

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of **Honours B.Sc. in Computer Science** in the Institute of Technology Blanchardstown, is entirely our own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Acknowledgment

In performing our assignment, we had to take the help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this assignment gives us much Pleasure. We would like to show our gratitude to our project supervisor Dr Luke Raeside, Institute of Technology, Blanchardstown for giving us a good guideline for the thesis throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in writing this thesis.

Abstract

The goal of the project was to create an interactive and enjoyable fantasy hurling style application. It was to be for Hurling, but would be flexible in that the backend could easily be used for a different sport if and when it was needed. This made the project very modular. It was designed to be very intuitive and easy to use.

Some of the key features were the ability for the user to be able to create and maintain a fantasy hurling team, receive hurling news dynamically and interact with other users via an instant messaging style forum.

The result was an application that was very simplistic in its front end design to allow novice users and experienced users alike to interact with all aspects of the application with ease. Administration of the application was simple using the administrator login to update and maintain all tables in the system. The application was very modular and could be expanded or used for a different sport very easily. It also tapped into the current phenomena of social media in its instant messaging style chat.

Table of figures

Figure 1 Prototyping SDLC	5
Figure 2 Gant Chart	9
Figure 3 IBM Analytics Benchmark.....	11
Figure 4 Desktop Browsers.....	12
Figure 5 Mobile Browsers	12
Figure 6 Console Browsers.....	13
Figure 7 Consumption.....	20
Figure 8 Website Traffic	22
Figure 9 Web Application general design	30
Figure 10 Fantasy Hurling Web Application in detail.....	32
Figure 11 Use Case: User checks current score.....	33
Figure 12 Use Case: User checks fixtures	34
Figure 13 Use Case: User uses social media aspect	35
Figure 14 Use Case: User creates team.....	36
Figure 15 Use Case: User makes transfer	37
Figure 16 Sequence: User checks score.....	38
Figure 17 Sequence: User checks fixtures	38
Figure 18 Sequence: User posts message	39
Figure 19 Sequence: User chooses team.....	39
Figure 20 Sequence: User makes a transfer	40
Figure 21 User Interface: Login Page	41
Figure 22 User Interface: Registration Page	42
Figure 23 User Interface: Team statistics page.....	43
Figure 24 User Interface: Player transfer page	44
Figure 25 User Interface: Standings Page.....	45
Figure 26 User Interface: Social Media area 1	46
Figure 27 User Interface: Social Media area 2	46
Figure 28 Fantasy Hurling database initial prototype.....	47
Figure 29 URLs to resources	52
Figure 30 JQuery Ajax Request Method	53
Figure 31 HulingAPI SQL server Azure	54

Figure 32 HurlingAPI Web server Azure	54
Figure 33 CORS enabled for api/teams route	55
Figure 34 Fantasy Hurling Database Schema	56
Figure 35 Admin Page for Teams	57
Figure 36 ConflictActionResult	58
Figure 37 Conflict Result in code and action.....	58
Figure 38 asynchronous method example.....	59
Figure 39 HurlingApi OData support	59
Figure 40 fantasyhurling Web Server dashboard Azure.....	60
Figure 41 Test API client in action	61
Figure 42 Login Dialog.....	62
Figure 43 get user by username request.....	63
Figure 44 register Form	63
Figure 45 add a user request	64
Figure 46 insert a new team request	65
Figure 47 Player details	66
Figure 48 Team details.....	66
Figure 49 Team by user id request.....	67
Figure 50 Team Table implementation.....	67
Figure 51 User Standings.....	68
Figure 52 Users sorted by overall points request.....	68
Figure 53 Update user request	69
Figure 54 Update User Dialog	69
Figure 55 Chat Board.....	70
Figure 56 Messages sorted by creation time request	70
Figure 57 making a transfer	71
Figure 58 remove a player from a team request	72
Figure 59 Adding Player.....	72
Figure 60 add a player to a team request	73
Figure 61 insert a new team request	74
Figure 62 session storage 2	74
Figure 63 round collar jersey design.....	76
Figure 64 V-neck collar jersey design	76

Figure 65 GAA Carlow hurling jersey manufactured by O’Neills.....	77
Figure 66 Fantasy hurling website representation of GAA Carlow hurling jersey	77
Figure 67 Fantasy hurling transfers page background for player positions.....	78
Figure 68 Testing Flow	81
Figure 69 GetPositionById (int id) method	82
Figure 70 HurlingApiTests Visual Studio Project	83
Figure 71 PositionController constructors	83
Figure 72 IRepository interface	84
Figure 73 IEntity interface	84
Figure 74 TestRepository implementation.....	85
Figure 75 GetPositions () method	85
Figure 76 GetAllAsync () method	85
Figure 77 GetPositions_ShouldReturnAllPositions () unit test method	86
Figure 78 unit test results	86
Figure 79 API get all users test	87
Figure 80 API insert player in team test	88
Figure 81 API insert player to team test 2	89
Figure 82 Login test	90
Figure 83 User Registration test	90
Figure 84 User Details test.....	91
Figure 85 remove a player from team test	91
Figure 86 Add player to team test.....	92
Figure 87 Repository Models Classes.....	145
Figure 88 Model Classes.....	148
Figure 89 Controller Classes.....	157

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 AIM AND OBJECTIVES	2
1.3 MAIN RESEARCH QUESTIONS	3
1.4 JUSTIFICATION AND BENEFITS.....	3
1.5 FEASIBILITY.....	4
1.6 PROPOSED METHODOLOGIES	4
1.7 EXPECTED RESULTS	5
1.8 WORK BREAKDOWN STRUCTURE.....	7
1.9 GAANT CHART	9
1.10 CONCLUSION	9
CHAPTER 2: LITERATURE REVIEW	10
2.1 INTRODUCTION	10
2.2 MODERN WEB	11
2.3 IMPROVING WEBSITE DESIGN	15
2.4 THE EFFECT OF FANTASY SPORT	17
2.5 THE EFFECTS OF FANTASY FOOTBALL PARTICIPATION	19
2.6 USING SOCIAL MEDIA TO BUILD COMMUNITY	21
2.7 HTTP PROTOCOL OVERVIEW.....	27
2.8 CONCLUSION	29
CHAPTER 3: ANALYSIS AND DESIGN.....	30
3.1 INTRODUCTION	30
3.2 PROPOSED METHODOLOGY	31
3.3 ASSIGNMENT OF INDIVIDUAL TASKS	32
3.4 USE CASES	33
3.5 SEQUENCE DIAGRAMS	38
3.6 USER INTERFACE DESIGN (WIREFRAMES).....	41
3.7 DATABASE SCHEMA DESIGN	47
3.8 BACK END DESIGN	47
3.9 PROPOSED VERSIONING CONTROL SYSTEM: GIT AND GITHUB	48

3.10	CONCLUSION	49
CHAPTER 4: IMPLEMENTATION.....		51
4.1	INTRODUCTION.....	51
4.2	METHODOLOGY.....	52
4.3	PROTOTYPE VERSION 2 (JANUARY 2015).....	54
4.4	PROTOTYPE VERSION 3 (JANUARY 2015).....	56
4.5	PROTOTYPE VERSION 4 (FEBRUARY 2015)	58
4.6	PROTOTYPE VERSION 5 (FEBRUARY 2015)	60
4.7	FANTASY HURLING WEB APPLICATION IMPLEMENTATION.....	62
4.8	GRAPHICS	75
4.9	CONCLUSION	79
CHAPTER 5: TESTING AND EVALUATION		81
5.1	INTRODUCTION.....	81
5.2	HURLING WEB API UNIT TESTS.....	82
5.3	HURLING API TESTS.....	87
5.4	FANTASY HURLING WEBSITE TESTS	89
5.5	CONCLUSION	92
CHAPTER 6: CONCLUSIONS AND FURTHER WORK		93
6.1	INTRODUCTION.....	93
6.2	ACHIEVEMENTS.....	93
6.3	PERSONAL GAIN.....	93
6.4	FURTHER WORK (POSSIBLE IMPROVEMENTS).....	94
6.5	CONCLUSION	95
Appendix A: Project Diaries.....		96
Appendix B: Code Listing of Leagues Admin Page		122
Appendix C: Code Listing of Hurling Web API.....		139
Appendix D: Fantasy Hurling Jerseys.....		186
Bibliography		188

Chapter 1: Introduction

1.1 Introduction

Fantasy sports games are a fun and interactive experience for sports fans. Existing fantasy sports games allow fans to immerse themselves in to a competitive environment and compete against other fans. Fantasy sports games operate with users registering, creating teams with their favorite sports stars and entering online leagues with fans that have also created fantasy teams of their own. Existing fantasy sports games such as fantasy premier league and fantasy baseball provide research statistics detailing the scale of online fantasy sports communities (1).

In Ireland there are two major sports, Soccer and Hurling. Neither of these two sports have a large involvement with fantasy sports games. Although attempts exist online, there are no successful implementations of a fantasy hurling game. With no established game available, hurling fans are deprived the experience of an interactive web based social environment. The purpose of this project is to implement a fantasy hurling game that attempts to fulfill a successful version of an already successful fan experience and applying it to a sport that is lacking an online interactive gaming community.

This paper describes our research and study of fantasy games, online communities and online Gaelic sports websites. We have taken the findings of our research and have detailed how we applied them to the creating of what we feel is a more intuitive, simplified and better social experience for hurling fans. Our application allows hurling fans to have an interactive social gaming outlet.

Our implemented project provides fans with the ability to create, edit and manage their own fantasy team but a key feature is the facility to converse with other online gamers through a social chat environment. Our project follows existing structures for gameplay once the user creates an account. The simplified rules are as follows.

- The user is allocated a maximum budget
- The user uses his budget to create a full team of real life hurling players
- The user can create or enter an existing league of other gamers

- After each real life game week the users points are calculated and added to their current points total
- The user can edit the players on their team but must always be within budget
- The game ends at the end of the real hurling season and the user with the most points in the fantasy league is the winner

The addition of the chat feature allows users to compare and discuss their teams throughout the season.

System Introduction

The system runs in two environments. The user has a front-end environment for playing the game playing through their browser. The administrator has a back-end environment for the maintaining and updating of scores as the game weeks happen.

The technologies used in the project are native to web based applications. Therefore we use HTML, JavaScript, JQuery and CSS to design the interfaces. To connect the front-end user environment to the back-end database we have implemented .NET Web API and Entity Framework, we will host the application on Microsoft's azure servers. These technologies are detailed in following chapters.

1.2 Aim and Objectives

The objective of the project is to build a fantasy football hurling website, with an element of social media capability built in. The reasoning behind making this project is as follows. Anyone who plays fantasy football games knows that although they are quite interesting, a lot of the time is spent just logging in, checking your score, maybe making a change or two, and then logging out. The amount of time actually spent on the site is minimal, we feel this is a mistake in the design of these applications and games.

In this document we will provide some analysis and conceptual design of the project using UML diagrams and modelling techniques. We will look at the following:

- Comprehensive Use Case Diagrams
- Development Methodology
- Sequence Diagrams
- Class Diagram of the final project
- Activity diagrams

- User Interface Design
- Database table and schema design and description

1.3 Main Research Questions

The main questions of research in regards this project are how can we improve on similar apps that are already out there, how much can we find out about the game of hurling, can we find out how to get live figures through a RSS feed, is it possible to use real Logos or are there licencing issues, and of course what technologies will we implement in our project. We should also try research if existing versions of what we might be able to create are currently available, we need to find a gap in the market and if one does not exist the project could be deemed unnecessary.

Is there a gap in the social media market for a fantasy game?

With over 175,000 people on Facebook interested in the GAA. With a worldwide audience now able to watch live hurling games. Is there enough interest to justify the creation of a fantasy hurling game?

Could a fantasy hurling game create revenue?

Could a successful fantasy hurling game be sold to the GAA, to capitalize on their social media presence and create revenue through their website?

Could a fantasy hurling game create social media interest?

If a successful implementation of this project was achieved, Could it generate interest on social media and subsequently expand the number of people engaging with the GAA?

1.4 Justification and Benefits

- There are no similar apps out there for hurling
- Hurling is a very well followed game
- The social media aspect can make it even more popular as outlined in the next section
- The instant messaging service would be unique to a fantasy gaming app
- The possibility of advertising revenues is very high
- Most importantly, hurling fans we have talked to want it!

1.5 Feasibility

This is a group project for three third year Computer science students. Each participant must have access to a personal computer and an Internet connection. That should not pose a challenge to acquire. We should be able to finish the project using open source tools exclusively. There are available open source tools for any task this project would require.

At this point it's not decided what technology (programming languages / frameworks) we are going to use for application development.

Because this project is part of the third year curriculum therefore we have dedicated time available for the project.

1.6 Proposed Methodologies

- First we will need to carry out a preliminary research of similar projects, in other words we will need to see how fantasy sport sites or applications works and what features they offer.
- Secondly we will need to decide what social networking features we want to implement in the application. Therefore should undertake additional research toward contemporary social network paradigms.
- Next challenge lies in the fact that either of us has any real experience developing web applications. Comprehensive study of web technologies and framework is therefore required.
- To carry out the research we will use internet and focus mainly on qualitative research methods
- To develop the application we should use adaptive approach.
- I would argue that Prototyping SDLC is the best approach to carry out the development. In this model developer basically re-analyzes, re-designs and re-implements application prototype until the product is accepted by the client. Considering our lack of experience with web application development we should expect major changes in design, but we should be able to have simple, but functioning prototype reasonably fast.

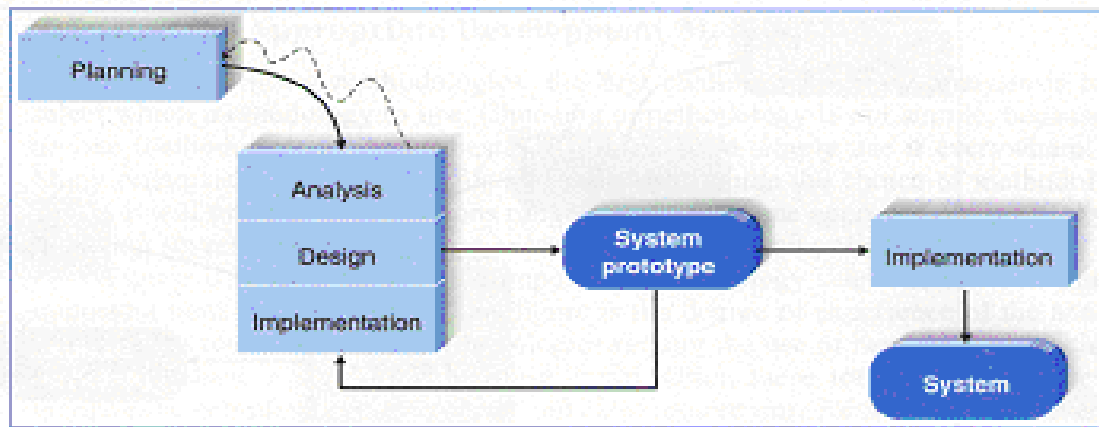


Figure 1 Prototyping SDLC

1.7 Expected Results

With no current social gaming outlet for hurling fans this project could prove fruitful. The potential gap in the market could exist for a game like this to succeed. A successful game could function by itself but could also be sold to the GAA and implemented on their website in a similar fashion to the premier league fantasy football (2).

The GAA could be interested in integrating a social media game on their website to attract the number of followers they have on social media. A fantasy hurling game could increase traffic on the GAA website and generate user interaction, leading to increased advertising revenue.

With social media friends challenging, debating and boasting about their fantasy team through various social media outlets, the GAA can reap some free publicity and advertising for upcoming games.

Successful Project

A successful implementation of this project would create an exciting social environment for hurling fans. The fantasy hurling game would allow users to post their results and scores on social media sites. To be successful users should find the website interface easy to navigate. Playing the game should be intuitive and simple. Users should be able to retrieve their fantasy team's results on the final day of a game week. The server should be responsive and load player information quickly. The user-to-user messaging service should be used frequently, for both challenging and talking about hurling. The system should be modular and deployable for a number of different sports.

The game should be complete and function well enough to potentially be sold to a third party, namely the GAA. Without deconstructing the system the game could be implemented on to a new website with little disruption. This would make the game profitable and marketable. This would be the ultimate goal and a truly successful project.

Successful System

A successful system would implement all the functionalities outlined in the methodology. Users could create an account, login, customize a profile, create a team, create a league, invite users, challenge users and use the messaging features within the system. The client side of the system would interact with the database. Data related to players scores and results would dynamically update on the client side after being updated by the administrator on the server. However if the game failed to attract interest from gamers and investors, the project may prove to have been created in vain. A successful system could be modified to adapt to different sports but without users playing the game the project would be considered incomplete.

Failed Project

A failure to create a functioning system that allows users to play the game would be considered a failed project. The project aims are to create a fantasy hurling gaming system with social media capabilities that can operate as a standalone website. Potentially if the system created was successful and the market was there for hurling fans, users of social media and an investor to interact and back the game, the project would be considered successful. Failure to create a functioning system however has a knock on effect to potential investors. How can an investor buy a system that does not exist? A failed project can only occur if the team fails to build the system.

Incomplete Project

Failure to complete the system before the deadline could be considered a failed project. The difference between incomplete and failed is that the team mismanaged the time allocated to create the project. The system could still be completed in future iterations. If the system is completed outside of the allocated time and the market still exists for a fantasy hurling game with social media capabilities, the project may not be considered failed and could still potentially be distributed at a later date.

The plan is to deliver fully tested and working application implemented on the web in early May 2015. We would consider to be a great success if we managed to add few additional features and make the application stand out from the competition. The most important reason why we agreed on this project is an exposition to cutting-edge web technologies such as HTML5, Angular.js, GWT, Node.js and Rubi on Rails etc. We are planning to exhaust this opportunity for learning.

1.8 Work Breakdown Structure

1.1. Design

- 1.1.1. User Interface
 - 1.1.1.1. Specification
 - 1.1.1.2. Interface Model
 - 1.1.1.3. Design review
- 1.1.2. SQL Database
 - 1.1.2.1. Identify table relationships
 - 1.1.2.2. Database model diagram
- 1.1.3. User case, class and sequence diagrams

1.2. Development

- 1.2.1. Research on web technologies and frameworks
 - 1.2.1.1. Find most appropriate technology
- 1.2.2. Front-end
 - 1.2.2.1. Code web pages
 - 1.2.2.2. Conduct unit tests
 - 1.2.2.3. Review web page design and functionality
- 1.2.3. SQL Database
 - 1.2.3.1. Build database
 - 1.2.3.2. Review database
- 1.2.4. Back-end and web services
 - 1.2.4.1. Code back-end
 - 1.2.4.2. Conduct unit tests
 - 1.2.4.3. Review back-end functionality

1.3. Quality Assurance

- 1.3.1. Front-end
 - 1.3.1.1. Perform integration tests
 - 1.3.1.2. Perform user acceptance tests
- 1.3.2. SQL Database
 - 1.3.2.1. Perform integration tests
- 1.3.3. Back-end and web services
 - 1.3.3.1. Perform integration tests

1.4. Implementation

- 1.4.1. Hardware
 - 1.4.1.1. Determine hardware needs
- 1.4.2. Software
 - 1.4.2.1. Determine system software needs
- 1.4.3. Deployment
 - 1.4.3.1. Make hardware production-ready
 - 1.4.3.2. Verify code
 - 1.4.3.3. Deploy

1.5. Post-Implementation

- 1.5.1. Verification
 - 1.5.1.1. Obtain user acceptance of production system
- 1.5.2. Monitoring
 - 1.5.2.1. Verify performance and functionality
- 1.5.3. Project Wrap-up

1.9 Gaant chart

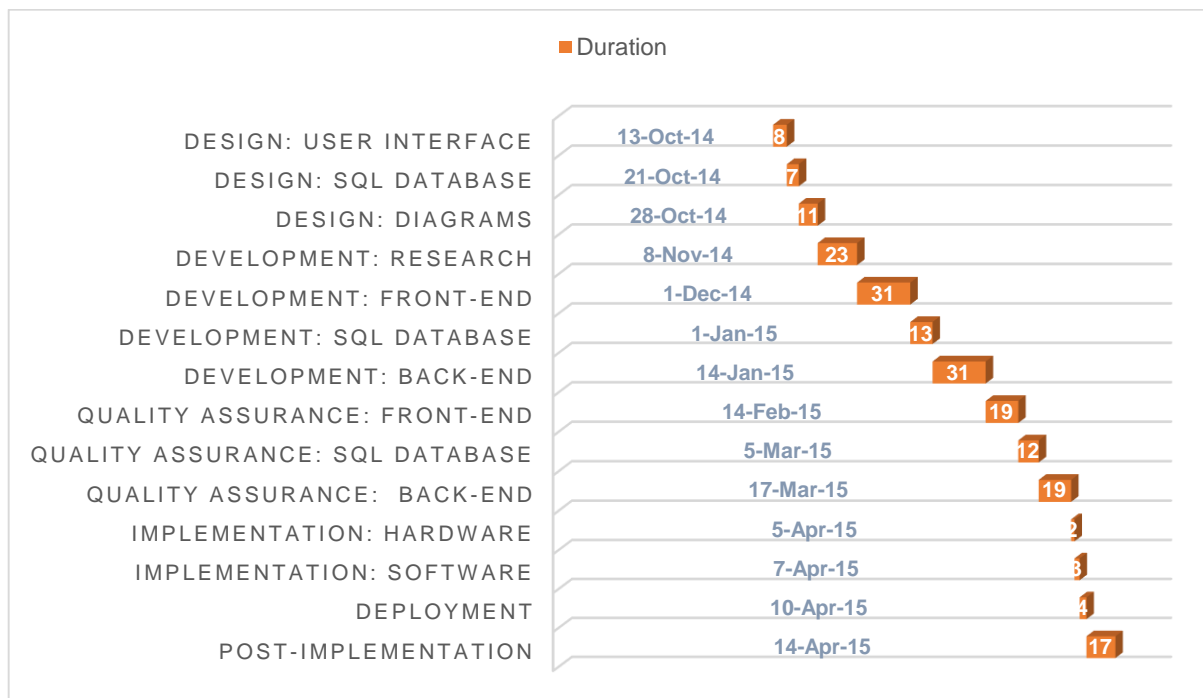


Figure 2 Gant Chart

1.10 Conclusion

In order to begin the analysis phase of the project the team must research all the data relevant to the proposed fantasy hurling game with social media capabilities, Information about social media interaction between the GAA and hurling fans. With no current game similar to the proposed project available, there appears to be room in the social media gaming market for a fantasy hurling game to exist and benefit fans of the sport and the organizers.

Fans can use the game as an outlet to interact with fellow hurling fans and feel involved in their sport. The GAA can use the game to attract visitors to their website. The project would need to consult the GAA over licensing laws and complying within the existing copyright laws.

The team involved has all the necessary skills to implement this project. Referencing the task time line in figure 2, the project is feasible within the allocated time. The most difficult parts of the project will be creating the website and updating the real life player ratings on the database.

Chapter 2: Literature Review

2.1 Introduction

This project offers users two levels of interaction. The primary level of this project offers users a fluid gaming experience. The secondary level is the ability to build an online community through the application.

Our group wanted to approach this project with a strong understanding of what a successful implementation of this project would entail. In order to do so, each member of the group chose a topic related to this project and researched literature connected with it.

The topics we chose to research individually are listed below,

2.2 Modern Web

A study on the ubiquitous nature of the modern Internet

2.3 Improving Website Design

A study on the usability and visual aspects of web design

2.4 The Effects of Fantasy Sports Participation through Social Media

A study on how fantasy sports games use social media to interact

2.5 The Effects of Fantasy Football Participation on NFL Consumption

A study on how fantasy football gamers consume NFL information

2.6 Using Social Media to Build Community

A study on how companies use social media to interact with clients

2.7 HTTP Protocol Overview

A study on HTTP

Our group felt that by studying literature on these six topics we would have a greater understanding for what was needed to create a successful project.

2.2 Modern Web

Abstract

Today's web is more complicated than ever. Web content is delivered to desktops, laptops, mobile devices, game consoles, and smart televisions. Each has a different screen size, performance capabilities and additional features. Web content is delivered in the form of dynamic web pages and feature rich applications in an environment without a common standard in web browsers. In addition to that, Internet usage reaches billions of users online with all the human diversity additional complexity can be problematic. The purpose of this paper is to review some studies and white papers that address these problems. The research done in this field is immense and produced many solutions in the form of frameworks, tooltips, plugins, design patterns and methodologies and it's out of the scope of this paper to cover all of those. I tried to name only general challenges in this review. Therefore I chose three white papers to study with additional research of relevant articles on the web. My work was driven by the need to acquire better understanding of the current state of the web, the web development challenges and methodologies. In conclusion I can claim that this paper delivers just that. It does not provide any specific solutions, but general recommendations.

Platforms: Mobile devices are on the move

Each month an increasing number of people are switching from PCs to tablets and mobile devices to browse the internet and by 2015 they'll be using those more than any other device. Mobile traffic grew 120% between Q2 2013 and Q2 2014 while desktop traffic remained flat.

GLOBAL	Website Visits by Device	Q2 2013	Q3 2013	Q4 2013	Q1 2014	Q2 2014
	Traditional	79.18%	76.20%	73.42%	72.50%	69.91%
	Tablet	12.16%	13.30%	14.69%	15.14%	13.84%
	Smartphone	8.66%	10.50%	11.89%	12.36%	16.25%

Figure 3 IBM Analytics Benchmark

From a development point of view certain areas on a website may function differently

Some of the issues are:

- No Flash support
- Popups stops a website to being used on a small screen
- Tiny text links unusable on the touch screen
- Website elements overlapping on the small screen

or may not function at all on mobile or tablet devices.

Screen size and performance restrictions are the main reasons for design re-evaluation of the application design. Web developers are using separate mobile pages, responsive web design and business apps to ensure a good user experience regardless of device type.

Browsers: Many knights, but no king

In the past, nobody questioned which browser ruled the land – it was Microsoft's Internet Explorer. According to the latest analysis of the Web browsers, situation is very different now. Web browser market share varies from platform to platform.

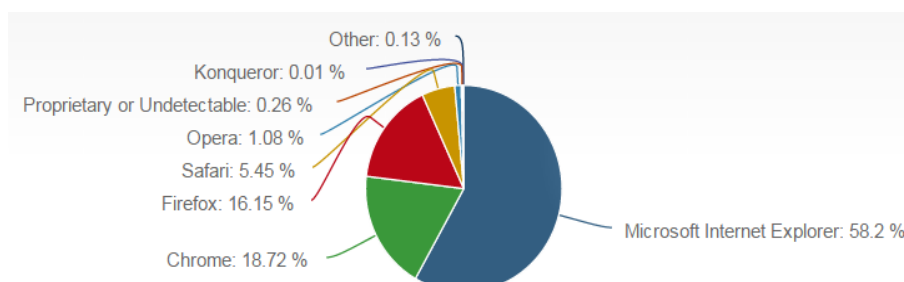


Figure 4 Desktop Browsers

Microsoft still wins desktop market with 58.2% share has acknowledged being critically late to mobile where his presence is almost negligible with 2.21% share. This led to domination of Web kit-based browsers: Safari, Chrome and Android with Apple's Safari being the leader with 48.91% share.

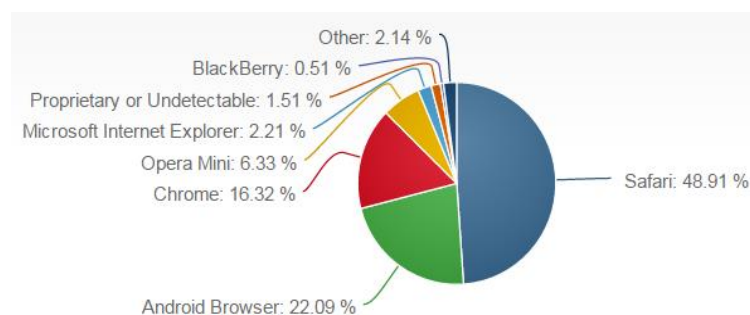


Figure 5 Mobile Browsers

Situation in the console market is very different, with most of the market shares in the hands of proprietary browsers embedded in a particular device. Only Opera is getting a 4.51% share as a result of Nintendo implementing Opera browser in some of their consoles. (1).

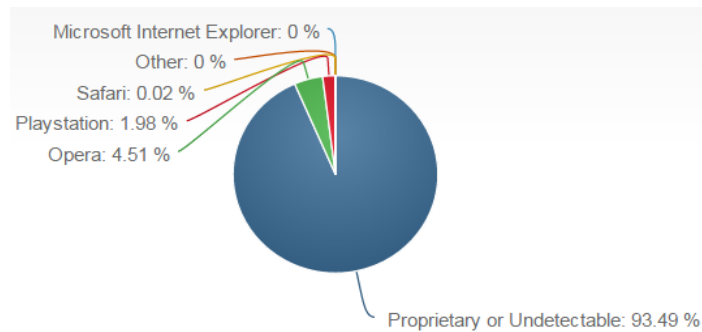


Figure 6 Console Browsers

It's great that Internet users have so many browser options. Unfortunately developers must put a lot of work into making sure that websites are compatible with as many browsers as possible.

Here are some of the most common browser compatibility issuesInvalid source specified.:

- Elements not correctly positioned
- Browser specific CSS styles
- Lack of Valid HTML/CSS
- JavaScript processing times
- Frames related problems

There is a vast research done to tackle these issues and many useful practices, methodologies, frameworks and tooltips have arisen in recent years. Technologies like HTML5 and CSS3 are making cross-browser problems more manageable and web developers are getting less concerned about browser compatibility issues (2).

Sources: More content is coming from third parties

Third-party content is a term that covers a lot of ground, from advertising to guest blog posts.

The term frequently include:

- Content delivery networks (CDNs)
- Advertising
- Web analytics
- Social Media
- Microblogs
- Miscellaneous Widgets

Analysis shows that the average web transaction involves 8.59 different third parties or hosts. (3). These web page components are dragging down the overall performance of the website. Various researches have been done to illustrate the impact that third-party additions have on website performance. Some of the third-party widgets can impact a site's performance by as much as 30% (4). In spite of these findings third-party components are virtually indispensable to many web sites today, whether they bring in revenue through advertising, or drive business by bringing in new users through social media. To tackle performance issues the best option includes continuous performance monitoring of the web site to pinpoint the components with the worst impact.

What developers think?

Recent study (5) took a different approach to name modern web challenges. They tried to find out what are the issues that web developers are discussing among themselves. To do so, they examined over 500,000 questions asked on <http://www.stackoverflow.com> over a four-year period (2009-2013).

Here are the big takeaways from their analysis:

- JavaScript is still most popular topic, but HTML5 and CSS is raising
- Cross-browser compatibility still big, but declining
- New HTML5 and JavaScript features questions are raising
- Web technologies are growing in importance in mobile development

Conclusion

Modern web is a complex and lively medium. New technologies arise on daily bases. Users are connecting to web with various devices through different browsers. Webpages today are dynamic applications with third party content interconnecting different domains. With all these fast paced changes, the contemporary web developers are facing many new and difficult challenges. Evolving and researching never stops in the life of web designers and developers. New frameworks, toolkits, methodologies are created to tackle major issues and improve web development productivity. Some of the examples of prominent frameworks are Bootstrap, Grails, JQuery, Node.js (6).

The user doesn't care about all these technical issues. He wants to browse his web sites from laptop, mobile phone or game console. He expects a web page to perform quickly

and properly every time and on every device. He wants to share content or buy a product by touching a widget. All this is possible on today's Web.

2.3 Improving Website Design

Who did what?

Melody Y.Ivory and Marti A.Hearst University of California, Berkeley 2002 investigated what makes a good design to a website, and developed a model to help users implement these changes that would make the website more pleasing to the eye and in terms of usability.

Summary

Two students undertook a paper to investigate current methods of website design, their effect on productivity and how it can be improved through better website design principles. They undertook this as part of the Web Tango project. They aimed to help steer the average web site builder away from poor design principles and toward an automated quality checking tool and a grammar checking tool. They made these tools available online at "webtango.berkeley.edu".

Melody Y.Ivory and Marti A.Hearst (2002) found that a website is a complex mix of text, links, elements and formatting, surmising that all these aspects affect a website's quality and usability. They came to the conclusion that these principles are important in thinking about website design to begin with.

The students came up with a table that created a way of calculating how many measures for each element on a web page were needed in order to come up with a design that is pleasing and easy to use. For example, they came to the conclusion that there are 31 separate measures that are important to think about in regards to a text element including amount, size and complexity. The table also related to site architecture and the performance of the page, all elements discussed had a set of measures that were important to that particular element to optimize design. In total there were 157 measures found.

Melody Y.Ivory and Marti A.Hearst (2002) then ran their crawler tool and used this to gather sample web pages. Initially it ran on the home page of a website and randomly selected pages at successive levels starting at that page and only selected informational pages

ignoring advertisements or flash pages totally. The analysis tool then runs on these pages and retrieves the information on each element in conjunction with a site metrics computation tool, forming the table they discussed with the elements and their measures.

