CSIS 3275 – Software Engineering

Term Project

Project Title: Development of an Online Gaming Social Network Web App

Student’s Name: Victor Correa Suleiman

Student’s ID: 300315653

Submitted to: Dr. Mahmood Al-Humaimidi

Douglas College – CSIS department New Westminster, BC, Summer 2020

# Introduction

## Project Description

The project is based on a social network website with the intent of connecting players to play games online, in order to reduce frustration and toxicity inside the online gaming community. The website is going to be named Ally.ca to demonstrate this teammate-finder intent and its functionality is going to be like this: users create an account, and after that, immediately create something called a Player Card. In it, they are going to choose various traits related to their profile: their age, location, languages, what games they play, what roles they play on each game, what are the heroes that they play the most, what is their playstyle (i.e. aggressive, defensive), if they like to communicate and how (i.e. text chat, voice chat), if they accept constructive criticism, if they get angry too fast with too little (commonly known as “tilt” level in the community), etc.

With their Player Card created, players can then see other players’ profiles via a browser feed. Then, they can filter which profiles they see by game, age, role, or any other trait that their Player Card has, in order to look for a teammate that resonates with their profile. Then, they like other Players Cards and, if they like them back, it’s a match, and they can talk to each other in order to go play together via a chat feature, which is the final objective of the web app.

### Rationale and Need for the Software

Investing in the videogame industry nowadays proved to be very profitable (with a 35% industry grow just in the first months of 2020) especially in the online multiplayer gaming field. However, that immense growth in popularity and profit doesn’t mean quality-of-life improvement for players. As new players come into online multiplayer and games become more complex, requiring teamwork and communication to win, situations that cause a lot of player frustration, often with unbalanced matches, start to become the norm and therefore, more relevant and more in need to be solved. If done right, this website would have the potential to reach the mainstream gaming community by making the players play with teammates that resonate more with their playstyle and profiles, making their in-game experience much more pleasant. Game companies would also like the website, as it would make a favour to them by making their player experience more enjoyable. It is a good opportunity to invest on, and a win-win for all the gaming community.

## Preliminary Investigation

**Limitations and Constraints –** The project would be residing inside the user’s web browser, so it needs to be designed with languages, frameworks and architectures that a web browser accepts. On a later iteration of the project, it can potentially be expanded to a mobile app to try to broaden the number of potential users.

**Costs and Benefits –** The costs will be basically on contracting staff (software engineers, full-stack web developers, UI/UX designers, etc.), a physical location for centralized development control, equipment for the staff to work on developing the web app, and of course, servers to store all data that will be used on the website, keeping an eye on maintenance costs as well. Money will be earned through ad-revenue and sponsors that believe in the website’s idea.

**Feasibility and Risks –** Nowadays, doing a social-media-based web app is very feasible, with lots of information and tips over the internet. The biggest risk will be if the revenue doesn’t pay off the costs nor generate profit or the stakeholders’ requirements are not met, so a healthy, growing community needs to be ensured to be built, and quantity, position and quality of the ads need to be discussed in order to mitigate those risks.

## Project Planning

### Project Deliverables

The construction of the project will follow an evolutionary process model. For the first iteration, the final deliverable project will be a web app prototype, with all the main features and modules, in which testers and end-users can sign-up for a beta phase, to stress-test server capacity and troubleshoot bugs and errors for the next iteration. These are the deliverables for each phase of the project in its first iteration:

- **Communication phase:** gathering all sponsors and/or start-up incubators that believe in our project after its pitch, so the team can have a budget to start hiring staff, renting a physical location and buying hardware, and also gathering all feasible and agreed-upon requirements that all stakeholders would like to have in the web app.

- **Planning phase:** planning and scheduling the project deliverables for each phase, as well as defining the project scope, to have an idea of what the constraints and deadlines of the project will be.

- **Modeling phase:** the deliverables of the modeling phase will be a series of diagrams that will describe how the requirements will be implemented inside the software and design models that will show a sketch of the architecture, interface, etc., all through UML diagrams, so the team will have a visual idea of how the web app will behaved and how it will be constructed and implemented.

- **Construction phase:** after having a solid knowledge of software requirements and design models, the team will solidify those models by building coded modules, interfaces, data structures and other components, and testing them to ensure a prototype for the web app is built in the end of this phase.

- **Deployment phase:** with a usable, tested prototype that fulfills its final purpose, with all its working components from server-side to client-side, the prototype is shipped in a beta-test phase for end-users to test. From this phase, a series of logs and observations must be taken to improve all areas of the web app: reported bugs, grabbing server processing performance, user reviews and feedbacks, etc., so the team can start patching bugs and resolving massive errors, in order to fully ship the product to end-users.

## Project Scope and Functionality

Because the project is a web app, it will rely on a web browser to download its front-end structure to it, so the website's constraints will need to consider average user download speed and memory in order to properly render the website without any delays or problems. Also, the servers in the server-side need to conform to the number and size of requests and posts they will receive, and that will need to evolve according to the number of users that are using the website.

As previously stated, the final objective of the web app is to connect players, so the data objects they will see will be their matches and their conversations with them. In between, players will also see player cards and filters in order to like them, and options regarding Player Card and account creation, update and deletion.

For the player to see this information, the web app will have 3 main functionalities: account creation, Player Card browsing and chatting with matches. First, the user will create an account. After authenticated by servers, he/she will need to create a Player Card in order to jump to the Player Card browser, filter Player Cards and like them, and then go to the chat page to talk to them so they can game together.

## Project Scheduling and Milestones

The project schedule is divided in two Gantt charts, the first one showing communication to modeling phases, and the other one showing construction and deployment phases with a review next to the middle of the construction phase. The project will follow an Agile principle, with a quick planning and modeling phase (10 weeks), and an emphasis on the construction and testing phase alongside the beta-deployment (26 weeks). The first project will begin on the Fall semester and is estimated to end on the Winter semester.

Figure 1 shows the first part of the schedule. The communication and planning phases happen together in order to align requirements to the plan, and the modeling phase go together with the planning phase so preliminary models can be drawn according to the plan, that will ideally be according to the requirements of all the stakeholders. Milestones are the red flags in the topside of the chart, showing the Project kickstart approval, then a consensus of requirements between all stakeholders so better and more accurate app designs can be modeled, and finally, the web app feasibility is approved and the construction phase begins.

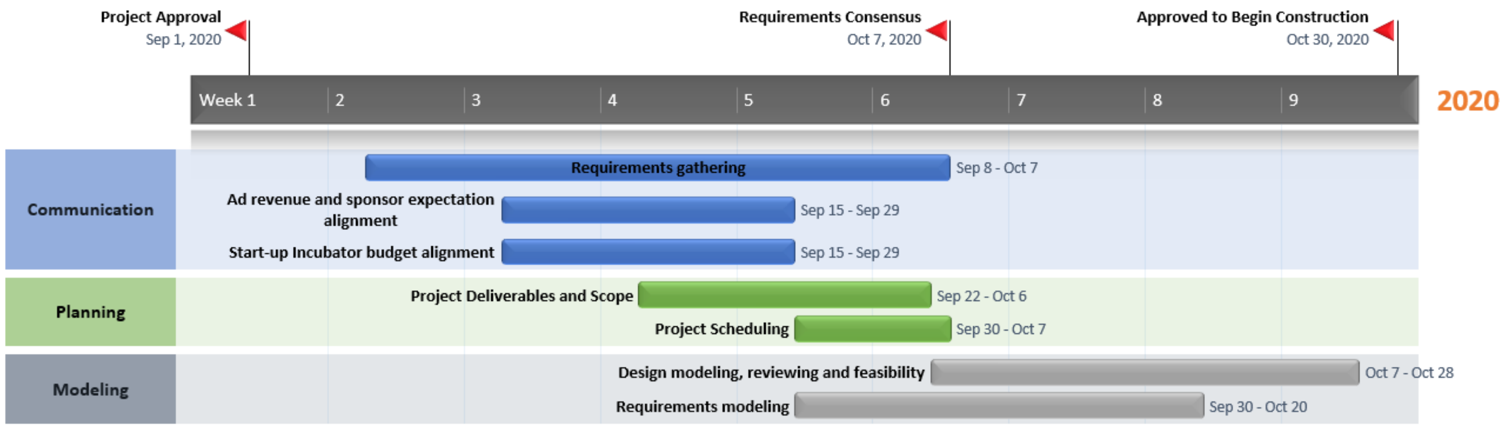


*Figure 1 - First Gantt Chart showing communication, planning and modeling phases.*

Figure 2 shows the second part of the schedule. The reviewing occurs in the middle of the buildings of the back-end and front-end, when the team analyzes the congruence between the requirements, the models and what they have already built in the date. The construction will emphasize the full-stack building and integration of the structure and UI of the web app, with an internal testing phase before being greenlit for end-user beta-testing (first milestone). After that, a team will be assembled to support and collect feedback from end-users, and the gathering and patching phase will follow for a little more than 2 months, to ensure the web app is properly ready for full deployment (second milestone).

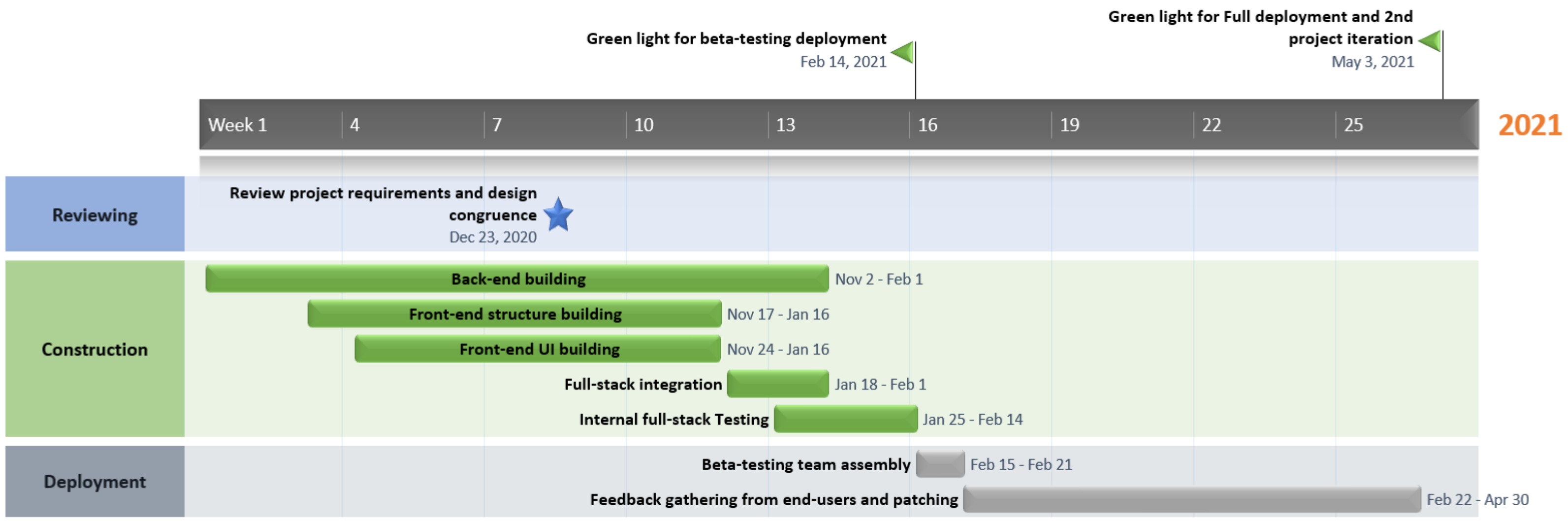


*Figure 2 - Second Gantt chart, showing construction and deployment phases.*

## Project Requirement Analysis

The preliminary communication was done by researching potential start-up incubators and sponsors and sending a short project proposal to them. Those that liked the scope of the project or at least got curious sent their interest of talking further to the team, and more conversation was done in order to gather the necessary support and budget from them. The team then found a pitch competition from a start-up incubator and, with a successful pitch, project proposal and with the support of the sponsors, the project got approved. All the stakeholders, that is, every person affected in some level by the web app, are listed below:

- Customer: the start-up incubator that believed in the team's project and makes the majority of the project's budget;

- Sponsors: all companies that will support the revenue of the project through ad revenue or other people that will make donations;

- Staff: every member of the team directly involved in the project. That includes but is not limited to software engineers, back-end, front-end and full-stack developers, customer support staff, UX/UI experts, server experts, network security experts etc.

