LMS

CA400 4th Year Project - Technical Guide

Students:
Tom Callaghan (16449672)
Philip Donnelly (17518149)
Supervisor:
Dr Donal Fitzpatrick
Date: 01/05/2021

Abstract: *The overview of our project Last Man Standing is to create a fully automated version of the well-known competition "Last Man Standing". Last Man Standing is a league based game where players pick a Premier League team each week to represent them for that week. If the player's team wins their game against the opposition, the player proceeds to next week's stage. If the player's team of choice loses or draws against their opposition, the player is knocked out and does not proceed onto the next stage. The last player standing wins the competition/prize. This web application automates this whole process to aid charities or organisations alike to perform this competition on their terms. This document contains a technical guide to the web application, how it was built and the reasons behind building it.*

Table of Contents

1 Introduction

1.1 Overview

Last Man Standing is a web application built for desktop and other devices. The main idea behind *Last Man Standing* is to eliminate the endless human hours put into maintaining and running a last man standing competition. Gone are the days of endless spreadsheets and human error.

Last man standing is a well-known competition where players pick one Premier League team each week. If this team wins they proceed to the next round. If the team draws or loses they will be eliminated. You can only select each team once throughout the competition. The last person standing wins the competition.

Within the application, each user will have their own account. They will be able to keep up to date with all information from the Premier League such as the standings, fixtures and results. Each user will also be able to select their favourite team and receive up to date news as the articles are released on Sky Sports.

Users will be able to create or join as many leagues/competitions as they desire. This is all free of charge... for the moment! After creating or joining a league the user will be presented with important information about that league, for example, how many people are left/eliminated, are the leagues locked or can you pick your team and so on. When participating in a game week you can see all other participants in the league. This way they can keep track of how many people are left and how close they are to winning the competition. Once the league is locked after the game week deadline the participants will be given an overview of how many people selected each team. This way they can keep throughout the matches to see how many people are being eliminated if a team loses or draws their fixture.

If the user has created a league, they become the league's admin. There is additional functionality for admins on their league. They can perform the following actions on the league:

- Close the league so no more people can join
- Copy and send the invitation code to their friends
- Remove a user from the league
- Reset the league
- Delete the league

The rest of this guide will explain exactly why this service was created and how we developed it. We wish you the best of luck in any competitions you participate in on www.mylastmanstanding.xyz.

1.2 Glossary

SRP -	Secure Remote Password protocol
User Pool -	A User directory for AWS Cognito
Amazon Web Services (AWS) -	Cloud Computing Provider
EC2 Instance (Elastic Computing Cloud) -	A virtual server partitioned on AWS
CI/CD Pipeline -	Continuous integration and continuous delivery pipeline
Cron Job -	Time based job scheduler which executes jobs at a specified time
Axios -	A promise-based HTTP client for the browser and Node

2 Motivation

There were several reasons for choosing this application. The first of which was being a part of a competition like this before, we saw the amount of work that had to be done to keep the competition running. Competitions were also not able to scale as much as they would have liked due to the amount of work that had to be put into maintaining it as the number of participants grew.

Another motivation was just how many competitions we were partaking in. With over 8 different active *Last Man Standing* competitions being participated in by the two of us, we felt there was quite a large target market for this application. From work organisations, charities to family and friends competitions. We had been approached by two different organisations about their interest when we spoke about developing this application. This gave us the motivation to build an application of this kind and potentially continue it as a business.

The lack of competitors was also another reason for the motivation. From our research, we could find one other organisation that had attempted this type of application but we felt we could build something far superior.

Last Man Standing removes the endless hours of inputting numbers and picks them into endless spreadsheets for competition admins. It automates the whole process and gives the participants a much better picture of how the competition is going.

3 Research

3.1 Cloud Computing Provider

One of the more difficult decisions that had to be made when researching this development was deciding which cloud computing provider to base the development around. Cloud computing providers have come so far in recent years and we knew no matter which provider we choose that we would be able to develop a high standard application. However, we needed to take into consideration several different factors when making this decision. These factors can be seen below:

- Cost (Student Budget)
- Ease of use
- Scalability & performance
- Serverless options

After receiving a positive response to some questionnaires we sent to family and friends we found that there would be a lot of interest in the application. This meant we needed to be cautious of scalability. We decided to go serverless to allow for scalability and only pay for what we were using.

Although we found that Azure and Google Cloud Platform were powerful providers we decided to go with Amazon Web Services. This was down to several reasons which are listed below:

- Ease of use
- Serverless options
- Cost (free credits from AWS Educate)
- Previous experience

3.2 Front End Framework

As this project has a user-facing UI we felt it was vital for us to pick the correct framework for the user and us. We began to look at all of the different frameworks such as React, Angular and Vue. After a discussion about them and all of their advantages and disadvantages, we felt most confident picking React as our front end framework. We chose this framework as it is a framework that we have both worked with previously. We felt the prior knowledge of React would be vital for the application as we were hoping to conduct a longitudinal study. Due to this, we felt it was vital that we introduce the least amount of bugs possible to production and this would be achievable using React compared to the other unfamiliar frameworks.

3.3 Back End Language

From the moment we decided to go with serverless architecture and AWS, the next decision was to decide on the back end language. The languages that are supported by AWS Lambda can be seen below.

- Java
- Go
- PowerShell
- Node. js
- C#
- Python
- Ruby

Although we both had some previous knowledge of Java, Node and Go, Python was chosen as the primary language. We chose this because we felt it is our strongest programming language and that is one of the most popular languages today.

3.4 UI Component Library

As *Last Man Standing* has a user-facing interface we felt that this needed to look professional. We felt using a UI component library would be best compared to creating our UI components using basic HTML and CSS. When looking into the different libraries the two that stood out to us were React Bootstrap and Material UI. *Last Man Standing* is primarily made with Material UI as it provided far more components than React bootstrap. Material UI is also a responsive framework. We knew that many users would access *Last Man Standing* on their mobile so we felt it was very important to have a responsive application to cater to these needs.

We occasionally use React Bootstrap also. This would be the case if Material UI could not provide us with the desired component or a particular style.

3.5 Premier League APIs

One of the most important components of *Last Man Standing* is retrieving the Premier League Information. It was critical to choose the correct API as there is high reliability in the information that it provides to the application. We first created a document of all the API's which provide football data comparing each API, on cost, API calls per hour and returned information. The document can be seen here: API Research Document

Since we have limited student budgets and the API will be called several times a day we needed to find an API that has a low cost or a free tier system that we can utilise. The API is pinged every hour to see if there were any updates in the score for the final game of the game week therefore we also needed to ensure we are not limited to the number of API calls we can make. Finally, the information returned needed to be useful. We needed the fixtures, results and standings to be returned from an API with in-depth information about these three headings. To do this we created a simple Python script that would print the response to our terminal so we could inspect the response and decide if it was a viable option.

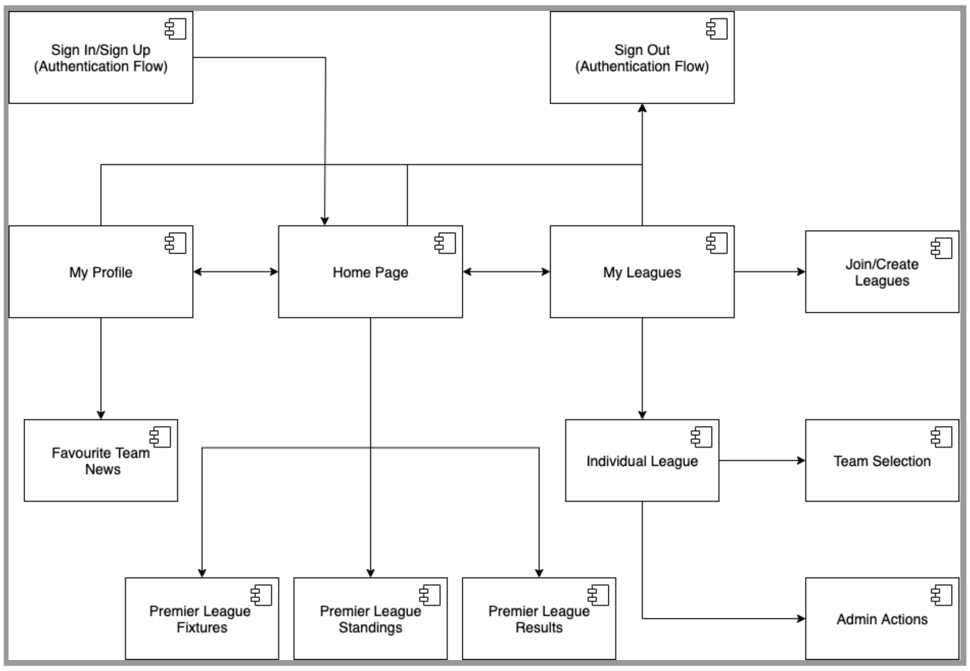
```python
import HTTP.client
import JSON
from datetime import datetime

connection = http.client.HTTPConnection('api.football-data.org')
headers = { 'X-Auth-Token': 'XXXXXXXXXXXXX' } #Removed Auth Token
connection.request('GET', '/v2/competitions/PL/', None, headers )
response = json.loads(connection.getresponse().read().decode())
connection.request('GET', '/v2/competitions/PL/matches', None, headers )
response2 = json.loads(connection.getresponse().read().decode())

conn = http.client.HTTPSConnection("api-football-v1.p.rapidapi.com")

headers = {
    'x-rapidapi-key': "XXXXXXX",
    'x-rapidapi-host': "api-football-v1.p.rapidapi.com"
    }

conn.request("GET", "/v2/fixtures/team/33", headers=headers)

res = conn.getresponse()
data = res.read()
with open ("test-api.json", "w") as file:
    file.write(json.dumps(response))

for fixture in response2["matches"]:
    print(fixture)are
```

After this investigation, we decided to go with www.football-data.org. We decided to choose this API as it provides in-depth information about the Premier League standings, fixtures and results. It also has a free tier that provides 10 API calls per minute which were more than enough for our needs. A bonus was the fact that it could provide game odds which we could incorporate into our probability script. These odds were at an extra cost however, after emailing the provider they generously provided the odds at no extra cost.

4 High-Level Design

In this section, the design web application will be outlined at a high level. Investigating each component and its role within the web application. Each component will be explained accompanied by some of the following diagrams depending on its role: data flow diagram, sequence diagram, use case diagram.

4.1 Component Diagram

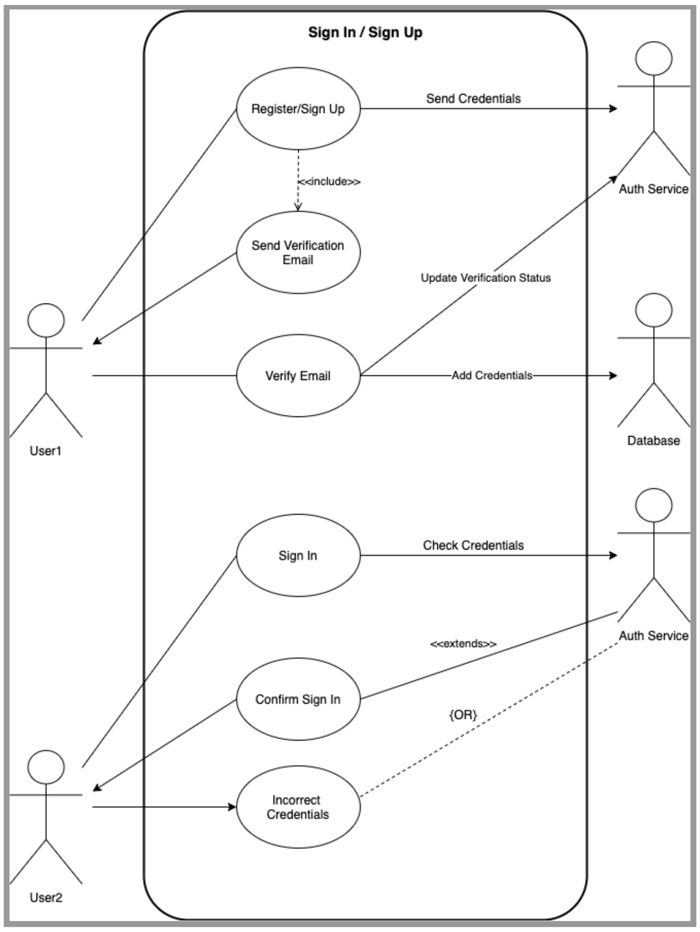


The above diagram conveys the overall component diagram of *Last Man Standing.* It displays the role which that component plays and its interaction with other components in the system. Each component will be explained in more detail below.