The students found there were three main principles to successfully designing a webpage; navigation design, graphic design and experience design. From these 3 main principles a hierarchical pyramid model was built, surmising that on the top level is the site architecture, while on the bottom are the actual site elements themselves. Web design literature and user studies were used to come up with the final model. A tool was then developed from this model that could compute 157 site level measures. The accuracy of this tool was tested on many websites and it was found to be 86% accurate on 154 measures.

The students performed three studies to try to predict page and site ratings. From these 3 they developed a simple prediction model. They called it the WebTango model. Firstly they drew up an analysis of 428 web pages and found expert reviews and ratings on these pages from PC Magazine's top 100 sites. They called sites either rated or un-rated and set out to find a way to predict which category a sample site would fall under. They then computed 12 quantitative measures related to page composition and design among other factors. They tried to see if they could predict with their model the page standings on this top 100. They found that 6 features were most important to design. The most prevalent were text cluster, reading complexity, and colour count and page size. They found that in rated sites these features needed to be tweaked a certain way to make the site very usable and stay in the rated group.

A second study was conducted and asked 6 website design experts to examine 1898 pages from the Webby awards winner's websites. These pages were judged on certain criteria including content, structure, navigation and visual aesthetics. They broke these pages into three groups, "good", "not good" and "poor". They wanted to see if their model, using the measures they had created, could predict which group a site would fall into. Predictive accuracy was 67%.

The third and final study was to analyse over 5000 pages from 300 sites. They used all 157 measures from their model and again had 3 groups, "good", "average" and "poor". They used the model to try predict which group the web pages would fall under. It was proven to be accurate on a page level 96% of the time, and accurate on a site level over 60% of the time.

Melody Y.Ivory and Marti A.Hearst (2002) talked about their final task of applying the model they had built to website design. They took a sample of 15 web pages and made minor tweaks to these based off their model parameters. They asked 13 people to analyse the pages both before and after the adjustments had been applied to them using the model. They made findings that 10 out of 13 people preferred the web pages after they had used to model to make adjustments to it.

The students then analysed their final tool, the WebTango system. They analysed how the tool worked and how it can be applied to a website. The tool compared all 157 site level measures from the website, and then makes suggestions on how to improve the website and also gives links to example websites who are similar in type but have been designed to a higher level.

The major findings were in essence that a model can be developed by 2 students with little or no web design experience that can enhance the usability and visual aspects of a website very quickly by identifying areas that can be improved. It found that although some studies have found automated tools find it difficult to find problems with usability issues, the tool developed is considered a practical and useful solution to solve design issues early in the design phase of a websites development.

2.4 The Effect of Fantasy Sport

Sport spectatorship is one of the largest forms of leisure behavior in the world (9). Large numbers of sports fans attend sporting events frequently and follow their sport ubiquitously through various mediums, TV, radio, newspapers, and magazines. With advancements in Internet technologies, fans are redirecting their consumption of sport to online platforms. Sports fans can now consume news and content via applications and social media on mobile devices. Fans can connect with each other through tweets, texts, blogs and Facebook.

Today the exponential growth of fantasy sports participation is fuelled by the increased televised broadcasting of sport, accessibility of sport and statistics on the Internet and the advent of social media as a communication outlet for fans worldwide, because of this, fantasy sport platforms are now formally aligned with many official major sporting leagues globally.

Fantasy sport provides engaged fans a unique and personal opportunity to be involved while also building the interest and knowledge of new participants with lower prior involvement. Further, fantasy sport provides additional social interaction between players. It can serve as an education tool about the sport, teams and players and can also serve to increase the commitment and interaction of consumers with a sport.

2.2 Existing Trends

In North America and Canada the fantasy sports industry has a body that monitors and provides information on all fantasy sports websites known as the Fantasy Sports Trade Association (FSTA) (1). The FSTA released figures on the number of participants in 2011, more than 35.9 million users are involved in some type of fantasy sport. The FSTA then surveyed the group to find out how many users participate using mobile devices. The survey found that 30% (12,000,000) of users are engaging through mobile devices. Further, of the 12,000,000 users that play through mobile devices 14% (1,680,000) use social media to discuss their fantasy games.

The Fantasy Premier League game has over 3,000,000 participants (2). If we apply the same calculations from FSTA to the Fantasy Premier League game we can get an estimate value of over 125,000 users that discuss their fantasy football game via social media.

Reason for Literature Review

This literature review is being conducted to ensure a thorough understanding of the connection between social media communities and fantasy gaming. I am currently working on a project that is combining a fantasy-hurling platform that enables social media capabilities.

Upon completion of this literature review I hope to identify potential areas of difficulty that may be unforeseen without specific research in to the topic of social media and fantasy gaming.

2.5 The Effects of Fantasy Football Participation

Drayer, Shapiro, Dwyer, Morse and White proposed three areas of interaction that may be affected for fantasy football users and non-fantasy football users (10).

1. Identification and loyalty
2. Consumption
3. Attitudes and behavior

Identification and loyalty

To conduct a test on the fantasy gaming NFL fans, the authors must first define a norm. A fan that does not participate in fantasy football and the attributes of one such fan. A normal fan is considered to have allegiance to one team. The normal fan supports the team and the players that play for their team and no one else.

The authors expected gamers to have no loyalty or identification to a certain team, but rather have identification to a player. This was not the case, gamers still identified themselves with a team but their affiliation to other players via the fantasy game was just an extension.

Those interviewed stated that their loyalty to a fantasy player ended with the season.

Only 2 people of the 13 interviewed said that they carried affinity for a player in to the next season and rooted for them if their favorite team wasn't playing.

Consumption

To conduct this test the authors once again had to define a norm. A normal fan is one that only consumes information about their favorite team.

The authors expected gamers to consume NFL information in a different manor. Having affiliation with many players in many different teams' leads to greater consumption of information.

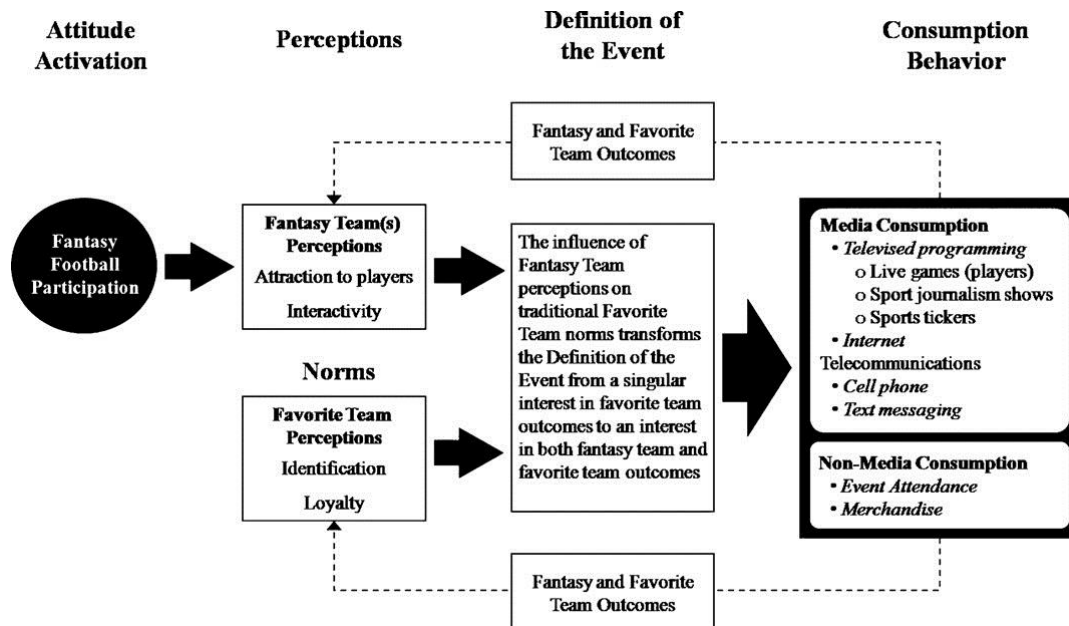


Figure 7 Consumption

Fantasy gamers have an increased media consumption of the NFL, spending time on the Internet and watching ESPN to gain information on starting rosters, injured players and to check which teams are playing in game weeks. Of those surveyed participants admitted to checking their mobile device and going on to the Internet every 10 minutes while a game is on. Some gamers paid for subscription sites to get informed advice on their fantasy teams. Fantasy football participants found themselves watching more matches on game weeks because they had numerous players playing in different games. When asked if they would watch as many games if not playing the fantasy football game, the answer was no.

No change was noted in the form of consumerism. Only one person said they would buy the jersey of a player because she began to like him in her fantasy team.

Attitudes and behavior

The control norm for this test was that of a fan, who regardless of their favorite teams result would support them.

The results found in the survey suggested that a fantasy football player would have allegiance for a team but if that team lost, the fantasy football player would adjust their attitude towards a real team to suit their fantasy game. When the 13 fantasy gamers were asked about their behavior towards an opposing player, 2 gamers admitted to supporting opposing players of their favorite team.

Those interviewed stated that if their fantasy football team began to lose and become uncompetitive they would lose interest and stop updating their team towards the end of the season.

Research Paper Conclusion

The research results show that fantasy gaming participation really does have an effect on how fans, consume, interact, behave and identify themselves with teams. A non-fantasy gamer does not consume as much information over a broad scale but rather focuses solely on their favorite team. With fantasy gamers this is not the case. Fantasy gamers spend more time online, on blogs, sports news websites, social media, TV, magazines trying to get information on their fantasy team. Fantasy gamers tend to be more fickle in their allegiance to a team and player, favoring and supporting teams and players that can generate points for them in their fantasy league.

2.6 Using Social Media to Build Community

Komaromi and Erickson follow a template laid out by Culnan, McHugh and Zubillaga that assesses the effectiveness of an organizations social media strategy. Effective implementation of a social media strategy is one that attracts an audience enabling them to engage and interact with both the organization and each other (11).

Although the study is not conducted on a sports category, the essential requirements for a successful social media community apply in all communities. The study instead researches 18 insurance firms over a two-month period in 2011. The authors then present specific results for three firms that display a range of social media outlets to support a community.

The three insurance companies presented are, Progressive, Liberty Mutual and State Farm. The study begins by analyzing the company's website homepages. The homepages all display a range of tools that allow customers to interact by getting quotes, mobile apps, price comparisons and calculations for various insurance plans. What is important on all these homepages is the prominence of social media resources. All three websites have social media icons to connect via Twitter, Facebook and YouTube. Liberty Mutual is the only site that has

a LinkedIn page that allows customer access. State Farm has an additional social media outlet through Flickr.

Metrics for each homepage visit can be seen in the graph below. Progressive has the highest unique visit rate at over 4 million visits per month. State Farm is second with 2.5 million and Liberty Mutual is third with 1 million unique visits.



Figure 8 Website Traffic

Facebook

Facebook is the first social media outlet covered in the findings. All three insurance companies are present on Facebook.

Progressive have a Facebook page that primarily provides information about the company, offers and customer feedback. Progressive also have a current marketing campaign that attracts the most followers of the three companies Facebook accounts. “Flo” is the name of that marketing campaign and has a Facebook account that is updated daily and has interactive videos and games to engage customers. At the time of research, Progressive has a following of over 2.5 million on their Facebook accounts.

The Liberty Mutual Facebook page is similar to Progressive, They have an about section to provide information about the company. Further Liberty Mutual has videos, comments, polls and games on their page to engage customers. Their Facebook wall has daily posts from employees and brokers that can be interacted with by visitors via likes and comments.

State Farm’s Facebook page offers a diverse content including offers, promotions, and sponsorships for social causes, games to rate drivers and apps that can be played with friends. The site also has a driving game application to complete a safe driving discount offer. Their wall has many comments both by State Farm and visitors.

	Progressive	Liberty Mutual	State Farm
Facebook	~2,500,000 followers	~10,000 followers	~90,000 followers

Although Progressive have an advantage with a larger hold on the market, the evidence is clear that an updated and well-maintained Facebook page that interacts with visitors gathers more followers. State Farm has a large number of followers because they engage their visitors with apps, polls, videos and respond to comments daily.

Twitter

Twitter is the second social media outlet researched in the report. All three insurance companies have an active twitter presence.

Progressive utilize twitter to connect with customers with posts daily and responding to complaints, questions and criticism. Progressives Twitter account post basic answers to very common customer questions in a timely basis.

The Liberty Mutual Twitter account is active once or twice a day. Their Twitter account is used to promote news and answer questions from customers. Liberty Mutual also uses their Twitter account to promote sponsorship activities and encouraging followers to attend social events.

State Farm populates their Twitter feed with posts daily, ranging from comments, questions, complaints, and re-tweets from other third party sources. The emphasis is on responding to customer queries. State Farm is more concerned with customer comments and feedback than Liberty Mutual and Progressive. State Farm shares a lot of content from their other social media sites through twitter, connecting their Facebook, YouTube and Flickr accounts.

	Progressive	Liberty Mutual	State Farm
Twitter	~7000 followers	~2500 followers	~30,000 followers

YouTube

YouTube is the third and final social media outlet researched in this paper. All three insurance companies again utilize YouTube as a connection tool to their customers.

Progressive don't only use their YouTube account as a video player but also implement a quote app link and link to their other social media outlets. The site has adverts,

promotions and visitor uploads. Progressive had 97 videos uploaded to their account at the time of research and the comments section on their YouTube channel is very active with over

	Progressive	Liberty Mutual	State Farm
YouTube	~2,000,000 followers	~250,000 followers	~3,000,000 followers

400 comments.

Liberty Mutual has 31 videos uploaded to their YouTube account, providing both advertising and more in-depth information. Liberty Mutual does not update or maintain their YouTube account frequently with only a hand full of comments on their account page.

State Farm has the most amount of content on their YouTube account, with over 180 videos at the time of research. State Farms videos range from adverts, promotion offers, to custom content. Fail Fridays is a popular video series that compares customer reasons for claiming insurance. State Farm also provides instructional videos to prevent accidents. Comments are extensive across all their videos and a find-an-agent app is available.

Research Paper Conclusion

Social media connections to customers are of huge importance to building a community around a product. The statistical analysis of three insurance firms provides evidence of such a statement. State Farm update their social media accounts more frequently than the other two firms as a result the community surrounding State Farm is larger and engages more with the company. State Farm makes it important to respond and interact with followers daily and update followers with relevant information regarding their industry and company.

Research Paper Connection

Although the topics in both research papers are very different, the ubiquitous nature of social media is the connection. Taking advantage of social media to connect fantasy sport gamers and allow them to share their information is easier said than done. In order to understand the best methods of engaging fantasy sport fans and encourage them to use social media as a platform to create and interact with a community is the problem.

The first paper reviewed states that fantasy sports gamers clearly consume information at a higher level than that of a normal spectator. Fantasy sports gamers gather more information regarding a sport through magazines, TV, radio, blogs and online. A part of making the transition to social media is by creating a community where user content is

updated and fans can interact with one another, sharing information, tactics, statistics and analysis.

The second paper reviewed contains data and an analysis of social media communities that already exist. This data gives an insight in to how a social media community works.

State Farm take advantage of their social media presence more than the other two firms. The number of followers proves that maintaining a regular social media presence attracts followers. Having a Facebook page and Twitter account is essential to connecting with an online community. All three firms in the research paper use these social media outlets regularly to post comments and just as importantly to receive questions, input and criticism.

Gaps in Research Papers

After doing research external of this literary review I would like to draw attention to some gaps that I feel exist in both research papers. Although the connection between the two papers is very real and an outlet for fantasy sport gamer discussion can be managed in a social media platform, questions must be asked about the data analyzed.

The research paper by, Joris Drayer, Stephen L. Shapiro, Brendan Dwyer, Alan L. Morse, Joel White. **“The effects of fantasy football participation on NFL (National Football League) consumption: A qualitative analysis”** is a research paper about fantasy football in America and not sports globally, including hurling. Questions about a fantasy sports player consumption of information must be asked in order to get a complete view of the fantasy sport industry.

- Do all fantasy sports players consume more information about their sport than that of a normal fan?
- Do fantasy sports games accommodate gamer interaction?

The survey documented in the research paper reports on only 13 test users. The information could be incorrect or corrupted by such a small number of test users.

Kurt Komaromi, Scott Erickson. **“Using Social Media to Build Community”** researched the social communities surrounding three large insurance firms in America. The data gathered in this research paper is informative and formidable but questions still exist in the form below.

Can the social community of an insurance firm be relevant to that of a fantasy sport community?

- How do insurance customers differ from fantasy sports gamers?
- Have all forms of social media been researched?

Literature Review Conclusion

It appears that fantasy sports gamers may consume information and data of their sport on a different level to that of a normal fan. Fantasy sports fans follow a greater number of players and teams connected to their fantasy game. A platform to cater for an online community of gamers could exist to allow gamers to share information with fellow gamers. Fantasy gamers spend more time on mobile devices during games and between game weeks researching real life player and team data.

Fantasy players can still identify themselves normally with one team but can also extend their identification to a number of other teams during the season. Gamers however have a tendency to lose interest in their fantasy games if their fantasy team is performing poorly or their real life favorite team is performing well.

Social media communities flourish when content is updated regularly and interaction between community members is high. Building a social community takes many platforms through YouTube, Facebook and Twitter. Connecting social media outlets together such as linking videos on Twitter to videos on YouTube engages the community to interact within itself.

Having platforms high in content produced by community such as comments and questions creates a larger following. Users tend not to follow social media outlets that rarely post content. The insurance company with the smallest amount of content on social media platforms had the fewest number of followers. The insurance company that had the largest amount of content, videos, posts, tweets and applications had the largest number of followers. The connection between a maintained up to date social media presence directly relates to the number of followers in the community.

2.7 HTTP Protocol Overview

Hypertext Transfer Protocol is a stateless application layer protocol for communicating between distributed systems (7). HTTP is specified by RFC 2616 (8).

HTTP is connectionless, stateless and media independent protocol. A client sends a request and server answers with a response, after that the connection is closed and communication is forgotten on both sides. Current version of HTTP is 1.1.

HTTP is using Uniform Resource Identifiers (9) for resource requests.

Generic URI format

```
URI = "http:" "://" host [":" port] [abs_path ["?" query]]
```

Example

```
https://www.youtube.com/watch?v=BNgU-ZaF06w
```

HTTP request

A Request-line

Zero or more header fields ending with CRLF

An empty line ending with CRLF

A message body (optional)

Request Line generic syntax

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Request methods

- GET: asking server for a resource located at given URI
- HEAD: similar to GET request method, but only header is returned from server
- POST: asking to send a data in request body to server
- PUT: asking to replace a resource located at given URI with new data in request body
- DELETE: asking to delete a resource located at given URI
- CONNECT: asking to establish a tunnel to server identified by given URI
- OPTIONS: describes the communication options for resource at given URI
- TRACE: test the resource availability

HTTP Request Simple example

```
GET http://www.itb.ie/ HTTP/1.1
User-Agent: Fiddler
Host: www.itb.ie
```

User-Agent and Host are some of many predefined header fields. Custom fields can be introduced as well. This request message doesn't contain any message body.

HTTP response

```
A Status-line
Zero or more header fields ending with CRLF
An empty line ending with CRLF
A message body (optional)
```

Status Line generic syntax

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Status Code

A three-digit integer. First digit defines the class of the status.

- 1xx : Informational
- 2xx : Success
- 3xx : Redirection
- 4xx : Client Error
- 5xx : Server Error

