**Main project Objectives**

* Stable working website
* Play a fantasy game
* Social media aspects to it
* User can register
* Ads can be placed on the site
* Instant messaging service
* Forum service
* Scores update using RSS feed
* Player score updated automatically on login
* Admin can log in to do maintenance

**Success Criteria**

* User should be able to register
* User should be able to select a team of 8 players
* User should be able to user instant messaging service to chat
* User can post on a forum
* Users points automatically updated each week
* User can chat with others near using a GPS system

**Management Deliverables:**

Feasibility Study and Project plan: A feasibility study will be conducted so that a decision on the viability of the project can be made. This should contain an outline of the project along with the benefits, requirements and alternatives if any.

Analysis and design document: This document will contain all the requirements and functions that the Application should have. This will contain our UML diagrams which will plot how the user interacts with the software and what extra functions algorithms we might need.

Source code: We will use the Analysis and Design document to build the code for our software. This file should contain all code, images and installation files.

Testing Document: All tests performed on the software will contained in this. Testing will done in all stages of the application and be documented for further record.

**Technical Deliverables:**

* A clear and easy way for a user to navigate through and complete the team selection
* User login to access an account or set up an account on first use.
* User should be able to change a player once a week
* Administrator login to change variables without interfering with ongoing game
* Database to store account details.
* Real time update of player scoring using RSS feeds
* Instant message style chat built in
* Forum messages used for discussion of games
* User able to join leagues and check overall standing

## **Development Methodology**

In this section we are going to discuss the methodology of software development life circle (SDLC) we chose to use to develop this project. After initial research we decided for Prototyping SDLC.

**Prototyping SDLC, what is it basically?**

In this approach the development team implements a ‘sample’ which have only very limited functionality of proposed project and show it to the customer. Customer provides the suggestions of improvements and finally the development team implements these. This circle then repeated itself numerous times until the project is fully functional, tested and accepted by customer. This approach differs from other methodologies by avoiding doing the ‘big design in advance’ followed by implementation, testing and deployment phase. The project is rather developed by mutating the prototype with numerous design, implementation testing and deployment phases until the final product is build ***[1]***.

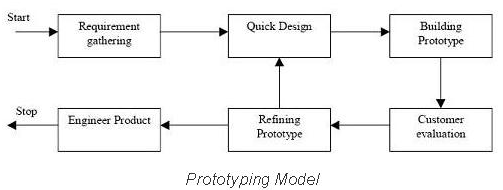


Figure 1 Prototyping SDLC work flow

**Why did we decided on this approach?**

We decided for prototyping approach for various reasons

* We wanted to avoid ‘big design in advance’. To design web application in advance it requires experience. Only experienced developers who already worked on projects with similar functionality and scope are able to do that. If ‘an amateur’ tries to design the application he will realize during the implementation phase that his design has flaws and redesign is required. This will lead to lose of work hours and any codebase the developer had already implemented. Basically you need an architect to design the house. We amateurs. We never build web application of this scope. With prototyping approach we be able to redesign the sample as many times as needed.
* We will learn with each prototype iteration. In our case we first build a simple but working sample build from prototypes. Data store prototype (relational database layer), data model prototype (objects to data mapping layer), controller prototype (business logic layer), and the view prototype (front end client layer). We make sure it all works together. Then we pick one proposed functionality and implement it whole way down through all the layers. And we learn from it. Implementation of next functionality will be easier and we get more productive over time. Hopefully after various iterations we will be able to call ourselves ‘the web developers’.
* Time restrictions. According to our project plan the first working prototype delivery is due to 14.February 2015. But we have been notified (by email 14.November 2014) that we must deliver the ‘working prototype’ due to 19. December 2014. According to our plan this is a two months ahead. Being still in research phase of project plan we assessed that only feasible approach under these circumstances is prototyping SLDC.

## **Project Plan**

*In this section we are going to discuss how we planned to develop this project.*

**What are we developing?**

Fantasy hurling project is basically fantasy sport game. It’s rich web application not a static web page. As a player manages his fantasy hurling team a content dynamically changes. At the same time it must have a persistence capabilities. Basically a changes player made must be stored somehow and not get lost when player finishes to play. Data must outlive the application runtime.

*Basically we will need to have three main layers.*

* Front-End (code running in browser delivered by initial http request from web server)
* Back-End (code running on server listening and answering to http requests)
* Persistence Storage (relational database server listening and answering CRUD queries)

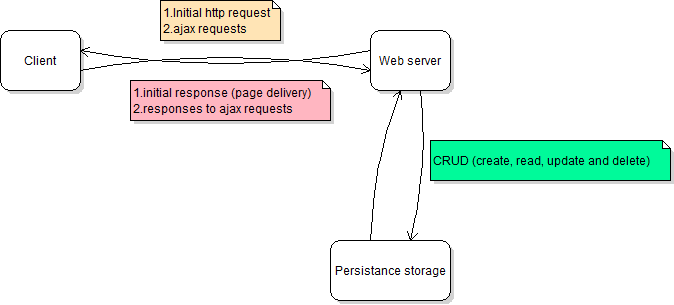


Figure 2 Web Application general design

## **Work Flow Plan**

* Due to change of SDLC methodology all DEVELOPMENT and QUALITY ASSURANCE items are merged into PROTOTYPING ITERATIONS item.
* All DESIGN items are reduced to PROTOTYPE DESIGNS (user interface prototype design etc.)

**Individual Task Assignment?**

**David Kelly**

* Front-End design ***responsibility*** (wire frames, photo shop, HTML5, CSS3)
* Documentation involvement
* Testing involvement

**Michael James**

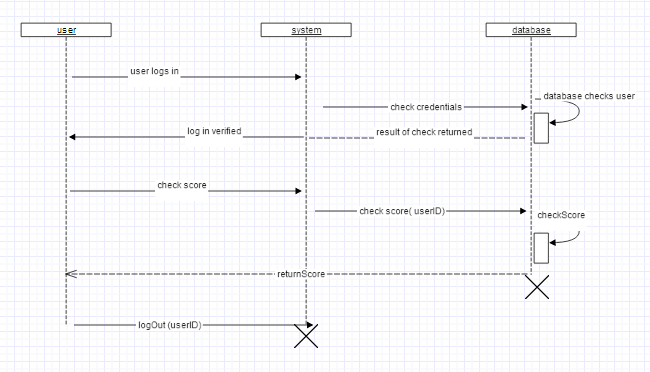
* Front-End development ***responsibility*** (HTML5, CSS3, vanilla JavaScript, JQuery, JQuery UI)
* Documentation involvement
* Testing involvement

**Martin Zuber**

* Back-End ***responsibility***(MySQL database, .NET Web API 2, Entity Framework ORM mapper, Azure Deployment)
* Documentation involvement
* Testing involvement

