[Analysis 3](#_Toc510965513)

[Aim: 3](#_Toc510965514)

[Identification of project: 3](#_Toc510965515)

[Who Is the project aimed at? 4](#_Toc510965516)

[What is meant by the casual golfers 4](#_Toc510965517)

[Why casual golfers 4](#_Toc510965518)

[Talks with Clients 4](#_Toc510965519)

[Mr H 4](#_Toc510965520)

[Mr G 5](#_Toc510965521)

[How the Game of Golf Works. 5](#_Toc510965522)

[Current Score Cards 5](#_Toc510965523)

[What applications are currently out there? 6](#_Toc510965524)

[Golf application 1: “Distance Calculator”, creator: “Qi Chen”. 6](#_Toc510965525)

[Golf application 2: “Golf Game Book”, creator: “GameBook Oy”. 6](#_Toc510965526)

[Golfer application 4: “Golf Breaks Sunshine”, creator: “Sunshine Golf Breaks Ltd” 7](#_Toc510965527)

[Golfer application 5: “Golf GPS Range Finder Simple”, creator: “Bryan Thornbury” 7](#_Toc510965528)

[Evaluation of Research 7](#_Toc510965529)

[User Needs 7](#_Toc510965530)

[Must Be Able To: 7](#_Toc510965531)

[Should Be Able To: 8](#_Toc510965532)

[Could Be Able To 8](#_Toc510965533)

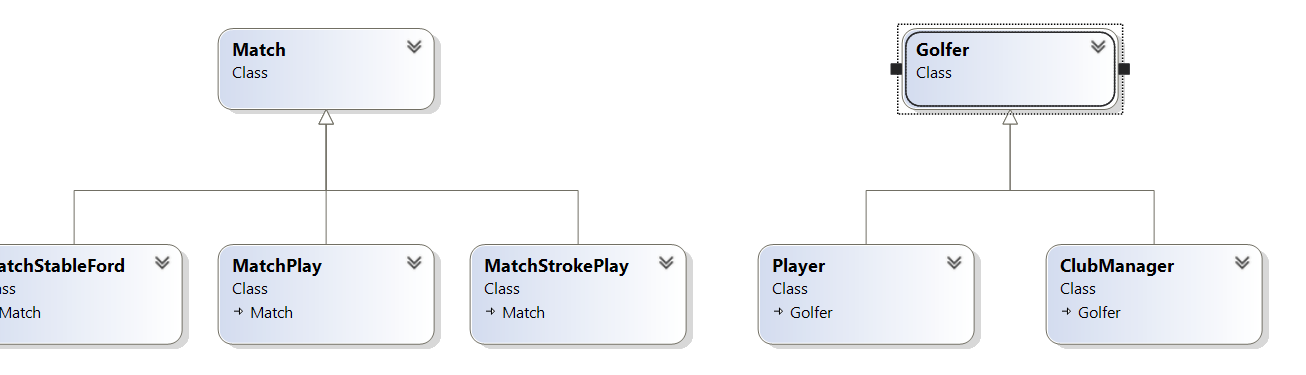
[Design 8](#_Toc510965534)

[General Design of Solution 8](#_Toc510965535)

[Why I chose an Object-Oriented approach? 9](#_Toc510965536)

[Why I chose to Naked Objects. 9](#_Toc510965537)

[Class Diagram showing my objects 10](#_Toc510965538)

[Class Diagram showing inheritance for matches. 11](#_Toc510965539)

[What Inheritance allows me to do 11](#_Toc510965540)

[Inheritance inside my program. 11](#_Toc510965541)

[Class Diagram showing associations. 11](#_Toc510965542)

[What is an association? 11](#_Toc510965543)

[Structure of Database. 12](#_Toc510965544)

[Technical Solution 12](#_Toc510965545)

[Overview 12](#_Toc510965546)

[Server 13](#_Toc510965547)

[Client 15](#_Toc510965548)

[Generic UI 16](#_Toc510965549)

[Customization 16](#_Toc510965550)

[Database 17](#_Toc510965551)

[Entity Framework 17](#_Toc510965552)

[Manual Database Mapping 18](#_Toc510965553)

[Database Customization 19](#_Toc510965554)

[Initializer 19](#_Toc510965555)

[Model 21](#_Toc510965556)

[Objects 22](#_Toc510965557)

[Properties 25](#_Toc510965558)

[Services 30](#_Toc510965559)

[Programing Techniques 32](#_Toc510965560)

[Specifics 32](#_Toc510965561)

[Testing 36](#_Toc510965562)

[Test 1 : Creating a New Account. 36](#_Toc510965563)

[What the user can and can’t do once logged in for the first time. 37](#_Toc510965564)

[How to create an account. 37](#_Toc510965565)

[Private Accounts 37](#_Toc510965566)

[Test 2: Enriching the User’s Profile 38](#_Toc510965567)

[Adding Friends 38](#_Toc510965568)

[Joining a Group 39](#_Toc510965569)

[Sending Messages 39](#_Toc510965570)

[Test 3: Creating a Match 39](#_Toc510965571)

[Browsing Courses 39](#_Toc510965572)

[Sending The invitations 39](#_Toc510965573)

[Creating the match 39](#_Toc510965574)

[Too many players want to join 39](#_Toc510965575)

[Playing the first hole. 39](#_Toc510965576)

[Test 4: Finishing a Match 39](#_Toc510965577)

[Evaluation 40](#_Toc510965578)

[Appendix 40](#_Toc510965579)

[All My Code 40](#_Toc510965580)

[References 40](#_Toc510965581)

# Analysis

## Aim

The aim of my computing project is to create a Golf application which can record scores from matches following various golfing match types, i.e Stableford, matchplay and strokeplay. The golf application should be for casual players who just want to have fun, record their matches and see whether their friends want to play a match or are currently in a match.

## Identification of project

Currently at Stowe Golf Club golfers record their scores on score cards which are pieces of card. These can be a problem because they are easily lost, damaged and can’t cope with rain. Personally, I believe that an online version will be much better as a lot more data can be stored, and an algorithm can be used calculate the score after the handicap and stroke index take effect. This application will run off a phone, so the scores can be filled during the match. This is the same belief as Mr. H, who is a professional golf coach at Stowe Golf Course. He will be acting as my primary client for this project. As a casual golfer a main reason to play the sport is the community that comes along side with it. To help the community with the application thrive a messaging service and a “Friends” system will be incorporated. A player should be able to see all the other golfers and if they are in a match the user can see which club they are playing at, the current score in that match and who they are playing against. Another aspect of the old card-based system is the score calculation. A lot of the golf match types can be confusing and involve a lot of confusing math which can lead to mistakes, with an online computer-based system this means that the computer will accurately calculate the score for each hole without failure if the algorithm is correct, therefore speeding up the game allowing more people to play per day. Finally, a way in which the new system shall help golfers is easy access to the clubs with pictures of the course, holes, descriptions of the clubs and a website link for each course. The user will also be able to see how many holes there are, the length of each hole. The final aspect as requested by Mr H is a match history system where a user can go through all their past matches see which holes they struggle on, leave a comment to say what went well, what when wrong in the match or even how windy it was on the day of the match. Being able to go through all your past matches is a very helpful aspect and helps each player to improve.

## Who Is the project aimed at?

The project shall be aimed at casual golfers rather than clubs who are running tournaments. This is there is a larger gap in the market for casual golfers. At tournament level there is a lot of rules required to prove validity of the results. One of which is that each golfer must score the other golfers but not them selves and therefore more than one scorecard is passed onto the official. This means that a separate scoring system will be required and proving validity will be very hard.

### What is meant by the casual golfers

A casual golfer is a golfer who wants to play golf but not in a competition. This means they don’t need validation in their results like a tournament would. A casual golfer will be playing at any time they want if the course is open.

### Why casual golfers

I have chosen to aim my project at causal golfers because these golfers like to analyses their results from passed matches. They also like to explore new clubs and having a database of clubs with information on each club. When casual golfers play a match with their friends they do not require validation to the extent of a tournament.

## Talks with Clients

### Mr H

As mentioned in the Identification of project Mr H is a golf professional coach at Stowe Golf Club. He teaches many golfers how to improve their game, plays tournaments and goes on golfing holidays. He loves to play causally at different clubs therefore has experience of causal golfer but also teaches many golfers and can get their opinion on the application.

In a talk with Mr H, I asked what was good with the card-based system and what was wrong with the old system. He responded, “The old system was very quick and easy to use. At a tournament you just write the score and a tournament official would calculate the actual score based on the match type. In a casual game the card-based system can be a pain as it takes a while to calculate the score. Another aspect of the card-based system is that everything is on it; the hole name, the yards for each hole and the stroke index. I still believe there is area to improve on the old card-based system. Another problem with the cards are that they are easy to lose to keep a accurate history of your matches where you can see which hole you struggle on is very hard. This is where an electronic system will improve the card-based system.”

### Mr G

Mr G is a teacher at Stowe but regularly helps with the Stowe Golf team. He’s very enthusiastic about the sport and will often spend his afternoons and weekends on the golf course. Mr G considers himself as a casual golfer since he doesn’t take part in any tournaments.

While talking to Mr G, he mentioned that a big part of the game is after the round of golf when you are in the club house talking about what went wrong on each hole or who played better when. Having a record of all the past matches with who won and how each hole went is a great way to impact this aspect of the sport. While talking to Mr G a spider diagram showing the possible areas where I could make an impact and the problems that I could face in making this application. Below shows the chart that was created with a few notes.

ADD Chart created with MR G

## How the Game of Golf Works.

How the main match types work will be discussed within the design section with a data flow diagram for each one. For small clubs like the Stowe Golf Club when you want to play a game you turn up, if you are a member of that club you can start straight away but if you are not a member you pay for your round of golf. On the course there a few standards that each golfer lives up to for example sweeping the bunker you found your self playing a shot from within the bunker, you only tee off once the previous players are out of your range or finished on the hole. When playing your round, you use the score card provided by the club to record the scores and you can either write down the raw score of the number of strokes or the score based on the match type you are playing. With certain match types there are little notations that get written on the card for example for match play the winner of the hole has a circle around their score. This will be discussed in the design section under Match Types.

### Current Score Cards

Below are two photos for the Stowe Golf Club Scorecard one of these scorecards has been filled in for a match play match type.



