



**COLORADO SCHOOL OF MINES**  
EARTH • ENERGY • ENVIRONMENT

# Dominion Statistics Tracker

Sam Newman

Joshua Boerma

Racyn Komata

Matthew Michal

Revised June 15, 2022

CSCI 370 Summer 2022

Dr. Jeffrey Paone

Table 1 - Revision history

Revision	Date	Comments
New	5-18-22	<p>Completed Sections:</p> <ul style="list-style-type: none"> <li>I. Introduction                             <ul style="list-style-type: none"> <li>a. Added complete introduction</li> </ul> </li> <li>II. Functional Requirements                             <ul style="list-style-type: none"> <li>a. Listed technical requirements about how the website should work once completed</li> </ul> </li> <li>III. Non-functional Requirements                             <ul style="list-style-type: none"> <li>a. Listed non-technical requirements about working on the website</li> </ul> </li> <li>IV. Risks                             <ul style="list-style-type: none"> <li>a. Listed some of the technical- and skill-based risks that will be faced while working on the project</li> </ul> </li> <li>V. Definition of Done                             <ul style="list-style-type: none"> <li>a. Listed the requirements that demonstrate when the project is complete</li> </ul> </li> </ul>
Rev – 2	5-19-22	<p>Updated Sections:</p> <ul style="list-style-type: none"> <li>II. Functional Requirements                             <ul style="list-style-type: none"> <li>a. Added stretch goals (Numbers 6 and 7)</li> </ul> </li> <li>III. Non-functional Requirements                             <ul style="list-style-type: none"> <li>a. Added requirement that everything continues to work (Number 2)</li> <li>b. Added requirement for deployment and version control (Number 5)</li> </ul> </li> <li>V. Definition of Done                             <ul style="list-style-type: none"> <li>a. Added requirement that everything that was previously working still works (Number 4)</li> </ul> </li> </ul>
Rev – 3	5-23-22	<p>Completed Sections:</p> <ul style="list-style-type: none"> <li>VI. System Architecture                             <ul style="list-style-type: none"> <li>a. Created blurb giving overview of how the system works from an external viewpoint, and a supporting diagram</li> </ul> </li> </ul>
Rev – 4	5-24-22	<p>Removed page breaks between sections</p> <p>Updated Sections:</p> <ul style="list-style-type: none"> <li>I. Introduction                             <ul style="list-style-type: none"> <li>a. Removed first person words like “we” and “the team”</li> </ul> </li> <li>II. Functional Requirements                             <ul style="list-style-type: none"> <li>a. Added context for functional requirements</li> <li>b. Reformatted functional requirements to show main goals and parts of each of those goals</li> </ul> </li> <li>III. Non-functional Requirements                             <ul style="list-style-type: none"> <li>a. Added context for non-functional requirements</li> </ul> </li> <li>IV. Risks                             <ul style="list-style-type: none"> <li>a. Removed first person words like “we” and “the team”</li> <li>b. Added context for risks</li> </ul> </li> <li>V. Definition of Done                             <ul style="list-style-type: none"> <li>a. Added context for definition of done</li> </ul> </li> <li>VI. System Architecture                             <ul style="list-style-type: none"> <li>a. Changed architecture diagram to be more descriptive and to group like elements together</li> <li>b. Added UI wireframes demonstrating the endpoint of the user interface</li> <li>c. Added Bulk Data Insertion flowchart to demonstrate how data travels from the textbox to database</li> <li>d. Added GameResultForm diagram to demonstrate how the API receives and sends data</li> </ul> </li> </ul> <p>Added Sections:</p> <ul style="list-style-type: none"> <li>Appendix B - Tables                             <ul style="list-style-type: none"> <li>a. Added appendix area for tables</li> </ul> </li> <li>Appendix C - Figures                             <ul style="list-style-type: none"> <li>a. Added appendix area for figures</li> <li>b. Updated all entries for figure values</li> </ul> </li> </ul>
Rev – 5	5-28-22	<p>Updated Sections:</p> <ul style="list-style-type: none"> <li>II. Functional Requirements                             <ul style="list-style-type: none"> <li>a. Reworded introduction paragraph</li> </ul> </li> </ul>

		<p>III. Non-functional Requirements</p> <ul style="list-style-type: none"> <li>a. Reworded introduction paragraph</li> </ul> <p>IV. Risks</p> <ul style="list-style-type: none"> <li>a. Changed “DominionStats” to “Dominion Statistics Tracker”</li> </ul> <p>V. Definition of Done</p> <ul style="list-style-type: none"> <li>a. Reworded introduction paragraph</li> </ul> <p>VI. System Architecture</p> <ul style="list-style-type: none"> <li>a. Updated introduction paragraph to more accurately represent the figures, instead of just saying “below”</li> </ul> <p>Appendix A - Key Terms</p> <ul style="list-style-type: none"> <li>a. Added a multitude of key terms, have not yet added definitions</li> </ul> <p>Appendix B - Tables</p> <ul style="list-style-type: none"> <li>a. Added revision history</li> </ul> <p>Appendix C - Figures</p> <ul style="list-style-type: none"> <li>a. Reformatted tables to remove redundant description</li> </ul>
Rev – 6	5-30-22	<p>Updated Sections:</p> <p>VI. System Architecture</p> <ul style="list-style-type: none"> <li>a. Added a figure representing the handling of log data</li> <li>b. Converted GameResultsForm figure to just show the JSON instead of a visualization</li> <li>c. Added a figure representing the storage of log data</li> </ul> <p>Appendix A - Key Terms</p> <ul style="list-style-type: none"> <li>a. Added definitions for all key terms so far</li> </ul> <p>Appendix C - Figures</p> <ul style="list-style-type: none"> <li>a. Added rows for added/updated figures above</li> </ul>
Rev – 7	5-31-22	<p>Completed Sections:</p> <p>VII. Software Test and Quality</p> <ul style="list-style-type: none"> <li>a. Added table detailing testing done</li> </ul> <p>VIII. Project Ethical Considerations</p> <ul style="list-style-type: none"> <li>a. Added some ethical considerations to keep in mind</li> </ul> <p>Updated Sections:</p> <p>VI. SystemArchitecture</p> <ul style="list-style-type: none"> <li>a. Updated system architecture diagram and wireframes for better coloring</li> <li>b. Removed figure descriptions below figures</li> <li>c. Added more text introducing figures</li> </ul> <p>Appendix B - Tables</p> <ul style="list-style-type: none"> <li>a. Added Software Testing table</li> </ul>
Rev – 8	6-6-22	<p>Completed Sections:</p> <p>X. Future Work</p> <ul style="list-style-type: none"> <li>a. Added a paragraph detailing features that could be added in the future</li> </ul> <p>XI. Lessons Learned</p> <ul style="list-style-type: none"> <li>a. Added a paragraph describing lessons learned</li> </ul> <p>Updated Sections:</p> <p>VII. Software Test and Quality</p> <ul style="list-style-type: none"> <li>a. Updated table 2 with a results column</li> </ul>
Rev – 9	6-13-22	<p>Completed Sections:</p> <p>IX. Project Completion Status</p> <ul style="list-style-type: none"> <li>a. Added project completion status</li> </ul> <p>XII. Team Profile</p> <ul style="list-style-type: none"> <li>a. Added each individual profile</li> </ul> <p>Updated Sections:</p> <p>IV. Risks</p> <ul style="list-style-type: none"> <li>a. Converted risks elements into tables</li> </ul> <p>VI. System Architecture</p> <ul style="list-style-type: none"> <li>a. Updated/converted Figures 9 and 10 to Tables 4 and 5</li> </ul> <p>VII. Software Test and Quality</p> <ul style="list-style-type: none"> <li>a. Added description of quality assurance</li> <li>b. Added code review section</li> </ul> <p>Appendix B - Tables</p> <ul style="list-style-type: none"> <li>a. Added rows for Tables 2, 3, 4, and 5</li> </ul>

