CSC468: Introduction to Cloud Computing

StatStrikeforce
Multiplayer Gaming Statistics Tracker and Prediction Generator

Team StatStrikeforce Members:

Katherine McCarthy

Maxwell Mendenhall

Ryan Sayre

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.

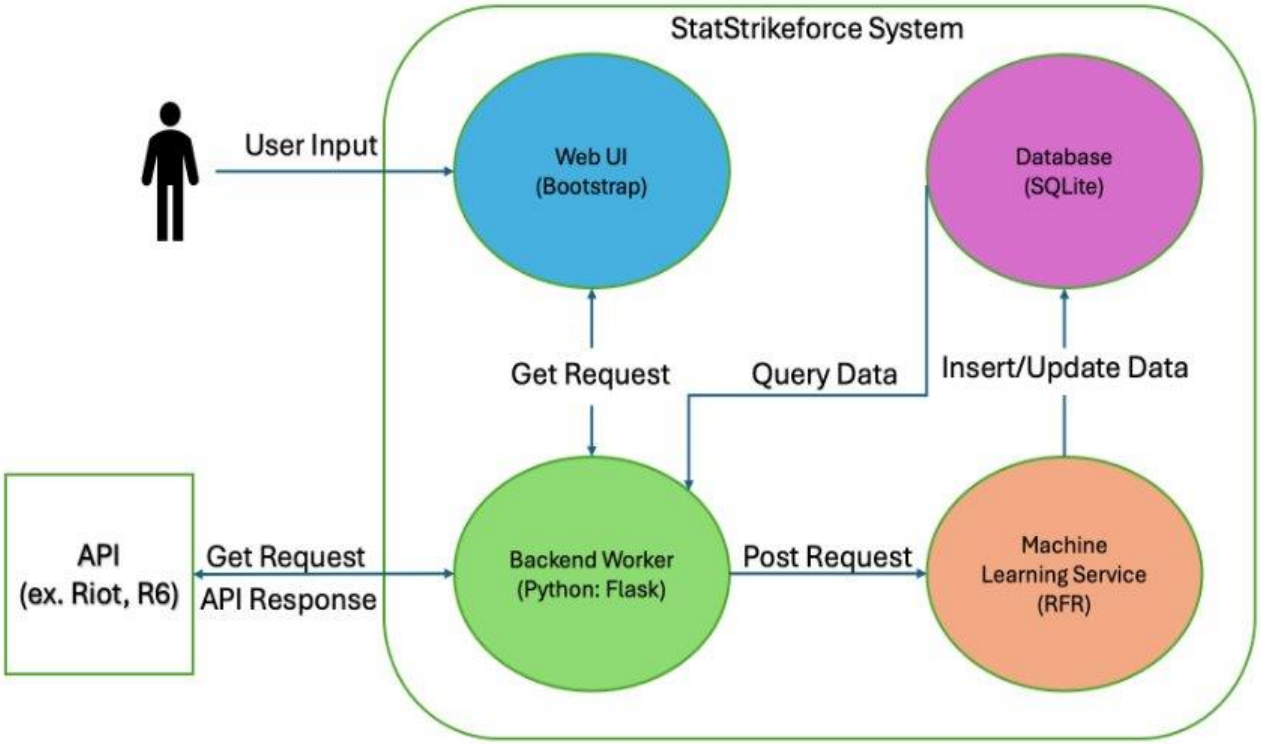

Figure 1: Design Document

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.

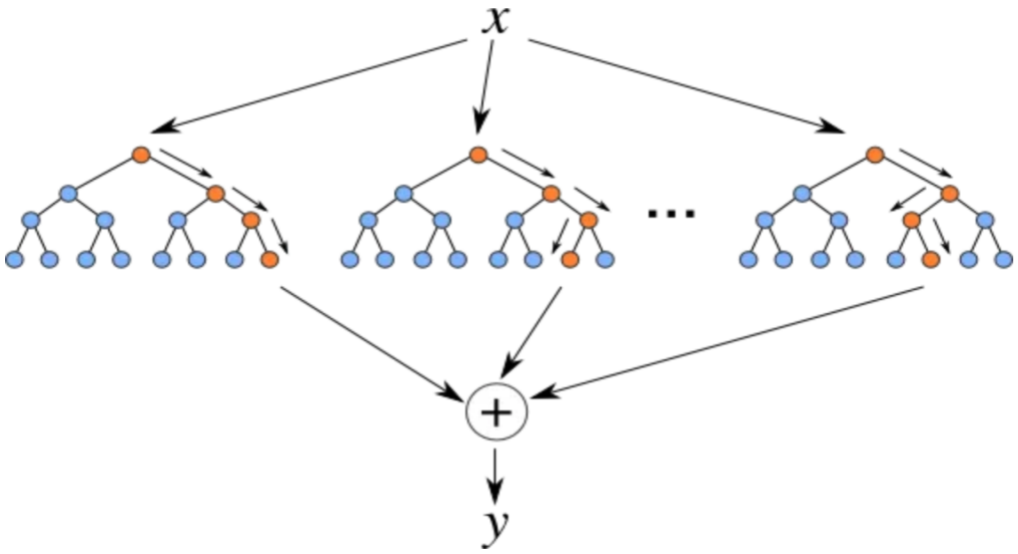

Figure 2: RFR Diagram via https://dsc-spidal.github.io/harp/docs/examples/rf/

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.

