Technical Guide
CA400
LIDELL

Ruadhán McCloskey
19444786
Daniel McCartney
19395186

Supervisor:
Renaat Verbruggen

Table of Contents:

1 Introduction:

1.1Overview:

We decided to create a game similar to WORDLE for guessing ranked UFC fighters. The game works as follows, the user gets 8 guesses to guess any current ranked UFC fighter. They will get hints based on the attributes of the fighter that they guess in comparison to the mystery fighter that they are trying to guess. Each fighter has 7 attributes (Name, Age, Reach, Rank, Record, Home Town and Division). Each time a user guesses a fighter their attributes will show up in the board with either a green, yellow or white colour indicating if they have the same, close or different attributes as the mystery fighter. These hints will help the user guess the mystery fighter. If the user runs out of guesses they are prompted with a pop-up indicating that they have lost and showing a picture of the mystery fighter along with their name. If they guess correctly they will receive a success message indicating that they have guessed the correct fighter and how many guesses it took them to get the right answer along with a picture of the mystery fighter and their name. There are 2 different game modes, Daily and Unlimited. The Daily mode you can only play once a day, each night at midnight a random fighter is generated and the user has 24 hours to guess the correct fighter. They can start the game and come back to it and their progress will still be there however you still only have 8 guesses to guess the correct fighter. In the unlimited mode the user can play the game as many times as they'd like with the game working the same as above.
We decided to use the Angular framework with Typescript to code the front end of our project. We chose Angular and Typescript because Daniel used Angular with Typescript during his internship so he was familiar with the language and framework. However I had to learn the language and framework by looking at a tutorial on youtube and I managed to pick it up as we went along.

1.2 Motivation:

The motivation for this project was that we both started playing WORDLE last year. WORDLE is a game where each day there's a new 5 letter word that you have to guess. You get hints based on the words that you guess and the characters of the word you guess in comparison to the word that you're trying to guess. We decided to use the same idea for this game and try to change it so it works for guessing UFC fighters. We chose the UFC as it was something that we both had an interest in. We decided that you would guess a fighter and get hints based off of the attributes of the fighter in comparison to the random fighter that you're trying to guess, which is the same idea as WORDLE just with a different implementation. We also found out that no games like this existed for guessing UFC fighters so we decided to go ahead and begin building our app.

2 Research:

Data:

Initially we wanted this game to be for guessing County GAA players, we checked online to see if this information was public and quickly found out that it was not. We then decided to email every county in Ireland to see if they would share the data of their players with us so we could use their attributes for our game. Only a few counties got back to us and all of them said that they couldn't share this information with us as it violates GDPR. We understood where they were coming from so we decided to move on and find another idea for our project.

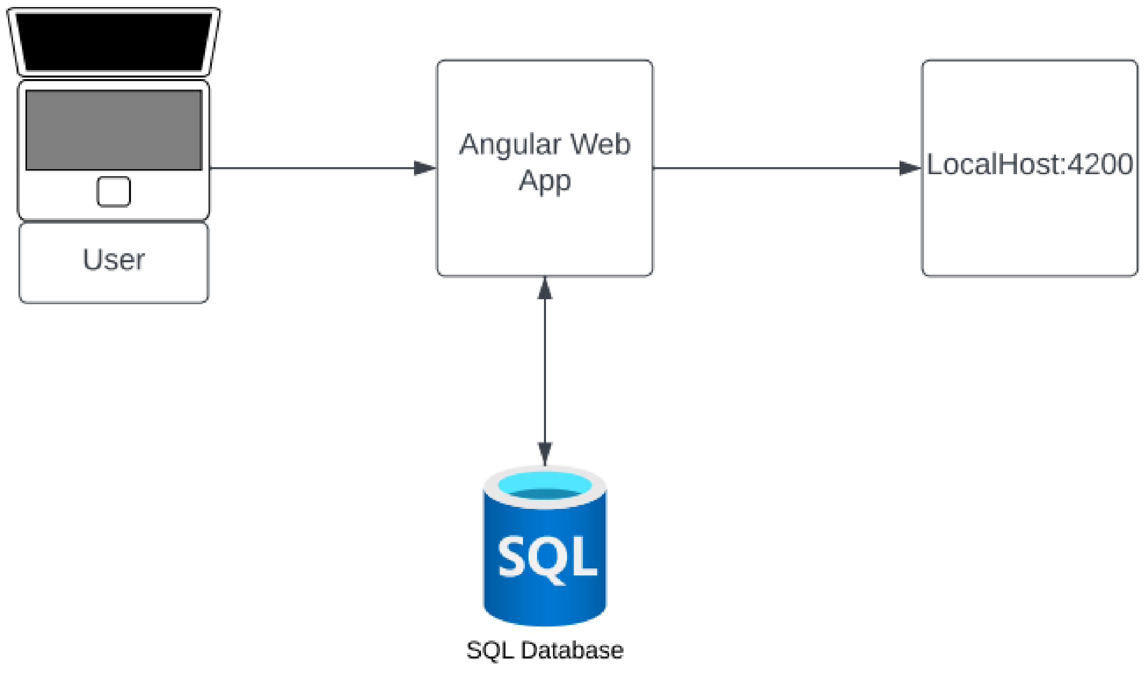
We quickly moved onto the UFC as it was something that we were both interested in so all we needed to do was find out if the fighters attributes and stats were available to the public. We quickly found out that all of the UFC fighters stats and attributes were available on the official UFC website. Now we had to find a way to get all of the data into our own database so we could query the data. We found a web scraper tool online that helped us scrape all of the data that we need into our own database. Here's a link to the website for the web scraper tool extension that we used https://webscraper.io/. We used this extension to get all the data that we needed for our database. Now that we had the data we could start looking at querying the data.

Does the game already exist?

We looked online to see if a game like this already exists for guessing UFC fighters, we found games for guessing UFC fighters but none of them were based on attributes and hints. It was mostly guessing fighters based on their photos, which for any real UFC fan would be quite easy in most cases. We wanted to create a game that would be challenging and competitive for UFC and MMA fans to play. We believed that if we implemented the game correctly we could create something worth playing.

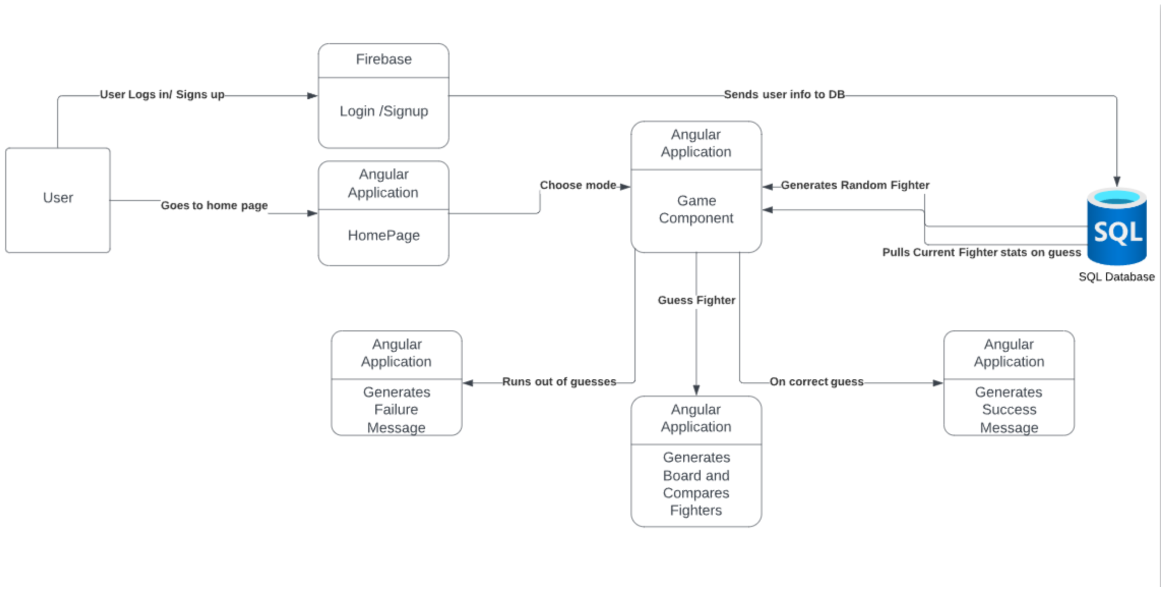
3 Design

3.1 System Architecture:



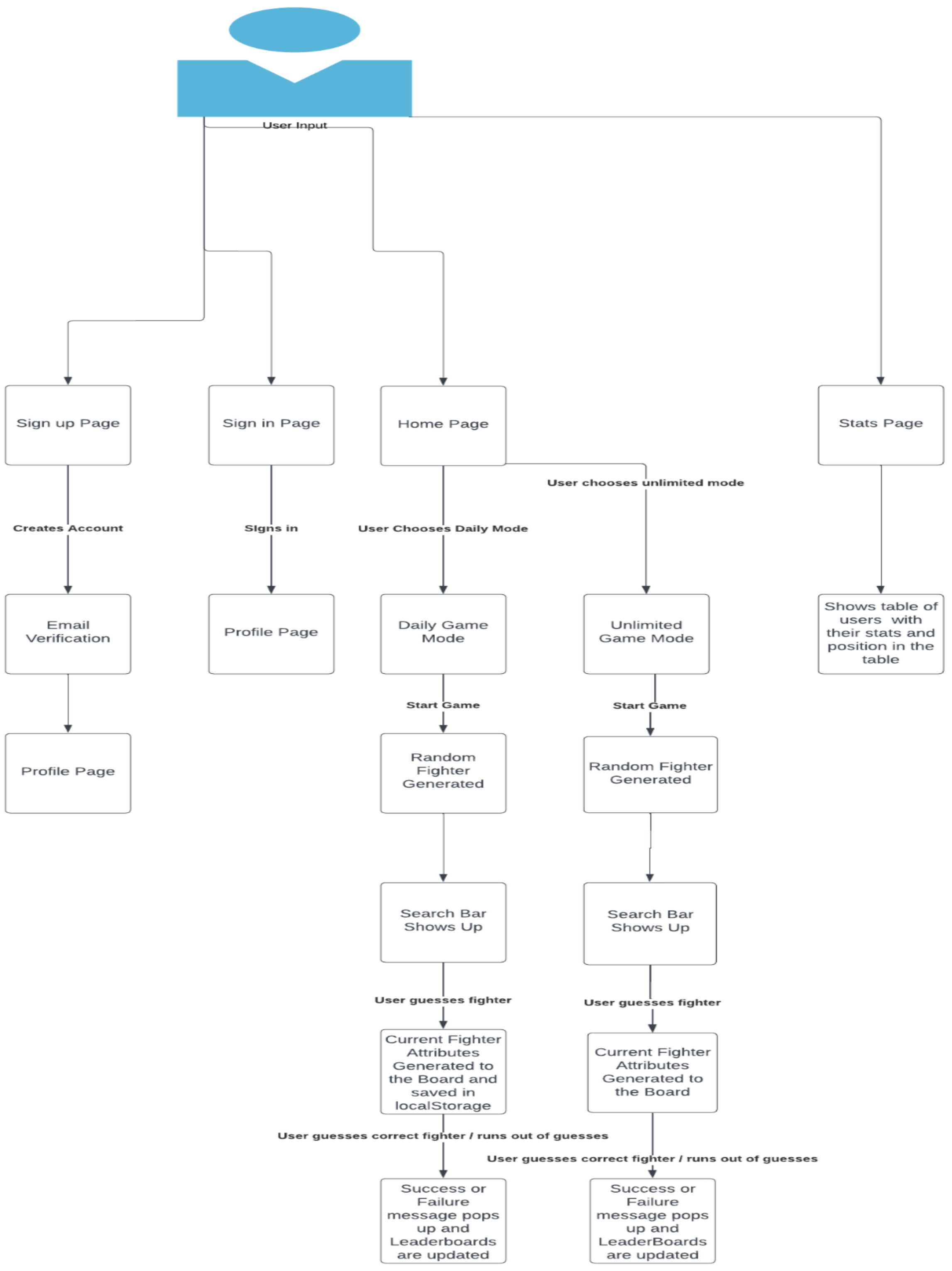
The diagram above shows the interactions in our system from the user to the front-end which was done in Angular and Typescript to our backend and database which was done in Microsoft SQL Server Management Studio. Then the app is deployed on localhost:4200.

3.2 Data Flow Diagram:



Above is our data flow diagram for a user interacting with our application and the functions that run when they do certain actions. It shows the data flow through our systems architecture also with the data that goes to and from the database.

3.3 User Input Diagram:

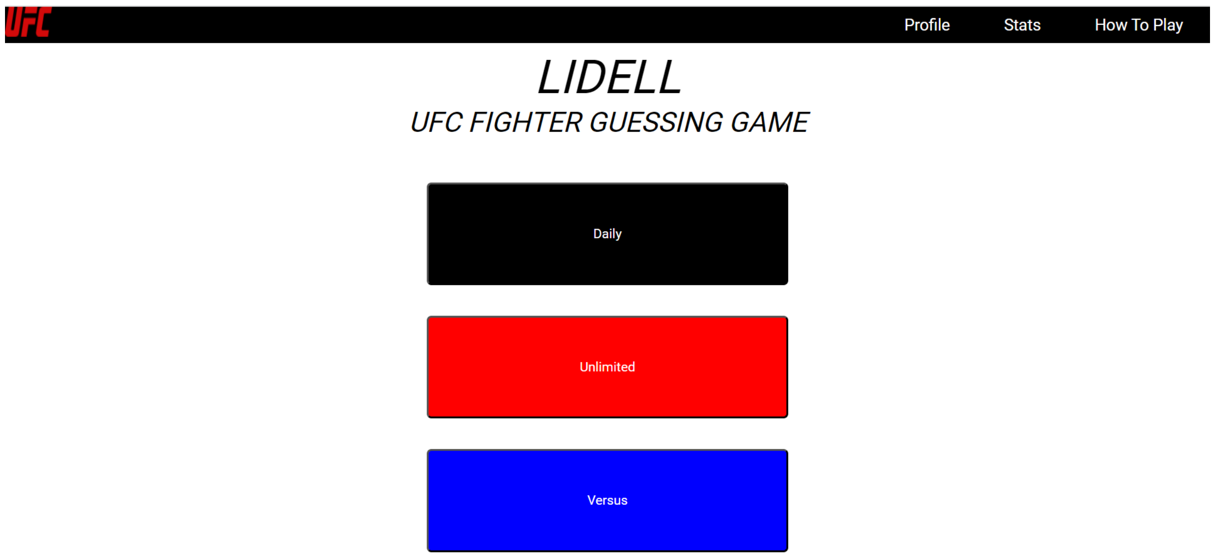

User Input
Sign up Page
Sign in Page
Home Page
Stats Page
Creates Account
Signs in
User Chooses Daily Mode
User chooses unlimited mode
Email Verification
Profile Page
Daily Game Mode
Unlimited Game Mode
Shows table of users with their stats and position in the table
Profile Page
Start Game
Start Game
Random Fighter Generated
Random Fighter Generated
Search Bar Shows Up
Search Bar Shows Up
User guesses fighter
User guesses fighter
Current Fighter Attributes Generated to the Board and saved in localStorage
Current Fighter Attributes Generated to the Board
User guesses correct fighter / runs out of guesses
User guesses correct fighter / runs out of guesses
Success or Failure message pops up and Leaderboards are updated
Success or Failure message pops up and LeaderBoards are updated

The above diagram shows a more in-depth overview of the user interaction with our game. It shows the events that occur when the user interacts with the system and

4 Implementation:

4.1 Home Page:

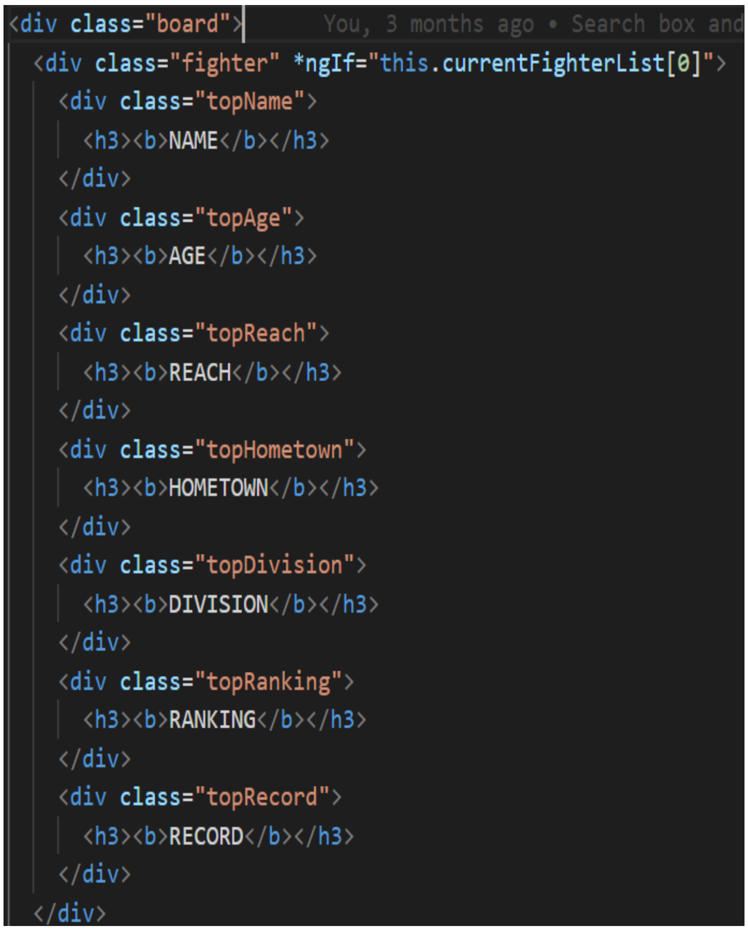
UI is very important to us and we wanted to create a UI that isn't too cluttered or complicated. We only wanted to have useful information on each web page. We chose bright colours for our buttons on the homepage to brighten up the page a bit. We also wanted to have a "How to Play " popup which when you click it opens a modal or "pop-up" that explains how to play the game and the rules of the game. We decided to create a homepage with 2 options, one to play the "Daily" version of the game and one to play the "Unlimited" version.