		<p>Appendix C - Figures</p> <ul style="list-style-type: none"> <li>a. Removed rows for Figures 9 and 10</li> </ul>
Rev – 10	6-14-22	<p>Completed Sections:</p> <p>VII. Technical Design</p> <ul style="list-style-type: none"> <li>a. Pulled elements from System Architecture down into Technical Design</li> <li>b. Updated wireframes to accurately represent the website</li> <li>c. Updated description of wireframes to accurately represent the figures.</li> <li>d. Added example of data input format under Bulk Data Insertion</li> <li>e. Completely redesigned flowcharts to rely less on function names and more on descriptions of what is happening</li> </ul> <p>Updated Sections:</p> <p>II. Functional Requirements</p> <ul style="list-style-type: none"> <li>a. Changed requirements to be more descriptive</li> </ul> <p>VI. System Architecture</p> <ul style="list-style-type: none"> <li>a. Split into additional Technical Design section</li> </ul>
Rev - 11	6-15-22	<p>Updated Sections:</p> <p>I. Introduction</p> <ul style="list-style-type: none"> <li>a. Added further description about how Dominion is played</li> <li>b. Updated description of the existing infrastructure from the last field session</li> </ul> <p>IV. Risks</p> <ul style="list-style-type: none"> <li>a. Added more risks</li> <li>b. Updated descriptive text</li> </ul> <p>VII. Technical Design</p> <ul style="list-style-type: none"> <li>a. Added Database section, describing how the database is structured</li> </ul> <p>XI. Future Work</p> <ul style="list-style-type: none"> <li>a. Updated text</li> </ul> <p>Appendix A - Key Terms</p> <ul style="list-style-type: none"> <li>a. Added PostgreSQL</li> </ul>

# Table of Contents

I. Introduction	5
II. Functional Requirements	5
III. Non-Functional Requirements	6
IV. Risks	6
V. Definition of Done	8
Features	8
Tests	8
VI. System Architecture	8
VII. Technical Design	9
User Interface	9
Bulk Data Insert Logic	12
Log Data Insert Logic	13
Data Transfer	15
Database	16
VIII. Software Test and Quality	17
IX. Project Ethical Considerations	19
X. Project Completion Status	19
XI. Future Work	20
XII. Lessons Learned	20
XIII. Team Profile	21
<b>References</b>	<b>22</b>
Appendix A – Key Terms	23
Appendix B – Tables	24
Appendix C – Figures	25

## I. Introduction

Dominion is a deck builder game, where several people play against each other to gain the most victory points by expanding their deck continuously. There are many different types of cards, ranging from attack, action, treasure and with even more added in the expansion packs. Each player starts with a basic deck, and gradually buys and gains cards with effects, such as drawing more cards, allowing more actions to be played, or giving them victory points. Certain cards can give tokens which can be redeemed at some point in your turn such as villagers, which will give an extra action to the player that uses it. Whichever player has the most victory points in their deck at the end of the game wins.

Dominion Online is a more recent rendition of the game, hosted online and available to anyone to play for free. However, it does not have strong stat-tracking capabilities. As such, a group of Salesforce employees have taken it upon themselves – and their Colorado School of Mines counterparts – to create a website called Dominion Statistics Tracker, capable of taking and processing data about different Dominion Online [1] games.

This project is an extension of a previous field session project. The goal is to build off of what was created by the previous team, and increase functionality. The website takes in information about a session of the card game Dominion, stores it in a database, and uses that information to generate a series of statistics about each player. It also leaves the raw data available for viewing.

The client is a small group of Salesforce employees who enjoy playing Dominion Online together. They want a website where, after each game that they play, they can report who played, how they did, and a log of the game. They would also like several other statistics, such as the number of attacks. The team would like this so that they can compare amongst themselves, and so that they can tell who the strongest of their players are.

At the beginning of the project, the website was fairly basic, but completely functional. In order to insert into the database, a user had to insert one game at a time, row by row. The game's data would be inserted in a table with each player, their place, and their number of victory points. Once the data was uploaded, it would go into the database and be displayed in various tables and graphs on the Dominion Stats Tracker website. Uploading data easily became a tedious process when the user needs to report multiple games at once. It was also not able to store the game's log; only the placement of each player and how many points they got. The goal of this project is to be able to bulk upload multiple games at the same time, and expand so that the game log can directly be sent to the website, which extracts relevant data about points and attacks, and use that to update the statistics demonstrated.

## II. Functional Requirements

There are several functional requirements for this project. While they are broken down in many small requirements, there are two main goals for the completion of the project: the website can handle bulk insertion of data, and the website can store game logs and use them to generate more data. There are also several stretch goals, which are tasks either for future iterations of the project or for the completion of this iteration.

The specific functional requirements are listed below, in order of descending priority.

1. Bulk Insertion:
  - The website can update its database without the user having to upload games individually through one of the following:
    - i. Update the insert functionality to support the bulk insertion of multiple games.
    - ii. Update the website to be able to accept copy/paste from the corresponding Google Sheet.
2. Log Tracking:
  - Update the schema of the website's database.
  - Update the database to keep track of new statistics.
    - i. Allow for the upload of JSON files from LogDog [2] to update the database with game logs.
    - ii. Parse through each log to pull out important/interesting statistics.

- Update the website to display new metrics based on the new statistics.
  - i. Main desired metric is the number of attacks used in a game.
  - ii. Use the number of attacks to calculate the % chance of playing an attack.
  - iii. Track the most played card by each player.
- 3. Stretch Goals:
  - More advanced statistics based on log data
  - Feature to filter games by year

### III. Non-Functional Requirements

There are also a variety of non-functional requirements to keep in mind. The top priority is that the website continues to work as well as it did at the beginning of the project. That is to say, all features that were previously implemented must continue to function. Furthermore, it is important that all new features run on the existing infrastructure of the website, with only minor changes.

All version tracking and code changes must be handled through GitHub. The client's deployment is constantly updated from a GitHub repository, and as such, all changes to the website must be pushed to said repository. It is important that each push is carefully being tested before being sent to deployment.

Finally, it is a necessity that all code is written in TypeScript, HTML, CSS, and SQL. These are the languages that the website is built in, and they are the languages that it should remain in.

The specific non-functional requirements are listed below.

1. Everything that is currently working, remains working.
2. Leave existing infrastructure in place with only minor changes.
3. Use GitHub repository for version control.
4. Use branches on local environments to test new features. Once confident, push to the team fork to test. Once confident in that, pull request main repo.
5. Code is written in CSS/TypeScript.
6. Use Lightning Web Components [3] to design the user interface.

### IV. Risks

The Dominion Statistics Tracker project is fairly low-stakes. There are few stakeholders, all of whom are programmers themselves. However, as with any project, there are still risks associated with it. These include both technology- and skill-based risks, including but not limited to breaking the currently functional website and damaging development machines.

As the website is currently functional, it is a necessity that it continues to be. All additions to the website should build on functionality, without damaging previous functionality. There is a fairly likely risk of this happening, as no members of the team are experienced in web development. The features that are most at risk are the database, which will continue to store live data as work is done on the website, and the UI.

The most significant risks faced while working on the Dominion Statistics Tracker website are listed below.

Table 2 - Technology Risks

Risk	Likelihood	Impact	Mitigation
Initiating a Pull Request that crashes or otherwise breaks the current website.	Somewhat Likely	Major, relative to the website itself	Be completely confident in each PR; have it reviewed by at least two team members before merging.
Breaking development machines while learning to use certain technologies.	Unlikely	Major	Be cautious while using new technologies, make sure their functionality is understood, ask the client if uncertain.
Damaging the existing database by altering the database creation and migration scripts.	Somewhat Likely	Major	Ensure that no previous database files are altered, and that all changes to the database are made in subsequent scripts. Also ensure that all modifications to the database move data if necessary, or preserve data in some other way.
Breaking the existing UI while attempting to add or alter an element.	Likely	Minor	If anything breaks the UI, the team can always revert to the previous change. It is important that commits are made with every major change, so that reverts can be made without too much lost time.

Table 3 - Skill Risks

Risk	Likelihood	Impact	Mitigation
Spending too much time learning or familiarizing with a new language.	Likely	Minor, may spend too much time learning	When running into problems, instead of trying to individually devise a new solution, communicate with other teammates and ask if they have a solution or can help with devising one.
Not meeting client's requirements due to lack of communication or understanding.	Somewhat Likely	Major	Communicating with the client frequently both through bi-weekly meetings and slack.
Not meeting time constraints due to inexperience budgeting time with large software projects.	Somewhat Likely	Major	Plan out work in a schedule and review the schedule weekly, making adjustments as needed.