Each column of the score card will be described in the Design Section under Scorecard but an area to point out is the red sentence in the middle of the card. “Enjoy your day – please repair pitch marks” although this is a very small part of the scorecard its still an intriguing aspect to add to my version.

## What applications are currently out there?

Within my research I have gone on to the app store, downloaded and tested multiple apps to see what ideas they have and what makes a strong application as a user perspective. Below are a few that I believe could help my design and propose an interesting design.

### Golf application 1: “Distance Calculator”, creator: “Qi Chen”.

Although this isn’t the golfing application as previously mentioned in my analysis it proposes a very interesting design idea. It works by taking a photo of the golf flag you then crop the photo so that it’s the height of the bottom of the flag to the top. Then the application uses the size of the flag to calculate the distance till the hole. This application requires a calibration to set up the correct distance and I’m unsure of the accuracy of application, but it is still a very interesting idea and one to take into consideration when thinking of what to add to really make my product stand out.

### Golf application 2: “Golf Game Book”, creator: “GameBook Oy”.

This is like the design of project as I originally started with and contains much of what I initially thought my project would incorporate. It contains a strong social media section where you can post photos videos, golf tips and much more. This is an advanced application, but it might be interesting to add a few 5-minute golfing videos for people to improve their game. This application also includes an ability to see which friends are on a course practicing or playing a match/tournament this a very beneficial concept to the application and is one of the most valuable credits. The scoring section of the game is very in depth with thousands of clubs and courses on its database each with various amounts of data. Upon selecting a course, you then have a choice of which type of match you are playing out of the extensive list you then add your players including members and temporary players who don’t have an account. Finally, you add the scores for each player at each hole and when you finish it shows the winner. The only problem is the lack of data in match and on the club. This is something that I shall aim to improve when creating my application.

Golfer application 3: “Golf Weather” , creator: “GolfWeather”.

Does exactly what the name suggest. It tells you the weather at a chosen course for a week. It only tells you the weather for 6:00, 9:00, 12:00, 15:00, 18:00, the information per time shown is: the temperature, wind speed and whether there is going to be rain. It also tells you Sunrise and Sunset. These are all very important sets of information for deciding whether to play a game. This could be an interesting concept for the project although I’m not sure how I would be able to keep it up to date without doing it manually. There is also a function called course near me which takes your location and finds all the courses near that location, this will be good for players choosing where to play if they are rather new.

### Golfer application 4: “Golf Breaks Sunshine”, creator: “Sunshine Golf Breaks Ltd”

This travel app has a selection of pre-packaged holidays which vary in duration, amount of people and rounds of golf which want to be played. It also has a selection of Green Fees at famous clubs with buggies or walking. This is value which could be added to each of the courses by their managers. Although this would also a require a payment system to keep the users data secure and ensure the authenticity of my application.

### Golfer application 5: “Golf GPS Range Finder Simple”, creator: “Bryan Thornbury”

This range finder works very different to the first one but is still interesting. It works by presenting a google earth view. From this view you can see the golf course and your current position. The next step is to press a position on the map and it will mark the line from your position to where you pressed and marks the length in yards you can then press another position and it will calculate the distance from the new position marked and the previous one. This can be done as many times as wanted and in the top right it has the total distance. This is perfect for recording the length of multiple shots. When your done you can press the rubber and it would delete all the lines you have made. Adding a google earth view of the course is great for golfers scouting out the hole they are playing on. To improve on this app though the fair way, rough, bunkers and green could be marked out to provide a clearer view of the hole.

### Evaluation of Research

From my research into current applications I have learnt there is a broad range on concepts which could be incorporated into the design: a range finder, a weather system, a photos and video feed, packaged holidays per course and a bird’s eye view of the courses.

## User Needs

### Must Be Able To:

* Create a profile as a golfer where they can add their information.
* Create a match.
* A score card for each match
* Add other golfers to the match or join someone else’s match.
* Add a score for each hole in the match.
* The system must be able to work out the winner of the tournament.
* Go through all their past matches.
* Add a friend.
* Send a friend a message.
* Browse through courses to decide which one to play at.
* To be able to see whether their friends are in a match.

### Should Be Able To:

* Create a group for golfers who regularly play at the same course or together.
* Make an account private if a user doesn’t want to share certain details.
* To go back through the scores of a match and change a score.
* To send someone who is not a friend a message.
* To follow a website link to visit the website for that course.
* Look at a bird eye view of the course and each hole.
* Annotations on the score to show elements of that Match Type.

### Could Be Able To

* Look up the current whether status of a course.
* Packaged holidays for courses.
* A little note on the match provided by the course.
* A range finder to see how far you are from the flag.
* To schedule more than one match.
* Have a calendar showing big events and matches.
* An events feed saying what tournaments are where with live score feed.
* A membership section to your profile.
* Leave a description of how the match went.

# Design

## General Design of Solution

The overall product shall be presented as an application that can be accessed on any device but mainly a phone which then can be used while playing golf as mentioned in the analysis. The application must be as simple as possible and therefore shall consist as few buttons as possible. As the creator I have decided to use a few external sources to aid my project but more importantly I will be able to focus on the main business side of the application. This means I have more time to work on the capability of my program and giving the user the best possible product given the time constraints. I shall be using an Object-Oriented approach which will be discussed below. For the development of the application it will be run completely on one device using different browsers to simulate the different users. I shall also be using a software called Naked Objects. The purpose of Naked Objects is also mentioned below.

The main section will be the homepage which consists of all the functionality. It will be split up into separate categories: Matches, Golfers and Courses. Keeping a rigid structure to the software is important for writing the code but also helpful for the user. I have chosen these 3 categories since they are the most important and each of the user needs from the analysis can be placed in into one of the categories.

## Why I chose an Object-Oriented approach?

### What is Object-Oriented programing

Object-Oriented programing is a very important concept of computer science which allows for the use of many other concepts for example; encapsulation, inheritance, polymorphism and works with functional programing. Object Oriented Programing is where the world is viewed as “Objects” these objects can either be physical, like a Golfer, or abstract, like a data structure. This is very important for my program where there are many physical objects, which need to be incorporated for example; course, hole and golfer but there are also multiple abstract objects, match, score and friends. Each of these objects can interact with each other allowing them to be treated as if they were physical. Each of these objects must have properties and methods. A property is value an assigned to object this could be a golfer’s name or handicap. An action is a function that the object can perform. This could range from adding a value to joining a match.

### The effect on the database

In the database objects are treated as tables. Each object has an ID which acts as a primary key. Objects also have relationships meaning that the database will be a relational database. The relations within my database will be shown in an entity relationship diagram under structure of the database.

### Inheritance

Inheritance is concept in object-oriented program where there is one super class and then there is subclasses of the superclass. An example of this would be strokeplay being a subclass of Match. In inheritance all the knowledge in the superclass can be accessed by a subclass but not the other way around. If two objects have the same properties but separate actions inheritance will be very useful. Inheritance will be used a lot throughout my program for matches and for golfers where there will be different types of accounts.

## Why I chose to Naked Objects.

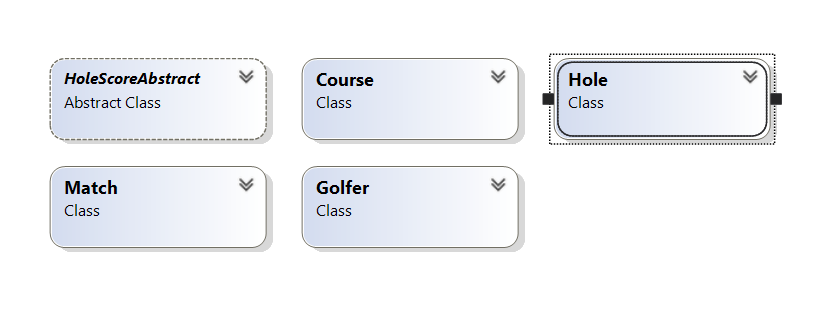
Naked Objects is an architecture used in the business world for the creation of various projects. Naked Objects is a framework consisting of 4 solutions: the client, the server, the model and the database. Each of the solutions is talked more in depth under technical solution. An object in Naked Objects can be one of two states: transient and persisted. When an object is transient it hasn’t been saved to the database but when its been persisted the object will be in the database. An object can not go from persisted to transient but can go from transient to persisted.

Naked Objects includes restful api..

Naked Object also contains a lot of files which create the initial interface for you. These files could take a long time to create and, in a time pressured project most of the time is centered towards the main functionality of the program and therefore having a foundation to build on is very useful. This has allowed me to change fonts, to authorize values and classes, and change the color of objects along with multiple different little things. Being able to change the color of objects is very useful for the user as they can easily differentiate between objects and now what they are doing. The interface also has a menu made for it. Allowing for easy navigation between objects and pages.

## The Main Objects

The list of objects below are the main objects to complete the user needs. Under each heading is the list of properties that object will have and their actions visible to the user.



### Course:

#### What is the Course object:

There will be no actions on the course but the only type of user that will be able to edit the course will be the course manager for that course. This will be done by authorization which is defined under Authentication and Authorization A course manager will be a subclass of Golfer which is mentioned under Golfer. From course you can navigate to all the holes by opening the ICollection. To create a new course as a club you will need to speak to a systems manager who will create the Club Manager and then a empty course will be created for the Club manager to add the values.

#### Properties

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Properties | Type | Visible to player? | Editable by player | Visible to Club Manager | Editable by Club Manager | Validation | Optionally |
| ID | Int | No | No | No | No | No | No |
| Course Name | String | Yes | No | Yes | Yes | No | No |
| Location | String | Yes | No | Yes | Yes | No | No |
| Address | String | Yes | No | Yes | Yes | Length | No |
| Par | Int | Yes | No | Yes | Yes | No | Yes |
| Club Manager | Golfer | Yes | No | Yes | No | No | No |
| Website Link | String | Yes | No | Yes | Yes | No | Yes |
| Yardage | Int | Yes | No | Yes | Yes | No | Yes |
| Phone Number | Int | Yes | No | Yes | Yes | Length | No |
| Photo | File Attachment | Yes | No | Yes | Yes | No | No |
| Holes | Hole collection | Yes | No | Yes | Yes | No | No |
| Facilities | Enum collection | Yes | No | Yes | Yes | No | No |
| Course Description | String | Yes | No | Yes | Yes | No | Yes |

### Golfer:

#### What is the Golfer object:

A golfer is the account that you as a user will create when joining the application. There two types of Golfer, Club Manager and Player. Club Manager accounts can only be set up by a systems manager therefore as a new user you will be forced into being a Player.