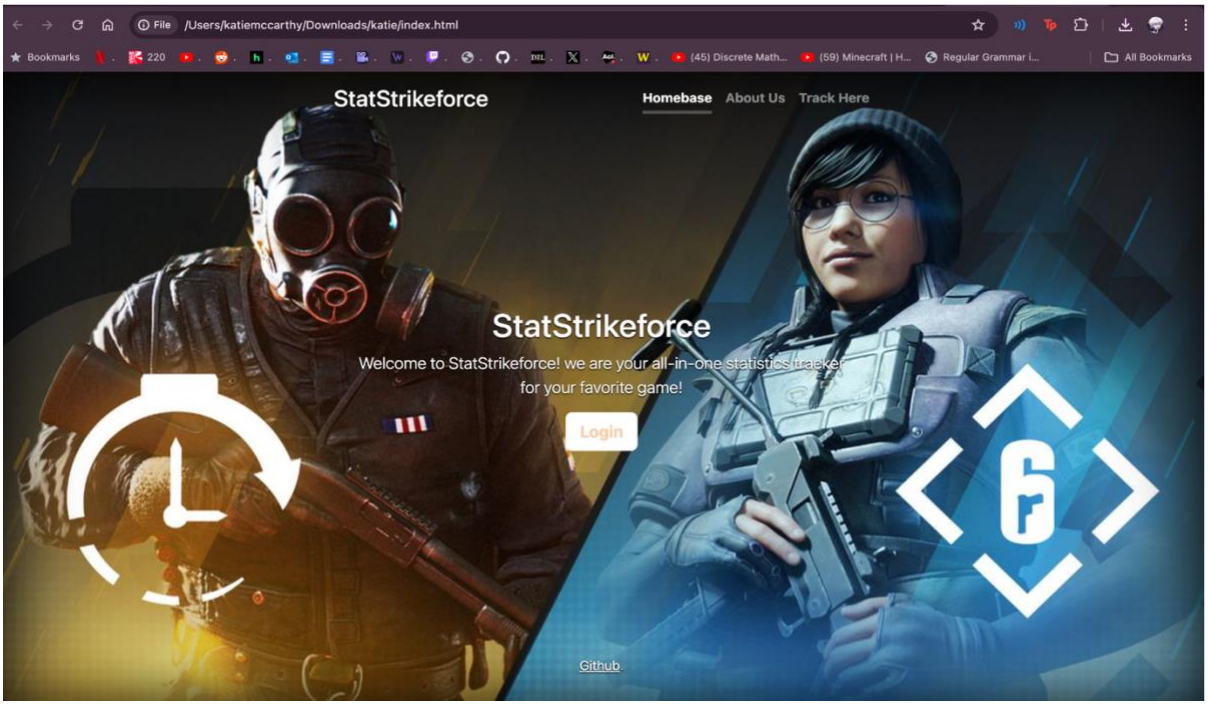

Figure 3: Web UI Design

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.

Cloud 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 our 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.

Chapter 4: Final Results

By working on creating our vision for StatStrikeforce for the 15-week duration of the course, we have created an application that has many of the features and functionalities we wanted from the early stages of this project. The process of developing our code for the application components has taught us so many practical skills, such as web-ui design, get and posts requests, database structure, and introducing us into how to integrate machine learning into a program to generate meaningful data for users. We want to share the final state of each of the components of our project and discuss some of the missed milestones for the project that we were not able to complete due to technical challenges or time limitations.

To begin, we wanted to implement a feature to compare two players' statistics against each other, but realized with timing that this was not possible. Towards the end of our project's development, we really wanted to prioritize cloud deployment of our project's basic concept, as we believed this to be the most crucial aspect of the project. After this course's end, our team agreed to work on this project and possibly implement this feature later. We also wanted StatStrikeforce to be a hub for all games, but again with timing, we only had the ability to implement Rainbow Six Siege statistics. In the future, we would like to add some of our other favorite games to track statistics for, such as Overwatch 2, Apex Legends, or maybe even finish our original plan to implement Valorant.

Web UI

For the Web UI, we wanted to implement better styling of our website. Instead of having a single image, we wanted to be able to implement different images and styling based on the page the user was viewing. Another implementation we wanted to add was utilizing more of the features of Bootstrap. Bootstrap had a plethora of tools and features that would add to the appeal of the website, and we plan to use these tools in future developments of StatStrikeforce. Furthermore, our group wanted to implement a feature to switch from light to dark mode based on the user's preferences. While this is an interesting feature, due to timing purposes, it was not attainable, as we wanted to make sure the frontend and backend could communicate properly before adding aesthetic features. Another aspect of the UI we wanted to add was a community page, showing aspects such as patch notes, popular social media posts, and having it so that the user could see interesting or popular articles about the game. We thought that this would be interesting to add, and would help keep the website population up, as users would then be using the website for longer periods of time and returning more frequently. Also, adding a community page would allow users to find all the knowledge they wanted to know in one place, without having them use external sites to obtain this information, so the convenience of this would draw more appeal to StatStrikeforce's website.

Presenting the final state of the Web-UI, we are proud to show that the UI shows statistics for a player, talks to the backend, and has a fully functional website. Our website, which utilizes a navigation bar, buttons, and aesthetic styling, can recognize statistics from the backend and display them readably for the user. When the web page is loaded, the user is brought to a Homebase page where they can sign up for an account or log back in if they are a returning user. This would be more convenient for users who do not want to type in their player

id each time, but the user can jump straight to the Track Here page on the navigation bar to bypass this.

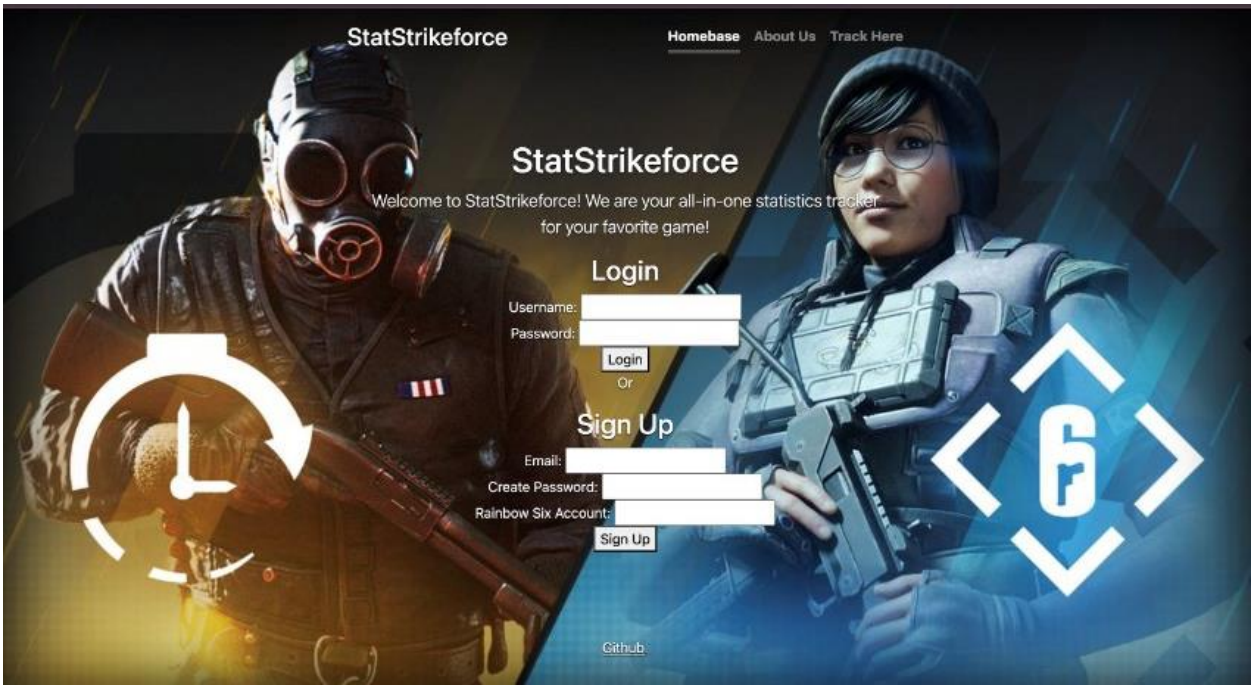

Figure 4: Homebase Page

Once arriving at the Track Here page, the user will be prompted to type in their R6 username to receive their statistics from the API. Due to our own security reasons, we stored our R6 passwords for any accounts we wanted to test in our database beforehand and did not require a password on the Track Stats page, which we will cover in further detail in the Database portion of this chapter. Once the R6 id is entered into the database, our backend will retrieve the statistics from the API, and populate the database where they can be pulled nearly instantaneously once the stats are in the database.