4.2 Game Page:

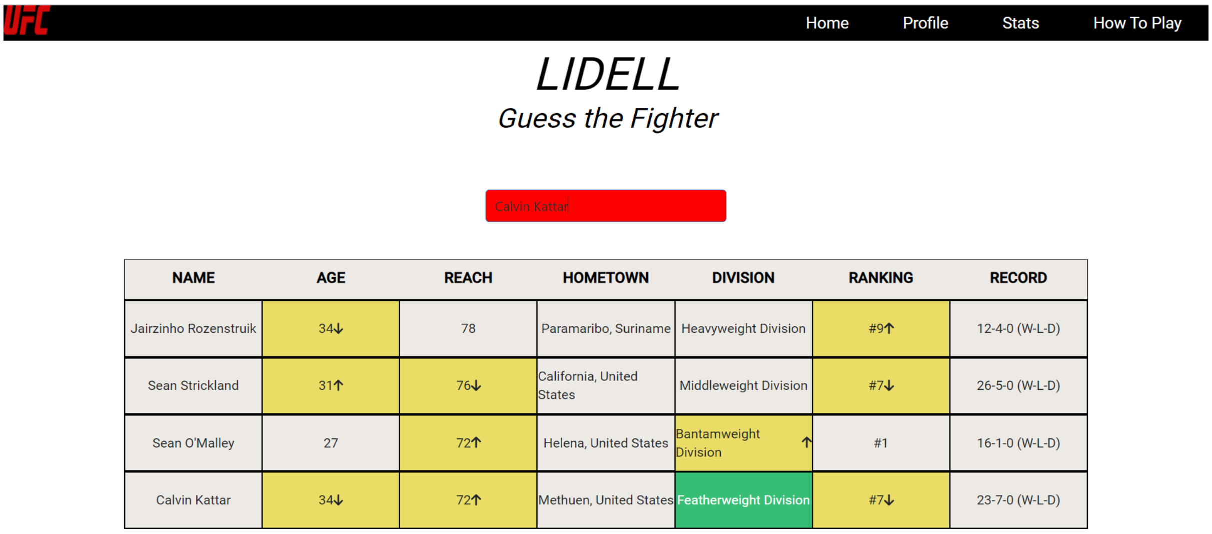
For the design of the game page we chose to have a "Start Game" button, which initialises the game and generates a random fighter to be guessed by the user. Once this button is clicked by the user the game will begin and the search bar will show up prompting the user to guess a fighter. Once the user guesses a fighter the attributes of the fighter will show up as a board, with all of their attributes (Name, Age, Reach, Record, Home Town, Ranking and Division). We also wanted to design the board so that users would get hints based on their fighters attributes in comparison to the random fighter. So we chose a yellow colour for attributes that are close, green for attributes that are the same and if the attribute is neither close nor the same it stays white indicating no match. We also didn't want to have too much information on this page as we don't want to take users' focus away from the board and the actual game.

Some code for the board component:



```
<div class="board">        You, 3 months ago • Search box and
  <div class="fighter" *ngIf="this.currentFighterList[0]">
    <div class="topName">
      <h3><b>NAME</b></h3>
    </div>
    <div class="topAge">
      <h3><b>AGE</b></h3>
    </div>
    <div class="topReach">
      <h3><b>REACH</b></h3>
    </div>
    <div class="topHometown">
      <h3><b>HOMETOWN</b></h3>
    </div>
    <div class="topDivision">
      <h3><b>DIVISION</b></h3>
    </div>
    <div class="topRanking">
      <h3><b>RANKING</b></h3>
    </div>
    <div class="topRecord">
      <h3><b>RECORD</b></h3>
    </div>
  </div>
```

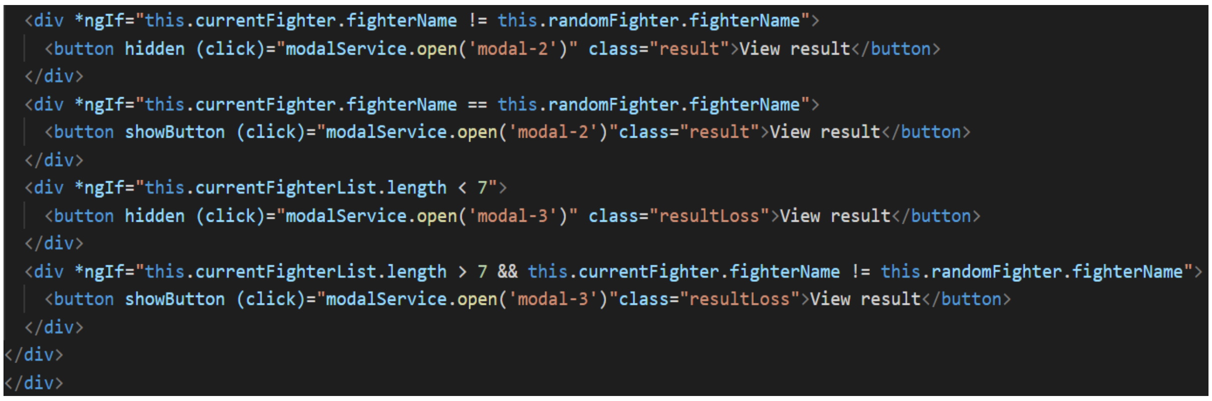
Example of the game board generation with multiple guesses:



NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Jairzinho Rozenstruik	34↓	78	Paramaribo, Suriname	Heavyweight Division	#9↑	12-4-0 (W-L-D)
Sean Strickland	31↑	76↓	California, United States	Middleweight Division	#7↓	26-5-0 (W-L-D)
Sean O'Malley	27	72↑	Helena, United States	Bantamweight Division ↑	#1	16-1-0 (W-L-D)
Calvin Kattar	34↓	72↑	Methuen, United States	Featherweight Division	#7↓	23-7-0 (W-L-D)

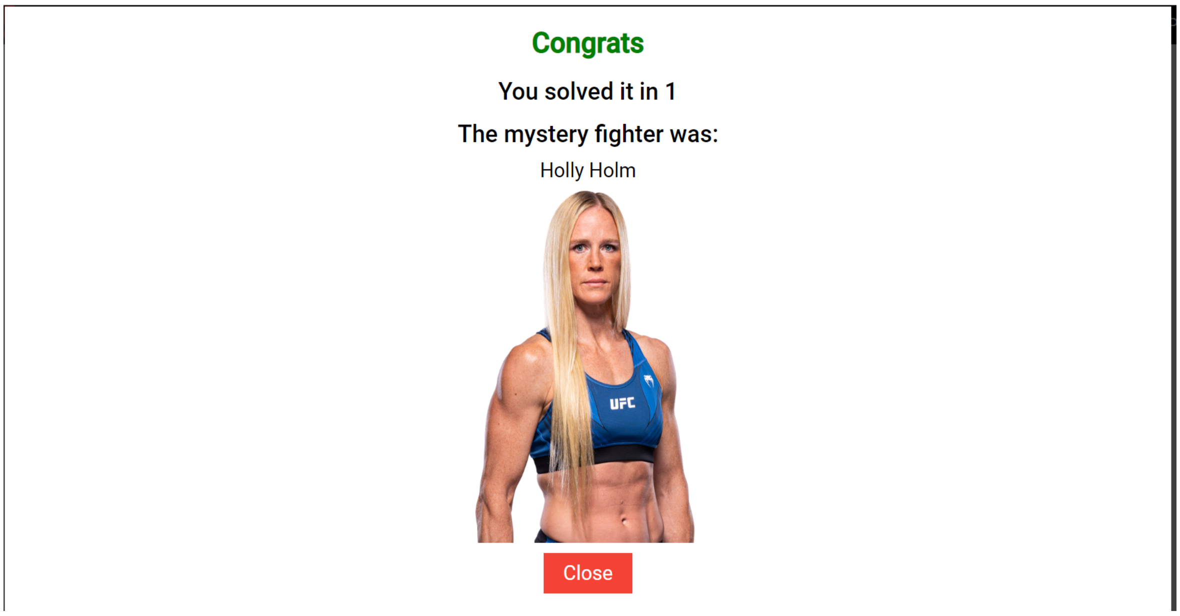
We also implemented pop-ups for users after they finish playing the game indicating whether they guessed the correct fighter or not and how many guesses it took them to get the correct answer. A picture of the random fighter is also displayed along with their name.

Code for when to show buttons to open these messages:



```
  <div *ngIf="this.currentFighter.fighterName != this.randomFighter.fighterName">
    <button hidden (click)="modalService.open('modal-2')" class="result">View result</button>
  </div>
  <div *ngIf="this.currentFighter.fighterName == this.randomFighter.fighterName">
    <button showButton (click)="modalService.open('modal-2')"class="result">View result</button>
  </div>
  <div *ngIf="this.currentFighterList.length < 7">
    <button hidden (click)="modalService.open('modal-3')" class="resultLoss">View result</button>
  </div>
  <div *ngIf="this.currentFighterList.length > 7 && this.currentFighter.fighterName != this.randomFighter.fighterName">
    <button showButton (click)="modalService.open('modal-3')"class="resultLoss">View result</button>
  </div>
</div>
</div>
```

Success Message:



Congrats

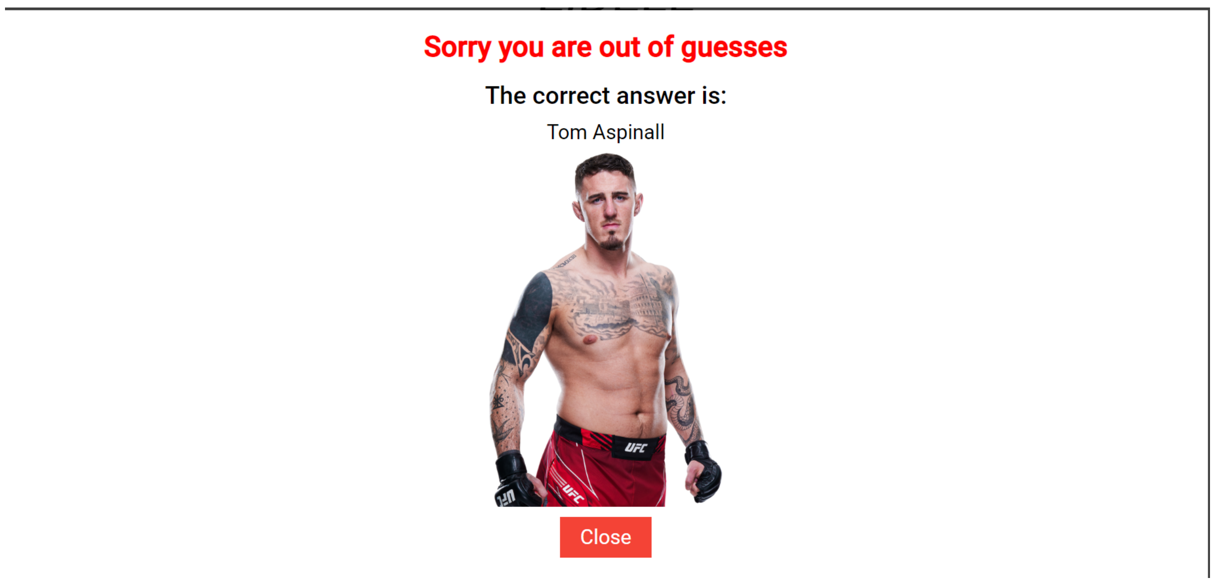
You solved it in 1

The mystery fighter was:

Holly Holm

If the user fails to guess the correct fighter in 8 guesses they will be prompted with a failure message indicating that they have lost the game.

Failure Message:


Sorry you are out of guesses

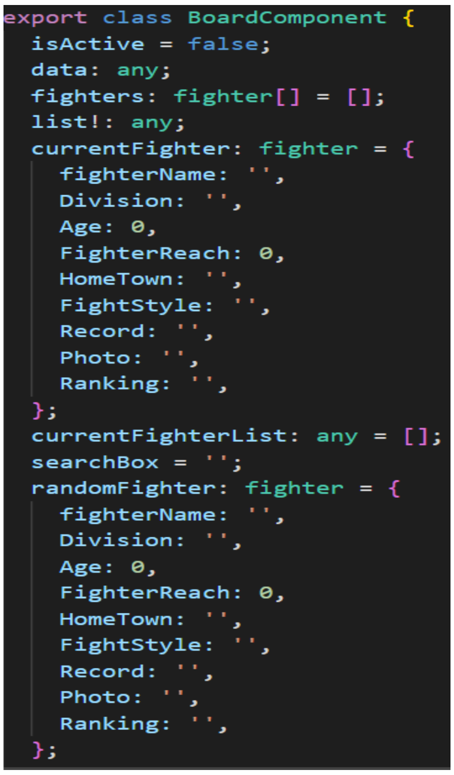
The correct answer is:

Tom Aspinall

Implementing the actual game and the board generation was probably the hardest part of the entire project. We had to get it so we could compare the data of the randomly generated fighter with the data of the current fighter, which is the fighter that the user guesses. We did this by creating a list called currentFighterList which every time a fighter is guessed it would be added to this list. Therefore we could keep the previous guesses and their attributes in the board so their hints could still be used.

4.3 Comparing Fighters:

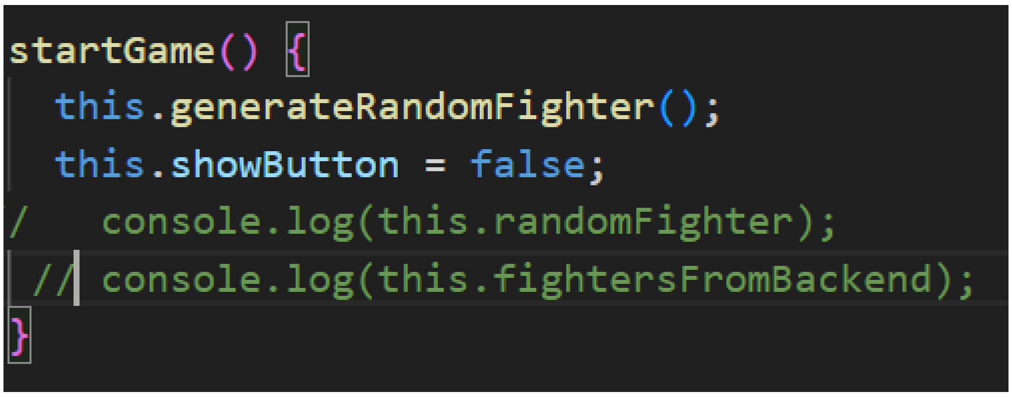
The first thing we had to do was declare our attributes for comparing the fighters to one another. We also have some other variables that we needed for our code in this snippet. We declared the attributes for currentFighter and randomFighter using separate dictionaries for each and giving each attribute a key value pair. Now we could call these values in our functions.



```
export class BoardComponent {
  isActive = false;
  data: any;
  fighters: fighter[] = [];
  list!: any;
  currentFighter: fighter = {
    fighterName: '',
    Division: '',
    Age: 0,
    FighterReach: 0,
    HomeTown: '',
    FightStyle: '',
    Record: '',
    Photo: '',
    Ranking: '',
  };
  currentFighterList: any = [];
  searchBox = '';
  randomFighter: fighter = {
    fighterName: '',
    Division: '',
    Age: 0,
    FighterReach: 0,
    HomeTown: '',
    FightStyle: '',
    Record: '',
    Photo: '',
    Ranking: '',
  };
```

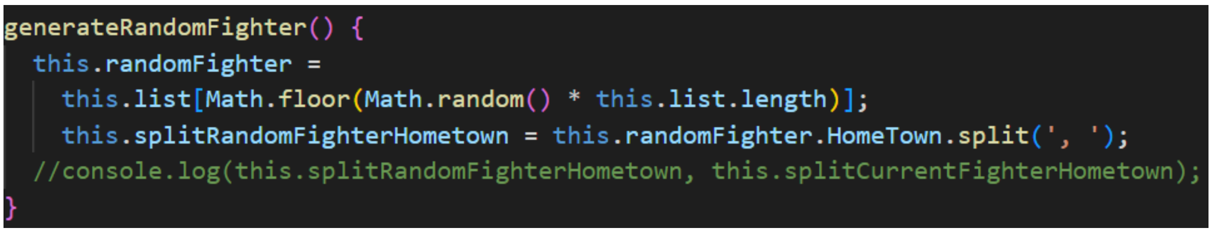
We also needed a button to generate our random fighter for our users to guess. We created a button that when pressed would generate a random fighter from our database using our generateRandomFighter() function.