#### Subclasses

##### Club Manager

##### Player

#### Properties and Actions

### Match:

#### What is the Match Object

Match is the super class for the other match types it contains the collection of golfers showing another many to one relationship as a golfer can only be in one match at a time. It also has the one to one relationship for course.

#### Subclasses of Match

##### Match Play

##### Stroke Play

##### Stableford

#### Properties and Actions

### Hole:

#### What is the Hole object:

As mentioned before holes interact with course but other than that they have no method on them self just optional data. This will effective be the exact some rows that are on a paper score card.

#### Properties and Actions

### HoleScores:

#### What is HoleScores

This score for each hole and portrayed as a table within the interface. This is the actual score sheet that you can see. There is two objects since match play only has 2 players but stableford and strokeplay has 4 players.

#### Subclasses of HoleScores

#### Properties and Actions

## Possible Further Classes



### Group:

is a collection of golfers with a group owner and a collection of members this just allows quick access to your friends or people you play with on a regular basis, there can also be a collection of messages that everyone in the group can see.

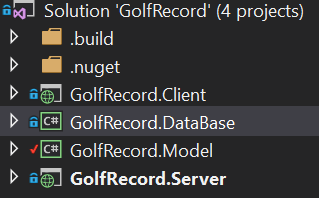
### Invite:

This is just a Boolean that says whether to accept to be in that match.

### Message:

# Technical Solution

## Overview



The screen shot above shows the layout of my solution. Since I am using Naked Objects to aid me in the project I have been given a template which consists of 4 projects; client, database, model, server. Each of these projects will be discussed below. The use of the template allows for separation of concerns, this means I can separate the code into their distinct objectives for example keeping all the code that affects the database separate from all the code that affects the client. For future this means that I could create a completely new client for my project without having to edit the rest. This is shown by the diagram below of the architecture for how my project would be deployed in future. At that this instance the project is all being run on one machine including the server and the client but for the future a separate machine would be used to run the server.

Runs on a Server machine

Database

Model

Client

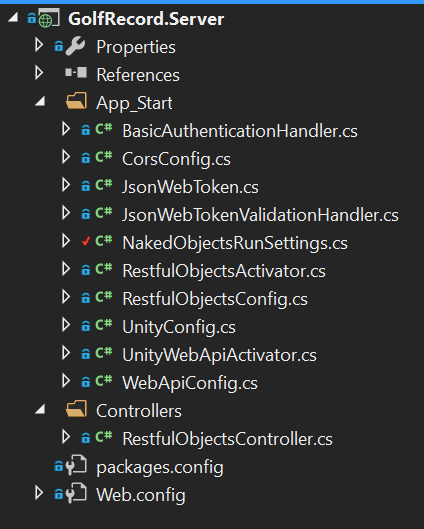
Server Project

The diagram shows the client is separate to the content that is run on the server machine this is because the client is generic in the web browser therefore can be run on a computer or a smartphone which was a request from the customer for the project. The dashed line shows how the RESTful API is used to communicate between the client and the server. The other arrows show the dependency of each project, i.e. the Database is dependent upon the model.

The use of naked objects also gives me various other code and files to create a more advanced project given the time constraints. This allows focus on the functionality and the business code rather than spending a larger proportion of my time on the set up and defining of the interface, which although is important would take a considerable length of time.

## Server

The server takes all the logic from the model and the database and uses the logic to create a RESTful API for the client to communicate with the server using the HTTP request, response techniques. I have not created the server project, but it has been supplied to me by Naked Objects. As mentioned in the Overview the server would eventually become a separate machine that the clients phone or laptop would access.



I have made configurations to the server within the NakedObjectsRunSettings class. The screen shot of the code is all the changes that I have personally made:

private static Type[] Types

{

get

{

return new Type[] { manually enter subclasses since the framework cant discover the objects.( put down the bottom)

typeof(MatchStableFord),

typeof(MatchStrokePlay),

Here I had to declare all the object types that the user can discover so the server knows what to search for when creating the menu.

typeof(MatchPlay),

typeof(FourPlayerHoleScore),

typeof(TwoPlayerHoleScore),

typeof(ClubManager),

typeof(Player)

};

}

}

private static Type[] Services

{

get

{

The services are also defined here so that the server doesn’t confuse the services as objects to be added to the menu. The services will be mentioned at a later stage within the model in more depth.

return new Type[] {

typeof(GolferServices),

typeof(MatchServices),

typeof(CourseServices),

typeof(HoleServices)

};

}

}

Here I am defining the main menu that is visible at the homepage. The reason for using the services since the homepage isn’t an object these are the methods that we can navigate with and retrieve or create domain objects.

public static IMenu[] MainMenus(IMenuFactory factory)

{

return new IMenu[] {

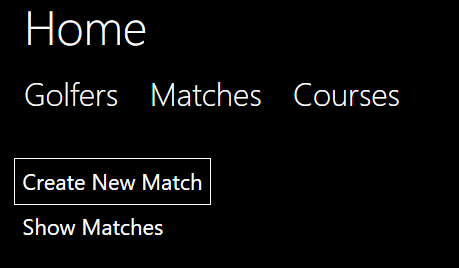
factory.NewMenu<GolferServices>(true, "Golfers"),

factory.NewMenu<MatchServices>(true,"Matches"),

factory.NewMenu<CourseServices>(true,"Courses")

};

}



This screen shot is the product of the code above. I have expanded the Matches showing all the methods in the MatchServices.

public static IAuthorizationConfiguration AuthorizationConfig()

{

var config = new AuthorizationConfiguration<DefaultAuthorizer>();

// config.AddNamespaceAuthorizer<MyAppAuthorizer>("MyApp");

// config.AddNamespaceAuthorizer<MyCluster1Authorizer>("MyApp.MyCluster1");

config.AddTypeAuthorizer<MatchStrokePlay, StrokePlayAuthoriser>();

config.AddTypeAuthorizer<MatchStableFord, StableFordAuthoriser>();

config.AddTypeAuthorizer<MatchPlay, MatchPlayAuthoriser>();

config.AddTypeAuthorizer<Player, PlayerAuthoriser>();

config.AddTypeAuthorizer<ClubManager, ClubManagerAuthoriser>();

config.AddTypeAuthorizer<Hole, HoleAuthorier>();

config.AddTypeAuthorizer<Group, GroupAuthoriser>();

config.AddTypeAuthorizer<Match, MatchAuthoriser>();

return config;

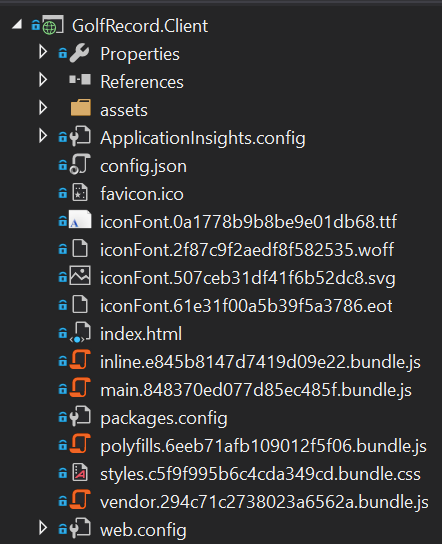
}

Finally, I have had to declare all the Authorizers so that the server doesn’t think they are objects than our accessed by the user. Authorizers will be further discussed in the model.

}

}

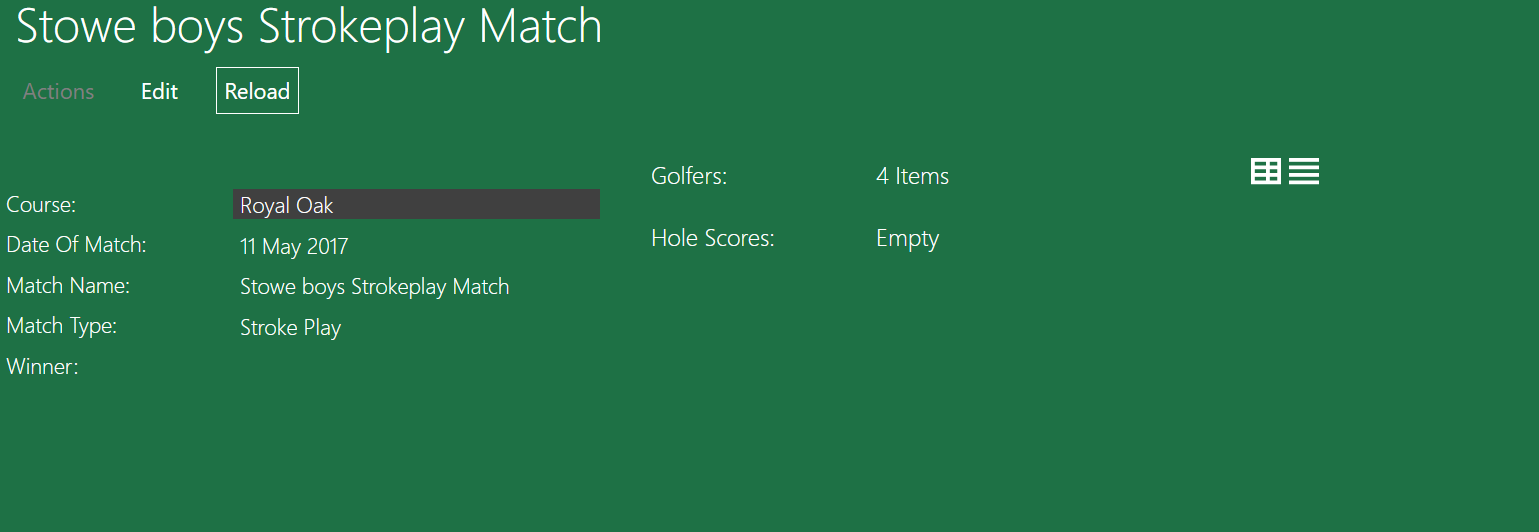
## Client



### Generic UI

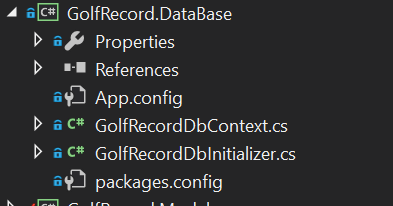
The client is a generic user interface that has been supplied by Naked Objects. There are thousands of lines of code which vary in language. For example some of the classes are written in HTML like the index.html, other classes are written in css. One of the benifical factors of using the NakedObjects generic UI is when executing an action that takes in an object, the user may drag and drop an object in. Here is a screen shot of the user dragging an object into a parameter.