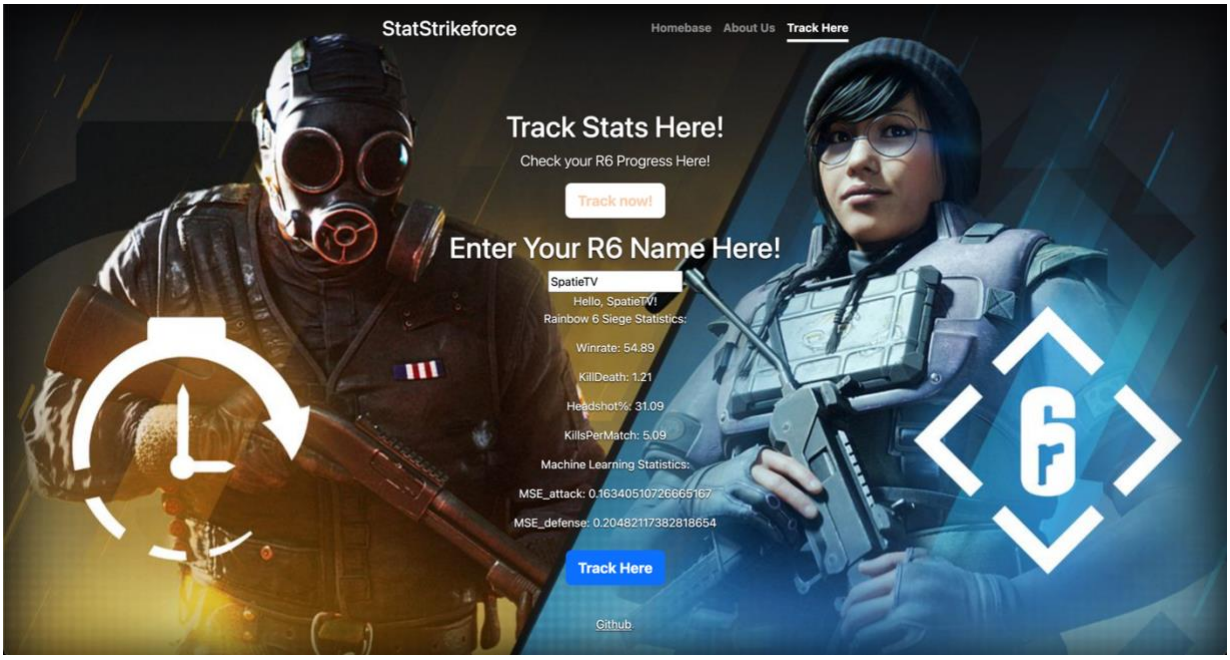

Figure 5: Track Stats Page

While this screenshot shows the success of the implementation, the process of integration was the most difficult aspect of developing the Web-UI. We spent countless hours troubleshooting the issue, and finally were able to display some of statistics from the backend. The main issue with displaying statistics from the database to the Web-UI was the size of the data response that we received from the R6 API.