## **Sequence Diagrams for Use Cases**

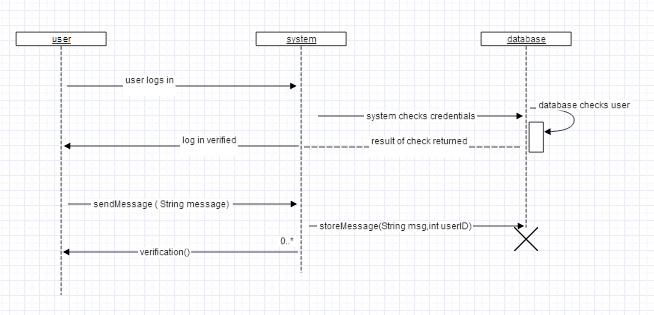
### User checks score



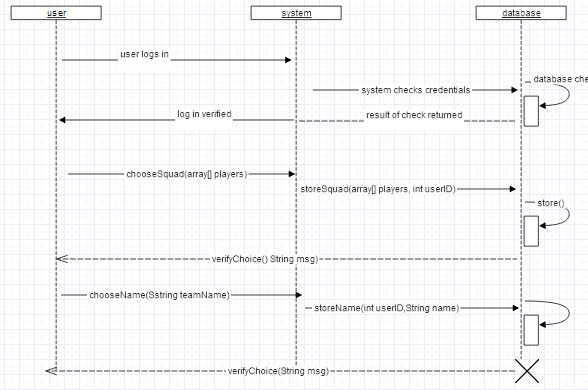
### User checks fixtures



### User posts message



### User chooses team

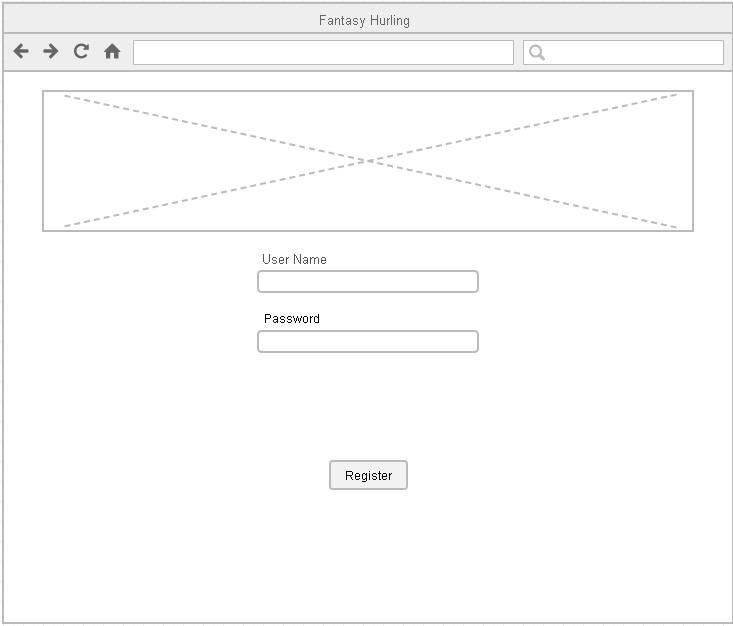


### User makes a transfer



## **Wire framing**

### Login Page



On this page the user can log into the system. They enter a username and password and are logged in. There is an image also.

Elements needed:

* 2 Text boxes
* One button
* One image

### Registration Page

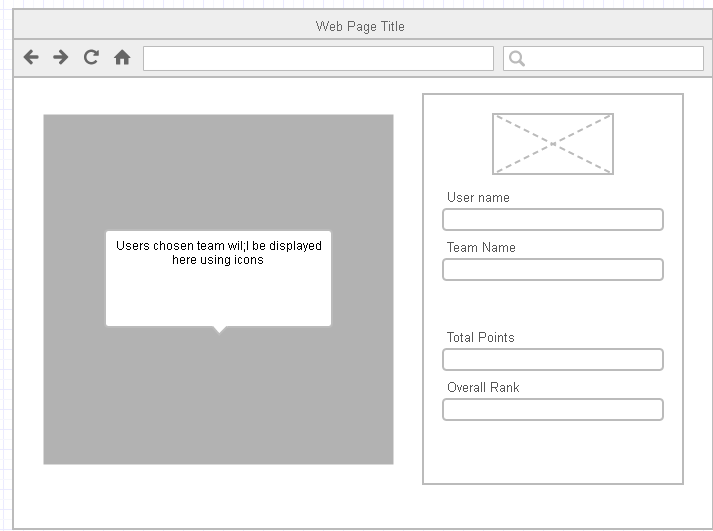


On this page the user can register if they have not already.

Elements needed:

* 5 Text boxes
* One button
* One image

### Team statistics page



On this page the user can view their team’s current standing and overall points. It shows the user name and team name and the team the user has selected on the left

Elements needed:

* One image
* Team box made up of images and text
* Some basic Divs to show some information based on the user

### Player transfer page

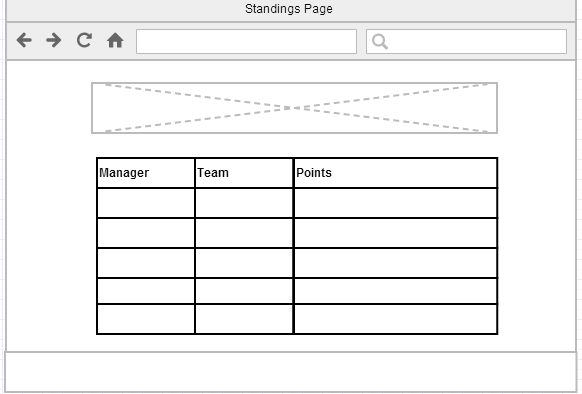


On this page the user can make transfers if they want to. They can search through the database of players, remove a player from their team, and see player score and value. Nothing is confirmed till the user clicks the add player button.

Elements needed:

* One combo box
* Basic Div. to show stats about the player
* One image to show player picture
* One button to confirm change
* A box on the left with all the players selectable and removable from the players team

### Standings Page

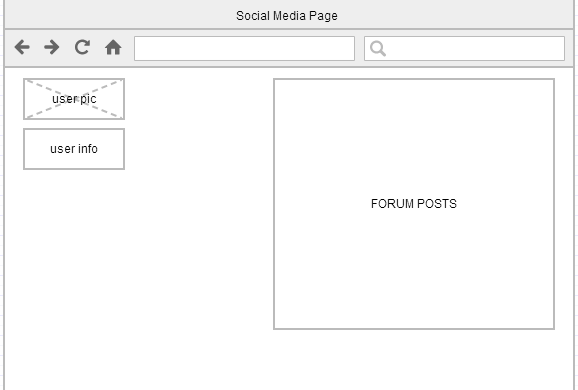


On this page the player can view the top players in the game by points scored. He may be in this list, it will be a simple SQL query displayed in a HTML table.

