# Six Nations Tracker

Brian Doyle – G00330969

Niall Devery – G00338932

Darragh King – G00326176

## Introduction

## System Requirements

Windows 10 is required to run the project along with Visual Studio 2017 and the Neo4j.  
The system requirements for Windows 10 are as follows:

* Processor: 1 gigahertz (GHz) or faster processor or SoC.
* RAM: 1 gigabyte (GB) for 32-bit or 2 GB for 64-bit.
* Hard disk space: 16 GB for 32-bit OS 20 GB for 64-bit OS.
* Graphics card: DirectX 9 or later with WDDM 1.0 driver.
* Display: 800x600.

The system requirements to install Visual studio 2017 are as follows:

* 1.8 GHz or faster processor. Dual-core or better recommended
* 2 GB of RAM; 4 GB of RAM recommended (2.5 GB minimum if running on a virtual machine)
* Hard disk space: up to 130 GB of available space, depending on features installed; typical installations require 20-50 GB of free space.
* Hard disk speed: to improve performance, install Windows and Visual Studio on a solid-state drive (SSD).
* Video card that supports a minimum display resolution of 720p (1280 by 720); Visual Studio will work best at a resolution of WXGA (1366 by 768) or higher.

The system requirements to install Neo4j are as follows:

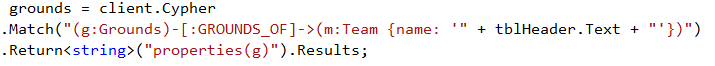
* Minimum Intel Core i3 processor, recommended Intel Core i7 processor
* Minimum memory to install is 3GB and the recommend is 16-32GB

## Technology Used and Why

For this project we have used Neo4j, UWP and Visual studio 2017 along with Neo4j client

**Neo4j**

* We decided to use Neo4j after discussing what database to use at first, we decided to use a mySQL database but after conferring with a lecturer, he suggested that we use a graphical database like Neo4j.
* After conducting research into Neo4j we arrived at the conclusion that it would be more suited to the architecture than mySQL as well as the graphical layout of the database made it easier to understand the relationships between the entities within the database.
* Neo4j uses its own query language called Cypher. Cyphers syntax is like SQL but uses nodes and relationships to organise data.
* Cypher contains a variety of clauses, the most common being MATCH and WHERE. These two functions are slightly different to SQL because of Neo4j using nodes and relationships. MATCH is used for describing the structure of the pattern searched for based on the relationships in the database.
* For example, the following query taken from TeamPage.xaml.cs will return the grounds of the given team.

****

Additionally, the CREATE and DELETE clauses are used to create and delete nodes and relationships. The following query shows nodes being created and being given properties.



**UWP**

* UWP was created by Microsoft for windows 10 and is used as a software platform to develop universal windows apps that will run on windows 10, windows 10 mobile, Xbox One and HoloLens without needing to be rewritten for each. The API is implemented in C++ and supported in C++.

**Visual Studio 2017**

* Visual studio 2017 provides a more comprehensive version of IntelliSense when compared to that of the 2015 version, it is more capable if differentiating between the most likely property to be used and the property at the top of the list, this is due to the capitalisation.

**Neo4j client**

* The Neo4j client allows the user to create local and remote databases through the desktop application the user has the option to create and manage a database under a project a project may consist of multiple graphical databases. Once a database has been created the user can manage it and track logs, plugins and upgrades to it.
* The user has the option to open a browser window in which they can manipulate the database using cypher queries, but the database must be running first.
* Once the browser window has opened the user has a terminal bar along the top of the browser window where they can enter queries to manipulate the database.
* By entering “RETURN (n) MATCH (n)” the user can see all nodes and relationships within the database. This is one of the reasons we chose to use this database as it makes the information more clear and easier to understand as opposed to the SQL database architecture.
* There is also a sidebar that displays information organised by nodes and relationships. This can be used to filter down the information to find the appropriate data. For example by filtering the information by “MATCH p=()-[r:GROUNDS\_OF]->() RETURN p” within our database the client will return all of the teams and their relationship to the grounds graphically within the client.
* Another reason we used the client is because of the inbuilt support for new users. On the sidebar there is a tab labelled “Documents” within this there are helpful hyperlinks that will input a command to the terminal bar that will display Cypher commands the user can use.

## Architecture of the Solution

The project starts up with the main page from there the user can navigate to the team pages by pressing the logos. When one of the logos is click a switch statement in the Teampage.xaml.cs page is used to handle the event from there depending on what was click the page is generated dynamically using the switch statement cypher requests are made to pull the team data from the database and that is then inserted into the tables within the page.

The database uses nodes and relationships to illustrate the data. As seen in figure 1 the whole database is connected through relationships and nodes. In figure 2 we can see, and clearer picture of a Many-to-One relationship used within the database.

* The database consists of Teams, Players, Coaches, grounds, 2018 6 nations results and the 2019 6 nations fixtures nodes.
* The teams have a One-to-Many relationship with the players under the name PLAYS\_FOR.
* The grounds have a One-to-One relationship with the teams under the name GROUNDS\_OF.
* The 2018 results have a Many-to-One relationship with the teams under the name PLAY\_AGAINST.
* The grounds have One-to-Many relationship with the 2018 results under the name PLAYED\_IN.
* The grounds also have a One-to-Many relationship with the 2019 fixtures also under the name PLAYED\_IN.