- End-users: The final end-users that will actually use the web app, from internal testers to regular users.

It is crucial that all these members are involved in constant communication to elicit feasible requirements and, after the communication phase, they keep the feedback coming, so the team learns whether what they are modeling or building (depending on the phase) reflects the intended requirements.

## Requirements Elicitation

After discussing with all stakeholders and pitching the idea of the project to our clients, investors and potential end-users, we determined the three most important functional requirements of the web app will be:

- Account and Player Card creation: the first step for using the web app will be first creating an account, providing a username and password with email confirmation, and after that, inputting all the necessary information for player profile (or Player Card) of the user so the system can use it in its main functionality;

- Player Card browsing: This is the main part of the web app and represents the action of the primal idea of the project in the first place. With the profile created, the user will search for other profiles that he/she wants to connect to, using the profile traits of the other player cards, and liking them. Users will have a list of player cards they've liked and another list of player cards that liked their card, so they can decide which ones to like back and connect to;

- Chatting and connecting to other players: This happens after one user likes the other and the other likes back: then they can use a chat feature to start chatting with each other about whatever game subject they need to discuss, and add each other on their respective game platforms (offsite). They will also be able to add each other as friends inside the web app and see their activity.

And the two most important Non-Functional Requirements are:

- Emphasis on high-quality user interface: A quality attribute that the team believes is crucial to the project success, as gamers are really strict about user interface and experience and a lot of well-developed platforms like Discord also have that as a priority, so this is needed to keep up with their modern design as well. The website cannot have an outdated look and needs to have a sense of fluidity and immediacy for as many actions as possible, like clicking a button, scrolling through player cards, opening a chat tab, etc.

- Inclusion of advertisement from sponsors: This is another constraint that is highly important to work on and needs to have an important relationship to the user interface as well. Because the service will be free, website sponsors and investors need to have the opportunity to advertise their products on the website. Advertisements need to have adequate sizes on the page, positioning, and the number of ads per page needs to be thought of, to please the people who trust the team with their resources and attract more companies that believe in the website.

## Requirement Analysis Models

With the elicitation of requirements, preliminary requirement analysis models were built to support them and test the feasibility of each functionality.

### Scenario-based Models

The scenario for each functional requirement is a flow of actions the actor (or user) will perform inside said functionality. Below, a logical flow of interaction between user, website and server is described for each functional requirement.

**Account and Player Card Creation Scenario**

1. Potential user becomes aware of the existence of the website through a friend reference, advertisement, or another form of spreading the knowledge and becomes interested in it;
2. User checks out website's default home page, sees its features and decides to create an account. He then clicks the "sign up" button;
3. User chooses a username, a password, and uses his email for verification. He then clicks the "Submit" button;
4. New user data is then sent to the server-side to allocate space for a new username entry and allocates space for all player card attributes. It then sends a confirmation link to user's email;
5. User confirms account creation using his email, the server will authenticate his session and he will be redirected to the home page, which will say: "Your Player Card is empty. For teammates to find you, create a Player Card". He then clicks the "Create Player Card" button;
6. The page for creating the player card is loaded with all the attributes possible, like gender, age, games played, roles, playstyle, tilt and communication level, etc. User fills his/her Player Card with whatever traits deemed necessary and clicks the "Create" button;
7. Player Card is then created, all its data sent to user's entry at the server-side and then the user gets prompted if he/she wants to go to the player browser or wants to keep editing. The former redirects to the player browser, the latter goes back to the last step.

**Browsing and Finding Players Scenario**

1. Upon first use, a tutorial will appear telling the user how the player browser works. User can go through the tutorial or skip it entirely;
2. After that, the user will see all the player cards the server can provide to him, without any filters (and advertisements in between them sometimes). User then will filter cards by any traits available, such as age, gender, games, roles, playstyle, etc.;
3. User will click the like button on any Player Cards he/she is interested on. Also, any other users that liked the user's Player Card can be seen on the website's notification menu, with the choice of immediately liking them back. They can also click on the username of the players to go to their page to see activity history and more details about them;
4. It is important to also clarify that the server will keep being updated on the fly with the user's likes and matches and providing the filtered Player Card list the user requests.

**Chatting and Connecting to matches Scenario**

1. In the chat page, a menu with all conversations will be opened, with all the user's matches with other players who liked them back. He is notified about new messages. He then clicks on a match chat room;
2. The chat opens, with its history of messages. He can click the textbox at the bottom to write a message and click "enter" to send it;
3. Server will log all messages and interactions between users, to keep message history updated;
4. Users also have the choice to delete a chat and block another user, and the server will also update its list of blocked/unmatched users.

Using the use-case formal template, more information is produced to go alongside the scenarios already written, so a UML Use-case diagram can be produced for each scenario.

**Account and Player Card Creation Use-case Diagram**

Primary Actor: User

Secondary Actor: Back-end Server

Preconditions: Intention of creating a new account to use the web app

Triggers: potential user clicks the "Sign up" button on the homepage

Frequency of Use: Frequent

Channels: user browser for User and server host and databases for Back-end server

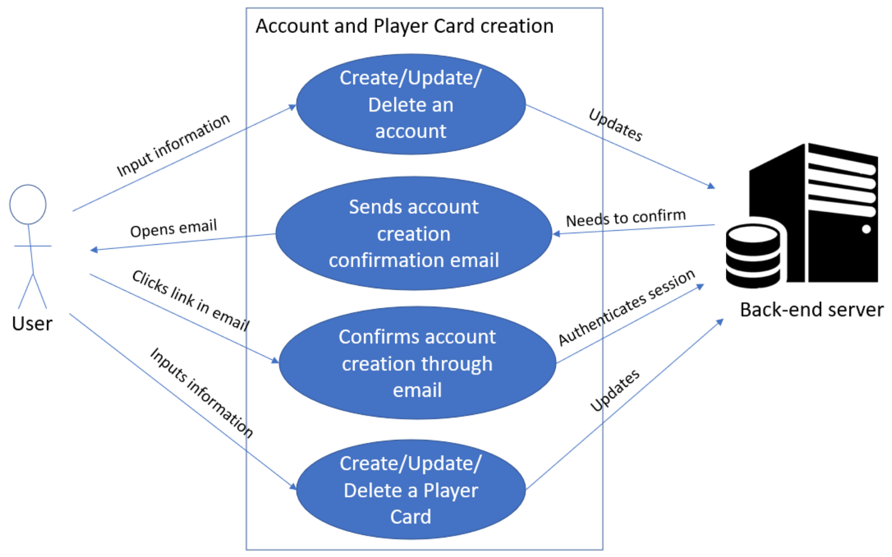

Account and Player Card creation
Create/Update/ Delete an account
Sends account creation confirmation email
Confirms account creation through email
Create/Update/ Delete a Player Card
Input information
Opens email
Clicks link in email
Inputs information
Updates
Needs to confirm
Authenticates session
Updates
User
Back-end server

*Figure 3 - Use-case Diagram for Account and Player Card Creation*

**Player Card browsing Use-case Diagram**

Primary Actor: User

Secondary Actor: Back-end Server

Preconditions: User Account created and Player Card already configured

Triggers: User clicks the "Player Card Browser" button, or gets redirected to the Player Card Browser page

Frequency of Use: Extremely Frequent

Channels: user browser for User and server host and databases for Back-end server

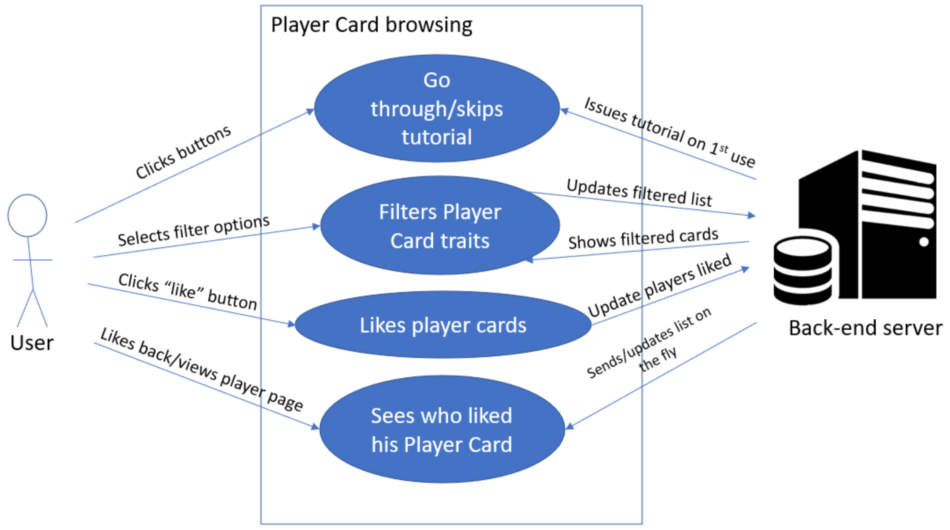


*Figure 4 - Use-case diagram for Player Card browsing*

**Chatting with matches Use-case Diagram**

Primary Actor: User

Secondary Actors: Back-end Server and Match

Preconditions: User matches with another user by liking a Player Card back or vice-versa

Triggers: User clicks the "Chat" button, or clicks on anything that redirects to the chat page, like a message notification

Frequency of Use: Highly Frequent

Channels: user browser for both User and Match and server host and databases for Back-end server