4.2.1 Sign In / Sign Up Components

If a user wants to participate in a *Last Man Standing* league/competition they must first register for an account. They navigate to the Sign up page where they will be prompted to fill out a form with a unique username, email address and password. Once successfully logged in the user will be redirected to the Sign in page and be prompted to enter their username and password.

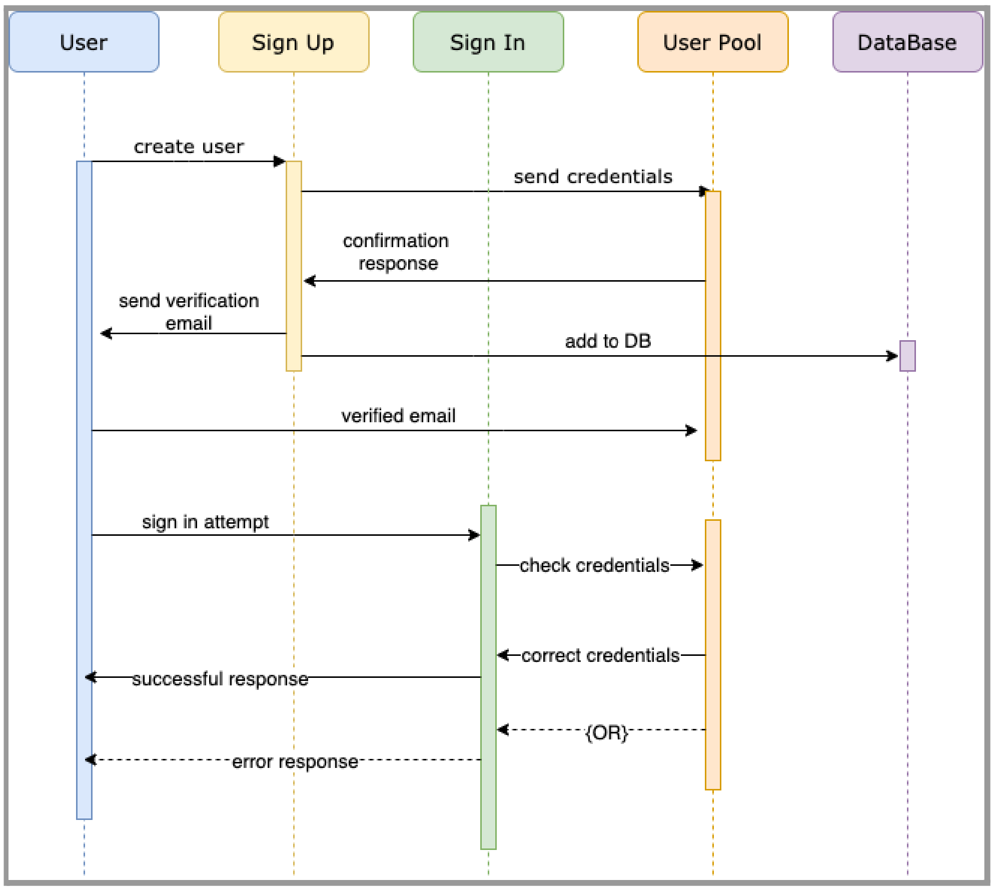
Use Case Diagram



The image above displays a use case for a user signing up/signing in to *Last Man Standing*. To start the user will fill out all fields for registration. From there, the credentials will be sent to our authentication service which will store the information in a user pool. This user will not be verified in the user pool, they will need to verify their email address first. Once the user verifies their email address, their information will be sent to our database for display purposes on the web application. They will then be able to sign in.

As we can see the “User 2” in the diagram attempts to sign in. The credentials are checked with our authentication service. There are two outcomes, either the credentials will be correct and the user can proceed as an authenticated user or the credentials will be incorrect and they will be prompted to try again.

Sequence Diagram



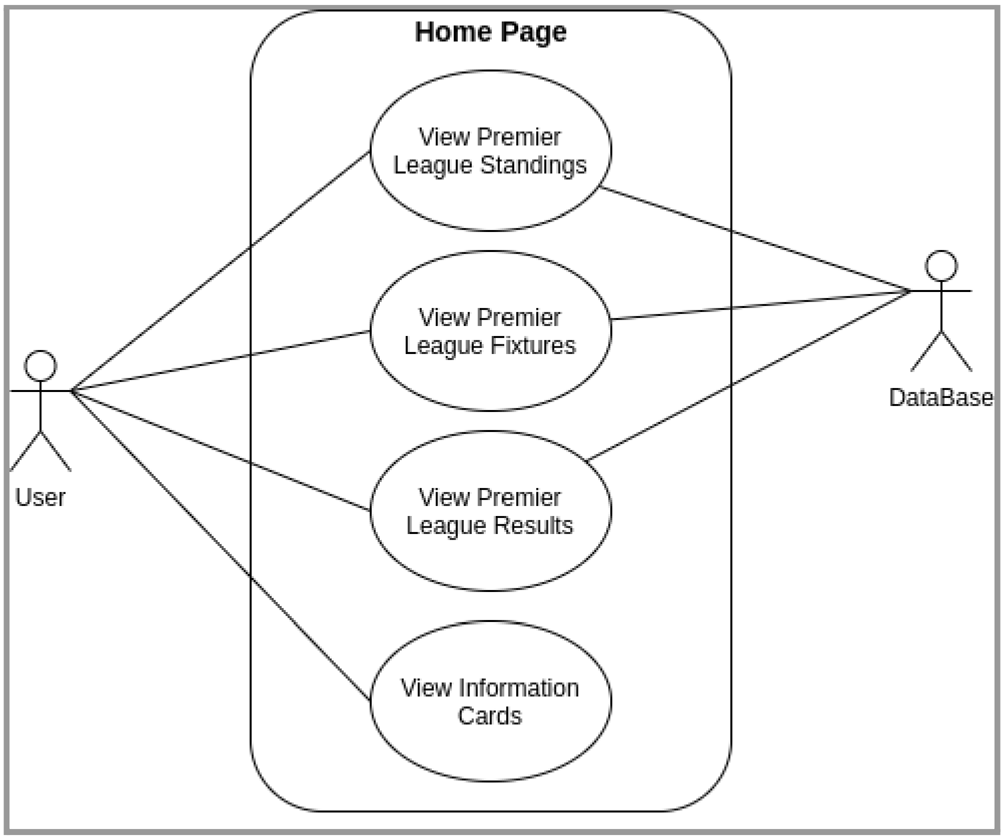
The above diagram shows the sequence of signing in/signing up. As we can see the user will first attempt to sign up which will then send the credentials to the user pool. The verification email will then be sent to the user. The user will then verify their email and become a confirmed user. Their non-sensitive account information will then be stored in the database.

Once successfully signed up the user can attempt to sign in. The credentials are checked against the user pool before allowing them to access the application. If the credentials are incorrect the user will be shown an error notifying the issue with their attempt.

4.2.2 Home Page Component

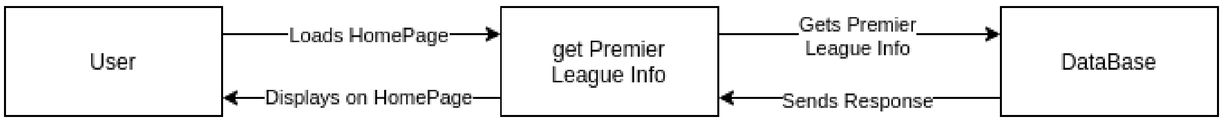
When a user first enters *Last Man Standing* they are greeted by the home page. The homepage is the same for both logged in and logged out users. When entering the application, the Premier League standing, fixtures and results are requested and displayed in three different tables. The Premier League standings is an up to date table of the Premier League and the position of each in accordance to others. The Premier League fixtures are client-side the next set of fixtures which *Last Man Standing* is counting as a game week. The Premier League results are the results of the previous fixtures which *Last Man Standing* has deemed to be a game week. Two simple cards are also displayed giving the user information about what *Last Man Standing* is and how to get started.

Use Case Diagram


Home Page
View Premier League Standings
View Premier League Fixtures
View Premier League Results
View Information Cards
User
DataBase

Above is a use case diagram for the home page component of *Last Man Standing*. The home page does not have many use cases as its main use is to display useful information to the user when they first enter *Last Man Standing*. When the user enters the home page they can view the different tables which provide Premier League information. These tables display information received from the database. The user can also view the information cards which help to better understand what *Last Man Standing* is.

Data Flow Diagram

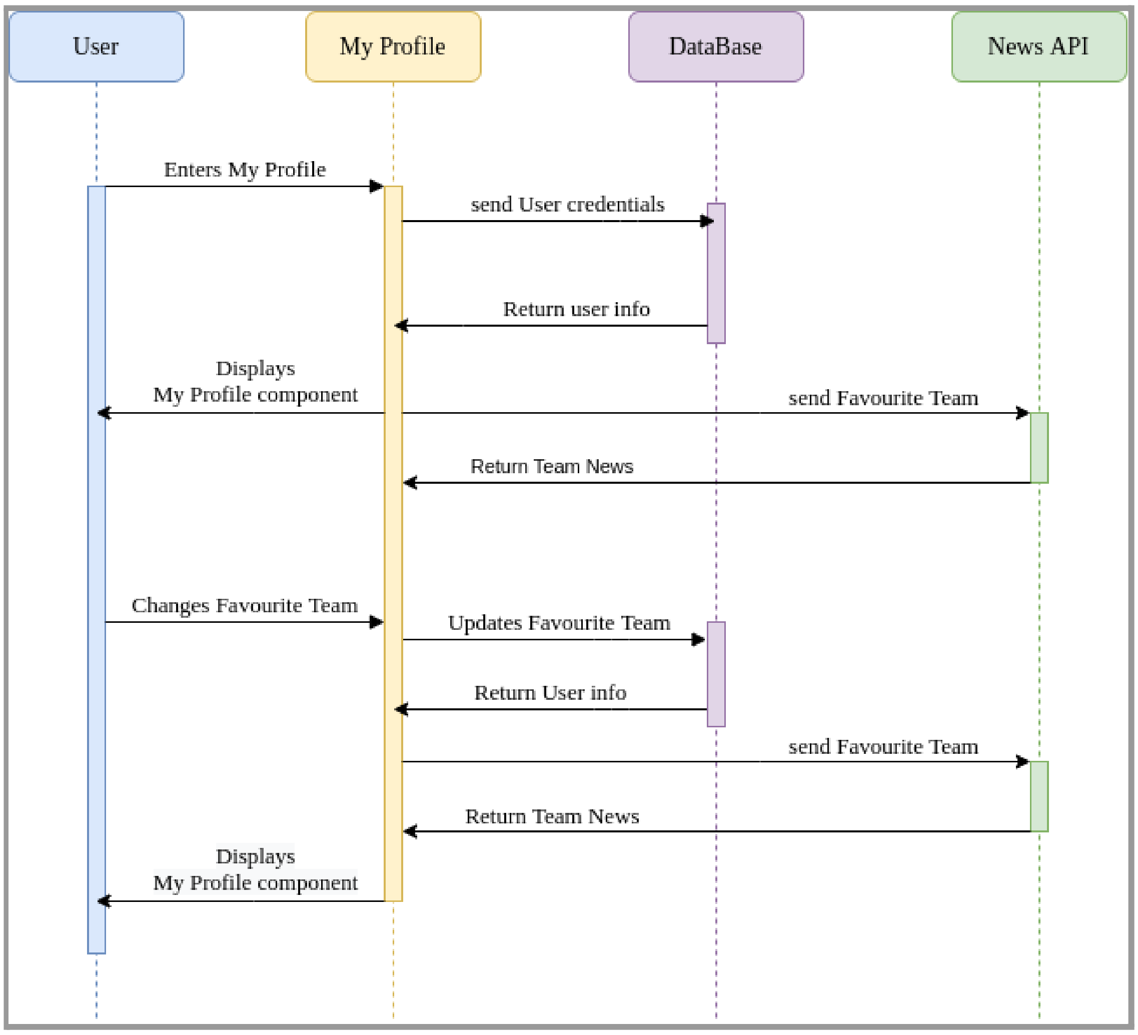


When a user loads the home page an API GET request is sent to a database to get the Premier League Info which will be displayed in the tables. The Database will receive this request and send the Premier League info as a response to the GET request which can then be rendered on the Home Page.