### Customization

I have made a few changes to code all of which are in the config.jason file here I have told the solution I am using authentication and an authorizer this is the demo version of auth0. I will talk about auth0 at a later stage in the Specifics section. In addition, in the config.jason file I have chosen to change a few colors of objects in order to help the user distinguish what they are looking at. An example of this is for the match object. 

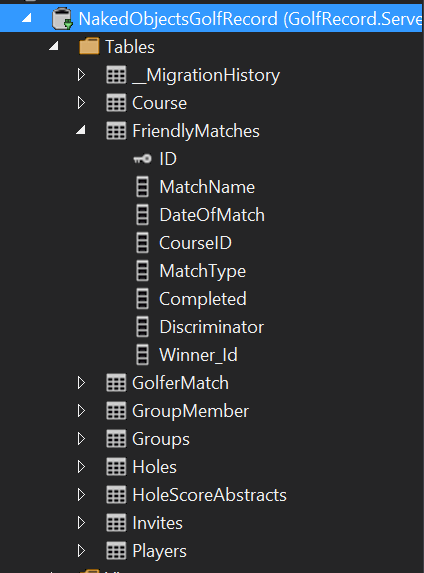
As you can see the Match is green, but the course is black this helps the user to know that from match they can travel to the course by clicking that link which now stands out.

## Database



### Entity Framework

The database is working off an object relational mapping framework made by Microsoft called entity framework. This synergizes well with my project as it automatically sets up the database through the code I have written. where each object is a separate table and an Icollection is a 1 to many relationships. The list of tables is shown below.



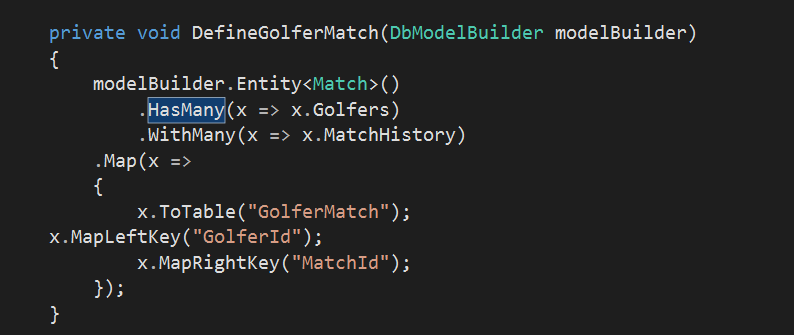
Here I have expanded the Matches table to show all the columns. The key next to the ID represents the primary key. I have customized the table here to called it “Friendly Matches” How I have done this will be mentioned Database Customization but the reason I have done this is because in future this system could be used for completive matches with separate rules.

There are a few tables which I have manually mapped which will be talked about in manual mapping section below. The use of Entity Framework means that I can use SQL reports to obtain certain information this is useful for analyzing the data collected and for a user to check the data for their past matches.

### Manual Database Mapping

When there is a problem where the entity framework hasn’t quite managed to map the database correctly I have had to manually do it. This is done by creating a function in the DBContext class which defines a new table with a primary key using modelbuilder this function is then called in a separate function called OnModelCreater

I have had to manually map a table twice; one is a many to many relationship f where many Golfers have many match histories, but a match can belong to many golfers. To solve this, I have defined a link table called Golfer match which can be seen in the screenshot above. This new table is a link table used to turn the many to many into two one to many relationships. In the new table golfer match it uses a compound foreign key as the primary key which are GolferID and MatchId. The code below shows how I have done this. The code is written within the DBcontext and is being called using a method from the modelbuilder function which can be seen below the screen shot.



Here I am stating that the many to many relationship that currently exists. Is this a linq statement?

Here I am defining the new tables name and what the compound foreign primary key is.

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

Only the bottom two are creating a new table. The other 3 methods are for customization which will be mentioned in Database Customization.

DefineGolfer(modelBuilder.Entity<Golfer>());

DefineMatch(modelBuilder.Entity<Match>());

DefineCourse(modelBuilder.Entity<Course>());

DefineGolferMatch(modelBuilder);

DefineGroupMember(modelBuilder);

}

The second example of manual mapping is a one to itself relationship. This is for the GolferFriends. Where a Golfer may have many golfers as friends. Once again to solve this problem I have created a new table called GroupMember which uses a new object that has been created in the model called Groups. A group has a golfer who is the owner and a collection of Golfers who belong to the group.

private void DefineGroupMember(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Golfer>()

Group is an entirely new object that I created from with the Model which can be seen In the Model. I created the new object so that I can have two different primary keys and therefor it can be mapped correctly.

.HasMany(x => x.Groups)

.WithMany(x => x.Members)

.Map(x =>

{

x.ToTable("GroupMember");

x.MapLeftKey("Id");

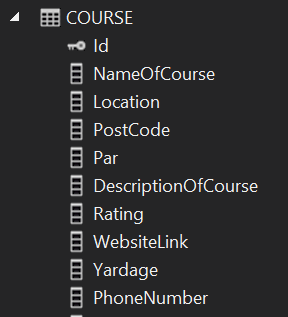
x.MapRightKey("GolferId");

});

}

### Database Customization

In addition to manual mapping I have also been able to customize the database where I feel a better name could be used. An example of this can be seen below.



Here is the table for course. I have renamed a few coloumns in the table so that they read better to the user. DescriptionOfCourse is actually CourseDescription this does change the property when the code is running its only for the database.

private void DefineCourse(EntityTypeConfiguration<Course> courseconfiguration )

{

courseconfiguration.ToTable("Course");

courseconfiguration.HasKey(c => c.Id);

courseconfiguration.Property(c => c.Address).HasColumnName("PostalAddress");

courseconfiguration.Property(c => c.CourseDescription).HasColumnName("DescriptionOfCourse");

courseconfiguration.Property(c => c.CourseName).HasColumnName("NameOfCourse");

}

### Initializer

For creating example data used for test runs I am using an Initializer. It is a separate class within the Database project. The class consists or a group of methods that I have created returning an object which are saved to the Context. These methods are then accessed above with sample data being passed in for each parameter required. There are various other techniques which could be used instead for example SQL Inserts but I chose to use an Initializer as it reads better with its syntactic sugar of “=”. This means that there are already golfers and matches inside the program before I run it so to do a test run I don’t need to create 4 separate golfers then add the 4 golfers to the match and then add 18 scores for each golfer. This becomes increasing important for when I’m doing my test. To add an example data, I need to create a help method which returns that object. This method is then called which each value declared and then saved to the database. Changes currently made while the project is running doesn’t get saved but for future when the application goes live this would be changed so that people matches get saved.

The code below shows an example of a method within the DBInitializer;

I have set the Initializer up so that if the user doesn’t enter a match when calling the method is automatically set the type to strokeplay since this is the most common.

private Match AddNewMatch(string name, DateTime date, int courseID, MatchType matchType = MatchType.StrokePlay)

{// work for each match type

Here I am using a switch case statement to ensure the match is one of the three types and therefore doesn’t break.

Match m = null;

switch (matchType)

{

case MatchType.StrokePlay:

m = new MatchStrokePlay() { MatchName = name, DateOfMatch = date, CourseID = courseID, MatchType = matchType };

break;

case MatchType.MatchPlay:

m = new MatchPlay() { MatchName = name, DateOfMatch = date, CourseID = courseID, MatchType = matchType };

break;

case MatchType.StableFord:

m = new MatchStableFord() { MatchName = name, DateOfMatch = date, CourseID = courseID, MatchType = matchType };

break;

default:

break;

}

Context.Matches.Add(m);

Context.SaveChanges();

return (m);

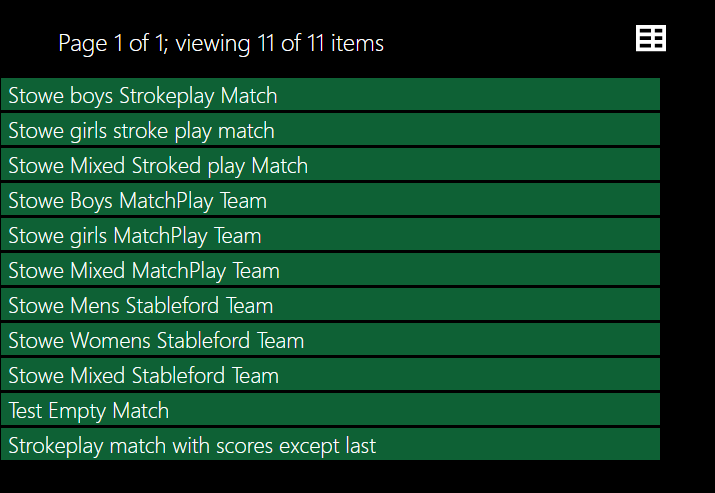
}

An example of how this method would be called is shown below. Once again at the end of the method I am saving the changes to the context. It doesn’t need to be at the end of each call but I have done it to be safe.

var s9 = AddNewMatch("Stowe Mixed Stableford Team", date1, 4, MatchType.StableFord);

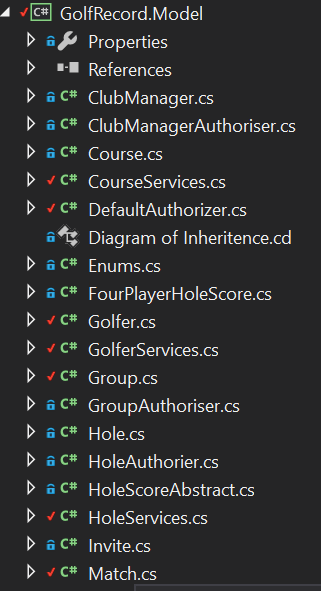
context.SaveChanges();

The screen shot below shows all the example matches I have created for testing. There is; an all-male, all-female and mixed gender match for each match type. Some of these matches have scores already added in but some don’t.