## V. Definition of Done

The clients are very fluid in what they would like, and understand that things are apt to change often. With that said, it is still important that there are some milestones that denote whether or not the project was a success. Most of these milestones are features, listed above in the requirements. There are also the basic standards that the website continues to work at least as well as before, and that it can pass functionality tests. Therefore, the definition of done for the Dominion Statistics Tracker app states that the website meets the requirements listed below.

### Features

1. Ability to bulk insert data from a Google Sheet into the website.
2. Ability to insert game logs from LogDog into the website.
3. Update website UI to display more metrics per player:
  - a. Number of attacks made in a game.
  - b. Likelihood of making an attack in a turn.
  - c. Percentage of cards bought that are attack cards.
  - d. Most played cards across all games.
4. Everything that was working previously still works.

### Tests

1. Unit tests for basic metrics using preset logs:
  - a. Number of attacks made in a game.
  - b. Likelihood of making an attack in a turn.
  - c. Percentage of cards bought that are attack cards.
  - d. Most played cards across all games.
2. Overall functionality tests
  - a. Is a user able to bulk upload and see new games added?
  - b. Is a user able to upload a log file and see new games added?

### Delivery

- Product is delivered through pull requests to the existing GitHub repository which is automatically uploaded to the live website.

## VI. System Architecture

The architecture of the system is fairly straightforward. All data is generated by games played on Dominion Online. Some data from Dominion Online is read by LogDog, which parses it into a JSON file. The user then manually inserts the JSON file, along with the results of the game itself, into the Dominion Statistics Tracker UI. Once submitted, the UI takes the data in on the client side and formats it appropriately to be sent to the server, through the Dominion Statistics Tracker API. Once in the server, the data is formatted to fit into the Postgres database, and is inserted.

When the user requests data from the Dominion Statistics Tracker User Interface, a request is sent through the client to the API and the server, which then extracts the relevant data from the database. That data is then sent back to the client, which formats it to be displayed on the UI. A visualization of the architecture of this project can be seen in Figure 1.

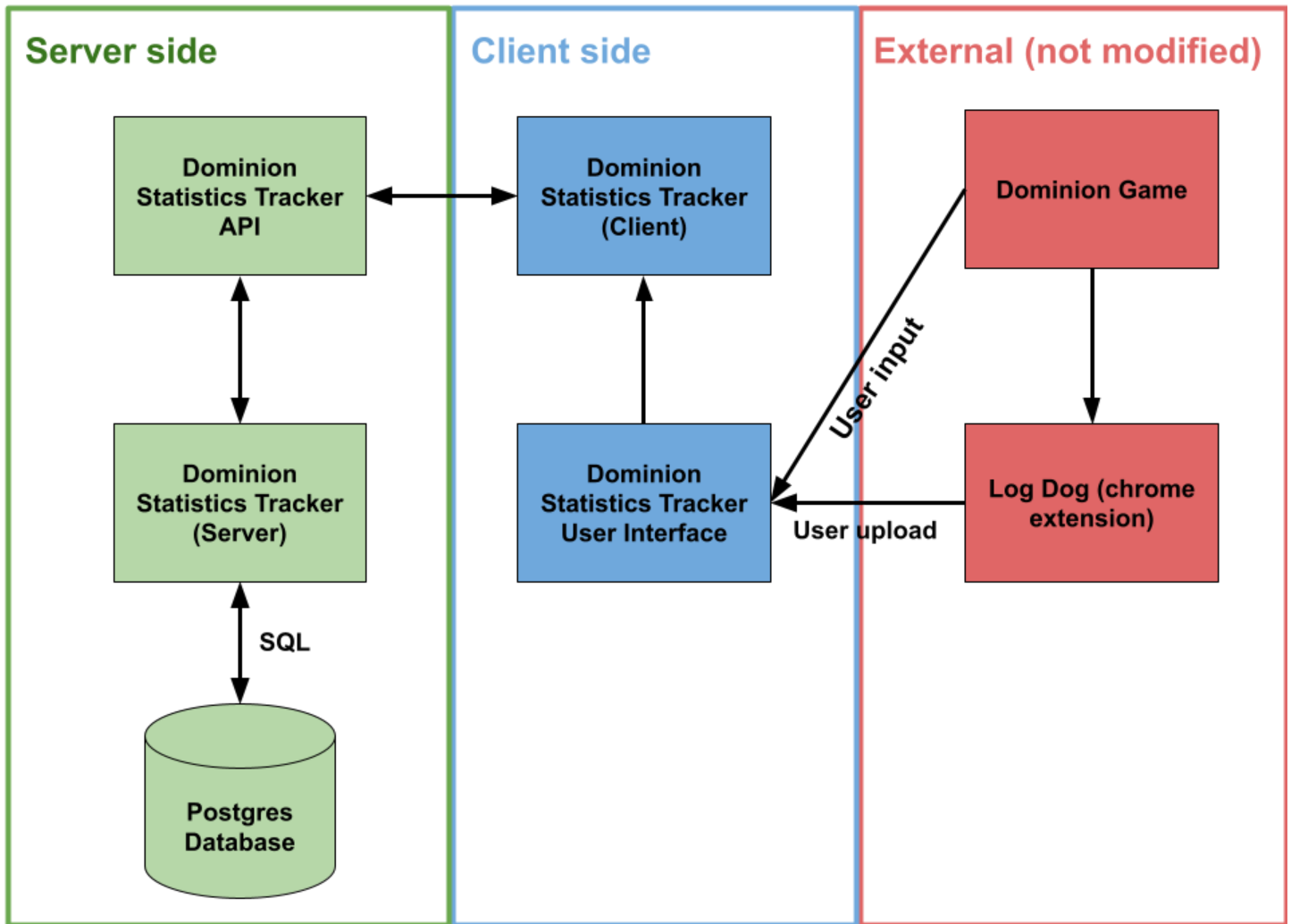


Figure 1 - System Architecture Diagram

## VII. Technical Design

### User Interface

Figures 2 - 5 are models of the website's UI, demonstrating how it looks in the finished product. The home page of the website is modeled in Figure 2 and displays generic game information. This includes a button that leads to another page that displays the raw data generated by the game logs, demonstrated in Figure 4. From all pages of the website, the user can open the menu and navigate to the data upload screen, as seen in Figure 3. From this page, the user can enter data for bulk insert into the textbox, or they can upload a log file to be parsed and stored. In both Fig. 2 and Fig. 4, clicking on a player's name redirects the user to that player's corresponding page which displays all of their statistics, seen in Figure 5.