Code for starting the game:
board.component.ts



```
startGame() {
  this.generateRandomFighter();
  this.showButton = false;
/   console.log(this.randomFighter);
 // console.log(this.fightersFromBackend);
}
```

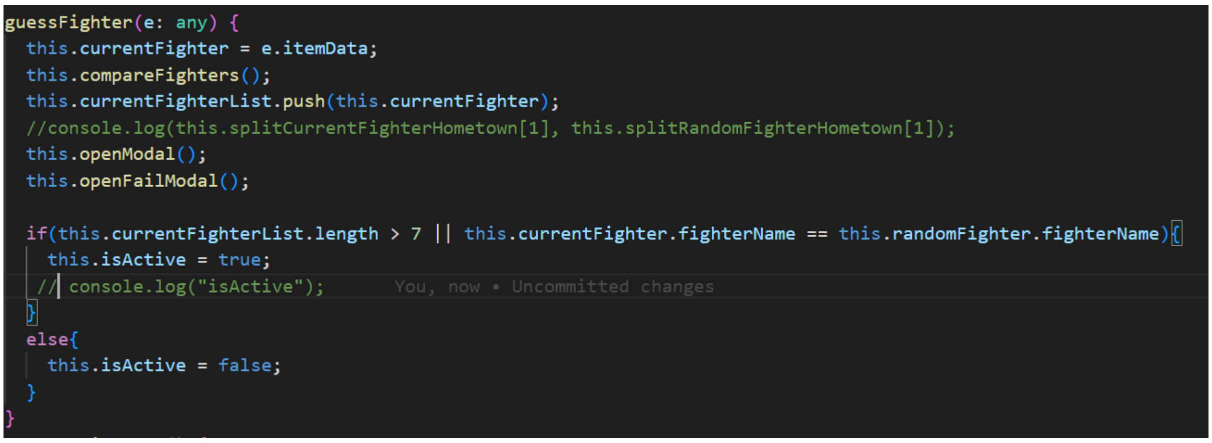
Code for generateRandomFighter:
board.component.ts



```
generateRandomFighter() {
  this.randomFighter =
    this.list[Math.floor(Math.random() * this.list.length)];
    this.splitRandomFighterHometown = this.randomFighter.HomeTown.split(', ');
  //console.log(this.splitRandomFighterHometown, this.splitCurrentFighterHometown);
}
```

We also created a function for guessing a fighter which runs the necessary functions inside it that we need for our checks. This function runs our compareFighters() function which splits the hometowns for the fighters so we can compare the country and city separately.

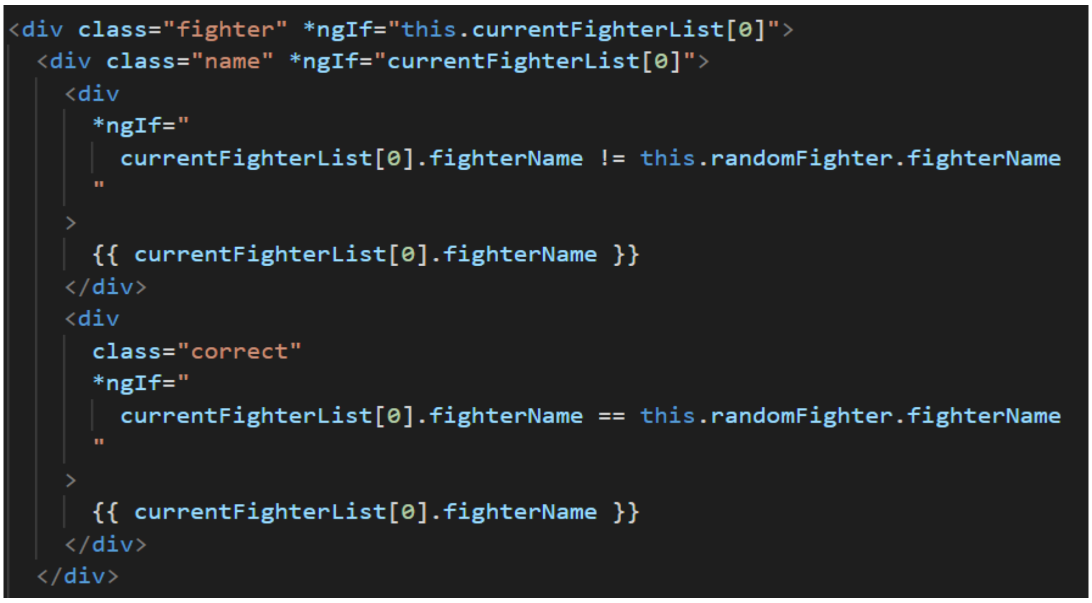
Code for guessFighter():
board.component.ts



```
guessFighter(e: any) {
  this.currentFighter = e.itemData;
  this.compareFighters();
  this.currentFighterList.push(this.currentFighter);
  //console.log(this.splitCurrentFighterHometown[1], this.splitRandomFighterHometown[1]);
  this.openModal();
  this.openFailModal();

  if(this.currentFighterList.length > 7 || this.currentFighter.fighterName == this.randomFighter.fighterName){
    this.isActive = true;
   //| console.log("isActive");       You, now • Uncommitted changes
  }
  else{
    this.isActive = false;
  }
}
```

Now we had to compare each attribute of the current fighter separately starting with “Name” which had only one comparison, equal to or not equal to. If the names were the same then you had guessed the correct fighter and won the game.

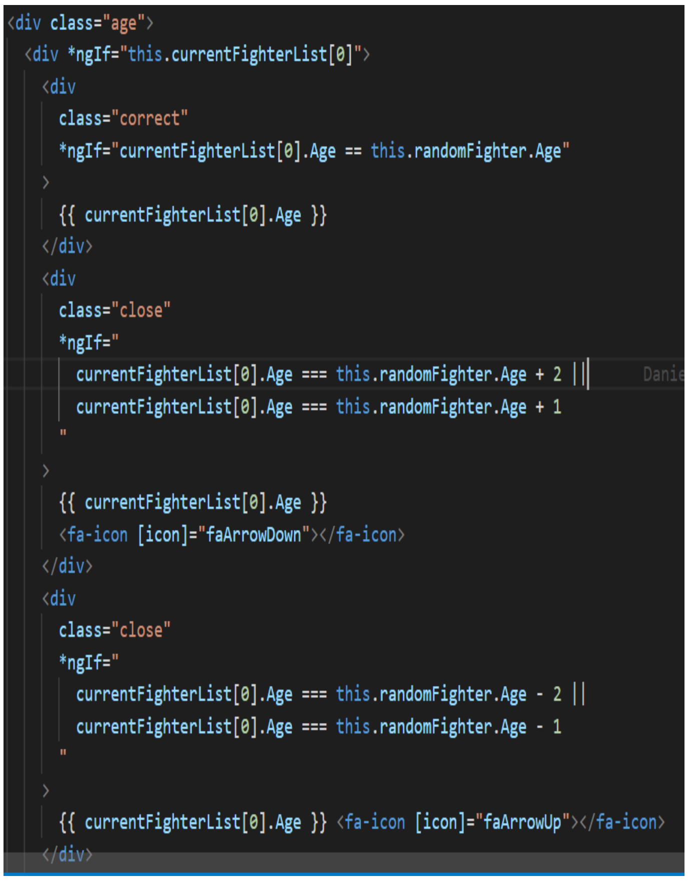
Code for comparing the fighter names:



```
<div class="fighter" *ngIf="this.currentFighterList[0]">
  <div class="name" *ngIf="currentFighterList[0]">
    <div
      *ngIf="
        currentFighterList[0].fighterName != this.randomFighter.fighterName
      "
    >
      {{ currentFighterList[0].fighterName }}
    </div>
    <div
      class="correct"
      *ngIf="
        currentFighterList[0].fighterName == this.randomFighter.fighterName
      "
    >
      {{ currentFighterList[0].fighterName }}
    </div>
  </div>
```

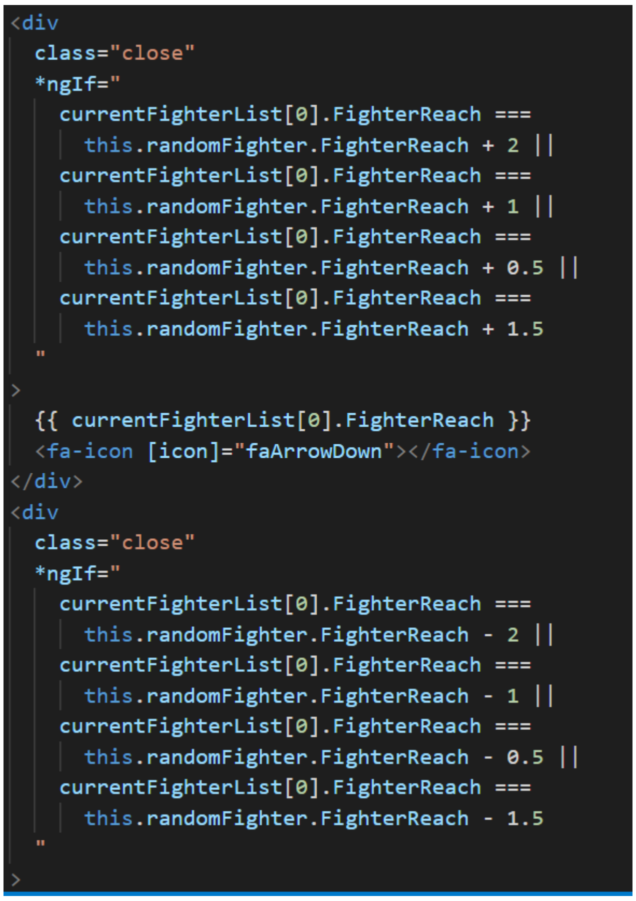
We then had to compare “Age” we wanted it so if you guess a fighter that is 1 or 2 years younger than the random fighter this attribute would show up yellow with an arrow pointing up indicating that the random fighter is 1 or 2 years older than the fighter that you have guessed. It would also work in reverse if you guessed a fighter that is 1 or 2 years older than the random fighter which would show up yellow with an

arrow pointing down. We also implemented the same feature for “Reach” working the exact same way.
Code for comparing the fighter age:



```html
<div class="age">
  <div *ngIf="this.currentFighterList[0]">
    <div
      class="correct"
      *ngIf="currentFighterList[0].Age == this.randomFighter.Age"
    >
      {{ currentFighterList[0].Age }}
    </div>
    <div
      class="close"
      *ngIf="
        currentFighterList[0].Age === this.randomFighter.Age + 2 ||
        currentFighterList[0].Age === this.randomFighter.Age + 1
      "
    >
      {{ currentFighterList[0].Age }}
      <fa-icon [icon]="faArrowDown"></fa-icon>
    </div>
    <div
      class="close"
      *ngIf="
        currentFighterList[0].Age === this.randomFighter.Age - 2 ||
        currentFighterList[0].Age === this.randomFighter.Age - 1
      "
    >
      {{ currentFighterList[0].Age }} <fa-icon [icon]="faArrowUp"></fa-icon>
    </div>
```

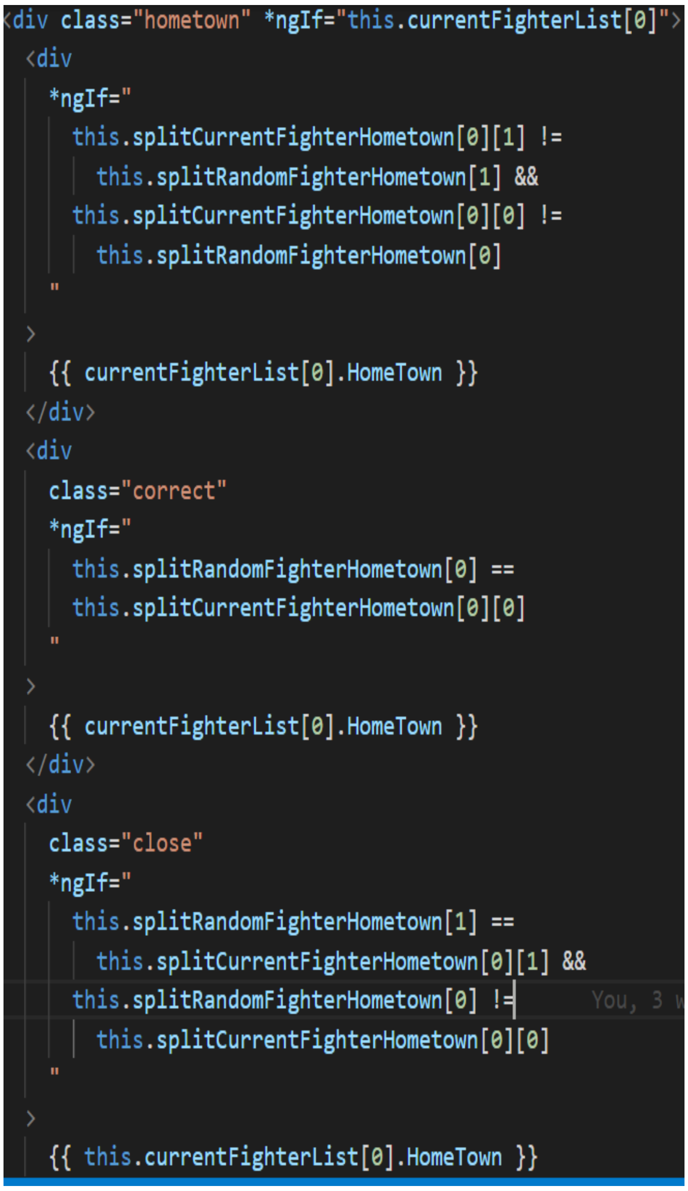
Code for comparing the fighter reach:



```
<div
  class="close"
  *ngIf="
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach + 2 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach + 1 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach + 0.5 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach + 1.5
  "
>
  {{ currentFighterList[0].FighterReach }}
  <fa-icon [icon]="faArrowDown"></fa-icon>
</div>
<div
  class="close"
  *ngIf="
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach - 2 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach - 1 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach - 0.5 ||
    currentFighterList[0].FighterReach ===
      this.randomFighter.FighterReach - 1.5
  "
>
```

Next we implemented checks for “Home Town”. We did this by splitting up the fighters home town on the comma between the country and city. If the country was the same as the random fighters but the city was different this attribute would show up yellow, indicating that the fighters were both from the same country.