## Model

The model is the most important part of my code as it is where I have written all the business logic and functional application. The model is made up of 3 types of classes. Each of the types of classes will be discussed below. Within the Model I have defined all of my objects, their services and there authorizers where necessary. A lot of my program evolves around the authorization since it is live multiuser application and therefore many players or managers will have access to different properties and actions. These will also be discussed below. The screen shot below shows the classes within the Model project.



### Objects

An object is a computing construct used to represent a either a physical object or a more abstract object for example a holescore. An object consists of properties and actions. The code for an Object can be seen below.

public class Golfer

At the top I have declared an Object Container this is so that I can access certain actions that are provided by Naked Objects where an object is not provided.

{

public IDomainObjectContainer Container { set; protected get; }

public GolferServices GolferConfig { set; protected get; }

public CourseServices CourseConfig { set; protected get; }

public MatchServices MatchConfig { set; protected get; }

Here I am importing the methods from other classes called services.

[NakedObjectsIgnore]

public virtual int Id { get; set; }

[Title][MemberOrder(1)]

public virtual string FullName { get; set; }

[MemberOrder(2)]

public virtual int Handicap { get; set; }

Each of these variables are properties that belong to the object. Some of the properties have words in [] these are attributes and will be discussed down below under attributes. A screen shot showing how the interface portrays these properties is under the code. Properties will be discussed below under

[Optionally][MemberOrder(4)]

public virtual string Mobile { get; set; }

[Optionally]

public virtual Gender Gender { get; set; }

public virtual Title Position { get; set; }

public virtual string Username { get; set; }

#region Friends (collection)

private ICollection<Golfer> \_Friends = new List<Golfer>();

public virtual ICollection<Golfer> Friends

{

get

An Icollection is Naked Objects version of a group of objects all under the same object type. Here is a group of Golfers named friends. Each Icollection has a method to add a value to the object this is the method below

{

return \_Friends;

}

set

{

\_Friends = value;

}

}

public void AddFriend(Golfer golfer)

{

Friends.Add(golfer);

}

[PageSize(3)]

public IQueryable<Golfer> AutoComplete0AddFriend([MinLength(2)] string matching)

{

return GolferConfig.AllGolfers().Where(g => g.FullName.Contains(matching));

}

#endregion

This shows a complementary method. Complementary methods will be discussed below under Complementary methods.

#region MatchHistory (collection)

private ICollection<Match> \_MatchHistory = new List<Match>();

public virtual ICollection<Match> MatchHistory

{

get

{

return \_MatchHistory;

}

set

{

\_MatchHistory = value;

}

}

public void AddMatchHistory(Match match)

{

MatchHistory.Add(match);

}

public IQueryable<Match> AutoComplete0AddMatchHistory([MinLength(2)] string matching)

{

return MatchConfig.ShowMatches().Where(m => m.MatchName.Contains(matching));

}

#endregion

#region Groups

public Group CreateNewGroup()

{

var group = Container.NewTransientInstance<Group>();

group.GroupOwner = GolferConfig.Me();

return group;

}

private ICollection<Group> \_Groups = new List<Group>();

public virtual ICollection<Group> Groups

{

get

{

return \_Groups;

}

set

{

\_Groups = value;

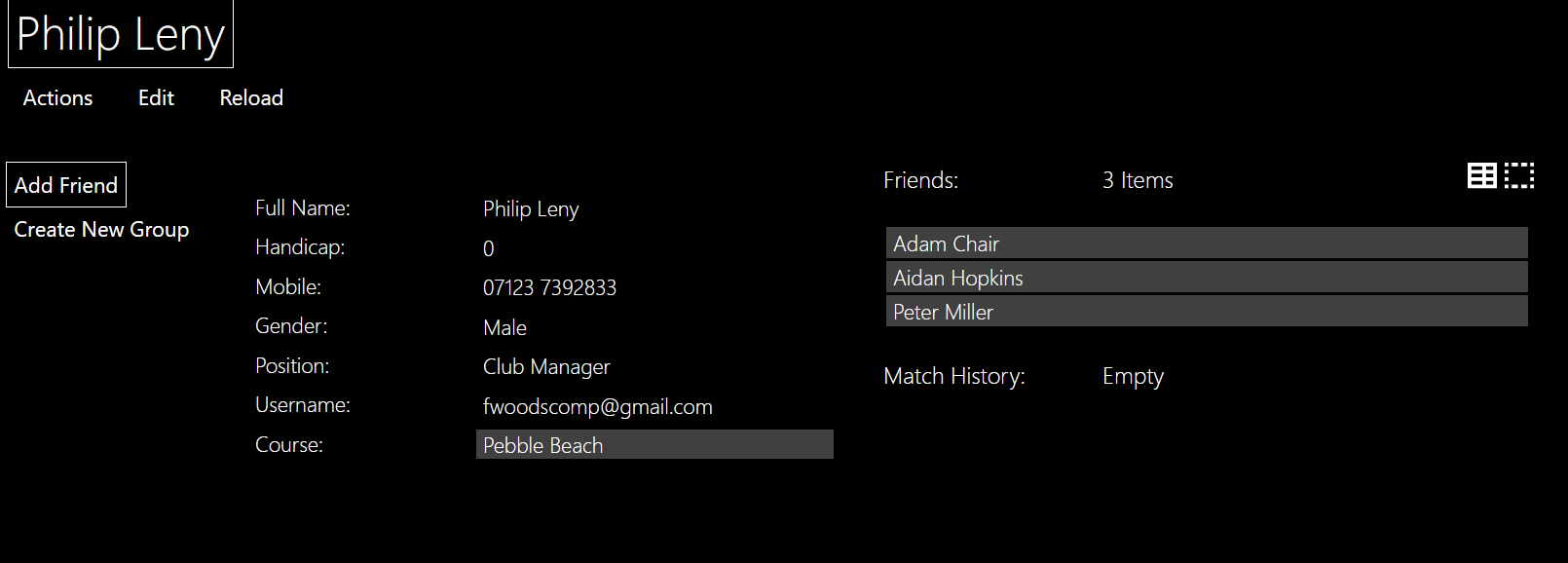
}

}

#endregion

}

}

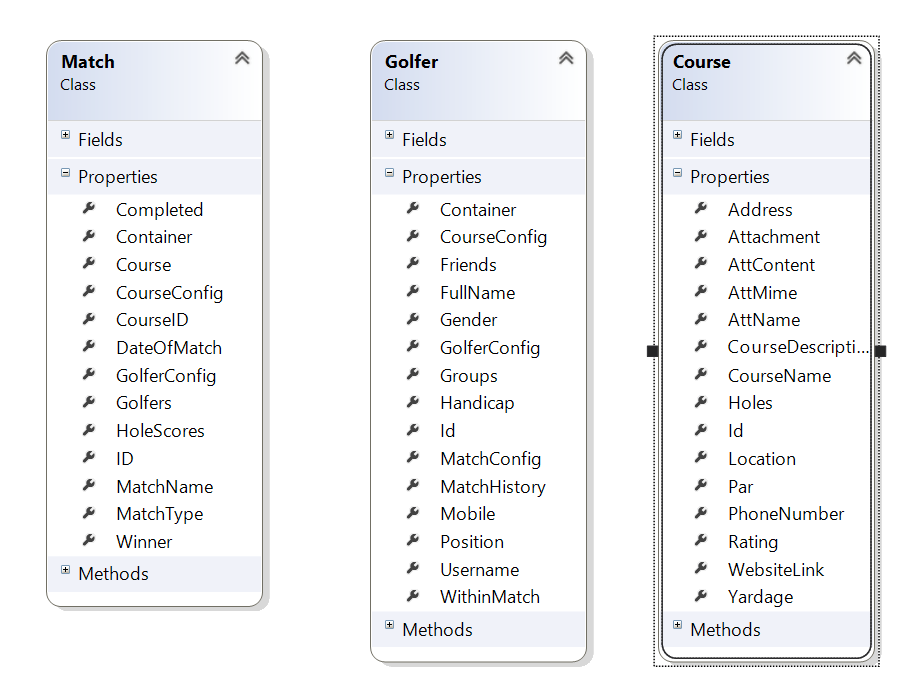


Here are the Icollections I have exapanded the collection of friends.

Since the propety Course is decalered as the object Course is can be used as a link for the user to access the course.

### Properties

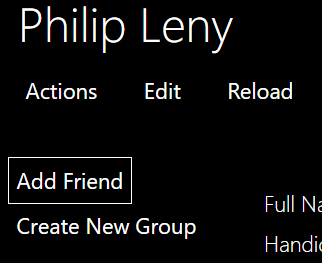
As mentioned above properties are the values assigned to an object. Some properties are hidden using the [NakedObjectsIgnore] attribute. Attributes will be mentioned below. With Inheritance If an object inherits from another object for example; player inherits from golfer this means that the properties declared in the super class, Golfer are accessible from player but not the other way around.



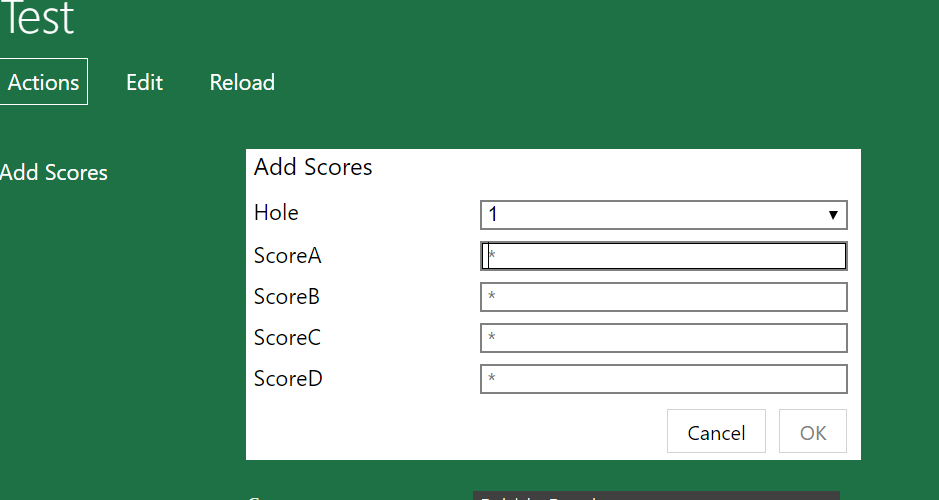
The 4 properties, “Attachment, AttContent,AttMime, AttName:” are used to portray a photo.