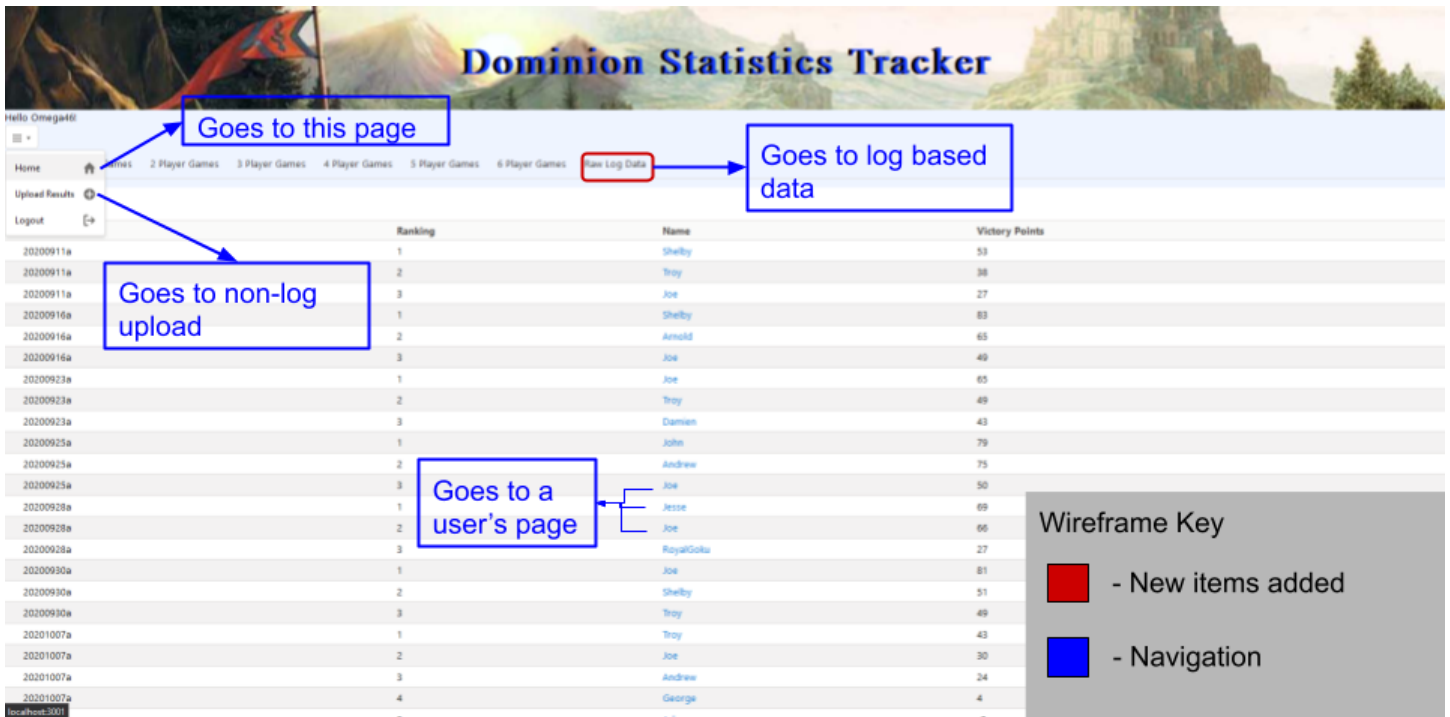


Figure 2 - Non-Log Based Data Wireframe

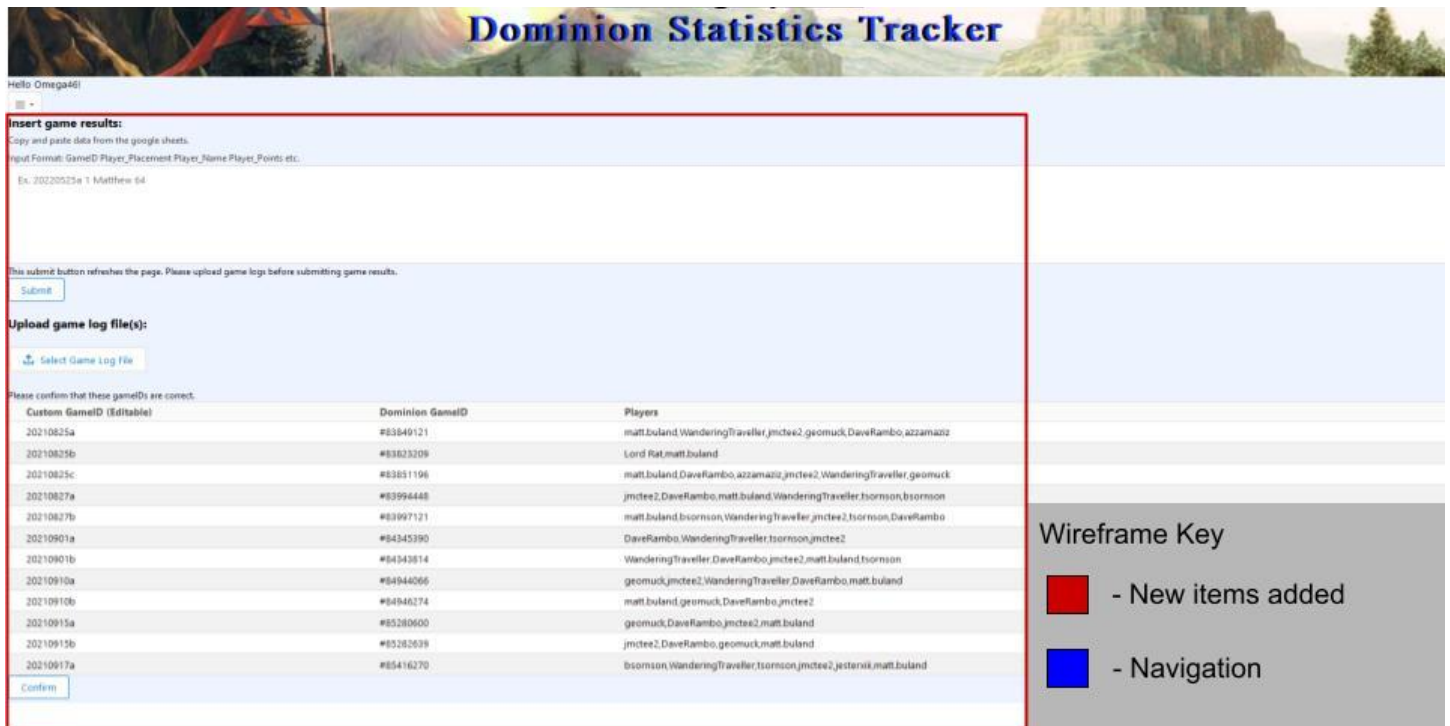


Figure 3 - Data Upload Wireframe

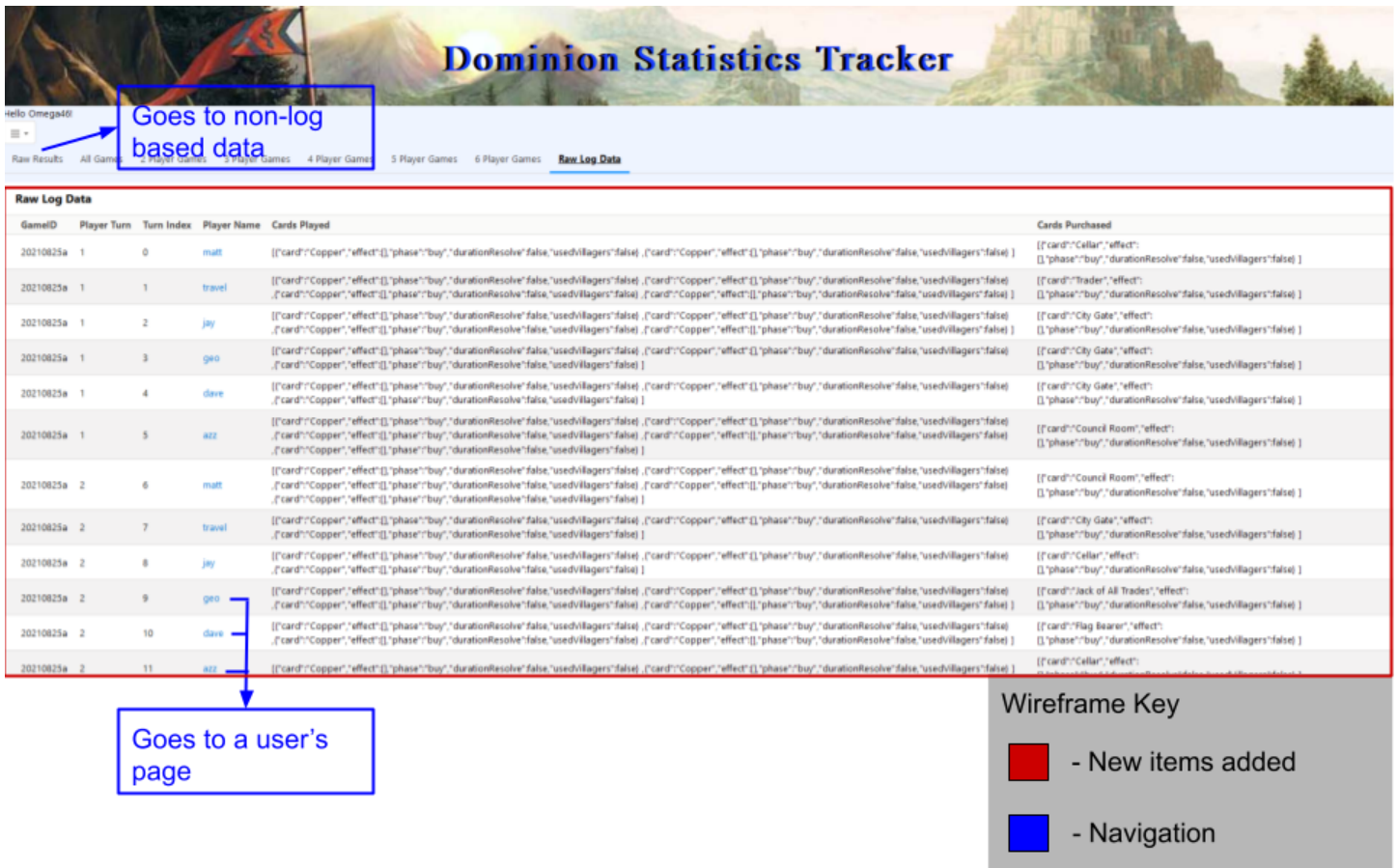


Figure 4 - Log Based Data Wireframe

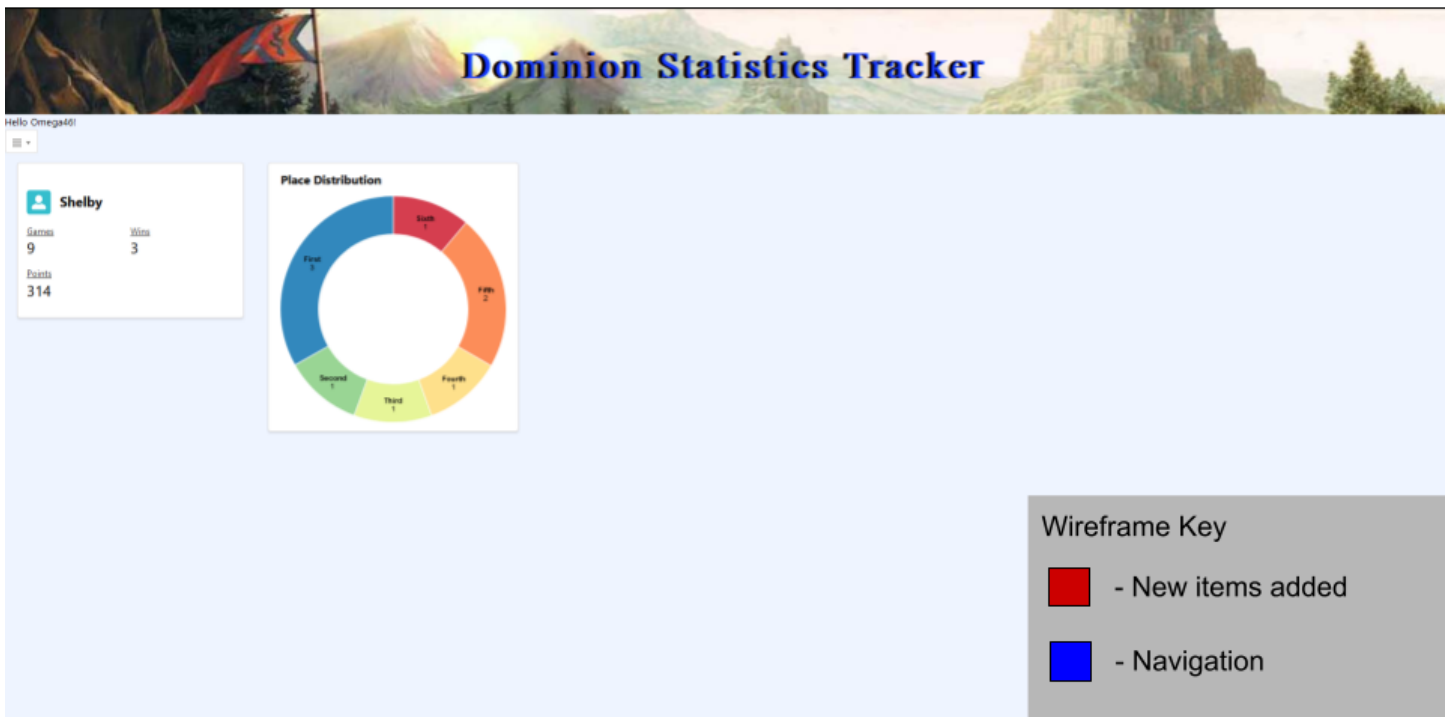


Figure 5 - User Information Wireframe

## Bulk Data Insert Logic

In order to use the bulk insert functionality, a user inserts a list of game results into the UI box seen in Figure 3, using the format seen directly below. The user can insert as many lines as they would like, as long as they all follow the expected format and as long as the placement logic makes sense.

### Bulk Data Insert Format

**game\_id player\_place player\_name player\_points**

**20220614a 1 Matt 11**

**20220614a 2 Troy 10**

That data is then collected by the client, and processed. In order to process, the client first splits up the input by a new line. It then checks to see if there is another line of input. If there is, it extracts the player's name, place, and victory points and inserts it into a PlayerData object, demonstrated in Table 4. The input line's game ID is then checked. If it is the same as the previous one, or if there is no previous one, then the entire PlayerData object is added to a list. If it is not the same game ID, then the PlayerData list is added to a GameData object, as seen in Table 4. The game ID is also added. That GameData is then validated, and if it passes, it is added to a list of GameData. The process repeats until there are no more lines of input.

Once all input has been sorted through, the list of GameData is sent via POST to the Dominion Statistics Tracker API, which is stored on the server. The API then transfers all of the received data to the server-side code. There, the server queries the database for any of the game IDs. If any exist, an error is returned to the UI, telling the user to verify that their game IDs are correct and to try again. Otherwise, all of the data is pushed into the database. The entire process can be seen in Figure 6.