4.2.3 My Profile Component

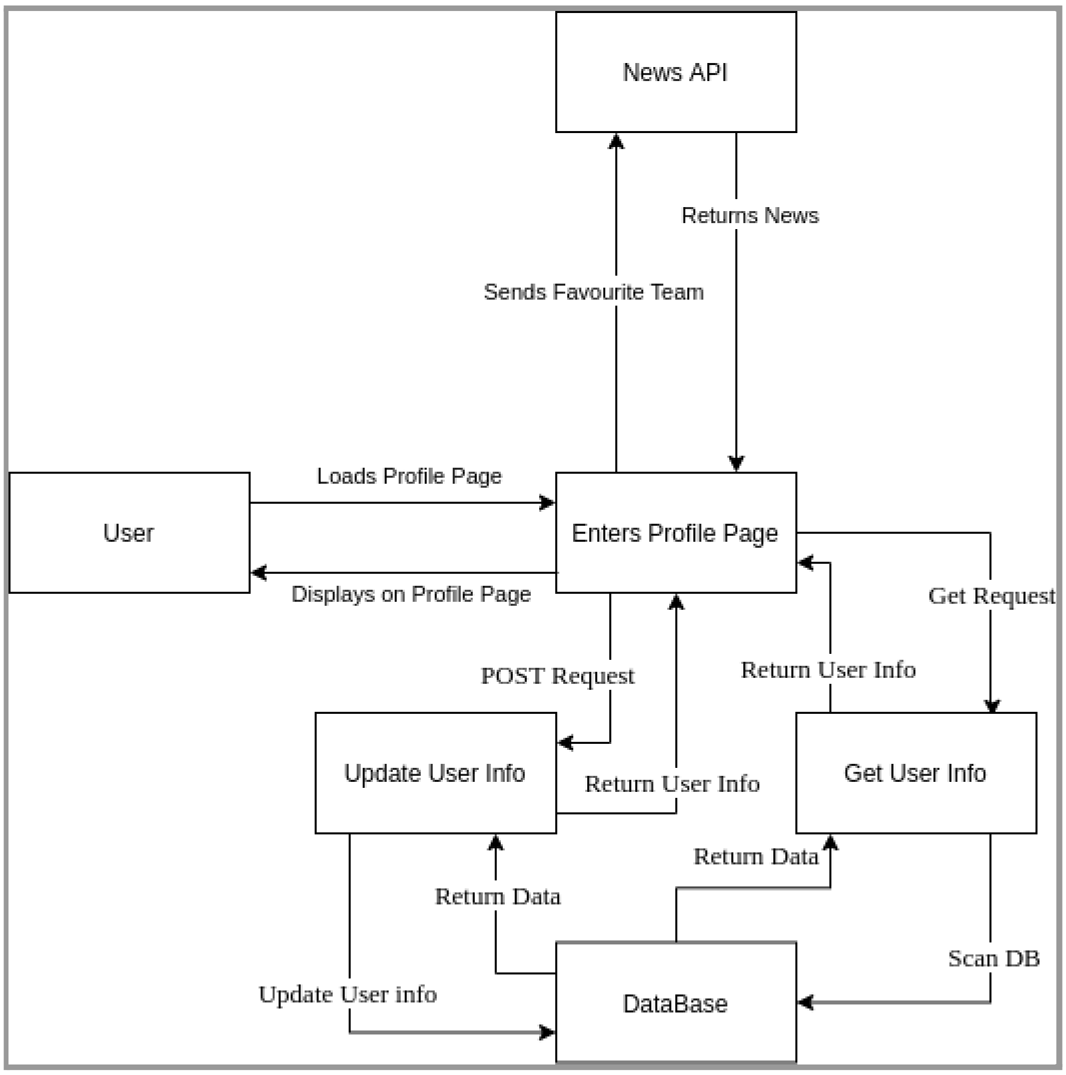
The My profile component gives users a more in-depth view of their journey on *Last Man Standing.* It provides the user with their user information such as username/email. The profile page will also provide the user with a list of all their active competitions and their win ratio. Finally, the My Profile component also provides the user with recent news about their favourite team which they can choose/change at any given time.

Sequence Diagram



When a User enters the My Profile component the user's credentials will be used in an API GET request to the database to display the user's information. Once the user's information has been returned and if the user has a favourite team an API POST request will be sent to the news API. The most recent news will then be returned and all information will then be displayed to the user. If a user has not picked a favourite team they may do so which will send an API POST request to the database which will update the user's information with their favourite team. This updated information will then be sent as a response. Once the My Profile components have received the response it will then send their new favourite team as a POST request to the news API.

Data Flow Diagram

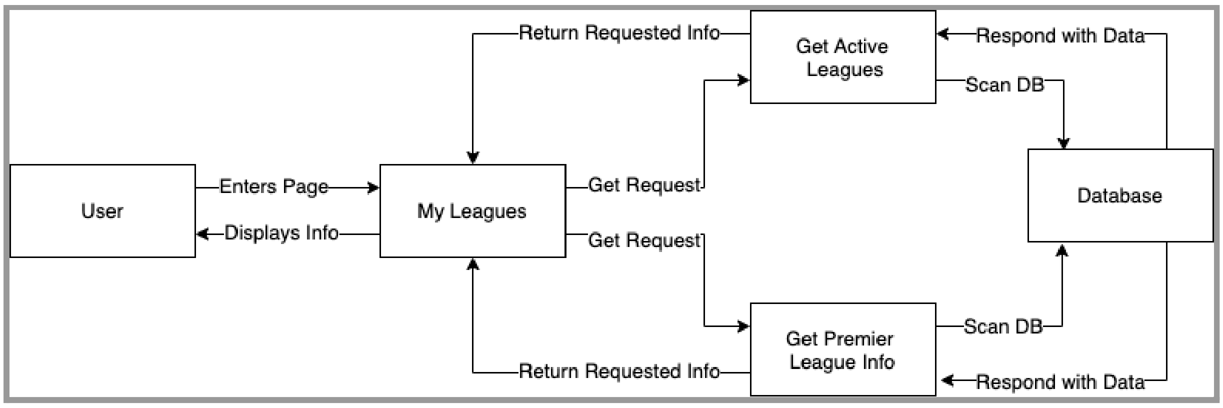


When the user enters the My Profile component a POST request is sent straight away to get the Users info. This uses the user’s unique sub number to scan the database for their information. This information is then returned to the front end to be rendered. For the News, after receiving the user’s information a request is sent to the news API which returns the 20 latest news articles about the given team. This will then be rendered in the front end. A user can change their favourite team which will send a POST request to the database. The database will be scanned with their sub, their new favourite team will be written into the database. This new user info is then returned to the Profile Page component and the News API has triggered again.

4.2.4 My Leagues Component

The My Leagues component gives the user an in-depth overview of all leagues they are currently in. The table that is displayed gives information such as leagueID, current player status and current pick. The user can also access the create/join leagues and individual leagues components from this page. The fixtures and results are also displayed on this page to aid the user when making a team selection for the upcoming game week.

Data Flow Diagram

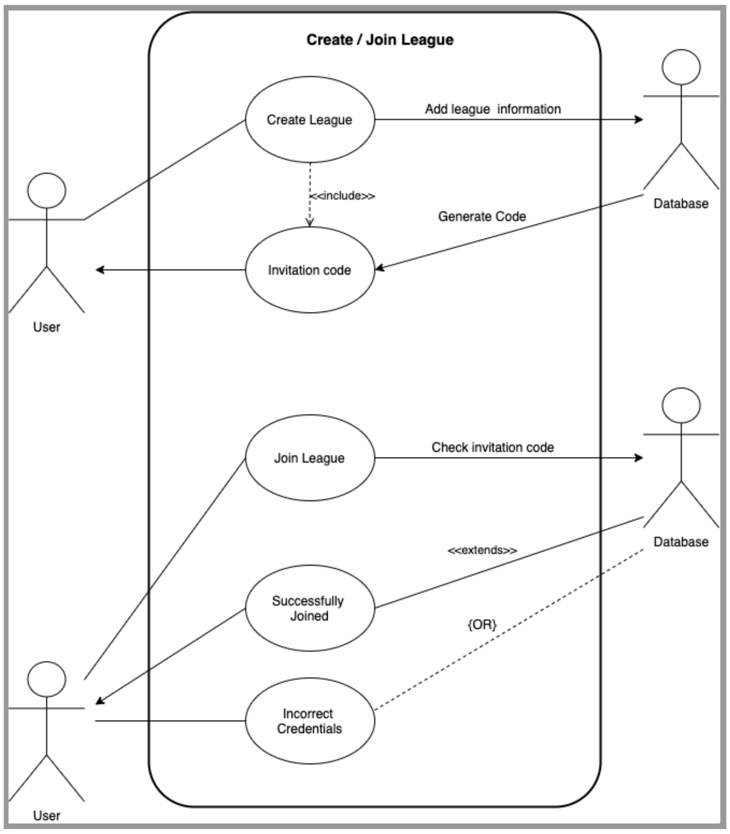


The above data flow diagram shows the flow for data throughout the My Leagues component. When the user enters the page several requests are sent from the component to the database to retrieve certain information. The information on the user’s active leagues must be retrieved along with the Premier League information for the fixtures and results.

4.2.5 Create/Join Leagues Component

The create and join components are subcomponents of the My Leagues components. They can be accessed through the My Leagues component. When selecting either of these components the user will be presented with a form to fill out. For both creating and joining leagues, they will be asked for the first and last name to be displayed on the league table. For the create league form they will be asked for a league name. The join leagues component asked for the invitation code for the league they want to join.

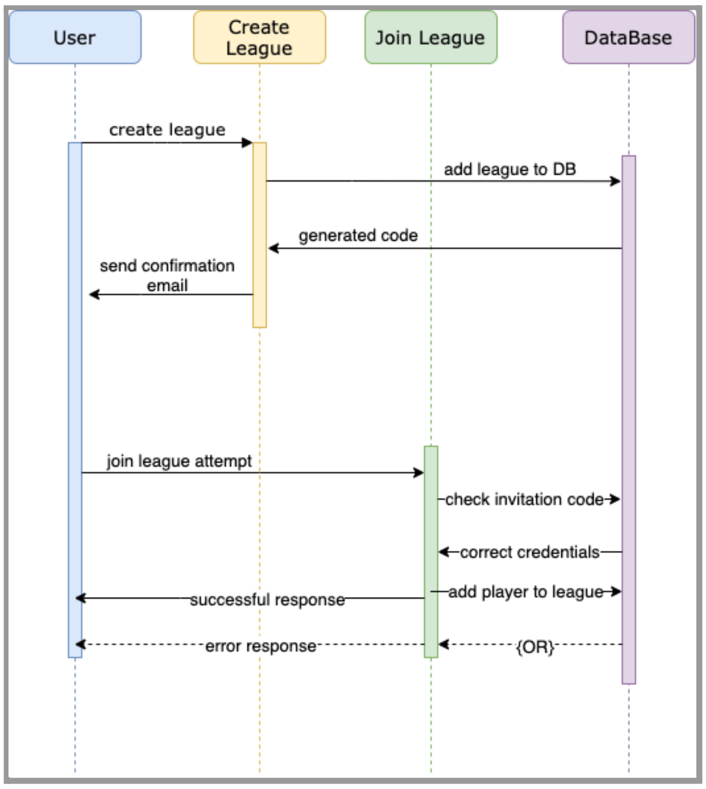
Use Case Diagram



The above diagram displays a use case for creating and joining a league. When creating a league the user fills out all credentials and the information is stored in the database. An invitation code is generated and sent to the user's email address.

When joining a league, a user inputs the required fields along with the league’s invitation code. This code is then checked in the database to see whether the league exists. If it does the user will be placed in the league. If it doesn't exist an error message is displayed to the user.