This photo shows the properties of the 3 main objects. As you can see some objects have a lot of properties therefore it is important to use authorization and attributes. Both of which are discussed below.

#### Actions

Actions are methods defined within an object. They are accessed by the user from within the object by pressing the tab in the top left. By pressing this tab, the list of actions accessible the user is displayed. An example of this is seen below.

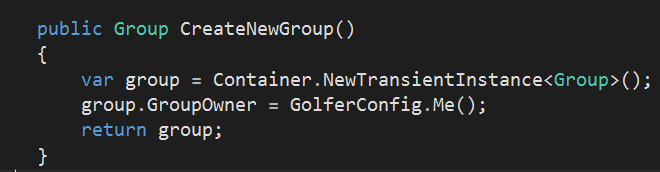
To prevent a user from seeing an action you can either use authorization or hide complementary method. Some actions take in parameters. Which using attributes can be optional otherwise Naked Objects will set to necessary by default. If a method requires another object as a parameter the Naked Objects will allow provide a drag and drop interface but the use of complementary methods will provide a user friendly technique of adding the object. All of these techniques are further discussed below. The screen shot below shows the user interface for entering the parameters for a action.



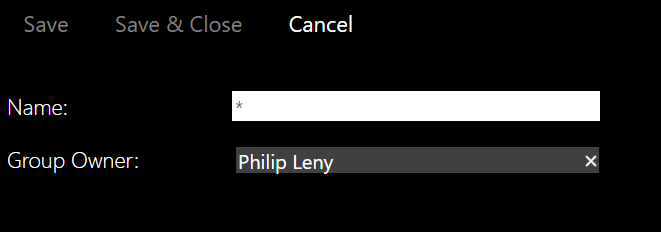
The asterixis within the boxes indicate that parameter is not optional. The Ok button is faded and cannot be accessed therefore preventing errors. The Hole parameter is a drop down of the number of holes which automatically increments each time the method is called using complementary methods.

Finally if a parameter is constant you can automatically provide the parameter by default. Since an object can either be transient or persisted. A value can be specified while the object is transient before it has been persisted therefore the value can be specified before the user can access it. (I think this is true but would like to check)

The code for this is shown below:

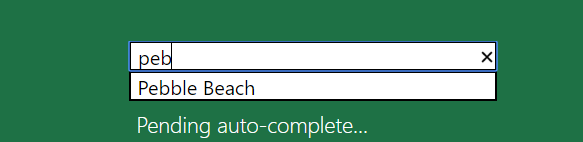


Since the variable, “group” is returned after the property group owner is specified therefore it is still in the transient stage. This is portrayed to the user in the following screen shot:



#### Complementary Methods

A complementary Method is a method which complements another method. An example of this is autocomplete. To make my program more user friendly I have added autocomplete to prevent the user having to type in the full name or search through pages of golfers. An example of this is seen below.



A complementary method is declared by using <ComplementaryMethod>0<methodtocomplement> An example of this is:

##### AutoComplete

public IQueryable<Golfer> AutoComplete0AddFriend([MinLength(2)] string matching)

Naked Object specifies a select group of complementary methods. The use of 0 specifies the difference between the two methods. This means that all the complementary methods are hidden from the user.

##### Choices

Choices specifies the possible options.

public IList<Hole> Choices0AddScores()

{

This provides a drop down menu of all the possible holes

if (HoleScores.Count == 0)

{

return Course.Holes.ToList();

}

else

{

// return Course.Holes.ToList();

return (from h in Course.Holes

from s in HoleScores

where h.Id != s.HoleId //a querry across two sources.

select h).ToList();

}

This sets the value for hole to the last value + 1

}

##### Default

Default is a method for obtaining an automatic parameter.

public Hole Default0AddScores()

{

int nextHole = 1;

if (HoleScores.Count > 0)

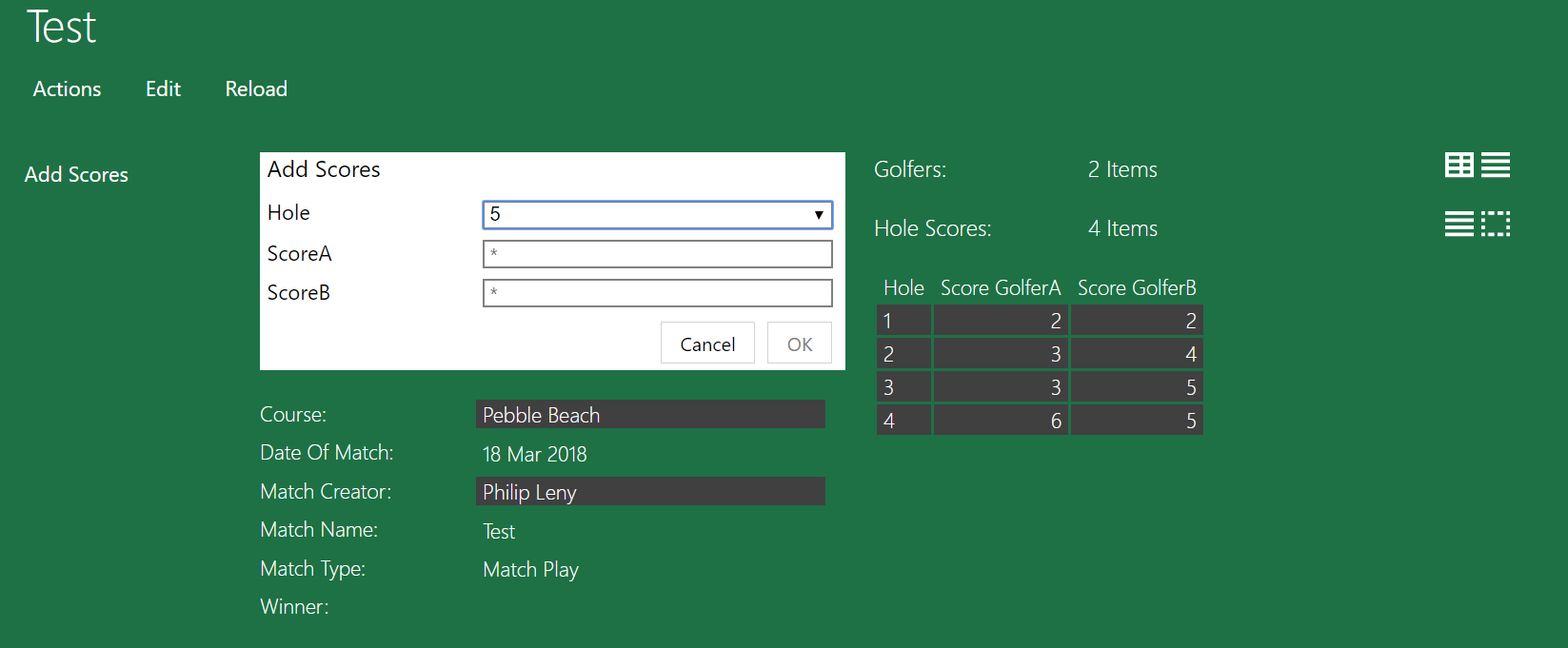
{

nextHole = HoleScores.Max(hs => hs.Hole.HoleNumber) + 1;

}

return Course.Holes.First(h => h.HoleNumber == nextHole);

}



As you can see there is 4 scores already added and the method has automatically set itself to 5. This is due to the default complementary method.

The choices method resulted in a drop down just like an enum would. This is very helpful since some courses require 9 holes and some 18 therefore 2 sets of enums would be required.

#### Attributes

Attributes are parameters used for the interface portrayed to the user. This can range from ordering properties, hiding properties or declaring a property as optional. There are many different attributes. Below is an example of: MemberOrder, Title, NakedObjectsIgnore, Pagesize and finally Optionally. These are all attributes that I have included in my program. The screen shot below shows most of these attributes. You can have more than one attribute per property. Attributes can also be applied to methods.

##### MemberOrder

MemberOrder is an attribute used to portray the properties to the user in a certain order.

As you can see from the screen shot the first property is full name the second is handicap and they are the most important mobile is next.

##### Title

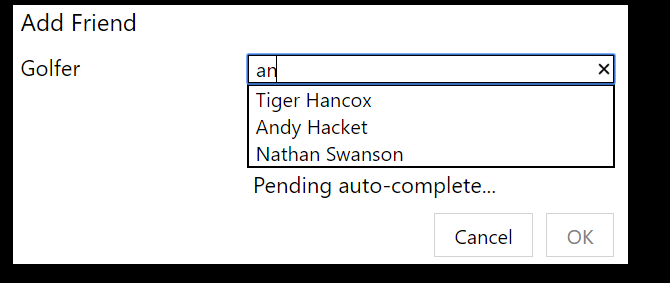
Title declares the title of a certain object. An example of this is the title of Golfer being the Golfer’s name. This can be seen in the screen shot as the title.

##### NakedObjectsIgnore

NakedObjectsIgnore is used to prevent the user from seeing a certain property an example of this is preventing the user from seeing the ID of a golfer. This is because the Id of the golfer provide no value to the golfer but is a necessary value for the program.

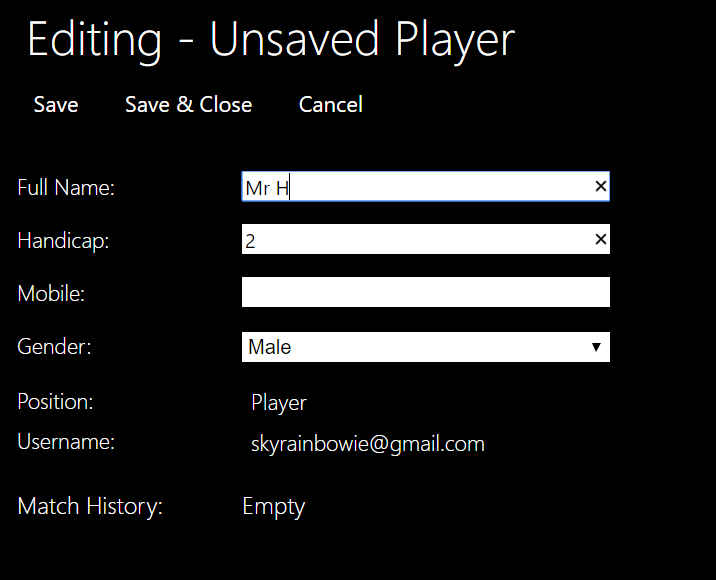
##### PageSize

PageSize is used in a complementary method of autocomplete. When the user in entering the name of the object they are searching for it will provide a list of 3 possible objects. This is not portrayed in the screen shot above but can be seen in the one below.