Elements Needed:

* HTML table
* One image
* Header and footer ECT

### Social Media area

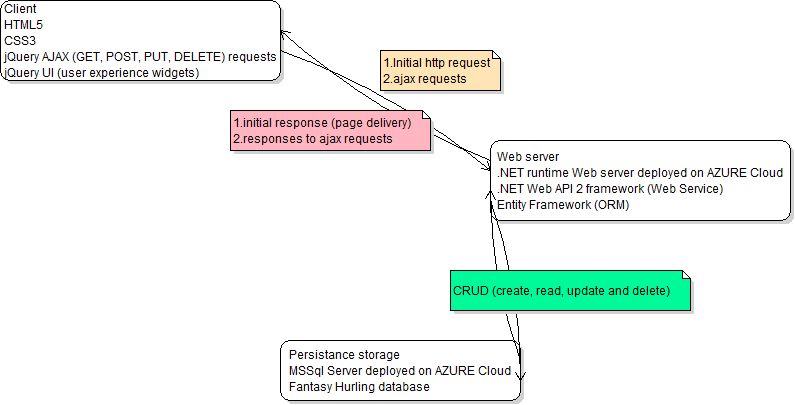


On this main social media page the user can view all recent forum posts and has some information about their profile displayed also, when the user clicks a post, which will be displayed as a title with a link they are brought to a separate page with that post and all replies in it. Here the user can reply to the post also.



Elements needed:

* One image
* One text area
* An area to store forum posts
* A button to post the message



*Figure 3 Web Application in detail*

**GIT and Github**

We are developing this application using Git distributed revision control system.

Fantasy hurling upstream repository is available at:

<https://github.com/Michaelcj10/Fantasy_Hurling>

Fantasy hurling repository clone is maintained and available at:

<https://github.com/zubidlo/group_project>

Web API prototype upstream repository is available at:

<https://github.com/zubidlo/FanHurApi>

These two repositories are going to get merged in later state of development.

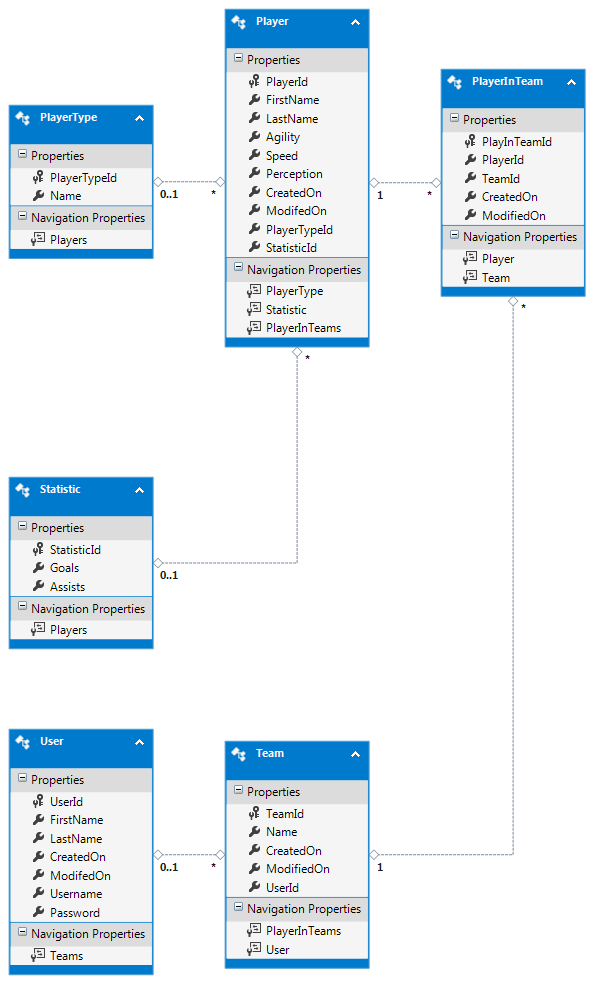
**Front-End**

First front end prototype is simple web page using HTML5 ***[2]*** and CSS3 ***[3].***AJAX ***[4]*** request are made through simple web forms using JQuery ***[5].*** In prototyping iteration this web page will gradually turned to user friendly application using JQuery UI ***[6]*** widgets and methods.

Front-End development will be done using IDE. Some recommended options are Sublime Text 2 ***[7]*** or Aptana Studio ***[8].***

**Data Store**

We store Fantasy Hurling data in relational database tables. Initial prototype is design to mimic football like sport team and player statistics and it will gradually mutate into final ‘hurling’ state during prototyping iterations.



*Figure 4 Fantasy Hurling database prototype*

Database will be deployed on Azure Data Store ***[9].*** At the end of development data store gets migrated to Azure Europe North Datacenter located in Dublin ***[10].***

**Back End**

Extensive research was done on web application back end development and we decided to build Web API Service ***[11].*** Initial idea was to use Node.js framework ***[12]*** to build the API, but after additional research a decision was made to use ASP.NET Web API ***[13]*** technology. Some of the reasons for this change include:

* Complexity of task versus maturity of Node.js technology
* Potential lack of documentation if a problem in code arises
* Lack of quality connectors to relational databases, Mongo DB (json based object database) is preferred with Node.js
* Additional frameworks to learn building Web API (express framework and more)
* Visual Studio 2013 Community edition free for use now
* 10 web servers free on Azure Cloud with Visual Studio installation
* Visual Studio fully implements Azure storage and web server deployment (on click in a menu and application and database is up on cloud)
* Comprehensive documentation and tutorials for all required frameworks and technologies.
* Technical support and huge community of developers if a coding problem arise.

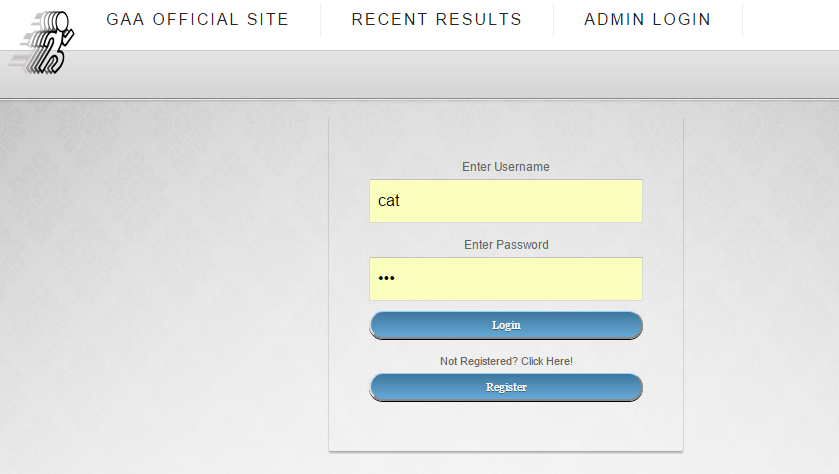
I must add here that we expect a usage of additional tools and technologies during the project development as a need arises.