Sequence Diagram



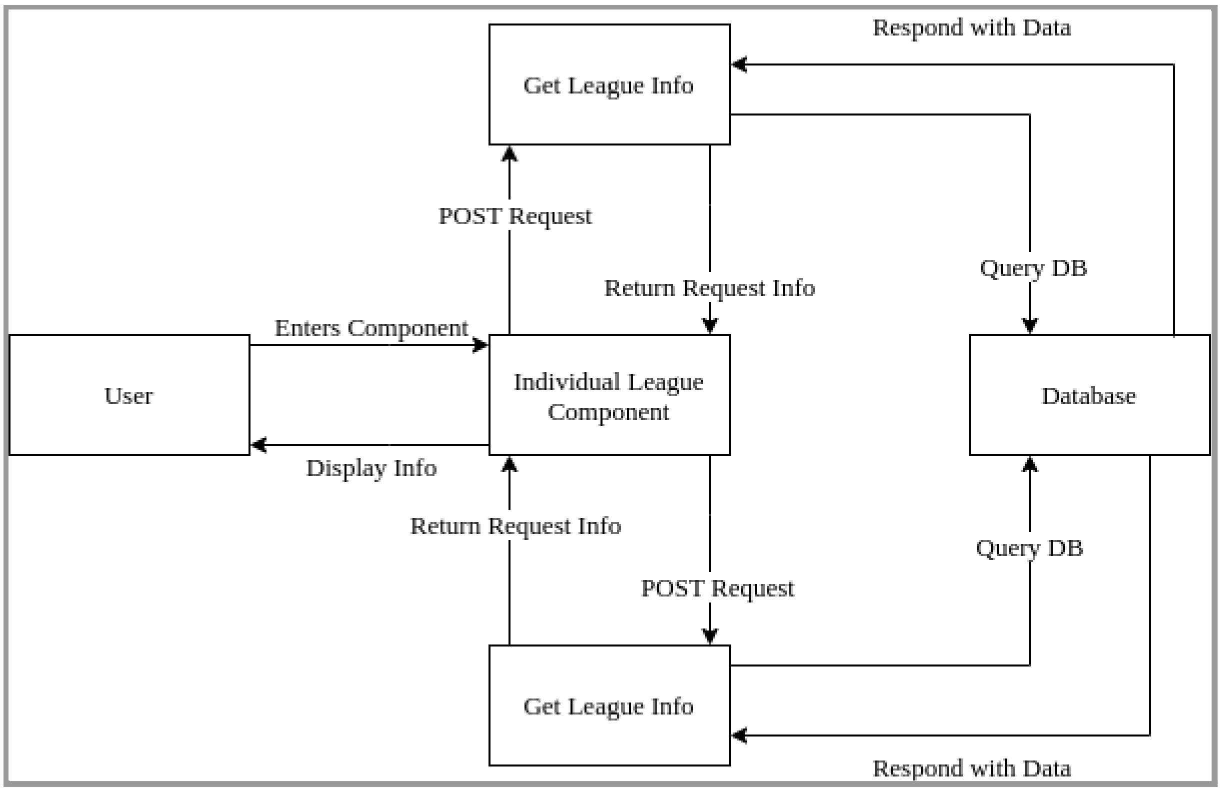
The above diagram shows the sequence of a user both creating a league and joining a league. When creating a league a user enters the credentials and they are added to the database. They are then sent a confirmation email for league creation with the invitation code that is generated.

When joining a league the user will input the invitation code given to them by the league's admin. This code is then checked in the database to see whether the league exists. If it does the users will be added to the league, if it does not exist an error message will be shown to the user.

4.2.6 Individual Leagues

To access an individual league a user must click the specific league in the My Leagues component. This will then display a table that consists of all of the users in the league, the admin of the league, the number of users remaining and eliminated, the invitation code. The user can view all other users in the league as well as their current picks. They can make their pick here also.

Data Flow Diagram



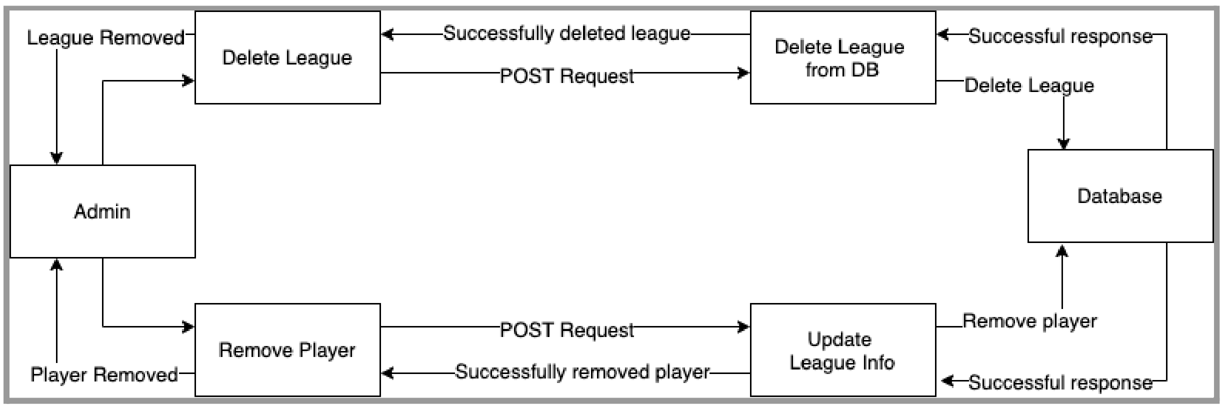
The above data flow diagram shows when a user enters the Individual League component how data then flows throughout the component. When a user enters the component several requests are sent to receive the relevant data about that user and that league.

4.2.7 Admin Actions

When a user creates a league they are the admin for this league. The admin actions components are only displayed to the admin. These components are sub-components of the individual league component. The admin can do the following:

- Remove player from the league
- Reset the league
- Make the league not joinable
- Delete the league

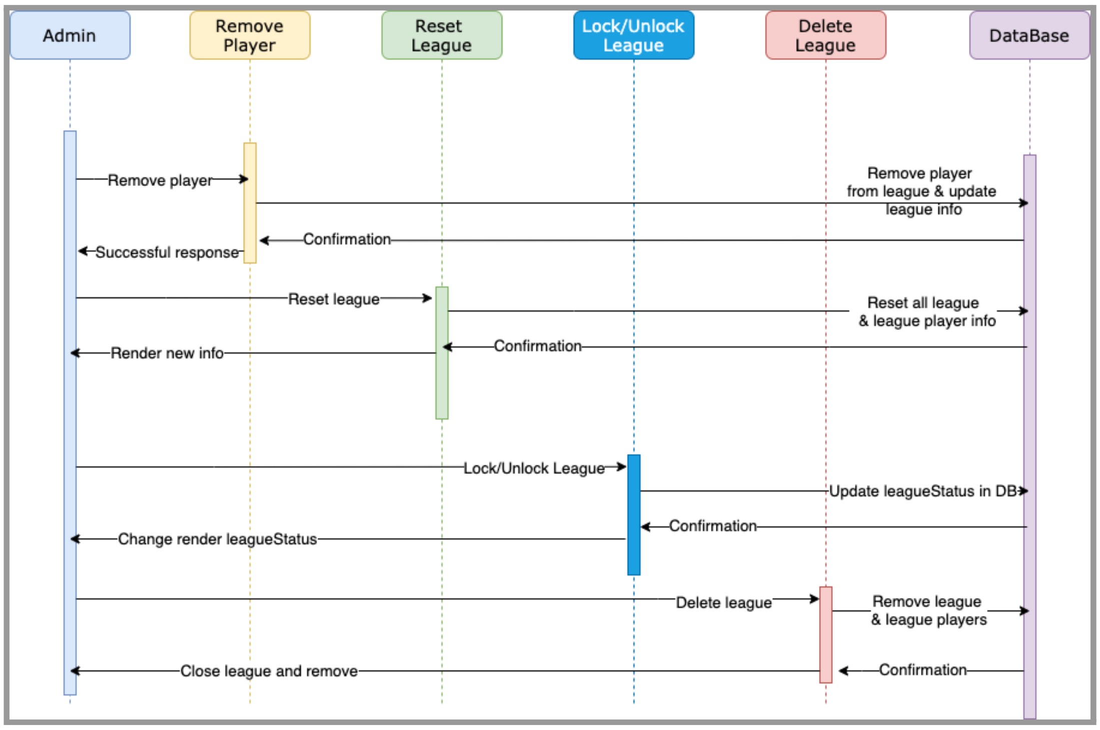
Data Flow Diagram



The above diagram shows both the delete league and removes player actions performed by the admin in terms of the data flow. As seen above when the admin deletes a league on this component this will send a request to remove the league from the database. This removes all information about this league and players in this league.

When an admin removes a player, it simply removes this player from the league, updating the league information and clearing all information attached to that player for that specific league.

Sequence Diagram

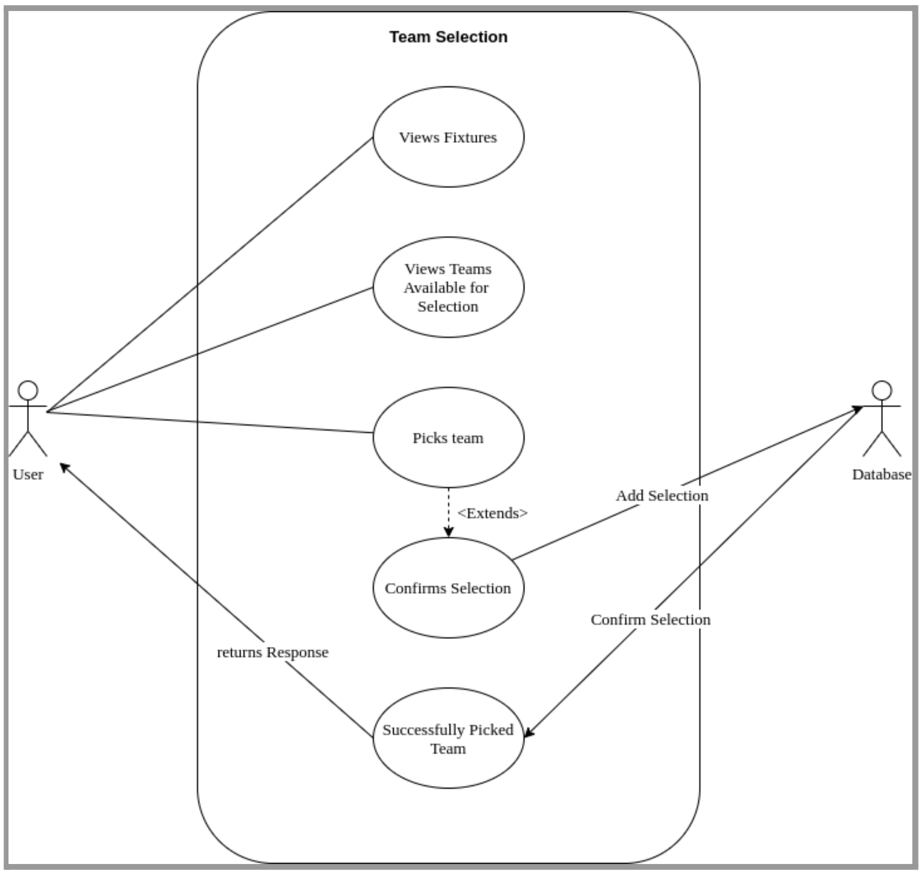


The above diagram shows the different actions the admin of a league can perform. It shows the sequence of each action from the request the admin makes to the removal of data from the database.

4.2.8 Team Selection

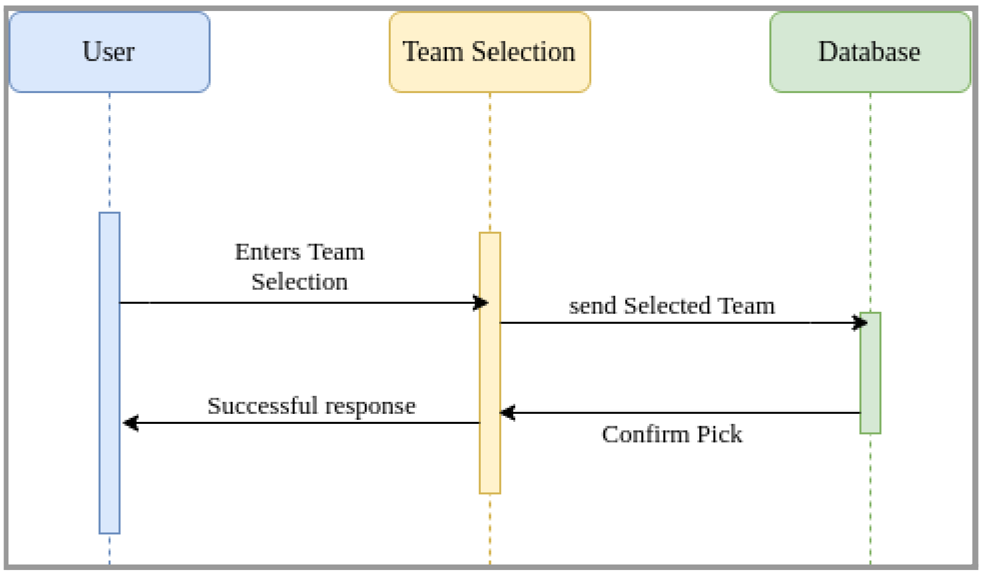
The team selection component allows the user to pick their desired team for the upcoming game week. It simply displays the fixtures for the upcoming week with all teams that are available for selection to appear as buttons. All teams that have previously been selected appear as normal text so therefore cannot be selected. The component also displays the probability of a given team winning its fixture. When a user selects a team a button will appear at the bottom of the card which can allow the user to confirm their selection for the upcoming week.