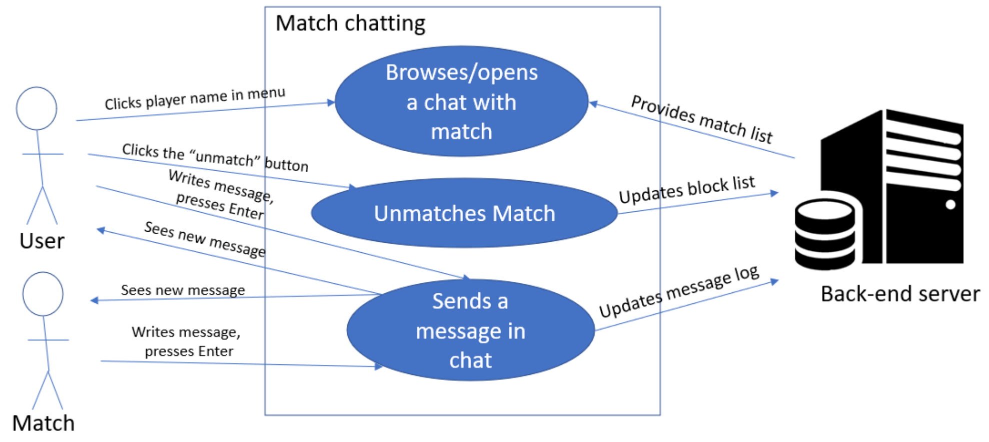


*Figure 5 - Use-case diagram for Chatting with matches*

**Activity and Swimlane Diagrams for Account and Player Card Creation Scenario**

Regarding the account creation use-case scenario, the team felt that creating a diagram showing all the events and conditions of the activity flow would better elicit the interaction between the actor and the system. Therefore, the team created an Activity Diagram for the scenario, showing how the system operates in relation to what the actor does. Furthermore, a Swimlane Diagram was created based on the former diagram that shows which actor does what with the system, detailing it even more. Below is the Activity and Swimlane Diagrams, respectively, for the Account and Player Card creation scenario.

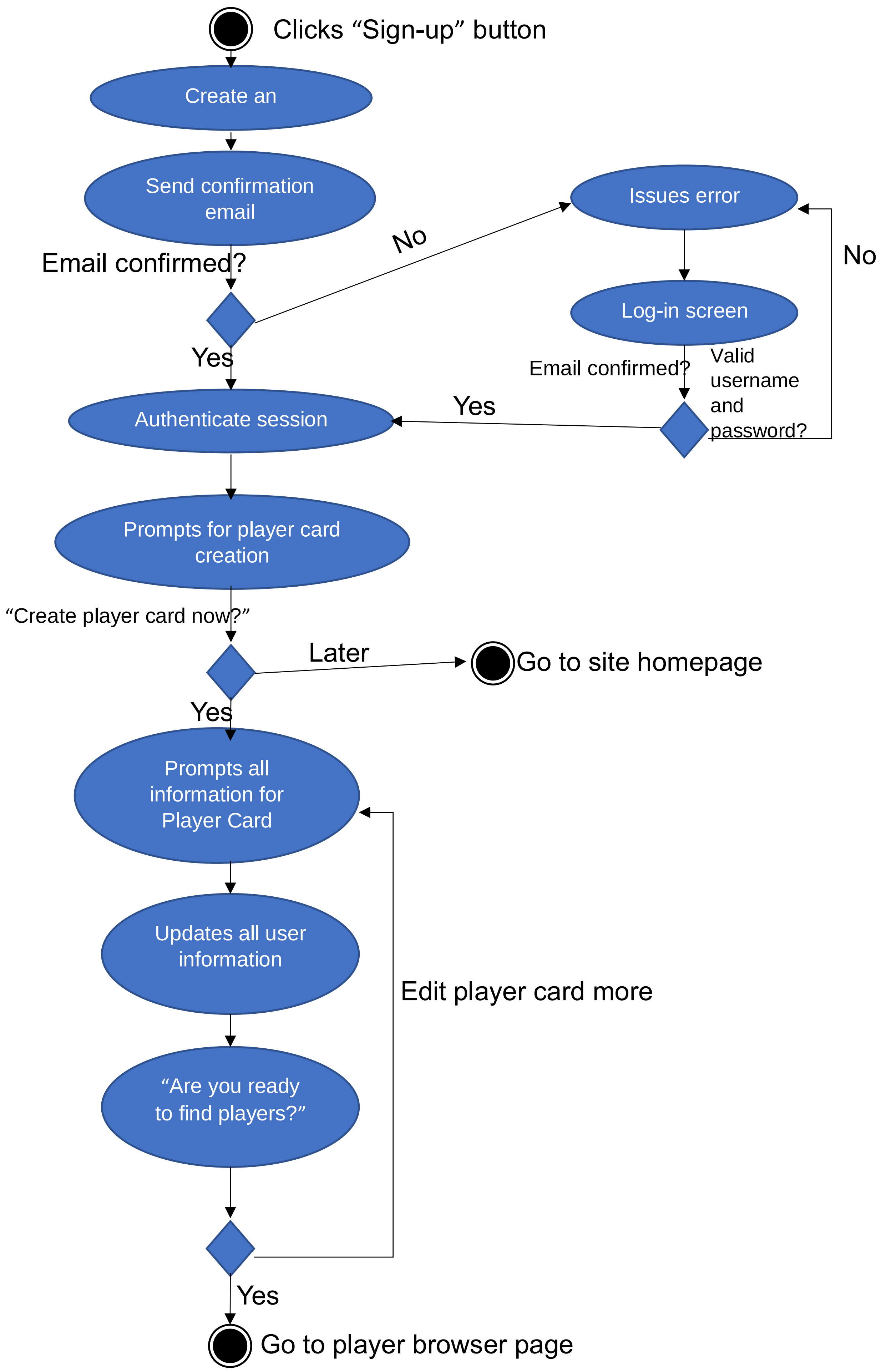

Clicks “Sign-up” button
Create an
Send confirmation email
Email confirmed?
No
Issues error
No
Log-in screen
Yes
Email confirmed?
Valid username and password?
Authenticate session
Yes
Prompts for player card creation
“Create player card now?”
Later
Go to site homepage
Yes
Prompts all information for Player Card
Updates all user information
Edit player card more
“Are you ready to find players?”
Yes
Go to player browser page

*Figure 6 - Activity Diagram for the account and Player Card creation scenario.*

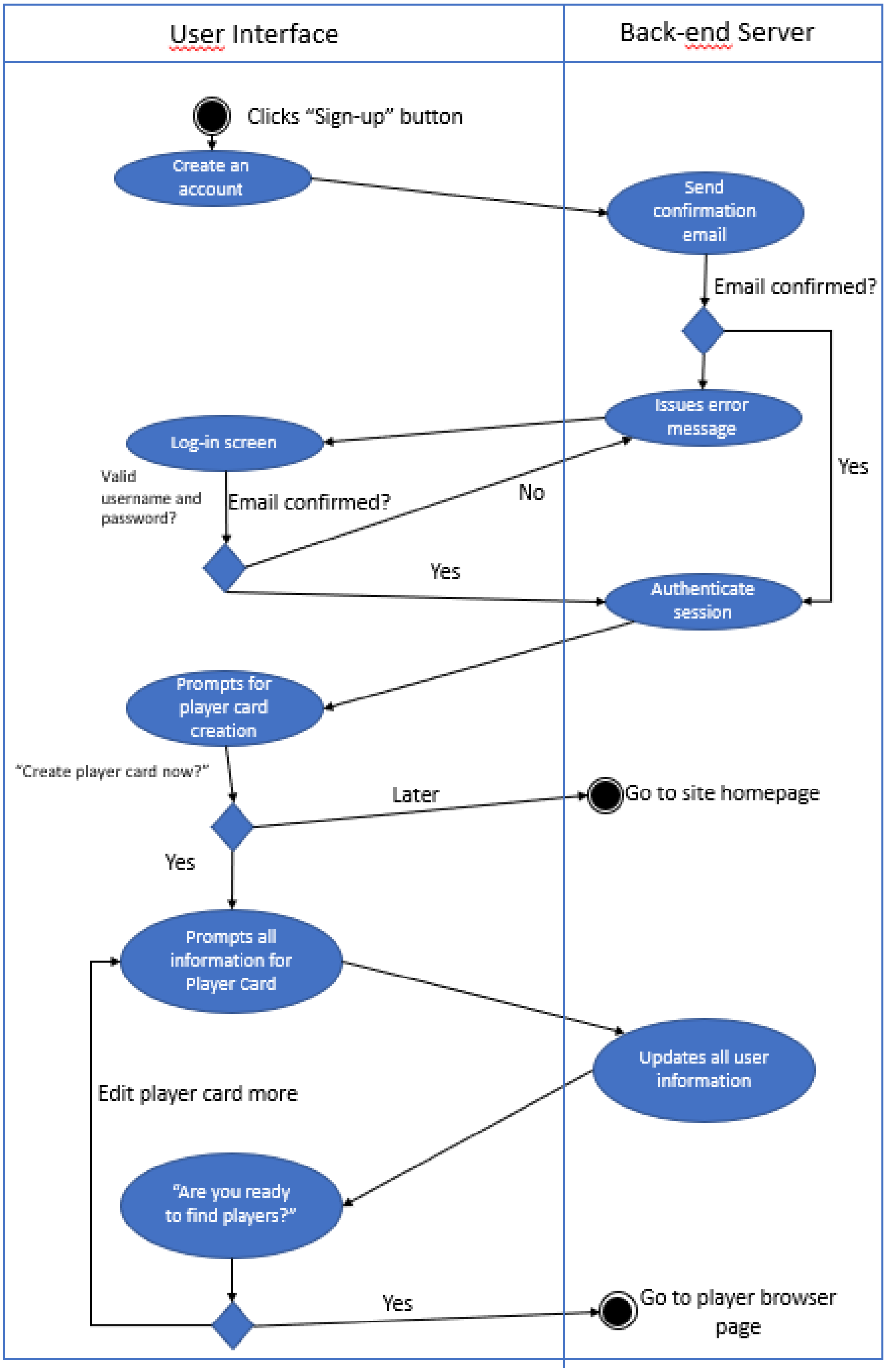


*Figure 7 - Swimlane Diagram for the account and Player Card creation scenario.*

## Class-based and Data Models

Thinking of an object-oriented design, the team deemed necessary to brainstorm what the important classes of the system would look like, how they connect together, their attributes and methods. Below is a preliminary Class Diagram for the web app. Briefly explaining, a User can have only one Player Card (at least on early development). Each Player Card can have one or more games in it, for the Player Card to be more visible by more people. Finally, a User can have none or multiple Matches, which technically are also users but with different attributes seen by the User, as seen below. Because of it, a match will also have only one Player Card. All the early attributes and methods that were thought of are on the class cards in the diagram.