.4445,"202":0.4231,"203":0.3333,"204":0.36,"205":0.2917,"206":0.32,"207":0.3929,"208":0.3704,"209":0.3846,"21":0.5588,"210":0.4231,"211":0.4667,"212":0.4827,"213":0.5333,"214":0.5715,"22":0.5882,"23":0.5814,"24":0.575,"25":0.5882,"26":0.6111,"27":0.6571,"28":0.6487,"29":0.6571,"3":0.5143,"30":0.6667,"31":0.7,"32":0.6875,"33":0.7143,"34":0.7241,"35":0.6786,"36":0.7,"37":0.7,"38":0.6957,"39":0.64,"4":0.5428,"40":0.6538,"41":0.6154,"42":0.6207,"43":0.5926,"44":0.6,"45":0.5357,"46":0.5517,"47":0.5384,"48":0.5,"49":0.5,"5":0.5278,"50":0.5172,"51":0.5357,"52":0.5417,"53":0.5555,"54":0.5926,"55":0.5926,"56":0.6207,"57":0.6154,"58":0.6667,"59":0.6897,"6":0.5789,"60":0.6897,"61":0.7576,"62":0.7742,"63":0.7576,"64":0.7576,"65":0.7714,"66":0.7187,"67":0.6765,"68":0.6969,"69":0.6774,"7":0.5789,"70":0.6667,"71":0.5946,"72":0.5313,"73":0.5428,"74":0.5,"75":0.5172,"76":0.5,"77":0.5,"78":0.4516,"79":0.4412,"8":0.6286,"80":0.4516,"81":0.4138,"82":0.4138,"83":0.4074,"84":0.5,"85":0.5384,"86":0.5218,"87":0.5238,"88":0.5715,"89":0.5555,"9":0.6471,"90":0.5,"91":0.5384,"92":0.5833,"93":0.5652,"94":0.5652,"95":0.5455,"96":0.5769,"97":0.5555,"98":0.6207,"99":0.6333},"KD":
{"1":1.0689,"10":1.1667,"100":1.1379,"101":0.9375,"102":1.0,"103":0.9667,"104":0.8965,"105":0.8276,"106":0.8621,"107":0.871,"108":0.9678,"109":0.9394,"11":1.1,"110":0.9143,"111":0.8235,"112":0.8571,"113":0.7778,"114":0.6945,"115":0.6842,"116":0.8421,"117":0.9459,"118":1.0,"119":1.0833,"12":1.1,"120":1.0572,"121":1.0286,"122":1.0,"123":0.8824,"124":0.9697,"125":1.0,"126":1.0303,"127":1.1613,"128":1.0645,"129":0.8965,"13":0.9678,"130":0.931,"131":0.8333,"132":0.7097,"133":0.6969,"134":0.7273,"135":0.75,"136":0.8065,"137":0.7097,"138":0.7,"139":0.7857,"14":1.0,"140":0.7,"141":0.75,"142":0.75,"143":0.75,"144":0.9643,"145":0.9259,"146":1.1482,"147":1.0715,"148":0.9667,"149":1.0,"15":0.9333,"150":1.1072,"151":1.1072,"152":1.0645,"153":1.1,"154":1.0909,"155":1.0882,"156":1.0294,"157":0.9394,"158":1.0625,"159":1.0,"16":0.9355,"160":1.0645,"161":1.1,"162":1.129,"163":1.1333,"164":1.1667,"165":1.2143,"166":1.2069,"167":1.1786,"168":1.2,"169":1.25,"17":0.9355,"170":1.1724,"171":1.0645,"172":1.0968,"173":0.9375,"174":1.0313,"175":1.0938,"176":1.0,"177":0.9394,"178":0.875,"179":0.7813,"18":1.0303,"180":0.8621,"181":0.8,"182":0.7879,"183":0.7647,"184":0.7576,"185":0.6176,"186":0.7813,"187":0.8125,"188":0.8438,"189":0.9355,"19":1.0313,"190":0.9355,"191":1.0909,"192":1.1515,"193":1.0556,"194":1.0606,"195":1.0303,"196":0.9375,"197":1.0968,"198":1.1,"199":1.0667,"2":1.1724,"20":1.125,"200":0.9063,"201":0.9,"202":0.8387,"203":0.9678,"204":0.8333,"205":0.8276,"206":0.8333,"207":0.9655,"208":0.931,"209":0.8667,"21":1.0625,"210":0.8387,"211":0.9678,"212":0.9063,"213":1.0345,"214":0.8485,"22":1.0968,"23":1.303,"24":1.2121,"25":1.0303,"26":1.0909,"27":1.0294,"28":1.1212,"29":1.1667,"3":1.1667,"30":1.1613,"31":1.0,"32":1.1034,"33":1.0,"34":1.0358,"35":1.0,"36":0.7143,"37":0.7408,"38":0.8846,"39":0.9616,"4":1.2069,"40":1.0,"41":0.963,"42":0.9667,"43":0.9,"44":1.0,"45":0.9032,"46":0.8788,"47":0.7647,"48":0.8,"49":0.8,"5":1.2857,"50":0.8286,"51":0.7567,"52":0.6667,"53":0.75,"54":0.7297,"55":0.7714,"56":0.8529,"57":0.7429,"58":0.7273,"59":0.8788,"6":1.2258,"60":0.9667,"61":1.0645,"62":1.0,"63":1.0645,"64":1.1,"65":1.0938,"66":1.0322,"67":1.0968,"68":1.0645,"69":1.0,"7":1.3103,"70":1.0645,"71":1.1212,"72":0.9697,"73":0.9722,"74":0.9375,"75":0.9063,"76":0.9375,"77":0.9697,"78":0.9688,"79":1.0968,"8":1.2069,"80":1.0333,"81":1.0,"82":0.9667,"83":0.931,"84":0.8276,"85":0.963,"86":0.8846,"87":0.7241,"88":1.0769,"89":0.9643,"9":1.0968,"90":0.8889,"91":0.9286,"92":0.8,"93":0.7667,"94":0.7931,"95":0.7857,"96":0.9286,"97":1.0385,"98":1.0,"99":1.0345},"KPR":
{"1":0.7381,"10":0.8974,"100":0.8462,"101":0.75,"102":0.7632,"103":0.7632,"104":0.7027,"105":0.6487,"106":0.6945,"107":0.7105,"108":0.7692,"109":0.775,"11":0.8462,"110":0.8,"111":0.7368,"112":0.7692,"113":0.7,"114":0.6098,"115":0.6046,"116":0.7273,"117":0.7955,"118":0.8182,"119":0.8667,"12":0.8684,"120":0.8605,"121":0.8372,"122":0.7857,"123":0.6977,"124":0.7442,"125":0.7907,"126":0.8095,"127":0.8781,"128":0.825,"129":0.7027,"13":0.7895,"130":0.7105,"131":0.6579,"132":0.5789,"133":0.575,"134":0.6,"135":0.6,"136":0.641,"137":0.5789,"138":0.5526,"139":0.6111,"14":0.7895,"140":0.5833,"141":0.6176,"142":0.6176,"143":0.6364,"144":0.7941,"145":0.7353,"146":0.8611,"147":0.8333,"148":0.8056,"149":0.8235,"15":0.7368,"150":0.8611,"151":0.8857,"152":0.8462,"153":0.8684,"154":0.8571,"155":0.8605,"156":0.7955,"157":0.7209,"158":0.7907,"159":0.7561,"16":0.7436,"160":0.7675,"161":0.7675,"162":0.7955,"163":0.8095,"164":0.7955,"165":0.7907,"166":0.7955,"167":0.7857,"168":0.8,"169":0.8333,"17":0.7436,"170":0.8095,"171":0.7857,"172":0.8095,"173":0.7317,"174":0.8049,"175":0.8333,"176":0.7907,"177":0.7561,"178":0.7368,"179":0.6579,"18":0.8095,"180":0.6945,"181":0.6667,"182":0.6842,"183":0.6667,"184":0.641,"185":0.5385,"186":0.6579,"187":0.6667,"188":0.7105,"189":0.7632,"19":0.8049,"190":0.7838,"191":0.8781,"192":0.9269,"193":0.8837,"194":0.8537,"195":0.8293,"196":0.85,"197":0.8718,"198":0.8684,"199":0.8421,"2":0.7907,"20":0.8781,"200":0.7632,"201":0.7297,"202":0.7027,"203":0.7692,"204":0.6945,"205":0.6667,"206":0.6757,"207":0.7778,"208":0.75,"209":0.7222,"21":0.8293,"210":0.6842,"211":0.75,"212":0.6905,"213":0.7317,"214":0.6364,"22":0.8293,"23":0.9773,"24":0.9302,"25":0.8095,"26":0.8571,"27":0.8333,"28":0.9025,"29":0.921,"3":0.8139,"30":0.9,"31":0.7895,"32":0.8421,"33":0.7778,"34":0.7838,"35":0.7778,"36":0.5882,"37":0.6061,"38":0.6765,"39":0.7353,"4":0.8333,"40":0.7429,"41":0.7222,"42":0.7436,"43":0.7297,"44":0.7895,"45":0.7179,"46":0.7073,"47":0.65,"48":0.6829,"49":0.6666,"5":0.9,"50":0.6905,"51":0.6666,"52":0.5854,"53":0.675,"54":0.6585,"55":0.675,"56":0.725,"57":0.65,"58":0.6487,"59":0.7632,"6":0.9269,"60":0.8056,"61":0.8684,"62":0.8378,"63":0.8684,"64":0.8684,"65":0.8537,"66":0.7619,"67":0.8095,"68":0.7675,"69":0.7209,"7":0.95,"70":0.75,"71":0.7432,"72":0.6956,"73":0.7143,"74":0.6667,"75":0.6304,"76":0.6522,"77":0.6956,"78":0.7045,"79":0.8095,"8":0.8974,"80":0.775,"81":0.7838,"82":0.7838,"83":0.75,"84":0.6857,"85":0.7429,"86":0.6969,"87":0.6,"88":0.8485,"89":0.7941,"9":0.8718,"90":0.7059,"91":0.7429,"92":0.6667,"93":0.6216,"94":0.6216,"95":0.6111,"96":0.7222,"97":0.75,"98":0.7632,"99":0.7692},"Name":"CNDRD","WL":
{"1":0.625,"10":0.857,"100":2.2499,"101":1.6001,"102":2.2499,"103":1.6001,"104":1.1668,"105":0.857,"106":0.857,"107":0.625,"108":0.857,"109":0.857,"11":0.857,"110":0.857,"111":0.857,"112":0.625,"113":0.625,"114":0.625,"115":0.4445,"116":0.625,"117":0.857,"118":1.1668,"119":0.857,"12":0.857,"120":0.857,"121":0.857,"122":1.1668,"123":1.1668,"124":1.1668,"125":1.1668,"126":1.6001,"127":2.2499,"128":2.2499,"129":1.6001,"13":0.857,"130":1.6001,"131":1.6001,"132":1.6001,"133":1.6001,"134":1.1668,"135":1.1668,"136":1.1668,"137":0.857,"138":1.1668,"139":1.1668,"14":1.1668,"140":0.857,"141":0.857,"142":1.1668,"143":0.857,"144":0.625,"145":0.625,"146":0.857,"147":0.857,"148":0.625,"149":0.4445,"15":1.1668,"150":0.625,"151":0.4445,"152":0.3001,"153":0.3001,"154":0.3001,"155":0.1818,"156":0.3001,"157":0.4445,"158":0.4445,"159":0.3001,"16":1.1668,"160":0.4445,"161":0.4445,"162":0.4445,"163":0.3001,"164":0.3001,"165":0.4445,"166":0.625,"167":0.625,"168":0.857,"169":0.857,"17":1.1668,"170":0.625,"171":0.857,"172":1.1668,"173":1.1668,"174":1.1668,"175":1.1668,"176":1.1668,"177":1.1668,"178":0.857,"179":0.625,"18":0.857,"180":0.857,"181":0.625,"182":0.625,"183":0.625,"184":0.4445,"185":0.3001,"186":0.3001,"187":0.4445,"188":0.4445,"189":0.625,"19":0.857,"190":0.857,"191":0.857,"192":0.857,"193":0.625,"194":0.857,"195":0.857,"196":0.857,"197":0.857,"198":1.1668,"199":1.1668,"2":0.857,"20":0.857,"200":0.857,"201":1.1668,"202":0.857,"203":0.625,"204":0.625,"205":0.857,"206":1.1668,"207":0.857,"208":0.625,"209":0.625,"21":0.857,"210":0.857,"211":0.857,"212":0.625,"213":0.857,"214":0.625,"22":0.857,"23":0.857,"24":0.625,"25":0.625,"26":0.625,"27":0.4445,"28":0.3001,"29":0.4445,"3":0.857,"30":0.625,"31":0.625,"32":0.625,"33":0.4445,"34":0.625,"35":0.4445,"36":0.3001,"37":0.3001,"38":0.4445,"39":0.625,"4":0.857,"40":0.857,"41":0.857,"42":0.857,"43":0.857,"44":1.1668,"45":1.1668,"46":1.1668,"47":1.1668,"48":1.1668,"49":1.6001,"5":1.1668,"50":1.6001,"51":1.1668,"52":1.1668,"53":0.857,"54":0.857,"55":0.857,"56":0.857,"57":0.625,"58":0.625,"59":0.857,"6":0.857,"60":0.625,"61":0.857,"62":0.625,"63":0.857,"64":0.857,"65":0.857,"66":1.1668,"67":1.6001,"68":1.6001,"69":1.6001,"7":1.1668,"70":2.2499,"71":2.2499,"72":1.6001,"73":1.6001,"74":1.6001,"75":1.6001,"76":1.6001,"77":1.6001,"78":1.6001,"79":1.1668,"8":0.857,"80":1.1668,"81":0.857,"82":0.857,"83":0.625,"84":0.625,"85":0.857,"86":1.1668,"87":1.1668,"88":1.6001,"89":1.1668,"9":0.857,"90":1.6001,"91":1.6001,"92":2.2499,"93":2.2499,"94":3.3328,"95":2.2499,"96":2.2499,"97":3.3328,"98":3.3328,"99":3.3328}}},"predictions":{"mseAttack":0.16340510726665167,"mseDefend":0.20482117382818654}}