Use Case Diagram



This use case diagram shows how a user may interact with the component. They can view the fixtures and teams available for selection. Once they pick a team the component will then extend to display the confirmation button. Once clicked this will interact with a database and finally return a successful response to the user.

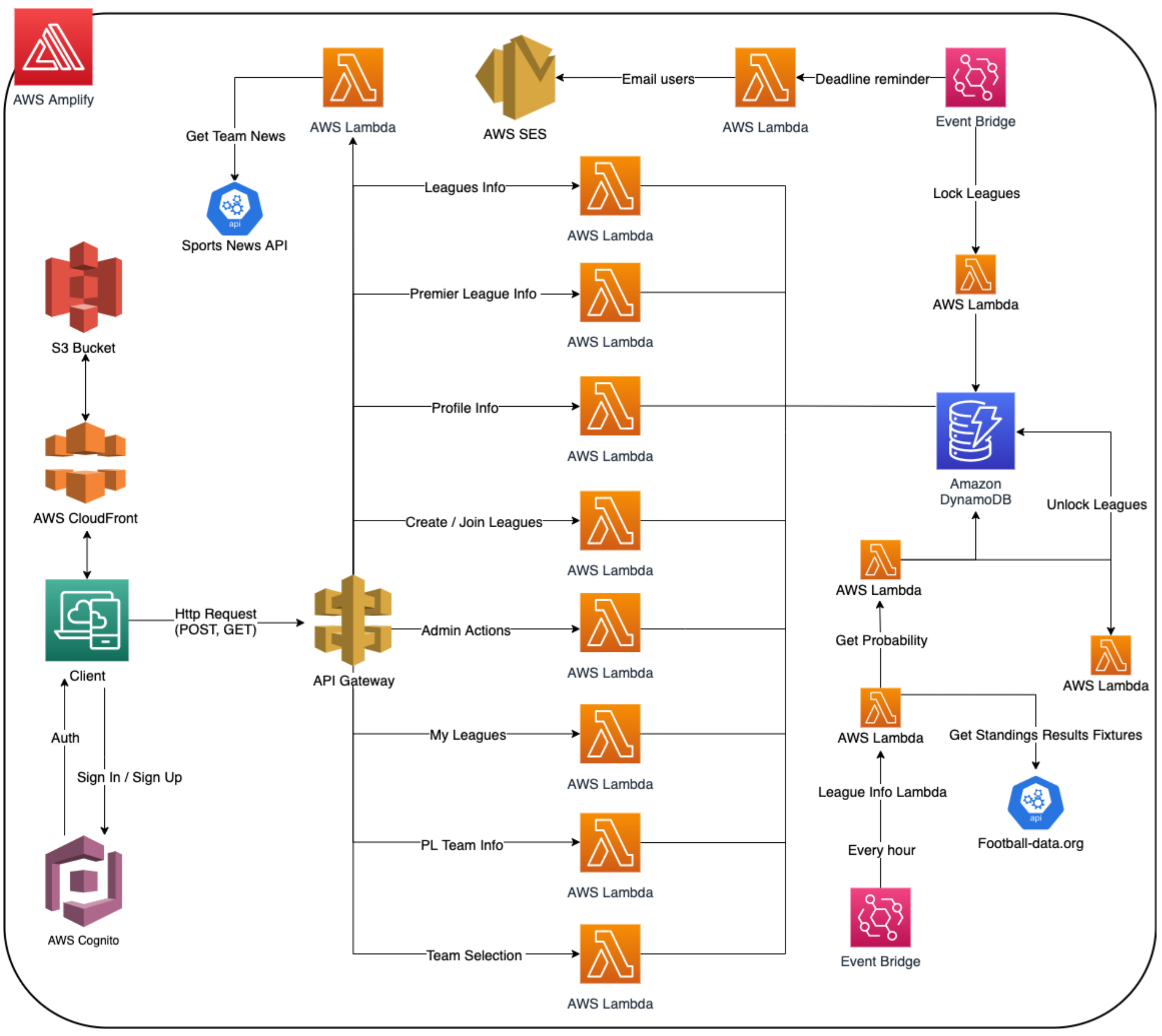
Sequence Diagram



A simple sequence diagram to display the sequence of actions that take place when a user interacts with the Team Selection component. A user selects a team from a list of unpicked teams. This then updates the database with the selected team. The database then returns a successful response.

5 Implementation

5.1 System Architecture



5.2 Architecture Decisions

The above diagram shows the overall system architecture for *Last Man Standing.* Originally, we imagined *Last Man Standing* to be a client-server application with the application running on an EC2 instance. However, after investigating other development styles we concluded that serverless was the best option.

Serverless architectures have several advantages over traditional cloud-based or server-centric infrastructure. They offer greater scalability, more flexibility, and quicker time to release, all at a reduced cost. Building the application serverless means we can focus on the development without having to manage the backend servers or worry about provisioning.

Due to the potentially large number of users the application may have down the line, scalability was a big factor that needed to be considered. Serverless applications allow for a lot of scalability within the application. Being on a student's budget is not easy so having a serverless architecture also means you only pay for what you use which is very convenient as it lowers the cost. We decided to go with AWS as a cloud provider as of our previous experience with the cloud services provider and the free credits for students.

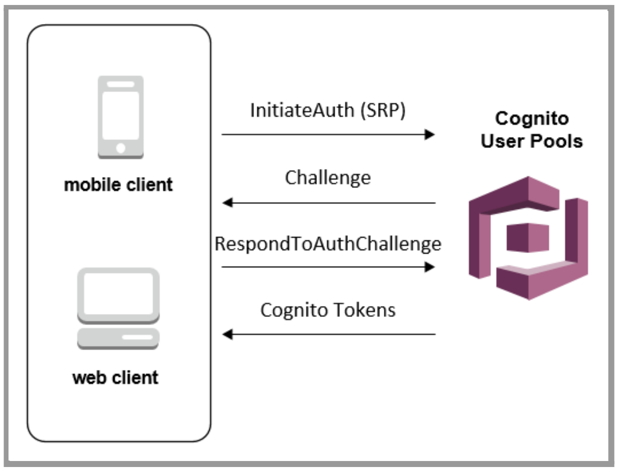
5.3 AWS Amplify

AWS Amplify is used to maintain and manage all services in *Last Man Standing*. It allows us to build a secure application making it easy to authenticate users. It helps to build scalable applications and secure data safely. Amplify was also used when deploying our application. Within our deployment system, Amplify runs an additional CI/CD Pipeline which is an additional safety net to ensure bugs don't get into production.

5.4 AWS Cognito

Cognito is used to assist with the verification and authentication of users. When a user signs up to Last Man Standing an initial SRP (Secure Remote Password protocol) is sent to our Cognito user pool which is then checked to see if the credentials are correct or if they already exist. The user's credentials are then added to the user pool.

The user then attempts to sign up using the credentials they signed up with. The user pool will challenge the SRP once again and the client sends back another response to this challenge. If the credentials are correct the user will then be authenticated and a Cognito token will be sent back to the client. We use this token to access the user's information throughout the application. The information is passed through in the props to customise the user experience.



https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-authentication-flow.html

5.5 AWS API Gateway

AWS API Gateway is a REST API that is the centre of operations on *Last Man Standing.* It handles all HTTP requests from the client side. There are several different paths on the API that all have different Lambda endpoints.

5.6 AWS Lambda Functions

Lambda is a serverless computing service. Lambda Functions run pieces of code (such as Python scripts) in stateless containers that are brought up on demand to respond to events (such as HTTP requests). The containers are then turned off when the function has completed execution.

5.6.1 Third-Party Premier League Information

This Lambda function is used to interact with a third-party Premier League API (footballdata.org). This function is invoked on a recurring schedule (every hour). As a result of us relying heavily on the third-party API for information to fill the databases we wanted to ensure that we were only updating our databases with this information when it had been updated correctly. Due to Covid-19, there were some matches cancelled or postponed. To combat this we set up several checks/conditions that must be met before the operations in the lambda function were performed.

For example, we did not want the fixtures of a certain game week to be changed in our database after every hour. The function only updates our databases when there has been a change or we have entered a new game week.

Another example of one of the conditions is to check if the last game of the previous game week result had been updated on the third-party API. This way we could assume that all other scores had been updated. Another condition was to check that the current time and date was at least 2 hours after the last game for that game week, this way we could ensure that the last game of the game week had finished. To do this we stored the date and time of the last game in our database and would check that time against the current date and time.

Once all checks pass, the lambda will parse through the response from the third-party API and populate our databases with the new standings, fixtures and results. The databases will be spoken about further in section 5.7.

Calculate Probability for Teams

Once the function has access to the upcoming game week's fixtures our probability function is then invoked. This function takes in several conditions to calculate the probability of each team winning a given fixture.

The main input to this function is live bettings odds received from the API. These odds take into account factors such as injured players, previous form and position in the league along with many others. Due to the nature of betting odds, bookmakers incorporate a value called an "over-round". An over round is created by bookmakers so that if a punter places a bet on every outcome of a match, the bookmaker will still earn money and the punter will lose money.

As the probability of a team winning in our application should not incorporate any over rounds created by bookmakers, a function had to be called to evaluate the over round for each fixture and remove it from the probability so that a true probability was found. To calculate over rounds a formula had to be created. Several fixtures were tested through several different formulae until we found the most correct result. The code/formula to eliminate the over round and convert the odds in probabilities can be seen below.

```
def getProbability(match):
   homeTeamOdds = match['odds']['homeWin']
   drawOdds = match['odds']['draw']
   awayTeamOdds = match['odds']['awayWin']

   overallOdds = 1/homeTeamOdds + 1/awayTeamOdds + 1/drawOdds
   overRound = overallOdds - 1

   homeTeamProb = 100 / (homeTeamOdds * 100)
   awayTeamProb = 100 / (awayTeamOdds * 100)
   drawProb = 100 / (drawOdds * 100)

   homeTeamProb = (homeTeamProb - (homeTeamProb * overRound)) * 100
   awayTeamProb = (awayTeamProb - (awayTeamProb * overRound)) * 100
   drawProb = (drawProb - (drawProb * overRound)) * 100

   return [homeTeamProb, drawProb, awayTeamProb]
```

Cron Jobs

Once we had access to information such as the date and time of the first game of each game week, we were able to set up cron jobs for both locking the leagues and sending out email reminders to pick a team to the competitors in each active league. The actions performed after these cron jobs were activated will be explained below.

5.6.2 Lock All Leagues

Based on the cron job that is set up in the Third Party Premier League Information lambda, Lock Leagues lambda is executed at a specific time each game week. Over the game week, *Last Man Standing* locks the leagues to prevent users from changing their teams in the middle of the game week. The Lock Leagues lambda is triggered one hour before the start of the first game of the game week.

The first action that this lambda undertakes is to retrieve all of the leagues. This is done by scanning the LeaguesDB. This scan returns all of the active leagues on *Last Man Standing* list which we then iterate through. After retrieving all of the leagues we then update these leagues by changing the “League Status” to closed which then updates a ternary operator on the front end to prevent users from being able to make a pick.

After the leagues are locked, the “leaguePlayerDB” is then scanned to retrieve the users picks for this week. When a user makes a pick, the teams available for picking are listed in unpicked teams. All picked teams are put into the picked teams list. When we retrieve the items from “leaguePlayerDB” we loop through each item updating the users' picks accordingly. We do this by removing their current pick from their unpicked list and add it to their picked teams and subsequently write this back to the “leaguePlayerDB”.