## **Implementation**

**Front end**

There were some key elements to creating the front end. A series of simple but cleanly designed web pages were built. JQuery UI widgets were used, and JavaScript was used to connect the front end to the back end database. This was done using an API that was built by one of the team members in C#. We won’t go into this now as it will be explained in detail later. Navigation on the site was made as simple as possible and there were help icons on each page to explain how the user could interact with the system. We will look at the site in detail now and fragments of code will be shown and explained.

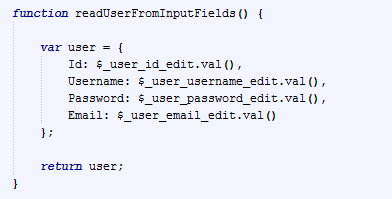
**Login and Register**



The user needed a password and username to login and interact with their team. The login page was kept minimal and straightforward to understand. The following is an example of the method that would log the user in if that username and password was correct. A URL was specified with the user ID the user had entered in the text field as their own and an AJAX request retrieved that users details if he existed. If he did exist, it retrieved the password associated with that account and checked it against the password the user entered, if it matched, the user was logged in, if not, a message prompted the user that he needed to re-enter the details.



If the user did not have a login, he could register to create one. He had to enter a username, password and a valid email address. The API would not allow duplicate usernames so long as it did not already exist, the account was created. A simple function was needed to add a new user to the account. The information to put into the database via the AJAX POST request was retrieved and stored in an array. This information was what the user had entered In the HTML fields.



The function to add the user into the database via the API was a simple POST request using the JSON data.



**Session Storage**

With local storage, web applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Local storage is per domain. All pages, from one domain, can store and access the same data.

Session storage was used largely in this application. The bonus to the session storage was that it got rid of the need to pass variables between pages in the style of PHP which made accessing variables such as the username very easy.

Example of setting is below, we set our current user ID to be called “id” in session storage. We could not access this any time we wanted across the entire site with a very simply command.

Set session storage

sessionStorage.setItem ("id", id);

Get session storage value

Var user = sessionStorage.getItem ("id");

Session storage could be cleared at the end of the session using a simple command.

sessionStorage.clear ();

The user’s players in his team were stored in this manner also which allowed them being interacted with very easily. For example when the players in the users team were being retrieved each player was assigned a session variable of the data retrieved for player position, so the first player out was called “1” as his player position is 1. The value for this name of 1, was his player ID value. The entire team could be stored this way in session storage and then accessed at any time.

sessionStorage.setItem ("" + object.PositionId, "" + object.Id);

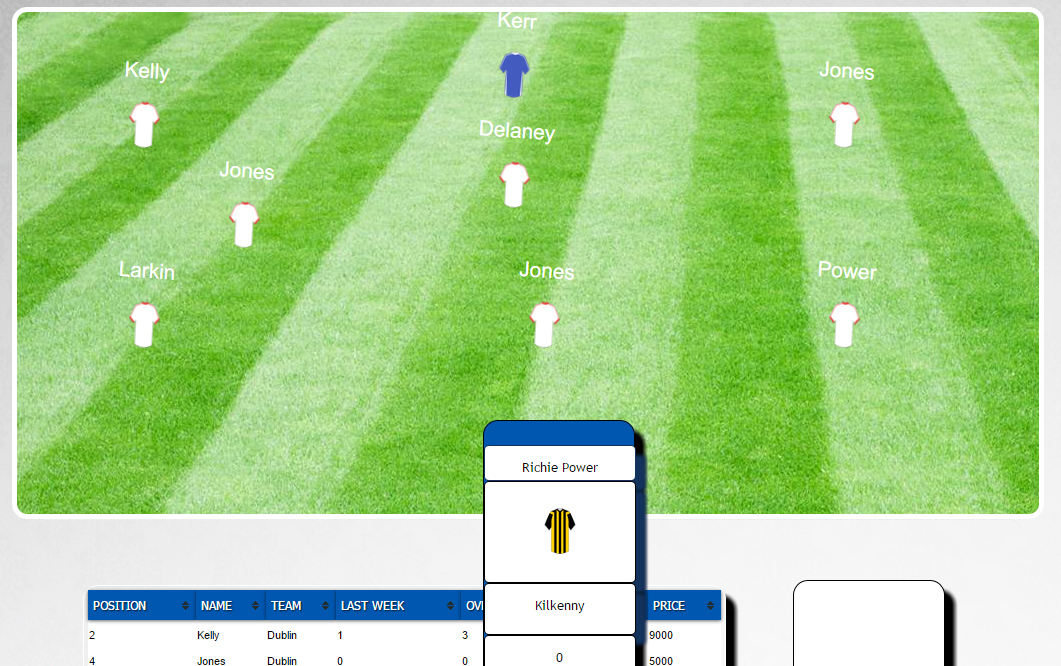
**Creating a new Team for the user**

When the user navigates to either the view team page or the transfer’s page, the application checks if a team exists with that user ID. If the team exists then nothing extra happens and the application acts as normal. If it does not exist, a new team is created for the user. The name is simply the username and the user ID combined. The type of request to do this is a POST request. The team ID is then stored in session storage.

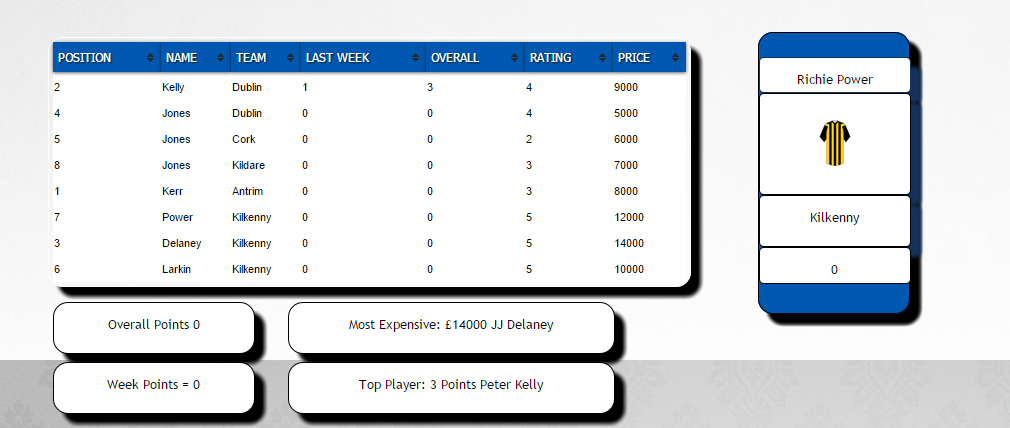


**How to view the team**

The user could view his team using the team view page. In here there was a graphical representation of his team using icons which were movable. Each time a user hovered over a player a baseball style card would pop up showing details about that player.