Figure 6: Raw API Response

The sheer amount of data shown above that we needed to filter through from the API was something that we had trouble with considering our memory limitations in our Kubernetes

cluster, something which we will cover in depth in the Cloud Deployment section. Essentially, there was so much data we received that we did not know how to display all of it on such a limited page with only a select amount of memory to process these requests. Because of this, we decided only to display the most important game statistics on our Web-UI, like win rate and kill/death ratio, as well as our machine learning generated MSE projections for attack and defense. By doing this, we sacrificed showing more statistics in favor of guaranteeing that we did not exceed our memory allotment for our backend and did not clutter our statistics page with non-meaningful statistics. This whole process of determining how to display our results on the frontend with a get_results script that requests data from our backend with a provided master id took some time and effort from all our group members, but we are pleased that this was possible, as our passion for having completed the core aspect of the project was very important to us.

Backend

Our final results underscore both the challenges and triumphs of our backend development process. After overcoming significant hurdles with the Riot APIs, we pivoted to the R6 API, which proved more conducive to our project needs. This transition, while beneficial, introduced unique challenges, particularly in interpreting and manipulating the data obtained from the Siege API wrapper. The API returned data in a proprietary object format, which is not inherently compatible with Python's data structures. To resolve this, we developed Python classes to encapsulate these API objects, enabling us to parse and reformat the data into Python dictionaries. This approach not only facilitated easier data manipulation and integration but also enhanced our system's overall robustness by creating a layer that could adapt to changes in the API's output without extensive modifications to our core logic.

This method of wrapping complex API data into manageable Python objects has been pivotal in maintaining the scalability and maintainability of our backend. It allowed us to efficiently process and analyze large datasets, integrate seamlessly with our machine learning service, and provide timely and accurate predictions and statistics to our users. The development of these classes marked a significant turning point in our project, shifting from struggling with API limitations to mastering the data flow through our system, thus ensuring the success and reliability of our application in a dynamic gaming landscape. Aside from these challenges, the core code for our backend has changed very little since our description of its functionality in Chapter 3. Once switching to using R6 API, programming our backend to complete the necessary tasks was mostly straightforward, and we spent most of our time working on the ML service and cloud deployment for the final third of the project. The blueprint we created for processing data is something that we could apply very seamlessly to APIs for other games as we would like to do in the future, but in our final state the backend program works very well for all the tasks required for working with the R6 API.

