration and includes a custom ‘Database’ class for database operations. The application primarily interacts with the Ubisoft API to fetch player statistics for Rainbow Six Siege.

Dependencies

- Flask: A lightweight WSGI web application framework used to handle web requests
- Asyncio: A python library to write concurrent code using the async/await syntax.
- Siegeapi: An unofficial API wrapper for interacting with Ubisoft's Rainbow six siege API.
- Dotenv: A python package for reading key-value pairs from a ‘.env’ file and setting them as environment variables.

- Os: A module for interacting with the operating system used here to access environment variables.

Configuration

The application loads its configuration from environment variables, which are set from a .env file using load_dotenv(). The configuration includes:

- SECRET_KEY: A secret key used by Flask for securely signing the session cookie.
- DATABASE: The database connection string or path, used by the custom Database class for database operations.

Database Interaction

The Database class, imported from database, is a custom utility for connecting to and querying a database. It is instantiated with the Flask app and a database connection string or path. The instance is then stored in app.extensions for easy access throughout the application.

Application Routes

Get Rainbow Stats (/get_rainbow_stats)

- Method: GET
- Parameters: uid (User ID)
- Function: get_rainbow_stats
- Description: This endpoint retrieves the Rainbow Six Siege player statistics for a given user ID. It demonstrates the use of asyncio to run asynchronous code within a Flask route. The user ID is passed to the sample async function, which fetches the player data from the Siege API, formats it, and returns it as JSON.

Asynchronous Data Retrieval

The sample async function showcases asynchronous interaction with an external API. It performs the following steps:

- Authenticate with the SiegeAPI using credentials obtained from environment variables.
- Fetch the player object using the provided user ID.
- Load additional player statistics, such as playtime.
- Format the data into a dictionary.
- Close the API session.
- Return the player data or handle exceptions.

Running the Backend Application/Service

For production deployment of Flask applications, leveraging a WSGI-compatible server such as Gunicorn is essential due to its capability to efficiently manage and scale concurrent requests. The Flask development server lacks the necessary performance optimizations, security enhancements, and scalability features required for handling production-level traffic. Gunicorn excels in these areas with its pre-fork worker model, which spawns multiple worker processes to handle incoming requests in parallel, effectively distributing the load and maximizing resource utilization. Additionally, integrating Gunicorn with a reverse proxy like Nginx further augments the application's ability to manage static assets, provide SSL termination, and balance loads, thereby ensuring a secure, scalable, and high-performance production environment.

Machine Learning Service

From the time of our first deliverable until now, most information we have provided about our machine learning service has not changed. In our GitHub repository, we have created a python script that runs the machine learning application, which accepts data from the Siege API to be trained so a prediction can be generated, but we have not determined at this point how we will configure the service to provide meaningful predictions that players would find interesting and helpful in playing R6 competitively. Up to this point we have focused our efforts on developing our backend worker because the completion of that was essential to beginning the development of the machine learning service. Moving forward, we will dedicate more time and effort to developing this service, as it is a crucial aspect of our project vision, and we are excited to begin planning how it will function and start development soon.

Kubernetes Deployment

Although we have mainly focused on component development locally up to this point, we have begun the process of developing the environment in which we will deploy the components of StatStrikeforce, and have also taken some pre-emptive action towards how we plan to deploy our project in Kubernetes. To start, we have completely configured our Kubernetes profile within our project GitHub repository and have tested it many times to ensure that it is completely functional. We have also configured the Kubernetes dashboard within our experiment to view our running containers and services there, which is much easier and will help to save us time debugging later on. We have configured our GitHub repository such that each part of our project has its own branch, which helps with version control and ensuring that similar files for the various parts of our project do not overlap or overwrite one another.

At this point, we have successfully written a Dockerfile for our backend worker, which will create the image needed for our backend worker to be deployed in our Kubernetes cluster. We have pushed this Docker image to DockerHub for record and convenience, and we can update the image as necessary as our backend code continues to develop. We have now begun the process of creating our deployment and service yaml files to successfully run the backend worker in our cluster, after which we can deploy the backend into Kubernetes and use the Dashboard to test its functionality. Now that we have made substantial progress with our Web UI as well, we plan to write a dockerfile for the UI, create the image for it in DockerHub, and begin the process of deploying our first version to our Kubernetes cluster to test the executability of GET requests between our Web UI and backend. Our plan for this deliverable was to focus on the development of the source code of our components, and now that we have a strong first version for the backend and Web UI, we can now begin with the Cloud deployment aspect of our project, which we are excited to work through and have completed all of the necessary prep work to complete this phase by our final deliverable date.