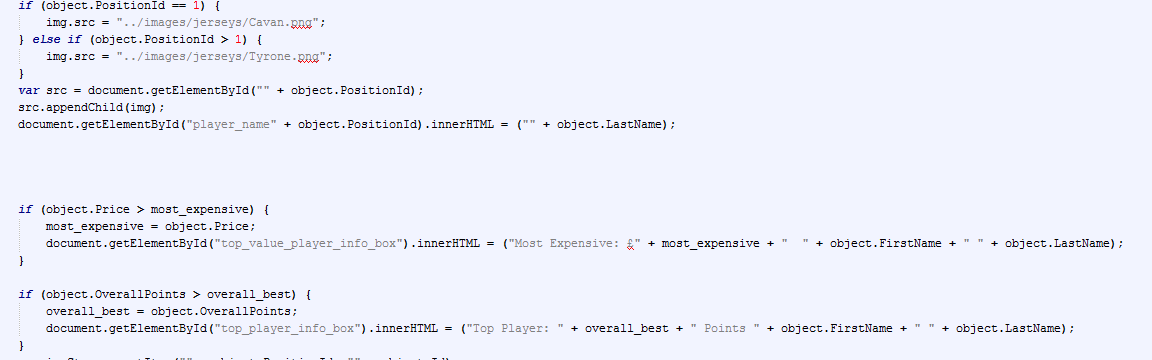


More detailed stats about the team were also shown in the form of a sortable table and some widgets showing information such as the top scorer or most expensive player.



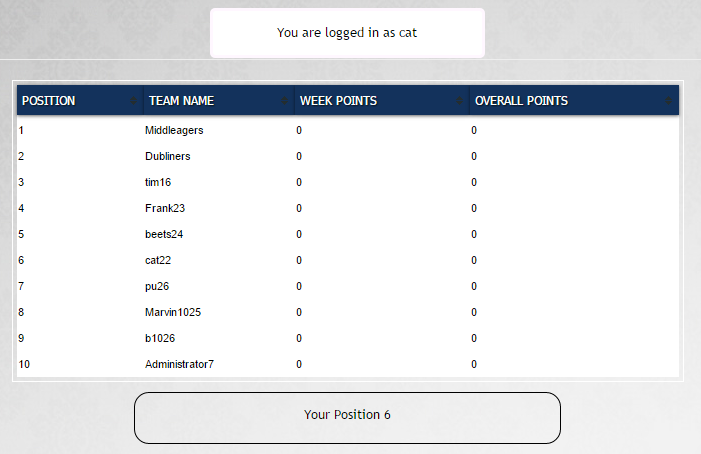
All this information was retrieved using the API and a simple GET request using the appropriate URL. Below is an example of how the full team is retrieved. As can be seen simple local variables were used to decide which the most valuable player was and so on, each iteration the player was checked against the current most expensive player, if he was worth more, he was set as the most expensive. When the request was finished, we then had the most expensive player, top player and so on in variables which were used to update the inner HTML of the DIV elements in the HTML page.





**Viewing the standings**

The user could view his overall position and the top 10 teams in the game in the standings page.

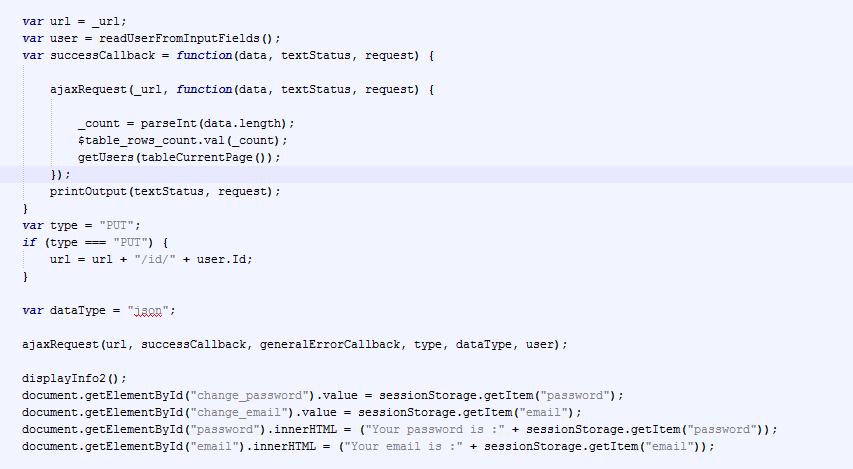


The simple GET request returned the top 10 users based on overall points. A second request was done to retrieve all users, a counter variable was incremented each time a user was found, when the user found matched the current user ID stored in session storage, that counter contained the current users overall position out of all users. Below is an example of the function to retrieve the top 10 users in the database sorted by the highest value for the user’s points.

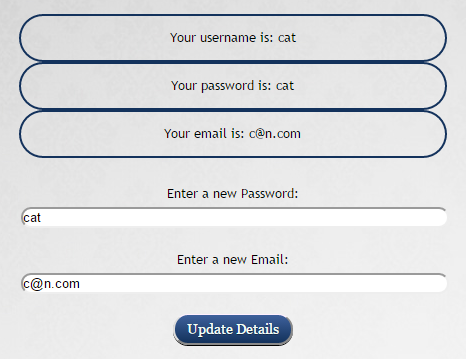


**Changing user details**

A page was created to allow the user change his login details if they wished. The request was made in the usual way, a PUT request was used this time however.

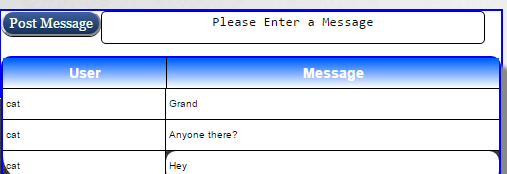


The page was kept simple with HTML DIV elements showing the current users details and text fields allowing the user to enter new details. On click of the button the request created a new JSON array of data, and PUT this data into the user’s details via an AJAX request. Below is an example of the HTML styling in the displayed DIV containing the current details and the fields and button required to change these details.



**Instant messaging in the application**

Social media was integrated into the application in the form of instant messaging on each page. Every functional page on the website has a small neat DIV containing the last 3 posted messages and a text area and button to allow the user post a new message.



The information is retrieved using a GET request in an AJAX function. Only the 3 newest messages are retrieved keeping the feeling of the application being in real time, and keeping the table itself small and efficient. An example of the code can be seen below.



**Making a transfer**

The transfer’s page is the most complex. On initially entering the page a function loads the user’s players. It knows what players to load based off the team ID which was set to session storage earlier. These players are stored in JQuery UI widgets which are resizable and moveable. An example of this can be seen below.