Machine Learning Service

The addition of machine learning to our project was conceptualized to give StatStrikeforce a unique aspect that many other online stat tracking services do not provide, but also because we were interested in the prospect of incorporating this into our project, even though we were not sure what it would look like or how it would function. Machine learning is something we knew very little about going into this project, so we had to experiment with it quite a bit to provide something of meaning to the player, while also keeping in mind our technological limitations with the application. Through a lot of experimentation and learning about machine learning, the best way we found to incorporate this into our concept was through an MSE

generator, which aggregates many statistics from the API into two MSE statistics, one for attack and one for defense, the two game modes of Rainbow 6 Siege. This service, which is run in a python flask program, takes the statistics Kill/Death ratio, Headshot%, and KillsPerRound for both attack and defense, and creates a data frame from them. The program then uses this frame, along with a series of target values that we determined based on desirable game statistics and trains the data frame using the Random Forest Regressor model we covered in Chapter 2. By training the data with this model, a prediction for performance is generated in the form of an MSE decimal. This MSE serves as a single metric of the error in predictability of a player's ability to win a game. Because MSE is a measure of error, this means that a higher MSE means that there is more error in the prediction to win a game, and a lower MSE means that there is less error in the prediction. Essentially, a lower MSE means that you are more likely to win your next game based on your statistics. While this is somewhat confusing for players not familiar with concepts like machine learning, we found that this was the best way to aggregate prediction into a single number for the player to use rather than a complex formula or set of numbers.

While we would have really liked to keep experimenting with this ML service and discovering new ways to provide meaningful data for the players, our time limitations prevented us from further exploration of this, and we decided to continue with the MSE model for our final build. We also chose to use this because we found that it worked well with our backend and database structures and did not want to overcomplicate this process before beginning our cloud deployment phase. The main technical challenge of the machine learning for us was just learning the equations and methodology of using the service and how to extract something meaningful from it, which was very interesting but also took much more time than we initially anticipated. However, we are glad that we were able to provide a working machine learning prediction for our users, and we hope to continue to develop this in the future to find better ways to communicate the prediction data to the users in ways that are more understandable and helpful.

Database

The main technical challenge we encountered with our database functionality was the account management system. With the account system, we wanted to fully implement a feature to connect and save user account information for easy login and tracking abilities but could not fully execute our vision due to timing limitations. We were able to design our database to save the statistics generated for a user account based on their R6 player id, and have our backend query that data and display it to the Web-UI, but we struggled with saving the login information directly from the user into our database and then retrieving that data instead of the data directly from the API.

schema.sql

Users > ryansayre > Downloads > schema.sql

```
1   CREATE TABLE user
2   (
3       id            INTEGER PRIMARY KEY AUTOINCREMENT,
4       username      TEXT UNIQUE NOT NULL,
5       password_hash TEXT        NOT NULL,
6       r6_user_id    TEXT,
7       created_at    DATETIME DEFAULT CURRENT_TIMESTAMP
8   );
9
10  CREATE TABLE user_stats
11  (
12      id         INTEGER PRIMARY KEY AUTOINCREMENT,
13      user_id    TEXT UNIQUE NOT NULL,
14      mse_attack REAL,
15      mse_defend REAL,
16      FOREIGN KEY (user_id) REFERENCES user (id)
17  );
18
```

Figure 7: Database Schema

The recommendation we were given was to use a timestamp function in our database to determine when the generated statistics became stale, and when that timestamp exceeded a predetermined limit, we would pull new statistics from the API to populate the database. However, implementation of this feature is something that we did not have enough time to fully develop, so we decided to omit it from our final build of our backend querying function and focus more on the ability of the backend to populate the database with the API statistics and the machine learning statistics so that they could be easily retrieved to be displayed on our Web-UI.

Other technological challenges also came up with our front-end and back-end interaction, when using our fromhandler.js to send the signup and login form data from the frontend to the database. At first, we were able to get the interaction working in a local development environment, where we could directly see and check if the signup and login would be sent to the local running database from the local running Web-UI. When attempting to scale this onto our full production environment on the Kubernetes, we had problems with the data endpoints, and it being sent from the frontend to the backend, which due time constraints we were unable to fully implement and resolve these issues before our deadline. Our other main challenge for this was figuring out how to ensure our own safety by keeping passwords safe from us as the administrators but also to the possibility of hacking or text scraping that would uncover these passwords. We wanted to create a hasher for the passwords that would keep this safe but found that this had negative implications on the functionality of the API request to retrieve statistics. The best way we discovered to keep our own testing passwords safe while also ensuring functionality of our project was to store the id and password for all of our testing accounts into our GitHub repository as secrets, that way they could not be found publicly in our

project's repository but would also ensure safety of any passwords within our application. However, by doing it this way we effectively removed the possibility of an account to be created in our Web-UI, as users would need to make their account through our GitHub repository instead. For the final build of this project, we thought it best to preserve the functionality of the core aspect of the project, providing statistics to the Web-UI, over the proposed account system that would require rewriting much of our backend code to implement. These challenges halted our progress with the account management system, but having put down the strong foundation in our database for this will give us the opportunity to finish that phase of the project in the future when we have more time to research and develop proper security features to our application. Overall, our database serves its functional purpose very well, and despite missing the milestone with our account management system we are pleased with the progress we made in this component of the project.

Cloud Deployment

The project's cloud deployment phase gave us the most challenge of everything in this project. Demonstrating successful cloud deployment with Kubernetes was ultimately the purpose of undergoing this project, and we had to conduct extensive research to build up our knowledge of how to achieve this for our project. After we had our first working local build of our application, we began cloud deployment, starting with the Web-UI. Originally, we thought it best to Vue.js to host the Web-UI in Kubernetes, build doing it this way was challenging for us due to our general inexperience and lack of knowledge of how to use that framework. We designed our Web-UI in Bootstrap using html and CSS, and transitioning from that to using Vue instead was challenging to figure out, and we ultimately decided it was not an option given the time limit. Our next course of action was to use a simple nginx server to host the UI, something that at first, we thought would not work with communicating between pods in our Kubernetes cluster, but through development and testing found that this was very possible. Using nginx was much simpler as well, because creating the Dockerfile for this was a very straightforward process which just required us to copy all the html and css files in a folder in our repository, and nginx was able to use the js code written in our index.html file to compile everything very easily into a working web page. We found this easy to configure while also completing all the necessary tasks of hosting our Web-UI on the cloud and making it accessible externally, so we chose to use it in our final build.

Once the Web-UI was hosted in Kubernetes, we next pursued the backend and machine learning services. Writing the Dockerfiles for these flask programs took some time and experimentation to see what worked, but thankfully we were able to determine most errors using the terminal error descriptions that docker provides. For example, when using the command docker build –t backendimage . when in the backend branch of our GitHub repository, the terminal would build the image based on our Dockerfile, and give any errors that prevented the image from being successfully built. The main error we had in this process of building images was not having the correct dependencies installed into our image when trying to build. Originally, we believed that using CMD{"python3","app.py"} would run the python script and execute it as part of the image, but this would not allow us to send requests between pods in our cluster. Instead, we came up with an alternate solution to use gunicorn to run the program by installing it as a dependency, and using gunicorn allowed us to fulfill these requests by using HTTP requests with the cluster IP and port numbers we allocated for each of our components, in the form of http://cluster_IP:desired_port/accept_data. We found this strategy to work well with posting the data to the machine learning service in Kubernetes, and in our get_results script in our Web-UI to retrieve the stats from our database.