Technical Challenges

While we have made substantial progress in developing our vision for StatStrikeforce into a tangible application, there are many challenges that we have had to overcome to reach this point, and still much more to do in terms of research and development of our components. Here, we will explain in detail the technical challenges of each of our components, and our current plan for action on how to resolve them and make continuous progress with development of our final vision for StatStrikeforce.

Web UI

One of the main technical challenges with the Web UI was the concept of taking the data from the backend and formatting it into the front-end. Our plan is to implement the concept of JSON, taking the key and value pairs from the backend worker and converting it to the website in an easy-to-read and aesthetic way for the user. Using this strategy, it has been a struggle to achieve communication with the front-end, as it does not always recognize the key and value

pairs being requested from the backend. Furthermore, another technical challenge we faced with our UI is the ability to have the website recognize typing patterns, and filter usernames based on the letters input by the user. Ideally, the user will begin typing out the username of the player they are trying to track, and it will filter users by similar letter patterns. For example, typing “csc” into the search bar will show users that have “csc” in their username. However, in its current state, the UI does not recognize this pattern, and you must type the full username to receive the player’s statistics. This is a novelty feature that we hope to research more about soon and determine a strategy on how to implement this into our final build for the Web UI.

Backend

The largest challenge in our backend development to this point has been deciding the most suitable game API to use for our project. Initially, we decided to use Valorant API, due to its inclusion of multiple APIs for all distinct types of data relating to the player. We thought that this would be helpful for aggregating many different player statistics, such as ranked data and specific match stats separately, but there were many unforeseen difficulties with this method. First, Riot Developer Portal, like many online game APIs, requires each user to develop an API key as part of their profile to authenticate the user’s identity when accessing data from each API. However, Riot Developer Portal is different than others because it requires each user to regenerate their API key every 24 hours, which cannot be done using a randomizer and must be done on Riot Developer Portal. This is very inconvenient for our project because it would require each user to visit Riot Developer Portal daily to regenerate their temporary key, which could draw people away from using our application, and this is would also be a constantly changing parameter to manage in our backend which would greatly complicate its functionality. It is also important to note that each user’s API key is a part of the URL that needs to be used to access Riot APIs, adding another parameter that would be needed for each player to receive their statistics from the different APIs.

Along with this, each API uses different parameters to receive the data from Riot Developer Portal. For example, the player data in VAL-CONTENT API requires only a player id, but to get the seasonal data for a user from VAL-RANKED API it requires a locale and actID to be input from the user. Furthermore, receiving statistics from a specific match from VAL-MATCH API requires a matchID, which can only be received by using the VAL-CONTENT API to receive a match summary, and then choosing which match to get more statistics from there. This necessity to use multiple different APIs in a chain to aggregate certain statistics for the more specific APIs would add a significant amount of computing time and power for each data request and would also make it so that users of StatStrikeforce would need to input more information than just their userID. By using Valorant API, we would have to program the API keys, parameters, and URLs for fetching data from each API differently, which would become extremely technically complicated and would add critical time towards the development of our backend. With all this, it became clear that using Riot APIs would complicate our backend process beyond the scope and timeline we have set for completing this project.

The other most crucial reason that led us to the decision to switch to a different game API was the fact that Riot Developer Portal requires registration of a product to access the API endpoints, which would be necessary for our project. While we did plan to register a product that uses Riot API, this would require a placeholder production URL to use their APIs in our project, something that we did not have at the time and decided that we could not wait until the point where we would have a product URL to begin working with the APIs and writing our backend code. These limitations of Riot Developer Portal which we learned after our initial conceptualization of our project are what led us to consider working with a different game, which is when we started researching other game APIs with compatibility in mind rather than only choosing a game that we all enjoyed, which was largely the motivation for choosing Valorant and Riot API when beginning our project. This research into many different APIs led us to discovering R6 API, which we have found is much easier to use and does not have many of the

deal breaking limitations that Riot APIs have. Our progress working with R6 API has been much more productive than before, but not without its own set of challenges that we have addressed as we continue with development.