## **Deleting a player from the team**

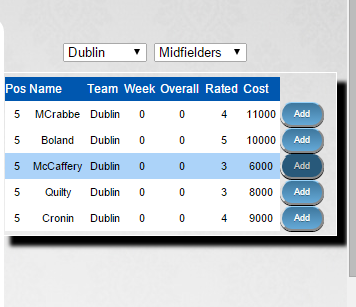
To delete a player, the user clicks on his icon. On click fires a function with the parameter of player position given. We now have the player’s position. We then run a new function iterating through the users team looking for the player who has a position equal to what we passed through originally. When we find it, we take his player ID and pass it to another function. This function removes the player with the ID we pass through from the team with the ID we have stored in session storage! Easy!

We can see below an example of the function to delete a player. It is of course another AJAX request and is of type DELETE. Once the delete function finishes we reload the page after a delay of 600 milliseconds which will reload the entire team again, which of course no longer includes the player we deleted.



**Adding a player to the team**

To add a player to the team, a user simply uses the table on the right to make searches based on team, when he finds a player he wishes to add, he simply clicks the button to add the player.



On click of the add button, a function is fired that retrieves the player ID. Another function is run then to actually add the player. It is of type PUT. If the user has enough budget and the position is vacant the player is added to the team.



## **Dealing with if a user has not enough players in his team**

## One issue that became apparent early on is that if a user creates his team and adds some players, he may not add in the full complement of 8. This should not be allowed. We came up with a simple solution, when the user tries to do anything on the site a check is done, essentially an AJAX request retrieves all the players from the user team and counts them with a simple local variable. If there is not 8 players on the team, then he is redirected to the transfer’s page and he must add into his team additional players.

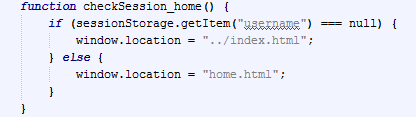
**Creating the user team**

When the user registers, he now has a username and password. At this stage he has no team created in the database. There were a number of solutions to this, but the simplest was to simply do a check when the user tried to view his team or make a transfer if he actually had a team. The user ID is passed into the URL that retrieves all teams, and if the AJAX request retrieves nothing, the user must have no team and one is automatically created for him. It is given a name derived from his username and user ID. We can see below the code to input the new team and the fields it requires in the function below that again.

## 

## **Keeping track of the session**

Another issue was making sure that if the session was ended, that if the user tried to navigate back to that page, he would be brought back to the login page. Also the user should not ever be allowed simply type in for instance the URL for the transfer’s page directly into the browser and be brought there without logging in. A simple way to do this was to use the session storage variables. The first thing that happened on all the pages was a check was done to see if there was a user ID in session storage, if there was not, then the user was re-directed to the login page. This avoided any potential glitches. We can see below a user tries to navigate to the home page, if username is not there then he is brought to the login page again automatically.

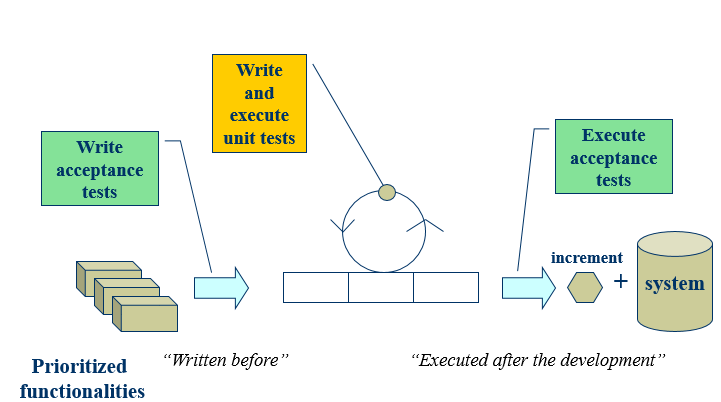


## **Testing**

Testing is the most important part of any software project. Eradicating errors and bugs is vital to the user experience. We came up with some main components to our testing phase.

* Acceptance Testing – this checks if the overall system is functioning as required.
* Unit testing – this is basically testing of a single function, procedure, class.
* Integration testing – this checks that units tested in isolation work properly when put together.
* System testing – here the emphasis is to ensure that the whole system can cope with real data, monitor system performance, test the system’s error handling and recovery routines.
* Regression Testing – this checks that the system preserves its functionality after maintenance and/or evolution tasks.

This testing phase is highly iterative and can be very effective in creating a bug free user experience.

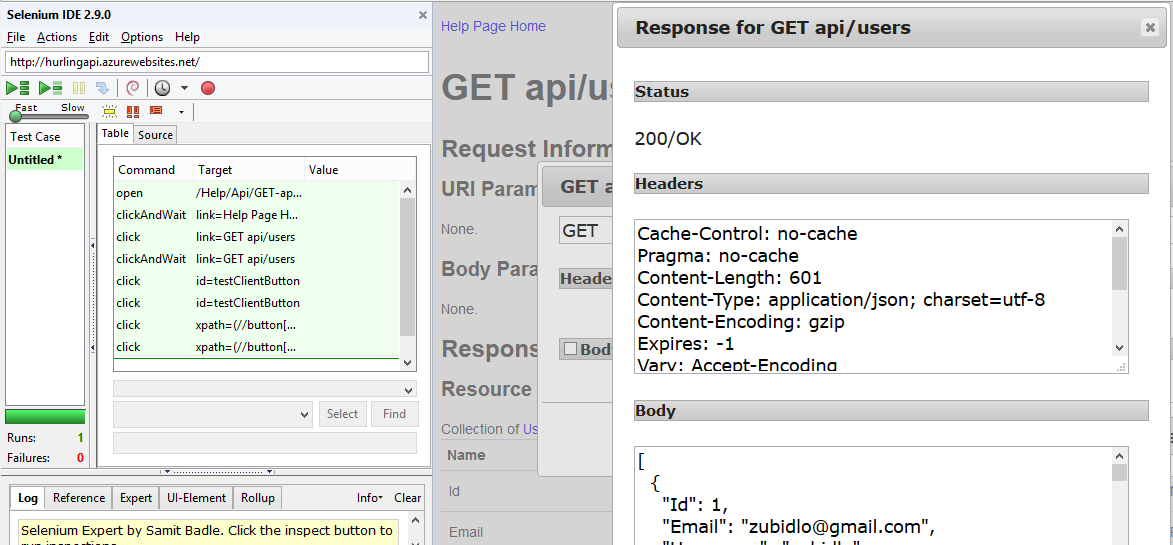
****

Testing the API

We tested the API using Selenium IDE for Firefox. Selenium is an automated web testing tool that allows the user to navigate and interact with a website, and have Selenium track those navigations and changes. You can then choose to let Selenium re-run these while looking for any inconsistencies or errors. The first step was to install the Selenium IDE for Firefox. We then began to test the API help page. This page essentially interacts with the API in the backend and simulates to query and retrieval of the JSON data. Any errors would show up in selenium.