HTTP Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 14887
Content-Type: text/html
Server: Microsoft-IIS/7.5
Set-Cookie: ASPSESSIONIDQAADTRCR=KIIFFBMDCAOAICBNIDKKACIH; path=/
X-Powered-By: ASP.NET
Date: Sun, 05 Apr 2015 09:36:25 GMT
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:Lang="en" Lang="en">
```

Rest of the html omitted...

Example above is the response to <http://www.itb.ie> request. Response returned with status code 200 which indicates the success. After Status Line the various header fields follows. Then there is an empty line followed by message body. The message body contains actual html document of requested web page. Most of the html document is omitted in this example.

2.8 Conclusion

The literature reviewed gave the group a better understanding of how the project would need to be approached. The reviews varied from one another on a scale that gave the project a strong foundation to build upon. The literature had both technical and observational research on components from our project, this helped give our group a better view of how separate aspects of our project would come together.

Reviewing literature on the modern web, web design and HTTP had a technical impact on the project. These reviews helped shape the core of the project. Reviewing literature on social media and how it can be utilised to create communities and facilitate user interaction helped direct the path our project would take. Some of the reviews confirmed that the technologies we intended on using were indeed the correct choice, while other reviews suggested that maybe social media is a stronger tool than we originally anticipated.

A challenge in finding the correct literature to research was difficult, not because the literature wasn't available but because we needed to choose the literature that connected aspects of our project to each other.

Chapter 3: Analysis and Design

3.1 Introduction

The aim of this project is to create a website with the ability of allowing visitor to play an online fantasy hurling game and accommodate social interaction between members. The website is a rich web application that will be backed by persistence storage database allowing the fantasy hurling data to be dynamic.

The user will only ever be able to interact with the client side of the website consisting of a playable game, customizable user profile and social media features including chat messenger. The administrator will interact with the server side of the system, maintaining and updating the application. This data will update the user's scores and information dynamically. Together the system would be fully functional with the client side interface backed by our database.

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)

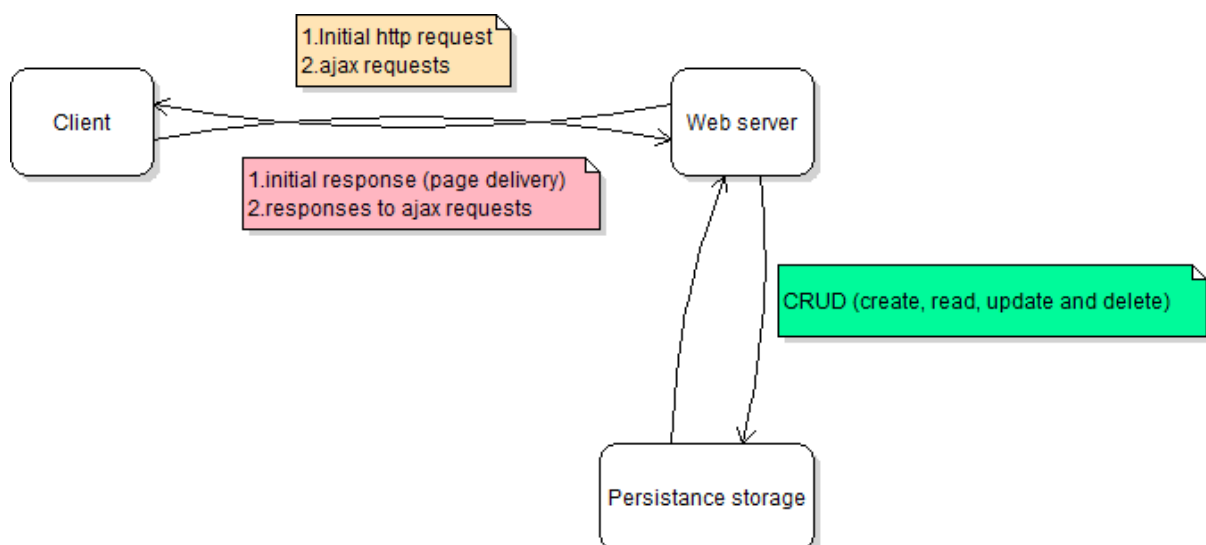


Figure 9 Web Application general design

3.2 Proposed 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.

In this approach the development team implements a ‘sample’ that has very limited functionality of the proposed project and then 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 built (10).

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 have never built a web application of this scope. With prototyping approach we have the ability to redesign the sample as many times as needed.

With each prototype iteration we will learn and progress the application. 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 on each layer throughout the system 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’.

3.3 Assignment of Individual Tasks

David Kelly

- Front-End design *responsibility* (wire frames, Photoshop, 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

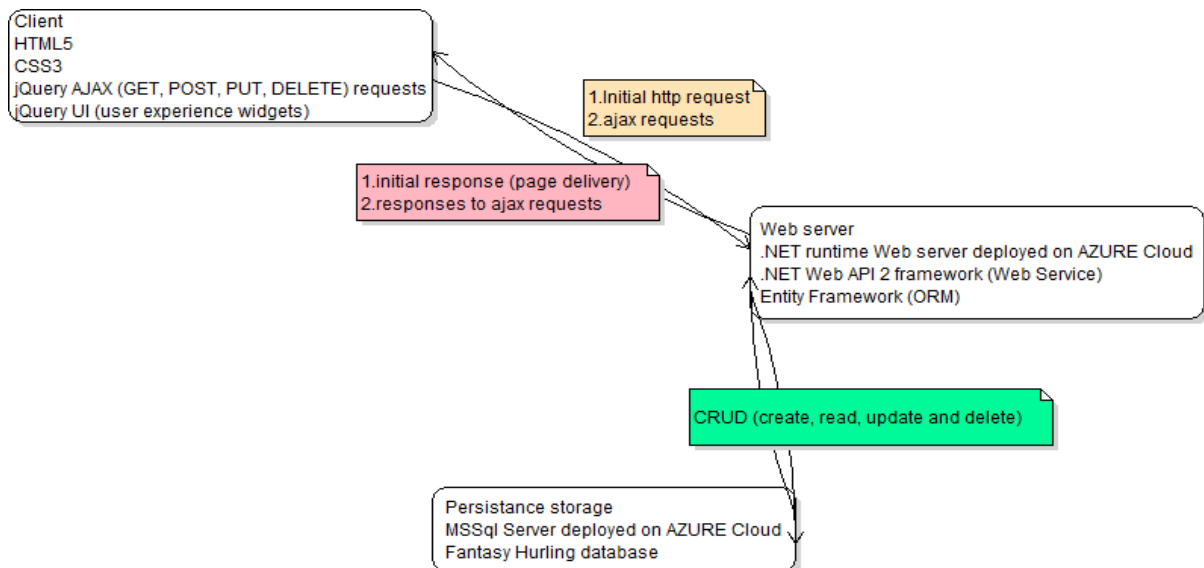


Figure 10 Fantasy Hurling Web Application in detail

3.4 Use Cases

User checks current score

Use case specification:

- 1: User logs in
- 1a: User logs off
- 1b: User not registered and is prompted to register
- 2: User checks their current score

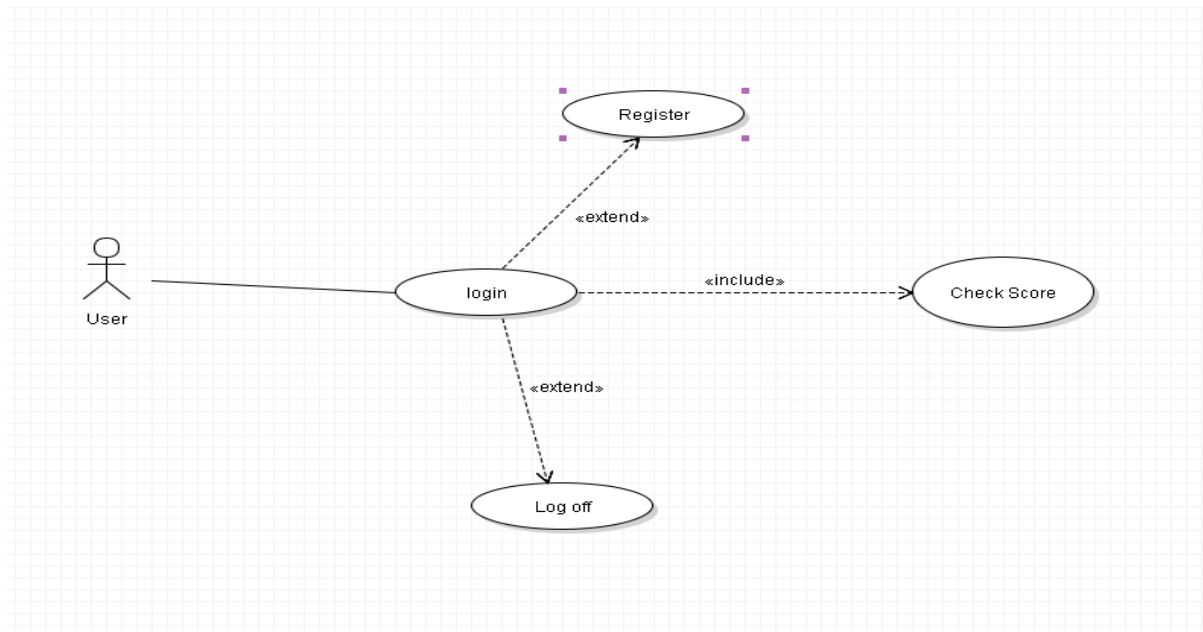


Figure 11 Use Case: User checks current score

User checks fixture

Use case specification:

1: User logs in

1a: User logs off

1b: User not registered and is prompted to register

2: User checks fixtures coming up

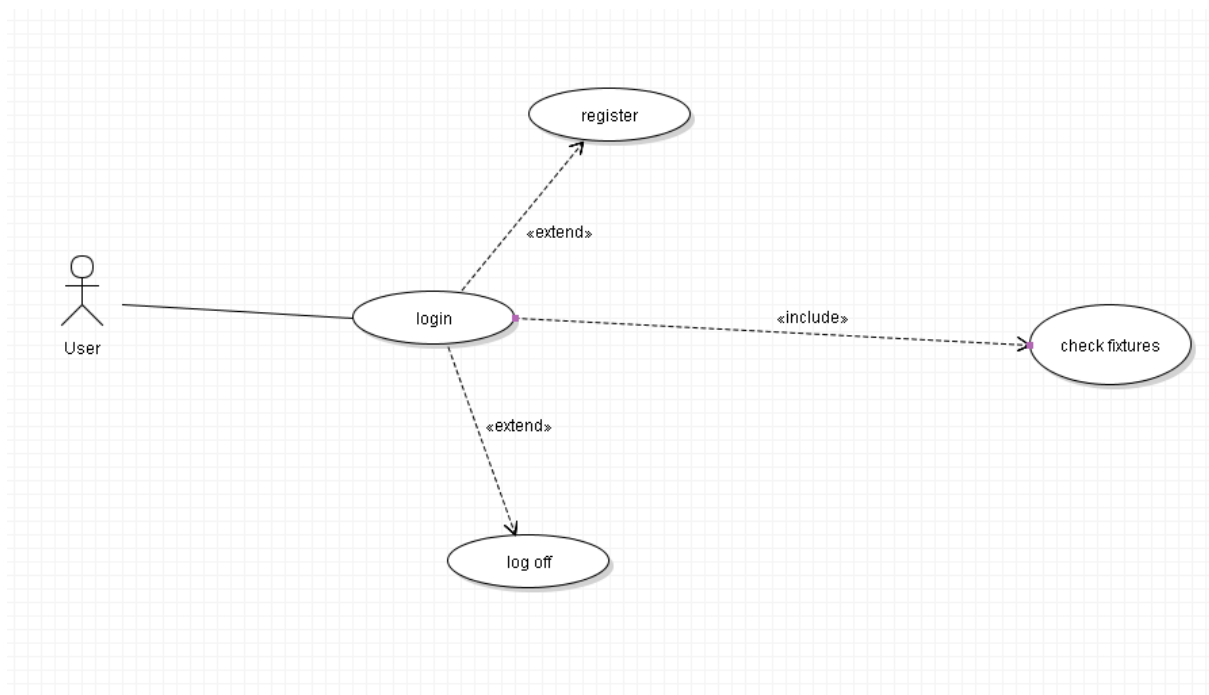


Figure 12 Use Case: User checks fixtures

User uses social media aspect

Use case specification:

1: User logs in

1a: User logs off

1b: User not registered and is prompted to register

2: User sends message

2a: User posts in the forum

2b: User sends an instant message

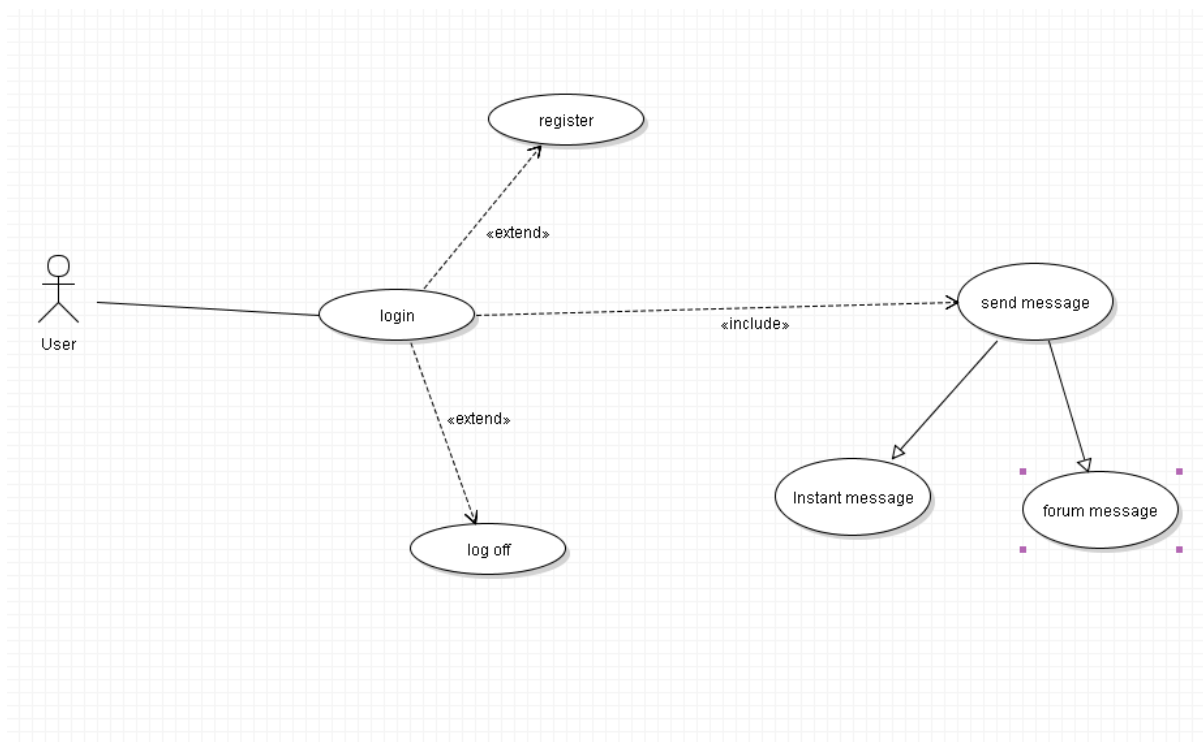


Figure 13 Use Case: User uses social media aspect

User creates team

Use case specification:

1: User logs in

1a: User logs off

1b: User not registered and is prompted to register

2: User makes team

3: User chooses squad

4: User chooses team name

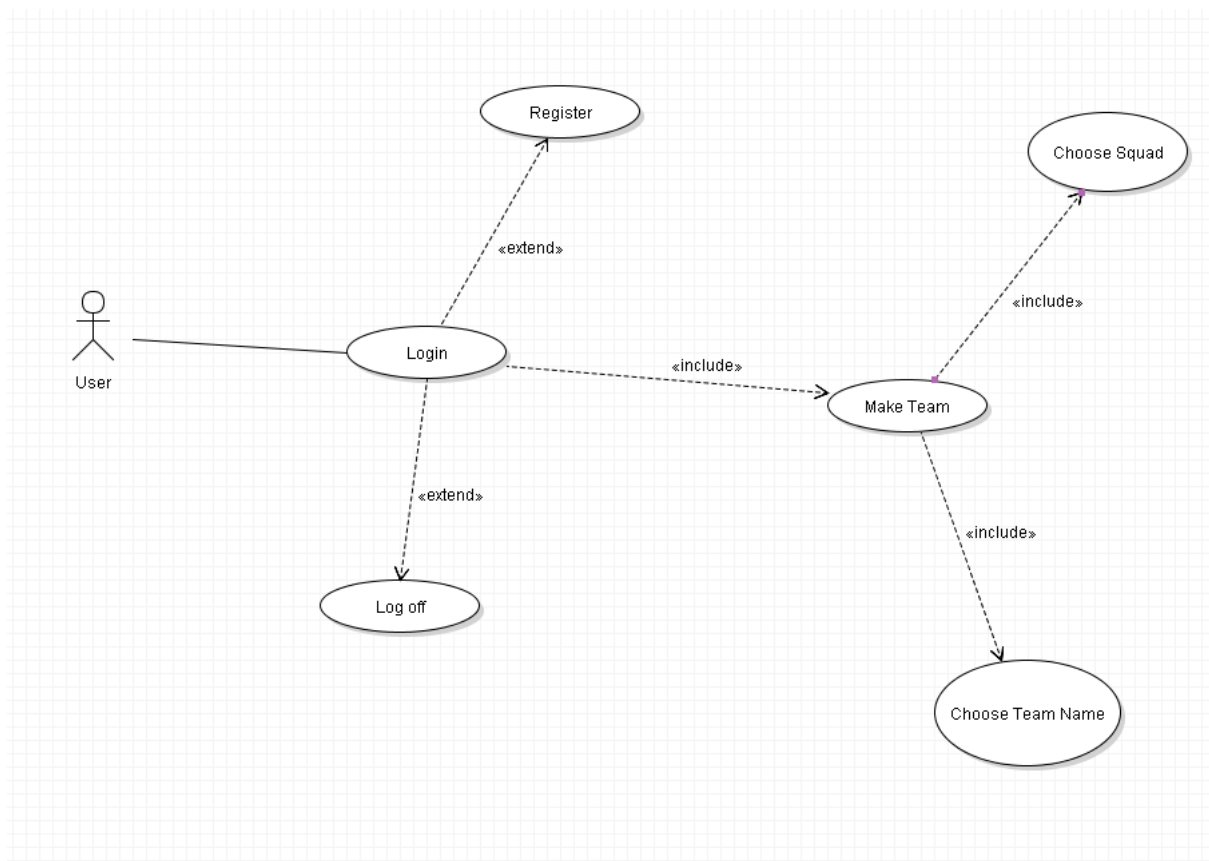


Figure 14 Use Case: User creates team

User makes transfer

Use case specification:

1: User logs in

1a: User logs off

1b: User not registered and is prompted to register

2: User makes transfer

2a: User has not enough funds to make transfer

2b: User selects and invalid team choice

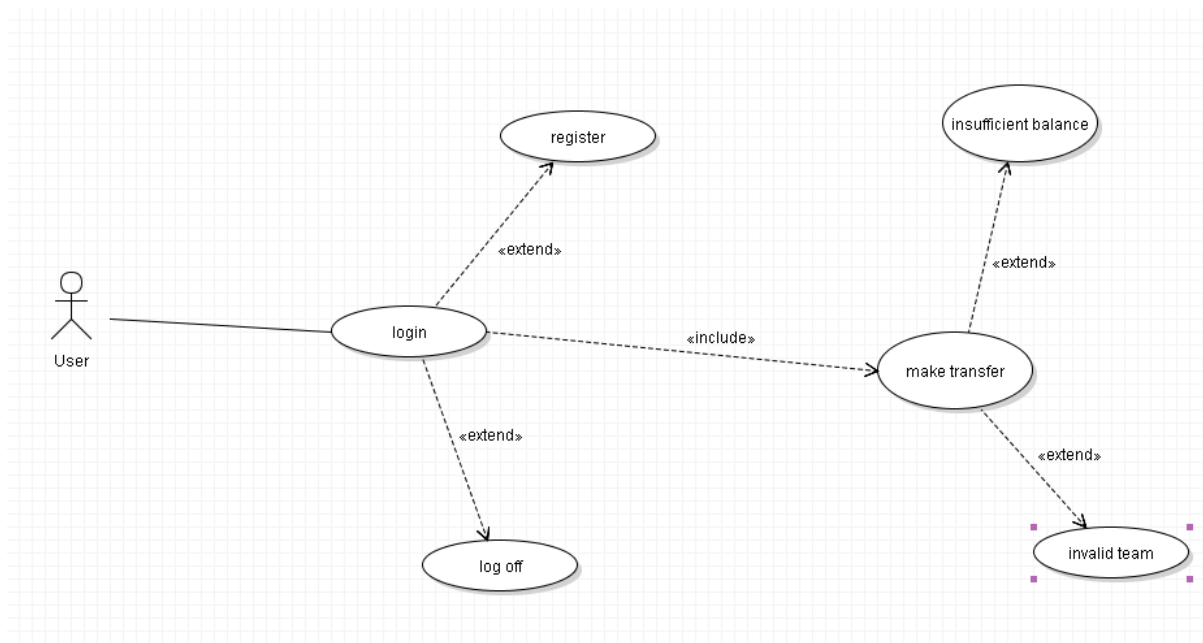


Figure 15 Use Case: User makes transfer

3.5 Sequence diagrams

User checks score

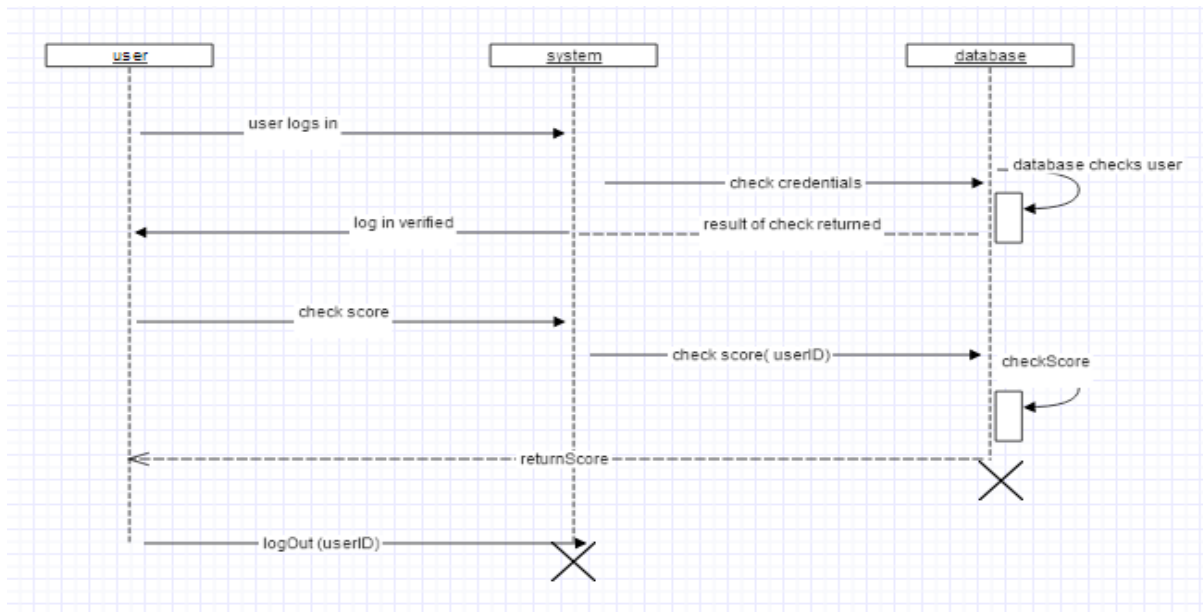


Figure 16 Sequence: User checks score

User checks fixtures

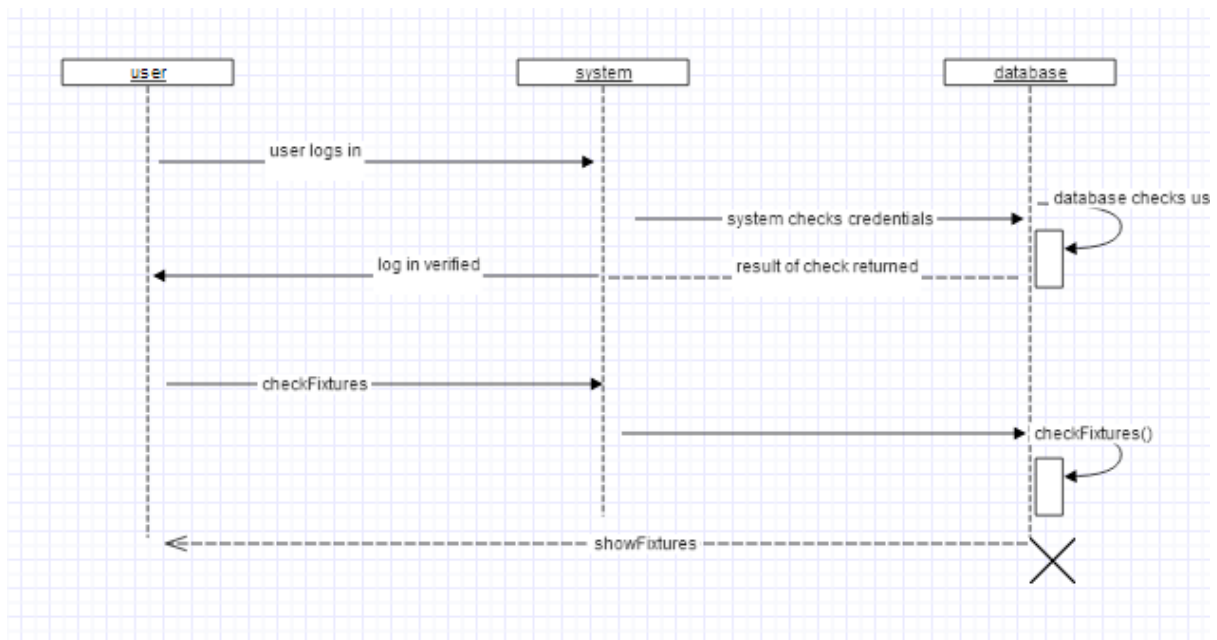


Figure 17 Sequence: User checks fixtures

User posts message

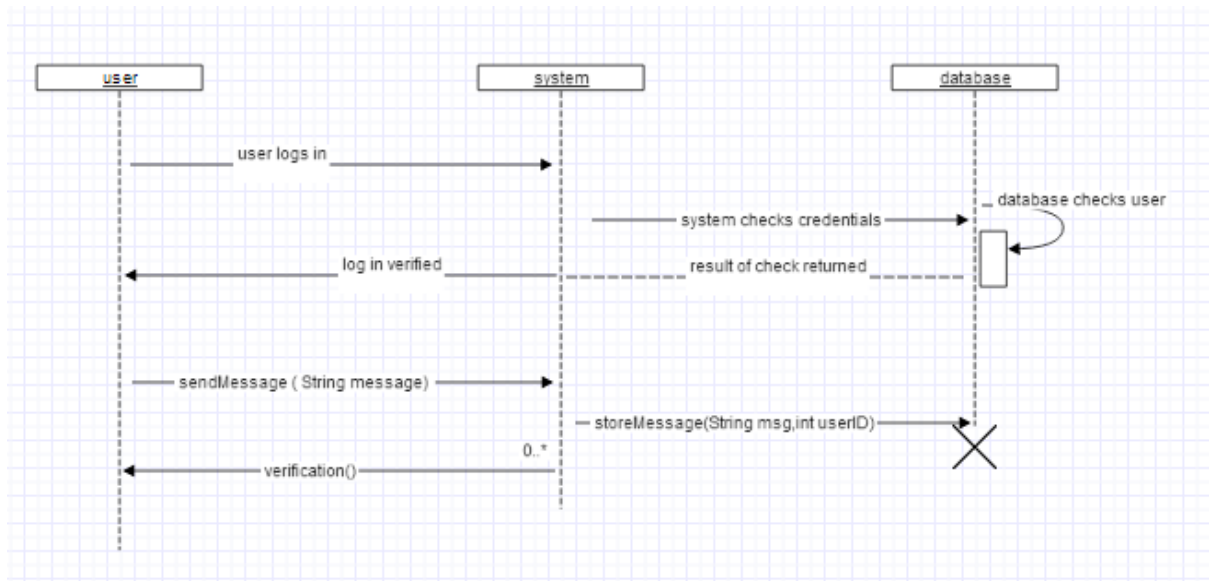


Figure 18 Sequence: User posts message

User chooses team

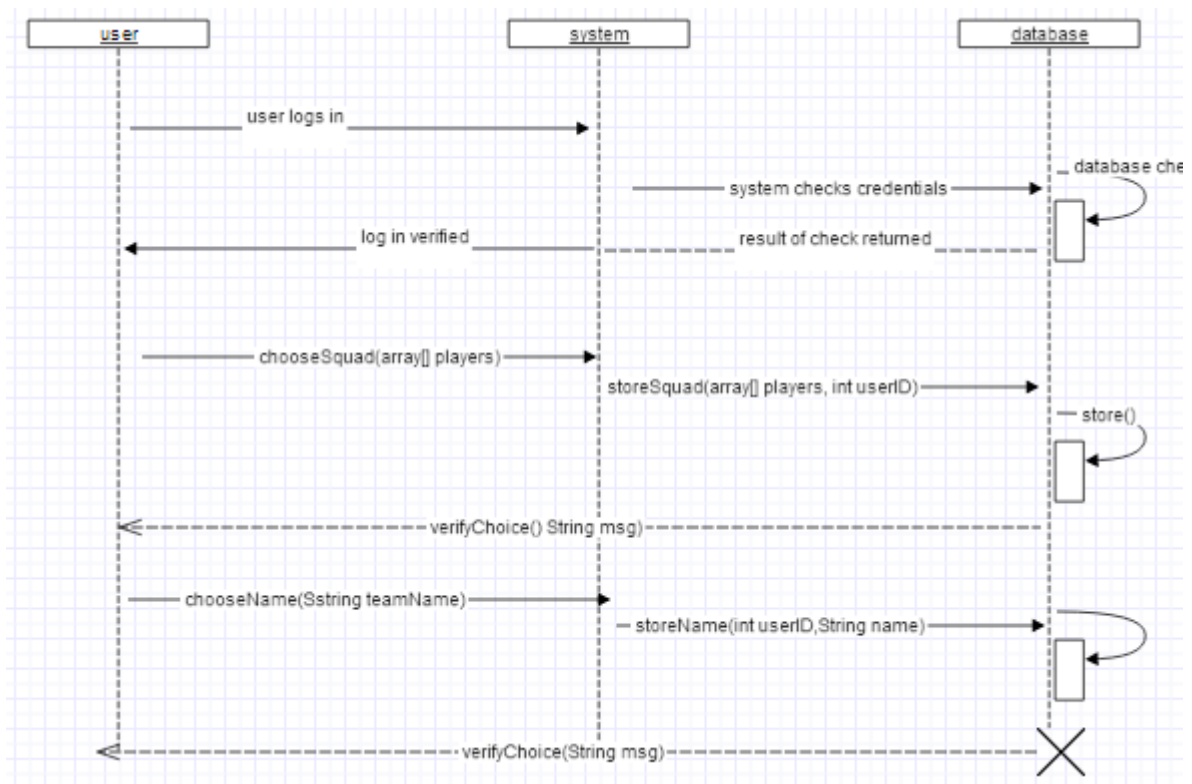


Figure 19 Sequence: User chooses team

User makes a transfer

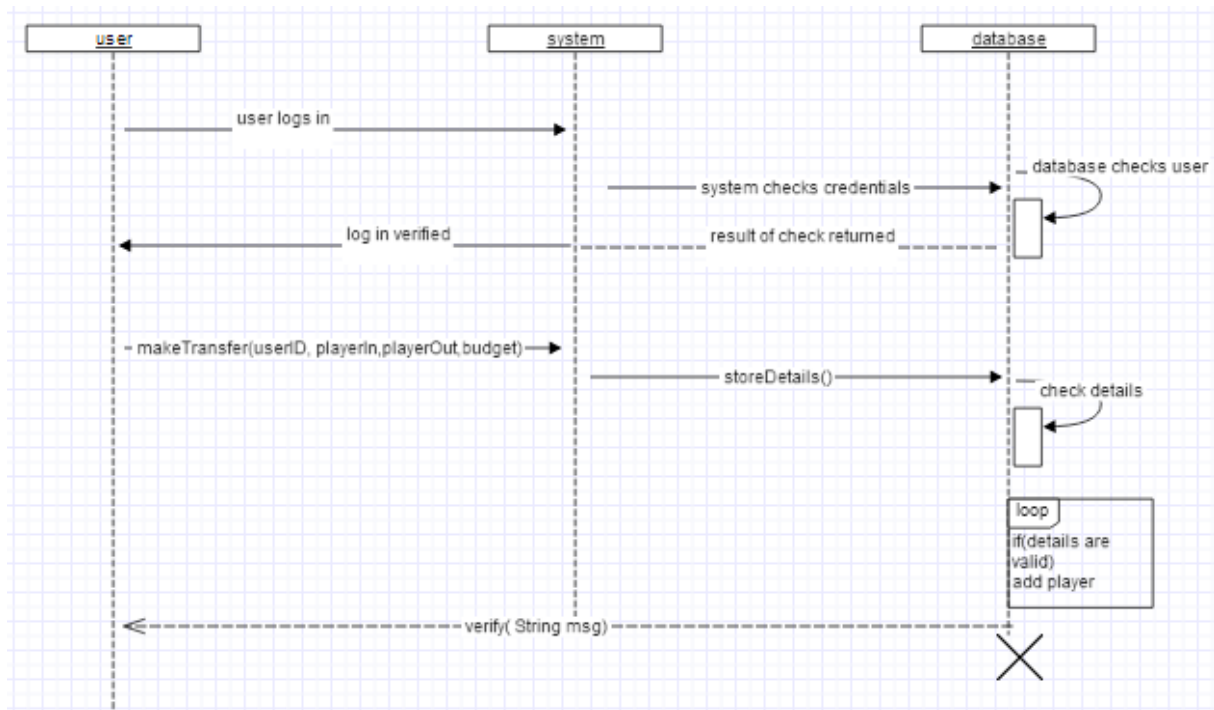


Figure 20 Sequence: User makes a transfer

3.6 User Interface Design (Wireframes)

Login Page

The wireframe shows a web browser window titled "Fantasy Hurling". The address bar contains navigation icons (back, forward, refresh, home) and a search bar with a magnifying glass icon. Below the browser window is a large rectangular area with a dashed 'X' across it, representing a placeholder for an image. Underneath this area are two text input fields. The first is labeled "User Name" and the second is labeled "Password". Below these fields is a button labeled "Register".

Figure 21 User Interface: 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

Fantasy Hurling

← → ↻ 🏠

Enter a Username

Enter a password

Enter a Team name

Enter your name

Enter an E mail address

Confirm

Figure 22 User Interface: 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

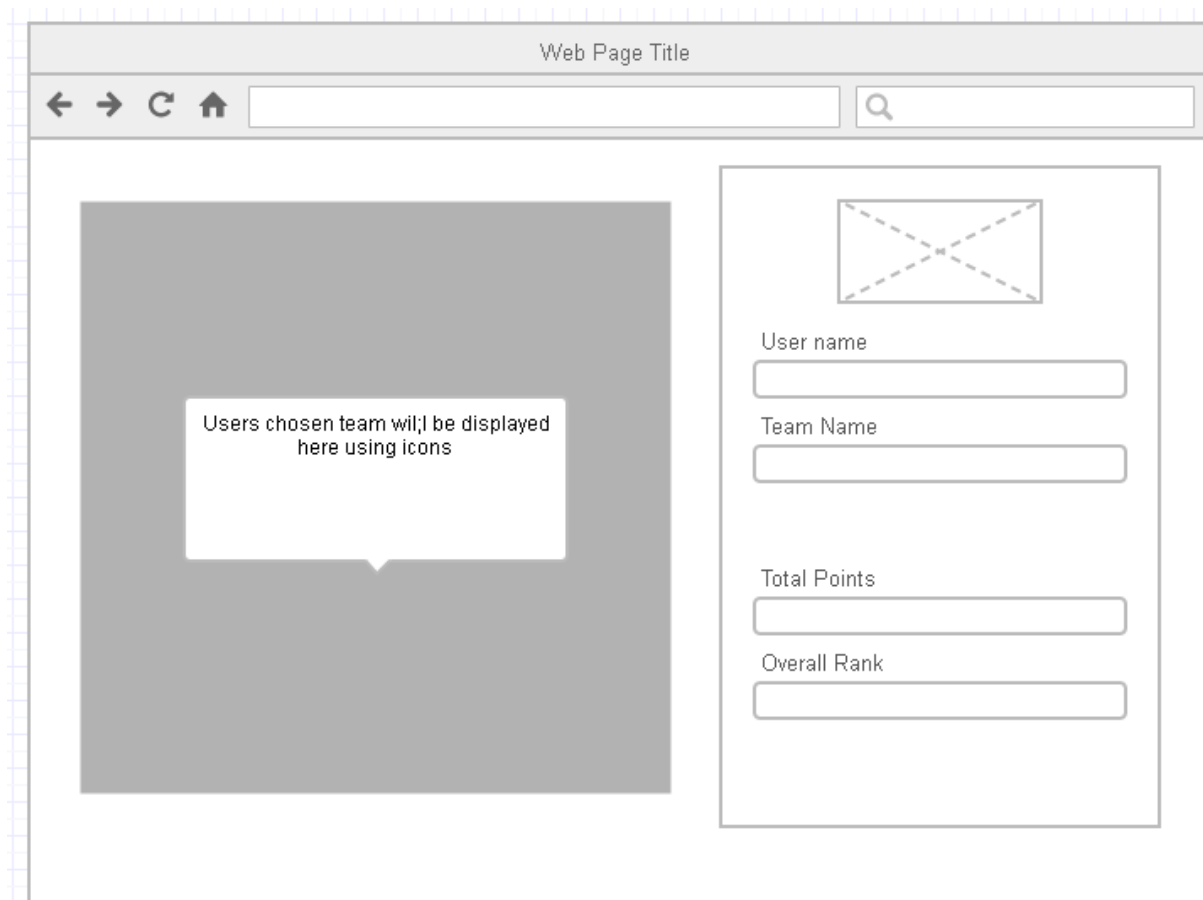


Figure 23 User Interface: Team statistics page

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

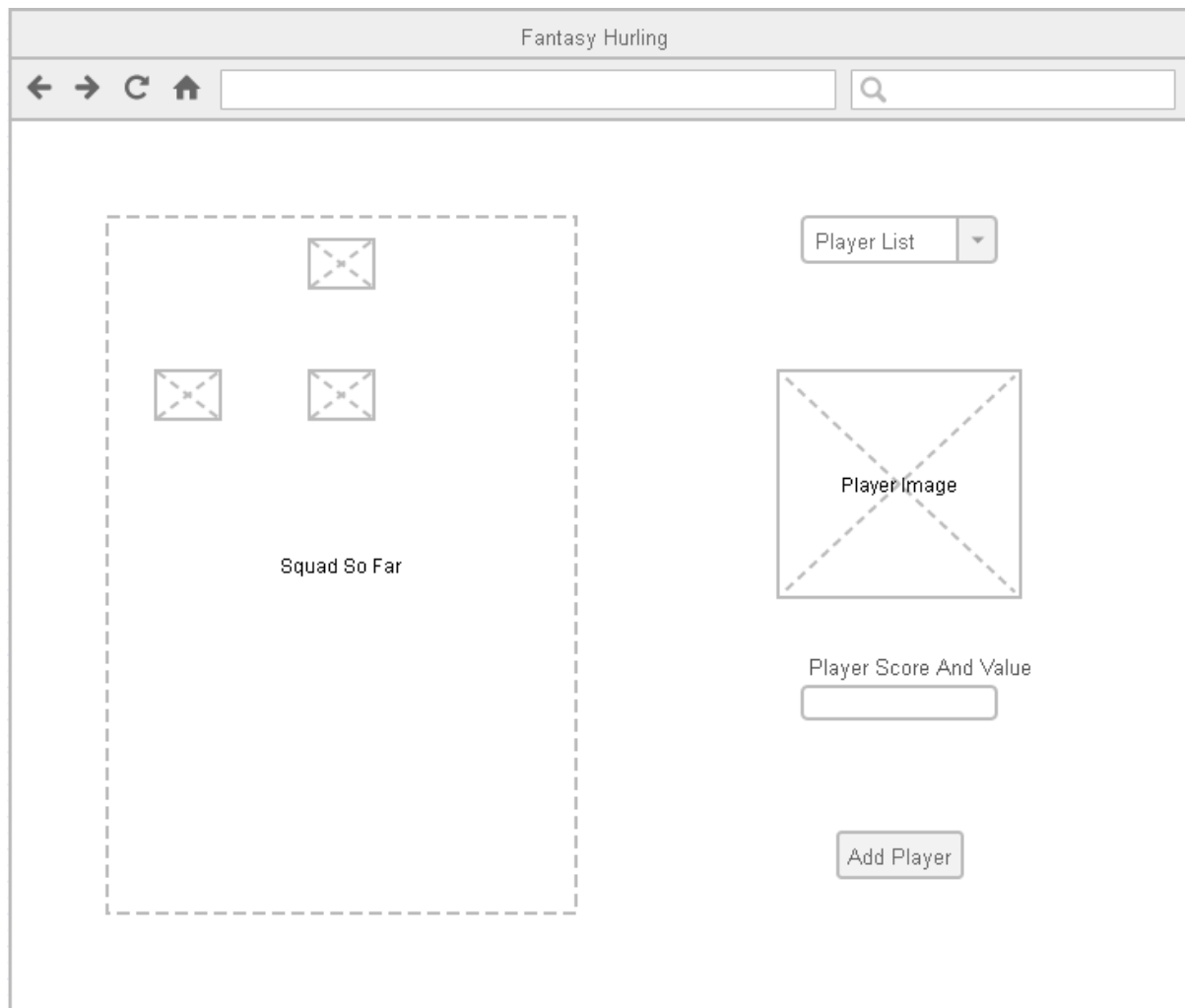


Figure 24 User Interface: Player transfer page

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 until 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

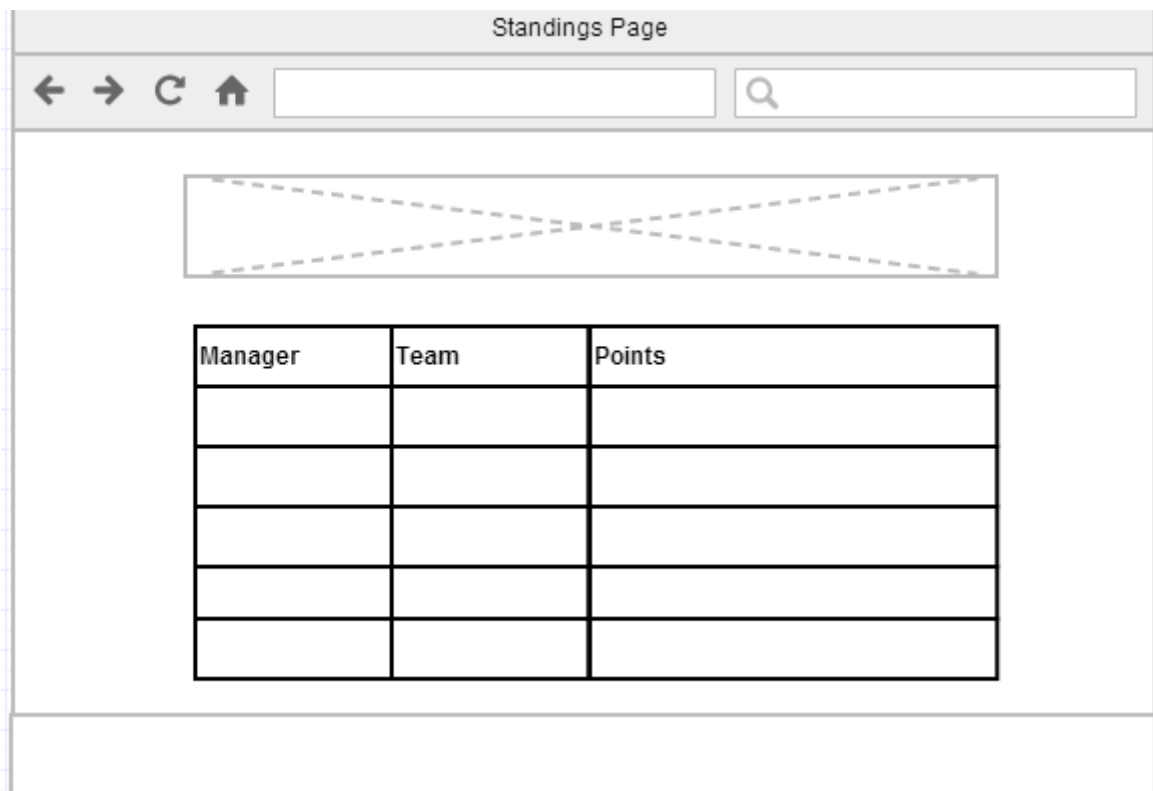


Figure 25 User Interface: Standings Page

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

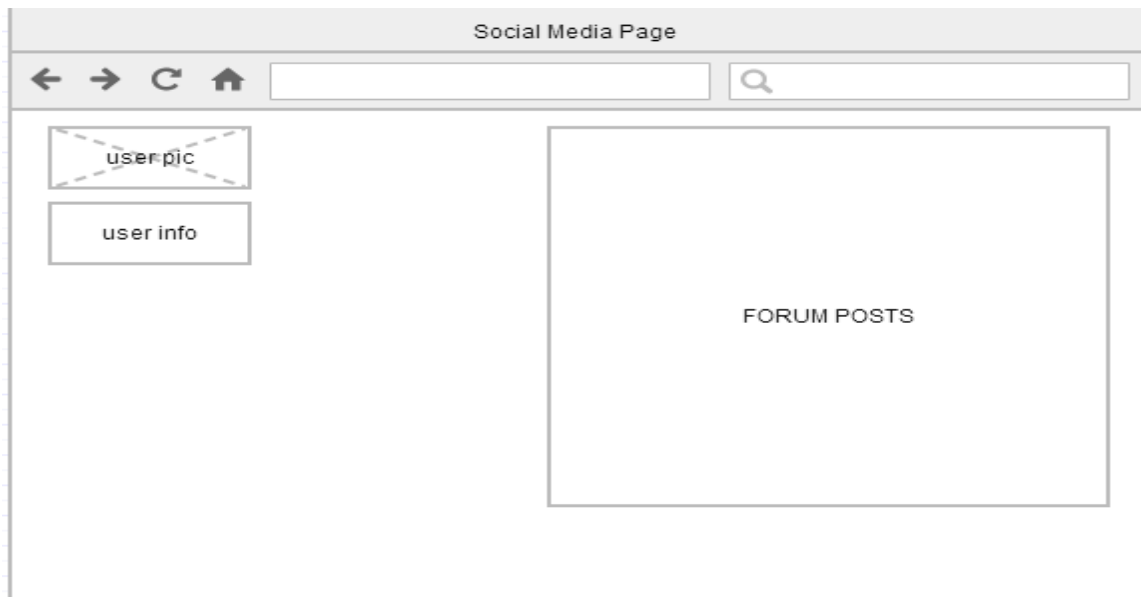


Figure 26 User Interface: Social Media area 1

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 also reply to the post.



Figure 27 User Interface: Social Media area 2

Elements needed:

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

3.7 Database Schema Design

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.

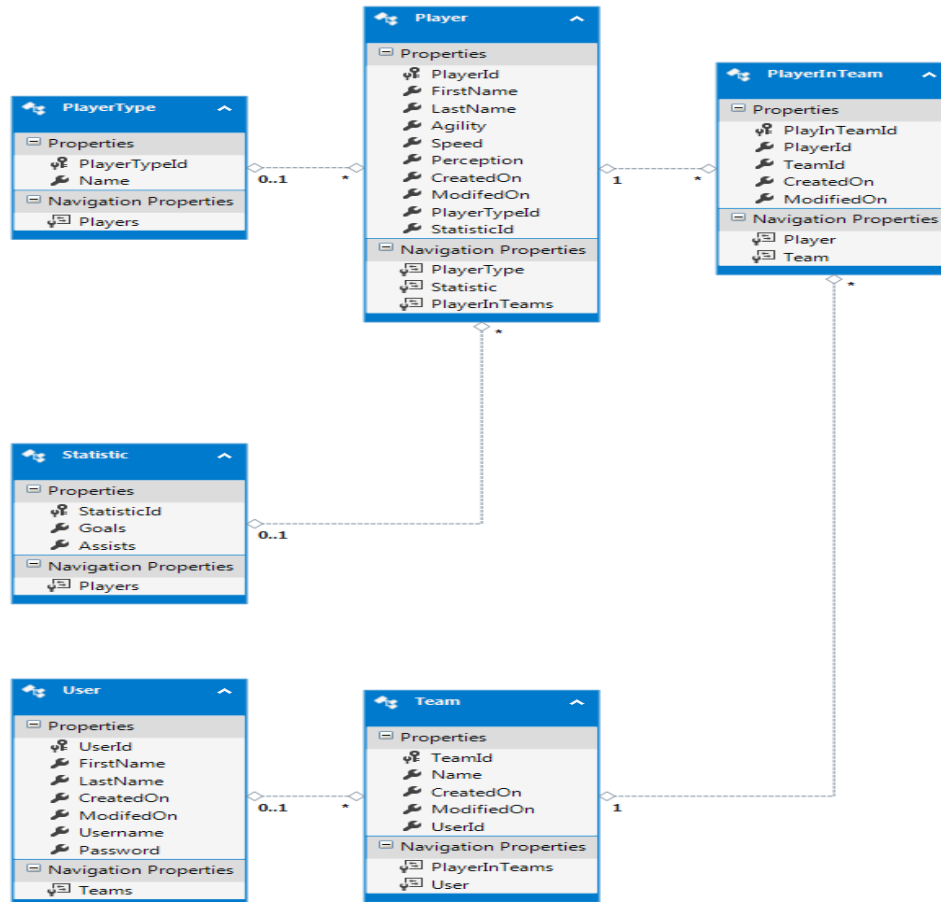


Figure 28 Fantasy Hurling database initial prototype

Database will be deployed on Azure Data Store (18). At the end of development data store gets migrated to Azure Europe North Datacenter located in Dublin (19).

3.8 Back End Design

Extensive research was done on web application back end development and we decided to build Web API Service (20). Our initial idea was to use Node.js framework (21) to build the API, but after additional research a decision was made to use ASP.NET Web API (22) 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.

3.9 Proposed Versioning Control System: GIT and Github

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

Fantasy hurling upstream repository is available at:

https://github.com/Michaelcj10/Fantasy_Hurling

Web API prototype upstream repository is available at:

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

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

3.10 Conclusion

On a project of this size it was very important to have a collective knowledge of the project technologies and ultimate design goals. The group needed to work together very closely on this stage of the project because this was the formative stage of the system. Using analysis and design models we learned in previous projects we began researching the technologies that we wanted to apply to this project.

In order to discover all the potential functionality of the website we employed UML diagrams. Using use case diagrams and sequence diagrams gave us an understanding of how users would and could interact with the system. Having primary paths and secondary paths for how a user would traverse the website, from simply logging in to check their scores to registering and creating a team. We needed to facilitate every path so the system would work with no errors. Creating all scenarios before moving on to another phase of the project was crucial to avoid potential failed systems.

The sequence diagrams showed how the system would interact with the users requests. As the users of the system are intrinsically linked to the database via HTTP requests and API requests, we needed to know the stages at which these interactions would happen. These sequence diagrams would be vitally important in the design of our database. Users must be able to get specific information related to their requests, but also users must be denied access to certain sections of the database. Users could not for instance request other users details or enter the administration section with invalid credentials. Before moving on to the design stage of the project including the database and the graphical interfaces for both the user and the administrator we needed to know what operations needed interaction from the user to call the database and get information sent back to the user.

After analysing the system and how all interactions would happen, we began the design phase.

Designing the website for the user was done with wire framing. The wireframe is a fundamental part of the design allowing us to visualize the users experience. Knowing what features the user need on each page of the website from our analysis. Our group had a clear understanding of creating a simple, easy to navigate website. Offering users a light layout with interactive buttons specific to the page they were visiting. A wireframe was created for each webpage with details on how many buttons and the name of the buttons that would be

needed. Although the final website may not follow the exact designs created during this phase, the wireframes gave our group a firm understanding of how the website would look.

The design of the database was one of the final stages in this phase of the project. The database needed to be built to accommodate all functions in the game. The functions were defined during the use case and sequence analysis. The database design comprised of game play features relating to how the game worked. The database also had to manage users and statistics. Defining all the relationships between users and the game allowed our group to implement a fully functional database that in essence is the core of the system. It was crucial that the database was correctly designed in order for the rest of the project to operate successfully.

Chapter 4: Implementation

4.1 Introduction

The implementation phase of this project is the natural next step after analysing and designing our application. Our group needed to convert the logic and design created in the analysis phase in to respective code for each technology we are using.

While implementing the application we divided the development in to three stages, back-end functionality, front-end functionality and front-end user experience. These three stages are all applied within our prototype life cycle model. To follow our prototype model we need to have at least the back-end functionality and front-end functionality as a basis to develop the application up on. With each iteration of our prototype life cycle we could then incorporate additional features and functionalities.

We began creating the back-end with a simple database confirming the relationships between classes functioned as we intended. As each iteration of the back-end was implemented succesfully, we added database tables and the API that would eventually connect the front-end to the back-end.

Whilst the back-end was being created work also began on the front-end functionality. Following the framework created in the analysis and design phase, the website was designed through HTML, CSS, JavaScript and JQuery. The key objective is to design the website with all the input values on each page to allow the front-end and back-end to communicate succesfully.

After achieving functionality on both sides of the system, the next phase of the implementation was to design the website and add visual features to enhance the end users experience. Key features included the layout of the website, the graphical design and the images.

Our project follows the prototype life cycle model so timelines for the creation of stages overlap and intersect. The rest of this chapter will elaborate on all stages of the implementation and how they developed over time to become final functioning application.

4.2 Methodology

As explained in design section, we choose to implement this application using Prototyping methodology. This approach allowed us to build a working prototype pretty early in the process. Because our application is a collaboration of many different web technologies, first step was to make sure we can actually make them work together on something remotely resembling fantasy sport game. If that's the case we would then mutate the working prototype toward the final project objective. During the each iteration we will add a new functionality or improve the existing one. Our project is a mutli-tier web application with a pipeline where data flows from persistent storage through server toward the clients and vice versa, we must simultaneously improve each tier.

Prototype Version 1 (November 2014)

1. A simple database with a few mock tables and simple relations between them was created in Microsoft SQL Server Compact on localhost.
2. Connection to database was assured.
3. New .Net Web API 2 with Entity Framework project was created in Visual Studio 2013 with a connection string to mock database.
4. One ORM Entity and Web API Controller with GET, POST, DELETE and UPDATE methods was created.
5. URLs to resources were crafted accordingly to a fantasy game like logic. The server will return JSON representation of requested table rows in the case of GET request or an appropriate http response for POST, DELETE, and UPDATE request.

Teams

API	Resource	Returns
GET api/teams	all teams	json
GET api/teams/id/{id}	team with given id	json
GET api/teams/name/{name}	returns a team with given name	json
GET api/teams/id/{id}/players	all the players from a team with given id	json
POST api/teams	inserts a team into the database	http response
DELETE api/teams/id/{id}	deletes a team with given id	http response

Figure 29 URLs to resources

6. Application was tested until worked
7. Simple HTML and JavaScript client which would consume the API was created.

8. JQuery Ajax function for GET, POST, UPDATE, and DELETE requests were crafted, tested and refactored to acceptable form.

