Project Report:

- **Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

We went with our original idea, a web app to help Valorant players improve by providing easily accessible information and recommendations.

- **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

Based on our project proposal, our application aimed to be useful in the following ways: to provide players a way to view their own stats, to provide a user-friendly GUI for important meta information, and to generate match-based predictions for win loss. In certain ways, our final application was able to achieve our broad goals, but might not have succeeded in achieving our smaller ones.

In terms of allowing players to view their own stats we used an external API to indirectly Riot API which allowed us to retrieve the last 5 games played from a player's match history. After sending a request for the match history, we go through each match by first checking to see if the riot_id of the game is already recorded in Game, as we don't want any overlap. We then used their class structure to then find the information that we wanted, which we used to update the Player_Stats table.

Next, I believe our application succeeded in providing a user-friendly interface that is accessible to casual Valorant players looking not just for insights into their own stats, but into stats on the overall meta as well. Though it does provide less data, it is less visually cluttered than blitz.gg and more relevant to individual players than vlr.gg, which focuses on the competitive scene. To this extent, we have achieved this goal. However, in terms of the smaller goals we had set out for ourselves, we have fallen short either due to a dearth of publicly available data, or time constraints. Examples of this that we had suggested in our proposal include determining whether maps were attacker- or defender-sided and providing additional information about a user's pro-player look-alike.

Lastly, we successfully implemented a predictive model that uses data on agent, maps, and player stats to predict the final probability of them winning or losing a game.

- **Discuss if you changed the schema or source of the data for your application.**

We did not change the source of our data, however, we did use our revised schema from Stage 2 that implemented the use of IDs and that had split up certain tables that allowed the IDs to be primary keys. In our initial plan, we intended to have team side be part of the primary key, however from the data returned by the API, team side is not specified, and the only way we could get it would be to go through the round data individually and count the appropriate number of rounds, as well as perform the

necessary aggregates and averages on the data per round. We would also need to assume “Red” would always start T side and Blue vise versa as that is all the information we are given. In order to accommodate for the games fetch when updating user info, we added a column to Game that stored specifically riot_id for the game so we wouldn’t add already stored games. We choose to do this instead of combining riot_ids and game_ids because fundamentally they come from different sources, and because the formats are different. We changed game_id into an auto incremented row, so we would get a unique game_id when inserting a row with a riot_id, which we then went on to use in Player_Stats.

- **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

As stated above, we did use our revised schema, which thereby meant we used our revised UML diagram. We did not have any differences between this revised design and the final design. However, if you mean the original design, as in from our first edition of our design, we did implement the changes listed above, which resulted in a database that we believed had a more suitable design. We made these changes after deciding that IDs were more appropriate as primary keys compared to individual names, and split up tables with lots of attributes for ease of use.

- **Discuss what functionalities you added or removed. Why?**

We added certain functionalities based on the requirements of Stage 3, which required us to create 4 queries. Some of these functionalities do contribute to our overall goal of providing players with insights into their performance and the Valorant meta. One example would be that we added a function that provides popular synergies with a player’s most played agent, which could give insights into potential agents a player’s teammates could pick based on the meta.

Initially we wanted to link RIOT accounts directly using Riot Sign On, but we found out that we need an approved production application: “RSO Clients are only available for applications that have an existing, approved production application ID.”. This also meant that we couldn’t query the official API to get a user’s stats and we had to resort to an unofficial API.

We were unable to implement a live win rate prediction because of the limitations of the Valorant API we used. Instead, we implemented an ML model to predict win rate given stats. We marketed this as a “should you have won this game” feature.

- **Explain how you think your advanced database programs complement your application.**

Overall, our advanced database programs either further our ability to obtain personalized data for better user experience, or increase functionality when it comes to

providing player insights. Our application is able to obtain and store user data through the use of transactions and triggers to add or delete users from the database. Our stored procedure gives another layer of functionality as it provides map-based insights for players to review whether their performance on a certain map is above or below their overall baseline.

- **Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.**

Daniel: We used “pymysql” without the ORM component to make SQL queries to the database and we found that it was not thread safe. In the beginning, we used it by creating a connection and storing it in the application state. Different queries would share the connection and it worked fine because we didn’t have too many queries. Once we had more logic that queried the database concurrently, there were errors like “pymysql.err.InternalError: Packet sequence number wrong”. After some debugging, we realized that it was because pymysql (and other libraries) do not support multithreaded applications. We fixed this by creating a new Connection object per API route like so:

```
with request.app.state.db.connect() as connection:
    result = connection.execute(query)
```

Acelynn: My main task + challenge was obtaining the data to be used in our models and database. The difficulty came from the scale of the data, with our final collection having 25k matches, and how difficult the html was to navigate on its own. Information we needed was spread out, most notably in order to obtain the winner of each game I had to connect the nickname of the team with the full name through logo as the winner was given by full name in another section of the html. The dataset being large posed an issue to both speed but also diversity as there were all sorts of edge cases that popped up often pretty far into the run. We also were changing the planned tables while collecting the data, so the scraping script had to be altered a few times for that as well. Because it had to be run quite a few times, and in order not to be IP blocked by VLR.gg, all the pages were first downloaded locally. In the end, in order to speed things up, I used threading, which made the process about 50% faster, with the final run taking about 2-3 hours.