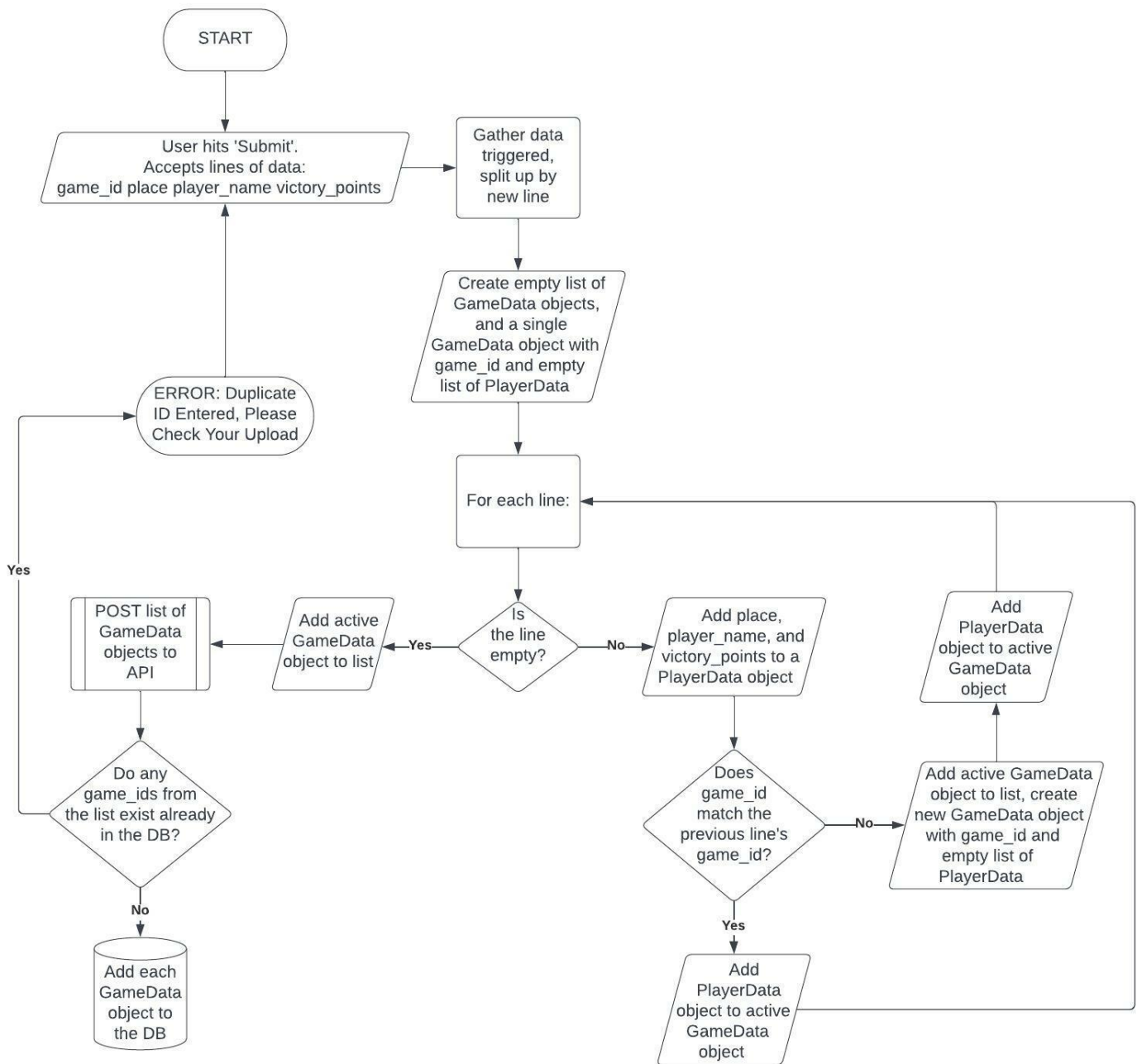


Figure 6 - Bulk Data Insertion Flowchart

## Log Data Insert Logic

The log tracking functionality of the website begins at the log insert page of the UI, seen in Figure 3. The uploaded log must be a JSON file. That file is accepted by the UI and is brought into the client. The client reads all of the data out of the file. For each log in the JSON file, it extracts the list of player usernames for that log and sends it via POST to the API. From there, the server creates a mapping for each username from username to the actual name of the player. Any username that does not have a database entry is left with an empty player name mapping. That map is then sent back to the client.

On the client side, the user is prompted to enter the player name corresponding to each username that was left with an empty mapping. Those inputs are accepted and added to the mapping. Once each empty player name has been filled, the client generates a table representing the file, where each row corresponds to a log. This table can be seen in Figure 3.

Each row contains the Dominion Online ID of the game, the Dominion Statistics Tracker ID of the game, and the players who played in that game. The user is allowed to edit the Dominion Statistics Tracker ID for each log. Once all IDs are correct, the user can hit the Confirm button. At this point, a list of all of the logs is sent via POST to the server.

The server loops through each log in the list individually. For each log, the parser first removes all of the HTML in the file, leaving a string of sentences. The parser then splits that string up by the keyword "Turn", so that the log becomes a list of strings, each representing a single player's turn. The server then proceeds to loop through each turn. For each turn, a PlayerTurn object is created, containing the game ID, the turn number, the player's name, a list of the cards played, and a list of the cards gained. Both the cards played and gained are formed of a list of PlayedCard objects. The PlayerTurn object and the PlayedCard object can both be seen in Table 5. Once every turn in a single log has been processed, the server checks if any errors occurred. If there were errors, it sends a message back to the UI, informing the user of exactly what went wrong and how to fix it. If there were no errors, each turn in the list is added to the database. The entire process can be seen in Figure 7.

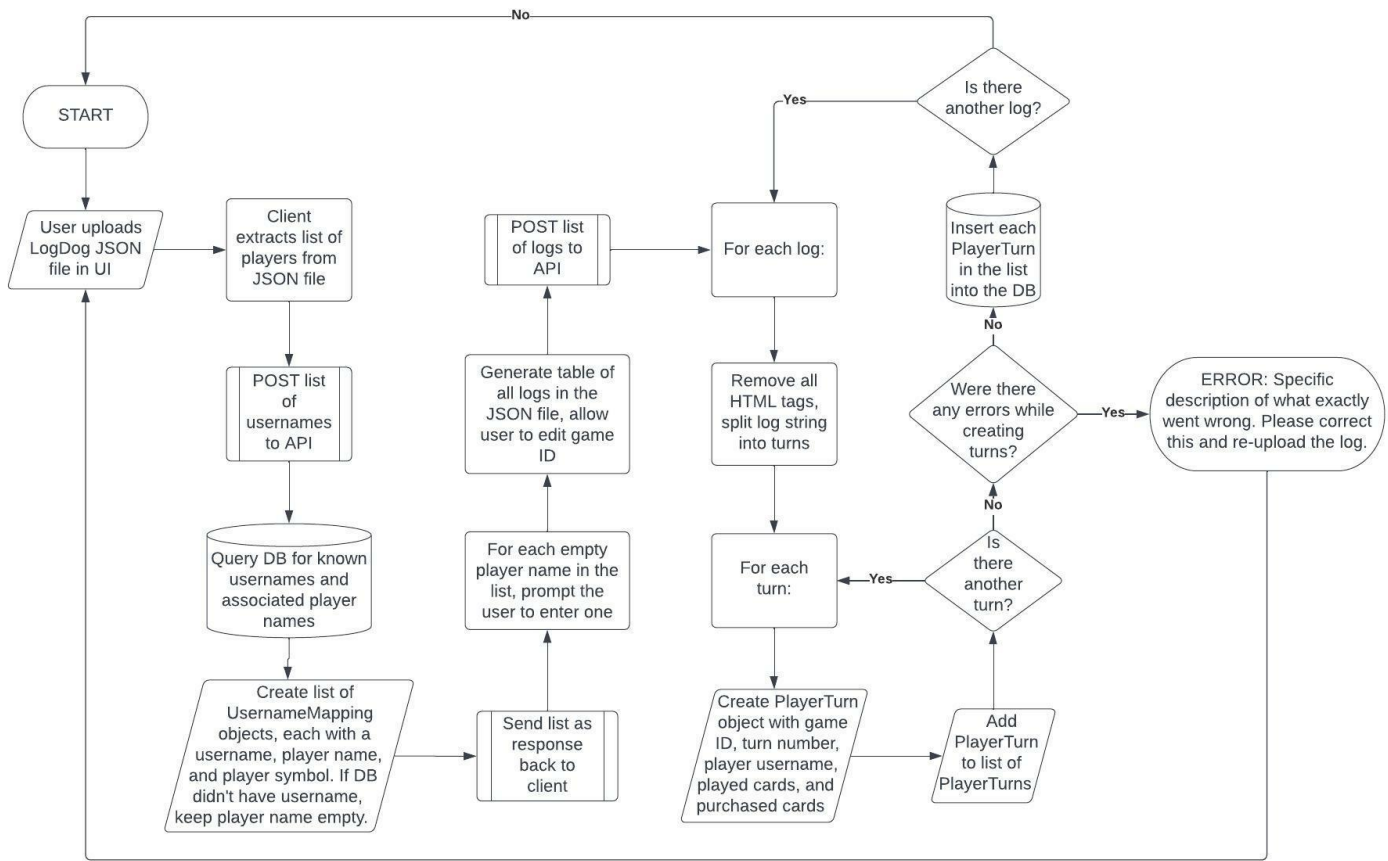


Figure 7 - Log Data Insertion Flowchart

## Data Transfer

Table 4 depicts how data from bulk data insertion is packaged to be sent over the API on the website. A GameData object is sent, containing the game ID along with a list of PlayerData objects, each of which contain relevant information about the player, such as player name, place, and how many victory points they finished with. That data is then individually extracted and added to the database.

Table 4 - Bulk Data Insertion Objects

GameData	PlayerData
gameId: string; playerData: PlayerData[];	playerName: string; playerPlace: number; victoryPoints: number;

Table 5 depicts how data from a log is split apart to store in the log database. The PlayerTurn models a player's full turn. All cards that the player used during their turn are stored in the PlayedCards array, and all cards that the player purchased or gained during their turn are stored in the PurchasedCards array. The PlayedCard interface models a card by tracking its effects, played phase, whether it was the result of a card played on a previous turn, and if villagers are used to play it. The effects of a card are stored as a list of PlayerEffects, which are broken down into type and the name of the player that they impacted.