```
//general ajax request, see http://api.jquery.com/jquery.ajax/  
//url : string - url of requested resource (mandatory)  
//successCallback : function (mandatory)  
//errorCallback : function  
//type : string - request type (default = "get")  
//dataType : string - data type (default = "json")  
//data : data passed in request body for "post" or "put " request (default =  
"undefined")  
var ajaxRequest = function(url, successCallback, errorCallback, type, dataType, data )  
{  
  
    type = typeof type === "undefined" ? "GET" : type;  
    dataType = typeof dataType === "undefined" ? "json" : dataType;  
    data = typeof data === "undefined" ? null : data;  
    errorCallback = typeof errorCallback === "undefined" ? function() {} :  
errorCallback;  
  
    $.ajax({  
        type : type,  
        url : url,  
        data : data,  
        dataType : dataType,  
        success : successCallback,  
        error : errorCallback  
    });  
}
```

Figure 30 JQuery Ajax Request Method

Prototype Version 1 Summary

- a “working prototype” of an AJAX application consuming REST like web service resources from the persistent storage

We knew we can do this on localhost. The next step was to make sure we can deploy this on the internet.

4.3 Prototype Version 2 (January 2015)

1. SQL Server was created on Azure Cloud North Europe datacentre and the localhost mock database was migrated there.

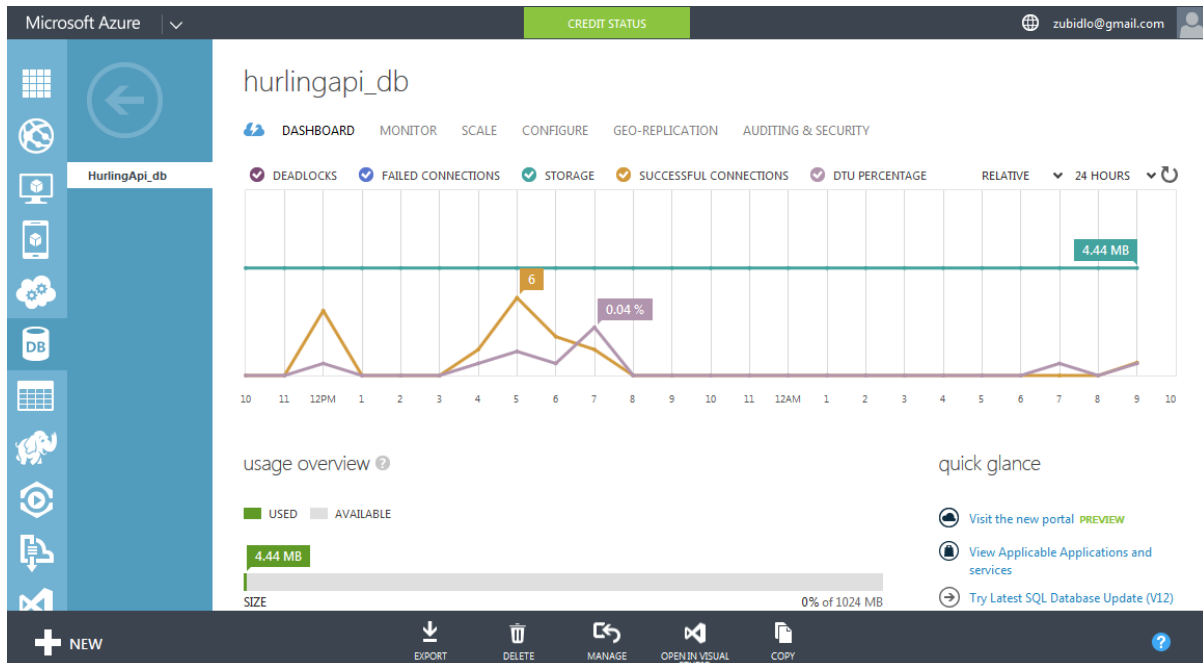


Figure 31 HulingAPI SQL server Azure

2. Visual Studio Web API project was configured with a new connection string to the database on the cloud.
3. A web server was created on Azure Cloud North Europe datacentre and Visual Studio was configured to publish Web API there.

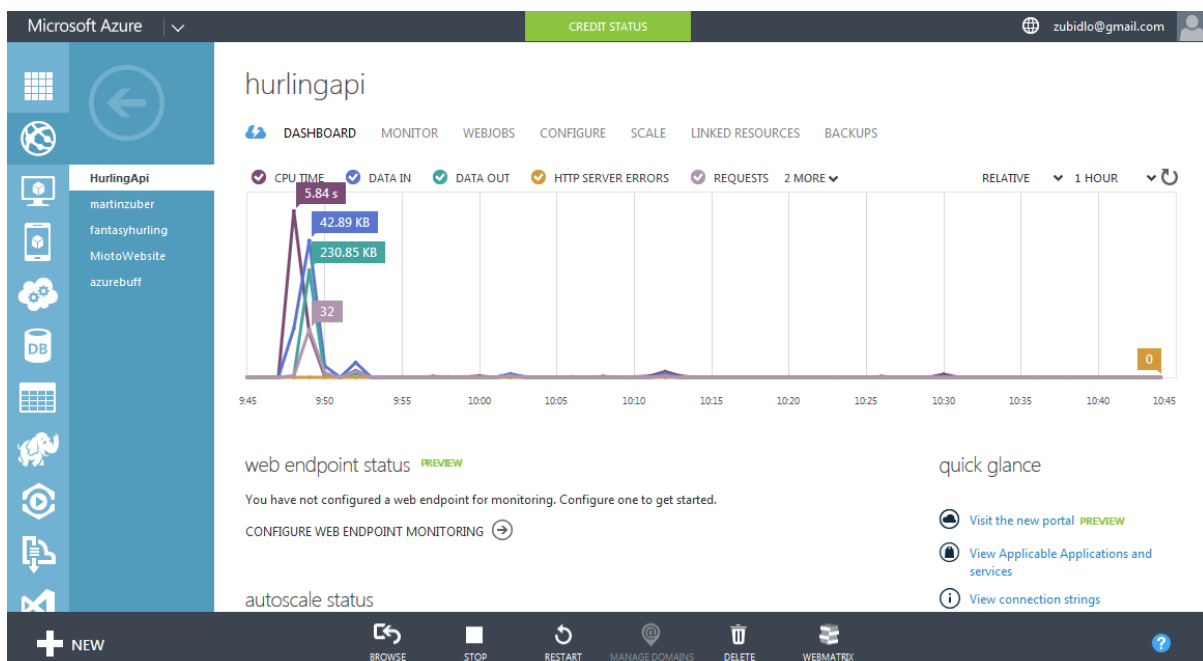


Figure 32 HurlingAPI Web server Azure

4. Deployment was tested.
5. Simple web forms and tables were created for our client. At this point we were turning the client into “database admin pages”. These will serve for populating the tables of the final version of fantasy hurling database.
6. At this point we faced first serious problem. Our application turned into Cross-Origin resource sharing (CORS) application. When the client running on localhost (Domain 1) requests the resources from Web API on Azure (Domain 2) the browser will deny the request.
7. After the research on CORS in .Net Web API, the CORS were allowed for each resource route.

```
namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/teams")]
    public class TeamsController : ApiController
    {
    }
}
```

Figure 33 CORS enabled for api/teams route

Prototype Version 2 Summary

- REST like web service deployed on the web
- simple admin pages client running on localhost

It was a time to start to turn this into ‘fantasy hurling’ application

4.4 Prototype Version 3 (January 2015)

1. With fairly good idea how 'fantasy hurling' should work, the database relations were created. It turned out there must be a many-to-many relationship between team and player entities, which added additional complexity to the project. But overall we design the database schema well and we never needed to change it in the future except for some cosmetic changes like renaming or adding the columns. So at this point we had final version of database deployed on Azure.

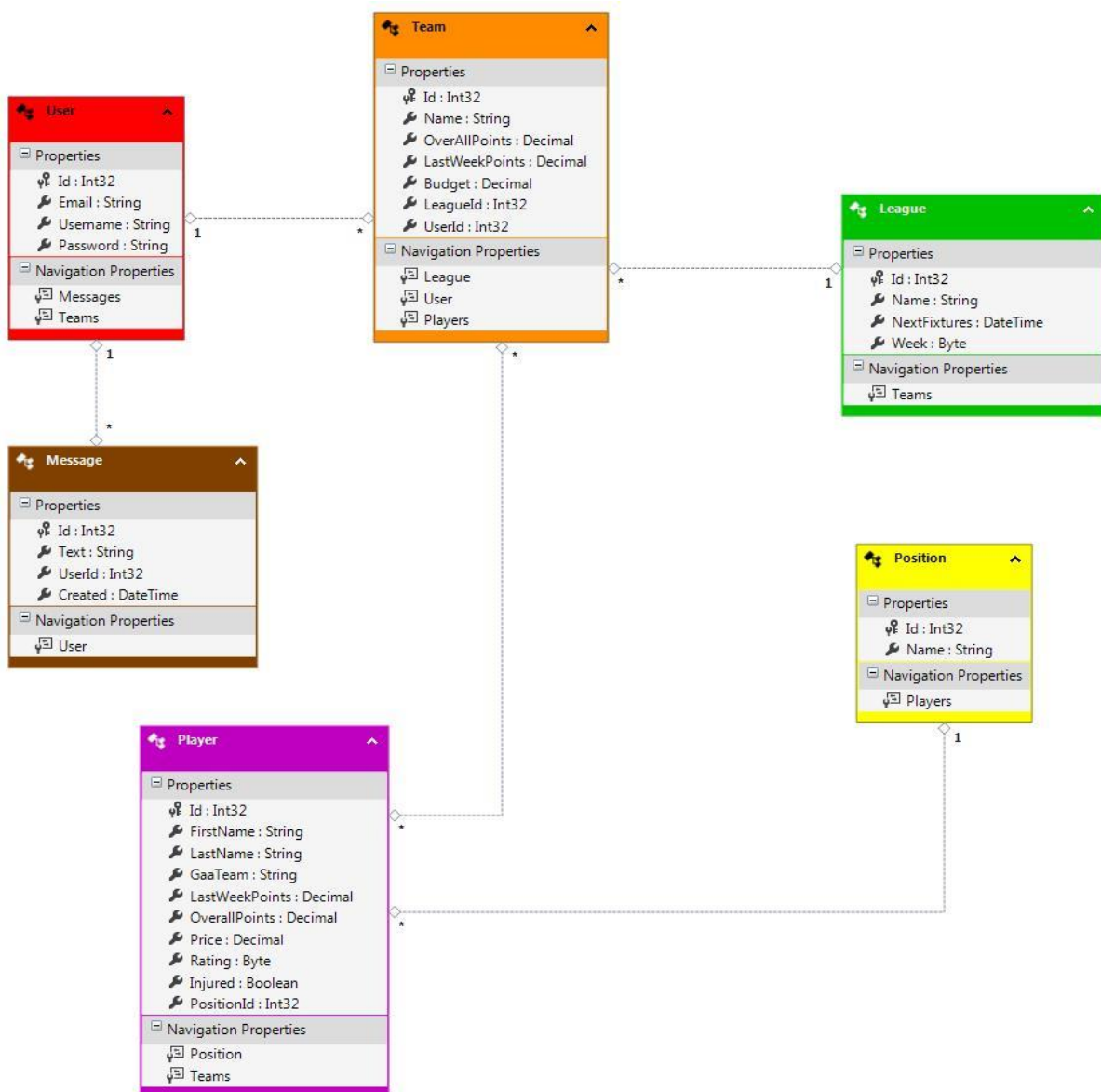


Figure 34 Fantasy Hurling Database Schema

2. Now the development process turned into the circles of
 - a) Populate a database table with few rows of data.
 - b) Mapping the table columns to plain old c# object POJO object with Entity Framework which is an Object-Relational Mapper (ORM)

- c) Writing the controller which would route the requests to POJO object or collection of POJO objects and return custom crafted http response with JSON representation of POJO object in the http message body.
- d) Creating client 'data admin page' for each database table.

USERS

MESSAGES

POSITIONS

PLAYERS

LEAGUES

TEAMS

LOGOUT

Team

Name:

Overall Points:

Last Week Points:

Budget:

League id:

User id:

Select the Request

Change(PUT)

Change(PUT)

Create(POST)

Delete(DELETE)

submit

Rows per page

10

submit

Repository

previous page

#	Id (PK)	Name	Last Week Points	Overall Points	Budget	League Id (FK)	User Id (FK)
11	1055	Administrator7	0	0	100000	1	7
12	1056	louis1027	0	0	90000	1	1027
13	1057	m19	0	0	91000	1	19
14	1058	bill1028	0	0	58000	1	1028
15	1059	podge1029	0	0	32000	1	1029
16	1060	sample string 2	0	0	50	1	1031
17	1061	sampfgd 2	0	0	90000	1	1032

next page

Number of items in the repository

17

Players in the team

Figure 35 Admin Page for Teams

3. There is 6 tables in our database so I needed to apply steps mentioned above for each and with each new iteration of the process I learned how to do it more efficient and code it more nicely. I faced a number of challenges during this process mainly because each step uses different languages and tools.
 - a) SQL insert queries
 - b) Entity Framework with C# with design patterns such as Abstract Factory, Repository. Visual Studio 2013
 - c) C# Web API 2 Framework with C#, LINQ C# extension, Visual Studio 2013
 - d) HTML5 , CSS3, JavaScript, JQuery, HTTP protocol
4. It became apparent that I will need to implement some kind of 'business logic' to the controllers in later prototype. For example if an administrator using admin pages client would like to delete an player which is registered in one or more teams the Entity Framework will throw exception with 3 page long stack trace. This exception will be then inserted into http response body and send back to client. So I will need to check for all such cases and create custom http responses like "you cannot delete this player because he is registered with one or more teams"
5. Other issue was 'cyclic entity references' when Entity framework will parse never ending data to JSON. For example a team has references to players and those have references to teams they play for and those have references to same players again and so on. This issue was handled by one more layer of object mapping where POCO entities are mapped to much simpler Data Transfer Objects (DTOs) and those are parsed to JSON and send to client.

Prototype Version 3 Summary

- Database in final form deployed on the web.
- Web API working prototype deployed on the web.
- Data admin pages prototype on localhost.

57

4.5 Prototype Version 4 (February 2015)

1. Custom http action results were created need for business logic.

```
public class ConflictActionResult : IHttpActionResult
{
    private readonly string _message;
    private readonly HttpRequestMessage _request;

    public ConflictActionResult(HttpRequestMessage request, string message)
    {
        _message = message;
        _request = request;
    }

    public Task<HttpResponseMessage> ExecuteAsync(CancellationToken
cancellationTokentoken)
    {
        HttpResponseMessage response =
_request.CreateResponse(HttpStatusCode.Conflict);
        response.Content = new StringContent(_message);
        return Task.FromResult(response);
    }
}
```

Figure 36 ConflictActionResult

2. These results will be returned by controllers instead of internal server error 500 with a stack trace.

```
//check if any player references this position
bool exist = await _repository.Players().ExistAsync(p => p.PositionId ==
id);

//if exists send bad request response
if (exist)
{
    return new ConflictActionResult(Request, "Can't delete this position,
because there are " +
                                "still some players referencing the position!");
}
```

Position

Name:

Select the Request:

Response

error: 409/Conflict: Can't delete this position, because there are still some players referencing the position!

Rows per page:

Repository

#Id (PK)	Position Name
1.1	Goalkeeper
2.2	Corner-Back
3.3	Full-Back
4.4	Half-Back
5.5	Midfielder
6.6	Half-Forward
7.7	Corner-Forward
8.8	Full-Forward

Number of items in the repository:

Figure 37 Conflict Result in code and action

3. Controller methods were turn into Asynchronous Tasks. This approach will not block the server thread.

```
[Route("id/{id:int}")]
[HttpGet]
public async Task<IHttpActionResult> GetPositionById(int id)
{
    Position position;

    //try to get requested position
    try { position = await _repository.Positions().FindSingleAsync(p => p.Id
== id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (position == null)
    {
        return new NotFoundActionResult(Request, "Could not find position id="
+ id + ".");
    }

    var positionDTO = _factory.GetDTO(position);
    //send ok response
    return Ok(positionDTO);
}
```

Figure 38 asynchronous method example

4. Web API 2 framework supports Open Data Protocol (OData). Controller methods can return OData query able collections. OData is powerful and well-crafted protocol that adds a lot of functionality for the web service. For example sorting, grouping, conditions, server side paging and much more.

API	Action
api/players?\$orderby=Price desc	Returns all players sorted by price descendant.
api/players ?\$filter=Price eq 10000.	Returns all the players with Price 10000.
api/players ?\$filter=Price lt 10000	Returns all the players cheaper then 10000.

Paging:

api/players?\$top=10&\$orderby=Price%20desc	Returns top 10 most expensive players.
api/players?\$top=20&skip=10&\$orderby=Price%20desc	Returns next 10 most expensive players

Figure 39 HurlingApi OData support

5. Database admin pages client was finished with highlight-able, click-able and page-able tables. Simple login dialog was created. Admin pages were deployed on new Azure web server.

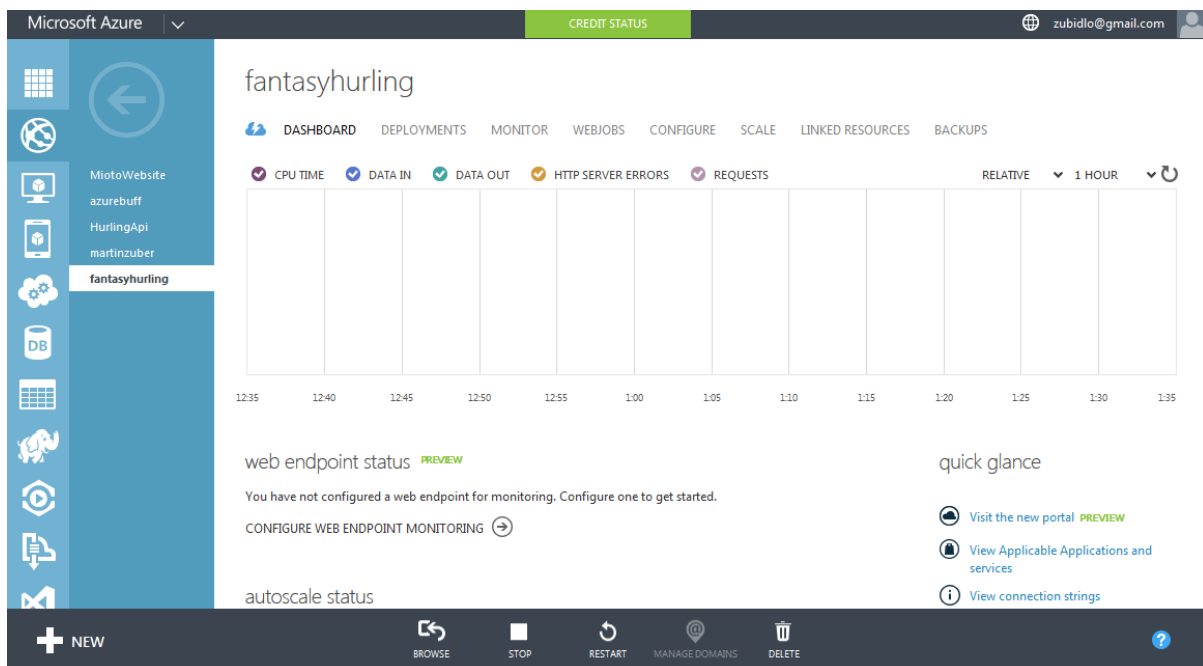


Figure 40 fantasyhurling Web Server dashboard Azure

Prototype Version 4 Summary

- Database in final form deployed on the web.
- Asynchronous Web API with OData support prototype deployed on the web.
- Database Admin Pages in final form deployed on the web.

4.6 Prototype Version 5 (February 2015)

1. With Database Admin pages deployed we started to populate database tables.
2. WebAPI business logic was finished and testing and debugging process on running service started. At this point we had an operational Web service, which could be considered as final product when properly debugged. There are some design decision, which should be reconsidered: I decided to tread following as additional functionalities and will be implemented only if time allows.
 - a) The business logic should be pushed one level lower to the repository dbContext. That way the controllers would be decoupled and could be tested more easily with help of Dependency Injection pattern.
 - b) Instead of manually mapping relation Entities to DTOs, an auto-mapper tool could be used.
 - c) There is a question of authentication. Web API 2 supports various methods over HTTP and HTTPS, OAuth2 protocol included. After some initial research I decided to leave the Web service without the authentication, because it would introduce a lot of complexity and we as a team are not equipped to handle such complexity just yet. The

web page clients will have their own authentication mechanism implemented using HTML5 session storage.

3. Web API have a default home page client with automatically generated help page. I decided to add 'API Test' client to it, which can be imported from NuGet package manager in Visual Studio 2013. This client will allow the front-end developers to see how craft the requests toward the API.

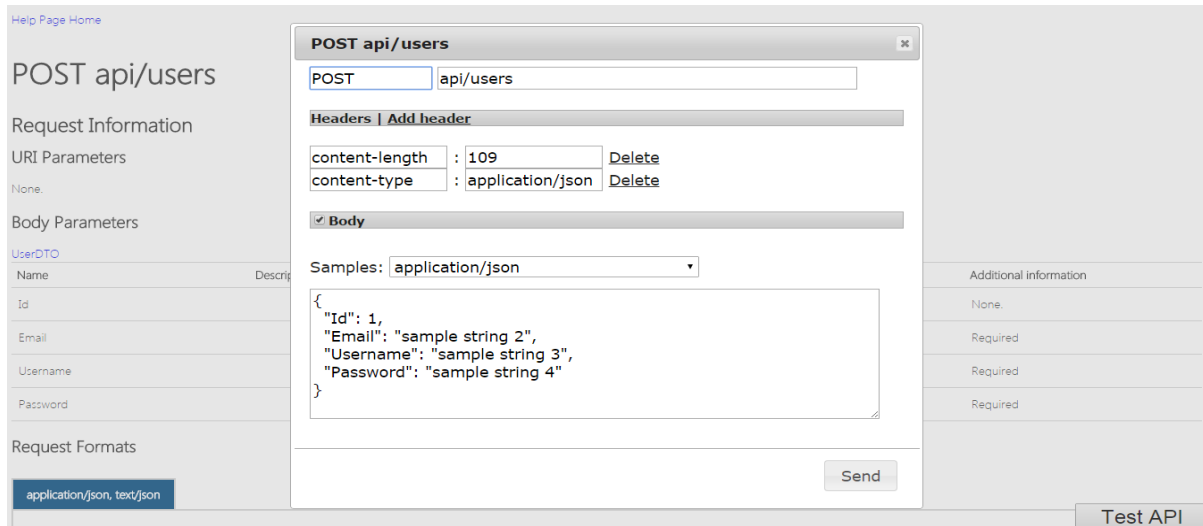


Figure 41 Test API client in action

Prototype Version 5 Summary

- Database in final form deployed on the web.
- Web API in final form deployed on the web:

<http://hurlingapi.azurewebsites.net/>

- Database Admin Pages in final form deployed on the web:

http://fantasyhurling.azurewebsites.net/consume_api_examples/data_admin/login.html

At this point we are ready for development of Fantasy Hurling Client

4.7 Fantasy Hurling Web Application Implementation

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

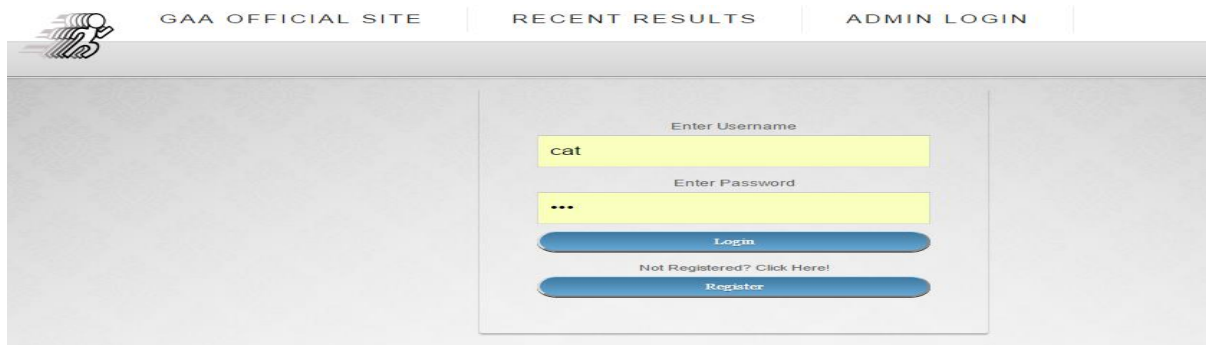


Figure 42 Login Dialog

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 those 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.

