# Six Nations Tracker

Brian Doyle – G00330969

Niall Devery – G00338932

Darragh King – G00326176

## Introduction

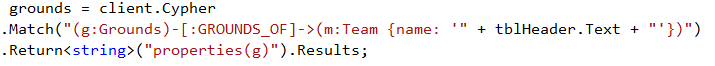
## System Requirements

## 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

## 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

## Known Bugs

## Recommendations for Future Development

## Conclusions