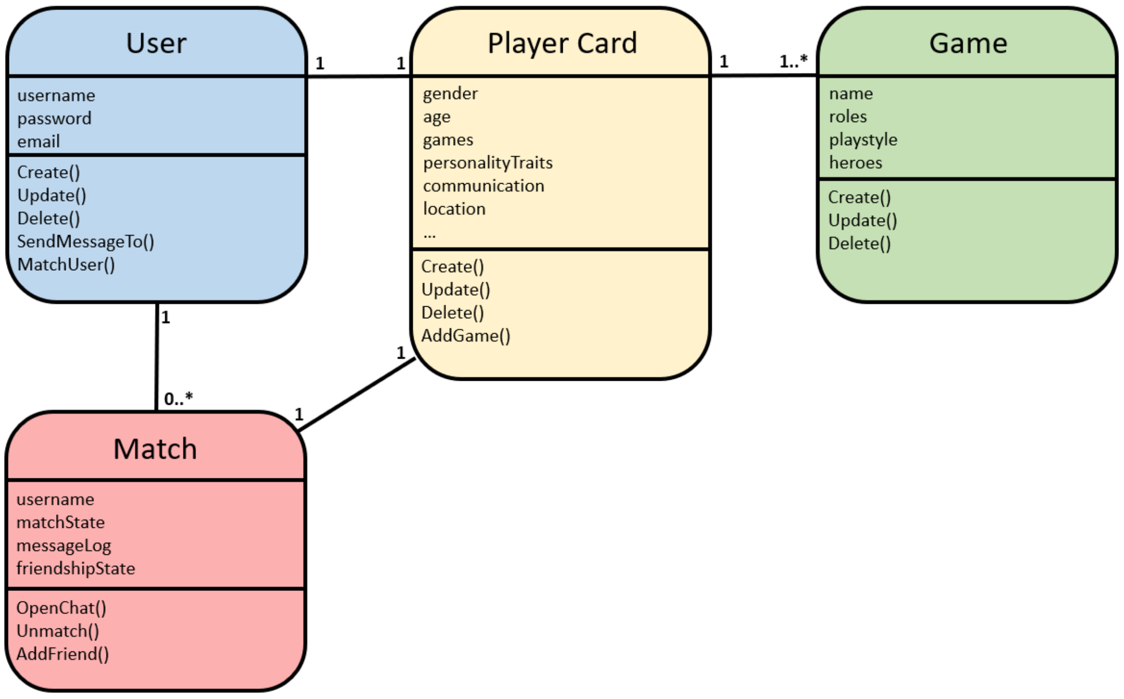


*Figure 8 - Preliminary Class Diagram for the web app.*

## Behavioral Models

To test how the web app would react to the user interaction, and also to demonstrate its states in relation to what the user does, three State Transition Diagrams were created, one for each requirement. In them, states are represented by the blue rounded rectangles, system events are written on top of the arrows, and the black circles represent the diagram transition from one feature or module to the other (in that case, each module is thought of being a page inside the website).

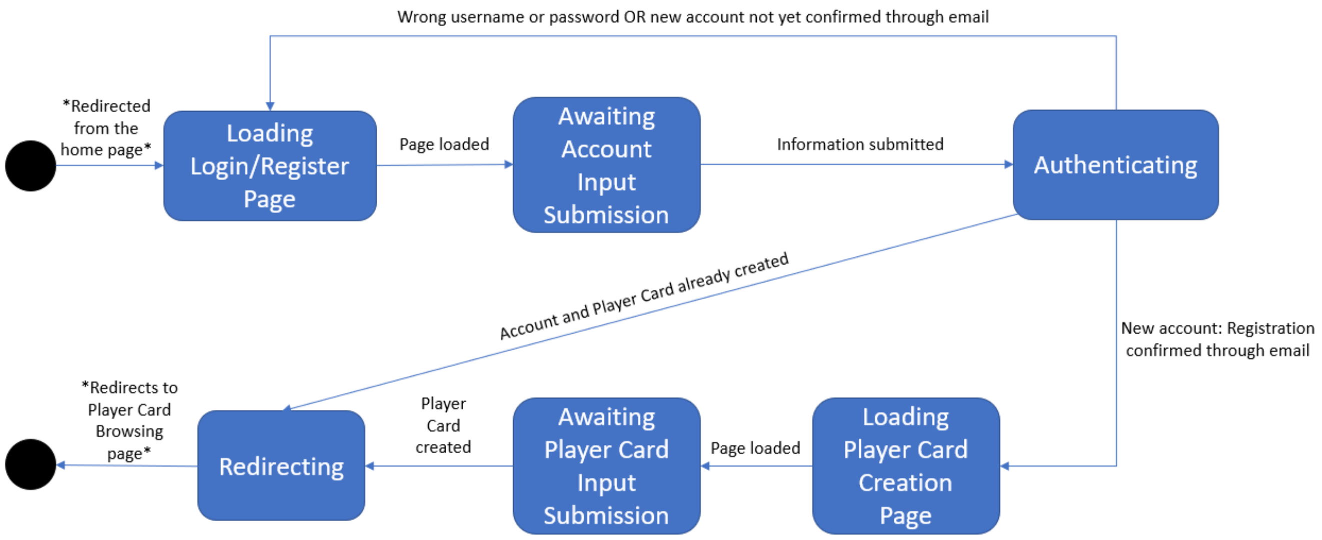

Wrong username or password OR new account not yet confirmed through email
*Redirected from the home page*
Loading Login/Register Page
Page loaded
Awaiting Account Input Submission
Information submitted
Authenticating
Account and Player Card already created
New account: Registration confirmed through email
*Redirects to Player Card Browsing page*
Redirecting
Player Card created
Awaiting Player Card Input Submission
Page loaded
Loading Player Card Creation Page

*Figure 9 - State Transition Diagram for Account Creation.*

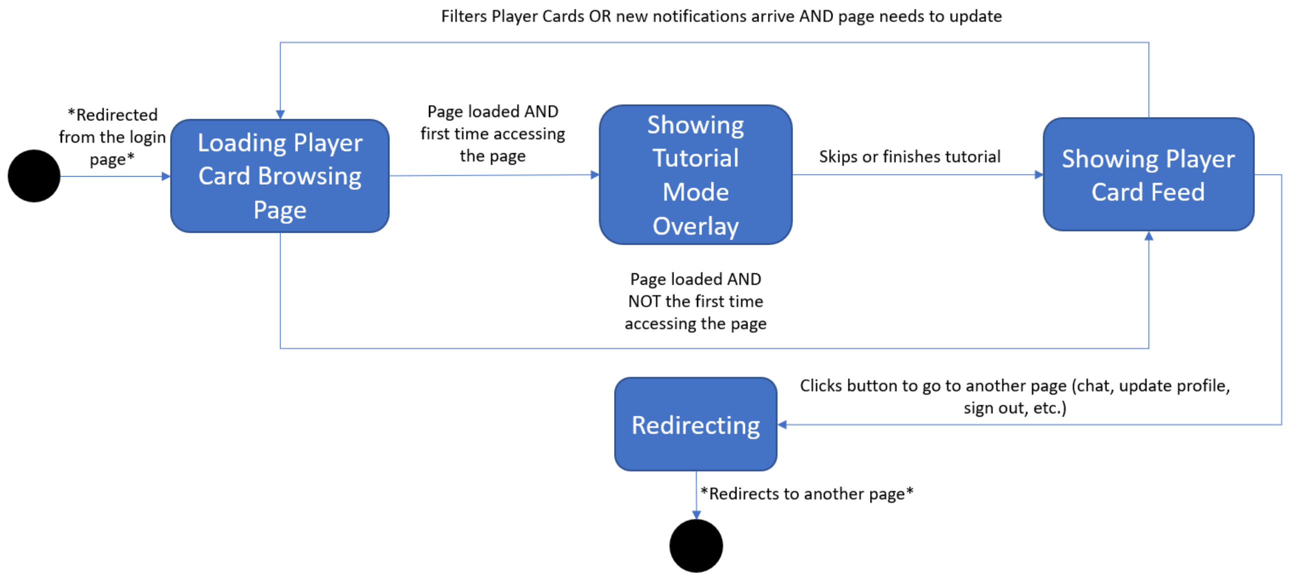

Filters Player Cards OR new notifications arrive AND page needs to update
*Redirected from the login page*
Loading Player Card Browsing Page
Page loaded AND first time accessing the page
Showing Tutorial Mode Overlay
Skips or finishes tutorial
Showing Player Card Feed
Page loaded AND NOT the first time accessing the page
Redirecting
Clicks button to go to another page (chat, update profile, sign out, etc.)
*Redirects to another page*

*Figure 10 - State Transition Diagram for Player Card Browsing.*

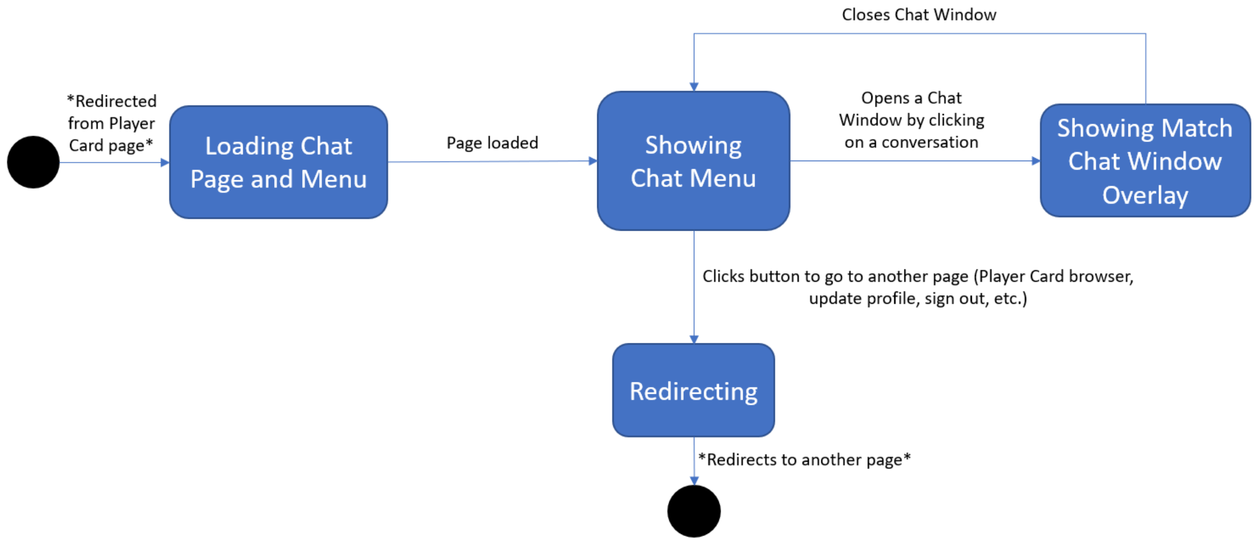

Closes Chat Window
*Redirected from Player Card page*
Loading Chat Page and Menu
Page loaded
Showing Chat Menu
Opens a Chat Window by clicking on a conversation
Showing Match Chat Window Overlay
Clicks button to go to another page (Player Card browser, update profile, sign out, etc.)
Redirecting
*Redirects to another page*

*Figure 11 - State Transition Diagram for Match Chatting.*

## Project Design Modeling

After all the requirements were consolidated and requirement models were drawn, the modeling phase started in order for the team to have a solid, less abstracted idea of what the software would look like, the tools that were needed to be used in order to conform to its constraints, and its overall structure.