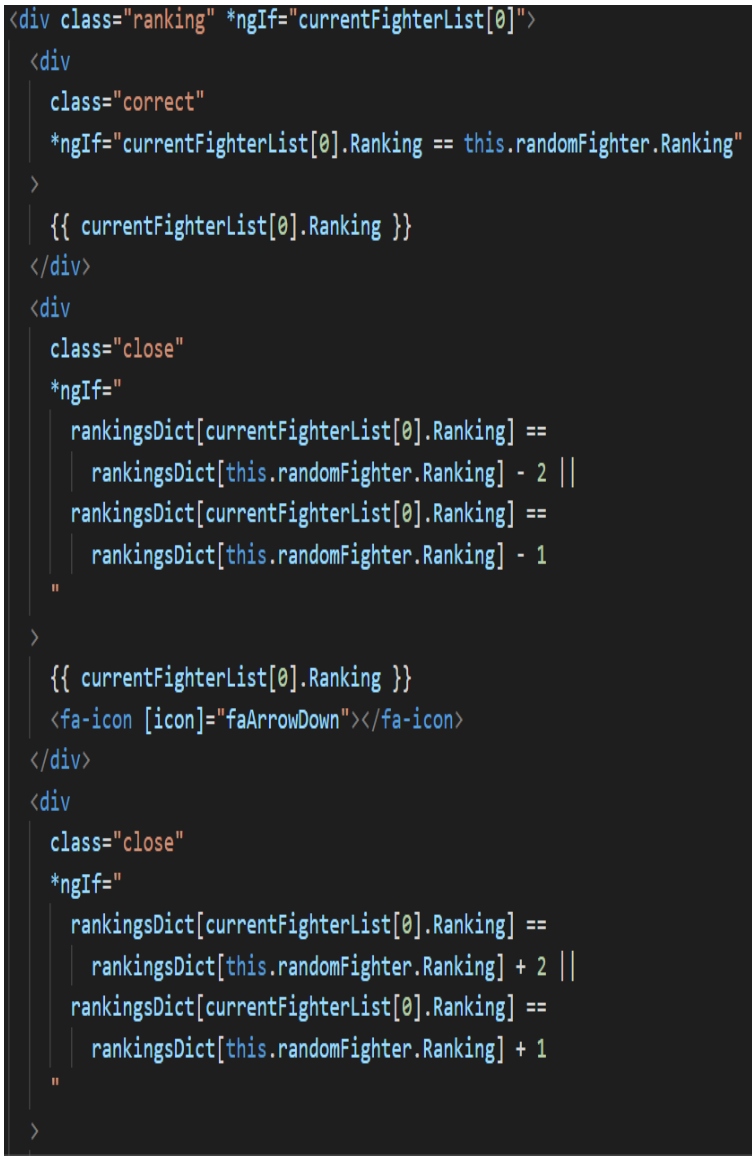
Code for comparing the fighter hometowns:



```
<div class="hometown" *ngIf="this.currentFighterList[0]">
  <div
    *ngIf="
      this.splitCurrentFighterHometown[0][1] !=
        this.splitRandomFighterHometown[1] &&
      this.splitCurrentFighterHometown[0][0] !=
        this.splitRandomFighterHometown[0]
    "
  >
    {{ currentFighterList[0].HomeTown }}
  </div>
  <div
    class="correct"
    *ngIf="
      this.splitRandomFighterHometown[0] ==
      this.splitCurrentFighterHometown[0][0]
    "
  >
    {{ currentFighterList[0].HomeTown }}
  </div>
  <div
    class="close"
    *ngIf="
      this.splitRandomFighterHometown[1] ==
        this.splitCurrentFighterHometown[0][1] &&
      this.splitRandomFighterHometown[0] !=        You, 3 w
        this.splitCurrentFighterHometown[0][0]
    "
  >
    {{ this.currentFighterList[0].HomeTown }}
```

We then moved onto “Ranking”, ranking indicates where the fighter ranks in the division from 15 to champion, because our rankings are a string we had to create a dictionary of all the rankings from 15 all the way to the champion. We then based our comparisons on their key word pairs in the dictionary. If the fighter we guess is ranked 1 or 2 positions lower than the random fighter eg. Current fighter ranking is 15 and random fighter ranking is 13. The current fighters ranking will show up yellow with an arrow pointing up indicating the random fighter is ranked higher than the current fighter within 1 or 2 places. It works the same if the current fighter is ranked higher than the random fighter it will show the ranking attribute with an arrow pointing down indicating the random fighter is ranked lower than the fighter that you guessed within 1 or 2.

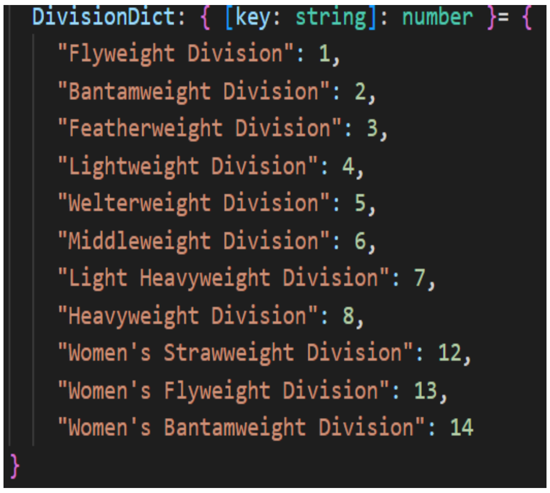
Code for comparing the fighter ranking:



```
<div class="ranking" *ngIf="currentFighterList[0]">
  <div
    class="correct"
    *ngIf="currentFighterList[0].Ranking == this.randomFighter.Ranking"
  >
    {{ currentFighterList[0].Ranking }}
  </div>
  <div
    class="close"
    *ngIf="
      rankingsDict[currentFighterList[0].Ranking] ==
        rankingsDict[this.randomFighter.Ranking] - 2 ||
      rankingsDict[currentFighterList[0].Ranking] ==
        rankingsDict[this.randomFighter.Ranking] - 1
    "
  >
    {{ currentFighterList[0].Ranking }}
    <fa-icon [icon]="faArrowDown"></fa-icon>
  </div>
  <div
    class="close"
    *ngIf="
      rankingsDict[currentFighterList[0].Ranking] ==
        rankingsDict[this.randomFighter.Ranking] + 2 ||
      rankingsDict[currentFighterList[0].Ranking] ==
        rankingsDict[this.randomFighter.Ranking] + 1
    "
  >
```

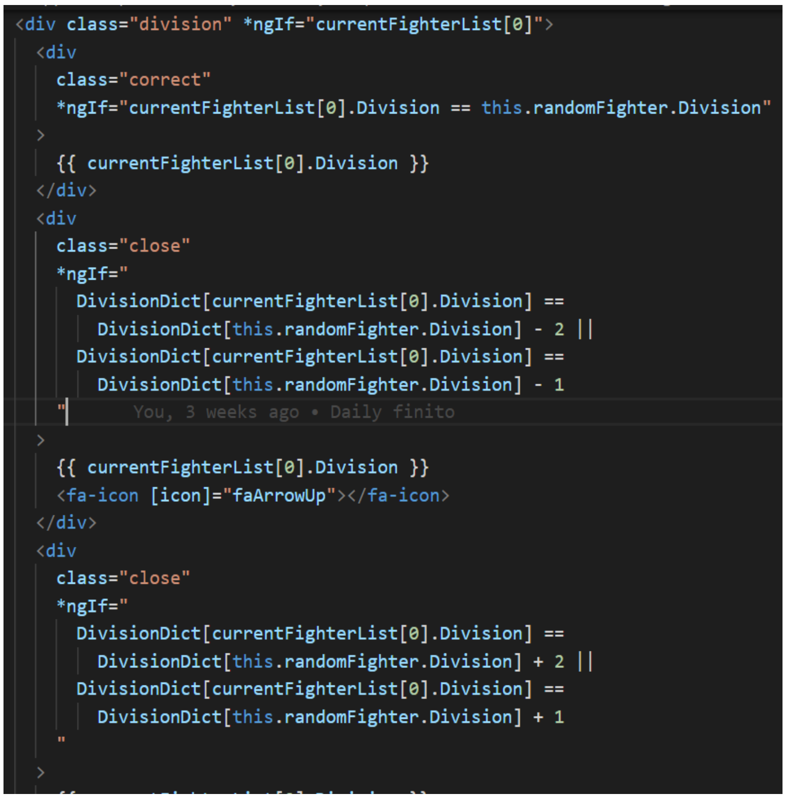
Next we compared “Division”, in total there are 8 men’s divisions in the UFC and there are 3 women’s divisions. The men’s divisions span from Flyweight up to Heavyweight, whereas the Women’s divisions only span from Strawweight to Bantamweight. We decided to split both the men’s and women’s divisions up to prevent confusion. We created a dictionary with all of the divisions in them with key value pairs so we could compare them. If you guess a female fighter and the fighter you’re trying to guess is a woman, you will get a hint based on division as there are only 3 divisions. It will either show up green or yellow with an arrow indicating whether the random fighter is in a higher weight class or lower. For male weight classes we compare them the same way as we did with other attributes by using a yellow colour with an arrow indicating whether the random fighter is in a higher or lower weight class within 1 or 2 divisions.

Here’s the dictionary of the divisions:



```
DivisionDict: { [key: string]: number }= {
  "Flyweight Division": 1,
  "Bantamweight Division": 2,
  "Featherweight Division": 3,
  "Lightweight Division": 4,
  "Welterweight Division": 5,
  "Middleweight Division": 6,
  "Light Heavyweight Division": 7,
  "Heavyweight Division": 8,
  "Women's Strawweight Division": 12,
  "Women's Flyweight Division": 13,
  "Women's Bantamweight Division": 14
}
```

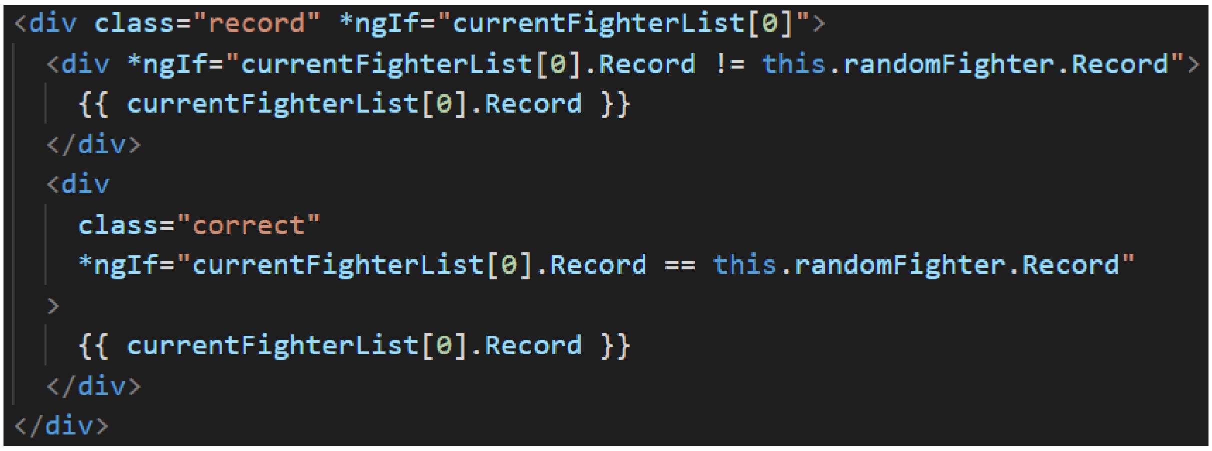
Code for comparing the fighter divisions:



```
<div class="division" *ngIf="currentFighterList[0]">
  <div
    class="correct"
    *ngIf="currentFighterList[0].Division == this.randomFighter.Division"
  >
    {{ currentFighterList[0].Division }}
  </div>
  <div
    class="close"
    *ngIf="
      DivisionDict[currentFighterList[0].Division] ==
        DivisionDict[this.randomFighter.Division] - 2 ||
      DivisionDict[currentFighterList[0].Division] ==
        DivisionDict[this.randomFighter.Division] - 1
    "      You, 3 weeks ago • Daily finito
  >
    {{ currentFighterList[0].Division }}
    <fa-icon [icon]="faArrowUp"></fa-icon>
  </div>
  <div
    class="close"
    *ngIf="
      DivisionDict[currentFighterList[0].Division] ==
        DivisionDict[this.randomFighter.Division] + 2 ||
      DivisionDict[currentFighterList[0].Division] ==
        DivisionDict[this.randomFighter.Division] + 1
    "
  >
```

Then finally the “Record” attribute, we could only really do one comparison for this attribute which was correct or incorrect. It is rare enough that fighters would have the same record but if they do the attribute shows up green indicating a match.

Code for comparing the record:

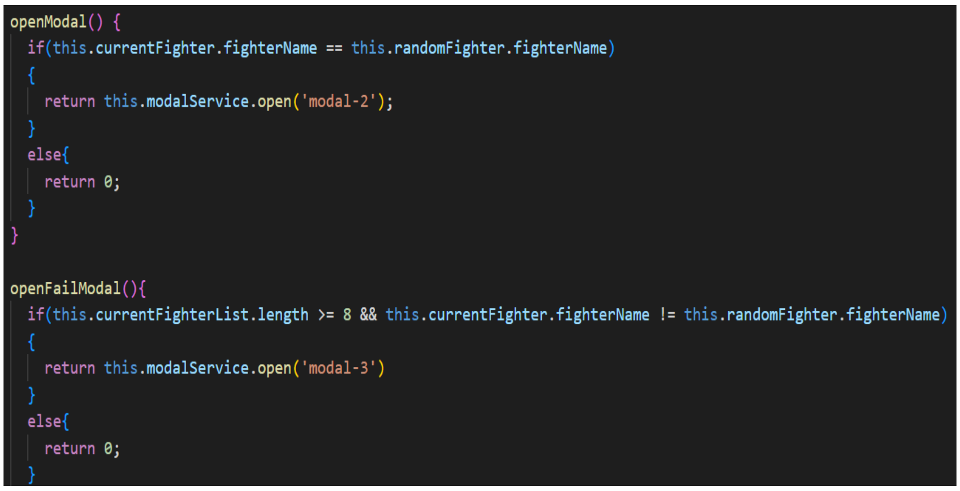


```
<div class="record" *ngIf="currentFighterList[0]">
  <div *ngIf="currentFighterList[0].Record != this.randomFighter.Record">
    {{ currentFighterList[0].Record }}
  </div>
  <div
    class="correct"
    *ngIf="currentFighterList[0].Record == this.randomFighter.Record"
  >
    {{ currentFighterList[0].Record }}
  </div>
</div>
```

Each attribute will show up green if they match that of the random fighter and will remain white if they do not match or if they aren't close either. We had to do these comparisons for each index in the current fighter list, so it would run these checks for each guess and post them to the board.

Once the user guesses the random fighter or runs out of guesses a pop-up will show up indicating if they have guessed the correct fighter with the number of guesses it took them to guess the random fighter or indicating that they have run out of guesses and both pop-ups will show the random fighters name along with a photo of them.

Code for opening success and failure modals:



```
openModal() {
  if(this.currentFighter.fighterName == this.randomFighter.fighterName)
  {
    return this.modalService.open('modal-2');
  }
  else{
    return 0;
  }
}

openFailModal(){
  if(this.currentFighterList.length >= 8 && this.currentFighter.fighterName != this.randomFighter.fighterName)
  {
    return this.modalService.open('modal-3')
  }
  else{
    return 0;
  }
```

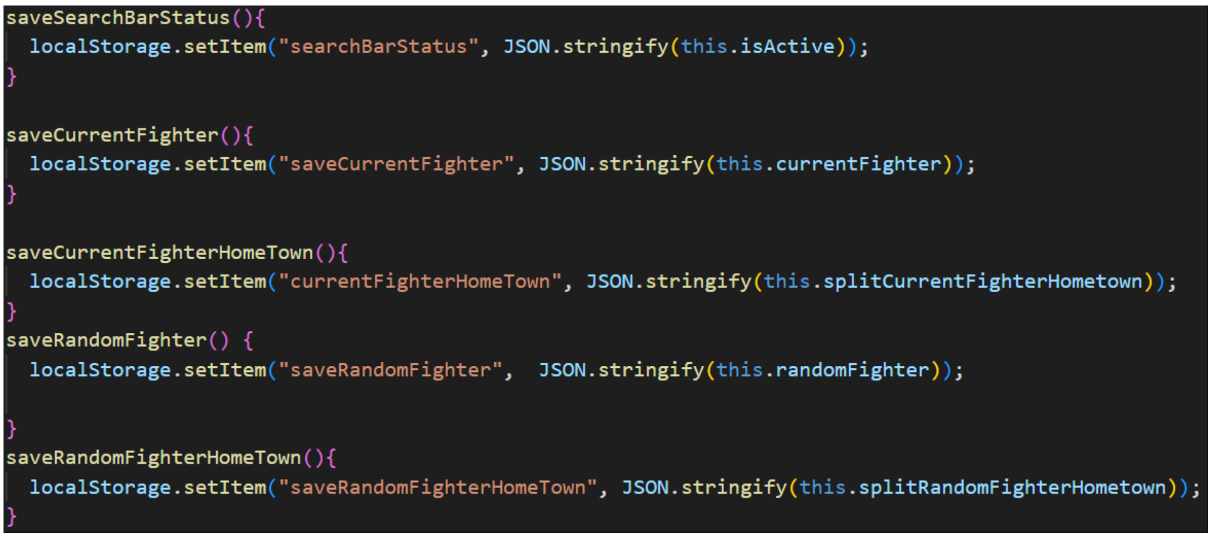
All of these checks are run in both the daily.component.html and board.component.html files for each guess.

4.4 Daily Game Mode:

The daily game mode you can only play once a day, each night at midnight a random fighter is generated and you have 24 hours to guess the fighter with a limit of 8 guesses. In the unlimited mode users can play the game as many times as they like, earning points based on the amount of guesses it took them to guess the correct fighter. Once they finish playing their scores are updated to the leaderboards.