**Test 1**

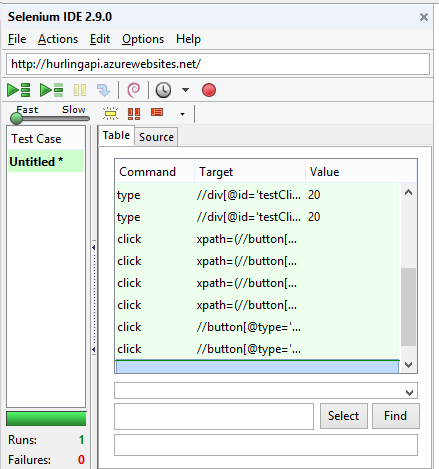
The first test was to try and retrieve all users from the API. This was done by simply setting the URL in Selenium to the main help page URL. Then we navigated to the users and performed the query as normal. The query was then re-run in Selenium.



We can see that the test passed successfully. This means the navigation and retrieval of the JSON data from the API was error free.

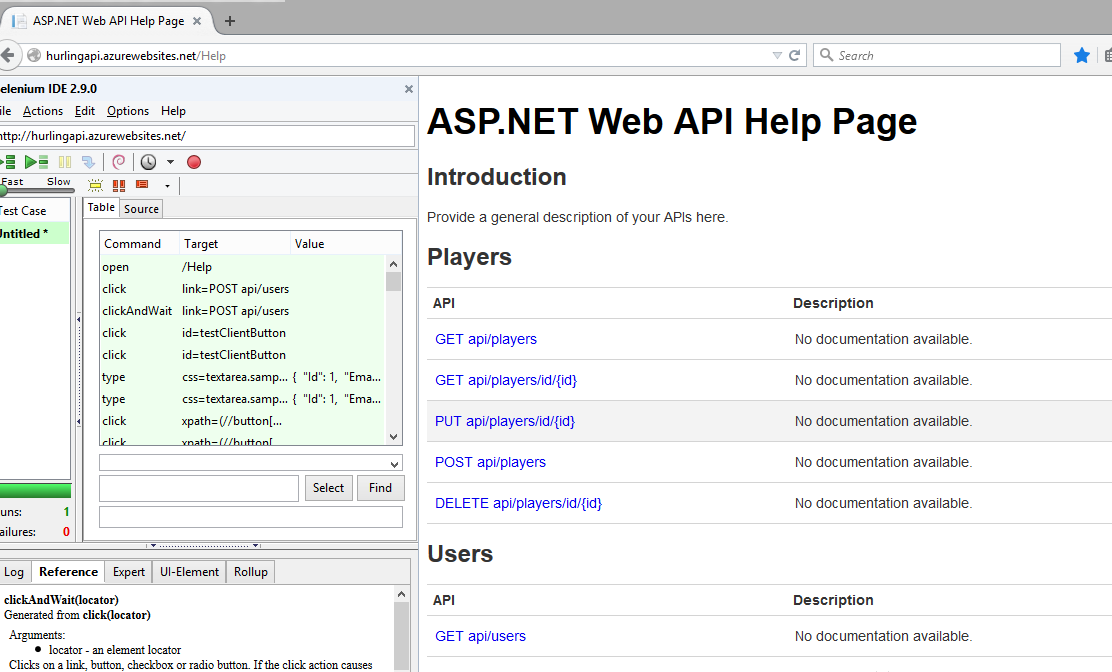
**Test 2**

The second test was to input a player into a specified user’s team via the API. We simulated the deleting of player with ID 20 from the team with ID 2. We then deleted the player so that when Selenium ran, it would not get an error trying to re-add a player that is already in a team! As we can see the test was successful and we received no errors.



**Test 3**

The final test we ran on the API was a little more complex. It involved adding a new user, creating a team for him, then adding a player to that team. We created a new user, then a new team, and added player with ID 1 to that team, who’s ID was 1061. The user ID was 1030. The test ran successfully and we encountered no errors.



**Testing the application Website**

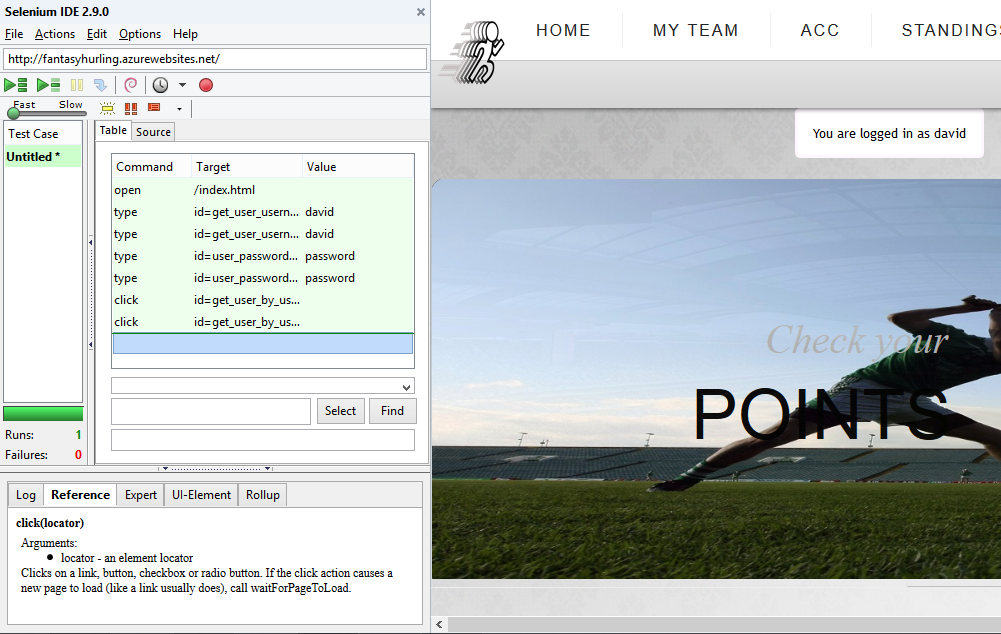
In regards the testing of the site of the site itself, there were a few factors deemed as very important and were considered to be so vital as even a small failure was unacceptable. These were as follows:

* The login of the user should be flawless
* Registration should be flawless
* Changing user details
* Deleting a player
* Adding a player

Due to the high level of importance for all of these interactions with the site, we decided to perform Selenium tests on all 5 scenarios.