### Data and Class Design Models

Because the project is a web application, the team decided to use Visual Studio Code as the development environment, with coding languages including but not limited to HTML, CSS and Javascript with JQuery AJAX. The refined class diagram is represented in Figure 12 below. The diagram also indicates the format of each attribute. Also, it is worth noting that these attributes will be included in the Javascript main functions, but there will be also other types of data structures. For instance, the data that will come from the server will be on a JSON format (sample represented in Figure 13) and will be stored in a database that will use SQL statements. The web app will then use the Javascript inside the page to transform this data structure into an instanced object.

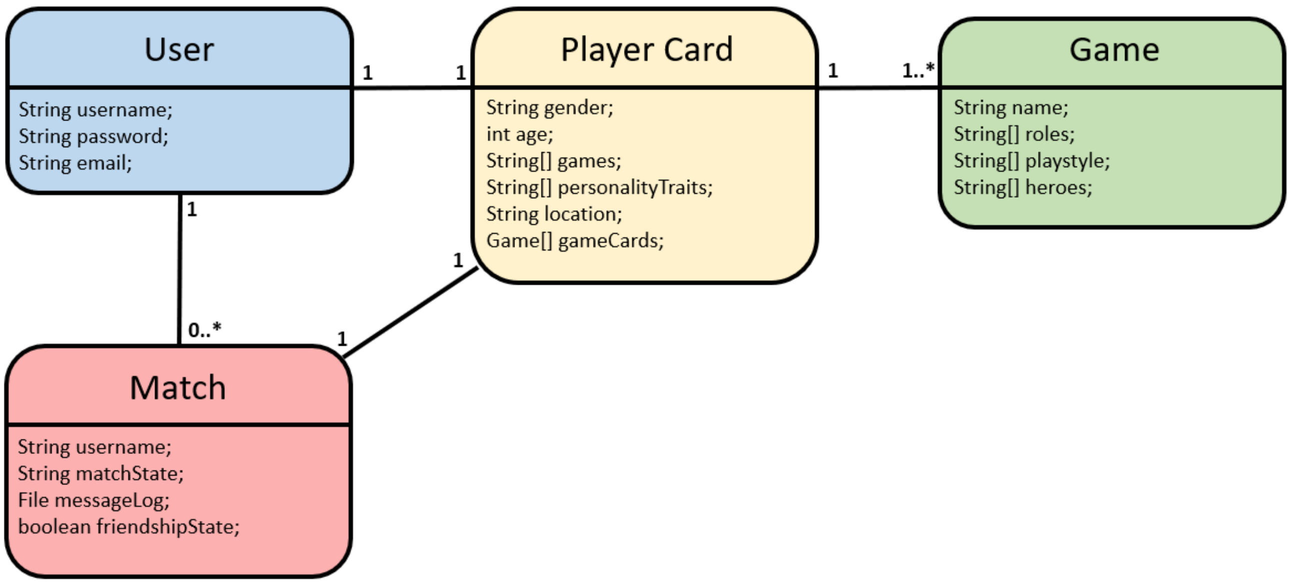
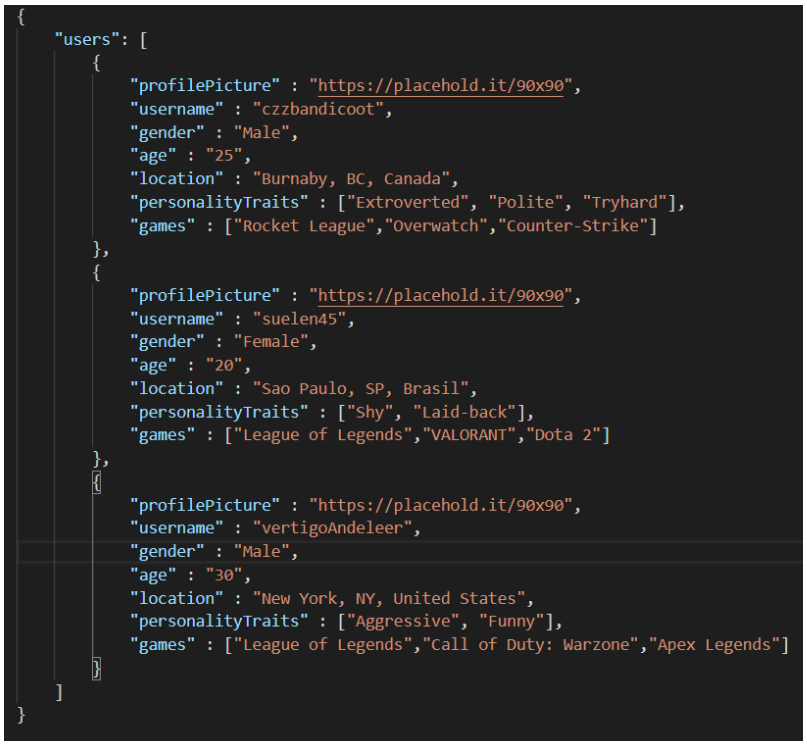


*Figure 12 - Refined class diagram for the web app.*



```
{
    "users": [
        {
            "profilePicture" : "https://placehold.it/90x90",
            "username" : "czzbandicoot",
            "gender" : "Male",
            "age" : "25",
            "location" : "Burnaby, BC, Canada",
            "personalityTraits" : ["Extroverted", "Polite", "Tryhard"],
            "games" : ["Rocket League","Overwatch","Counter-Strike"]
        },
        {
            "profilePicture" : "https://placehold.it/90x90",
            "username" : "suelen45",
            "gender" : "Female",
            "age" : "20",
            "location" : "Sao Paulo, SP, Brasil",
            "personalityTraits" : ["Shy", "Laid-back"],
            "games" : ["League of Legends","VALORANT","Dota 2"]
        },
        {
            "profilePicture" : "https://placehold.it/90x90",
            "username" : "vertigoAndeleer",
            "gender" : "Male",
            "age" : "30",
            "location" : "New York, NY, United States",
            "personalityTraits" : ["Aggressive", "Funny"],
            "games" : ["League of Legends","Call of Duty: Warzone","Apex Legends"]
        }
    ]
}
```

*Figure 13 - Data structure for responses from the database server in the JSON format.*

## Architectural Design Models

For the architecture of the web app, the team focused on the context that it will be inserted and a list of archetypes, so a broader structure can be drawn to help developers understand each functionality.

### <u>Context Diagram</u>

The context diagram helped builders understand the constraints of the system and who and what interacts with it, as well as its peers and sub/super systems. For the Context Diagram in Figure 14, the actors are essentially the users and testers (or builders). The systems that the Web App depends on are the software that will build the app itself (in that case, the aforementioned Visual Studio Code as an IDE), and the software that will render the Web App, which is the user's web browser. For the Web App to be complete, it also needs to use the Player Card and Account data stored in the database server, so it is its subordinate.

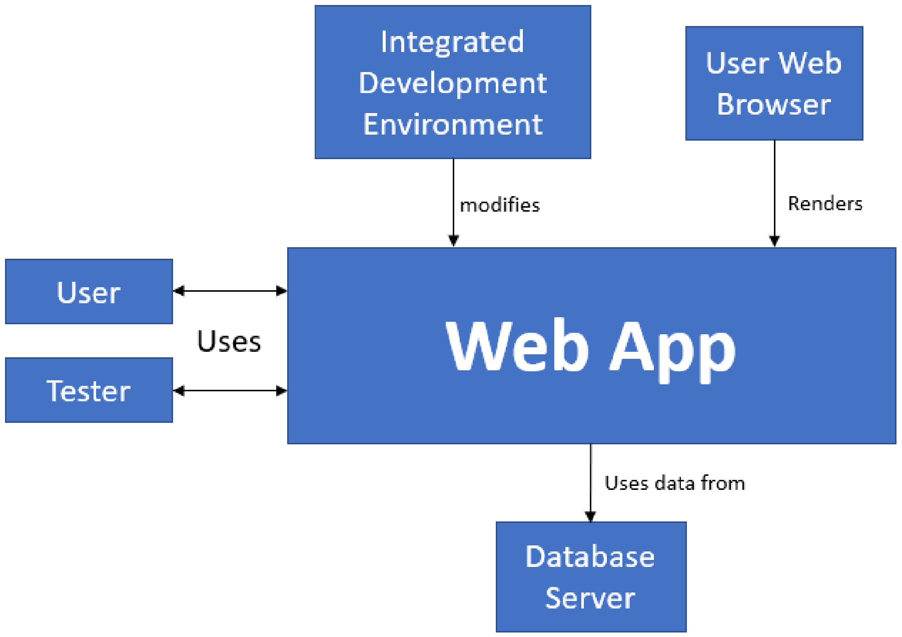


*Figure 14 - Context Diagram of the Architectural model for the web app.*

## **Archetypes and Archetype Diagram**

The archetypes help the team to understand the overall structure of the web app as a whole. They are abstract concepts that help build a hierarchy diagram, forming its preliminary architecture. Regarding the archetypes, they are:

- Server – represents the database or authenticating server, that will provide data for the App;
- Data – the data itself that the server will provide and will be included on the page;
- Structure – this represents the structure of the website, the HTML code that will contain the user interface and the data, divided into elements;
- Style – represents the code for the aesthetic aspects of each element of the page, using CSS and add-ons;
- Page – Structure, Style, and Data together will render the Page, which is what the user will interact with in order to properly use the app's features;
- App – a combination of the usage of several Pages (in a low level of abstraction, Player Card browsing page, match chat page, etc.) will result in the proper usage of the software's features, therefore, the final App.