If a user does not pick a team before the leagues are locked, the lock leagues lambda will pick a team for them. We do this by scanning the Premier League Standing's database. We then pick the team highest in the league that is available for selection.

```
standingsData = sorted(standings, key = lambda i: int(i['position']))
```

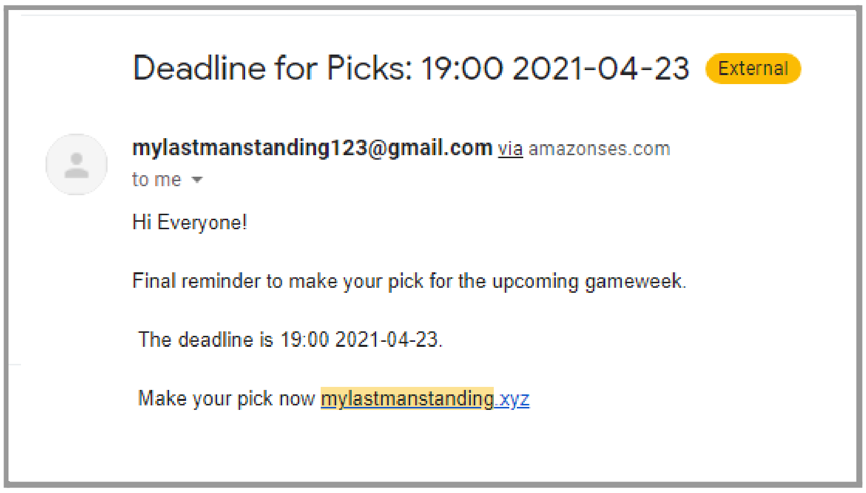
The above code snippet sorts the standings database based on position. We then assign the highest team available as the current pick as seen in the code below.

```
for team in standingsData:
   if team['TeamName'] in unPickedTeams and currentPick != 'Eliminated':
       currentPick = team['TeamName']
       unPickedTeams.remove(currentPick)
       pickedTeams.append(currentPick)
       break
```

5.6.3 Deadline Reminder

Before we lock the leagues we remind the user to make their picks. The deadline reminder lambda is invoked using a cron job which is created in Third Party Premier League Information lambda. The deadline reminder is executed one hour before the lock leagues lambda. This lambda will email users with a reminder to make their picks before the deadline.

The lambda firstly scans the "leaguePlayerDB" to retrieve the users for each league. We then loop through each user checking to see if they are knocked out of the league or not. If they are knocked out of the league the lambda will skip that user. If the user is still competing in the league we will then send them an email notifying them of the deadline to ensure they get their pick in before that.


Deadline for Picks: 19:00 2021-04-23 External

mylastmanstanding123@gmail.com via amazonses.com
to me

Hi Everyone!

Final reminder to make your pick for the upcoming gameweek.

The deadline is 19:00 2021-04-23.

Make your pick now mylastmanstanding.xyz

5.6.4 Unlock All Leagues

It was critical that the unlocking of the leagues was executed at the right time. If this function was not executed at the correct time the wrong users could be eliminated in a certain game week. To ensure that this was invoked at the correct time, the function is only invoked if all operations in the Third Party Premier League Lambda function work successfully.

The most important of these operations was the results from the previous game week being updated correctly. Once invoked this lambda function gets access to all active leagues from the “LeaguesDB”. From here we then get all results from the previous game week and create a list of all the winning teams. A query is then sent to the “LeaguePlayerDB” to get all players that have not been eliminated from their leagues. We then go through all players current picks and compare their picks to the list of match winners.

If the player’s picked team lost or drew their match the player’s status will be updated to eliminate. Once eliminated, the user will not be able to participate in that league until it is reset when there is a winner. A dictionary object is created with all leagueIDs and eliminated players from that week. If a player is eliminated the value for that league in the dictionary is incremented. This can be seen below.

```
for player in leaguePlayerResults:
  if player['CurrentPick'] != 'Eliminated':
    if player['CurrentPick'] not in winners:
      updateEliminatedPlayer(leaguePlayerDB, player)
      leagueDics[player['LeagueID']] += 1

    else:
      updateSuccessfulPlayer(leaguePlayerDB, player)
```

When updating each league's remaining and eliminated players, there is a check to see if there is a winner or not. The logic behind finding a winner is if the remaining player value is = 1 and the eliminated players is >= 1. If there is a winner the “leaguesDB” is updated with the username of the winner.

5.6.5 Get Premier League Information

When first loading onto Last Man Standing we send an Axios GET request through API Gateway to retrieve the Premier League Standings, Fixtures and Results from their respective databases. When the app first mounts this GET request is sent. This request then invokes the “getPremierLeagueInfo” lambda. This lambda scans the three respective databases to retrieve the information. As you can see below this information is then sorted with respect to the position of the team in the Premier League Table for standings or the kickoff time from fixtures.

```python
def scanTable(table, sorter):
	response = table.scan()
	res = response['Items']
	if sorter == 'position':
		data = sorted(res, key = lambda i: int(i[sorter]))
	else:
		data = sorted(res, key = lambda i: i[sorter])
	return data
```

5.6.6 Get Profile Information

When landing on the My Profile, an Axios POST request is sent inside the useEffect hook to API Gateway that triggers a lambda function to retrieve the user's profile information as seen below. This request sends the “Sub” of the user from the AWS Cognito token. This allows us to query the “PlayerDB for the user’s profile information.

```javascript
useEffect(() => {

axios.post('https://ida5es25ne.execute-api.eu-west-1.amazonaws.com/develop/profileInfo',
{sub: user['attributes']['sub'], flag: 'getTeam'})
         .then(response => {
           setmyInfo(response['data']);
},[user])
```

5.6.7 Get Team News

Once the profile information comes back from the database on the My Profile page, there is a check to see whether the user has picked their favourite team. If they have not picked their favourite team they will be prompted to do so. When they pick a favourite team this will send another Axios POST request to API Gateway to update their profile information with this favourite team. Once successfully updated, an Axios post request will be sent to API Gateway to trigger a lambda function to send a request to a third-party API to get up to date news on the user’s favourite team. These POST requests can be seen below.

```
const selectTeam = (team) => {

axios.post('https://ida5es25ne.execute-api.eu-west-1.amazonaws.com/develop/profileInfo',
{sub: user['attributes']['sub'], team: teams[team], flag: 'setTeam'})
       .then(response => {
         setmyInfo(response['data']);
         alertify.set('notifier','position', 'top-center')
         alertify.success('Successfully Updated Favourite Team')
       });


axios.post('https://ida5es25ne.execute-api.eu-west-1.amazonaws.com/develop/getNews',
{team: teams[team]})
       .then(response => {
         setTeamInfo(response['data'])
       });
   }
```

5.6.8 Create / Join League

The create and join leagues lambdas are invoked when a user clicks the “Create” or “Join league” buttons on the frontend. When clicked it calls a “handleSubmit” function which takes the information filled out in the forms and sends an Axios POST request to the respective API endpoint.

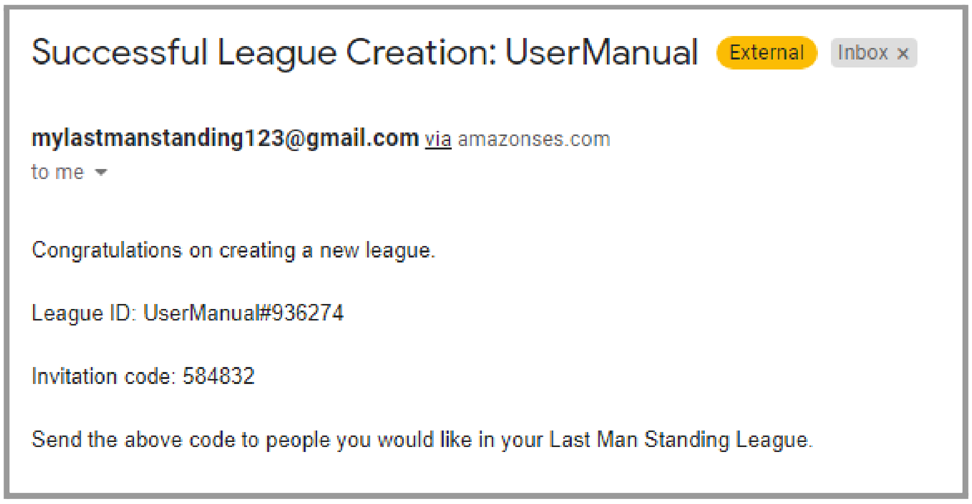
When a user creates a league the first operation in the createLeague lambda is to check if the current game week has started. If this is the case the user may not create a league as it will be open during the game week which will cause issues for that league. If it is not during the game week is to create a unique id for the league and a unique invitation code. At the beginning of the development process, we felt it would be important for users to be able to pick a league name and for there to be no issue if there are two leagues with the same name. To get around this problem we create the unique id which incorporates the league name and a unique 6 digit code appended to the end of the league name. Similarly, we wanted users to be able to join a league easily. To do this we create a unique invitation code that corresponds solely to that league which ensures users join the correct league.

After this, the league is created. This process involves writing to the “leaguesDB” with the following parameters:

```
tableName.put_item(
      Item={
            'LeagueID': leagueID,
            'LeagueName': result['leagueName'],
            'admin': result['sub'],
            'fullName': result['firstName']+ ' ' +result['lastName'],
            'createdTime': str(datetime.today()),
            'invitationCode': invitationCode,
            'LeagueStatus': 'Open',
            'RemainingPlayers': '1',
            'EliminatedPlayers': '0',
            'Joinable': 'Yes',
            'Winner': '-'
})
```

Once the league is created an email will be sent to the admin of the league. This email notifies the admin that the league has been successfully created and the invitation code for that league. Following this, the lambda adds the admin into the league by adding them to the “leaguePlayerDB” and adding the league id to their account in “playerDB”.

Below is an example of an email sent when a league was created.


Successful League Creation: UserManual

mylastmanstanding123@gmail.com via amazonses.com
to me

Congratulations on creating a new league.

League ID: UserManual#936274

Invitation code: 584832

Send the above code to people you would like in your Last Man Standing League.

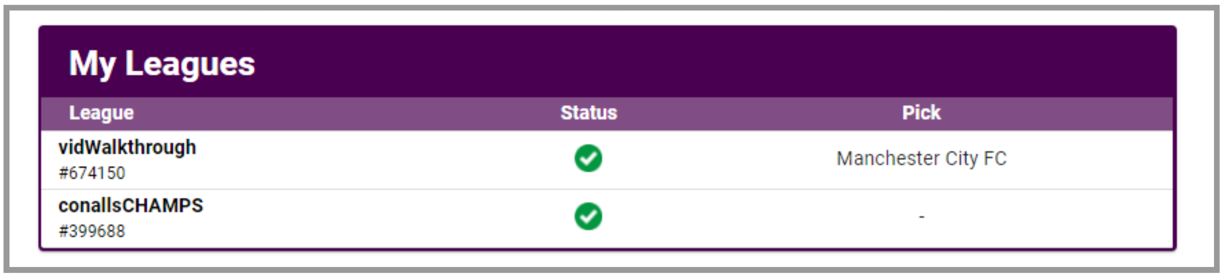
If a user wishes to join a league they will need to enter the invitation code for that league into the input field on the front end. Once entered and the “Join League” button is clicked the “joinLeague” lambda will be invoked. This lambda firstly queries the playerDB with the user’s sub to check if the user has already joined the league. If the user has not already joined the league and the league is set to joinable then the user will be added to the league.

This operation involves adding the user in leaguePlayerDB for that league, incrementing the number of remaining players for that league and adding the league name to the user in playerDB.

```
def checkUser(leagueID, result, playerDB, resp):
  dynamodb = boto3.resource('dynamodb', 'eu-west-1')
  # lookup users leagues
  data = playerDB.query(
    KeyConditionExpression=Key('Sub').eq(result['sub'])
  )
  res = data['Items']
  leagueIDs = res[0]['leagueIDs']
  # check if user already in league and it is joinable
  if leagueID not in leagueIDs and resp[0]['Joinable'] == 'Yes':
    # update players leagueIDs
    updatePlayerDB(playerDB, leagueIDs, result, leagueID)
    # add player to leaguePlayerDB
    leaguePlayerDB = dynamodb.Table('LeaguePlayerDB-develop')
    updateLeaguePlayerDB(leaguePlayerDB, result, leagueID)
    # update remaining players
    leaguesDB = dynamodb.Table('LeaguesDB-develop')
    updatedRemainingPlayers(leaguesDB, leagueID)

    return 'Successfully joined league'

  elif resp[0]['Joinable'] == 'No':
    return 'League is not joinable'

  return 'Player already in the league'
```

5.6.9 Get My Leagues

When a user clicks on the “My Leagues" tab on the navbar it will direct them to My Leagues component. When this component is called an Axios GET request is sent from a useEffect hook to get the users leagues. Using the users unique sub we first query the “playerDB" for that sub to get the user’s active leagues. Following this we then acquire the individual league’s information using the league is obtained from the player query. The results from the “leaguesDB" query is then returned to the front end to be parsed into a table for the user.