Table 5 - Log Insertion Objects

PlayerTurn:	PlayedCard:	PlayerEffect:
gameId: string; playerTurn: number; playerName: string; playedCards: PlayedCard[]; purchasedCards: PlayedCard[];	card: string; effect: PlayerEffect[]; phase: 'action'   'buy'   'night'   'attack'   'reaction'; durationResolve: boolean; usedVillagers: boolean;	type: string; player: string;



# Database

The database is built in PostgreSQL, and contains three tables relevant to the website. One table, the game\_results table, stores all information relevant to the bulk insertion. However, this table was not altered at all during this iteration of the project, and as such is not modeled below.

Table 6 describes how data from each turn pulled from a log is stored within the database. The primary key for this table is a compound of id (not pictured) and game\_label and player\_name. It also has the unique attribute of storing several JSON objects, in order to be accessed later. This raw data is displayed on the UI, and is easily accessible for later use in visualization statistics.

Table 6 - log\_game\_round Database Table

	log_game_round					
<b>Column Name:</b>	game_label	player_turn	turn_index	player_name	cards_played	cards_purchased
<b>Column Data Type:</b>	VARCHAR	INT	INT	VARCHAR	JSON	JSON
<b>Purpose:</b>	Stores the Dominion Statistics Tracker unique ID for each turn	Stores turn number	Stores the the order that the turn was played in	Stores the player whose turn it is	Stores a JSON object that is a list of every card that the player played, and their effects	Stores a JSON object that is a list of every card that the player purchased, and their effects

Table 7 describes the known\_usernames data table which holds the username data for the website. The username is the primary key of the data table which describes a username on the dominion.games website. The player\_name stores a string for a player's name on the dominion statistics tracker website. The purpose of this data table is to track existing users and automatically map their dominion.games username to a player name on the website.

Table 7 - known\_usernames Database Table

	known_usernames	
<b>Column Name:</b>	username	player_name
<b>Column Data Type:</b>	VARCHAR	VARCHAR
<b>Purpose:</b>	Stores all known usernames of users that have played	Stores corresponding name to each username

## VIII. Software Test and Quality

With so many moving parts in this software, thorough testing and quality assurance is essential. Many aspects, such as the several functions of the log parser, can be tested via unit testing. By feeding some input into a function and ensuring the output is exactly what it should be, much of the website’s parsing functionality can be tested. However, some more in-depth features – like data transfer and database access – are not as simple to test. Functional testing can help solve this: by physically uploading accurate data and ensuring that the website responds as expected, the website’s functionality as a single unit can be checked.

Table 8 - Software Testing

Test method	Description	Results
Unit Testing (Existing Tests)	The existing code on the website has some built in unit tests which test the functionality of the methods to verify input data and process it. These existing tests are used to make sure that any new code added on does not break the existing system which is one of the project goals. With the existing tests implemented already they can be used to verify that previous methods still function even with the new code added on.	Existing unit tests have been slightly updated to account for consistency errors with the new insert method. All previous tests created still pass with the new functionality.
Unit Testing (Bulk Insert)	Unit testing is used with the bulk insert methods to verify that the input is valid and can be safely sent to the server. These tests verify that the input into bulk insert is properly processed and sent to the server. They also test against bad data that should be rejected – rather than sent to the server – resulting in an error being displayed to the user.	<p><b>Frontend:</b> On the frontend, there are tests to ensure that the data being uploaded to the text area is properly formatted before the data is parsed through. If the data is improperly formatted, the data upload is rejected and an error is returned. All tests pass.</p> <p><b>Backend:</b> On the backend, there are a few unit tests to ensure that when the data is being uploaded to the server, the data isn’t already in the server (i.e. ensuring that duplicate data isn’t being uploaded to the server). All tests pass.</p>
Unit Testing (Log Upload)	Unit testing is very important for managing and checking the functionality of log uploads. Not only is there huge variability in what the user can actually upload, but there are tons of niche details that could break the log parser. It is important that there are a wide array of tests to check all of the moving components, including but not limited to the file upload, acceptance of the data on the server side, and the parsing of the JSON data.	<p><b>Frontend:</b> On the frontend, tests ensure that the file uploader rejects any file that’s not a JSON file. Tests also check that the file acceptor can check for unexpected elements within the file itself. All tests pass.</p> <p><b>Backend:</b> On the backend, there are dozens of unit tests that experiment on the parser using a wide variety of potential mistakes made by the log, along with all of the valid inputs that can be expected. One of the important features to test is that the parser is able to accept a list of names or aliases for the players of the game, and can attribute them to each action that they take. A series of tests ensure that valid and invalid</p>

		<p>usernames can be handled. There are also tests that ensure that each line of a log is properly processed, ranging from card creation to effects creation to the accumulation of all of that into a player turn. All tests pass.</p>
<p>Functional Testing (Bulk Insert)</p>	<p>In order to verify that the bulk insert methods handle all inputs correctly, it cross-references the input with the server to ensure that the input is valid. To test this functionality, a variety of data is inserted to verify that valid inputs were uploaded to the server and invalid inputs returned an error.</p>	<p>The functional testing for bulk insert involves uploading some example data to the bulk insert both in the form of a single game as well as multiple games at once. The games are sourced from created games and existing data in the client's google sheet. With the functional testing the bulk insert feature works as desired, adding to the database and throwing errors when necessary. All game data tested pass.</p>
<p>Functional Testing (Game Log Upload)</p>	<p>With the log upload, logs from existing and known games are uploaded to verify that the database has the expected results from the logs. These tests also verify that the database is storing and processing the data correctly to yield the expected result of the log file. In addition bad log files are uploaded to verify that the server rejects the bad data.</p>	<p>The system is tested with multiple existing logs provided from the client as well as being tested with logs that the group generates. The purpose of the client's logs is to test general functionality of the system on a wide variety of cards. The logs that the group creates are mainly for testing the edge cases of the dominion game logs. All logs tested pass.</p>
<p>Code Reviews</p>	<p>Even after extensive functional and unit testing, the physical quality of code still must be checked. This is where code reviews come in. By checking each other's code, the team is able to look for any flaws or minor bugs. Furthermore, the team's code is reviewed by the client himself upon PR. This ensures that the code is up to the client's standards, and allows the client to provide insight on how to improve the logic of the code.</p>	<p>The client was able to identify many areas of code that could be improved in terms of simplicity and efficiency. By providing feedback directly to the Pull Request, the team was able to implement several more efficient methods of database access and insertion.</p>

## IX. Project Ethical Considerations

Unlike many projects, the Dominion Statistics Tracker does not have a wide impact on people. It is only used by a small team of software engineers to track the statistics of one of their freetime hobbies. With that said, it is still important to take into consideration the ethical implications of this project. It is the duty of the Dominion Statistics Tracker to accurately and efficiently store, preserve, and handle the data of the client. Failure to do so would violate the client's trust in the website, and could result in the loss of years of game data.

One of the main ethical considerations for the Dominion Statistics Tracker is the preservation of existing data. No updates to the website should be allowed to damage or delete any of the data that is already stored in the database. Furthermore, no updates should remove any of the statistical views that are currently visible. None of the website's functionality from the beginning of the project should stop existing; it should only be improved. While failure to comply with this would not have impacts on a large number of people, it could be devastating for those who enjoy this as their hobby.

On a similar note, it is important to keep security in mind. While this may not be a high-profile target for hacks or cyberattacks, that doesn't mean that the Dominion Statistics Tracker is immune. Allowing someone to gain access to the UI without proper permission could result in them changing, adding to, or deleting data elements, resulting in the same effects as the damage done above. It is the responsibility of the Dominion Statistics Tracker website to keep the information given to it safe and secure, altered only by those authorized to do so.