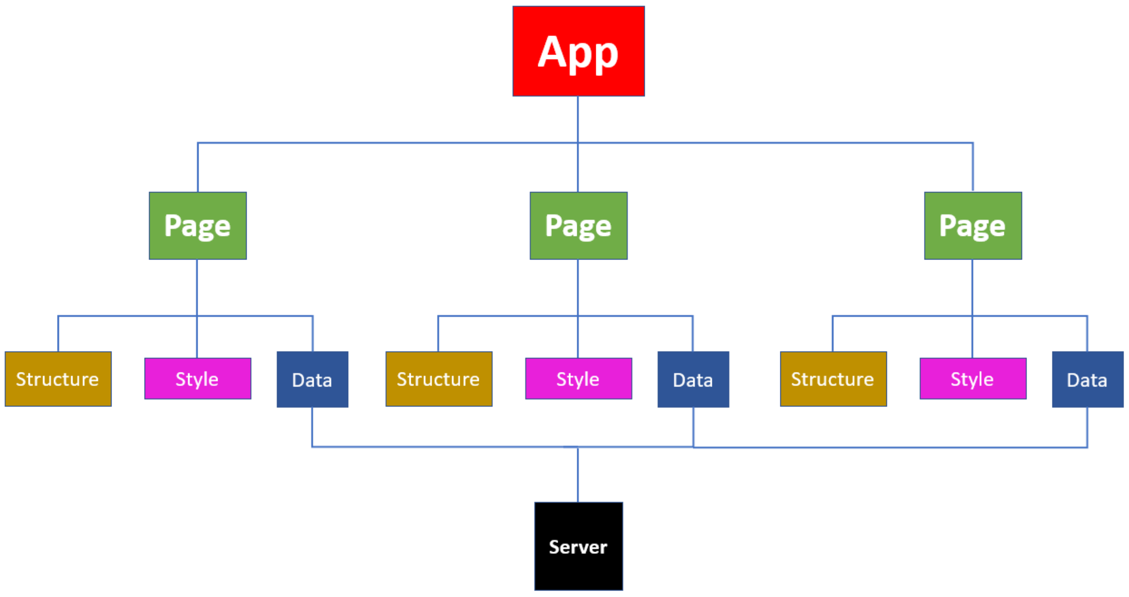
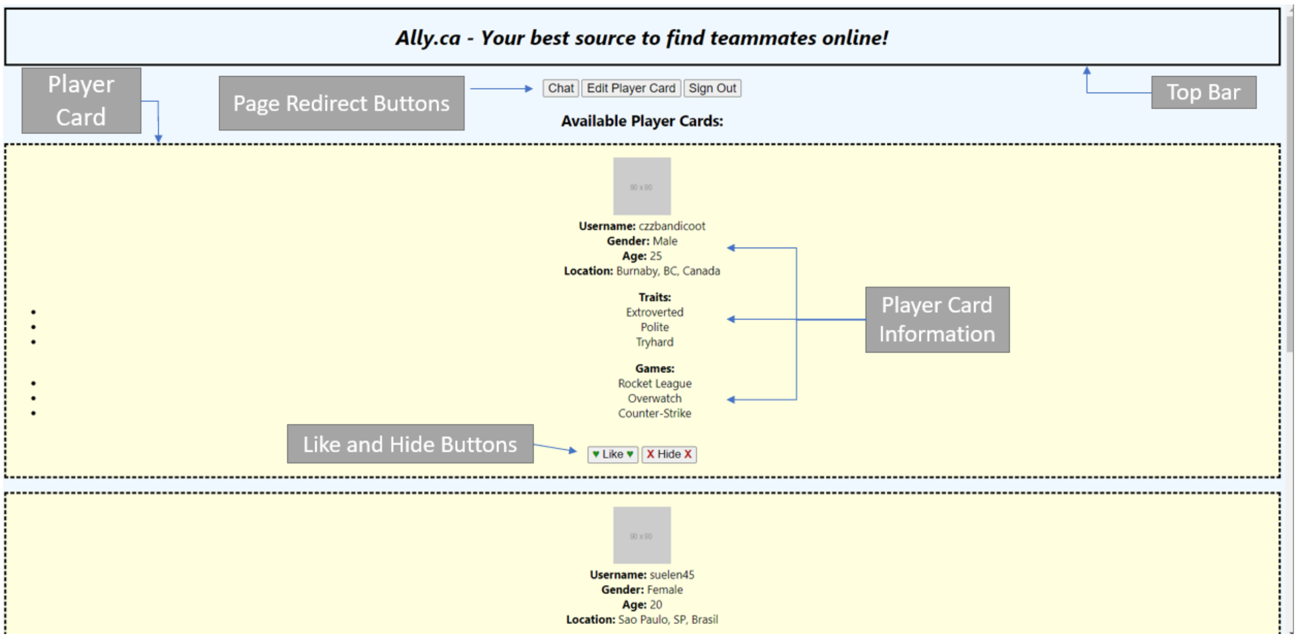


*Figure 15 - Archetype Diagram of the Architectural model for the web app.*

## Interface Design Models

For the interface models, the Player Card Browsing and Match Chatting features were prioritized and thought of as fluid, visual and easy to use. Samples of the user interface for some of these are represented in figures 16 and 17. The solid grey rectangles and blue arrows are not part of the interface and indicate user objectives and elements for the interface. Figure 8 represents a prototype for the interface in the Player Card Browsing feature made with HTML and CSS, while figure 9 also represents a prototype interface for the Match Chatting feature made with Windows PowerPoint Symbols.



**Username:** czzbandicoot
**Gender:** Male
**Age:** 25
**Location:** Burnaby, BC, Canada

**Traits:**
Extroverted
Polite
Tryhard

**Games:**
Rocket League
Overwatch
Counter-Strike

**Username:** suelen45
**Gender:** Female
**Age:** 20
**Location:** Sao Paulo, SP, Brasil

*Figure 16 - User Interface prototype for the Player Card Browsing feature.*

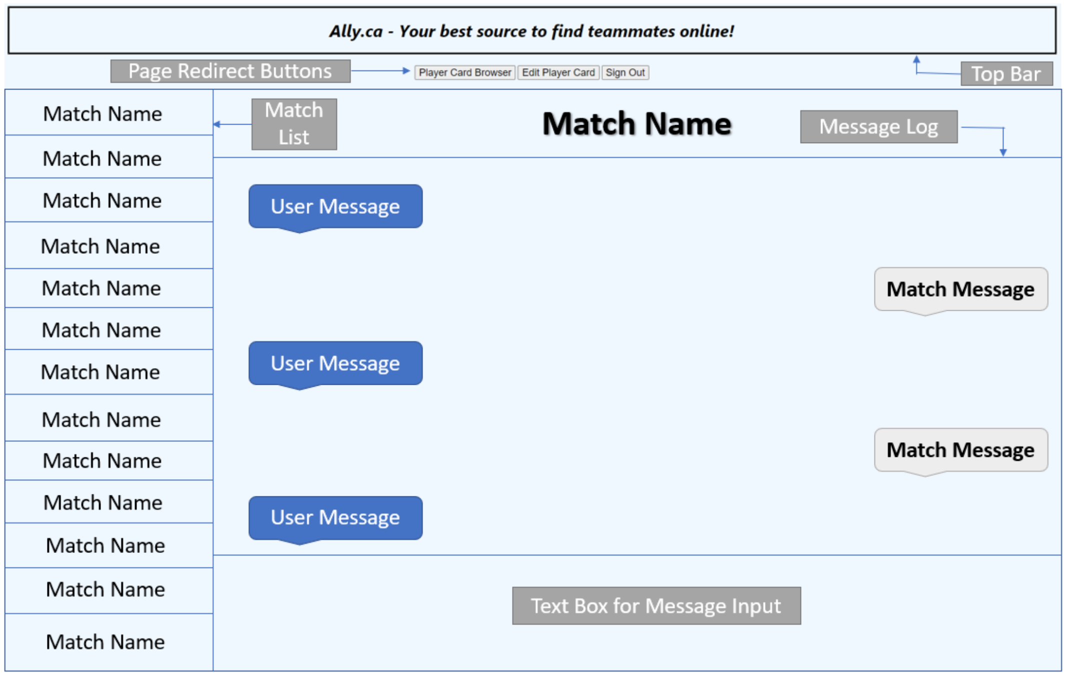


*Figure 17 - User Interface prototype for the Match Chatting feature.*

## Component Design Models

The last model of the modeling phase is the Component Design Model, which identifies what are the components (or modules) that the system needs to have to properly function, what is their hierarchy and how they are connected. Figure 18 represents the component diagram for the Component Design Model for the web app. Briefly explaining, the main component and intention of the website as whole is to connect users through it so they can meet each other and play online games together. That comes with user matching and a chat feature. To match users, one has to create an account, a Player Card and go to the Player Card Browser to like Player Cards and be liked. That is backed up by account authentication, the server always updating information to match users and redirecting the user to where he/she wants to go through the website.

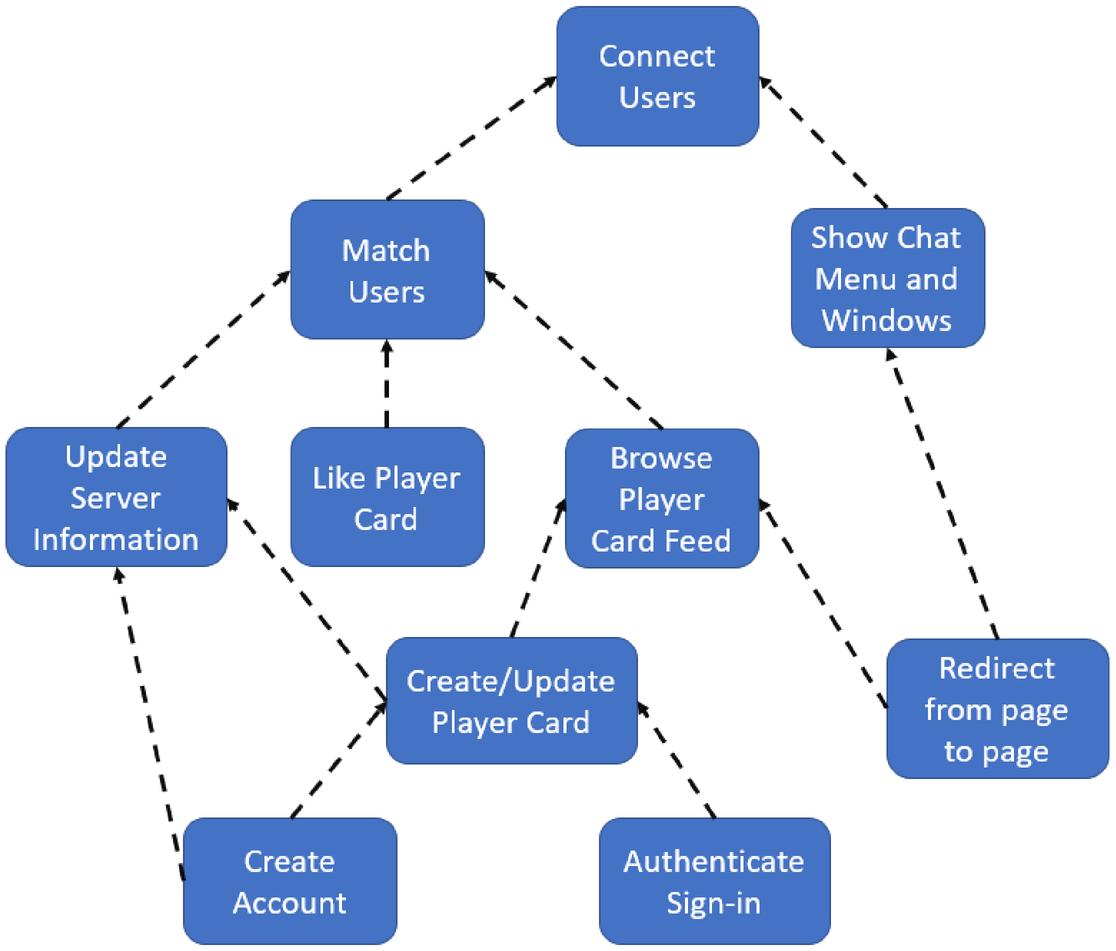

Connect Users
Match Users
Show Chat Menu and Windows
Update Server Information
Like Player Card
Browse Player Card Feed
Create/Update Player Card
Redirect from page to page
Create Account
Authenticate Sign-in