Andrew:

- Difficulty developing backend API calls and respective SQL query for the graph information. Date was formatted in VARCHAR and was difficult to parse into a format that allowed me to aggregate appropriately. In future it would be better to

pre-process date data and turn it into data type more suitable for the role before inserting into the database.

Bowen: since I joined later on, my biggest technical challenge was to figure out the existing team structure and code base. It was confusing since I do not have any experience with application development, so it was kind of difficult to figure out which code went where. For example when I was developing the account deletion code, I originally put it in auth.py, next to all the other accounts authentication stuff, because I had envisioned it being something you have to type out the password and authenticate before you are allowed to delete, however I later learned that we can just figure out who the current authenticated user is by their token and since the token was acquired from authentication, we could treat the request as authentic and can put it in the main.py code for the Web app.

Kris: One of my main contributions was the predictive model for our proposed ML creative component. Since the data hadn't been cleaned prior, I had to work with dirty data that contained null values, values that had to be coerced into the correct data type, or contained values that did not make much sense realistically. I had to drop the "Average Combat Score" column entirely due to some challenges I faced from the aforementioned issues. Despite these challenges, the model was still able to achieve an accuracy of 70% at predicting win-loss, with a mean squared error of around 0.15 at predicting probability, which is pretty decent given our challenges. I would suggest future teams, if they want to implement a predictive model, to use easily usable or clean data, so as not to have to grapple with data cleaning and preprocessing.

- **Are there other things that changed comparing the final application with the original proposal?**

Most of the things that had changed were, as previously stated, due to time and data-based constraints. The project was majorly downsized in order to fit the realities of the situation. One example would include removing the functionality to view pro matches, one reason besides reducing workload is the heavy overlap with the functionality of vlr.gg. Another possibility that we had dropped since the original proposal was the implementation of an LLM that provides suggestions to the player based on their stats. We dropped this in favor of other "creative component" functionalities that we had listed in our original proposal, such as interactive visualizations, machine learning predictions, and player-lookalikes. Out of the 4 proposed creative components, we had implemented 3, so the dropping of our LLM suggestion could be said to be trivial.

- **Describe future work that you think, other than the interface, that the application can improve on**

There were a lot of restrictions placed on what we could do because of the data we had. The data we had was in turn restricted by time, as dissecting apart the HTML + even the

API responses was a very time consuming task, not to mention the edge cases we could run into. Some interesting information that is available included the locations of all players at the moment of a notable event, such as a bomb plant. Round details were provided in both the VLR.gg information as well as the API information, so we would go back and create another table with rounds. With this additional information, and also more access to RIOT API data, it would be possible to give live predictions based on the state of the game as well as the positions of players during any given moment. We would also be able to give recommendations during agent selection based off of teammates selections + map + rank instead of having to manually input map + rank. The application could also be improved on by making more use of the pro data, as not many websites exist that show overall stats (ie win rate) on just pro matches. These would be global, and wouldn't require login.

- **Describe the final division of labor and how well you managed teamwork.**

Overall, the frontend was done by Andrew and Daniel. Backend work was distributed between Acelynn, Bowen, Daniel and Kris.

Daniel was the main player in creating our front and backend, and integrating it with our database. Setting up our Google Cloud Platform, being the anchor for our GitHub, and creating the site nearly from scratch with his vast knowledge of libraries, his contributions to our team cannot be understated. He created the entire framework and workflow on which we built on top of. He truly was our fragging IGL, our entry Duelist and indisputably the MVP of the team.

Andrew contributed to frontend and backend development, anecdotally spending ten minutes spacing buttons. Besides spacing buttons, he was responsible for the interactive graphs displaying performance stats over time. He also spent a significant amount of time (and patience) integrating and developing (wrestling with them) relevant backend API calls and SQL queries to play nice with the frontend. You could say he was our team's Controller and smokes, a player who rarely receives the spotlight but is essential to every team, controlling the pace of the game and how it plays out.

Acelynn was a key player in obtaining the data we used upon which our entire database was built. She helped to scrape our data, and once our database was up and running, she helped in backend tasks, such as implementing our pro-lookalike function and user_update_info. If Daniel was our entry, then Acelynn would clearly be our Initiator, the one leading the rest of the team in the charge onto the site.

Kris contributed to backend work, dealing mainly with SQL queries, such as implementing the advanced queries from Stage 3. They also helped with creating the predictive model used in the ML creative component. Supporting from the backlines, they were our team's Sentinel.

Lastly, Bowen also contributed to backend work, helping to implement advanced database procedures like transactions. Like Kris, he would be the team's second Sentinel.