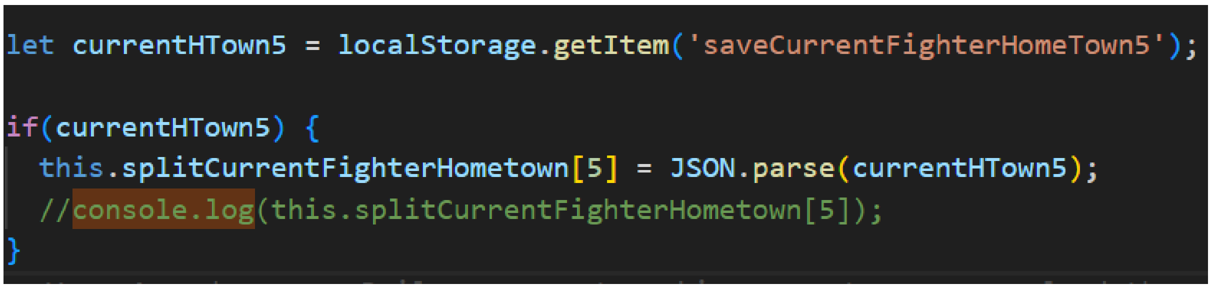
For the Daily game mode we needed to save the users progress in the game so they could leave the game page and come back to it. We did this by using localStorage which allows you to save data in local storage for each user. We would then clear all the data at midnight so they can start again with a new random fighter to guess. We initialised new variables for the saved items using localStorage.setItem() to save the variables and localStorage.getItem() to retrieve them. We then used localStorage.clear() to clear all the data at midnight. We also run our functions to save each fighter in currentFighterList in guessFighter().

Code for localStorage.setItem():



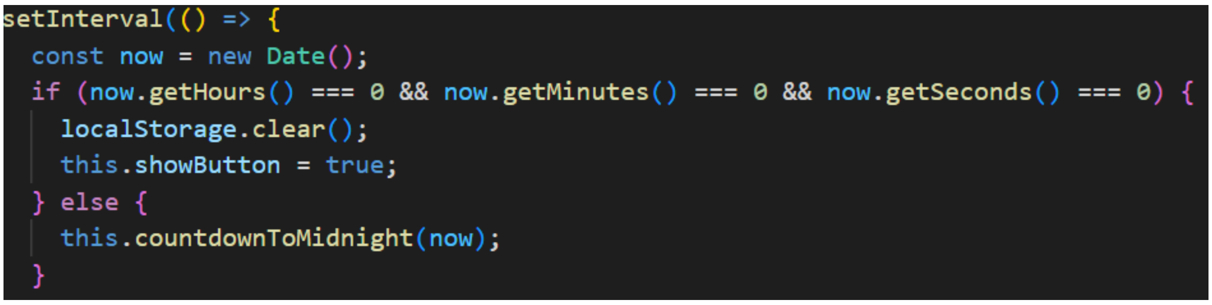
```
saveSearchBarStatus(){
  localStorage.setItem("searchBarStatus", JSON.stringify(this.isActive));
}

saveCurrentFighter(){
  localStorage.setItem("saveCurrentFighter", JSON.stringify(this.currentFighter));
}

saveCurrentFighterHomeTown(){
  localStorage.setItem("currentFighterHomeTown", JSON.stringify(this.splitCurrentFighterHometown));
}
saveRandomFighter() {
  localStorage.setItem("saveRandomFighter",  JSON.stringify(this.randomFighter));

}
saveRandomFighterHomeTown(){
  localStorage.setItem("saveRandomFighterHomeTown", JSON.stringify(this.splitRandomFighterHometown));
}
```

Code for localStorage.getItem():



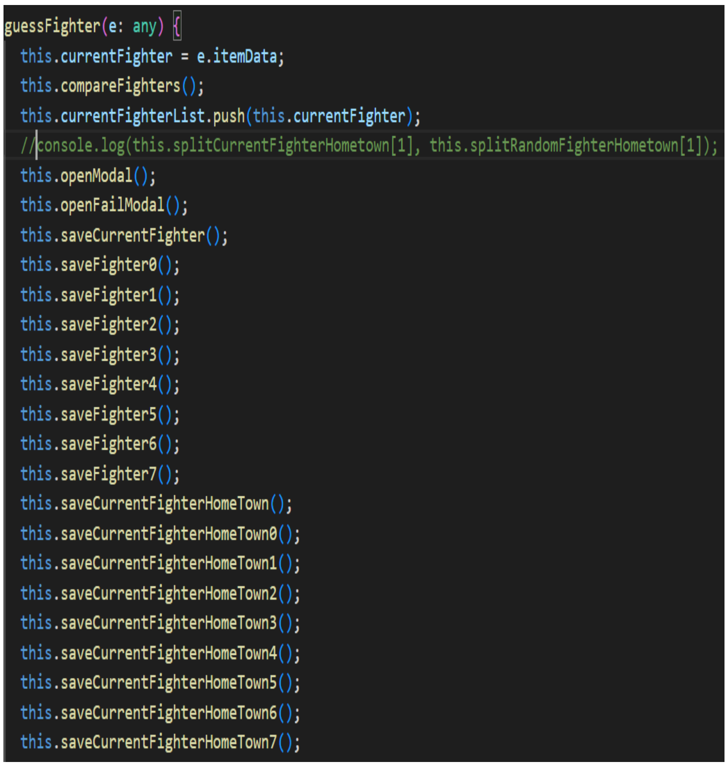
```
let currentHTown5 = localStorage.getItem('saveCurrentFighterHomeTown5');

if(currentHTown5) {
  this.splitCurrentFighterHometown[5] = JSON.parse(currentHTown5);
  //console.log(this.splitCurrentFighterHometown[5]);
}
```

Code for clearing local storage at midnight:



```
setInterval(() => {
  const now = new Date();
  if (now.getHours() === 0 && now.getMinutes() === 0 && now.getSeconds() === 0) {
    localStorage.clear();
    this.showButton = true;
  } else {
    this.countdownToMidnight(now);
  }
```

Code running our functions to save the fighters and their attributes:

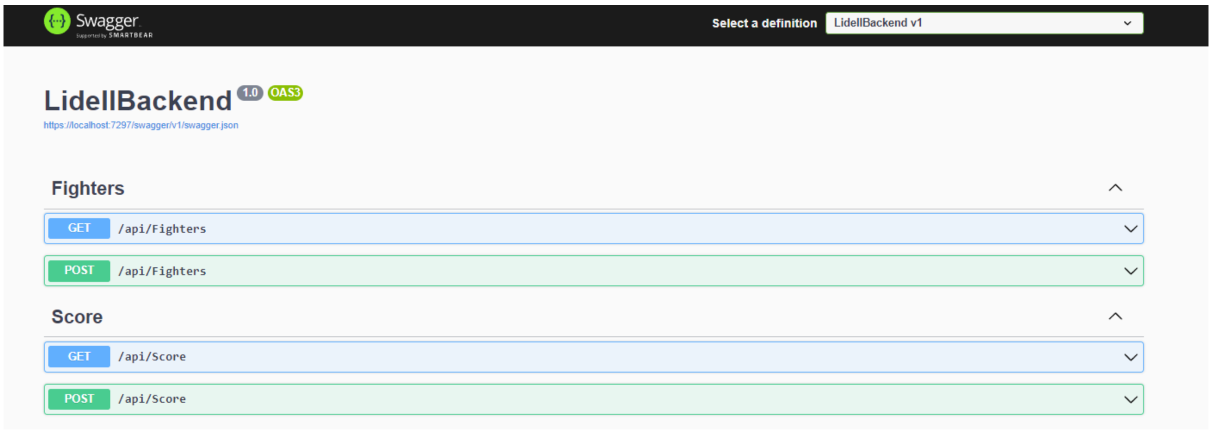


```
guessFighter(e: any) {
  this.currentFighter = e.itemData;
  this.compareFighters();
  this.currentFighterList.push(this.currentFighter);
  //console.log(this.splitCurrentFighterHometown[1], this.splitRandomFighterHometown[1]);
  this.openModal();
  this.openFailModal();
  this.saveCurrentFighter();
  this.saveFighter0();
  this.saveFighter1();
  this.saveFighter2();
  this.saveFighter3();
  this.saveFighter4();
  this.saveFighter5();
  this.saveFighter6();
  this.saveFighter7();
  this.saveCurrentFighterHomeTown();
  this.saveCurrentFighterHomeTown0();
  this.saveCurrentFighterHomeTown1();
  this.saveCurrentFighterHomeTown2();
  this.saveCurrentFighterHomeTown3();
  this.saveCurrentFighterHomeTown4();
  this.saveCurrentFighterHomeTown5();
  this.saveCurrentFighterHomeTown6();
  this.saveCurrentFighterHomeTown7();
```

4.5 Backend:

The backend for this project was coded using the .net framework and c#. We wanted communication between our frontend, backend and database to be fluent and quick so we kept our design fairly simple. We used the swaggerAPI to run tests on our get/post requests to make sure that they worked before implementing them into our frontend code.

SwaggerAPI


Swagger
Select a definition LidellBackend v1

LidellBackend 1.0 OAS3
https://localhost:7297/swagger/v1/swagger.json

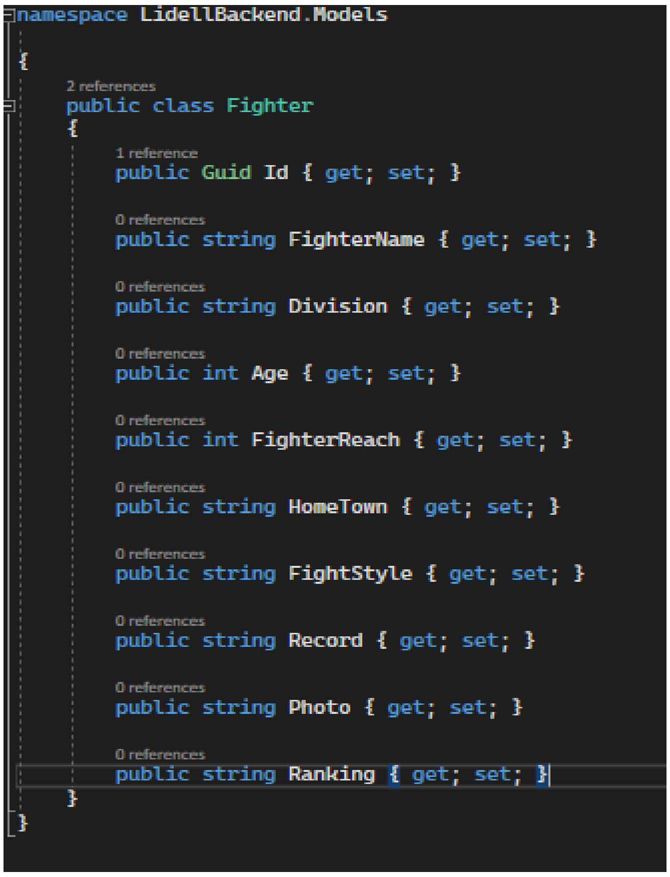
Fighters
GET /api/Fighters
POST /api/Fighters

Score
GET /api/Score
POST /api/Score

4.5.1 Class Models:

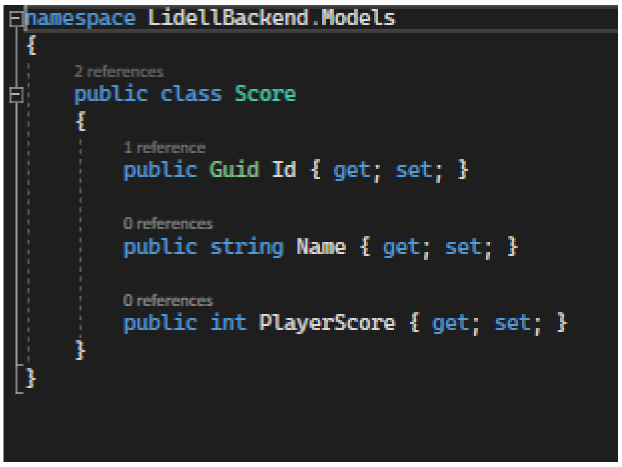
We created two models for the backend to send get/post requests, one for fighters and one for the score for the leaderboards. They follow the same structure as the interfaces in our frontend, allowing the objects built in the frontend to be ready to post straight away.

Fighter Model:



```
namespace LidellBackend.Models
{
    public class Fighter
    {
        public Guid Id { get; set; }

        public string FighterName { get; set; }

        public string Division { get; set; }

        public int Age { get; set; }

        public int FighterReach { get; set; }

        public string HomeTown { get; set; }

        public string FightStyle { get; set; }

        public string Record { get; set; }

        public string Photo { get; set; }

        public string Ranking { get; set; }
    }
}
```

Score Model:

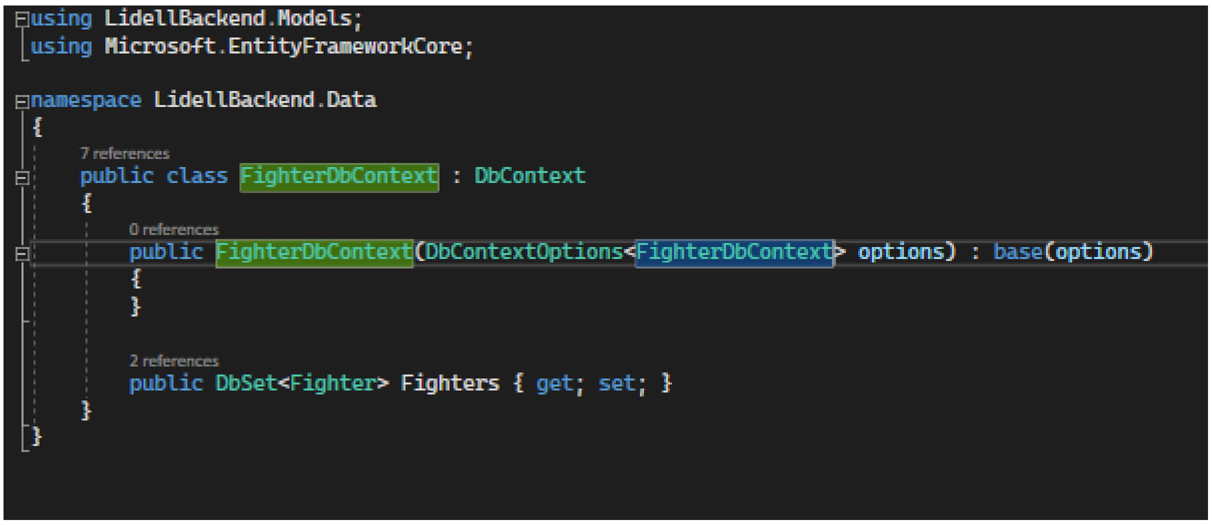


```
namespace LidellBackend.Models
{
    2 references
    public class Score
    {
        1 reference
        public Guid Id { get; set; }

        0 references
        public string Name { get; set; }

        0 references
        public int PlayerScore { get; set; }
    }
}
```

4.5.2 DbContext:

We used the DbContext functionality in the microsoft.entityframeworkcore.tools library to connect our project to an Microsoft SQL server database and also to add migrations and create the tables inside of the database, based on our models. The DbContext is also used to query from the database with migrations made.

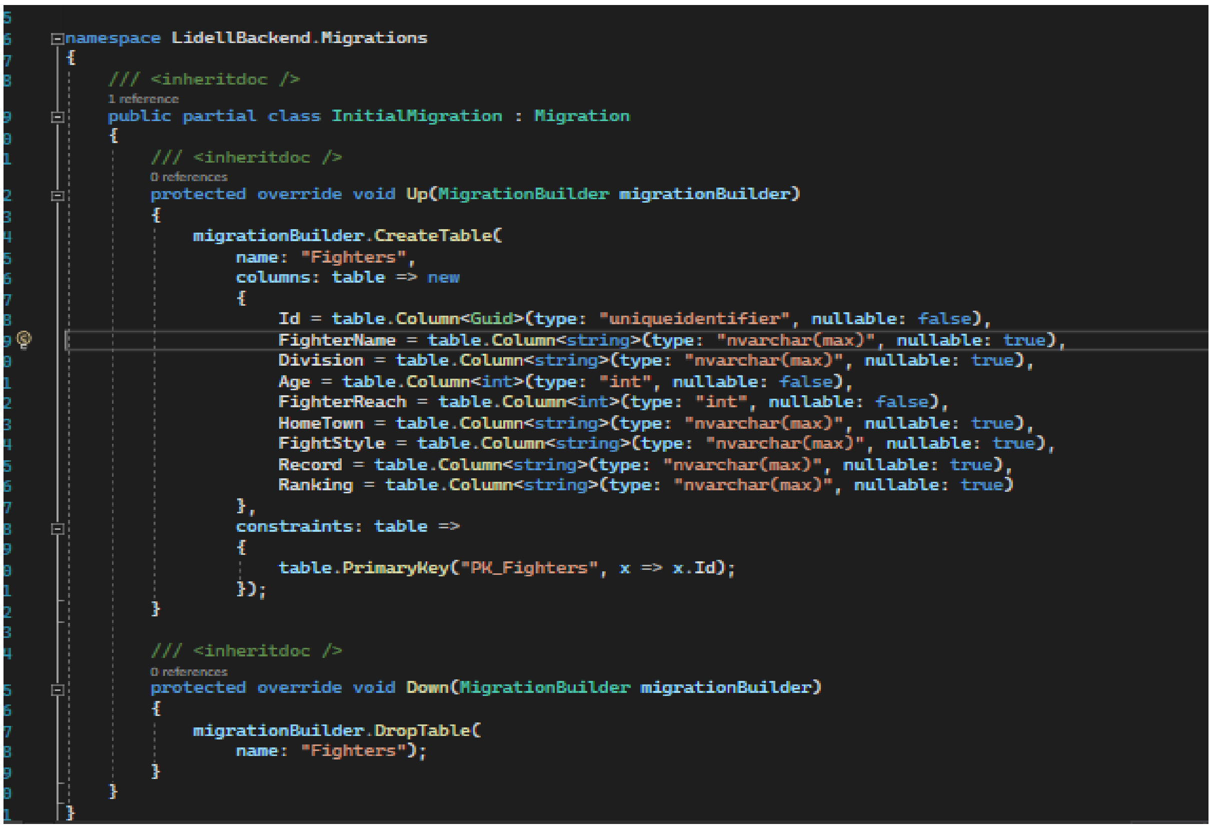
Fighter DbContext



```
using LidellBackend.Models;
using Microsoft.EntityFrameworkCore;

namespace LidellBackend.Data
{
    7 references
    public class FighterDbContext : DbContext
    {
        0 references
        public FighterDbContext(DbContextOptions<FighterDbContext> options) : base(options)
        {
        }

        2 references
        public DbSet<Fighter> Fighters { get; set; }
    }
}
```

The score DbContext is identical to the fighter one above.