*Figure 18 - Component Diagram for the web app.*

## Software Construction

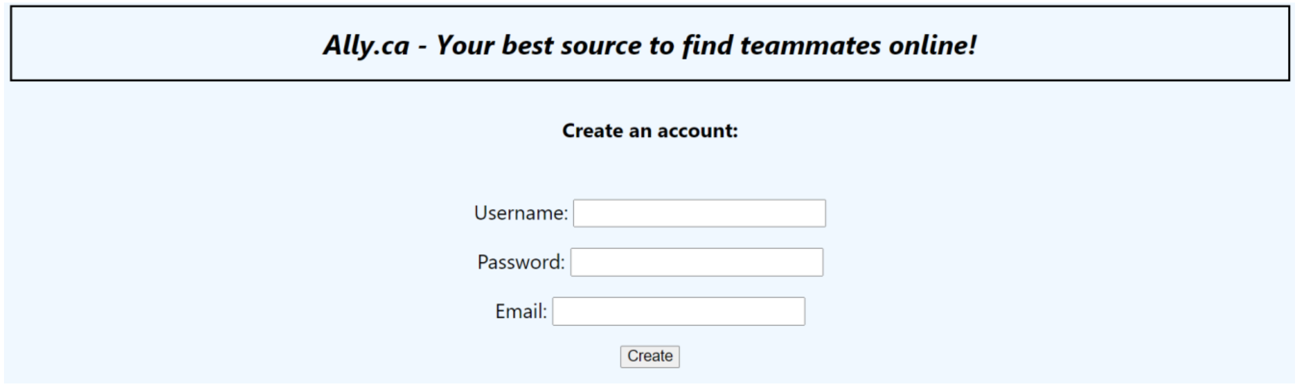
The team just started the construction phase of the project and came up with the three prototypes for the main functionalities already mentioned on the sections above. This report will include all main Javascript codes for each of the three pages. If you wish to see the HTML and CSS structure, you can clone the repository at https://github.com/correasuleiman-v5653/mySEJavaProject. All codes were written using Visual Studio Code.

### Account Creating Code

Below is the code for the account creating module, the class User, and its interface, respectively. It reads the entries written in the fields and instantiates a User object with them. The User class has the same attributes and types as figure 12, and the main class has the same ones but they are bound to the HTML element in order to retrieve the desired information, along with an account attribute to instantiate a new User object.

```
//code for the main account creating class
function submitAccount(){
    //grabbing values from the page
    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;
    let email = document.getElementById("email").value;

    //instantiating a new User object
    let account = new User();

    //putting the values to each of its attributes
    account.setUsername(username);
    account.setPassword(password);
    account.setEmail(email);

    //resetting the textboxes in the page
    document.getElementById("username").value = "";
    document.getElementById("password").value = "";
    document.getElementById("email").value = "";

    //alerting user to check email for confirming account
    alert("We've sent an email to confirm the account creation. Please confirm it s
o you can start creating your Player Card!");
}
```

```
//code for User class. This is the closest one from the models drawn on the modelin
g phase.

class User{
    //defining attributes. The constructor here is default
    _username = "";
    _password = "";
    _email = "";

    //setters to put page values onto the instatiated object
    setUsername(username){
        this._username = username;
    }

    setPassword(password){
        this._password = password;
    }

    setEmail(email){
        this._email = email;
    }
}
```

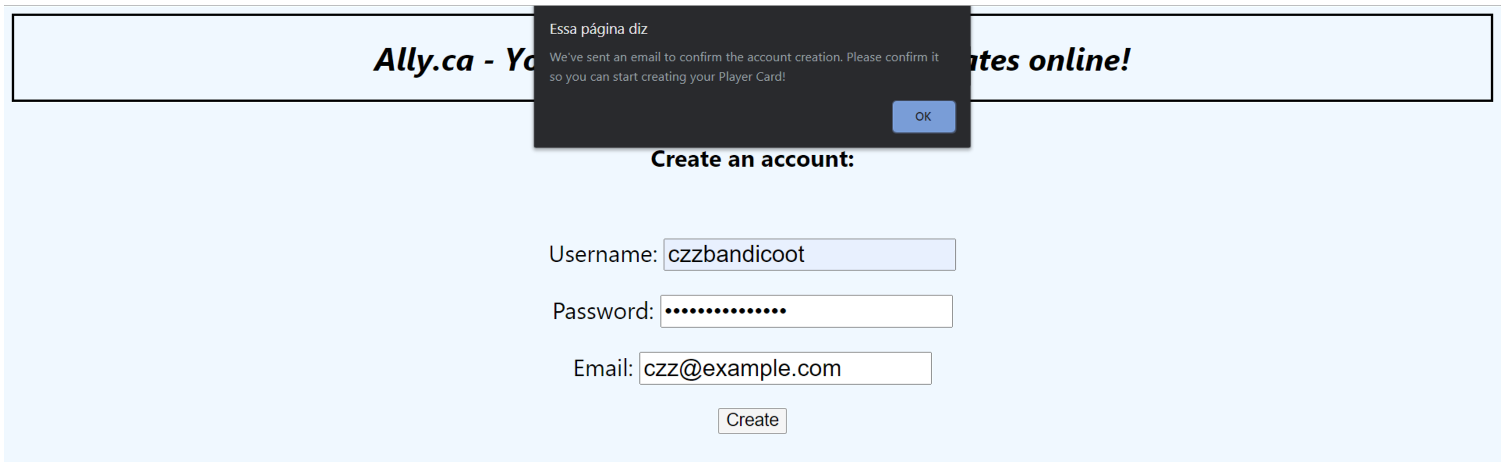


**Ally.ca - Your best source to find teammates online!**

**Create an account:**

|  |  |
|---|---|
| Username: |  |
| Password: |  |
| Email: |  |

*Figure 19 - Account creating interface just as the page loads.*



**Ally.ca - Yo**… **tes online!**

Essa página diz

We've sent an email to confirm the account creation. Please confirm it so you can start creating your Player Card!

**Create an account:**

|  |  |
|---|---|
| Username: | czzbandicoot |
| Password: | ••••••••••••••• |
| Email: | czz@example.com |

*Figure 20 - Account creating interface when the input is submitted.*

## Player Card browsing Code

Below, written in Javascript jQuery functions, is the code for the main function for the Player Card Browsing module, respectively. Like said previously, its interface can be seen on figure 16. The purpose of the main class is to grab a response from the web server (in that case, the file is stored inside the repo) in a raw file in the JSON format, and for each user inside the file, create a player card element and append the user's data inside it. Its main attributes are:

- "users": array for storing the JSON file;
- "pc": Player Card object to store each user data;
- "pcHTML": String for creating the HTML element

Besides the class "Main.js", there are two entity classes that were used to create the objects. One of them is "userData.json", which is a JSON file that contains raw data from the server's response (Figure 5 is in fact a snippet of this class), and the other one, also written in Javascript, is the "Player-card.class.js", which is below the Main.js class. This one is used to properly pick up the attributes in the JSON file and store them into variables using this class, so that the main function can create an object with those attributes, which are:

- "_profilePicture": String to store the user's profile picture URL;
- "_username": String to store the user's username;
- "_gender": String to store the user's gender;
- "_age": Integer to store the user's age;
- "_location": String to store the user's location;
- "_personalityTraits": Array of Strings to store the user's personality traits;
- "_games": Array of Strings to store the games that the user plays.

And its methods are just getter functions that will return each of these attributes, so the attributes themselves are encapsulated inside the class.

Because this class in particular uses AJAX features, if you wish to see the website working, you need to pull the files from the repo, open it in Visual Studio Code and open the HTML file with a Live Server using an extension, so it can use the AJAX features and show the site properly. If you just execute it with the web browser, the Player Cards will not appear.

Because this is the first primitive prototype for this feature, the array of Game objects that is represented in Figure 12 is still not created.

```
//code for the main for the Player Card Browsing feature
$(document).ready(function(){

    //Getting the JSON response from the server
    $.getJSON("userData.json", null, function(data){
        let users = data.users;

        //Loop - "do this for each user in the userData.json file"
        $.each(users,function(i,user){

            //Creating a PlayerCard object to store the user's data that is inside
the JSON file
            let pc = new PlayerCard(user.profilePicture,user.username,user.gender,u
ser.age,user.location,user.personalityTraits,user.games);
```

```
            //Creating an HTML element to display the user's information on the pag
e feed
            let pcHTML = "";

            pcHTML += "<div id = \"player-card\">";
            pcHTML += "<img src = \"https://placehold.it/90x90\" alt = \"profile pi
cture\"> <br>";
            pcHTML += "<b>Username:</b> " + pc.getUsername() + "<br>";
            pcHTML += "<b>Gender:</b> " + pc.getGender() + "<br>";
            pcHTML += "<b>Age:</b> " + pc.getAge() + "<br>";
            pcHTML += "<b>Location:</b> " + pc.getLocation() + "<br>";
            pcHTML += "<ul><b>Traits:</b> ";

            let traits = pc.getPersonalityTraits();
            for (let i = 0; i < traits.length; i++){
                pcHTML += "<li>" + traits[i] + "</li>";
            }
            pcHTML += "</ul>";

            pcHTML += "<ul><b>Games:</b>"
            let games = pc.getGames();
            for (let i = 0; i < games.length; i++){
                pcHTML += "<li>" + games[i] + "</li>";
            }
            pcHTML += "</ul><br>";

            //Creating "like" and "hide" buttons for the Player Card
            pcHTML += "<button><span id = \"heart\">♥</span> Like <span id = \"hear
t\">♥</span></button> <button><span><b>X</b></span> Hide <span><b>X</b></span></but
ton>";

            pcHTML += "</div><br>";

            //Appending the Player Card to the appropriate element inside the page
            $('#player-cards').append(pcHTML);
        });
    });
});
```

```
//code for the Player Card class
class PlayerCard{
    //defining attributes
    _profilePicture = "";
    _username = "";
    _gender = "";
    _age = 0;
    _location = "";
    _personalityTraits = new Array();
    _games = new Array();

    //Creating the constructor to instantiate the object inside the Main.js class
    constructor(profilePicture,username,gender,age,location,personalityTraits,games
){
        this._profilePicture = profilePicture;
        this._username = username;
        this._gender = gender;
        this._age = age;
        this._location = location;
        this._personalityTraits = personalityTraits;
        this._games = games;
    }

    //Getter functions for returning desired attributes in the Main.js class
    getProfilePicture(){
        return this._profilePicture;
    }

    getUsername(){
        return this._username;
    }

    getGender(){
        return this._gender;
    }

    getAge(){
        return this._age;
    }

    getLocation(){
        return this._location;
    }

    getPersonalityTraits(){
        return this._personalityTraits;
    }

    getGames(){
        return this._games;
    }
}
```