My Leagues

League	Status	Pick
vidWalkthrough #674150	✓	Manchester City FC
conallsCHAMPS #399688	✓	-

5.6.10 Admin Actions

As an admin of a league, there are several operations that you can do that other users can’t. An admin may reset the league, lock the league from users joining, remove players from the league and delete the league. All of these actions take place in the admin actions lambda. Admin actions are set up in a way where when an admin clicks a certain operation and a flag gets passed into the lambda to determine the operation in the lambda.

```
if flag == 'deleteLeague':
    resp = deleteLeague(result)
elif flag == 'toggleLeague':
    resp = toggleLeague(result)
elif flag == 'removePlayer':
    resp = removePlayer(result)
elif flag == 'resetLeague':
    resp = resetLeague(result)
```

If the admin deletes the league the deleteLeague function will be called. This will firstly delete the league from “LeaguesDB” using the “leagueID”. After this, it will then query the leaguePlayerDB with the leagueID to obtain all of the users in that league. We then loop through the results of the query adding each user's unique sub into a list for future proceedings. After this, all items which contain the “leagueID” in “leaguePlayerDB” are removed from the database. Finally, for each sub in the list of subs, we then remove the league name from the user’s leagues in playerDB. The league is then officially deleted.

If the admin decides the lock or unlocks the league the toggle flag will be passed to admin actions. This will then update the leaguesDB changing the status of “Joinable” to either “yes” or “no”.

If a player is removed from the league a similar operation to the delete league function will take place. We first query the leaguePlayerDB with the user’s sub and remove that user from the leaguePlayerDB. After this, the league is then removed from the user’s active leagues in playerDB using the user’s sub as the primary key for the query. Finally, the number of players in the league is now incorrect and will need to be altered. This is done by altering the remaining players of the league in “LeaguesDB”.

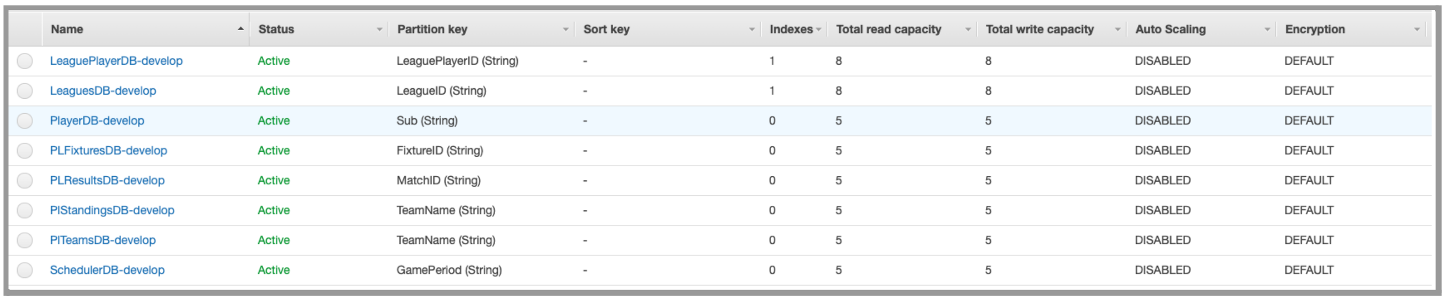
The final action is to reset the league. This queries the leaguePlayerDB to reset all users in the league. We loop through each user resetting their unpicked teams and pick teams to the original values. After which we update the eliminated and remaining values for the given league.

5.6.11 Team Selection

When a user picks a team for the given game week this will send an Axios POST request to the pickTeam lambda through API Gateway. LeaguePlayerDB will then be queried using the user’s sub and the league id to get the correct player and league. This query will then be updated with the user's pick by updating the “currentPick” item in the database for that user. A simple response is sent back to the front end to tell the user that they have made their pick.

5.7 AWS DynamoDB

Our DynamoDB is the centre of all storage of data in *Last Man Standing.* DynamoDB is a key-value and document database. The reason for choosing DynamoDB is that it delivers single-digit millisecond performance at any scale. We have multiple tables in the database. The tables and their primary keys can be seen below.



	Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity	Auto Scaling	Encryption
	LeaguePlayerDB-develop	Active	LeaguePlayerID (String)	-	1	8	8	DISABLED	DEFAULT
	LeaguesDB-develop	Active	LeagueID (String)	-	1	8	8	DISABLED	DEFAULT
	PlayerDB-develop	Active	Sub (String)	-	0	5	5	DISABLED	DEFAULT
	PLFixturesDB-develop	Active	FixtureID (String)	-	0	5	5	DISABLED	DEFAULT
	PLResultsDB-develop	Active	MatchID (String)	-	0	5	5	DISABLED	DEFAULT
	PlStandingsDB-develop	Active	TeamName (String)	-	0	5	5	DISABLED	DEFAULT
	PlTeamsDB-develop	Active	TeamName (String)	-	0	5	5	DISABLED	DEFAULT
	SchedulerDB-develop	Active	GamePeriod (String)	-	0	5	5	DISABLED	DEFAULT

LeaguesDB

The LeaguesDB is a table that contains general information for each league. The primary key to the table is the “LeagueID”. It also contains information such as the invitation code, remaining players, eliminated players, winner of the league and the username of the admin.

PlayerDB

The PlayerDB is a table that contains each user’s profile information. The primary key to the table is the player’s “sub” generated from the Cognito token. It also contains information such as the email, favourite team, list of active leagues, wins, losses and their username.

LeaguePlayerDB

The LeaguePlayerDB is a table that contains each user’s information in a particular league. The primary key to the table is the player’s “LeaguePlayerID” which is a combination of the leagueID and the user’s sub. It also contains the following information: username, leagueplayerID, picked teams, unpicked teams and whether the user is eliminated or not.

StandingsDB, FixturesDB, ResultsDB

These three tables contain the updated information on the Premier League. The standings of all teams, the upcoming game week’s fixtures and the previous game week’s results.

SchedulerDB

The SchedulerDB contains all the important information to do with the current game week. It contains information on the first game time and date. The game week number, the previous game week and the next game week.

PLTeamsDB

The PLTeamsDB contains all information on the 20 teams in the Premier League. The team name, abbreviations and crest images.

6 Problems and Solutions

6.1 Premier League API Hates Covid-19

These are unprecedented times and no one/nothing can avoid COVID-19 in some form or another. We found this out quite early on in our development of this application as the Premier League was constantly battling COVID-19. In a normal season, the API would be reliable and update with the latest information. However, as COVID-19 tightened its grip around the world we found irregularities in the Premier League which translated into the API.

As the Premier League has a very large team of operations to make it run smoothly, it was only inevitable that one of these large teams would stumble against COVID-19. There were numerous occasions where matches were postponed only hours before kick-off due to COVID outbreaks resulting in the API and our application then being incorrect as the API could update in time.

We also found that the regular rescheduling of games was an issue this season. We found games part of GameWeek 33 being played during GameWeek 26 which caused issues for our results and fixtures as we moved through the season.

To combat these issues we have implemented several conditions in our LeagueInfoLambda. An example of one of these conditions is to create a scheduler database that contains the current game week, the previous game week and upcoming game week. This way we can ensure that we are adding the correct fixtures into the table and retrieving the correct results for the previous game week.

6.2 DCU Gitlab Hates Amplify Deployment

One of the biggest issues that we came across in the development was when we tried to launch the web application online. AWS Amplify has a deployment system that can be used to deploy all of your back end services and to launch the front end.

AWS Amplify makes it easy to deploy applications online straight from a repository... we thought. Due to the DCU Gitlab environment not being classified as the same type of repo as a normal GitLab repository, this meant a fix needed to be found. AWS Amplify could not pick up any repositories at GitLab.computing.dcu.ie. We tried to find several workarounds, to get our college repo deployed but this was a harder task than we thought.

After hours of research and attempting different work rounds, we found that the best way to do this was to link a personal Github repository to our college repo. We used a git hook to link any push made to master on the Gitlab repository to the personal Github. Master was essentially our production branch that would be deployed live whenever we pushed to it.

Once we linked the two repositories together it was straight forward to connect Amplify to the personal Github account. In the end, doing this worked in our favour as Amplify's pipeline is now acting as an additional safety net on top of our CI/CD Pipeline in the Gitlab repo. This stops bugs from getting into production.

6.3 Live Share Duplicating Code / Files

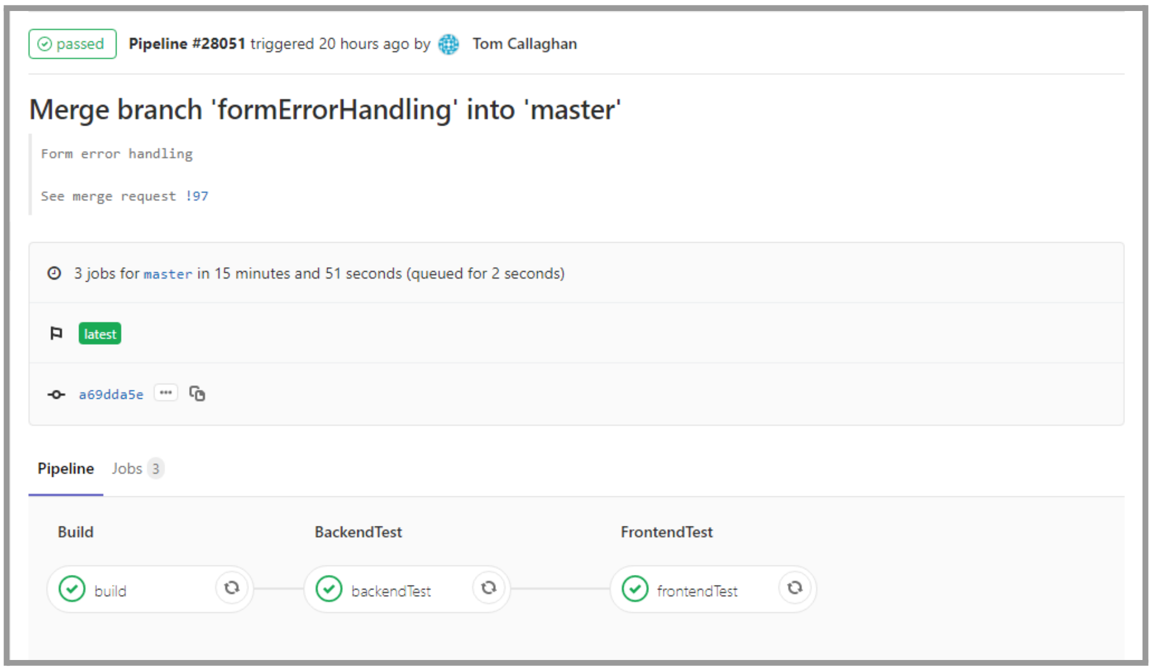
Due to the current global pandemic, it was inevitable that we would have to work remotely. As we are so accustomed to pair programming and we know the benefits of pair programming we knew this would be difficult. The solution to this was VS Code's Live Share extension, however, this did not come without its complications.

While using this extension, we found that the extension had been causing duplicate code to be created along with duplicate files to be created. This was creating many errors within the development. To combat this, source control was actively checked regularly as well as tickets being raised with VS Code. We found that the issue did not continue after raising this ticket, but we continued to actively check our source control to ensure it was not going to happen again.