After overcoming the challenge of writing the Dockerfiles, we were able to build all the images and push them to Dockerhub, but the greatest challenge in cloud deployment for us came after this when pulling all the images into Kubernetes and deploying them. While our Web-UI and ML service pods were running, we ran into a recurring error in Kubernetes with our backend, with the error code being CrashLoopBackOff. Upon researching this, we determined that this error could be caused by many things, including conflicting ports, failure to receive a deployment update, or exceeding a memory limit for the pod. We went down the list of these possible causes and concluded that the only possibility for causing this error was that the memory for the pod was exceeded, the pod crashed, and restarted itself in an infinite loop. We were able to confirm this by checking the docker logs of the pod, where we discovered that the pod would consistently execute all commands up to command 11, stall, and then restart, and upon checking our Dockerfile we learned that command 11 was to run our app.py, which was essentially the step that was conducting the API data request. This confirmed our suspicion that it was a memory issue, and we believed that the sheer size of data given back from the API, which we showed in figure 6, was causing the memory issue. To resolve this, we tried to change many things in our cloud experiment's configurations all in one change, and somewhat to our surprise, when we next built our components, we did not receive the error and our backend was able to reach the running state. The changes we made to our CloudLab experiment were to change the site from CloudLab Clemson to Emulab, change the configuration from XenVM to RawPC, and to increase the CPU usage from 10m to 100m. Because we made all these changes in one build, we are not sure which one specifically resolved the issue or if it was a combination of multiple, but we did not want to risk terminating the experiment we guaranteed to work in order to figure that out, so we decided to extend this experiment as long as possible in hopes that we would not encounter that issue again, which thankfully we did not.

After these challenges, we were finally able to run all our component images in pods, first by pulling the images from DockerHub with a docker pull command, then updating our Github repository clone within Kubernetes with the git pull command, then finally using "kubectl apply deployment component_name" and "kubectl apply svc component_name-svc" to deploy the component images on the given ports with a running service through the yaml files that we stored in each branch of our GitHub repository. By doing this we could run the commands shown in Figure 8 to confirm that each component was running and all on their correct ports.

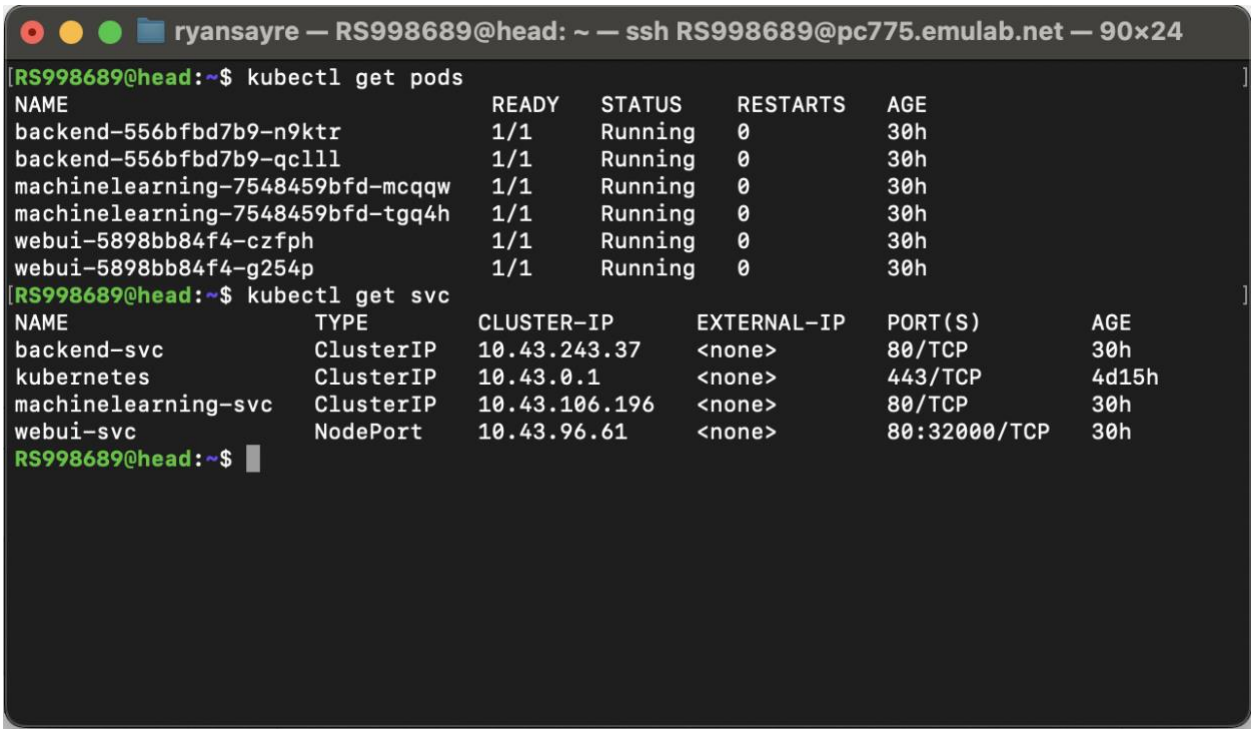

Figure 8: Kubernetes Pods in Terminal

Once all our components were running, we confirmed their ability to execute get requests, and access the Web-UI externally by using our experiment URL with port 32000 which we exposed for testing. We monitored the functionality of our components in the Kubernetes dashboard, which we configured and labeled our deployments to make this easy to view the logs of each component and find any remaining errors that were not crashing the pods. From here, we would update our deployments as necessary when we made source code updates by killing each component with “kubectl delete deployment component_name” and “kubectl delete svc component_name-svc", then we would run git pull on each branch of our cloned repository to update the deployments, and finally reapply the deployments in our cluster. By doing this through GitHub, this process was very quick and easy, so we were able to test the functionality of new features rapidly, even without the inclusion of Jenkins in our project.