Each node contains properties for the respective data

* Each team node contains name, captain, total points, penalties conceded, penalties won, tries scored, lineouts won, lineouts lost, most passes, metres gained, total red cards, total yellow cards and most tackles.
* Each person node contains player name, caps, points and position also the coach details containing the coaches name and the amount of years they’ve coached the 6 nations team.
* Each ground node contains the name, date it was opened and its max capacity
* The 2018 results nodes are under the respective names of the teams that played in the match for example the match where Ireland played England is under the name IreVEng.

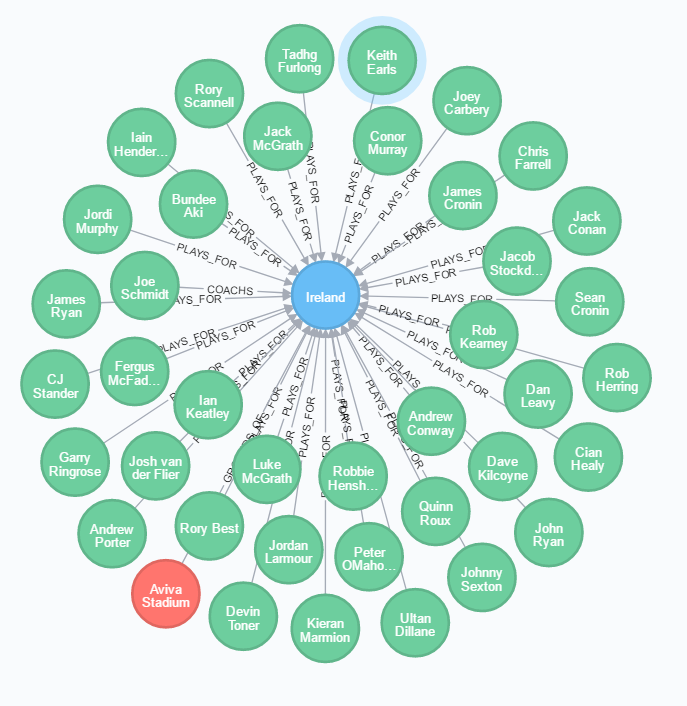
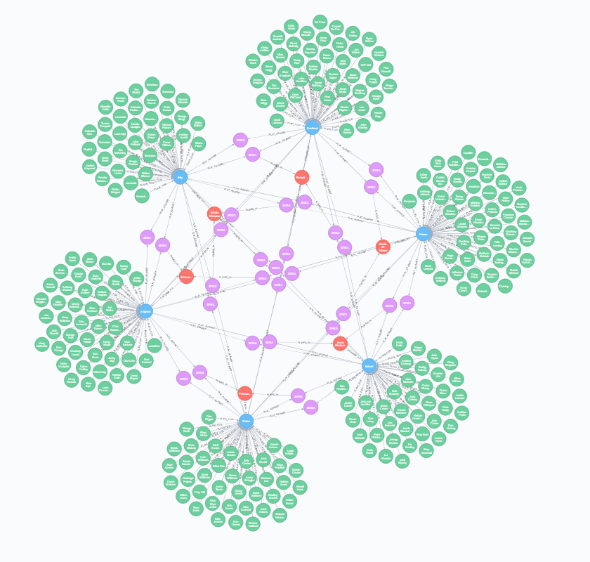


Figure 1. Figure 2.

## Design Methodology

Our main Design Methodology is to maintain a Flat Design throughout the program. This is a minimalist design style that suite information-oriented applications. A simple looking UI keeps user interaction easy and self-explanatory. Using solid practical colours and basic fonts also keeps everything readable and visible. A grid layout is also very common in this design methodology, allowing everything to be easily resized, and kept relatively symmetrical.

## Features of the Implementation

## Limitations

Our main limitation for this project was that despite our best efforts we could not get the project to serve our database from a remote VM run through google cloud. Also, we aren’t writing to the database. Had we done the dream team we would have implemented a write to the database allowing the user to create teams based on the data within the database using the players based on their positions.

## Known Bugs

## Recommendations for Future Development

In the future, our main objective is that we would like to have this hosted on a server. There are many advantages to having an app hosted on a server rather than just locally, such as on a server we would be able to update the app i.e. player details or actual code without having to manually update the app or reinstall it again. We would also be able to have more features.

One of the features we could implement once the app is hosted on a server is the ‘Dream Team’ page. You will be able to select players for each position they play from different teams and place them on the one team. Once you have filled up each position on the team you can save your team to the app and name it. Although it would work without being hosted on a server, the main part of this feature is to be able to compare your team to another team from another user who has saved it to the app. Having it on a server allows the user to submit and view as many fantasy teams as they want.

A recommendation we have for future reference would be to write this code in Android rather than UWP. We enjoyed using UWP, but there are limitations. Our application would only be available through the Microsoft store and would have to meet their requirements. But with Android, it would be available across more platforms and could do more with it. It is always advancing and would work best with what we were aiming for.

Another recommendation we have is if we were to rewrite this application in Android instead of UWP, we would use SQLite instead of Neo4j. SQLite works best with Android for a few reasons, such as the android.database and android.database.sqlite packages offer a  higher-performance alternative where source compatibility is not an  issue, the content can be viewed using third-party tools and content is updated continuously and automically so that there is no work lost in the event of a power failure or crash.

## Conclusions