Auto Generated Fighter Migration

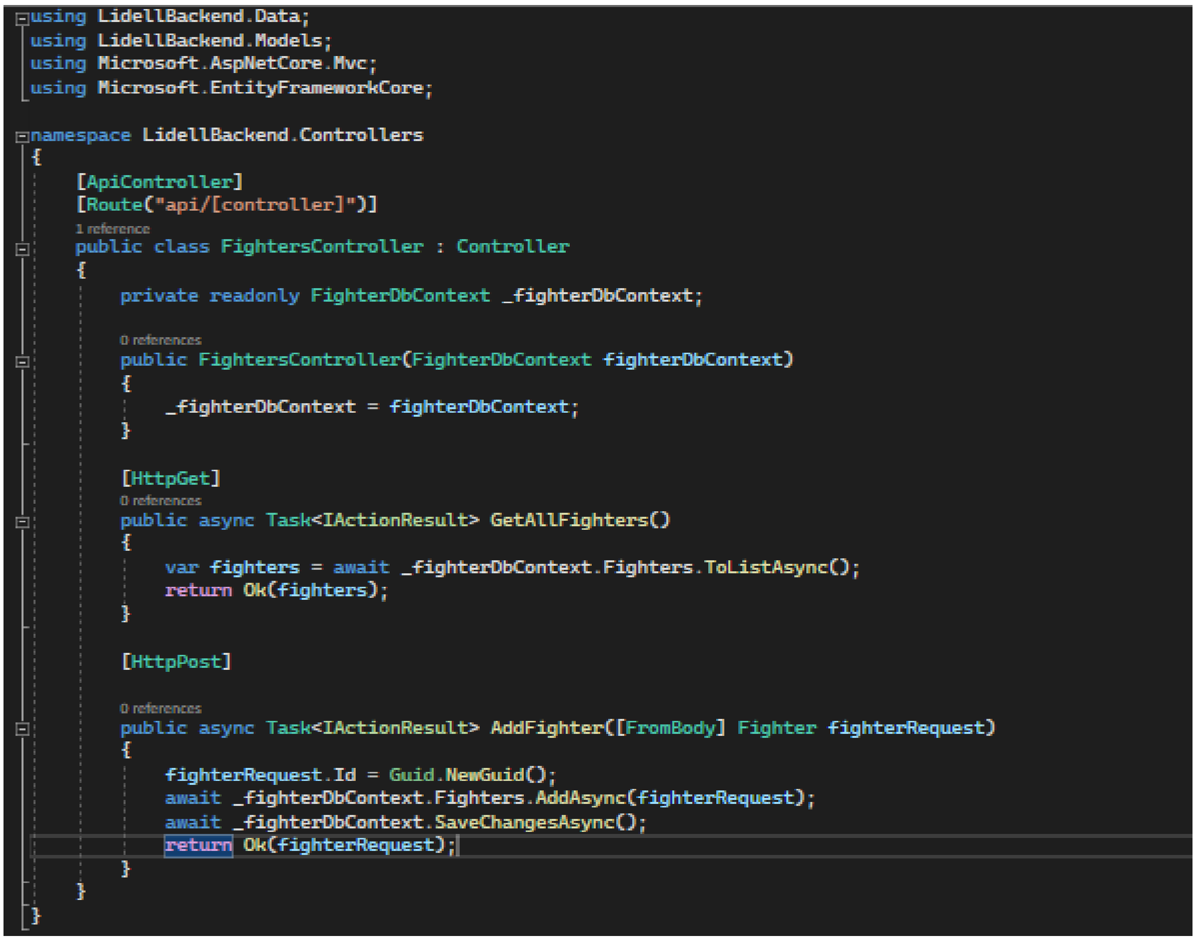


```
namespace LidellBackend.Migrations
{
    /// <inheritdoc />
    1 reference
    public partial class InitialMigration : Migration
    {
        /// <inheritdoc />
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Fighters",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    FighterName = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Division = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Age = table.Column<int>(type: "int", nullable: false),
                    FighterReach = table.Column<int>(type: "int", nullable: false),
                    HomeTown = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    FightStyle = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Record = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Ranking = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Fighters", x => x.Id);
                });
        }

        /// <inheritdoc />
        0 references
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Fighters");
        }
    }
}
```

4.5.3 Controllers:

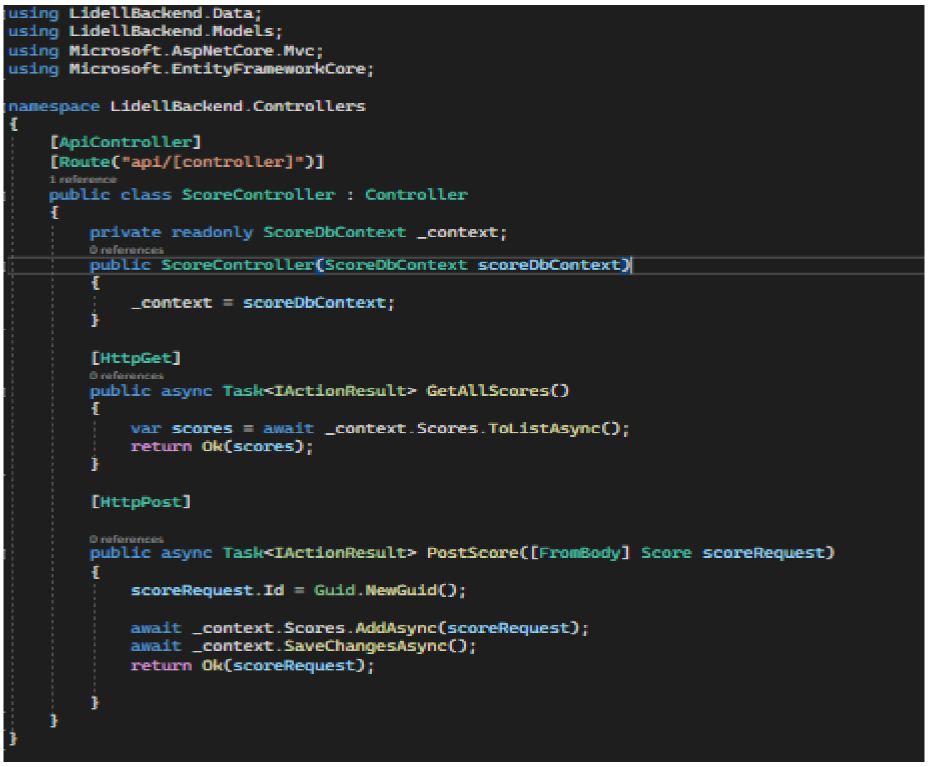
We have two separate controllers for both the scores and fighters. The controllers job is to handle the post and get requests for both the scores and fighters. This is done by using basic HttpPost and HttpGet requests that will be accessed in the frontend.

Fighter Controller



```
using LidellBackend.Data;
using LidellBackend.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LidellBackend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class FightersController : Controller
    {
        private readonly FighterDbContext _fighterDbContext;

        0 references
        public FightersController(FighterDbContext fighterDbContext)
        {
            _fighterDbContext = fighterDbContext;
        }

        [HttpGet]
        0 references
        public async Task<IActionResult> GetAllFighters()
        {
            var fighters = await _fighterDbContext.Fighters.ToListAsync();
            return Ok(fighters);
        }

        [HttpPost]

        0 references
        public async Task<IActionResult> AddFighter([FromBody] Fighter fighterRequest)
        {
            fighterRequest.Id = Guid.NewGuid();
            await _fighterDbContext.Fighters.AddAsync(fighterRequest);
            await _fighterDbContext.SaveChangesAsync();
            return Ok(fighterRequest);
        }
    }
}
```

Score Controller

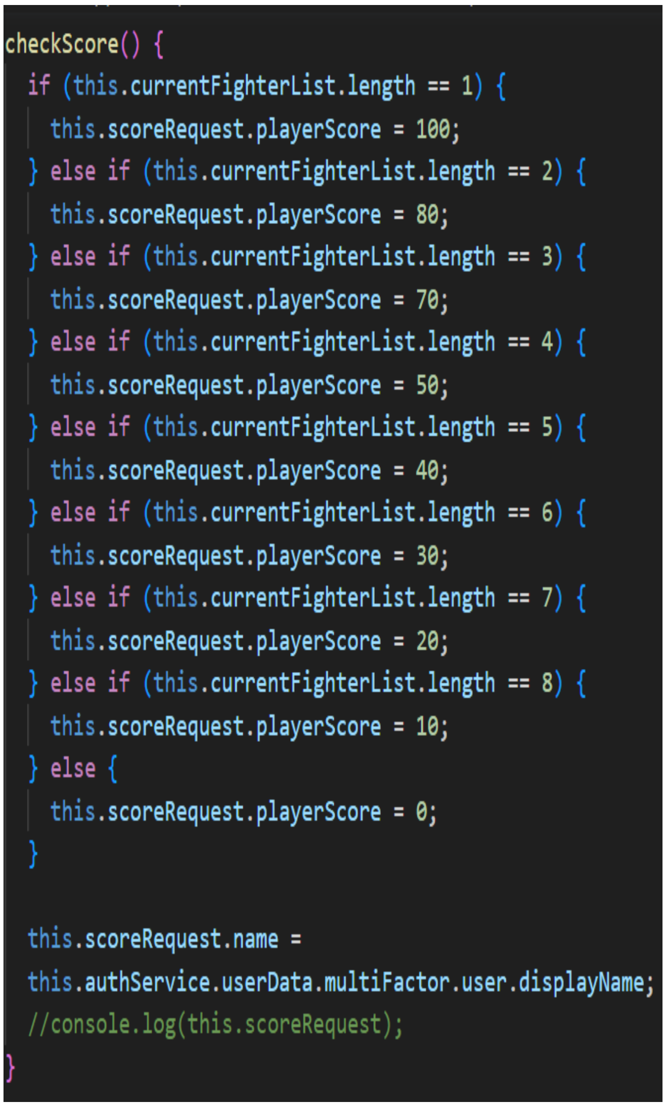


```
using LidellBackend.Data;
using LidellBackend.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LidellBackend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class ScoreController : Controller
    {
        private readonly ScoreDbContext _context;
        0 references
        public ScoreController(ScoreDbContext scoreDbContext)
        {
            _context = scoreDbContext;
        }

        [HttpGet]
        0 references
        public async Task<IActionResult> GetAllScores()
        {
            var scores = await _context.Scores.ToListAsync();
            return Ok(scores);
        }

        [HttpPost]

        0 references
        public async Task<IActionResult> PostScore([FromBody] Score scoreRequest)
        {
            scoreRequest.Id = Guid.NewGuid();

            await _context.Scores.AddAsync(scoreRequest);
            await _context.SaveChangesAsync();
            return Ok(scoreRequest);

        }
    }
}
```

4.6 Leaderboards:

We implemented a scoring system for our game to add a competitive aspect to our game. We made it so the fewer guesses it takes you to guess the correct fighter the more points you score. The scores are then posted to a leaderboard which is displayed on the stats page.

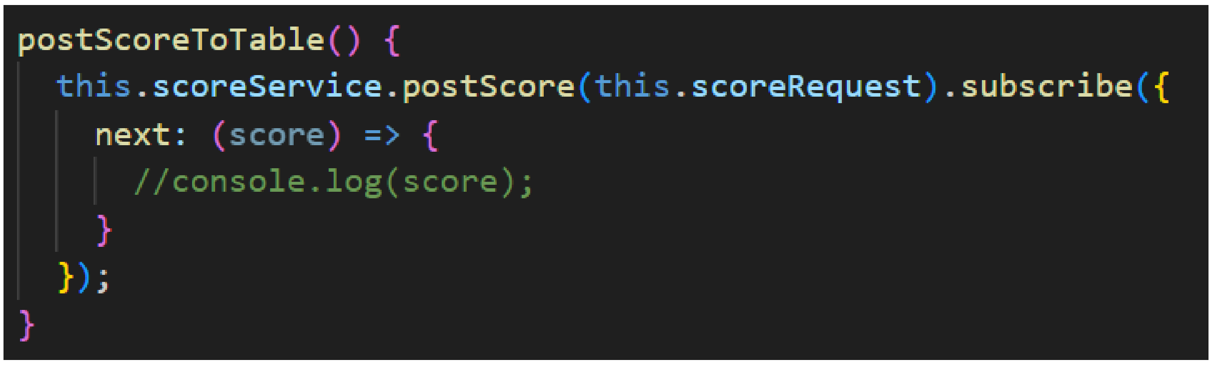
checkScore():



```
checkScore() {
  if (this.currentFighterList.length == 1) {
    this.scoreRequest.playerScore = 100;
  } else if (this.currentFighterList.length == 2) {
    this.scoreRequest.playerScore = 80;
  } else if (this.currentFighterList.length == 3) {
    this.scoreRequest.playerScore = 70;
  } else if (this.currentFighterList.length == 4) {
    this.scoreRequest.playerScore = 50;
  } else if (this.currentFighterList.length == 5) {
    this.scoreRequest.playerScore = 40;
  } else if (this.currentFighterList.length == 6) {
    this.scoreRequest.playerScore = 30;
  } else if (this.currentFighterList.length == 7) {
    this.scoreRequest.playerScore = 20;
  } else if (this.currentFighterList.length == 8) {
    this.scoreRequest.playerScore = 10;
  } else {
    this.scoreRequest.playerScore = 0;
  }

  this.scoreRequest.name =
  this.authService.userData.multiFactor.user.displayName;
  //console.log(this.scoreRequest);
}
```

This function calculates the current score of the player playing, based on the amount of guesses they have taken. It then sets the scoreRequest.playerScore value to the correct value, in the correct form ready to post

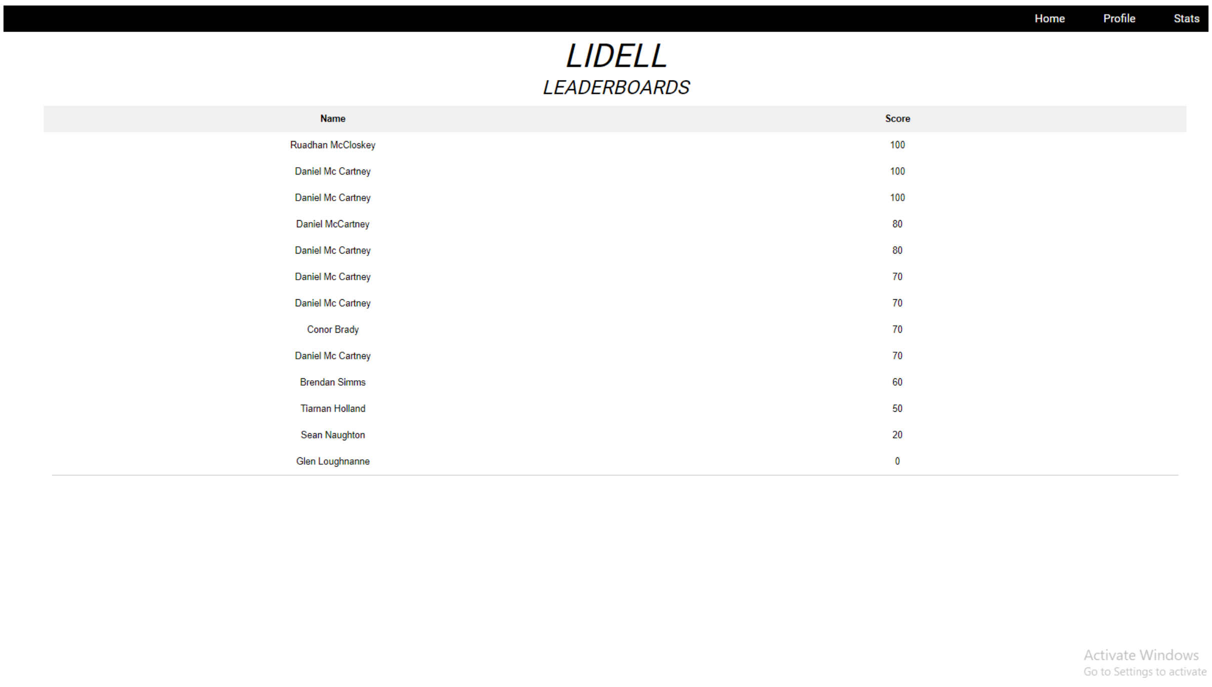
postScoreToTable():



```
postScoreToTable() {
  this.scoreService.postScore(this.scoreRequest).subscribe({
    next: (score) => {
      //console.log(score);
    }
  });
}
```

This function carries out the post request. It will send an object to the leaderboard table, which will then appear in the stats tab.