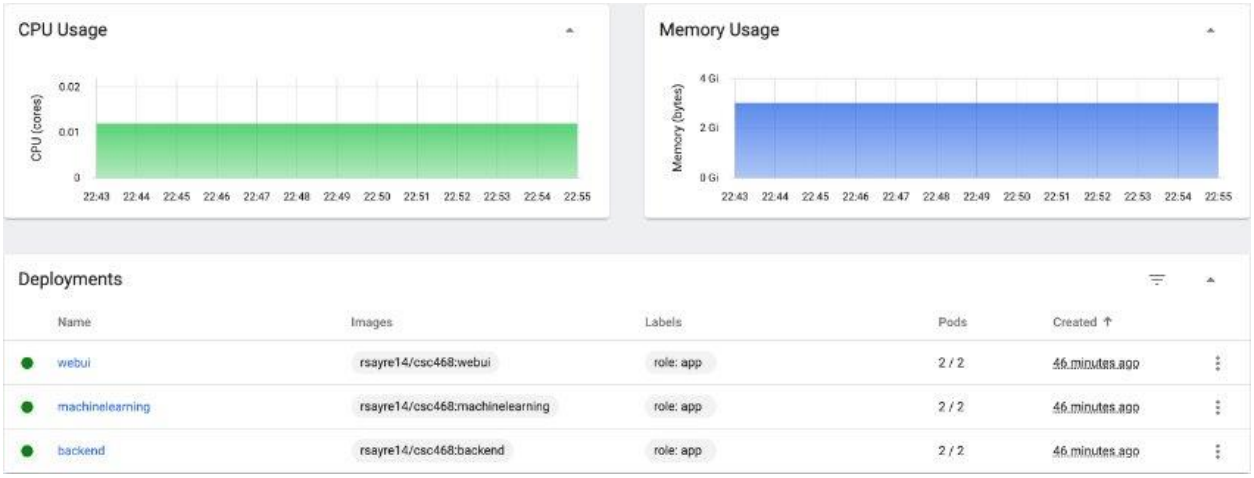

Figure 9: Kubernetes Dashboard

As far as cloud deployment goes, we really did not miss any milestones for this aspect of the project, because we wanted to run our application in the cloud and were able to successfully achieve this by the end of the project timeframe. The process of deploying our project in Kubernetes was very challenging and required a significant portion of our time since Project Deliverable 2 to figure out, which did not leave as much time as we'd hoped to finish other features of our individual components, but in the end we are happy to have deployed a functional application in the cloud, which was the main goal of the project. StatStrikeforce is very much a passion project for us, as we all love video games, and we're very proud of the overall progress we made for the final build of our application.

Chapter 5: Reflection

Debriefing with ourselves and reflecting on the final state of our project and how far we have come to get to this point, a few initial changes we could have made come to mind. Firstly, during conception, we should have utilized the free API's that were available to us rather than finding a loophole to obtain the RIOT API for Valorant. We wasted a sizeable amount of time doing this, and we could have added more to our project in the meantime instead of finding this loophole. Additionally, we would have added features such as a community page, adding the ability to track other games, and finally, having the option to track multiple games, rather than just Rainbow Six Siege. These features we wanted to implement into in our final build, and tried to do so, but we simply did not have enough time, and we could have better coordinated our efforts to try and achieve this. Cloud integration was the most difficult and time-consuming aspect of our project, and our group concluded that some of our efforts would have been better used for this aspect of the project to give us more time in the last week to focus on implementing other functionalities.

One of the concepts we learned about and gained practical experience with was the concept of http requests. Before this course, we all had a basic understanding of HTML, but integrating this into a backend was a new concept for all of us. Through trial and error and by researching how to execute these requests, we integrated our frontend and backend into a complete application. Using gunicorn to run our backend was really crucial in learning how to apply this to a cloud-based application and is a skill we certainly will use in future projects.

A concept we learned for the Web UI was that of using Bootstrap. Bootstrap is an excellent tool in building and customizing web applications, and one which was a new experience for us. Luckily, Bootstrap is a user-friendly software, and has dozens of templates users can implement and use in their own projects. Knowing how these features work together was essential in making the website aesthetic and easy to use, and paid off, as we are exceptionally proud of the UI due to Bootstrap and its libraries. We believe the website's styling and implementation of its features made it easier to use our application and test functionalities in the cloud deployment phase towards the end of our project. Bootstrap is also simple to use, and their website houses multiple tutorials to show how all the pieces of a Web-UI, like the index, background, menuswap, and styles work together to create a beautiful and aesthetically pleasing website for users.

Another key takeaway from this project was learning database design. While we had introductory level MySQL knowledge going into this project, we did not have any practical experience in designing a database to house all the data we needed and being able to query data effectively. Learning the concept of primary and foreign keys along with table structure really helped us come up with the final design for our database, which we are happy is functional with storing the API data. We had hoped to finish implementing the account system, but even just by making good progress with that feature we gained good experience in real world problems like computer security in the cloud, and how to effectively store user data where it is hidden from both administrators and possible invaders.

Although we were all mostly familiar with Python flask before delving into this project, working with the backend and ML service helped to reinforce this skill for the future. We all have become stronger Python programmers by working through this project and expanded our knowledge into complex topics like complex class structure, large data transfers, and runtime optimization, which are all extremely important in creating an efficient application. Learning about the Machine Learning service and being able to transfer that knowledge into a working Python script also reinforced these skills, and we are confident that this project has given us good preparation for the technical challenges of working in a professional setting by providing us with an understanding of the strengths and weaknesses of different frameworks that we tried out when developing our application.

The skill we learned aligns with the class lectures and learning objectives of this course most was the concept of cloud deployment. The class assignments and hands-on lectures gave us a strong background for skills like docker image creation, containerization, and the concept of utilizing cloud resources, and we were able to solidify these skills by applying them into our own project. By doing this, we learned many important cloud computing skills, for example how to write Dockerfiles, how to online tools like GitHub and DockerHub effectively, how to containerize an application’s components, how to configure external ports, understanding terminal commands and flags, and how to troubleshoot errors without having a modern IDE. This class gave us full understanding of necessary skills about how the web applications we use daily are developed and deployed, which will be very valuable in our professional endeavors. It

is also interesting to note that once you become aware of components of cloud computing, you will see them everywhere. Our group had multiple discussions about how applications we use everyday use concepts of cloud computing, and how these can relate back to our own project, which helped to further enrich us with practical knowledge and understanding of computer science topics.

The last, and perhaps most important skill we took away from working on this project was the ability to collaborate. This was really our first experience with a collaborative programming project, which comes with its own set of challenges for certain people. We agree that what allowed us to thrive in this project was having a shared passion for the concept of StatStrikeforce, and designing our project in a way that allowed each group member to highlight a skill or feature they were personally passionate about while also working towards a shared goal. Also, having a strong and clear concept for what we wanted the project to achieve from the beginning was very important, because this kept us on track with the main objectives we wanted to see implemented rather than having our scope for too broad and trying to implement too many features from the beginning. Having weekly team meetings to work on the project and being friendly outside of the project also helped us become more comfortable with each other, which removed any personal confliction that could halt project progress, so even while working through technical challenges we felt that progress was always being made. The ability to collaborate on a computer science project is a very valuable skill to have professionally, and we are glad to have been given the opportunity to experience this for our futures.

Overall, we are extremely pleased with our final project, and our success with applying the concepts we learned in class by implementation into StatStrikeforce. We agreed that this was a positive experience for all of us, with both the successes and challenges allowing us to better understand how the Computer Science industry will look once we graduate. We really enjoyed our time together, and created a bond that will last outside of StatStrikeforce. Our group is lucky in the sense that we were all knowledgeable in our respective components of building an application, so we were able to share our knowledge and help one another learn new skills through collaborating. In the end we developed our project in a way that aligns with what we originally envisioned and look forward to continuing to work on it in the future. While this project posed many challenges for us, the reward of seeing its final stage speaks volumes to our dedication and ability to demonstrate knowledge of complex cloud computing concepts.