```

$_get_user_by_username_form.submit(function(event) {
    event.preventDefault();
    $.ajax({
        url: _url + "/username/" + $_user_username_field.val(),
        success: function(data, textStatus, request) {

            if ($_user_password_edit.val() == data.Password) {

                var pass = data.Password;
                sessionStorage.setItem("password", pass);
                var email = data.Email;
                sessionStorage.setItem("email", email);
                var id = data.Id;
                sessionStorage.setItem("id", id);

                window.location = "html/home.html";
                setStorage();
            } else {
                alert("Wrong password");
            }

        },
        error: function(request, textStatus, errorThrown) {

            $alert("Not found");
        }
    });
});

```

Figure 43 get user by username request

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.

```

function readUserFromInputFields() {

    var user = {
        Id: $_user_id_edit.val(),
        Username: $_user_username_edit.val(),
        Password: $_user_password_edit.val(),
        Email: $_user_email_edit.val()
    };

    return user;
}

```

Figure 44 register Form

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

```

function add_user() {
    $.ajax({
        type: "POST",
        url: "http://hurlingapi.azurewebsites.net/api/users",
        data: readUserFromInputFields(),
        dataType: "json",
        success: function(data) {
            getAllUsers();
            clearUserTextFields();
            alert("New User Created");
            window.location = "../index.html";
        },
        error: function(request, textStatus, errorThrown) {
        }
    });
}

```

Figure 45 add a user request

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 users 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.

```
function put_new_team() {  
  
    $.ajax({  
        type: "POST",  
        url: "http://hurlingapi.azurewebsites.net/api/teams",  
        data: readUserFromInputFields(),  
        dataType: "json",  
        success: function(data) {  
            alert("created new team");  
            location.reload();  
        },  
        error: function(request, textStatus, errorThrown) {  
        }  
    });  
}
```

Figure 46 insert a new team request

How to view the team

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

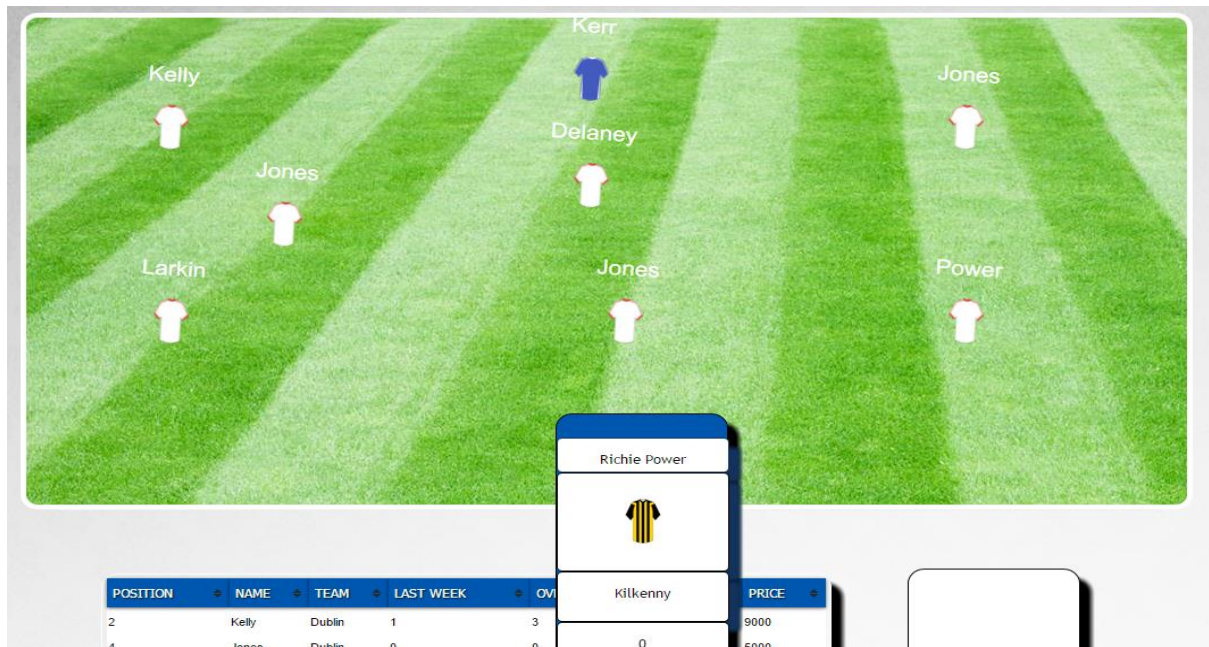


Figure 47 Player details

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.

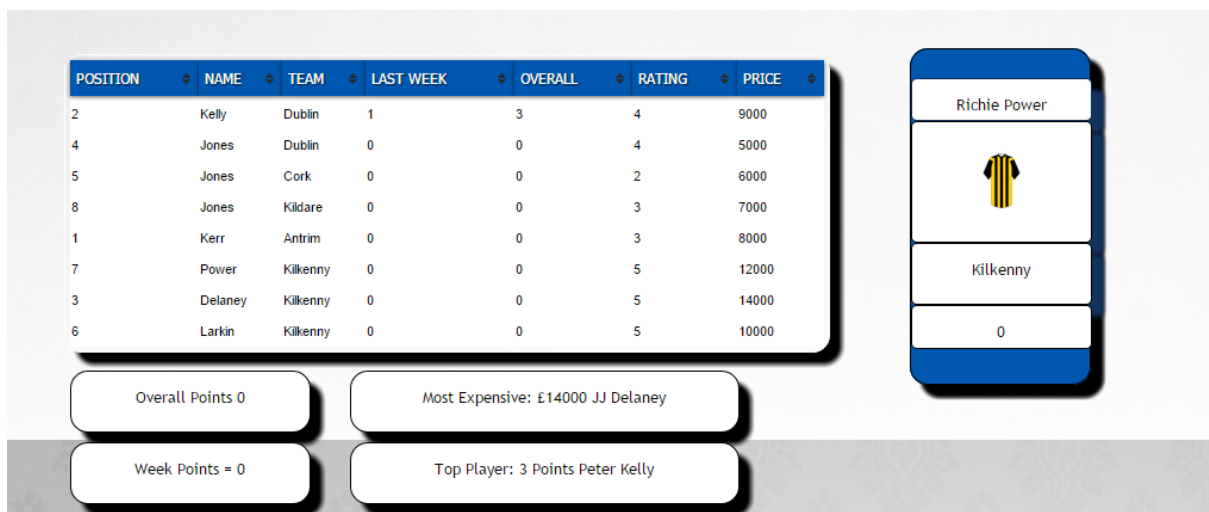


Figure 48 Team details

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.

```
function set_table() {
    var i = 1;
    var user_id = sessionStorage.getItem("teamid");
    var user = (user_id + "/players");

    var _url = "http://hurlingapi.azurewebsites.net/api/teams/id/" + user;
    var overall_points = 0;
    var week_points = 0;
    var most_expensive = 0;
    var overall_best = 0;

    $.ajax({
        url: _url,
        async: true,

        success: function(data) {
            if ($.isArray(data)) {
                $.each(data, function(index, object) {
                    var tr;

                    tr = $('<tr>');
                    tr.append("<td>" + object.PositionId + "</td>");
                    tr.append("<td>" + object.LastName + "</td>");
                    tr.append("<td>" + object.GaaTeam + "</td>");
                    tr.append("<td>" + object.LastWeekPoints + "</td>");
                    tr.append("<td>" + object.OverallPoints + "</td>");
                    tr.append("<td>" + object.Rating + "</td>");
                    tr.append("<td>" + object.Price + "</td>");

                    week_points = week_points + object.LastWeekPoints;
                    overall_points = overall_points + object.OverallPoints;
                    $('#table_1').append(tr);
                    $('#table_1').tablesorter();
                    var img = document.createElement("img");
```

Figure 49 Team by user id request

```
if (object.PositionId == 1) {
    img.src = "../images/jerseys/Cavan.png";
} else if (object.PositionId > 1) {
    img.src = "../images/jerseys/Tyrone.png";
}
var src = document.getElementById("" + object.PositionId);
src.appendChild(img);
document.getElementById("player_name" + object.PositionId).innerHTML = (" " + object.LastName);

if (object.Price > most_expensive) {
    most_expensive = object.Price;
    document.getElementById("top_value_player_info_box").innerHTML = ("Most Expensive: €" + most_expensive + " " + object.FirstName + " " + object.LastName);
}

if (object.OverallPoints > overall_best) {
    overall_best = object.OverallPoints;
    document.getElementById("top_player_info_box").innerHTML = ("Top Player: " + overall_best + " Points " + object.FirstName + " " + object.LastName);
}
```

Figure 50 Team Table implementation

Viewing the standings

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

You are logged in as cat

POSITION	TEAM NAME	WEEK POINTS	OVERALL POINTS
1	Middleagers	0	0
2	Dubliners	0	0
3	tim16	0	0
4	Frank23	0	0
5	beets24	0	0
6	cat22	0	0
7	pu26	0	0
8	Marvin1025	0	0
9	b1026	0	0
10	Administrator7	0	0

Your Position 6

Figure 51 User Standings

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.

```
function set_table() {  
  
    var i = 1;  
  
    var _url = "http://hurlingapi.azurewebsites.net/api/teams?orderby=OverAllPoints";  
  
    $.ajax({  
        url: _url,  
        async: true,  
  
        success: function(data) {  
  
            if ($.isArray(data)) {  
                $.each(data, function(index, object) {  
                    var tr;  
  
                    if (i < 11) {  
                        tr = $('<tr/>');  
                        tr.append("<td>" + i + "</td>");  
                        tr.append("<td>" + object.Name + "</td>");  
                        tr.append("<td>" + object.LastWeekPoints + "</td>");  
                        tr.append("<td>" + object.OverAllPoints + "</td>");  
                    }  
  
                    i++;  
                    $('#table_1').append(tr);  
  
                    $("#table_1").tablesorter();  
                });  
            }  
        }  
    });  
}
```

Figure 52 Users sorted by overall points request

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.

```
var url = _url;
var user = readUserFromInputFields();
var successCallback = function(data, textStatus, request) {

    ajaxRequest(_url, function(data, textStatus, request) {

        _count = parseInt(data.length);
        $table_rows_count.val(_count);
        getUsers(tableCurrentPage());

    });
    printOutput(textStatus, request);
}
var type = "PUT";
if (type === "PUT") {
    url = url + "/id/" + user.Id;
}

var dataType = "json";

ajaxRequest(url, successCallback, generalErrorCallback, type, dataType, user);

displayInfo2();
document.getElementById("change_password").value = sessionStorage.getItem("password");
document.getElementById("change_email").value = sessionStorage.getItem("email");
document.getElementById("password").innerHTML = ("Your password is :" + sessionStorage.getItem("password"));
document.getElementById("email").innerHTML = ("Your email is :" + sessionStorage.getItem("email"));
```

Figure 53 Update user request

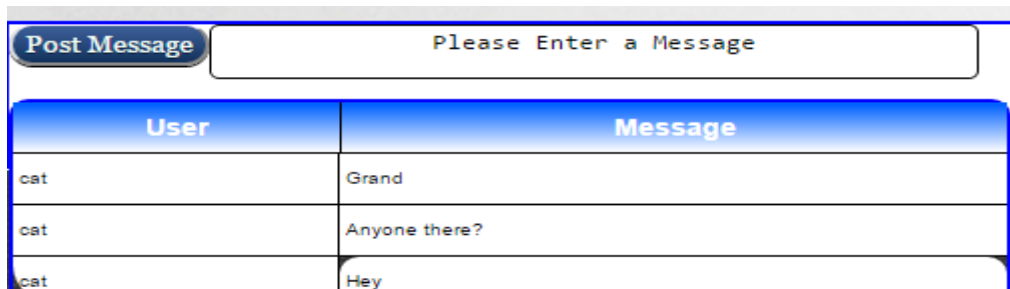
The page was kept simple with HTML DIV elements showing the current users details and text fields allowing the user to enter new details. On click of the button the request created a new JSON array of data, and PUT this data into the user's details via an AJAX request. Below is an example of the HTML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

The image shows a web form titled "Update User Dialog". It features three rounded rectangular boxes at the top, each containing a label and a value: "Your username is: cat", "Your password is: cat", and "Your email is: c@n.com". Below these are two text input fields. The first is labeled "Enter a new Password:" and contains the text "cat". The second is labeled "Enter a new Email:" and contains the text "c@n.com". At the bottom of the form is a blue button with the text "Update Details".

Figure 54 Update User Dialog

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.



User	Message
cat	Grand
cat	Anyone there?
cat	Hey

Figure 55 Chat Board

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.

```
unction set_table2() {  
  
    var user = sessionStorage.getItem("username");  
  
    var _url = "http://hurlingapi.azurewebsites.net/api/messages?orderby=Created desc&$top=3";  
  
    $.ajax({  
        url: _url,  
        async: true,  
  
        success: function(data) {  
  
            if ($.isArray(data)) {  
                $.each(data, function(index, object) {  
  
                    var user = return_username(object.UserId);  
                    var tr;  
                    return_username(object.UserId);  
                    var user = sessionStorage.getItem("temp");  
                    tr = $('<tr/>');  
  
                    tr.append("<td>" + user + "</td>");  
                    var user = sessionStorage.getItem("temp");  
                    tr.append("<td>" + object.Text + "</td>");  
  
                    sessionStorage.removeItem("temp");  
                    $('#table_2').append(tr);  
  
                }  
            }  
        }  
    });  
}
```

Figure 56 Messages sorted by creation time request

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 that 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.



Figure 57 making a transfer

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 and with the exception of the player deleted.

```
function delete_player(player_postion_id) {

    var type = "DELETE";
    var user_id = sessionStorage.getItem("teamid");

    var item = sessionStorage.getItem("" + player_postion_id);

    var _url = "http://hurlingapi.azurewebsites.net/api/teams/id/" + user_id + "/player/id/" + item;

    $.ajax({
        type: type,
        url: _url,
        async: true,

        success: function(data) {

            sessionStorage.removeItem("" + player_postion_id);

            setTimeout(function() {
                location.reload()
            }, 600);
        }
    })
}
```

Figure 58 remove a player from a team request

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.



Pos	Name	Team	Week	Overall	Rated	Cost	
5	McCrabbe	Dublin	0	0	4	11000	Add
5	Boland	Dublin	0	0	5	10000	Add
5	McCaffery	Dublin	0	0	3	6000	Add
5	Quilty	Dublin	0	0	3	8000	Add
5	Cronin	Dublin	0	0	4	9000	Add

Figure 59 Adding 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 of a budget and the position is vacant the player is added to the team.


```

function button(id) {
    var type = "PUT";
    var user_id = sessionStorage.getItem("teamid");

    var _url = "http://hurlingapi.azurewebsites.net/api/teams/id/" + user_id + "/player/id/" + id;

    $.ajax({
        type: type,
        url: _url,
        async: true,

        success: function(data) {
            setTimeout(function() {
                location.reload()
            }, 600);
        },
        error: function() {
            displayInfo3();
        }
    });
}

```

Figure 60 add a player to a team request

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 are 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.

```

function put_new_team() {
    window.alert = function() {};
    $.ajax({
        type: "POST",
        url: "http://hurlingapi.azurewebsites.net/api/teams",
        data: readUserFromInputFields(),
        dataType: "JSON",
        success: function(data) {
            alert("created new team");
        },
        error: function(request, textStatus, errorThrown) {
            window.alert(textStatus + ": " + errorThrown + ": " + request.responseText);
        }
    });
}

function readUserFromInputFields() {
    var name = sessionStorage.getItem("username");
    var name2 = name + ' ' + sessionStorage.getItem("id");
    var user = {
        Id: sessionStorage.getItem("id"),
        Name: name2,
        OverAllPoints: 0,
        LastWeekPoints: 0,
        Budget: 100000,
    };
}

```

Figure 61 insert a new team request

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.

```

function checkSession_home() {
    if (sessionStorage.getItem("username") == null) {
        window.location = "../index.html";
    } else {
        window.location = "home.html";
    }
}

```

Figure 62 session storage 2

4.8 Graphics

Jerseys

To enhance the user experience on the front-end interface of the application, we wished to create visual representations of current GAA hurling jerseys for each team. These jerseys would be interactive and give the user a graphical layout for their fantasy team. On the my team page, a user can hover their mouse over a jersey and the real life players information would be displayed in an intuitive design. On the transfers page, when a user wishes to add a player to their team they select a player from the available options in the transfer table. The transfer table is directly related to the graphics on screen and the player they selected is added to their team by representation of their team county jersey and the player name above. If a user wishes to remove an existing player from their team an intuitive command is in place. The user clicks on the jersey representing the player the user wishes to remove and the users team table is updated with the player no longer existing in the team.

Giving the user a mouse click input to interact with their team creates a more fluid and user-friendly experience. By removing a select, type and confirmation button to edit a user team the process of editing a fantasy team becomes fast and easy.

Our group was aware of the trademark and copyright connected to each GAA jersey. These factors made it important for us to recreate look alike versions of real life jerseys. These look alike jerseys would side step the obstacle of copyright law and prevent us from infringing on existing trademarks and licenses between the GAA, other third parties and the licensed GAA jersey manufacturers.

To begin the process of making look alike version of the real GAA county jerseys we needed to catalogue each team unique designs. We undertook the process of collecting images for all current GAA County hurling jerseys.

We realized after collecting all the jerseys that there are many different styles but actually only two different designs impacting the jersey cuffs and collars. This meant that we would need two mock up jerseys as a blank canvas to build the styles upon. The template jersey design is shown in the following figures.



Figure 63 round collar jersey design



Figure 64 V-neck collar jersey design

Following these templates as guide for all county jerseys we are then able to implement designs for each unique county. The GAA has 35 active hurling counties with their own unique colors and designs. 32 counties are from Ireland and 3 external teams.

To create a jersey we used Adobe Photoshop. Photoshop has a layer tool that allowed us to build our designs either the round collar design or the V-neck design. We then needed to reference each real life jersey individually to create a look alike version for the website. We have provided one example of a real jersey and the resulting look alike version for our fantasy hurling website.



Figure 65 GAA Carlow hurling jersey manufactured by O'Neills



Figure 66 Fantasy hurling website representation of GAA Carlow hurling jersey

Using color overlays, layers, effects and shape tools within Photoshop we were able to recreate look alike jerseys for each team participating all GAA hurling competitions. The process involves loading the template jersey design and adding the unique style features individually. Initially we would add a base color for the most prominent color on the real jersey. Then iteratively add color to the collars, sleeves, stripes and shapes.

The look alike jerseys used in the application are void of any manufacturer logos and jersey sponsorships. The final images are clean and simple. Any fan of hurling can clearly distinguish each county jersey from another. The creation of these jerseys was key to the users experience while on my team and transfers webpage of the application. All jerseys created for the application are shown in the Appendix D section of this thesis.

GAA pitch background

The addition of images representing a hurling pitch is added to enhance the user experience. The ability to see player positions on the pitch is key to intuitive interaction while the user is selecting and customizing their team. Having a jersey to represent a player is one step in facilitating the user experience. We included an image of a hurling pitch to position players on depending on their real life hurling positions.

Other fantasy sports games also take this approach as it stops the user from referencing the players position ID and allows them to instantly assess which player positions on the pitch need updating. The image we used is a simple green pitch design shown below.

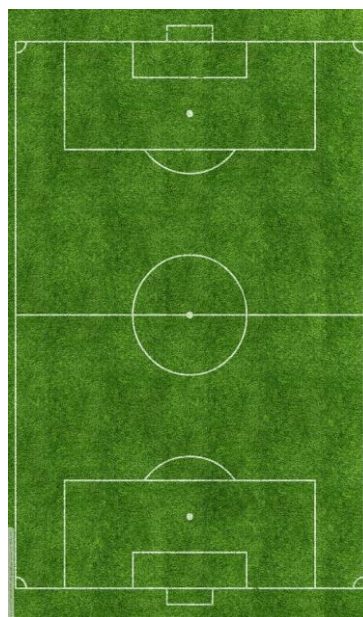


Figure 67 Fantasy hurling transfers page background for player positions

4.9 Conclusion

The implementation phase was a hugely important phase of the project. In order to successfully achieve all aspects of our research, analysis and design we spent a great deal of time working in the implementation phase of this project.

Following our SDLC we iteratively produced working components of our database and back-end administrator environment while implementing functionality with the front-end user environment. Our group focused on the merging of these two environments to have a fully functioning application. After creating a functioning application we then began adding design features and graphical aspects to enhance the users experience.

The prototype model proved to be the correct choice for our group as we are inexperienced developers. The ability to create, alter and adjust our application as we built it impacted the speed at which we were able to develop such a big project. Having each section of our project (back-end, front-end and user experience) functioning individually meant that the merging of all three environments was relatively simple with no major issues arising.

Learning the technologies to implement this application was a tough process but very enjoyable. The technologies on the back-end environment are very heavily code based and involved learning how to create a REST API in C#. The back-end also required an in depth knowledge of database relationships and database design. Achieving a functional back-end environment was the first step to completing the implementation phase.

The front-end environment needed to interact with the back-end database. Although we had functions created for the front-end user environment before the back-end was complete, it did not hinder our development. Having partly functioning environments was part of the SDLC we chose. After the completion of the back-end database the real work could begin on the front-end. Using technologies such as HTML, JQuery, JavaScript and CSS we merged the front-end to the back-end. This process was one of the most difficult processes in the implementation phase. Now the application had a user interface that could make requests to the database and retrieve information.

The final step involved enhancing the users experience on the front-end of the website. This required much research in to how users view and traverse websites. We used

examples of existing fantasy sports games to help our development. Making the users interaction intuitive and easily navigatable through all pages of the application was integral.

The entire implementation process was a large learning curve for our group. Working on a project of this scale as a member of a group proved to be a challengeing and new experience. To help document our contribution and prevent duplicate work being created, we develpoed our project through Github. Github offered a distributed version control system that allowed each member of the group to have up to date documentation.

Using Github to control our documentation and code greatly improved our individual ability to assemble each section of the project. During the imlementation stage we learned new technologies like C# and database design but also advanced our abilities with JQuery, JavaScript and CSS.

We feel that we succesfully implemented all the features we set in the analysis and design phase. The implementation phase was enjoyable to work on. Each member of the group was crucial to the overall success of the application.

Chapter 5: Testing and Evaluation

5.1 Introduction

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.

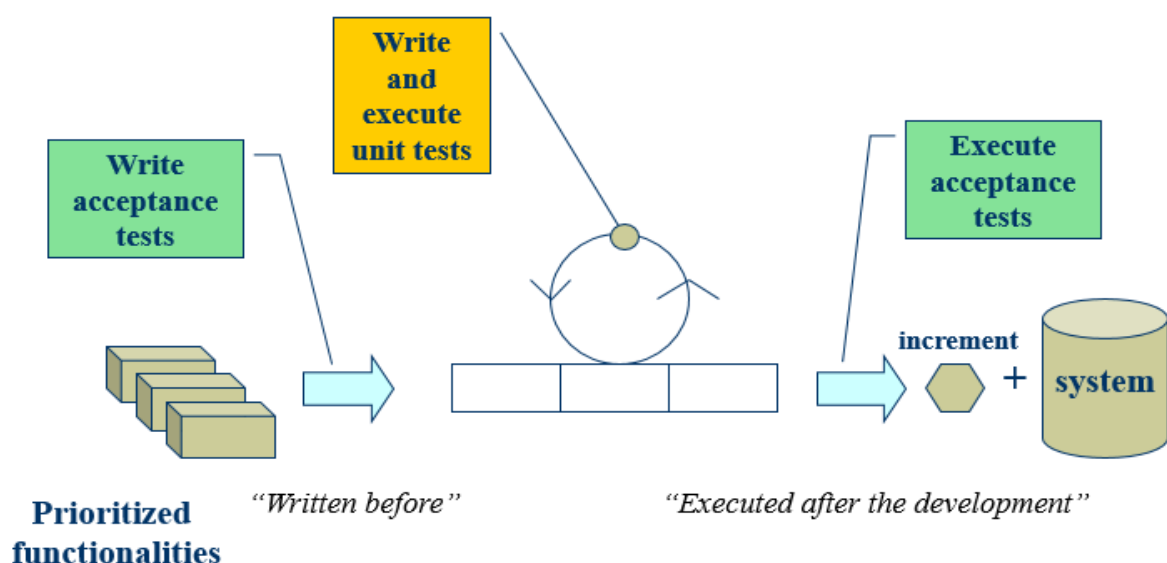


Figure 68 Testing Flow

5.2 Hurling Web API Unit Tests

Controller class overview

I'm going to perform unit tests on Hurling Web API controllers. First I will overview their responsibilities in the application design.

Controllers are responsible for routing http requests to asynchronous methods. For example HTTP GET request for **api/positions/id/1** resource will execute following controller method with number 1 passed as id method parameter.

```
public async Task<IHttpActionResult> GetPositionById(int id)
{
    Position position;
    //try to get requested position
    try { position = await _repository.Positions().FindSingleAsync(p => p.Id
== id); }
    catch (InvalidOperationException) { throw; }
    //if doesn't exist send not found response
    if (position == null)
    {
        return new NotFoundActionResult(Request, "Could not find position id="
+ id + ".");
    }
    var positionDTO = _factory.GetDTO(position);
    //send ok response
    return Ok(positionDTO);
}
```

Figure 69 GetPositionById (int id) method

This method and for that matter every controller method will perform the following operations:

- Try to find requested position in repository. Repository is Entity Frameworks interface to actual database.
- If not found it will return HTTP **Not Found** response
- Otherwise it will create new DTO object from found position and return HTTP **Ok** response

This is a simplest form of a controller method. Most of the controllers must perform some checking based on Fantasy Hurling game rules. For example if there is a request to DELETE a player from repository, the controller method must first check if the player is not registered in any teams.

Most of the controller methods are capable of creating HTTP responses other than **OK** or **Not Found**, for example above mentioned DELETE request will result in HTTP **Conflict** response with custom text message in response body.

To unit test Web Api controllers I will use Unit Testing Tool available in Visual Studio 2013. It's similar to Junit in Java ecosystem where I will write annotated test classes and test methods which can be then run from Visual Studio Menu and the results are display in a side pane.

First step is to create a Test Project in Hurling API Solution. This action will generate a test project scaffolding with first test class template.

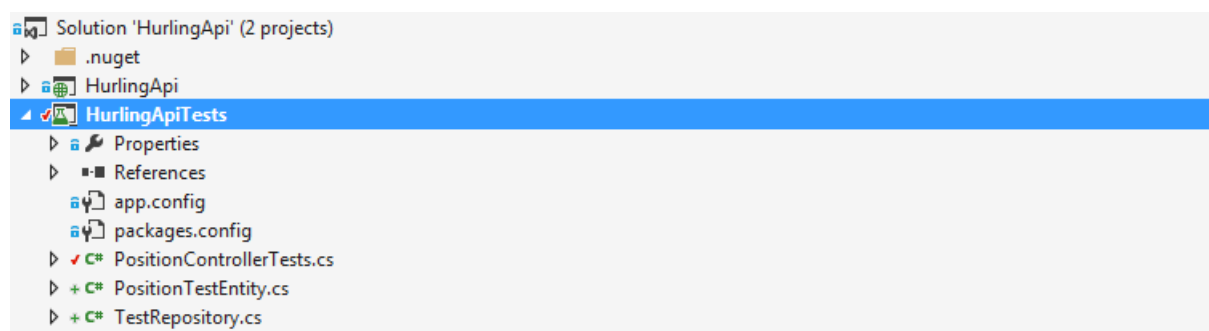


Figure 70 HurlingApiTests Visual Studio Project

A common pattern in unit tests is "arrange-act-assert"(Wasson 2014):

- Arrange: Set up any prerequisites for the test to run.
- Act: Perform the test.
- Assert: Verify that the test succeeded.

Every controller is dependent on a repository for the resources look-up. To mock an actual repository in unit tests I must first add a custom constructor to wire the mock repository to the constructor.

```
public class PositionsController : ApiController
{
    private IRepository _repository;
    private readonly PositionDTOFactory _factory = new PositionDTOFactory();
    private bool _disposed;

    public PositionsController() { _repository = new FantasyHurlingRepository(); }

    public PositionsController(IRepository repository)
    {
        _repository = repository;
    }
}
```

Figure 71 PositionController constructors

Now I can create a mock repository for testing purposes. The repository will implement all the interfaces as the real one, but will contain only a small amount of data. In Hurling API implementation I decided to create abstraction for a repository.

```
public interface IRepository : IDisposable
{
    IEntity<User> Users();
    IEntity<League> Leagues();
    IEntity<Player> Players();
    IEntity<Position> Positions();
    IEntity<Team> Teams();
    IEntity<Message> Messages();
}
```

Figure 72 IRepository interface

This has a number of methods which will return other abstraction for actual database tables called *IEntity*.

```
public interface IEntity<T> : IDisposable where T : class
{
    Task<IEnumerable<T>> GetAllAsync();
    Task<T> FindSingleAsync(Expression<Func<T, bool>> match);
    Task<bool> ExistAsync(Expression<Func<T, bool>> match);
    Task<int> UpdateAsync(T t);
    Task<int> InsertAsync(T t);
    Task<int> RemoveAsync(T t);
    Task<int> SaveChangesAsync();
}
```

Figure 73 IEntity interface

IEntity interface is an abstraction to actual CRUD (create, read, update, remove) SQL queries. So for example *FindSingleAsync* will return a table row based on given Boolean function. Notice that *IEntity* is a generic interface and forces asynchronous implementation for all the methods. *SaveChangesAsync* method is a proxy method to Entity Framework support for Database transactions. In case of updating, removing and inserting all changes must be saved or rolled back in a case of an error.

As mentioned above for testing purposes I will implement mock *IRepository* for overall testing project with mock *IEntities* for each controller. Let's do mock repository first.

```

namespace HurlingApiTests
{
    class TestRepository : IRepository
    {
        public IEntity<Position> Positions() { return new PositionTestEntity(); }
        public IEntity<User> Users() { throw new NotImplementedException(); }
        public IEntity<League> Leagues() { throw new NotImplementedException(); }
        public IEntity<Player> Players() { throw new NotImplementedException(); }
        public IEntity<Team> Teams() { throw new NotImplementedException(); }
        public IEntity<Message> Messages() { throw new NotImplementedException(); }
        public void Dispose() { throw new NotImplementedException(); }
    }
}

```

Figure 74 TestRepository implementation

Let's say the first controller to unit test is *PositionController*. *IEntity Positions ()* called *PositionTestEntity* must be created first. The rest of the *TestRepository* methods are left unimplemented for now. In *PositionTestEntity* I will implement each asynchronous method as testing progresses. First step is to test controller method routed to **api/positions**. This resource is routed to *PositionsController.GetPositions ()*.

```

public async Task<IQueryable<PositionDTO>> GetPostions()
{
    IEnumerable<Position> positions = await
    _repository.Positions().GetAllAsync();
    IEnumerable<PositionDTO> positionDTOs =
    _factory.GetDTOCollection(positions);
    IQueryable<PositionDTO> oDataPositionDTOs =
    positionDTOs.AsQueryable<PositionDTO>();
    return oDataPositionDTOs;
}

```

Figure 75 GetPositions () method

GetPositions () method is calling *Positions ().GetAllAsync ()* of wired repository. That's the first method I want to implement in *PositionTestEntity* mock repository entity.