Leaderboards:



LIDELL

LEADERBOARDS

Name	Score
Ruadhan McCloskey	100
Daniel Mc Cartney	100
Daniel Mc Cartney	100
Daniel McCartney	80
Daniel Mc Cartney	80
Daniel Mc Cartney	70
Daniel Mc Cartney	70
Conor Brady	70
Daniel Mc Cartney	70
Brendan Simms	60
Tiarnan Holland	50
Sean Naughton	20
Glen Loughnanne	0

5 - Problems Solved:

Generating Board Component

One of the first problems we ran into was trying to generate the board component for our game. We could query the database but we couldn’t get our current fighter to show up properly with their attributes in the right place and formatted correctly. Once we select a fighter to guess we wanted their attributes to show up in a board format but it proved to be more difficult than we initially thought. We had to create a board class in our HTML files and we had to style it to our preferences in CSS. Also getting each guess to show up on a new line and have each attribute box be the same size for each guess was also quite difficult.

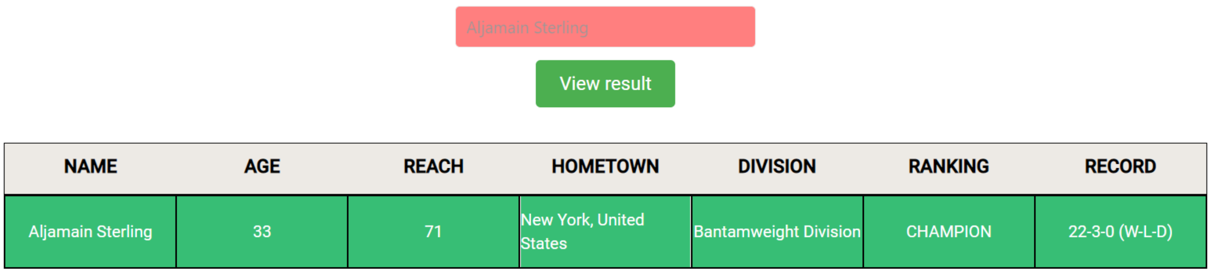
Comparing Divisions and Rankings
Another problem that we faced was comparing divisions and rankings as they were strings so we couldn’t do mathematical comparisons. To fix this problem we created dictionaries for each attribute to create a key value pair for each string corresponding to an integer. Now we could compare the values for each key. This allowed us to implement the close class for these attributes and compare the attributes of the current fighter to that of the random fighter.

Comparing HomeTowns
We also wanted to have it so we could compare the fighters' hometowns to one another. We thought about how we could implement this feature and came to the conclusion that we had to split up the hometown string on the comma that separates the country and city. Now we could check to see if the fighters were from the same country and if they were we could light up that attribute yellow.

6 Testing and Results:
We had to do some thorough testing to ensure that our game worked to a high standard and that it was free of any potential bugs. Considering we were developing a game we thought it would be best to do functional testing on our application. This entailed us playing the game over and over again and testing each component and attribute to ensure that they all worked as they should.

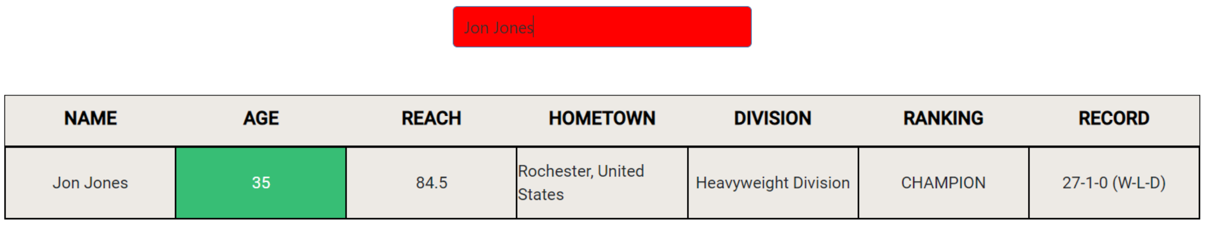
6.1 Functional Testing:
There are 7 attributes that each fighter has (Name, Reach, Age, Record, HomeTown, Division and Ranking). We had to test each of these attributes to see that they would be compared to the random fighter correctly when a user makes a guess. Testing the “Name” attribute was easy enough, we just logged the random fighter to the console and tested an incorrect fighter and then guessed the correct fighter to see if the attributes would show up green for the correct fighter and remain white for the incorrect fighter.

Test for correct fighter Name:



NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Aljamain Sterling	33	71	New York, United States	Bantamweight Division	CHAMPION	22-3-0 (W-L-D)

Test for incorrect fighter Name:



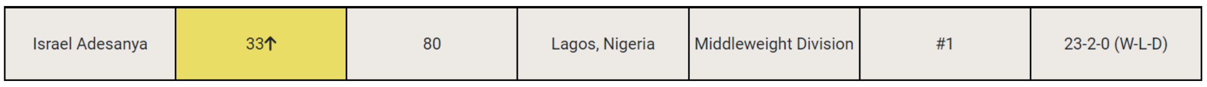
NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Jon Jones	35	84.5	Rochester, United States	Heavyweight Division	CHAMPION	27-1-0 (W-L-D)

As we can see the name attribute shows up white.

Next we tested the (Age, Ranking, Reach and Division) attributes, to test this properly and thoroughly we would guess fighters who are within 2 of the random fighter up or down, and we would check to see if the attribute would show up yellow with an arrow pointing up or down depending on whether the current fighters attribute was more or less than the random fighter. We also tested with fighters who had the same attributes as the random fighter to see if the attribute would show up green. We also tested each of these attributes with fighters that had attributes that were neither close nor the same as the random fighter. We tested each of these attributes thoroughly with a variety of different fighters to ensure that each of the attributes worked as expected.

Here are some examples of the tests that we ran for each attribute:

Test for close fighter Age:



Israel Adesanya	33↑	80	Lagos, Nigeria	Middleweight Division	#1	23-2-0 (W-L-D)

Test for close fighter Reach:

NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Karol Rosa	27↑	67.5↓	Vila Velha, Brazil	Women's Bantamweight Division	#9	16-4-0 (W-L-D)

Test for close fighter Division:

Petr Yan	29	67↓	Krasnoyarsk Krai, Russia	Bantamweight Division ↓	#2	16-5-0 (W-L-D)

Test for close fighter Ranking:

Sean Strickland	31↓	76	California, United States	Middleweight Division	#7↑	26-5-0 (W-L-D)

Test for correct fighter Age, Reach and Division:

NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Colby Covington	34	72	Miami, United States	Welterweight Division	#2↓	17-3-0 (W-L-D)

Test for correct Ranking:

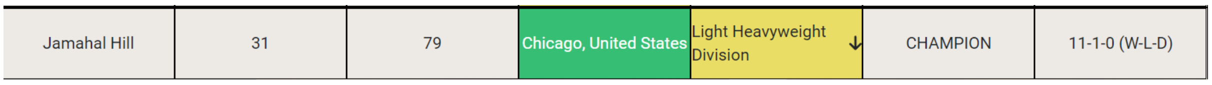
Marvin Vettori	28	74↓	Trento, Italy	Middleweight Division ↓	#4	19-6-1 (W-L-D)

Test for incorrect Age, Reach, Division and Ranking:

Erin Blanchfield	23	66	New Jersey, United States	Women's Flyweight Division	#10	10-1-0 (W-L-D)

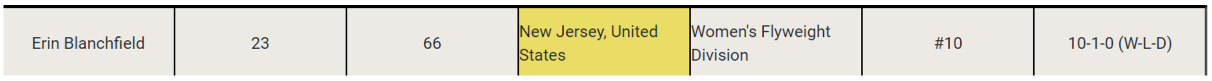
To test the “HomeTown” attribute we found fighters from the same country but different cities to see if they would show up yellow and we also tested fighters from the same country and city as the random fighter to see if it would show up green as expected.

Test for correct HomeTown:



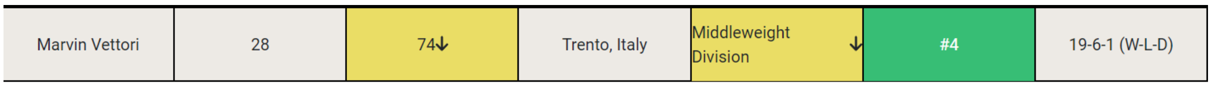
Jamahal Hill	31	79	Chicago, United States	Light Heavyweight Division ↓	CHAMPION	11-1-0 (W-L-D)

Test for close HomeTown:



Erin Blanchfield	23	66	New Jersey, United States	Women's Flyweight Division	#10	10-1-0 (W-L-D)

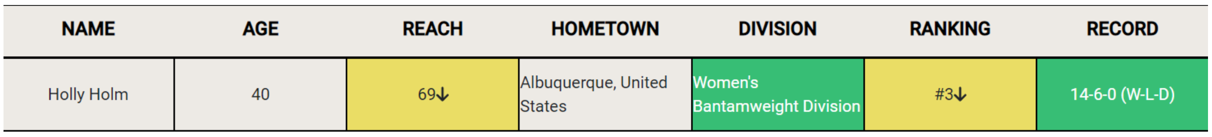
Test for incorrect HomeTown:



Marvin Vettori	28	74↓	Trento, Italy	Middleweight Division ↓	#4	19-6-1 (W-L-D)

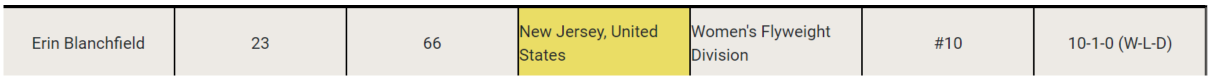
The “Record” attribute was just tested to see if fighters with the same record would make the attribute show up green.

Test for Correct Record:



NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Holly Holm	40	69↓	Albuquerque, United States	Women's Bantamweight Division	#3↓	14-6-0 (W-L-D)

Test for incorrect Record:

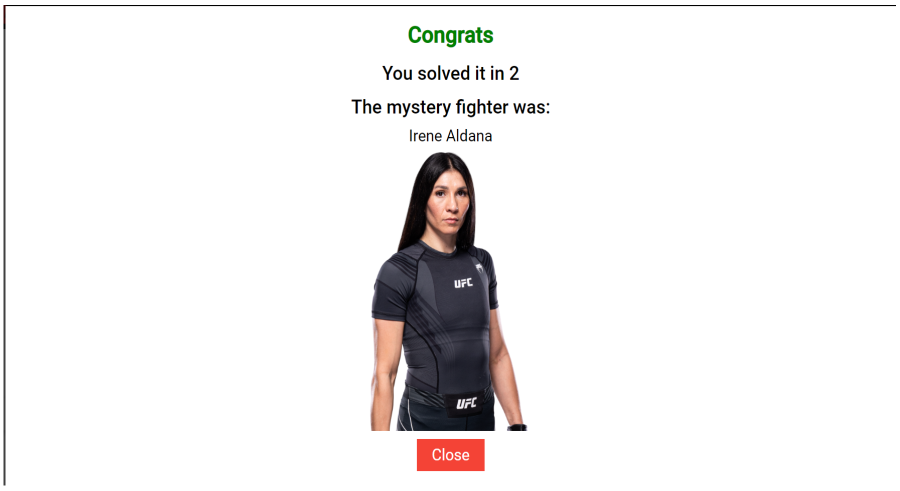


Erin Blanchfield	23	66	New Jersey, United States	Women's Flyweight Division	#10	10-1-0 (W-L-D)

We now had to run tests to see if the user guessed correctly what would happen. We wanted it so the user would be prompted with a success message and that they would be unable to keep guessing fighters once the game is over.

Test for user winning the game:

Success message pops up:



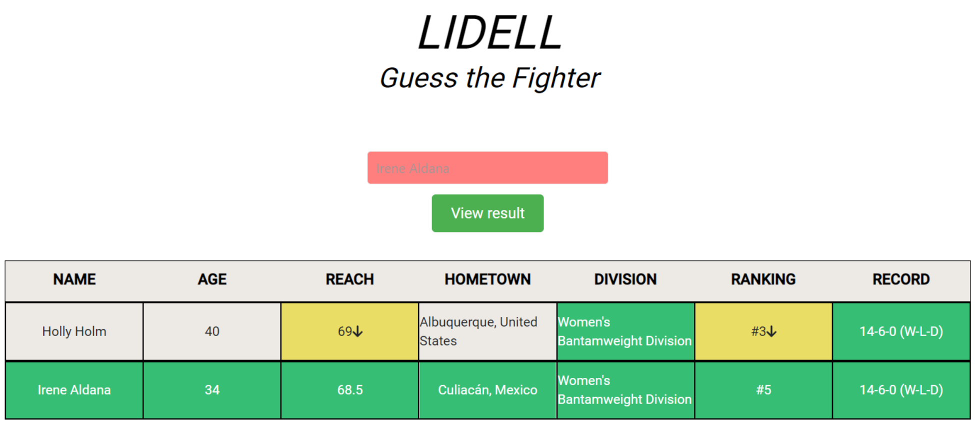
Congrats

You solved it in 2

The mystery fighter was:

Irene Aldana

Search bar can no longer be used and the user can open and close the success message also the board shows up green for all attributes as expected:



LIDELL

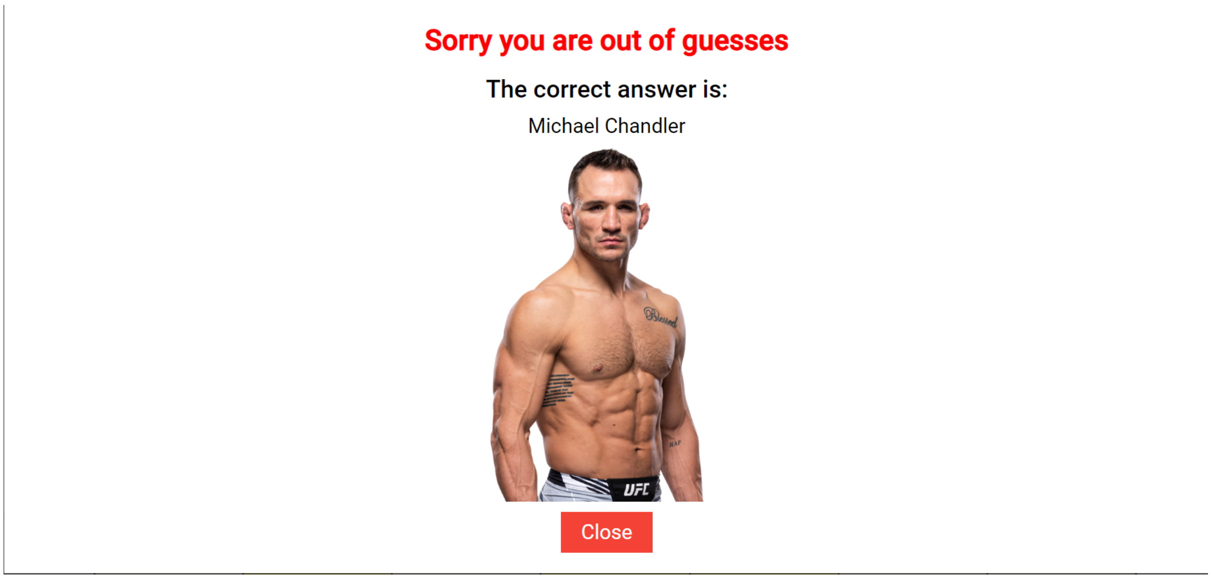
Guess the Fighter

NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Holly Holm	40	69↓	Albuquerque, United States	Women's Bantamweight Division	#3↓	14-6-0 (W-L-D)
Irene Aldana	34	68.5	Culiacán, Mexico	Women's Bantamweight Division	#5	14-6-0 (W-L-D)

When a user runs out of guesses we wanted a failure message to pop up indicating that the user has lost the game and we also wanted it so they wouldn't be able to keep guessing fighters once they have run out of guesses.