##### Optionally

This declares a property as optional therefore doesn’t need to filled in. An example of this is a golfer entering their mobile. Some users may not want to share such personal information to all the users therefore I have made it optional. When creating an object using a action or service method an optional parameter can be seen as having no astrix.



As you can see the Save and Save & Close actions are available to be used meaning I can create this golfer without entering a mobile.

### Services

Each of the main objects also have a services class this contains the helper methods that are used to access an object. When click the main header you are given a drop down of options. These options are the methods from within the services. Service methods are held outside the main class since they are methods that don’t require an action to access it. For example if you need to create a golfer but you don’t have a golfer to access it.

The code for a service is shown below:

namespace GolfRecord.Model

{

public class GolferServices

{

#region Injected Services

//An implementation of this interface is injected automatically by the framework

public IDomainObjectContainer Container { set; protected get; }

#endregion

Once again I am using the Container to access methods since I have no object.

public IQueryable<Golfer> AllGolfers()

{

return Container.Instances<Golfer>();

These methods return a type IQueryable this means if there is a lot of the that object it returns in page by page rather than all in one table like a ICollection

}

public IQueryable<ClubManager> AllManagers()

{

return Container.Instances<ClubManager>();

}

public Golfer Me()

{

var username = Container.Principal.Identity.Name;

var user = AllGolfers().Where(g => g.Username.ToUpper().Contains(username.ToUpper())).SingleOrDefault();

if (user == null)

{

user = Container.NewTransientInstance<Player>();

user.Username = Container.Principal.Identity.Name;

user.Position = Title.Player;

return user;

}

else

{

return user;

}

}

[NakedObjectsIgnore]

public bool IsPlayer()

{

if (Me() != null)

{

return true;

}

else

{

return false;

}

}

public IQueryable<Golfer> FindGolferByName(string name)

{

return AllGolfers().Where(c => c.FullName.ToUpper().Contains(name.ToUpper()));

}

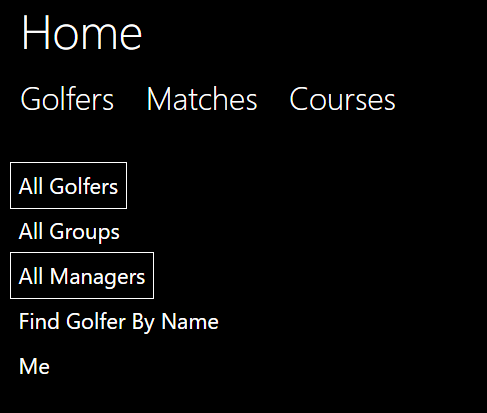
public IQueryable<Group> AllGroups()

{

return Container.Instances<Group>();

}

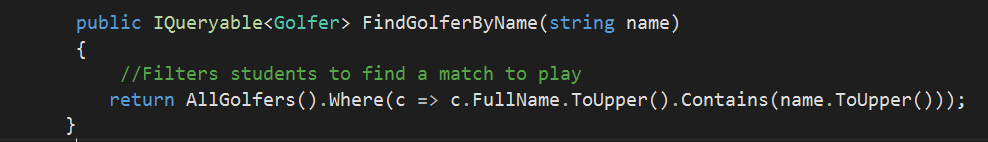
These are portrayed in the user interface as:



As you can see they don’t require an object to access the methods.

### Programing Techniques

#### LINQ

Throughout multiple stages of my programing I have used C# multi paradigm ability to add “LINQ” statements to my code whether deemed useful an example of this is within the class GolferConfig.

The use of functional programing here is useful as it is easy to read, and you know exactly the result that you should obtain, therefore its very reliable. Finally, it reduces the amount of statements of code significantly as function program requires no sequential statements. This can be shown in the line of code above without a Linq statement this could have taken multiple lines of code.

### Specifics

#### Algorithms

Within golf there are some very intricate rules for calculating scores and finding the winner. The most confusing for me was the stableford. The rules are mention in the design section. To come up with an algorithm I decided to split the problem up into separate functions and then work from there. This technique worked well.

Inherits from match.

public class MatchStableFord : Match

{

public int[] TotalScores;

public void AddScores(Hole hole, int ScoreA, int ScoreB, int ScoreC, int ScoreD)

{

var hs = Container.NewTransientInstance<FourPlayerHoleScore>();

Container.Persist(ref hs);

The user inputs the score. I have had to persist hs so that the user enters the score and then the algorithm can work.

hs.ScoreGolferA = ScoreA;

hs.ScoreGolferB = ScoreB;

hs.ScoreGolferC = ScoreC;

hs.ScoreGolferD = ScoreD;

hs.Hole = hole;

HoleScores.Add(hs);

Here I am calling the functions and returning them as an array.

int[] StrokeIndexs = StrokeIndexEffect(hole);

int[] GenderEffectOfGolfer = GenderEffect(hole);

int[] Scores = { ScoreA, ScoreB, ScoreC, ScoreD };

TotalScoreCalculated(hole, Scores, hs, StrokeIndexs, GenderEffectOfGolfer);

Container.Persist(ref hs);

HoleScores.Add(hs);

if (hole.HoleNumber == Course.Holes.Count)

{

Winner = Golfers.ElementAt(FindWinner());

for (int i = 0; i < 4; i++)

{

Golfers.ElementAt(i).MatchHistory.Add(this);

}

This Iteration will automatically end the match and calulate the winner once the score for the final hole has been added. It will also add the match to match history

}

}

private int[] GenderEffect(Hole hole)

{

int[] ParsForEachG = new int[4];

for (int i = 0; i < 4; i++)

{

if (Golfers.ElementAt(i).Gender == Enums.Gender.Female)

{

ParsForEachG[i] = 1;

}

Since Gender plays an effect in calculating the score per golfer I need to access whether they are male or female and return a value based on that.

else

{

ParsForEachG[i] = 2;

}

}

return ParsForEachG;

}

private int[] StrokeIndexEffect(Hole hole)

This method returns the effect per each golfer based on gender and the stoke index of the hole.

{

int[] Difficulties = new int[4];

for (int i = 0; i < 4; i++)

{

if (Golfers.ElementAt(i).Gender == Enums.Gender.Female)

{

Difficulties[i] = 19 - hole.RedStrokeIndex;

}

else

{

Difficulties[i] = 19 - hole.StrokeIndex;

}

}

int[] Handicaps = new int[4];

for (int i = 0; i < 4; i++)

{

Handicaps[i] = Golfers.ElementAt(i).Handicap - Difficulties[i];

}

return Handicaps;

}

private int ModifiedPar(FourPlayerHoleScore hs, int handi, int intitialPar)

{

int FinalPar = 0;

Using all the values from the previous methods calulates the final effect of the par which will act on the score.

if (handi >= 1)

{

if (handi >= 18 & intitialPar == 2)

{

FinalPar = hs.Hole.Par + 2;

}

else if (handi >= 1 & handi < 18 & intitialPar == 2)

{

FinalPar = hs.Hole.Par + 1;

}

else if (handi >= 18 & intitialPar == 1)

{

FinalPar = hs.Hole.RedPar + 2;

}

else if (handi >= 1 & handi < 18 & intitialPar == 1)

{

FinalPar = hs.Hole.RedPar + 1;

}

}

return FinalPar;

}

private int FindScore(int Score, int Par)

{

int TotalScore = 0;

Using the par obtained from the other methods it calculates the final score.

if (Score - Par == 1)

{

TotalScore += 1;

}

else if (Score - Par == 0)

{

TotalScore += 2;

}

else if (Score - Par < 0)

{

TotalScore += ((Score- Par) - 2)\* (-1);

}

else

{

TotalScore += 0;

}

return TotalScore;

}

[NakedObjectsIgnore]

public void TotalScoreCalculated(Hole hole, int[] Scores, FourPlayerHoleScore hs, int[] handicaps, int[] ParsForEachG)

{

int[] FinalPar = new int[4];

int[] TotalScores = new int[4];

for (int i = 0; i < 4; i++)

{

FinalPar[i] = ModifiedPar(hs, handicaps[i], ParsForEachG[i]);

}

for (int i = 0; i < 4; i++)

{

TotalScores[i] += FindScore(Scores[i], FinalPar[i]);

}

}

[NakedObjectsIgnore]

Calls the other methods to obtain the final score for each Golfer

public int FindWinner()

{

int Gwin = 0;

for (int i = 0; i < 4; i++)

{

if (TotalScores.Min() == TotalScores[i])

{

Gwin = i;

Calculates which golfer has the lowest score and therefore is the winner.

}

}

return Gwin;

}

#### Authorization & Authentication

Authorization and authentication are the most important aspects of the project since the project is meant to be a multi user application if there was no authorization the project wouldn’t know who was playing and then you could change anyone’s user information. To obtain the golfer Identity I am using Auth0 which is very used as a login system. I have not created Auth0 I am just using it. Auth0 requires an email to login which then obtained in the Me() method is GolferServices. If the Email has not already been associated with a golfer then this acts an account creation and you need to create a golfer and fill out your details otherwise this is how you access you profile so you can see your invites and other properties.

Each object has an authorizer which contains two function. IsVisable() and IsEditable(). This means that a user might be able to see something like another golfers name but they can’t change that. Which allows for the capability of multi user.

The code for a authorizer is shown below:

public class ClubManagerAuthoriser : ITypeAuthorizer<ClubManager>

{

public bool IsEditable(IPrincipal principal, ClubManager manager, string memberName)

{

if (manager.Username == principal.Identity.Name)

{

return true;

Only you can edit your profile.