The key challenge in our backend development with R6 stems from the handling of data returned by the unofficial Siege API wrapper. While the data superficially resembles a Python dictionary, it's a specialized API Object proprietary to the Siege API. This discrepancy complicates data manipulation, as the object doesn't natively support conversion into a mutable, Pythonic format. To address this, we need to manually extract and convert the data into a standard Python data structure, such as a dictionary. This involves implementing custom logic to navigate the API Object's structure, extract its contents, and map them into more flexible, Python-compatible formats. This manual conversion introduces added complexity and potential error points, especially in supporting alignment with the Siege API wrapper's evolving structure. It also affects backend performance, highlighting the importance of developing an efficient and adaptable data processing strategy to ensure the backend's functionality and reliability.

Kubernetes Deployment

The main challenge in the Cloud deployment aspect of our project has been taking what we have developed in our source code and creating Dockerfiles from that. We wanted the first version of our components to have something meaningful to highlight or test in our Kubernetes cluster, but in our current state this was not practical. We have made large strides just only recently in our source code development, which has not left enough time to gain an understanding of the challenges that await us in deploying our projects into the cloud-based model, but so far, we have been successful in creating our first version image of our backend and writing the associated yaml files as well. From working on the assignments and researching both creating dockerfiles and yaml files, we have a strong understanding how to pull images from DockerHub into nodes, how to pull deployment updates from GitHub into the namespace we have made in our experiment and apply them to their respective components.

A major consideration to be made regarding how we will deploy and support our application in CloudLab is time and on-site maintenance. The default timing for our CLoudLab experiment is 16 hours, with extensions up to six days without valid reasoning for the extension. However, we do not know how long our experiment can be running and ready to use before time extensions will no longer be granted, and our project will run out. Also, in the time we have spent working with CloudLab we have noticed that is not unusual for CloudLab resources to suspend for periods of time, due to maintenance, outages, etc. The reason I bring these considerations to light is because our project revolved around continuously updating our database with user statistics, such that a user can pull data straight from our database without the need to execute an API request if recent statistics have been generated for a user already. CloudLab resources being suspended either due to time or outages would effectively wipe the data from our database each time, which could create issues with requesting redundant data from R6 API and increase the amount of API requests from our users. This would use more time and resources than we have planned for our application, with our main goals being ease of use and a quick response cycle through our project’s different components. These considerations could also have a negative effect on our CI/CD plan to test our application in Kubernetes while developing future builds and modifying our source code. The ability to have our application deployed continuously and uninterrupted is particularly important given the end goal for our project, so this is something we will need to research and derive possible backup plans for when we arrive at that stage in our project timeline. For now, we will dedicate a larger focus towards cloud deployment now that we have more developed component source code, and we will update on our progress and technical challenges of this aspect in our final deliverable when we have spent more time working on this aspect of the project.

Future Goals

Through working on this project in more depth, we have learned much valuable information about working with APIs, web development, database architecture, and cloud deployment. Learning these new skills has come with many obstacles to overcome and while we have made decisions on how to adapt our project, our overall vision for StatStrikeforce is unchanged. We want to create an application that gives players real user data for Rainbow Six, with different statistics such as Kill/Death ratio, headshot percentage, Win/Loss ratio, etc. as well as meaningful machine learning predictions of future match performance and outcome to give players a competitive advantage. We want StatStrikeforce to be aesthetically pleasing so users keep coming back and telling their friends about the application, but we also want it to be functional, fast, accurate, and fun to use.

We are extremely confident in the ability to achieve this vision, as the project is moving forward rapidly, with willing team members and constant communication to progress on the project. We know that some of the more complex features, such as community notes pages, compatibility with other games, chatting systems, etc. that we would like to implement will only come with time beyond what we have for the course, but being able to show current statistics for Rainbow Six Siege and a outcome prediction for future games is our main focus at this point in our project timeline. We have designed our project in a scalable way where the ability to add features later after the end of the course is something that is achievable, and we have agreed on and are looking forward to. We all have an immense passion for gaming that we want to share with others through our unique vision, and we are working hard to develop our vision for StatStrikeforce into a functional application that utilizes the core concepts of cloud computing that we have learned throughout this course.