```

class PositionTestEntity : IEntity<Position>
{
    private readonly List<Position> positions;

    public PositionTestEntity()
    {
        positions = new List<Position>();
        positions.Add(new Position { Id = 1, Name = "position1" });
        positions.Add(new Position { Id = 2, Name = "position2" });
    }

    public async Task<IEnumerable<Position>> GetAllAsync()
    {
        return await Task.Run(() => positions);
    }
}

```

Figure 76 GetAllAsync () method

GetAllAsync () method will simply return a list of two positions asynchronously. At this point I finally ready to write first unit test on *PositionsController*.

```
namespace HurlingApiTests
{
    [TestClass]
    public class PositionControllerTests
    {
        //Arrange
        private static readonly IRepository _repository = new TestRepository();
        private static readonly PositionsController _controller = new
PositionsController(_repository);
        [TestMethod]
        public async Task GetPositions_ShouldReturnAllPositions()
        {
            //Act
            var expected = await _repository.Positions().GetAllAsync();
            var returned = await _controller.GetPostions();
            //Assert
            Assert.AreEqual(expected.Count(), returned.Count());
        }
    }
}
```

Figure 77 *GetPositions_ShouldReturnAllPositions ()* unit test method

After running the test Visual Studio the results are displayed below.

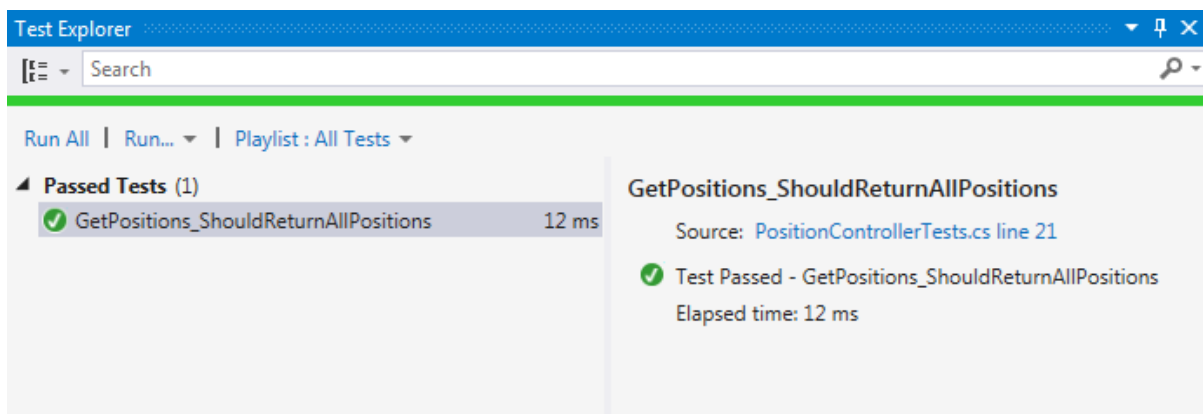


Figure 78 unit test results

First unit test passed. *PositionsController.GetPositions ()* returned the same resources that *TestRepository.Positions ().GetAllAsync ()* method. We cannot assume yet that the controller method is implemented without a bug. According to Wasson 2014 (16) these are the things I must test every controller method for:

- The action returns the correct type of response.
- Invalid parameters return the correct error response.
- The action calls the correct method on the repository or service layer.
- If the response includes a domain model, verify the model type.

Controller unit testing workflow overview.

In the section above I outlined my approach for unit testing of Hurling API controllers. I showcased the partial test applied to one method from one controller. In this section I will summarize what has to be done to properly test controllers. Each of the steps below must be applied to all controller methods for all controllers.

1. Create a controller custom constructor to be wired with any repository abstraction.
2. Create implementation for a method in *IEntity* mock implementation.
3. Wire it up in *IRepository* implementation.
4. Wire up a controller with mock *IRepository* implementation in test class.
5. In test class create test methods to test various corner cases on a controller method.

5.3 Hurling API Tests

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: Retrieve all users

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

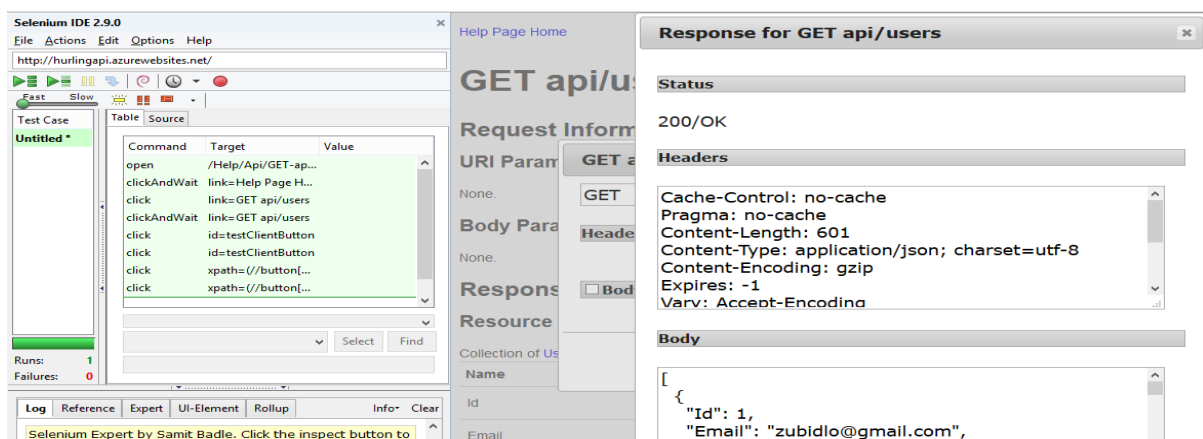


Figure 79 API get all users test

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: Input a player into team

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.

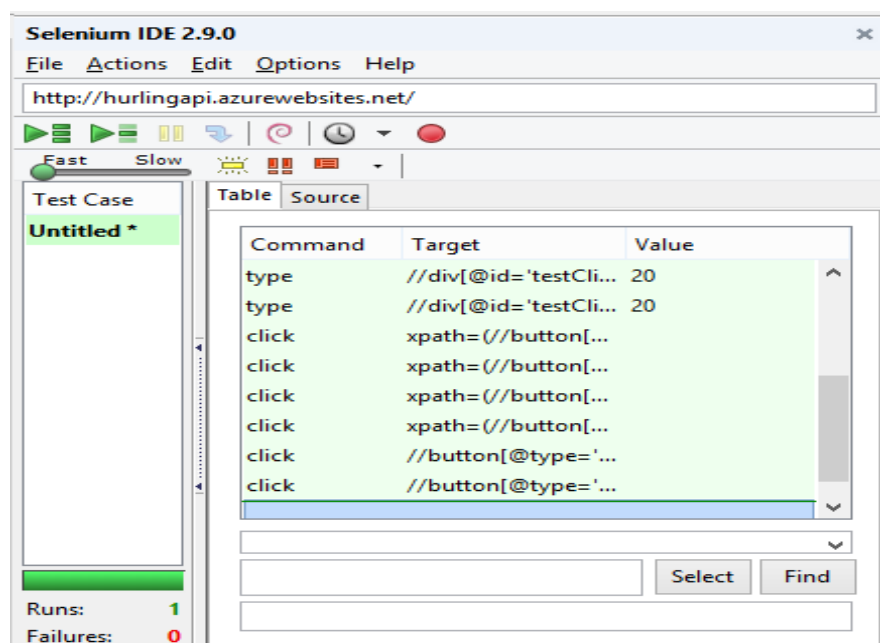


Figure 80 API insert player in team test

Test: Add user, create team, add player, remove player

The final test we ran on the API was a little more complex. It involved adding a new user, creating a team for him and 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.

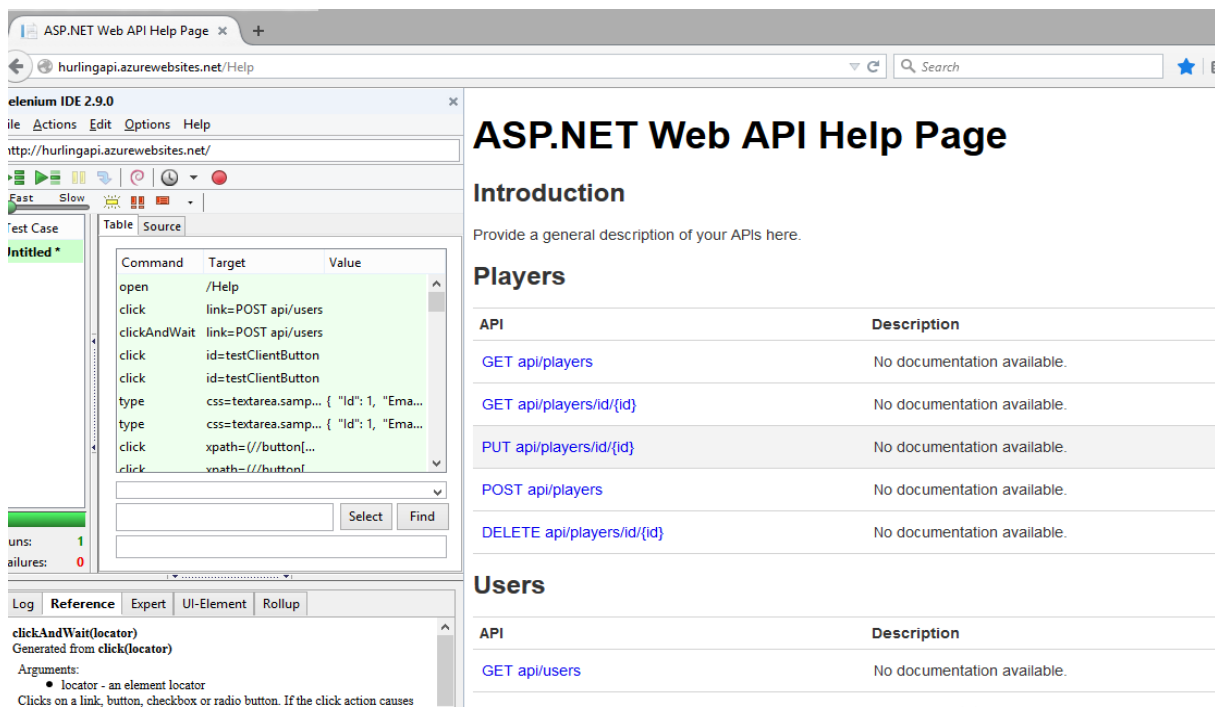


Figure 81 API insert player to team test 2

5.4 Fantasy Hurling Website Tests

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: Login

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

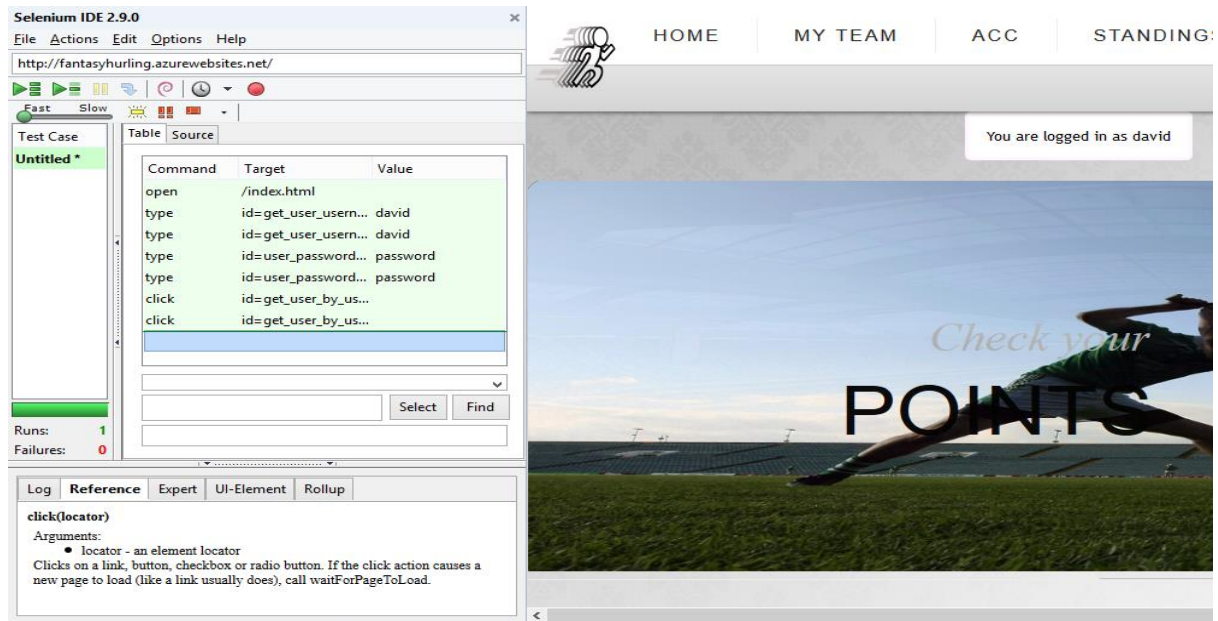


Figure 82 Login test

Test: Register

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:

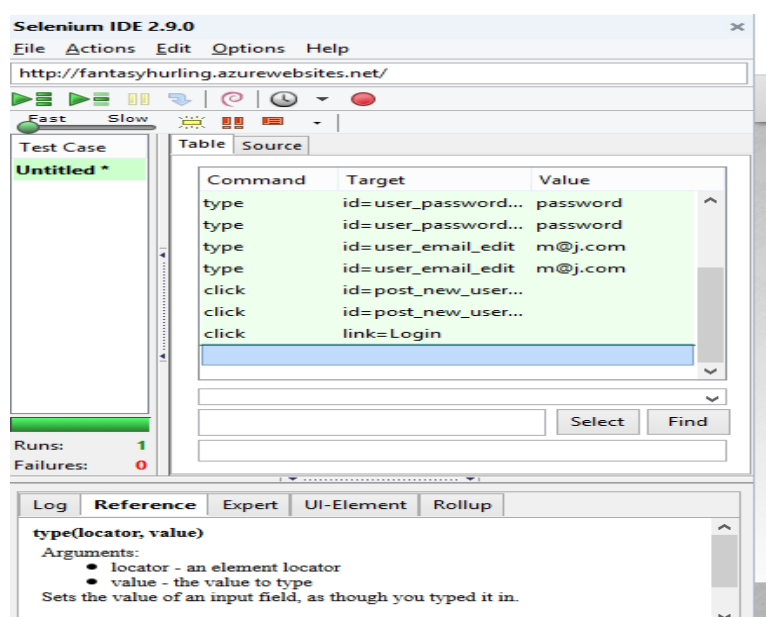


Figure 83 User Registration test

Test: Change user details

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.

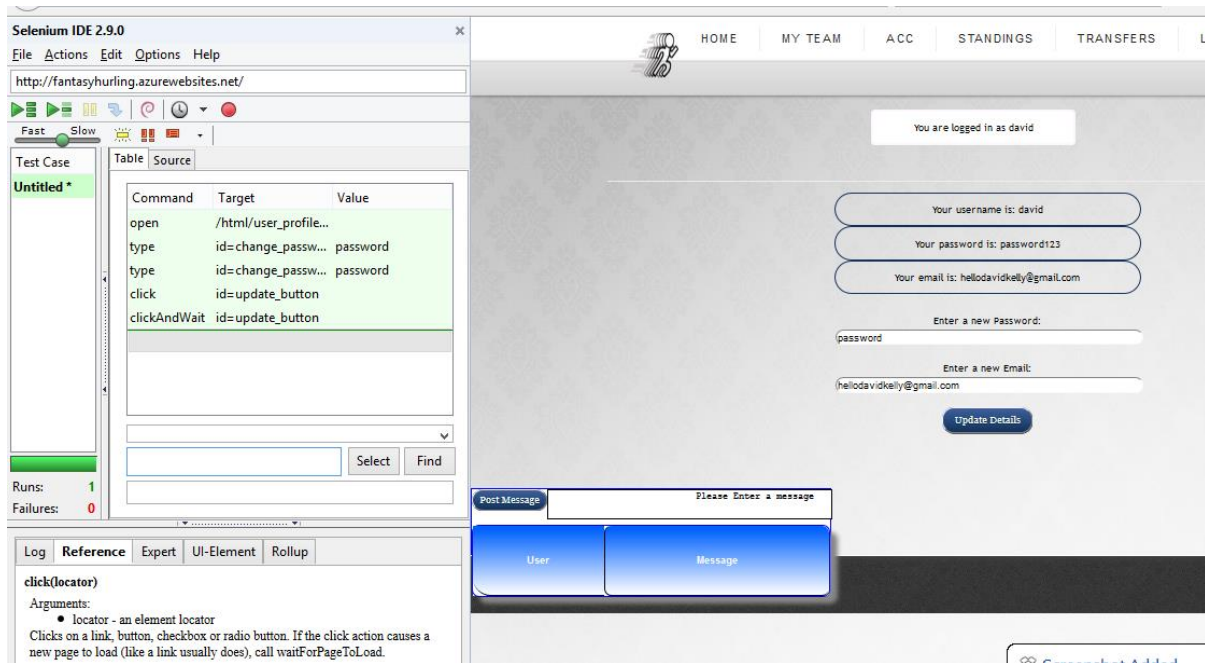


Figure 84 User Details test

Test: Delete player

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.

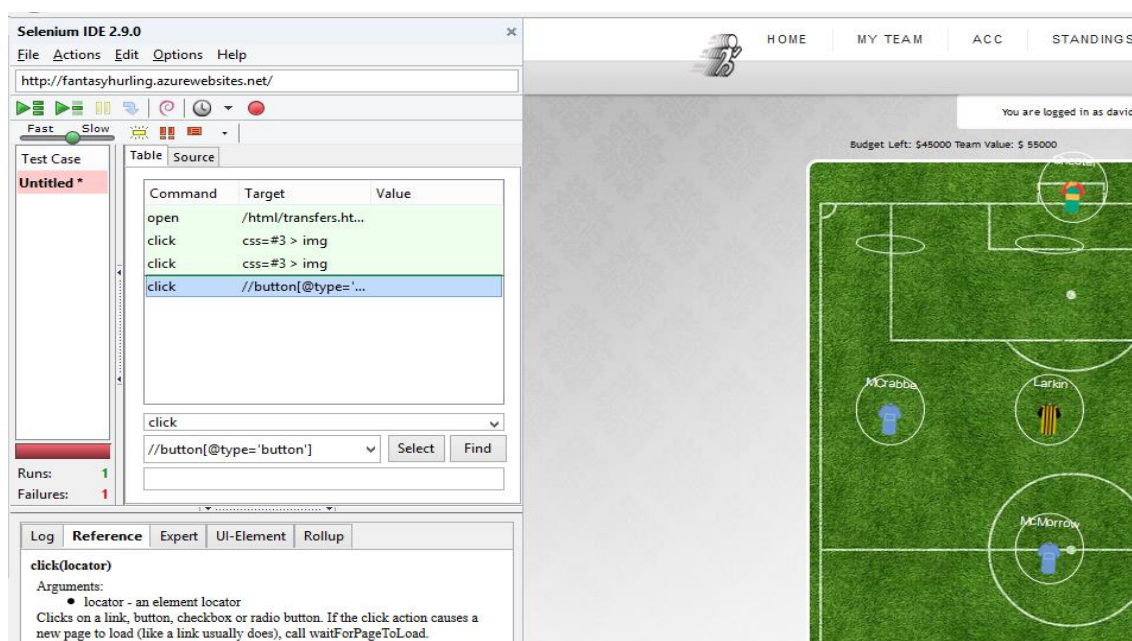


Figure 85 remove a player from team test

Test: Add player

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 automating his addition to the team again. It worked with no issues.



Figure 86 Add player to team test

5.5 Conclusion

The testing phase ran without much in the way of error. Due to the prototype model we chose for our SDLC it meant that we were simultaneously testing each environment as we were implementing. This allowed us to develop the project with fewer issues and the couple of small issues we found in the testing and evaluation phase were relatively small and simple fixes. The testing phase itself gave us valuable experience in using technologies and methodologies such as Unit tests and Selenium testing for web pages. It can be concluded that the application was well enough designed and implemented that few bugs arose.

Chapter 6: Conclusions and Further Work

6.1 Introduction

In this section we will discuss our own personal achievements and what we think we gained from the project, as well as looking at how the application could be improved upon in the future.

6.2 Achievements

We developed a full working front and back-end application using some interesting and relevant technologies. We implemented all of this having worked through a full project life cycle from project scope up until the demo of the application and handing over of the documentation.

6.3 Personal gain

Michael

I gained valuable experience in front end design and implementation. I feel that my ability to use JavaScript vastly improved and I became much more efficient in using JQuery to develop a multi functional and dynamic web site. I learned a vast amount about CSS also. I discussed with Martin a lot about the back-end, so feel I have a working understanding of how a back-end is developed, and I look forward to working on some back end projects myself in the near future. I also gained vital experience working with a version control system, GitHub.

Martin

By working on this project I gained valuable insights into contemporary web development. Using Microsoft technologies such as Azure, Visual Studio, C# .NET with Web API 2 and Entity Frameworks, writing asynchronous methods and functional code with LINQ library, I learned new technologies and methods and was able to compare Microsoft

development ecosystem with Java development. My exposure to JavaScript as fully functional language sparked new interest into the paradigm. The experiences and skills I acquired could serve as the foundation for upcoming fourth year project as well.

David

I found working as a member within a group to be a very important stage in my development as a programmer. Having team members allowed me to consult and incorporate ideas and methods for development I may not have achieved by myself. Throughout the course of this project I learned many new skills and also improved skills I already had. I had past knowledge with Photoshop, CSS, JQuery, HTML, JavaScript and Database design but working on a project of this scale I improved all those features of my programming ability. Although Martin was instrumental to the development and completion of the back-end database, I feel that working with him and understanding the technologies he used progressed my understanding of cloud based computing. Understanding Azure, Entity Frameworks and web API is an invaluable step in my progression as a programmer. Exposure to version control software such as Github will undoubtedly help me work within more group projects in future.

6.4 Further work (Possible improvements)

Optimization and Security

1. There is no authentication implemented for Web API service. Framework fully supports OAuth (Open Authentication Protocol), which could be used. We felt that this functionality is out of the scope of the project because of the complexity of the protocol and initial steep learning curve.
2. Fantasy Hurling Web Page has no encryption implemented in authentication mechanism. This should be considered as a topic for the further research and implementation.
3. Web API has no caching service implemented. HTTP protocol fully supports caching of repeating request in a browser cache if the server includes specific headers in response message.
4. HTML 5 local storage should be used more for optimization in Fantasy Hurling Web Page. Never changing data should be requested from Web API once at web page startup and kept in local session storage. That would immensely improve the page performance and take load of Web API.
5. Some of the Web API components design decisions are questionable. Because of our inexperience with a multitier cloud deployed project we fail to take all the aspects into

consideration in early prototypes. An example would be proper usage of MVC pattern or an evolvable Web API design. Further Web API code analysis and refactoring would improve the application and its testability.

Design and User Interface

1. Visual style of player display upgraded to drag and drop JQuery UI
2. All pages fully AJAX orientated
3. Design of site more user friendly
4. The site allows 8 players to be chosen into a team, it could be improved to have a full GAA hurling team of 15 allowed.
5. More players and Counties represented
6. JQuery implemented more professionally into the site
7. Addition of extra features such as mini games or GPS based leagues

6.5 Conclusion

This project was both challenging and rewarding for all involved. It provided us with an opportunity to both learn new technologies as well as becoming more familiar with those with which we had already worked with. We developed a professional application to a high level with a responsive and interactive front end and a well designed and implemented back end including an API. We successfully implemented all the specifics we outlined in the original proposal. We conducted a high level of testing in the application and found that there were very few issues that has arisen in the application. There were some limitations of course to the application, more features could have been implemented, and working in a group is always challenging, But overall the project was a rewarding experience. We hope to make improvements to the application in the future and use the whole process as valuable experience moving forward in our careers.

Appendix A: Project Diaries

Martin Zuber B00066378 Diary

Week 1 (September 15 – September 20)

- David Kelly and Michael James were chosen as this group project collaborators.
- We began discussing what kind of technologies we like and what kind of project we wish to implement
- We discussed who should supervise our project

Week 2 (September 22 – September 28)

- We chose Luke Raeside to supervise our team.
- Each of us began to note down some thoughts on the type of application.
- We decided we would do a fantasy hurling game web application with social media integration.

Used resources:

- Fantasy football websites for research, the GAA website for player names.

Week 3 (September 30 – October 6)

- Luke becomes our project supervisor.
- We began discussing the technologies involved in web development.
- We all began working on our individual scope documents.

Used resources:

- Internet as a source of initial information on web development process and methodologies.

Week 4 (October 8 – October 14)

- Research on how to write Project Scope section of this project documentation.
- We did some initial research on SDLC methodologies.
- I began researching SDLC methodologies.

Used resources:

- Internet as source of information.

Week 5 (October 16 – October 22)

- Research on how to write Project Proposal section of this project documentation.
- We made final decision on work division.
- David: Web design
- Michael: Front-End development.
- Me: Back-End development.
- I started specific research on modern back-end development methodologies and technologies.
- Decision was made to follow Prototyping SDLC methodology to develop this project.
- Each of us started to focus on specific research of technologies involved in his part of work.

Used resources:

- Node.js documentation
- Web Services on Internet

Week 6 (October 24 – October 30)

- Research on how to write Literature Review section of this project documentation.
- Research on Version Control Systems.
- All three of us had research paper topics to choose.
- We met up to discuss how each of our topics may benefit the group and its project as a whole and decided to choose topics based on this.
- I did learn how to use GIT version control system.
- I did some research on cooperative writing.

Used resources:

- Github.com
- Internet

Week 7 (November 1 – November 7)

- Final decision on Version Control System was made and we decided to cooperate on this project using GIT and GitHub.com
- Github repository was established and local repositories initialized.
- Each of us continued to research his specific topics.
- My research on back-end development is focused on Web API services.

Used resources:

- Github help page
- Various Web API tutorials with PHP, node.js and ASP.NET

Week 8 (November 9 – November 15)

- Work on Analysis and Design documentation started.
- Final decision to build Web API was made.
- Research continues.
- Each of us continued to research his specific topics.
- Each of us starting to write his part of Analysis and Design documentation.
- I extended my research on deployment options, which lead me to cloud computing.

Used resources:

- Web API tutorials and Microsoft Azure Cloud documentation.

Week 9 (November 17 – November 23)

- Work on Analysis and Design documentation continues.
- We designing prototypes for database, controllers and front-end models
- Use cases are being defined and UML graphs are getting made.
- I decided to deploy this project up on Microsoft Azure Cloud
- Cloud account is signed-in
- Each of us continues of his own specific research.
- Each of us is working on his part of Analysis and Design documentation.
- I signed-in to Azure Cloud and learned how it's used.
- I continue to find out the best way to build Web API service.

Used resources:

- UML tutorials, Web API tutorial, Azure Cloud Documentation.

Week 10 (November 25 – November 31)

- Work on Analysis and Design documentation continues.
- Coding of prototypes starts.
- Decision to build Web API using is made.
- Each of us starting to code prototypes locally and commit the work in progress on upstream Github repository: https://github.com/Michaelcj10/Fantasy_Hurling
- I installed Visual Studio 2013, deployed Azure data store for project database and Deployed web server for Web API.
- I started to research Web API .Net and Entity Framework and started to build the database and code Web API: <https://github.com/zubidlo/FanHurApi>

Used resources:

- Designing Evolvable Web APIs with ASP.NET (apress)

Week 11 (December 2 – December 8)

- Analysis and Design documentation finished
- Coding of prototypes continues.
- Each of us continues to code.
- I deployed the database on the data store and first Web API prototype
- I feel more comfortable working with Visual Studio 2013 now.

Used resources:

- Designing Evolvable Web APIs with ASP.NET (apress)

Week 12 (December 10 – December 16)

- Coding of prototypes continues.
- As changes in design are made documentation is updated.
- Each of us continues to code.
- I'm adding controller prototypes one by one.
- I started to work on database administration web page, which will allow access to CRUD methods.

Used resources:

- Designing Evolvable Web APIs with ASP.NET (apress)
- Various tutorial on JavaScript, JQuery and Microsoft Sql

Week 13 (January 5 – January 11)

- Database Schema Prototype Redesigned, User Table in Final Form
- Users Controller Prototype implemented.
- Data Admin Page for Users prototype implemented.
- Problem with JSON parser fixed.

Used resources

- Web API Online Tutorials
- Practical ASP.NET Web API (apress)

Week 14 (January 12 – January 18)

- Database Deployed to Azure Cloud
- Web API prototype deployed to Azure
- Research on Cross-Origin-Resource-Sharing done
- Work on Database Admin Pages Continues

Used Resources

- Azure Cloud Tutorials and Stack Overflow answers
- CORS articles and how-tos

Week 14 (January 19 – January 25)

- Message and Position database Table created
- Message and Position controller prototype created
- Admin page for Message and Position created
- Admin pages JavaScript refactored to more acceptable form

Week 15 (January 26 – February 1)

- Work on database schema continues
- With new tables, new controllers are created.
- For each controller Admin Page is created
- Problem with cyclic references approached with creation of DTO factories

User resources

- Entity Framework Recipes (apress)
- JavaScript and JQuery the Missing Manual 3rd Edition (O'Reilly)

Week 16 (February 2 – February 8)

- Database in final stage
- Custom ActionResult Created
- Abstractions: IEntity, IRepository and AbstractDTOFactory created
- All controller method turned to asynchronous tasks

Used Resources

- C# in Depth, 3d Edition (manning)
- Practical ASP.NET Web API (apress)
- Entity Framework Recipes (apress)

Week 17 (February 9 – February 15)

- OData support explored
- Work on business logic started
- Data Admin Pages finished, with Login dialog implemented
- Testing Data populated into the database

Used Resources

- OData, HTML5 and CSS3 online tutorials and how-tos

Week 18 (February 16 – February 22)

- Business logic in all controllers finished
- Web API test client installed
- Web API help page documentation automatic creation attempt failed
- Possible Authentication approaches researched and considered

Used Resources

- Pro ASP.NET Web API Security (apress)
- Web API online tutorials

Week 19 (February 23 – March 1)

- Code analysis
- Refactoring to abstract Repository
- Starting to commenting the code

Used Resources

- C# reference documentation

Week 10 (March 2 – March 8)

- Commenting the controller and Admin Pages continues
- Additional refactoring of controllers
- W3C markup and CSS validation performed

Week 21 (March 9 – March 15)

- Initial work on documentation agreed
- Table of contents prototype created
- Work for Project Skill collected and revised

Used Resources

- ITB library
- Microsoft Office Word Online Tutorials

Week 22 (March 16 – March 22)

- Research on Web Api testing started
- Web API testing design attempted
- Documentation Chapter on testing started
- Code listings parsed to project documentation

Used Resources

- Various online tutorials

Week 23 (March 23 – March 29)

- Testing design second and final attempt
- Testing Chapter finished
- Introduction Chapter contributions added to documentation prototype

Week 24 (March 30 – April 5)

- Holidays

Week 25 (April 6 – April 12)

- Holidays

Week 26 (April 13 – April 19)

- Chapter: Literature Review contributions to project documentation added
- Chapter: Analysis and Design started
- Additional research on modern web done

Used Resources

- Research papers
- Online articles

Week 27 (April 20 – April 26)

- Chapter Implementation started
- Chapter Conclusion and Further work contributions added to project documentation
- Project Documentation Properly reformatted

Used Resources

- Microsoft Word Online Tutorials

Week 28 (April 27 – May 3)

- Chapter on Implementation contributions added
- Table of Figures and Table of Contents created

Week 29 (May 4 – May 10)

- Documentation final formatting revisited
- Demo of the project presented

Michael James B00019330 Diary

Week 1 (September 15 – September 20)

- David Kelly and Michael James were chosen as this group project collaborators.
- We began discussing what kind of technologies we like and what kind of project we wish to implement
- We discussed who should supervise our project

Week 2 (September 22 – September 28)

- We chose Luke Raeside to supervise our team.
- Each of us began to note down some thoughts on the type of application.
- We decided we would do a fantasy hurling game web application with social media integration.

Used resources:

- Fantasy football websites for research, the GAA website for player names.

Week 3 (September 30 – October 6)

- Luke becomes our project supervisor.
- We began discussing the technologies involved in web development.
- We all began working on our individual scope documents.

Used resources:

- Internet as a source of initial information on web development process and methodologies.

Week 4 (October 8 – October 14)

- Research on how to write Project Scope section of this project documentation.
- We did some initial research on SDLC methodologies.
- I began research for my scope document

Used resources:

- Internet as source of information.

Week 5 (October 16 – October 22)

- I began to research for the proposal document.
- We made final decision on work division.
- David: Web design
- Michael: Front-End development.
- Me: Back-End development.

Used resources:

- JavaScript documentation and lecture material reviewed
- Web Services on Internet

Week 6 (October 24 – October 30)

- Research on how to write Literature Review section of this project documentation.
- Research on Version Control Systems.
- All three of us had research paper topics to choose.
- We met up to discuss how each of our topics may benefit the group and its project as a whole and decided to choose topics based on this.

Used resources:

- Github.com
- Internet

Week 7 (November 1 – November 7)

- Final decision on Version Control System was made and we decided to cooperate on this project using GIT and GitHub.com
- GitHub repository was established and local repositories initialized.

Used resources:

- GitHub documentation
- I reviewed JQuery UI and fantasy football websites

Week 8 (November 9 – November 15)

- Design document began
- Research on JavaScript

Used resources:

- JavaScript documentation

Week 9 (November 17 – November 23)

- Work on Analysis and Design
- Front end modelling
- UML diagrams done

Used resources:

- UML tutorials

Week 10 (November 25 – November 31)

- Work on Analysis and Design
- Each of us starting to code prototypes locally and commit the work in progress on upstream GitHub repository: https://github.com/Michaelcj10/Fantasy_Hurling

Week 11 (December 2 – December 8)

- Analysis and Design documentation finished
- I began to choose a front end visual theme
- I chose the general structure to the front end coding
- I reviewed Martins web API pages to become familiar with them

Used resources:

- WC3 Schools, various HTML and JQuery library's

Week 12 (December 10 – December 16)

- Study for exams limits work
- Design document updated

Used resources:

- The design document

Week 13 (January 5 – January 11)

- I start becoming familiar with the admin pages for inputting data
- I add some users and experiment with some tests

Week 14 (January 12 – January 18)

- I begin work on the front end
- Web pages built and tested

Used Resources

- Front end documentation and eBooks

Week 15 (January 19 – January 25)

- Testing of previous work
- Jerseys added to the prototype from David
- Design document updated
- Work continued on building the front end website

Week 16 (January 26 – February 1)

- Website work continues with multiple pages complete
- Some JQuery added
- Initial 30 players input to database
- Initial users input to database for testing

Week 17 (February 2 – February 8)

- First JavaScript connection to web API done and tested
- Functionality of web pages worked on
- Start to implement the instant message chat feature
- Experiment with asynchronous AJAX requests

Used Resources

- AJAX Documentation
- Design document
- HTML and CSS EBooks

Week 18 (February 9 – February 15)

- Work on adding some players to a user's team for testing
- Attempt to display user team in a visually appealing format
- Change to initial design of how users login

Week 19 (February 16 – February 22)

- Users can now login and view their team
- Tweaks to CSS made in the front end application

Used Resources

- WC3
- Web application development tutorials

Week 20 (February 23 – March 1)

Work on the standings and user accounts page Week 17 (February 16 – February 22)

- Users can now login and view their team
- Tweaks to CSS made in the front end application

Used Resources

- WC3
- Web application development tutorials

Week 21 (March 1 – March 8)

- Work on how the transfers page will function
- More thought on session storage and its implications

Used Resources

- WC3 for session storage research
- Web application development tutorials

Week 22 (March 9 – March 16)

- Work on transfers page begins
- Tweaks to CSS made in the front end application

Used Resources

- WC3
- Web application development tutorials

Week 23 (March 17 – March 24)

- Transfers page works to add or remove a player from a user team
- Tweaks to CSS made in the front end application including chat style

Used Resources

- Web application development tutorials

Week 24 (March 25 – April 2)

- Entire front end now functional
- Fix to if user has not enough players in their team, user is now automatically directed to the transfers page and must fix this issue.

Used Resources

- Tutorials on JavaScript online
- Web application development tutorials

Week 25 (April 3 – April 10)

- Work on documentation solely
- Gathering of research projects to add to document

Used Resources

- Thesis research
- Harvard referencing tutorials

Week 26 (April 11 – April 18)

- Work on documentation continues
- Tweaks to CSS made in the front end application

Used Resources

- WC3
- Web application development tutorials

Week 27 (April 19 – April 26)

- Fix of bug to create new user team if that user has no team registered to them
- Tweaks to CSS made in the front end application

Used Resources

- WC3
- Web application development tutorials

Week 28 (April 27 – May 3)

- Site testing complete
- Documentation work continues and is near complete

Used Resources

- WC3
- Web application development tutorials

Week 29 (May 4 – May 11)

- Project demoed to our group supervisor
- Documentation complete

David Kelly B00060572 Diary

Week 1 (September 15 – September 20)

- Activities:
- Found two other team members to form a group for the project
- Martin Zuber and Michael James
- Roles:
- The group initially began discussing the project
- We selected technologies we would like to work with
- We discussed potential project supervisors

Week 2 (September 22 – September 28)

- Activities:
- The group decided that the project supervisor we would like was Dr. Luke Raeside. We all had Luke for previous modules and thought his supervision would suit our project.
- Each group member explored project ideas and reviewed the list of designated projects
- The group decided to create a fantasy hurling website with social media capabilities
- Each member of the group explored technologies involved in creating a web based gaming platform
- Roles:
- I collected information about hurling competitions, players, teams
- Michael and Martin researched existing fantasy hurling platforms

Used resources:

- Fantasy football website
- GAA website
- Fantasy Hurling searches

Week 3 (September 30 – October 6)

- Activities:
- The group received confirmation of our group members from Michael O'Donnell
- Dr. Luke Raeside confirmed he would be our project supervisor
- Each member of the group began working on our scope documents
- Luke added the social media aspect to our project
- Roles:
- Each member began creating a scope document
- Michael, Martin and I reviewed technologies we researched independently
- Each member continued research

Used resources:

- Lecture notes
- Java Documentation
- Web Development books
- GAA books

Week 4 (October 8 – October 14)

- Activities:
- Each member of the group submitted their scope document. Although the documents were different, the scope outline was decided as a group
- The group decided a structure to implement the fantasy hurling website
- We agreed on technologies to use for implementation of the project, Java Server, JQuery, CSS and HTML
- The group decided on using theme roller for site design
- Roles:
- I began researching layout designs for the website and other technologies that could assist our project
- Martin began researching Java Server information
- Michael researched the database design using SQL
- Each group member reviewed existing fantasy gaming platforms

Used resources:

- Online web design projects, existing fantasy sports games websites
- Java Server documentation
- SQL database design documentation

Week 5 (October 16 – October 22)

- Activities:
- The group decided on the project specification and was satisfied that we were moving in the right direction
- We finalized the software we would use on the project
- We began delegating roles to each group member
- We all went to Luke's lab and discussed the project scale and timeframe
- The group shared research information regarding the project
- Each member submitted their proposal document
- Roles:
- This week the group was writing summary articles
- I chose social media
- Michael chose web development design
- Martin chose JavaScript

Used resources:

- Research paper documents “Social Media and Prosumerism”

Week 6 (October 24 – October 30)

- Activities:
 - Each member continued working on the summary document
 - We discussed potential changes to the back end of the project
 - In rich web apps we learned new techniques to storing and receiving data
 - The group discussed documents for our individual research papers that could help the group as a whole
- Roles:
 - I continued with the summary document and researched website design and logo
 - Martin researched and explored Microsoft Azure cloud based back side database Retrieving data via JSON
 - Michael began writing documentation and UML

Used resources:

- Research paper documents “Social Media and Prosumerism”
- Each member of the group reviewed research papers to do a literature review on

Week 7 (November 1 – November 7)

- Activities:
 - Each member of the group selected the literature review research papers
 - The group met up and discussed which aspects of the project they felt their literature review would cover
 - Martin and I submitted our summary documents
 - The group agreed that Martins new database was the best way forward, using new JSON skills retrieving data from the cloud server via a REST API
- Roles:
 - Martin continued exploring the technology and language needed to create this new database
 - Michael continued working on documentation
 - Michael and I began working on the website
 - Each member selected the literature they would be reviewing
 - I chose the impact of fantasy sports games and their involvement via social media

Used resources:

- Each member of the group was working on their literature review with respective resources
- Literature review documents “ Sport Fandom in the Digital World ” and “ Focus On Fantasy (an overview of fantasy sport consumption) ”

Week 8 (November 9 – November 15)

- Activities:
- Each member of the group was working on their individual literature review
- Michael continued to create UML designs and the documentation of the front end of the website
- Martin called a meeting to confirm the new back end design of the project, using JSON API
- Roles:
- I began creating images for the website
- Michael was building the website
- Martin began learning how to create REST API's

Used resources:

- JavaScript tutorials, Photoshop tutorials, API tutorials
- All the academic research papers in our literature reviews

Week 9 (November 17 – November 23)

- Activities:
- Each member of the group was working on their individual literature review
- Michael created a GitHub repository so we could all work on the project from one source
- Martin and I installed GitHub and added the project locally
- Roles:
- I began creating county jerseys for the website using Photoshop
- Michael created the GitHub account
- I began learning how to use GitHub
- Martin began writing a unique REST API for our website

Used resources:

- GitHub
- Azure.Microsoft
- Continued learning of JavaScript, JSON, AJAX, C#, Photoshop
- GitHub & Azure.Microsoft

Week 10 (November 25 – November 31)

- Activities:
- Each member of the group submitted their literature review
- Each member of the group continued working on last weeks tasks
- The individual presentations were handed out and we had to give a group presentation with an amalgamation of all three presentations
- The group met to discuss areas of the presentation for each member to focus on
- Roles:

- I focused my presentation on the introduction and market of fantasy sports with parts
- Michael Focused on the progress of the project so far
- Martin focused on the technologies we used while creating the project
- Each of us starting to code prototypes locally and commit the work in progress on upstream GitHub repository: https://github.com/Michaelcj10/Fantasy_Hurling

Used resources:

- GitHub & Azure.Microsoft
- Continued learning of JavaScript, JSON, AJAX, C#, Photoshop

Week 11 (December 2 – December 8)

- Activities:
- The group amalgamated all three presentations in to one presentation
- The group discussed how the project was progressing
- Roles:
- Martin created APIs to retrieve data from our server
- Michael completed the documentation with UML and wireframe for the website design
- I continued working on the website images for the county jerseys

Week 12 (December 10 – December 16)

- Activities:
- Each group member continued working on the project
- The group designed and implemented a prototype v1.0
- Martin created the back end
- Michael and I creating the front end
- Roles:
- Martin was working with .net technologies
- Michael created a registration page
- I created a standings page
- The group began merging the front end with the back end for the prototype

Week 13 (January 5 – January 11)

- Activities:
- I began to familiarize myself with the CSS and HTML
- Each member continued to action functionality to the website
- We began implementing more aspects of the analysis and design phase
- Roles:
- I focused on getting to know the website and the back end API
- Researched hurling players and counties

Week 14 (January 12 – January 18)

- Activities:
- Had a meeting with the group and discussed more specific roles
- Looked at researching web design
- Roles:
- Michael began building the website
- I reviewed literature on web design

Used Resources

- Panda.com
- Front end documentation

Week 15 (January 19 – January 25)

- Activities:
- Work was divided and documented on Github
- Each member added functionality to the website
- Roles:
- Work continued on fine tuning the UML
- Michael and I worked on the website

Used Resources

- WC3
- UML books from ITB library
- Web application development tutorials

Week 16 (January 26 – February 1)

- Activities:
- We met up for a group meeting to evaluate progression
- Michael created JavaScript to add functionality to the website
- Martin documented his API framework
- I began creating lists of hurlers from teams and counties
- Group meets to discuss the thesis
- Roles:
- I familiarized myself with the JavaScript and CSS
- Work continues on the project with documentation beginning for thesis
- We all worked individually this week

Week 17 (February 2 – February 8)

- Activities:
- Connections are made between the JavaScript and the back end database
- Designs are implemented and the style of the website is implemented
- Update current Thesis documentation with groups projects from last semester
- We use the administrator login to edit Information about the game
- Roles:
- I work on the last of the new jerseys I created
- Michael and I work on HTML

Used Resources

- Color Scheme designer engine
- Photoshop
- HTML and CSS EBooks

Week 18 (February 9 – February 15)

- Activities:
- The last of the Jerseys are created
- Each member pushes their local projects to the git repository
- Roles:
- I Find real life player positions and statistics
- Michael Continues with the merging of the front end and back end
- Martin hosts the website live on his Azure server
- We add iterations through the development of the website

Used Resources

- WC3
- Azure Cloud
- JavaScript eBook
- GAA websites
- Web application development tutorials

Week 19 (February 16 – February 22)

- Activities:
- Functionality of the website is progressing steadily
- Michael created a new login system connected to the API
- I use CSS on model to clean the design of the website
- Add Repository on GitHub for documentation work for thesis
- Roles:
- We all work on the Website

- Continue JavaScript functions to API

Used Resources

- WC3
- Web application development tutorials

Week 20 (February 23 – March 1)

- Activities:
 - Michael populates a few players in to the database for functionality testing
 - I provide statistics and user testing for website
 - Martin works on the API to ensure correct database table relationships
 - We meet up for a group meeting to discuss how much we love hurling
- Roles:
 - I work on the user experience

Week 21 (March 1 – March 8)

- Activities:
 - Documentation of thesis begins heavily
 - Assembly of code and diagrams begin on Github
 - Population of player database to connect to API and images
 - Michael creates functionality between GUI and API
 - I work on chat feature and CSS
 -
- Roles:
 - We all submit documentation to Github
 - Michael and I are working on the website

Used Resources

- Github
- WC3 for session storage research
- Web application development tutorials

Week 22 (March 9 – March 16)

- Activities:
 - More documentation from last semester is added to the Github repository
 - Work begins on the transfers page of the website
 - The syncing of players to the database is taking place
- Roles:
 - Martin work son the players database on back end to ensure connection

- I provide user testing and website layout for design
- Michael uses JavaScript and API to connect the transfers of players

Used Resources

- Panda.com
- WC3
- CSS books from ITB library
- Web application development tutorials

Week 23 (March 17 – March 24)

- Activities:
- Group meets to discuss roles for thesis writing and how to use Github to edit text files
- Stages of thesis are being assembled for final document
- Introductions and conclusions to be written as content is added
- Roles:
- We all work on the thesis
- I work on finishing the population of database through administrator
- Michael works on JavaScript
- Martin does testing for speeds and database analysis

Used Resources

- ITB library website
- Harvard referencing websites
- W3C
- Web application development tutorials

Week 24 (March 25 – April 2)

- Activities:
- Content for thesis is added to final document
- Website is almost functionally complete
- Images are made for website design
- Roles:
- I use Photoshop mockups to create content for website
- Michael fine tunes website for functionality
- Martin finishes testing document
- We all contribute to the thesis final document on Github

Used Resources

- Photoshop
- Mouckups.com website for free templates

- ITB Library for thesis revision
- W3C
- GAA.com
- Tutorials on JavaScript online
- Web application development tutorials

Week 25 (April 3 – April 10)

- Activities:
- Front end is functional
- Chat feature on website needs fixing
- Feature added to direct user to transfers if missing player in team
- Documentation to Github is added
- All group members work individually on project
- Roles:
- Martin add documentation to Github final document
- I add documentation and alter CSS and HTML on website code
- Michael finish website functionality

Used Resources

- Thesis research
- Github
- ITB Library
- JavaScript online tutorials
- Harvard referencing tutorials

Week 26 (April 11 – April 18)

- Activities:
- Final alterations are made to the chat feature of the website
- CSS is amended to pass validation
- HTML is amended to pass validation
- Roles:
- Michael completes final functionality of website JavaScript
- I add content to website including images, jerseys and styling
- We all work individually on documentation
- Martin finalizes his documentation

Used Resources

- WC3
- Photoshop
- JavaScript online tutorials
- CSS online design layout website

- Web application development tutorials

Week 27 (April 19 – April 26)

- Activities:
- Final design and layout changes are made to the website
- Bugs are fixed for user experience
- Roles:
- Work on thesis is nearly completed with all members of the group adding their content
- I design chat room design layout
- Michael fixes the last known bugs in the system

Used Resources

- ITB library reference thesis
- JavaScript online tutorials
- CSS document features
- WC3
- Web application development tutorials

Week 28 (April 27 – May 3)

- Activities:
- Finalizing documentation of thesis
- Adding conclusions to chapters and formatting styles
- Final user tests on chat feature and website interface are complete
- Roles:
- Martin finishes his documentation part of the thesis
- Michael and I grammar check the thesis to date
- Final edits are made to chapters within thesis
- Michael fixes the last of the bugs in the front end of the system

Used Resources

- ITB library thesis shelf
- Harvard referencing tools
- CSS online tutorials
- Testing software from Martin and Michael
- Web application development tutorials

Week 29 (May 4 – May 11)

- Project demoed to our group supervisor
- Documentation complete

Appendix B: Code Listing of Leagues Admin Page

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Login Page</title>
<link rel="stylesheet" type="text/css" href="css/login_styles.css">
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script>
    //jquery solution
    $(document).ready( function() { // the DOM will be available here
        //register listener for form submit button
        $("form").on("submit", function(event) {
            //supress default html form action which is jump to ac-
tion="#"

            event.preventDefault();
            //get references to form input fields
            var $username = $("#username");
            var $password = $("#password");
            //request user by given username
            $.ajax(
            {
                url : "http://hurlingapi.azurewebsites.net/api/us-
ers/username/" + $username.val(),
                success : function(data) { //if username match
                    //well returned data is user actually
                    var user = data;
                    //if password match
                    if ($password.val() === user.Password) {

                        //put user Id into session storage
                        sessionStorage.setItem("signed", user.Username);
                        //go to index.html
                        window.location = "users.html";
                    }
                    //if password doesn't match
                    else {

                        //clean form input fields
                        $username.val("");
                        $password.val("");
                    }
                },
                error : function() { //if username doesn't match
                    //clean form input fields
                    $username.val("");
                    $password.val("");
                }
            })
        });
    });
});
```

```

</script>
<body>
  <section id="loginBox">
    <h2>Login</h2>
    <form action="#" class="minimal">
      <label for="username">
        Username:
        <input type="text" name="username" id="username" place-
holder="Username must be between 8 and 20 characters" pattern="^[a-zA-Z][a-
zA-Z0-9-_\.\]{8,20}$" required="required" />
      </label>
      <label for="password">
        Password:
        <input type="password" name="password" id="password" place-
holder="Password must contain 1 uppercase, lowercase and number" pat-
tern="(?=^.{8,}$) ((?=.*\d) | (?=.*\W+)) (?![.\n]) (?=.*[A-Z]) (?=.*[a-z]) .*$"
required="required" />
      </label>
      <button type="submit" class="btn-minimal">Sign in</button>
    </form>
  </section>
</body>
</html>