## Match chatting Code

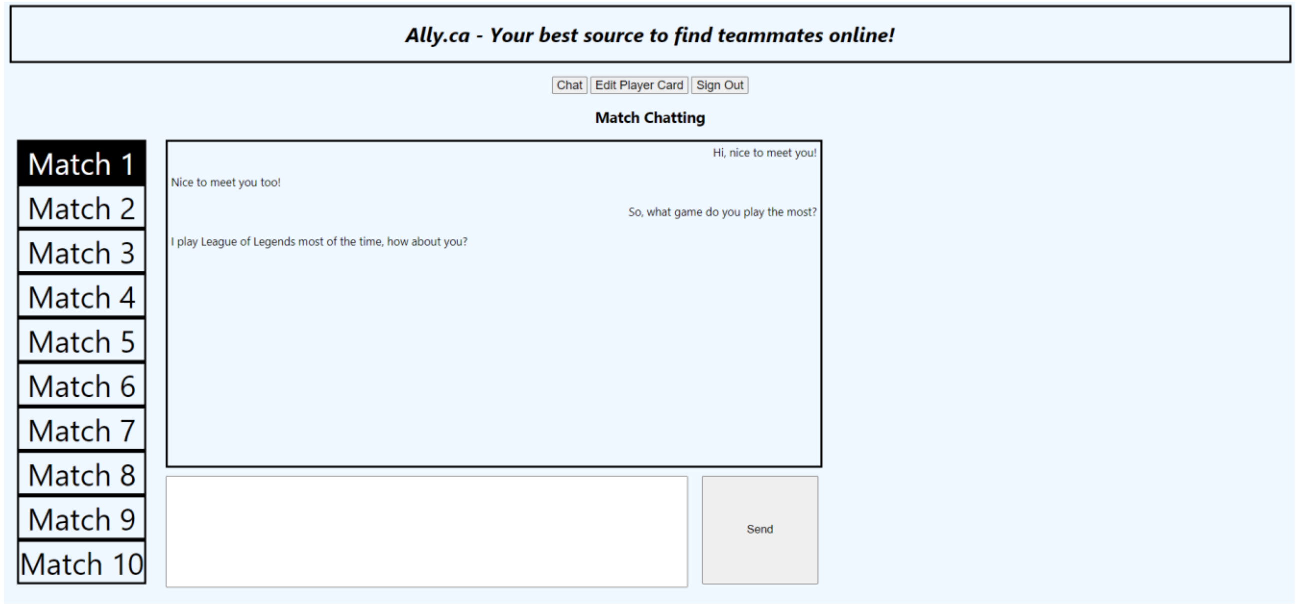
Below are the main Match chatting class, alongside the ChatHistory class next to it, and its interface. This class will show the chat history with a match to the user, as well as enable the user to write new messages and send them. After that, the message is logged in the ChatHistory instantiated object. There are 2 functions in the main class: one for loading the chat history and the other one for sending a new message and logging it in the chat history.

In the main class userDiv, matchDiv and newDiv will create “div” elements each one with a different style, so match messages are displayed in the left side and user messages on the right side. chatLog grabs the proper chat “div” element from the page to manipulate it, and textToBeSent grabs the text the user inputted. Finally, chatHistory creates and example of a chat history to demonstrate the class functionality.

In the ChatHistory class, there are 2 String-array-type attributes _userMessages and _matchMessages to store messages from the user and from the match, respectively. For the methods, there are getters that return a specific message from the user or from the match and loggers to insert a message sent by the user or the match.

```
//Code for the main class for the Match Chatting feature
//Creating an example chatHistory
let chatHistory = new ChatHistory(["Hi, nice to meet you!", "So, what game do you p
lay the most?"],
    ["Nice to meet you too!", "I play League of Legends most of the time, how about
 you?"]);

//Load chat history into the page
loadHistory();

//function for loading the chat history when page loads
function loadHistory() {
    let chatLog = document.getElementById("chatLog");

    //this is a simplified loop for appending the messages in the chat log
    for (let i = 0; i < chatHistory.getUserMessagesLength(); i++) {

        //for each message, create a div with an id of either user or match, so the
y are bound left or right of the div
        //User message
        let userDiv = document.createElement("div");
        userDiv.setAttribute("id", "userMessage");
        userDiv.textContent = chatHistory.getUserMessage(i);
        chatLog.appendChild(userDiv);
        chatLog.appendChild(document.createElement("br"));

        //Match message
        let matchDiv = document.createElement("div");
        matchDiv.setAttribute("id", "matchMessage");
        matchDiv.textContent = chatHistory.getMatchMessage(i);
        chatLog.appendChild(matchDiv);
        chatLog.appendChild(document.createElement("br"));
    }
}

//Function triggered when button is pressed. It will log the text onto the ChatHist
ory class and send it to the chat log of the page.
function logText() {
    //Instantiating chat history inside chatLog
    let textToBeSent = document.getElementById("textBox").value;

    let newDiv = document.createElement("div");
    newDiv.setAttribute("id", "userMessage");

    newDiv.textContent = textToBeSent;

    //appending message to the chat log
    chatLog.appendChild(newDiv);

    chatLog.appendChild(document.createElement("br"));

    //logging message into the class chat history
    chatHistory.logUserMessage(textToBeSent);

    //Resetting textbox
    document.getElementById("textBox").value = "";
}
```

```
//code for ChatHistory class
class ChatHistory
{
    //defining attributes
    _userMessages = new Array();
    _matchMessages = new Array();

    //constructor to instantiate example chat history
    constructor(userMessages,matchMessages){
        this._userMessages = userMessages;
        this._matchMessages = matchMessages;
    }

    //getters for appending messages in the chat log and the length
    getUserMessage(i){
        return (this._userMessages[i]);
    }

    getMatchMessage(i){
        return (this._matchMessages[i]);
    }
    getUserMessagesLength(){
        return this._userMessages.length;
    }

    getMatchMessagesLength(){
        return this._matchMessages.length;
    }

    //functions for logging new messages sent into the object
    logUserMessage(message){
        this._userMessages.push(message);
    }

    logMatchMessage(message){
        this._matchMessages.push(message);
    }
}
```



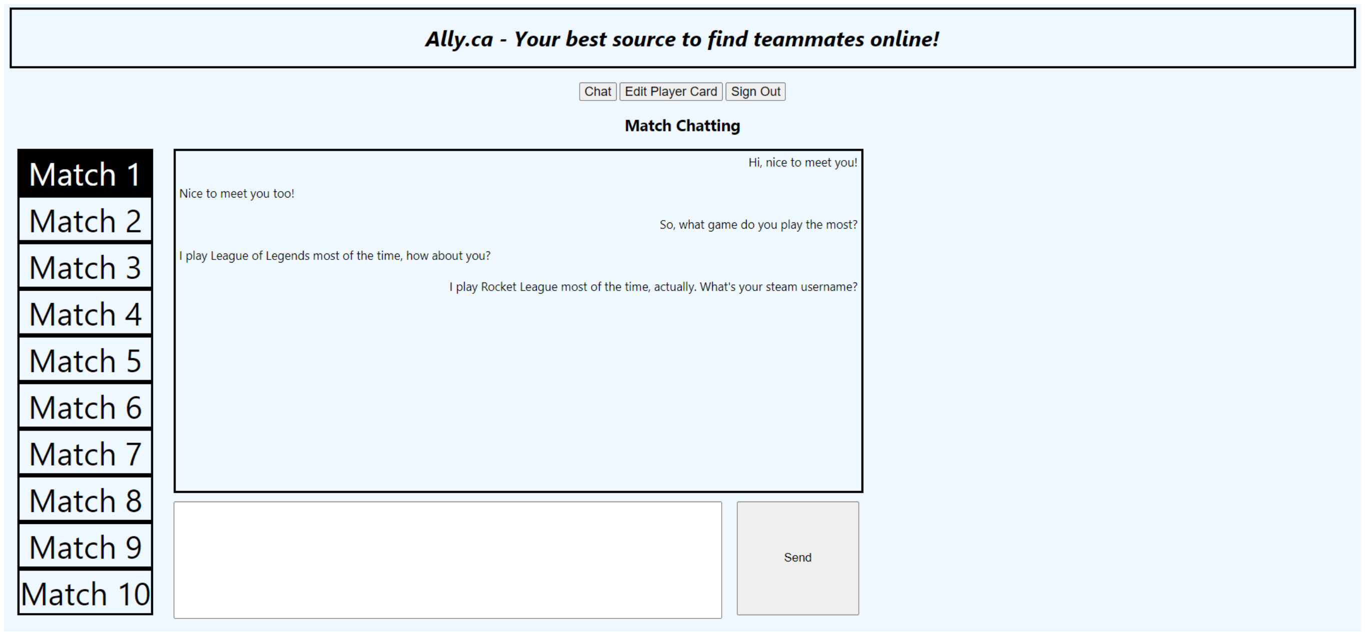
Hi, nice to meet you!

Nice to meet you too!

So, what game do you play the most?

I play League of Legends most of the time, how about you?

Figure 21 - Match Chatting interface when page loads.


Ally.ca - Your best source to find teammates online!

Match Chatting

Hi, nice to meet you!

Nice to meet you too!

So, what game do you play the most?

I play League of Legends most of the time, how about you?

I play Rocket League most of the time, actually. What's your steam username?

*Figure 22 - Match Chatting interface after a message has been sent.*

## Conclusion

The team's effort in making this web app has been tremendous, from pitching it to potential believers to sponsors in order to have a kickstart budget in working with them. Along the way, a broad idea of the web app was made, and the user interactions with the preliminary features were discussed. In general, the construction phase responded really well and connected to the planning and modeling phase, even though frontend prototypes were constructed before the backend. However, there is obviously still an immense amount of work to make the web app shippable to its beta testing phase. The frontend prototypes already have some structure, but the backend needs to be built with its databases and be set onto dedicated servers and authentication servers, so they can be integrated with the frontend. The latter also needs working on its aesthetic and more functionality needs to be added like filters for the Player Card browsing, a tutorial overlay for it, a notifications tray to show the user if he's been liked by other users, the Player Card creation interface after the account is successfully created, a stylish home page that shows the web app functionality and benefits to attract users to create new accounts, among countless other features the team needs to work on. So far, it has been a pleasure to work on the web app especially in the construction phase and we will try to keep the results satisfactory in order to kickstart this project the right way.

## References

1. Previous CSIS 3275 Assignments from Victor Correa Suleiman, ID 300315653
2. Maxim, Bruce R._ Pressman, Roger S. - Software engineering _ a practitioner's approach-McGraw-Hill Education (2015)
3. https://www.ejinsight.com/eji/article/id/2280405/20191022-video-game-industry-silently-taking-over-entertainment-world
4. https://gamedaily.biz/article/1709/coronavirus-leads-to-35-growth-for-the-video-games-industry
5. https://www.cleanpng.com/png-computer-servers-computer-icons-database-server-cl-2007816
6. https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/
7. https://github.com/correasuleiman-v5653/mySEJavaProject