In addition, the work on the project was done in compliance with the ACM/IEEE principles, specifically principles 3.10 and 3.15. Principle 3.10 states that the software and documentation produced is adequately tested, debugged and reviewed. The code that was produced was tested extensively using a combination of unit tests and functional tests, as well as code reviews performed within the team and with the client (refer back to Section VIII). Principle 3.15 states that software maintenance and new development should be treated with the same level of professionalism. It's important that the existing code on the website is kept up to date with new code so that they work together as a one cohesive entity rather than interfering with one another.

## X. Project Completion Status

The current project website meets most of the stated requirements for the definition of done. The bulk upload feature is fully implemented, which allows multiple games to be inserted at once. Verification is run on the input of the bulk insert to verify that duplicate game ids aren't being added to the database. The log upload feature is fully implemented with a file upload option added to the upload page of the website. In the backend, the logs are parsed through to collect the relevant data and break it into player turns. The log data display is also implemented, but it is only in the form of a basic table which lacks some of the features that the client desired. Log metrics aren't implemented; however, the necessary data to generate the metrics is in the database. The stretch goals of tracking number of trashes in a game and filter by year are not present in the current website.

## XI. Future Work

No software engineering project is ever truly finished, as there are always more bugs to fix and features to add. Yet due to the limited time frame of this project, there are bound to be additions that will be passed to future contributors. For the Dominion Statistics Tracker, there are a number of things yet to be implemented, and multiple features that could be added to improve the project overall. Things that were unable to be implemented in this field session include multiple of the stretch goals from the functional requirements, such as the ability to filter by year, visualizing the log-based data and a statistic to track the number of cards trashed. With the multitude of data read in from the log file, all of the information and infrastructure needed to implement these features exists in the project, one would just need to add additional table columns and graphs to show the new data.

The client was aware that these features would be difficult to include on top of the other requirements, and they will likely be pushed back into the future. Some other implementations that were not a part of this project, but could still work to improve the website such as: showing a representation of how the Dominion game actually was played in the site, having a table of players for comparison purposes, and the ability to sort by any statistic to see specific statistics from all games played.

## XII. Lessons Learned




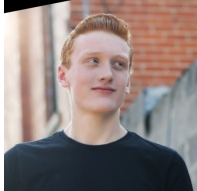
The team learned a variety of lessons from this project, both on the side of actual software development and on the social and interpersonal side. Among these lessons, the most important lesson learned was the value of communication and trust between the client and developers and between the team members. It became very apparent that the team would struggle to make significant progress on the project without the assistance of the client. By reaching out to the client and explaining in detail the struggles that were encountered, the team was able to gain the assistance necessary in understanding the software and could make good progress. Even in situations where the client may not be as technically skilled as the Salesforce employees that the team interacted with, it is necessary to communicate thoroughly with them. In some situations, they may have insight into the project that no outsider could hope to have. The trust between team members was also incredibly valuable: being able to reach out to another member of the team with a challenge and walking through it together saved a lot of time in the long run.

To expand on the value of communication, the team learned the value of SCRUM and other software development workflow tools. Being able to get together and know exactly what needed to be done – along with an estimate as to how long it would take – enabled the team to prioritize features of the website. This, in turn, resulted in a higher quality product all around.

Finally, the team discovered the significance of test-driven development and the value of tests in general. By the time the project was nearing completion, there were so many moving parts that it was impossible to manually check each and ensure that it was working to the fullest of its ability. A full suite of tests allowed for near-instantaneous verification of functionality. This provided the team with a sound mind while pushing features, and allowed for more efficient production.

### XIII. Team Profile

Table 9 - Team Profiles

Photo	Description
	<p>Racyn Komata Senior Computer Science Hometown: Mililani, Hawaii GitHub: <a href="https://github.com/rkomata">https://github.com/rkomata</a></p> <p>Role: Client-side user interface and data transfer</p>
	<p>Matthew Michal Junior Computer Science Hometown: Allen, Texas GitHub: <a href="https://github.com/Matthew-Michal1">https://github.com/Matthew-Michal1</a></p> <p>Role: Client-side user interface and data transfer</p>
	<p>Joshua Boerma Junior Computer Science Hometown: Highlands Ranch, Colorado GitHub: <a href="https://github.com/JoshBoerma">https://github.com/JoshBoerma</a></p> <p>Role: Server-side log parsing and database interaction</p>
	<p>Sam Newman Senior Computer Science Hometown: Aurora, Colorado GitHub: <a href="https://github.com/snewman4">https://github.com/snewman4</a></p> <p>Role: Server-side log parsing and database interaction</p>

## References

[1] Dominion Online. 2016. [dominion.games](https://www.dominion.games)

[2] Goatze, Micheal. May 26, 2020. "LogDog". <https://www.github.com/michaelgoetze/LogDog>

[3] Salesforce.com. 2019. <https://lwc.dev/>

## Appendix A – Key Terms

Term	Definition
Dominion Online	An online rendition of the card game Dominion. Dominion Online is a website where players from across the world can play Dominion against each other, competitively or casually.
LogDog	A Chrome Extension designed to take and store log data from Dominion Online. LogDog watches for when a Dominion Online game ends and automatically saves the log as both a JSON and as a .txt file.
TypeScript	A superset of JavaScript, TypeScript adds static typing to the functionality of JavaScript. Static typing means that the types of variables are determined at compile time, and ambiguity in types can be used to cause errors that would otherwise be runtime errors.
HTML	The abbreviated name of HyperText Markup Language, HTML is the language used to design forms and pages that will show up on a web browser. Most website front ends are written in HTML.
CSS	The abbreviated name of Cascading Style Sheets, CSS is a language used to describe the style of the above-mentioned forms written in HTML. CSS allows for more precise and flexible styling.
SQL	SQL is a language designed for working with databases. SQL is used to enter information into a database, extract information from a database, and edit the database format.
PostgreSQL	PostgreSQL is a database management system that utilizes SQL in order to store data in a series of user-defined tables.
JSON	JSON is a file format that makes the storage and transmission of data simpler, in a human-readable format. Consisting of attribute-value pairs, JSON is useful for condensing and simplifying information to be sent.
Lightning Web Components (LWC)	Lightning Web Components are CSS-styled template elements of a website, built by Salesforce, to make website styling easier.
UI	The abbreviated name for a user interface, a UI is the part of a website or software that the average user will interact with. It hides all of the behind-the-scenes data transfer and processing, and aims to be simple and easy to understand.
API	The abbreviated name for an application program interface, an API is a type of software that goes between two other pieces of software in terms of data transfer. It accepts data from one software and reformats it to be accepted by the other software.



## Appendix B – Tables

<b>Table Title/Number</b>	<b>Table Description</b>	<b>Page Number</b>
Table 1 - Revision History	This table details each change that was made to this document, in order of date and section.	1 - 3
Table 2 - Technology Risks	This table details the technological risks that the team faces while working on this project.	7
Table 3 - Skill Risks	This table details the skill-based risks that the team faces while working on this project.	7
Table 4 - Bulk Data Insertion Objects	This table describes two of the objects used to transfer data between client- and server-sides while processing a Bulk Data Insertion. A GameData object transfers all relevant data, and contains a list of PlayerData objects, which contain data pertaining to each player.	15
Table 5 - Log Insertion Objects	This table describes how log data is stored in the database. A list of PlayerTurns contains all information, including several lists of PlayedCards, each of which also has a list of PlayerEffects.	15
Table 6 - log_game_round Database Table	This table represents how data from each log is stored in the PostgreSQL database. It describes each column name, its data type, and its purpose.	16
Table 7 - known_usernames Database Table	This table represents how username data for every player is stored in the database. It describes the column names, data types, and purposes.	16
Table 8 - Software Testing	This table goes into detail about what tests are run to insure that the finished website works as intended.	17 - 18
Table 9 - Team Profiles	This table gives an overview of the team members.	21

## Appendix C – Figures

<b>Figure Title/Number</b>	<b>Figure Title</b>	<b>Page Number</b>
Figure 1	System Architecture Diagram	9
Figure 2	Non-Log Based Data Wireframe	10
Figure 3	Data Upload Wireframe	10
Figure 4	Log Based Data Wireframe	11
Figure 5	User Information Wireframe	11
Figure 6	Bulk Data Insertion Flowchart	13
Figure 7	Log Data Insertion Flowchart	14