```

login_styles.css

```

*{
  margin: 0;
  padding: 0;
  outline: none;
}

body {
  background: #eee;
  color: #444;
  -webkit-font-smoothing: antialiased;
  font-family: "Open Sans", Arial, Helvetica, Geneva, sans-serif;
  font-size: 16px;
  font-weight: 400;
  height: auto !important;
  height: 100%;
  line-height: 1.6em;
  min-height: 100%;
}

h2 {
  color: rgb(34,34,34);
  font-size: 2.2em;
  font-weight: 200;
  margin: 0 0 24px 0;
}

[class*='btn-'] {
  border: none;
  border-bottom: 2px solid rgba(0,0,0,.15);
  border-top: 1px solid rgba(255,255,255,.15);
  border-radius: 3px;
  color: #fff;
  display: inline-block;
  font: -webkit-small-control;
}

```

```

font-size: .7em;
letter-spacing: 1px;
line-height: 140%;
padding: 10px 20px;
text-decoration: none;
text-shadow: 0 1px 1px rgba(0, 0, 0, 0.25);
text-transform: uppercase;

-webkit-transition: all 0.1s linear;
-moz-transition: all 0.1s linear;
-o-transition: all 0.1s linear;
transition: all 0.1s linear;
}

.btn-minimal {
background-color: rgb(255,255,255);
border-radius: 0;
border: 1px solid rgb( 186, 186, 186 );
color: rgb( 186, 186, 186 );
text-shadow: 0 1px 1px rgba(255, 255, 255, 1);
}

.btn-minimal:hover {
background-color: #4195fc;
border: 1px solid rgba(0,0,0,.1);
color: rgb(255,255,255);
cursor: pointer;
text-shadow: 0 1px 1px rgba(0, 0, 0, 0.25);
}

.btn-minimal:active {
box-shadow: 0 1px 1px rgba(0,0,0,0.15) inset;
text-shadow: 0 -1px 1px rgba(0, 0, 0, 0.25);
}

section#loginBox {
background-color: rgb(255,255,255);
border: 1px solid rgba(0,0,0,.15);
border-radius: 4px;
box-shadow: 0 1px 0 rgba(255,255,255,0.2) inset, 0 0 4px rgba(0,0,0,0.2);
margin: 40px auto; /*aligns center*/
padding: 24px;
width: 500px;
}

form.minimal label {
display: block;
margin: 6px 0;
}

form.minimal input[type="text"],
form.minimal input[type="email"],
form.minimal input[type="number"],
form.minimal input[type="search"],
form.minimal input[type="password"],
form.minimal textarea {
background-color: rgb(255,255,255);
border: 1px solid rgb( 186, 186, 186 );
border-radius: 2px;

```

```

-webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
-moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
  box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
display: block;
font-size: 14px;
margin: 6px 0 12px 0;
padding: 8px;
text-shadow: 0 1px 1px rgba(255, 255, 255, 1);
width: 90%;

-webkit-transition: all 0.1s linear;
-moz-transition: all 0.1s linear;
-o-transition: all 0.1s linear;
  transition: all 0.1s linear;
}

form.minimal input[type="text"]:focus,
form.minimal input[type="email"]:focus,
form.minimal input[type="number"]:focus,
form.minimal input[type="search"]:focus,
form.minimal input[type="password"]:focus,
form.minimal textarea:focus,
form.minimal select:focus {
  border-color: #4195fc;
-webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
-moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
  box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
  color: rgb(0,0,0);
}

form.minimal input[type="text"]:invalid:focus,
form.minimal input[type="email"]:invalid:focus,
form.minimal input[type="number"]:invalid:focus,
form.minimal input[type="search"]:invalid:focus,
form.minimal input[type="password"]:invalid:focus,
form.minimal textarea:invalid:focus,
form.minimal select:invalid:focus {
  border-color: rgb(248,66,66);
-webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px
rgb(248,66,66);
-moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px
rgb(248,66,66);
  box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px rgb(248,66,66);
}

```

header.html

```

<nav div class="invertedshiftdown">
  <ul>
    <li><a href="users.html">Users</a></li>
    <li><a href="messages.html">Messages</a></li>
    <li><a href="positions.html">Positions</a></li>
    <li><a href="players.html">Players</a></li>
    <li><a href="leagues.html">Leagues</a></li>
    <li><a href="teams.html">Teams</a></li>
    <li><a id="logout"
href="javascript:clearSessionAndReload();">Logout</a></li>
  </ul>
</nav>

```

leagues.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Data_Admin_Leagues</title>
  <link rel="stylesheet" href="css/styles.css">
  <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
  <script src="js/constants_and_functions.js"></script>
  <script src="js/leagues.js"></script>
  <script> loadCommonsAndStartProgram(dataAdminLeaguesCode); </script>
</head>
<body>
  <header><!-- inject "header.html" --></header>
  <main>
    <section>
      <article id="table_article">
        <div id="table_div"><!-- inject "table_article.html" --></div>
      </article>
      <article id="forms_article">
        <form id="edit_form" action="#">
          <fieldset>
            <legend>League</legend>
            <input id="id" type="hidden" value="0"/>
            <label for="name">Name:</label>
            <input id="name" type="text" placeholder="a string"
required/>
            <label for="nextfixtures">Next Fixtures:</label>
            <input id="nextfixtures" type="text" placeholder="yyyy-
mm-ddThh:mm:ss" value="2015-12-31T23:59:59" required/>
            <label for="week">Week:</label>
            <input id="week" type="number" min="0" max="255"
placeholder="<0 - 255>" required/>
            <div id="request_div"><!-- inject "request_select.html" -
--></div>
          </fieldset>
        </form>
        <div id="response_div"><!-- inject "response_fieldset.html" --
--></div>
      </article>
    </section>
  </main>
  <footer></footer>
</body>
</html>
```

request_select.html

```
<label for="request">Select the Request</label>
<select id="request">
  <option value="PUT">Change (PUT) </option>
  <option value="POST">Create (POST) </option>
  <option value="DELETE">Delete (DELETE) </option>
</select>
<input type="submit" value="submit">
```

response_fieldset.html

```
<fieldset id="response">
  <legend>Response</legend>
  <p><span id="text"></span></p>
</fieldset>
```

table_article.html

```
<form id="table_rows_form" action="#">
  <fieldset>
    <legend>Rows per page</legend>
    <input id="table_rows" type="number" value="10" min="1" max="100"
placeholder="<1 - 100>">
    <input type="submit" value="submit">
  </fieldset>
</form>
<fieldset>
  <legend>Repository</legend>
  <form id="previous_page_form" action="#">
    <input type="submit" value="previous page">
  </form>
  <div id="table"><!-- table injecting here --></div>
  <form id="next_page_form" action="#">
    <input type="submit" value="next page">
  </form>
</fieldset>
<fieldset>
  <legend>Number of items in the repository</legend>
  <input id="rows_count" type="number" readonly>
</fieldset>
```

styles.css

```
* {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

body {
  margin-top: 40px;
  font: bold 14px Arial;
}

header {
  position: fixed;
  top: 0;
  left: 0;
  height: 40px;
  width: 100%;
  background-color: white;
}
```

```

header {
    position: fixed;
    top: 0;
    left: 0;
    height: 40px;
    width: 100%;
    background-color: white;
}

section {
    position: relative;
}

article {
    /* position is static by default */
    display: inline;
    float: right;
    margin-top: 40px;
}

table {
    border-collapse: collapse;
}

legend {
    padding: 0.2em 0.5em;
    font-size: 110%;
}

fieldset {
    border: none;
    background-color: #fbfbfb;
    border-radius: 4px;
    box-shadow: 4px 4px 4px #bbbbbb;
}

label {
    display: block;
}

input {
    border-radius: 4px;
}

header, footer {
    text-align: center;
}

span {
    color : #D10000;
}

ul li {
    display : inline;
}

```



```

#table_article {
  width: 75%;
}
#forms_article {
  width: 25%;
}
[placeholder] {
  width: 170px;
}

input[type="submit"] {
  background-color: rgb(255,255,255);
  border: 1px solid rgb( 186, 186, 186 );
  color: rgb( 186, 186, 186 );
  text-shadow: 0 1px 1px rgba(255, 255, 255, 1);
}

input[type="submit"]:hover {
  background-color: #D10000; /*#4195fc*/
  border: 1px solid rgba(0,0,0,.1);
  color: rgb(255,255,255);
  cursor: pointer;
  text-shadow: 0 1px 1px rgba(0, 0, 0, 0.25);
}

input[type="submit"]:active {
  box-shadow: 0 1px 1px rgba(0,0,0,0.15) inset;
  text-shadow: 0 -1px 1px rgba(0, 0, 0, 0.25);
}

form input[type="text"],
form input[type="email"],
form input[type="number"],
form input[type="search"],
form input[type="password"],
form textarea table, th, td{
  background-color: rgb(255,255,255);
  border: 1px solid rgb( 186, 186, 186 );
  -webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
  -moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
  box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.08);
  font-size: 14px;
  text-shadow: 0 1px 1px rgba(255, 255, 255, 1);

  -webkit-transition: all 0.1s linear;
  -moz-transition: all 0.1s linear;
  -o-transition: all 0.1s linear;
  transition: all 0.1s linear;
}

form input[type="text"]:focus,
form input[type="email"]:focus,
form input[type="number"]:focus,
form input[type="search"]:focus,
form input[type="password"]:focus,
form textarea:focus,
form select:focus {

```

```

border-color: #4195fc;
-webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
-moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px #4195fc;
color: rgb(0,0,0);
}

form input[type="text"]:invalid:focus,
form input[type="email"]:invalid:focus,
form input[type="number"]:invalid:focus,
form input[type="search"]:invalid:focus,
form input[type="password"]:invalid:focus,
form textarea:invalid:focus,
form select:invalid:focus {
border-color: rgb(248,66,66);
-webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px
rgb(248,66,66);
-moz-box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px
rgb(248,66,66);
box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1), 0 0 8px rgb(248,66,66);
}

.invertedshifttdown{
padding: 0;
width: 100%;
border-top: 5px solid #D10000; /*Red color theme*/
background: transparent;
voice-family: "\"}\"\"";
voice-family: inherit;
}

.invertedshifttdown ul{
margin:0;
margin-left: 40px; /*margin between first menu item and left browser
edge*/
padding: 0;
list-style: none;
}

.invertedshifttdown li{
display: inline;
margin: 0 2px 0 0;
padding: 0;
text-transform:uppercase;
}

.invertedshifttdown a{
float: left;
display: block;
color: black;
text-decoration: none;
margin: 0 1px 0 0; /*Margin between each menu item*/
padding: 5px 10px 9px 10px; /*Padding within each menu item*/
background-color: white; /*Default menu color*/
-moz-border-radius-bottomleft: 5px;
border-bottom-left-radius: 5px;
-moz-border-radius-bottomright: 5px;
border-bottom-right-radius: 5px;
}

```

```

.invertedshifttdown a:hover{
  background-color: #D10000; /*Red color theme*/
  padding-top: 9px; /*Flip default padding-top value with padding-bottom */
  padding-bottom: 5px; /*Flip default padding-bottom value with padding-
top*/
  color: white;
}

.invertedshifttdown .current a{ /** currently selected menu item */
  background-color: #D10000; /*Red color theme*/
  padding-top: 9px; /*Flip default padding-top value with padding-bottom */
  padding-bottom: 5px; /*Flip default padding-bottom value with padding-
top*/
  color: white;
}

```

constants_and_functions.js

```

//loads parts of the DOM which are common for all pages and then executes
given function
var loadCommonsAndStartProgram = function(callback) {
  if(sessionStorage.getItem("signed") !== "Administrator") {
    //go to login page
    window.location = "login.html";
  }
  else {
    //after DOM is ready
    $(document).ready(function() {
      setTableDOMElements();
      var req1 = $.get("html_include/table_article.html", function(data)
{
        $table_div.html(data);
      });
      var req2 = $.get("html_include/request_select.html",
function(data) {
        $request_div.html(data);
      });
      var req3 = $.get("html_include/response_fieldset.html",
function(data) {
        hideElement($response_div);
        $response_div.html(data);
      });
      var req4 = $.get("html_include/header.html", function(data) {
        $("header").html(data);
      });
      //execute callback only when all requests are successful
      $.when(req1, req2, req3, req4).then(function () {
        callback();
      });
    });
  }
}

```

```

//constants
//url for development and production
//var url = "http://localhost:51642";
var _url = "http://hurlingapi.azurewebsites.net";
//how many rows will table have
var _top = 10;
//starting point in the table (0)
var _skip = 0;
//items in the table
var _count = 0;

//DOM element jquery objects common for all html files
var $logout;
var $table_div;
var $request_div;
var $response_div;
var $table_rows_form;
var $table_rows;
var $previous_page_form;
var $next_page_form;
var $table_rows_count;
var $table;
var $text;
var $edit_form;
var $id;

//this can be called only after document is ready
var setTableDOMElements = function() {

    $logout = $("#logout");
    $table_div = $("#table_div");
    $request_div = $("#request_div");
    $response_div = $("#response_div");
    $edit_form = $("#edit_form");
    $id = $("#id");
    $table_rows_form = $("#table_rows_form");
    $table_rows = $("#table_rows");
    $previous_page_form = $("#previous_page_form");
    $next_page_form = $("#next_page_form");
    $table_rows_count = $("#rows_count");
    $table = $("#table");
    $text = $("#text");
}

var hideElement = function($element, time_ms) {

    time_ms = typeof time_ms === "undefined" ? 0 : time_ms;
    setTimeout(function() { $element.hide(); }, time_ms);
}

var showElement = function($element, time_ms) {

    time_ms = typeof time_ms === "undefined" ? 0 : time_ms;
    setTimeout(function() { $element.show(); }, time_ms);
}

```

```

//this can be called only after document is ready
//prints a message when ajax request is successfull
//textStatus : passed by jquery ajax success function , see:
http://api.jquery.com/jquery.ajax/
//request : passed by jquery ajax success function , see:
http://api.jquery.com/jquery.ajax/
var printOutput = function(textStatus, request) {

    $text.empty().append(textStatus + ": " + request.status + "/" +
request.responseText);
    showElement($response_div);
    hideElement($response_div, 3000);
}

//this can be called only after document is ready
//prints a message when ajax request is successfull
//request : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
//textStatus : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
//errorThrown : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
var printError = function(request, textStatus, errorThrown) {

    $text.empty().append(textStatus + ": " + request.status + "/" +
errorThrown + ": " + request.responseText);
    showElement($response_div);
    hideElement($response_div, 3000);
}