}

else

{

return true;

}

}

public bool IsVisible(IPrincipal principal, ClubManager manager, string memberName)

Prevents the user from seeing the AddMatchHistory method so that they cant fake their history

{

if (memberName == "AddMatchHistory")

{

return false;

}

else if (((manager.Friends.Count == 0) & (memberName == "Friends"))

| ((manager.Groups.Count == 0) & (memberName == "Groups")))

{

return false;

}

else if (manager.Username == principal.Identity.Name)

{

return true;

}

else

{

if ((memberName == "Mobile") | (memberName == "AddFriend") | (memberName == "AddCourseToFavourites"))

{

Prevents over users adding friends for you, a course to your favorites or your mobile.

return false;

}

else

{

return true;

}

}

}

}

}

As mentioned in the server all the authorizers had to be declared so the server doesn’t think they are separate classes.

The testing of my authorization will be shown in the Testing Section.

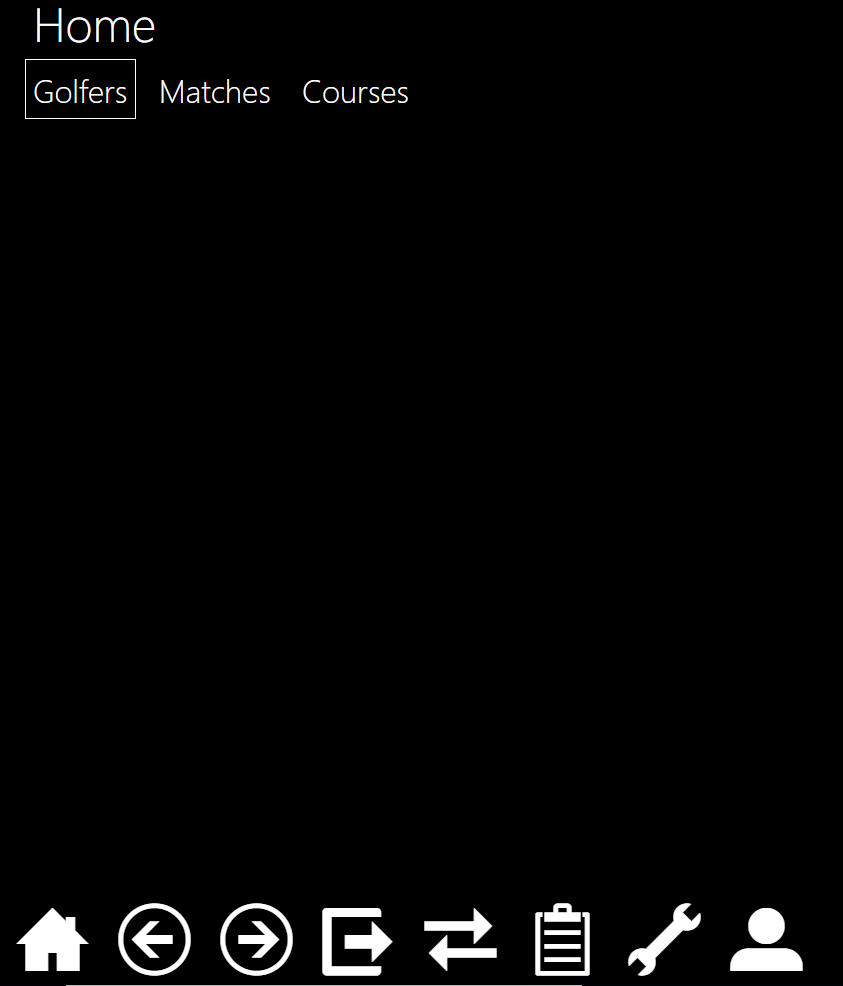
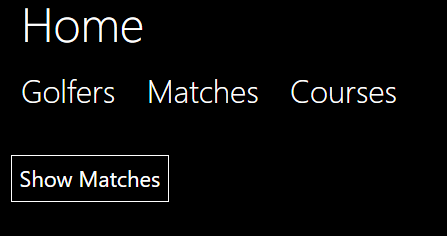
# Testing

Since my project is a multi-user application which runs on one device currently I will be using Microsoft Edge and Google Chrome to simulate two different users on two different devices.

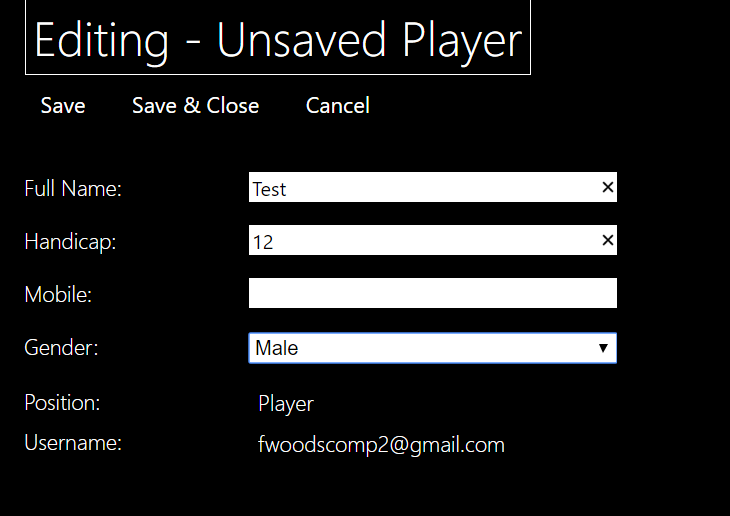
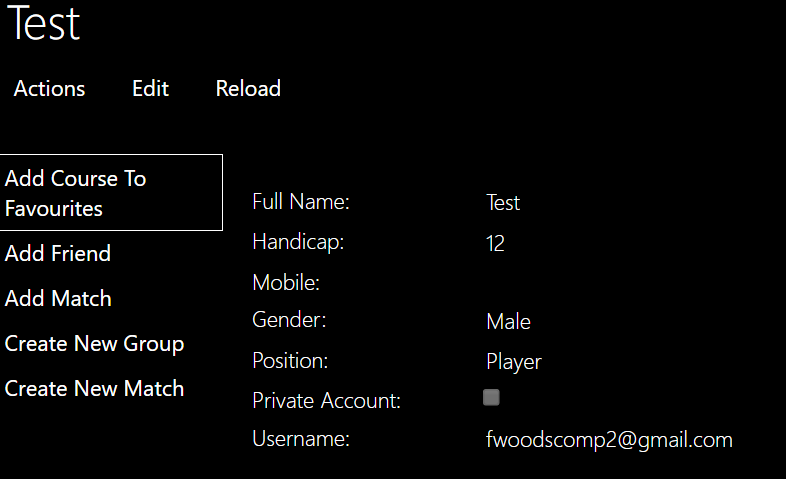
## Test 1 : Creating a New Account.

Creating a new account is a really simple process which involves two steps. The first step is logging in using the auth0. This can be done using a Facebook, Google or Auth0 account. This was mention in the Authorization section of the Technical Solution. The screen shot below shows the interface that meets the user after they have just login to Auth0.

### What the user can and can’t do once logged in for the first time.

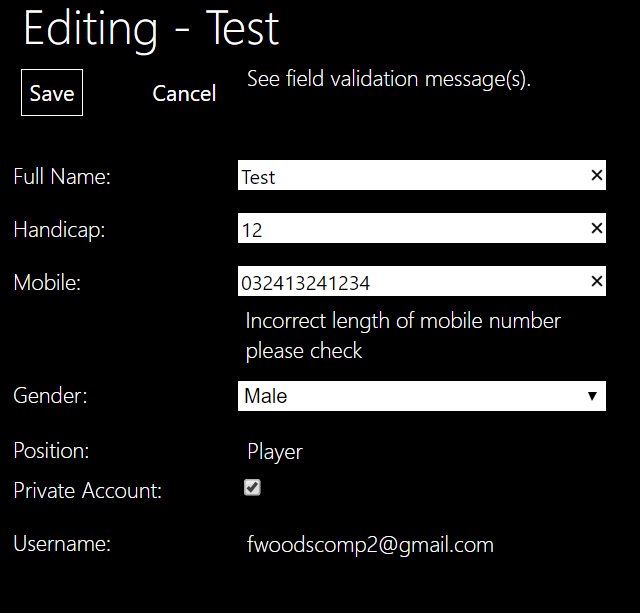
As a new account the user hasn’t currently created a golfer, this means that certain aspects of the application should not be accessible. An example of this is that the user shouldn’t be capable of creating a match. The new user shouldn’t be able to create a group join a group of send a message to a golfer. All of this methods have been hidden using authorization. As each of these methods and actions are on separate objects I have only shown the screen shot for not being able to create a new match.

### How to create an account.

To create a Golfer the new user must press “Golfers” and then press me upon pressing “Me” they are met with this screen shot. The “Position” and “Username” is not editable this so that only a systems manager can create a club manager or change their position. The username is used to identify a golfer and is taken from your login with Auth0 therefore it can’t be changed.

### Private Accounts

The new user then fills in all the details but has left mobile blank as it is an optional parameter. From this moment he can create a new match join groups add friends as seen in the expanded actions. The new user than decides to edit their self and add their phone number but they want their account to be private meaning that no-one can see their mobile. To do this they press edit and tick the box labelled private account.



When the user enters their phone number they make a mistake but due to one a validation complementary method added to Mobile the user is warned and can’t save the changes until the mobile is the correct length.

To show the private account is working another user logins and looks at the new user’s profile. The two screen shots below show the new user’s profile once with private account ticked and the other when the account is not private.



## Test 2: Enriching the User’s Profile

For the purposes of these test we shall be using a golfer who has been created in the DBInitializer. The golfer’s name is John Smith. The screen shot below shows the Golfer’s current profile before the tests have taken place.

### Adding Friends

#### Sending the friend request

John is a new users but some of his friends are already apart of the application so he decides to add them as a friend. To do this he goes on his profile via the Me method in Golfers.

#### Accepting the friend request

#### Declining the Invite

#### Error Catching

##### Already Being Friends

##### Adding the wrong Invitation

### Joining a Group

#### Creating a group

#### Requesting To join the group

#### Being sent an invitation to join a group

#### Sending a Group Message

### Sending Messages

#### Sending the Message

#### Responding to a message

## Test 3: Creating a Match

### Browsing Courses

### Sending The invitations

### Creating the match

### Too many players want to join

### Playing the first hole.

#### Inputting an invalid score for a golfer

#### Score calculation

##### Matchplay

##### Strokeplay

##### Stableford

## Test 4: Finishing a Match

# Evaluation

# Appendix

## All My Code

## References