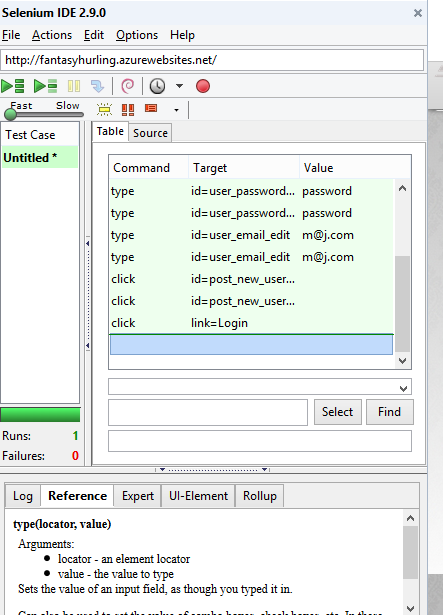
**Test 1**

The first test was to see if the user could login properly and without an errors. We ran the Selenium test and we can see from the results below the test was successful with no errors:



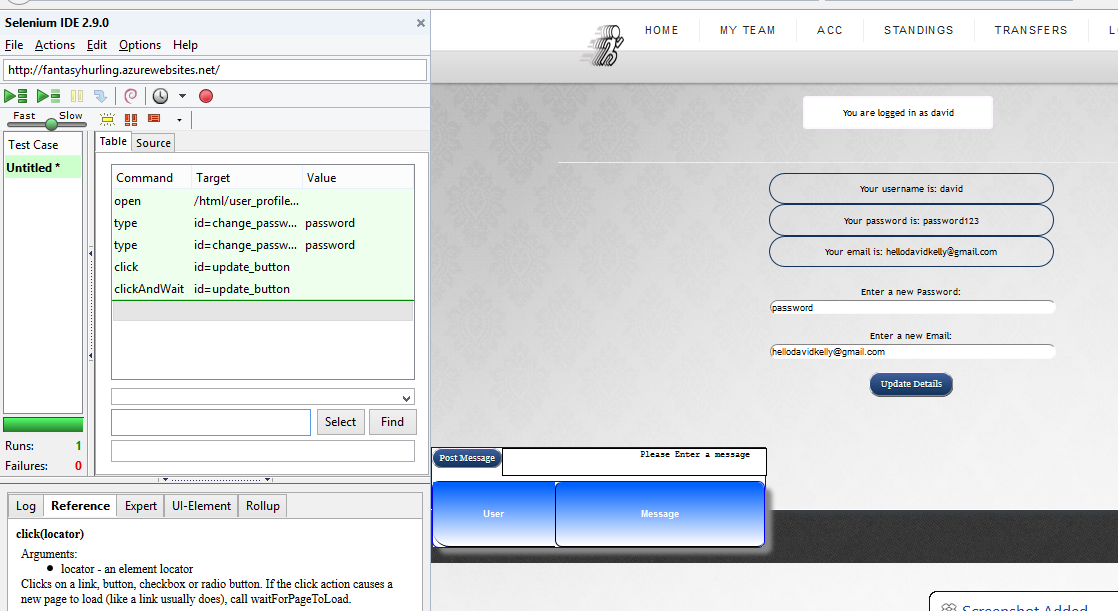
**Test 2**

The second test we ran was to make sure the user could register with no issues. Again we ran the selenium test and as expected there were no problems that needed to be resolved. We can see the output of the successful test below:



**Test 3**

The next test was to see if the user could change their details without issue. We ran the Selenium test and the results were again as expected, the test was successful with no errors. We can see the output below.



**Test 4**

We next ran a test to see if the user could delete a player from his team. The results were expected to be successful with no failures and this was indeed the case.



**Test 5**

The last test was to see if we could add a player into the team, the results were as expected and with no failures. We removed player with position 2 and the re-added him capturing the input with Selenium. We then stopped the capture, removed him and used Selenium to try automate his addition to the team again. It worked with no issues.



## **Conclusion**

We did not choose a trivial project. Being unexperienced developers we didn’t foresee the complexity and challenges involved in developing a fantasy sport web application. After realizing the scope of the project we changed our initial ideas how to approach this project development. We decided to follow prototyping methodology implementing and test one functionality at the time. Hoping to learn from each iteration and deliver a modern optimized rich user experience application.

In this occasion we decided to use the most modern technologies and designs. We are building a Web API Service so that different clients could get access to CRUD methods. This way the additional clients can get developed easily. For example in the future we can easily implement Android client or Windows 8 metro client for our fantasy hurling application. We are using Entity Framework ***[14]*** to automatically map database table rows to entity objects so we don’t need to write one sql query. We are implementing our Web API using .NET Web API 2 framework which is powerful toolkit capable of asynchronous responses, cross-origin requests and much more. We are deploying the application up on Microsoft Azure Cloud.

This project is an opportunity for us to learn new technologies and get valuable firsthand experience and skills in web development.

## **References**

1. What is Prototype model- advantages, disadvantages and when to use it? 2014. What is Prototype model- advantages, disadvantages and when to use it? [ONLINE] Available at: <http://istqbexamcertification.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/>. [Accessed 14 December 2014].
2. HTML5. 2014. HTML5. [ONLINE] Available at: <http://www.w3.org/TR/html5/>. [Accessed 14 December 2014].
3. Introduction to CSS3. 2014. Introduction to CSS3. [ONLINE] Available at:<http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/>. [Accessed 14 December 2014].
4. Ajax | MDN. 2014. Ajax | MDN. [ONLINE] Available at: http. [Accessed 14 December 2014].
5. JQuery. 2014. JQuery. [ONLINE] Available at: <http://jquery.com/>. [Accessed 14 December 2014].
6. Menu | JQuery UI. 2014. Menu | JQuery UI. [ONLINE] Available at:<http://jqueryui.com/m/>. [Accessed 14 December 2014].
7. Sublime Text: The text editor you'll fall in love with. 2014. Sublime Text: The text editor you'll fall in love with. [ONLINE] Available at:<http://www.sublimetext.com/>. [Accessed 14 December 2014].
8. Aptana. 2014. Aptana. [ONLINE] Available at: <http://www.aptana.com/>. [Accessed 14 December 2014].
9. Storage Documentation | Azure. 2014. Storage Documentation | Azure. [ONLINE] Available at: <http://azure.microsoft.com/en-us/documentation/services/storage/>. [Accessed 14 December 2014].
10. Azure Regions. 2014. Azure Regions. [ONLINE] Available at:<http://azure.microsoft.com/en-us/regions/>. [Accessed 14 December 2014].
11. Web API - Wikipedia, the free encyclopedia. 2014. Web API - Wikipedia, the free encyclopedia. [ONLINE] Available at:<http://en.wikipedia.org/wiki/Web_API>. [Accessed 14 December 2014].
12. node.js. 2014. node.js. [ONLINE] Available at: <http://nodejs.org/>. [Accessed 14 December 2014].
13. ASP.NET Web API | the ASP.NET Site. 2014. ASP.NET Web API | the ASP.NET Site. [ONLINE] Available at: <http://www.asp.net/web-api>. [Accessed 14 December 2014].
14. Cross-origin resource sharing - Wikipedia, the free encyclopedia. [ONLINE] Available at: <http://en.wikipedia.org/wiki/Cross-origin_resource_sharing>. [Accessed 14 December 2014].
15. Entity Framework. 2014. Entity Framework. [ONLINE] Available at:<http://msdn.microsoft.com/en-ie/data/ef.aspx>. [Accessed 14 December 2014].