7 Git Practices / Project Management

7.1 CI/CD usage & Git practices

One of the first tasks we believed that needed to be done was to set up a working CI-CD to ensure all code pushed will be tested to ensure no bugs were introduced into the code. The CI-CD is set up to run all frontend and backend tests as well as building the application. When code is pushed to our repo whether on master or a branch the CI-CD will be triggered. If the CI-CD fails we know that there is an issue with the code which can be rectified. Gitlab also prevents branches from being merged to master unless the CI-CD has passed. This also helped with regression testing. The pipeline running all tests on components after a change was made meant that we could see if the changes to one component affected another component. Below is an example of the CI-CD pipeline.



passed **Pipeline #28051** triggered 20 hours ago by Tom Callaghan

Merge branch 'formErrorHandling' into 'master'

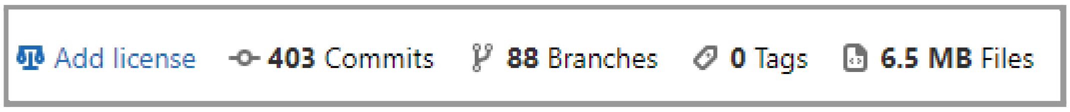
```
Form error handling

See merge request !97
```

3 jobs for `master` in 15 minutes and 51 seconds (queued for 2 seconds)

latest

`a69dda5e`

Before starting any development we discussed the importance of using branches. This enables us to review each other's code before being merged to master. This added piece of security reduces the number of bugs entering production. For any change or new feature being incorporated we created a new branch labelling the branch what the feature or fix is about. An example of this would be
“signUpUnitTesting”. This naming convention allows us to know what component it relates to in this case being “signUp” and what the change is which in this case is “Unit Testing”. All branches must be approved by the other person before merging into master. Below you can see the number of branches used in the development of *Last Man Standing*.



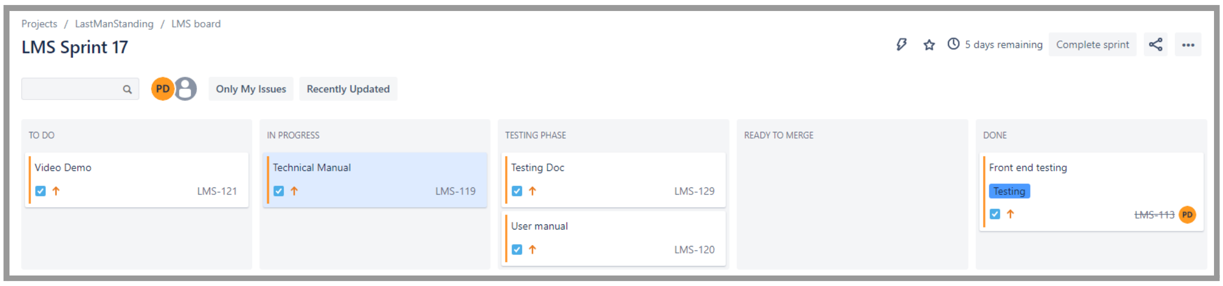
Add license 403 Commits 88 Branches 0 Tags 6.5 MB Files

We also tried to ensure our commits were small and often. This allowed us to get a clear understanding of what changes have been made when reviewing each other's code. It also meant that if there was an issue we can identify quickly what commit it was pushed on. As you can see above we have over 400 commits which confirms this practice.

7.2 Project Management

JIRA

Due to the nature of the application development being done remotely (Covid-19 😔) and at the same time as trying to juggle other college assignments, we felt it was ideal to run the development in an agile environment. After realising there was a substantial interest from potential users for this application we decided that it would be best to perform a longitudinal study, where we could take user feedback and fulfil their requests at a fast pace. This was another pointer towards agile development as we could run 2 weeks sprints that we could take in user feedback and have them in production by the end of the following sprint.
To incorporate an agile environment into the development we decided to make use of the framework JIRA. Jira is a proprietary issue tracking product developed by Atlassian that allows bug tracking and agile project management. A total of 17 sprints (34 weeks) were conducted over two week periods starting at the very start of the project development. All tasks were categorised in the backlog and added from the backlog into the sprints accordingly. There was a sprint meeting at the end of each sprint where we spoke about what we got done, if there were any issues or bottlenecks and also what tasks we wanted to add to the upcoming sprint. This helped us to stick to deadlines and put some pressure on us to ensure all tasks were being taken care of in a timely fashion. The below image shows the active sprint at the time of writing this document.



Projects / LastManStanding / LMS board

LMS Sprint 17

5 days remaining Complete sprint

Only My Issues Recently Updated

TO DO
- Video Demo LMS-121

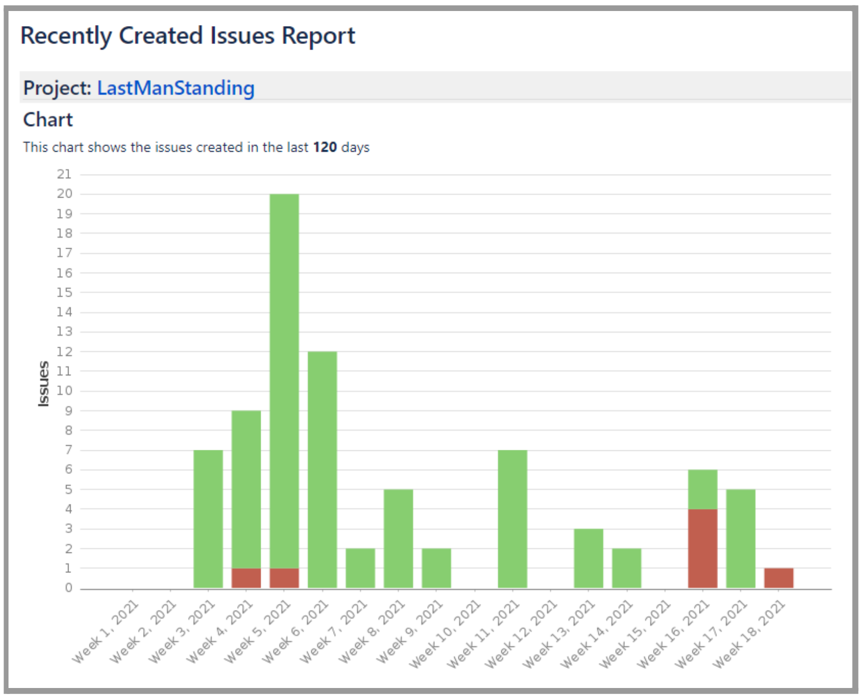
IN PROGRESS
- Technical Manual LMS-119

TESTING PHASE
- Testing Doc LMS-129
- User manual LMS-120

READY TO MERGE

DONE
- Front end testing Testing ~~LMS-113~~

By using JIRA we were able to check analytics to see whether we were keeping on top of the tasks in the active sprint. The below graphic shows the tasks that were completed (green) and not completed (red) in a given sprint. As you can see we kept on top of the majority of tasks in each sprint, this helped us to stay on track. The weeks that there are a few incomplete tasks can be traced back to weeks with many other college commitments/assignments or else bottlenecks to do with the task.



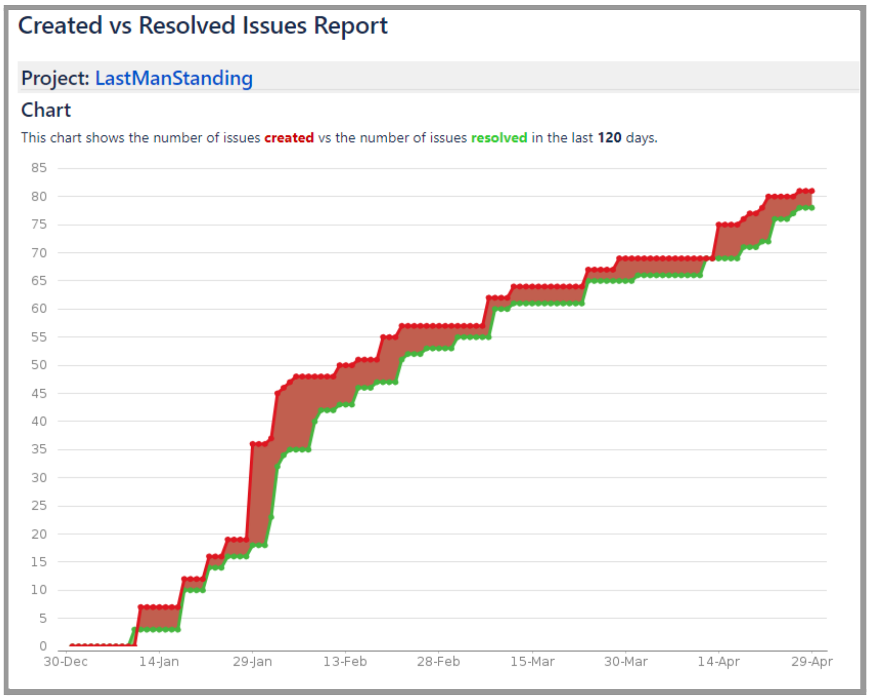
Recently Created Issues Report

Project: LastManStanding

Chart

This chart shows the issues created in the last **120** days

The graphic below shows the total number of tasks created and the subsequent total number of tasks resolved over 120 days. It's important to see that there was a constant flow of tasks being added to the backlog as they arose and the fact that the resolved tasks are only slightly below suggesting that our velocity matches the backlog input.



To help with the longitudinal study for the web application, we also made use of the issue tracker on JIRA. This helped us to set severity levels on any issues that users would point out to us on the site. By having this tracker we were able to ensure that the most critical issues were attended to first.

Notion

Due to the large scale of the development of *Last Man Standing,* we felt it was essential to document some of the journeys. With many difficult tasks having to be resolved throughout the development process. By simply documenting some of the issues, it is easy to reference this documentation if the same issue arose more than once. To document this information we used an application called Notion. Notion is an application that provides components such as notes, databases, kanban boards, wikis, calendars and reminders.

Automatic Deployment

As *Last Man Standing* is a live application with over 100+ users, we plan to continue the development after the submission of CA400. As a result, we decided to set up a personal GitHub repo to have full control of the deployment after we left college. Due to the requests of potential we launched the application before the submission of this course. To allow us to keep all commits and changes record in this external GitHub repo we decided to set up an automatic deployment. This meant linking our DCU Gitlab repo to the personal GitHub repo. This is done through a git hook.

It is set up that any changes pushed to master on the Gitlab repository would automatically push our own GitHub repository. From here AWS would run deployment/build tests to deploy the application. This is an additional safety net to ensure no bugs reach production. Once this pipeline finishes the changes made to the Gitlab repository will be live on the site in several minutes.

Due to the longitudinal study that is being performed on the application with frequent user feedback, many changes have had to be made to satisfy ongoing user requirements. We thought it would be vital to have this process be automatic to eliminate any human error when deploying to production resulting in better consistency.

8 Testing

With 100+ users using the web application since March, extensive levels of testing were needed. Due to the large volume of testing that was done, a separate testing document was created. This document can be accessed in the document section in the repository of the project.

9 Future Work

With over 100+ users actively using *Last Man Standing* within the first 2 months of launching without any social media presence or marketing, we have decided that we will continue to develop and maintain the application further. Below are some of the plans for future work.

9.1 Mobile App Development

Last Man Standing is a web application that is compatible with mobile and tablet devices. However, from the analytics we have found from user interaction in the first 2 months of the applications life, we found that a large number of our users are accessing the site from mobile devices. Although the application works well on mobile devices as a web application, there is something that appeals to us about making it a mobile app with an accompanying web application. With some previous experience in react native development, it would be a shame not to proceed and build it into a web application.

9.2 Marketing

As stated above with over 100 users in two months without any social media presence or marketing, we feel that there is a gap in the market for a service like this. To grow the product we plan on running some market campaigns on social media platforms to spread the word about this application. We had refrained from doing this before the submission of this project in case any issues were to arise.

9.3 Potential Payment

Last Man Standing is currently a free web application and you can make as many leagues as you would like for free! As interest in the application continues, we are potentially going to start charging for the service.

9.4 Variety of Competitions

Last Man Standing is an application that could be used for many different competitions. Although throughout our development the application is designed for the Premier League, in the future we hope to extend into different leagues and sports. *Last Man Standing* is an app that could easily increase the scope to incorporate the NFL or NBA. It is also an application which could special events such as the Olympics. This would increase the number of users on the site as we cater to users sport preferences.

10 Conclusion

Thank you for taking the time to read the technical guide for *Last Man Standing*. We hope that it was informative and you enjoy using our web application. Best of luck in your competitions at mylastmanstanding.xyz! Feel free to leave us any feedback with the form below.

Feedback form: https://forms.gle/MxFA3Ce9zpepYmuc8