Test for user losing the game:

Failure message:

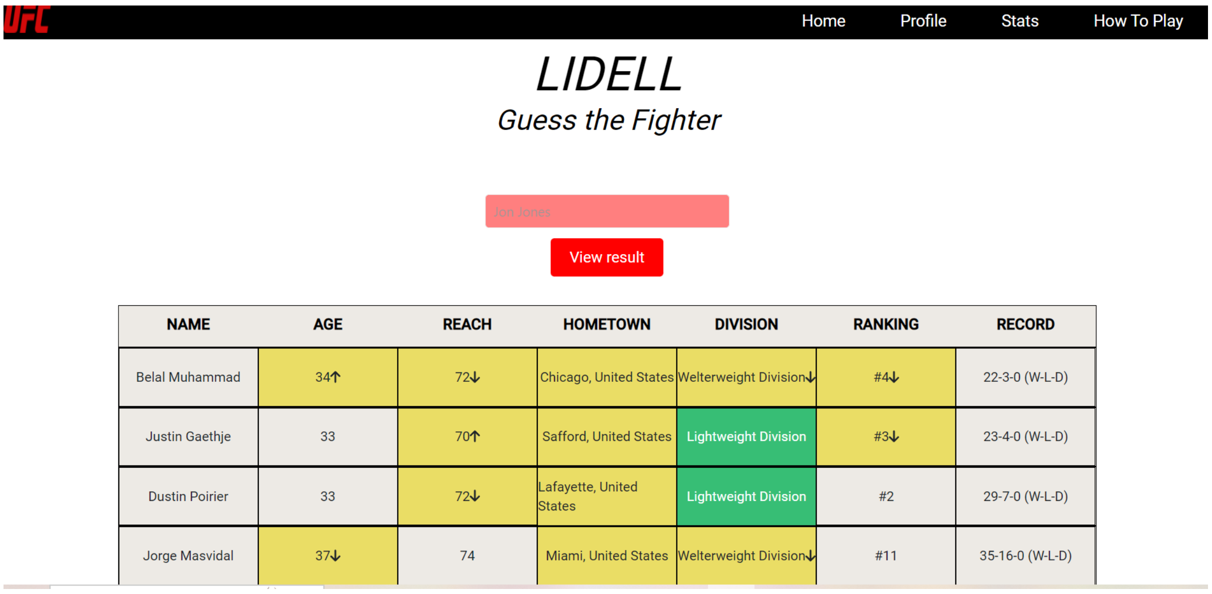


Sorry you are out of guesses

The correct answer is:

Michael Chandler

The user can’t guess anymore fighters and can open and close the failure message:



LIDELL

Guess the Fighter

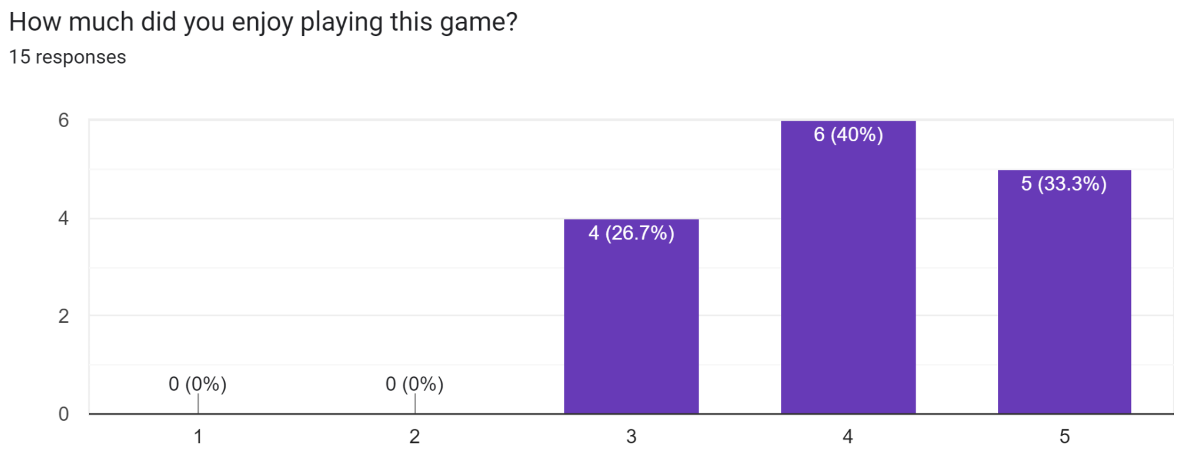
NAME	AGE	REACH	HOMETOWN	DIVISION	RANKING	RECORD
Belal Muhammad	34↑	72↓	Chicago, United States	Welterweight Division↓	#4↓	22-3-0 (W-L-D)
Justin Gaethje	33	70↑	Safford, United States	Lightweight Division	#3↓	23-4-0 (W-L-D)
Dustin Poirier	33	72↓	Lafayette, United States	Lightweight Division	#2	29-7-0 (W-L-D)
Jorge Masvidal	37↓	74	Miami, United States	Welterweight Division↓	#11	35-16-0 (W-L-D)

These are some of the tests that we ran on our game to ensure that it worked as expected for each attribute and component of the game. We ran even more tests than this to ensure that the game worked as expected and the way we wanted it to work. All of the tests that we ran gave us the results that we were expecting. If they did not then we fixed the code for it to work the way we wanted it to.

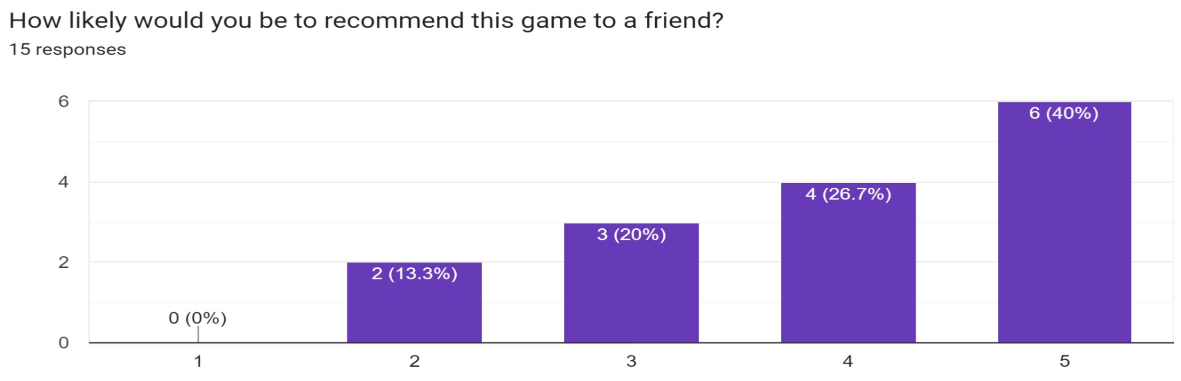
6.2 User Testing:

For our user testing we asked some of our friends and family that were familiar with the UFC and watched the sport to test our game for us. We wanted to know if they enjoyed playing the game and what they would change about the game. Since our app is not hosted online we used our laptops to show them the game and get them to play it for us. Afterwards we sent them an anonymous questionnaire for them to fill out so we would have some feedback.

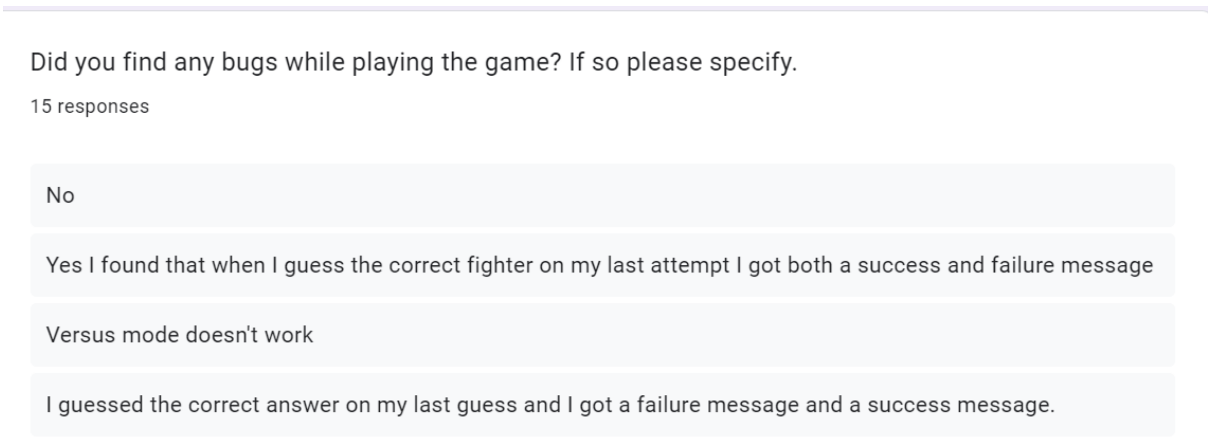
Here are the results of the questionnaire:



There is some mixed feedback on how much the users enjoyed playing our game however nobody didn't enjoy playing the game which was good and only a few were in between.



Majority of our users said they would be likely to recommend this game to a friend and only 2 were unlikely to recommend the game to a friend.



Did you find any bugs while playing the game? If so please specify.

15 responses

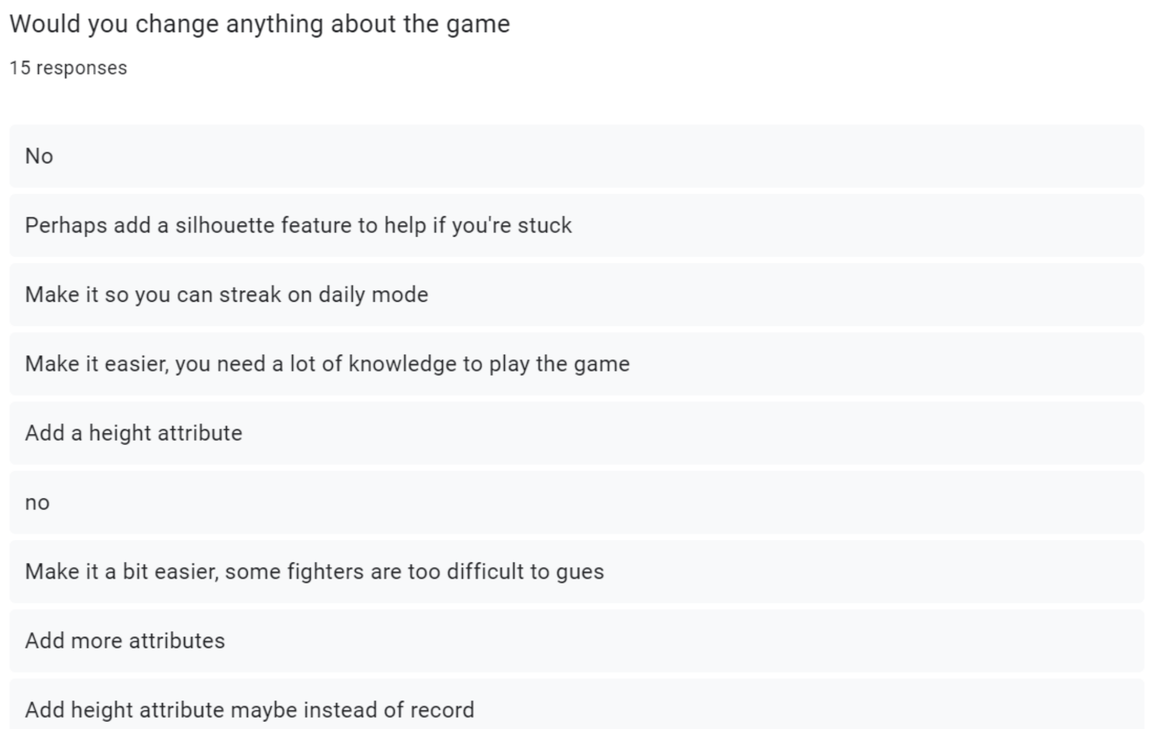
No

Yes I found that when I guess the correct fighter on my last attempt I got both a success and failure message

Versus mode doesn't work

I guessed the correct answer on my last guess and I got a failure message and a success message.

Next we asked them if they found any bugs, one user said that versus mode doesn’t work which we already knew. 2 other users said they received both a failure and success message when guessing the correct fighter on their last guess. This helped us a lot as we didn’t run into this error when we did our initial testing. Thanks to their feedback we were able to fix this error.



Would you change anything about the game

15 responses

No

Perhaps add a silhouette feature to help if you're stuck

Make it so you can streak on daily mode

Make it easier, you need a lot of knowledge to play the game

Add a height attribute

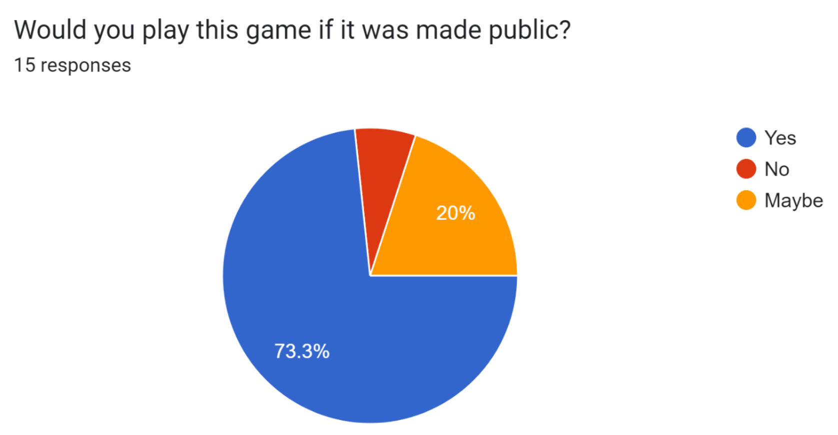
no

Make it a bit easier, some fighters are too difficult to gues

Add more attributes

Add height attribute maybe instead of record

Next we asked what they would change about the game, there were some interesting answers to this question with some good ideas for things we could implement. We found that we didn’t have a height feature as one of our attributes which we could consider adding at some point. Some users found the game too difficult which is understandable as you need a lot of knowledge on the sport to play the game. The silhouette feature is also a good idea as we already have the fighter pictures so this could be an interesting feature to add to our game.



Majority of our users said they would play the game if it was made public which makes sense because our testing pool was mostly people we knew who followed the UFC and would be familiar with the fighters.

7 Conclusion:

7.1 Future Work:

Set up application on a server:
Our next steps as far as this project goes is to deploy the application online. We have an AWS server up and running and linked to the project using Amplify but it is unable to query from the database as of right now. However once we finish up the project and everything we will be able to fix that issue and deploy our project for the world to see. We will be able to promote the app through social media such as Instagram and Twitter. There is a large community of MMA and UFC fans online that we could reach out to to spread the word about our game.

Implement Multiplayer:
We were supposed to implement multiplayer functionality in our application however this proved to be more difficult than we anticipated. Instead we decided to implement leaderboards to keep the game competitive for users. We tried to implement multiplayer but we didn't have enough time to finish implementing this feature. Had we had more time we would have focused more on this aspect but we believe that we have enough to keep the game interesting for the user as it is.

Add a Silhouette Feature
From our user testing one of the users suggested adding a silhouette feature to our game to help people who are struggling to get the answer. This is an interesting concept that we could look into in the future. We have photos for each fighter so generating the photo would be no problem and we believe that it wouldn't be too hard to black out the photo so you can't see exactly who it is. However this feature would be very helpful for those struggling to get the answer it may nudge them in the right direction.

7.2 Final Conclusion:

As we finish off the last bits of this project we can't help but feel sad that it's over although the project proved to be difficult and frustrating at times we both really enjoyed solving problems together and seeing our game develop as we made progress. The fact that we chose to do a project on something that we like made the whole process a lot easier and more enjoyable to create. We also have made a game that we enjoy playing. For our testing we had to play the game a number of times and we both enjoyed playing the game as we tested it, we even found ourselves losing the game quite a bit as we don't know every fighter on the roster. Overall we both gained a lot from this project and feel it will benefit us greatly as we head into our careers after college.

7.3 Self Reflection:

Ruadhán

For myself it was nice to be able to learn a new language in TypeScript and it was also very interesting using Angular as it is a framework that I was not familiar with before we began this project. This project has helped me develop my programming skills to another level and has broadened my horizons as far as my ability to code in different languages. I have now found that TypeScript is one of my favourite languages to use now and I love working with Angular also. Daniel helped me a lot and I learned a lot from him also as he was already familiar with the language and framework from his internship.

Daniel

For myself it was a great experience coding this project. It was a project both of us were passionate about and I wanted to make sure it came out the best way possible. It was nice to use the skills I had gained from my INTRA placement in real college work as we coded in the angular framework on INTRA. I also got more familiar with c# and the .net framework in the backend side of the project, which was great for me as that is one part of the internship that i struggled with. This project has given me confidence and I am proud of how it turned out.