//this can be called only after document is ready
//general error callback
//request : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
//textStatus : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
//errorThrown : passed by jquery ajax success function , see
http://api.jquery.com/jquery.ajax/
var generalErrorCallback = function (request, textStatus, errorThrown) {

    printError(request, textStatus, errorThrown);
}
//general ajax request, see http://api.jquery.com/jquery.ajax/
//url : string - url of requested resource (mandatory)
//successCallback : function (mandatory)
//errorCallback : function
//type : string - request type (default = "get")
//dataType : string - data type (default = "json")
//data : data passed in request body for "post" or "put " request (default
= "undefined")
var ajaxRequest = function(url, successCallback, errorCallback, type,
dataType, data ) {

    type = typeof type === "undefined" ? "GET" : type;
    dataType = typeof dataType === "undefined" ? "json" : dataType;
    data = typeof data === "undefined" ? null : data;
    errorCallback = typeof errorCallback === "undefined" ? function() {} :
errorCallback;

```

```

$.ajax({
    type : type,
    url : url,
    data : data,
    dataType : dataType,
    success : successCallback,
    error : errorCallback
});
}
//returns new object {top:_top, skip:_skip}
//_top, _skip are global variables
var tableCurrentPage = function() {
    return {
        top : parseInt(_top),
        skip : parseInt(_skip)
    };
}
//calculates previous table page, manipulates skip and returns new object
{top:_top, skip:_skip}
//_top, _skip are global variables
var tablePreviousPage = function() {
    _skip = parseInt(_skip) > 0 ? parseInt(_skip) - parseInt(_top) : _skip;
    return tableCurrentPage();
}
//calculates next table page, manipulates _skip and returns new object
{top:_top, skip:_skip}
//_top, _skip, _count are global variables
var tableNextPage = function() {
    _skip = parseInt(_skip) + parseInt(_top) < parseInt(_count) ?
    parseInt(_skip) + parseInt(_top) : _skip;
    return tableCurrentPage();
}
//returns a table header row html string built from given array of strings
//headersNamesArray : string array of headers
var buildTableHeaders = function (headersNamesArray) {
    var headers = "<tr><th>#</th>";
    for(var i = 0; i < headersNamesArray.length; headers += "<th>" +
headersNamesArray[i++] + "</th>");
    return headers + "</tr>";
}

//returns one table row string built from given javascript object
//rowNumber: int - number of the row
//properties : string array - object property names to put in the row
//object : object
var buildTableRow = function(rowNumber, properties, object) {
    var row = "<tr class='editable'><td>" + rowNumber + "</td>";
    for(var i = 0; i < properties.length; i++) {
        row += "<td>" + object[properties[i]] + "</td>";
    }
    return row + "</tr>";
}

```

```

var mouseEnterCallback = function() {
    $(this).children().css("background-color", "#D10000").css("color",
"white");
};
var mouseLeaveCallback = function() {
    $(this).children().css("background-color", "white").css("color",
"black");
};
//this can be called only after document is ready
//returns table html string
//counter_start: int - what number first table row starts with
//headers : string array - example: ['name', 'address', 'email']
//properties : string array - example: ['Name', 'Address', 'Email']
data[i].Name, data[i].Address, data[i].Email will be put in each table row
in that order
//data : object array - objects to put in the table
//output : DOM element jquery object - to append the table to
//function to execute if user clicks on table row
var buildTable = function (counter_start, headers, properties, data,
output, rowClickCallback) {

    var counter = counter_start;
    var table = "<table><thead>";
    table += buildTableHeaders(headers);
    table += "</thead><tbody>";
    if ($.isArray(data)) {
        $.each(data, function(index, object) {
            table += buildTableRow(++counter, properties, object);
        });
    }
    else {
        table += buildTableRow(++counter, properties, data);
    }
    table += "</tbody></table>";
    output.empty().append(table);

    if (typeof rowClickCallback !== "undefined") {
        $(".editable").mouseenter(this, mouseEnterCallback)
            .mouseleave(this, mouseLeaveCallback)
            .click(this, rowClickCallback);
    }
}
//clear form fields to default value
//forms = array of form element jquery objects
var clearFormFields = function(forms) {

    forms.forEach(function($form) {

        $form.each (function(){ this.reset(); });
    });
}

var clearSessionAndReload = function() {

    sessionStorage.clear();
    location.reload();
}

```

leagues.js

```
//this code uses jquery so that must be already loaded
//this code uses functions from "data_admin_functions.js" so that must be
already loaded
//execute only after DOM is ready
var dataAdminLeaguesCode = function() {
    //add path to URL prefix
    _url += "/api/leagues";

    //set common DOM element jquery objects
    setTableDOMElements();

    var $name = $("#name");
    var $nextfixtures = $("#nextfixtures");
    var $week = $("#week");

    var readLeagueFromInputFields = function () {

        return {

            Id : $id.val(),
            Name : $name.val(),
            NextFixtures : $nextfixtures.val(),
            Week : $week.val()
        };
    }
    var fillLeagueTextFields = function (tableRowData) {

        $id.val(tableRowData[1]);
        $name.val(tableRowData[2]);
        $nextfixtures.val(tableRowData[3]);
        $week.val(tableRowData[4]);
    }
    var rowClickCallback = function() {

        var data = [];
        $.each($(this).children(), function(key, value) {
            data.push(value.textContent);
        });

        fillLeagueTextFields(data);
    };
    //examples:
    //.../api/leagues?$orderby=Name --> get all leagues ordered by username
    //injects table of top items into DOM
    //page.top : how many rows the table will have
    //page.skip: how many rows to skip
    //example: page.top=10 and page.skip=0 --> table with first 10 items
    //page.top=10 and page.skip=10 --> table with from 11 to 20 items
    //page.top=10 and page.skip=20 --> table with from 21 to 30 items
    var getLeagues = function(page) {

        var url = _url + "?$top=" + page.top + "&$skip=" + page.skip;

        var successCallback = function (data, textStatus, request) {
```



```

        var counter_start = page.skip;

        var headers = [
            "Id <span>(PK)</span>",
            "Name",
            "Next Fixtures",
            "Week"
        ];

        var properties = [
            "Id",
            "Name",
            "NextFixtures",
            "Week"
        ];

        buildTable(counter_start, headers, properties, data, $table,
rowClickCallback);
    }

    ajaxRequest(url, successCallback);
}

_top = $table_rows.val();
ajaxRequest(_url, function(data, textStatus, request) {

    _count = parseInt(data.length);
    $table_rows_count.val(_count);
    getLeagues(tableCurrentPage());
});

$table_rows_form.submit(function(event) {

    event.preventDefault();
    _top = $table_rows.val();
    _skip = 0;
    getLeagues(tableCurrentPage());
});

$previous_page_form.submit(function(event) {

    event.preventDefault();
    getLeagues(tablePreviousPage());
});

$next_page_form.submit(function(event) {

    event.preventDefault();
    getLeagues(tableNextPage());
});

//POST PUT DELETE request
$edit_form.submit(function(event) {

    event.preventDefault();

    var url = _url;

```

```

var league = readLeagueFromInputFields();

var successCallback = function (data, textStatus, request) {
    ajaxRequest(_url, function(data, textStatus, request) {
        _count = parseInt(data.length);
        $table_rows_count.val(_count);
        getLeagues(tableCurrentPage());
    });

    clearFormFields([$edit_form]);
    printOutput(textStatus, request);
};

var errorCallback = function(request, textStatus, errorThrown) {
    clearFormFields([$edit_form]);
    generalErrorCallback(request, textStatus, errorThrown);
};

var type = $("option:checked").val();

if (type === "PUT") {
    url = url + "/id/" + league.Id;
}
else if (type === "DELETE") {
    url = url + "/id/" + league.Id;
    league = "undefined";
}

var dataType = "json";

ajaxRequest(url, successCallback, errorCallback, type, dataType,
league);
});
}

```

Appendix C: Code Listing of Hurling Web API

HurlingModel.Context.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class HurlingModelContext : DbContext
    {
        public HurlingModelContext()
            : base("name=HurlingModelContext")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Position> Positions { get; set; }
        public virtual DbSet<User> Users { get; set; }
        public virtual DbSet<League> Leagues { get; set; }
        public virtual DbSet<Team> Teams { get; set; }
        public virtual DbSet<Player> Players { get; set; }
        public virtual DbSet<Message> Messages { get; set; }
    }
}
```

IEntity.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace HurlingApi.Models
{
    public interface IEntity<T> : IDisposable where T : class
    {
        Task<IEnumerable<T>> GetAllAsync();
        Task<T> FindSingleAsync(Expression<Func<T, bool>> match);
        Task<bool> ExistAsync(Expression<Func<T, bool>> match);
        Task<int> UpdateAsync(T t);
        Task<int> InsertAsync(T t);
        Task<int> RemoveAsync(T t);
        Task<int> SaveChangesAsync();
    }
}
```

Entity.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace HurlingApi.Models
{
    public class Entity<T> : IEntity<T> where T : class
    {
        private readonly DbContext _context;

        bool _disposed;

        public Entity(DbContext context)
        {
            _context = context;
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (_disposed) return;

            if (disposing)
            {
                _context.Dispose();
            }

            // release any unmanaged objects
            // set the object references to null

            _disposed = true;
        }

        ~Entity()
        {
            Dispose(false);
        }

        public async Task<IEnumerable<T>> GetAllAsync()
        {
            IEnumerable<T> items = await _context.Set<T>().ToListAsync<T>();
            return items;
        }
    }
}
```

```

public async Task<T> FindSingleAsync(Expression<Func<T, bool>> match)
{
    T singleItem = null;
    try
    {
        singleItem = await _context.Set<T>().SingleOrDefaultAsync(match);
        return singleItem;
    }
    catch (InvalidOperationException)
    {
        throw new InvalidOperationException("More than one requested " +
            singleItem.GetType().Name + " found in the repository.");
    }
}

public async Task<bool> ExistAsync(Expression<Func<T, bool>> match)
{
    bool exist = await _context.Set<T>().AnyAsync(match);
    return exist;
}

public async Task<int> UpdateAsync(T t)
{
    try
    {
        _context.Entry(t).State = EntityState.Modified;
        int result = await _context.SaveChangesAsync();
        return result;
    }
    catch (Exception)
    {
        throw new Exception("An error occurred during " + t.GetType().Name + "
repository modification.");
    }
}

public async Task<int> InsertAsync(T t)
{
    try
    {
        _context.Set<T>().Add(t);
        int result = await _context.SaveChangesAsync();
        return result;
    }
    catch (Exception)
    {
        throw new Exception("Error occurred during adding " + t.GetType().Name
+ " to repository.");
    }
}

```

```

        public async Task<int> RemoveAsync(T t)
        {
            try
            {
                _context.Entry(t).State = EntityState.Deleted;
                int result = await _context.SaveChangesAsync();
                return result;
            }
            catch (Exception)
            {
                throw new Exception("Error occurred during deleting " +
t.GetType().Name + " from repository.");
            }
        }

        public async Task<int> SaveChangesAsync()
        {
            try {
                int result = await _context.SaveChangesAsync();
                return result;
            }
            catch (Exception)
            {
                throw;
            }
        }
    }
}

```

IRepository.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HurlingApi.Models
{
    interface IRepository : IDisposable
    {
        IEntity<User> Users();
        IEntity<League> Leagues();
        IEntity<Player> Players();
        IEntity<Position> Positions();
        IEntity<Team> Teams();
        IEntity<Message> Messages();
    }
}

```

FantasyHurlingRepository

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;

namespace HurlingApi.Models
{
    public class FantasyHurlingRepository : IRepository
    {
        private readonly DbContext _context;
        private readonly IEntity<User> userTable;
        private readonly IEntity<League> leagueTable;
        private readonly IEntity<Player> playerTable;
        private readonly IEntity<Position> positionTable;
        private readonly IEntity<Team> teamTable;
        private readonly IEntity<Message> messageTable;

        bool _disposed;

        public FantasyHurlingRepository()
        {
            _context = new HurlingModelContext();
            userTable = new Entity<User>(_context);
            leagueTable = new Entity<League>(_context);
            playerTable = new Entity<Player>(_context);
            positionTable = new Entity<Position>(_context);
            teamTable = new Entity<Team>(_context);
            messageTable = new Entity<Message>(_context);
        }

        public IEntity<User> Users()
        {
            return userTable;
        }

        public IEntity<League> Leagues()
        {
            return leagueTable;
        }

        public IEntity<Player> Players()
        {
            return playerTable;
        }

        public IEntity<Position> Positions()
        {
            return positionTable;
        }
    }
}
```

```

public IEntity<Team> Teams()
{
    return teamTable;
}

public IEntity<Message> Messages()
{
    return messageTable;
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

protected virtual void Dispose(bool disposing)
{
    if (_disposed) return;

    if (disposing)
    {
        _context.Dispose();
        userTable.Dispose();
        leagueTable.Dispose();
        playerTable.Dispose();
        positionTable.Dispose();
        teamTable.Dispose();
        messageTable.Dispose();
    }

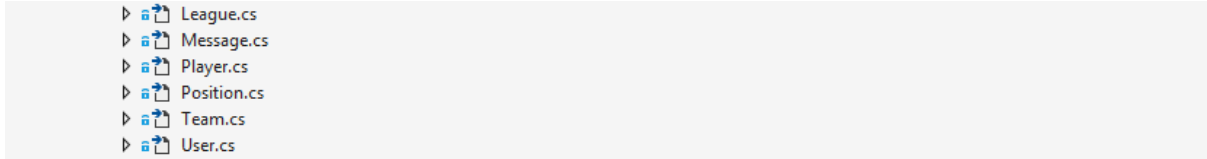
    // release any unmanaged objects
    // set the object references to null

    _disposed = true;
}

~FantasyHurlingRepository()
{
    Dispose(false);
}
}

```


Repository Models



```
▶ League.cs
▶ Message.cs
▶ Player.cs
▶ Position.cs
▶ Team.cs
▶ User.cs
```

Figure 87 Repository Models Classes

League.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class League
    {
        public League()
        {
            this.Teams = new HashSet<Team>();
        }

        public int Id { get; set; }
        public string Name { get; set; }
        public System.DateTime NextFixtures { get; set; }
        public byte Week { get; set; }

        public virtual ICollection<Team> Teams { get; set; }
    }
}
```

Position.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class Position
    {
        public Position()
        {
            this.Players = new HashSet<Player>();
        }

        public int Id { get; set; }
        public string Name { get; set; }

        public virtual ICollection<Player> Players { get; set; }
    }
}
```

Team.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class Team
    {
        public Team()
        {
            this.Players = new HashSet<Player>();
        }

        public int Id { get; set; }
        public string Name { get; set; }
        public decimal OverAllPoints { get; set; }
        public decimal LastWeekPoints { get; set; }
        public decimal Budget { get; set; }
        public int LeagueId { get; set; }
        public int UserId { get; set; }

        public virtual League League { get; set; }
        public virtual User User { get; set; }
        public virtual ICollection<Player> Players { get; set; }
    }
}
```

Player.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class Player
    {
        public Player()
        {
            this.Teams = new HashSet<Team>();
        }

        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string GaaTeam { get; set; }
        public decimal LastWeekPoints { get; set; }
        public decimal OverallPoints { get; set; }
        public decimal Price { get; set; }
        public byte Rating { get; set; }
        public bool Injured { get; set; }
        public int PositionId { get; set; }

        public virtual Position Position { get; set; }
        public virtual ICollection<Team> Teams { get; set; }
    }
}
```

User.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class User
    {
        public User()
        {
            this.Messages = new HashSet<Message>();
            this.Teams = new HashSet<Team>();
        }

        public int Id { get; set; }
        public string Email { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }

        public virtual ICollection<Message> Messages { get; set; }
        public virtual ICollection<Team> Teams { get; set; }
    }
}
```

Message.cs

```
namespace HurlingApi.Models
{
    using System;
    using System.Collections.Generic;

    public partial class Message
    {
        public int Id { get; set; }
        public string Text { get; set; }
        public int UserId { get; set; }
        public System.DateTime Created { get; set; }

        public virtual User User { get; set; }
    }
}
```

Model DTOs

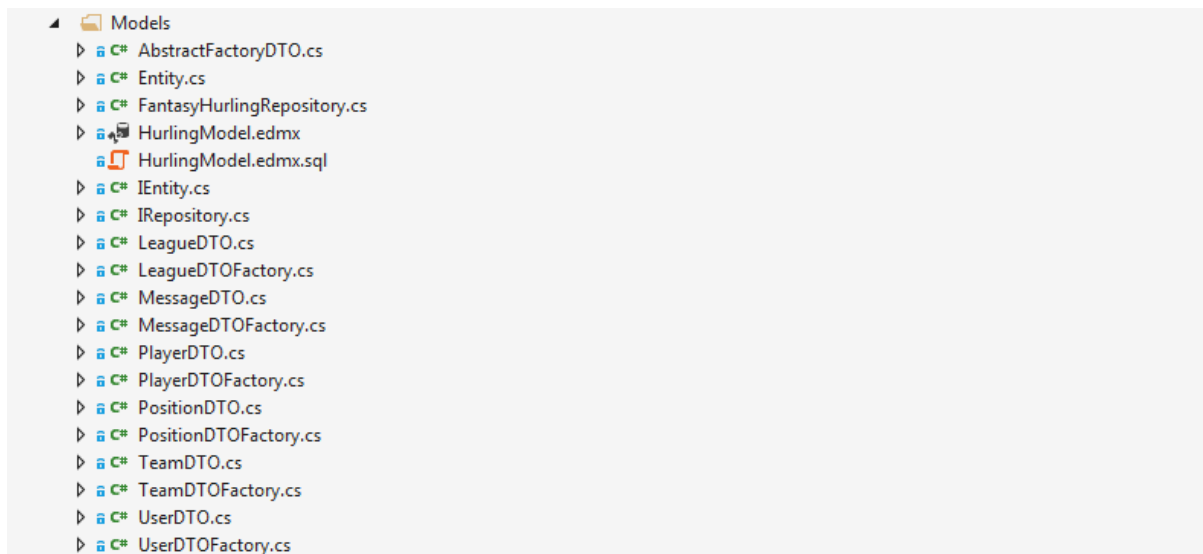


Figure 88 Model Classes

AbstractFactoryDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public abstract class AbstractFactoryDTO<Model, DTO>
        where Model : class
        where DTO : class
    {
        public abstract DTO GetDTO(Model model);
        public abstract Model GetModel(DTO dto);
        public IEnumerable<DTO> GetDTOCollection(IEnumerable<Model> models)
        {
            var DTOs = new HashSet<DTO>();
            foreach (var model in models)
            {
                DTOs.Add(GetDTO(model));
            }
            return DTOs;
        }
    }
}
```

TeamDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace HurlingApi.Models
{
    public class TeamDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public decimal OverAllPoints { get; set; }
        [Required]
        public decimal LastWeekPoints { get; set; }
        [Required]
        public decimal Budget { get; set; }
        [Required]
        public int LeagueId { get; set; }
        [Required]
        public int UserId { get; set; }
    }
}
```

TeamDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class TeamDTOFactory : AbstractFactoryDTO<Team, TeamDTO>
    {
        public override TeamDTO GetDTO(Team model)
        {
            return new TeamDTO
            {
                Id = model.Id,
                Name = model.Name,
                OverAllPoints = model.OverAllPoints,
                LastWeekPoints = model.LastWeekPoints,
                Budget = model.Budget,
                LeagueId = model.LeagueId,
                UserId = model.UserId
            };
        }

        public override Team GetModel(TeamDTO dto)
        {
            return new Team
            {
                Id = dto.Id,
                Name = dto.Name,
                OverAllPoints = dto.OverAllPoints,
                LastWeekPoints = dto.LastWeekPoints,
                Budget = dto.Budget,
                LeagueId = dto.LeagueId,
                UserId = dto.UserId
            };
        }
    }
}
```

LeagueDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace HurlingApi.Models
{
    public class LeagueDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public DateTime NextFixtures { get; set; }
        [Required]
        public byte Week { get; set; }
    }
}
```

LeagueDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class LeagueDTOFactory : AbstractFactoryDTO<League, LeagueDTO>
    {
        public override LeagueDTO GetDTO(League model)
        {
            return new LeagueDTO
            {
                Id = model.Id,
                Name = model.Name,
                NextFixtures = model.NextFixtures,
                Week = model.Week
            };
        }

        public override League GetModel(LeagueDTO dto)
        {
            return new League
            {
                Id = dto.Id,
                Name = dto.Name,
                NextFixtures = dto.NextFixtures,
                Week = dto.Week
            };
        }
    }
}
```

MessageDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace HurlingApi.Models
{
    public class MessageDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Text { get; set; }
        [Required]
        public int UserId { get; set; }
        public Nullable<System.DateTime> Created { get; set; }
    }
}
```

MessageDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class MessageDTOFactory : AbstractFactoryDTO<Message, MessageDTO>
    {
        public override MessageDTO GetDTO(Message model)
        {
            return new MessageDTO
            {
                Id = model.Id,
                Text = model.Text,
                UserId = model.UserId,
                Created = model.Created
            };
        }

        public override Message GetModel(MessageDTO dto)
        {
            return new Message
            {
                Id = dto.Id,
                Text = dto.Text,
                UserId = dto.UserId,
                Created = DateTime.Now
            };
        }
    }
}
```


UserDTO.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class UserDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Email { get; set; }
        [Required]
        public string Username { get; set; }
        [Required]
        public string Password { get; set; }
    }
}
```

UserDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class UserDTOFactory : AbstractFactoryDTO<User, UserDTO>
    {
        public override UserDTO GetDTO(User model)
        {
            return new UserDTO()
            {
                Id = model.Id,
                Email = model.Email,
                Username = model.Username,
                Password = model.Password,
            };
        }

        public override User GetModel(UserDTO dto)
        {
            return new User()
            {
                Id = dto.Id,
                Email = dto.Email,
                Username = dto.Username,
                Password = dto.Password
            };
        }
    }
}
```

PlayerDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace HurlingApi.Models
{
    public class PlayerDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
        [Required]
        public string GaaTeam { get; set; }
        [Required]
        public decimal LastWeekPoints { get; set; }
        [Required]
        public decimal OverallPoints { get; set; }
        [Required]
        public decimal Price { get; set; }
        [Required]
        public byte Rating { get; set; }
        [Required]
        public bool Injured { get; set; }
        [Required]
        public int PositionId { get; set; }
    }
}
```

PlayerDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class PlayerDTOFactory : AbstractFactoryDTO<Player, PlayerDTO>
    {
        public override PlayerDTO GetDTO(Player model)
        {
            return new PlayerDTO
            {
                Id = model.Id,
                FirstName = model.FirstName,
                LastName = model.LastName,
                GaaTeam = model.GaaTeam,
                LastWeekPoints = model.LastWeekPoints,
                OverallPoints = model.OverallPoints,
                Price = model.Price,
                Rating = model.Rating,
                Injured = model.Injured,
                PositionId = model.PositionId
            };
        }

        public override Player GetModel(PlayerDTO dto)
        {
            return new Player
            {
                Id = dto.Id,
                FirstName = dto.FirstName,
                LastName = dto.LastName,
                GaaTeam = dto.GaaTeam,
                LastWeekPoints = dto.LastWeekPoints,
                OverallPoints = dto.OverallPoints,
                Price = dto.Price,
                Rating = dto.Rating,
                Injured = dto.Injured,
                PositionId = dto.PositionId
            };
        }
    }
}
```

PositionDTO.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class PositionDTO
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
    }
}
```

PositionDTOFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HurlingApi.Models
{
    public class PositionDTOFactory : AbstractFactoryDTO<Position, PositionDTO>
    {
        public override PositionDTO GetDTO(Position model)
        {
            return new PositionDTO()
            {
                Id = model.Id,
                Name = model.Name
            };
        }

        public override Position GetTModel(PositionDTO dto)
        {
            return new Position()
            {
                Id = dto.Id,
                Name = dto.Name
            };
        }
    }
}
```

Controllers

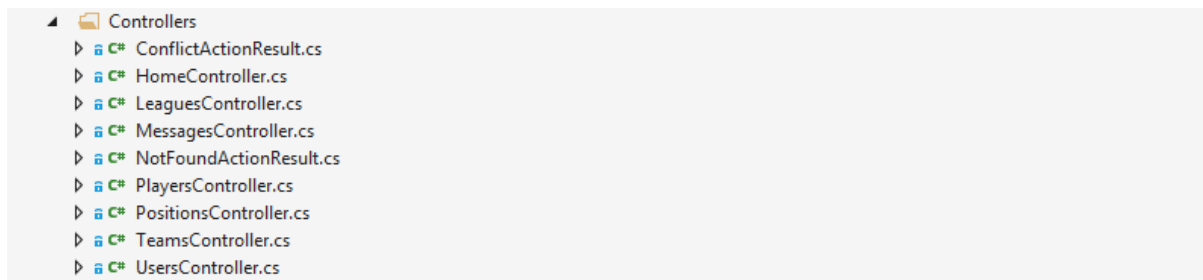


Figure 89 Controller Classes

ConflictActionResult.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Threading;
using System.Threading.Tasks;
using System.Net.Http;
using System.Net;

namespace HurlingApi.Controllers
{
    public class ConflictActionResult : IHttpActionResult
    {
        private readonly string _message;
        private readonly HttpRequestMessage _request;

        public ConflictActionResult(HttpRequestMessage request, string message)
        {
            _message = message;
            _request = request;
        }

        public Task<HttpResponseMessage> ExecuteAsync(CancellationToken
cancellationTokentoken)
        {
            HttpResponseMessage response =
            _request.CreateResponse(HttpStatusCode.Conflict);
            response.Content = new StringContent(_message);
            return Task.FromResult(response);
        }
    }
}
```

HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace HurlingApi.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

LeagueController.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/leagues")]
    public class LeaguesController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly LeagueDTOFactory _factory = new LeagueDTOFactory();
        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<LeagueDTO>> GetLeagues()
        {
            IEnumerable<League> leagues = await _repository.Leagues().GetAllAsync();
            IEnumerable<LeagueDTO> leagueDTOs = _factory.GetDTOCollection(leagues);
            IQueryable<LeagueDTO> oDataLeagueDTOs =
leagueDTOs.AsQueryable<LeagueDTO>();
            return oDataLeagueDTOs;
        }
    }
}
```

```

[Route("id/{id:int}")]
[HttpGet]
public async Task<IHttpActionResult> GetLeagueById(int id)
{
    League league;

    //try to get requested league
    try { league = await _repository.Leagues().FindSingleAsync(l => l.Id ==
id); }

    catch (InvalidOperationException) { throw; }

    //if doesn't exists send not found response
    if (league == null)
    {
        return new NotFoundActionResult(Request, "Could not find league id=" +
id + ".");
    }

    var leagueDTO = _factory.GetDTO(league);

    //send ok response
    return Ok(leagueDTO);
}

[Route("id/{id:int}")]
[HttpPut]
public async Task<IHttpActionResult> EditLeague([FromUri] int id, [FromBody]
LeagueDTO leagueDTO)
{
    //if id from URI matches Id from request body send bad request response
    if (id != leagueDTO.Id)
    {
        return BadRequest("The id from URI: " + id + " doesn't match the" +
        " Id from request body: " + leagueDTO.Id + "!");
    }

    //if model state is not valid send bad request response
    if (!ModelState.IsValid) { return BadRequest(ModelState); }

    League league;

    //try to get requested league
    try { league = await _repository.Leagues().FindSingleAsync(l => l.Id ==
id); }

    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (league == null)
    {
        return new NotFoundActionResult(Request, "Could not find league id=" +
id + ".");
    }

    //leagueDTO is ok, update the league's properties
    league.Name = leagueDTO.Name;
    league.NextFixtures = leagueDTO.NextFixtures;
    league.Week = leagueDTO.Week;
}

```

```

//try to update repository
try { int result = await _repository.Leagues().UpdateAsync(league); }
catch (Exception) { throw; }

//send no content response
return Ok("League Id=" + id + " was successfully updated.");
}

[Route("")]
[HttpPost]
public async Task<IHttpActionResult> PostLeague([FromBody] LeagueDTO
leagueDTO)
{
    //if model state is not valid send bad request response
    if (!ModelState.IsValid) { return BadRequest(ModelState); }

    //get all leagues
    IEnumerable<League> leagues = await _repository.Leagues().GetAllAsync();

    //if there is a league in the repository already send bad request response
    if (leagues.Count() > 0)
    {
        return new ConflictActionResult(Request, "There is already a league in
the repository!" +
            " We allow only one league to exist.");
    }

    var league = _factory.GetModel(leagueDTO);

    //try to insert the league into the repository
    try { int result = await _repository.Leagues().InsertAsync(league); }
    catch (Exception) { throw; }

    //InsertAsync(league) created new id, so leagueDTO must reflect that
    leagueDTO = _factory.GetDTO(league);

    //send created at route response
    return Created<LeagueDTO>(Request.RequestUri + "/id/" +
leagueDTO.Id.ToString() ,leagueDTO);
}

[Route("id/{id:int}")]
[HttpDelete]
public async Task<IHttpActionResult> DeleteLeague([FromUri] int id)
{
    League league;

    //try to get requested league
    try { league = await _repository.Leagues().FindSingleAsync(l => l.Id ==
id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (league == null)
    {
        return new NotFoundActionResult(Request, "Could not find league id=" +
id + ".");
    }
}

```



```

//find out if any team referencing this league exists
    bool exist = await _repository.Teams().ExistsAsync(t => t.LeagueId == id);

    //if exists send bad request response
    if (exist)
    {
        return new ConflictActionResult(Request, "Can't delete league Id=" +
id + ", because there are still "
                                + "some teams referencing it! Delete
all the team in this league first.");
    }

    //try to remove the league from the repository
    try { int result = await _repository.Leagues().RemoveAsync(league); }
    catch (Exception) { throw; }

    //send ok response
    return Ok("League Id=" + id + " deleted.");
}

protected override void Dispose(bool disposing)
{
    if (!_disposed)
    {
        if (disposing)
        {
            _repository.Dispose();
        }

        // release any unmanaged objects
        // set object references to null

        _disposed = true;
    }

    base.Dispose(disposing);
}
}
}

```

MessagesController.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/messages")]
    public class MessagesController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly MessageDTOFactory _factory = new MessageDTOFactory();

        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<MessageDTO>> GetMessages()
        {
            IEnumerable<Message> messages = await
            _repository.Messages().GetAllAsync();
            IEnumerable<MessageDTO> messageDTOs = _factory.GetDTOCollection(messages);
            IQueryable<MessageDTO> oDataMessageDTOs =
            messageDTOs.AsQueryable<MessageDTO>();
            return oDataMessageDTOs;
        }

        [Route("id/{id:int}")]
        [HttpGet]
        public async Task<IHttpActionResult> GetMessageById(int id)
        {
            Message message;

            //try to get requested message
            try { message = await _repository.Messages().FindSingleAsync(m => m.Id ==
            id); }

            catch (InvalidOperationException) { throw; }

            //if doesn't exist send not found response
            if (message == null)
            {
                return new NotFoundActionResult(Request, "Could not find message id="
            + id + ".");
            }
        }
    }
}
```

```

        var messageDTO = _factory.GetDTO(message);
        //send ok response
        return Ok(messageDTO);
    }

    [Route("")]
    [HttpPost]
    public async Task<IHttpActionResult> PostMessage([FromBody] MessageDTO
messageDTO)
    {
        //if the model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        //find out if a user with given id exists in the repository
        bool exist = await _repository.Users().ExistAsync(u => u.Id ==
messageDTO.UserId);

        //if doesn't exists send bad request response
        if (!exist)
        {
            return new ConflictActionResult(Request, "Could not find message Id="
+ messageDTO.UserId + ".");
        }

        //messageDTO is ok, make new message
        var message = _factory.GetModel(messageDTO);

        //try to insert message into repository
        try { int result = await _repository.Messages().InsertAsync(message); }
        catch (Exception) { throw; }

        //InsertAsync(message) created new id, so messageDTO must reflect that
        messageDTO = _factory.GetDTO(message);

        //send created at route response
        return Created<MessageDTO>(Request.RequestUri + "/id/" +
messageDTO.Id.ToString(), messageDTO);
    }

    [Route("id/{id:int}")]
    [HttpDelete]
    public async Task<IHttpActionResult> DeleteMessage([FromUri] int id)
    {
        Message message;

        //try to get a message with given id
        try { message = await _repository.Messages().FindSingleAsync(m => m.Id ==
id); }
        catch (InvalidOperationException) { throw; }

        //if doesn't exists send not found response
        if (message == null)
        {
            return new NotFoundActionResult(Request, "Could not find message id="
+ id + ".");
        }

        //try to delete the message
        try { int result = await _repository.Messages().RemoveAsync(message); }
        catch (Exception) { throw; }
    }

```

```

        //send ok response
        return Ok("Message Id=" + id + " deleted.");
    }

    protected override void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _repository.Dispose();
            }

            // release any unmanaged objects
            // set object references to null

            _disposed = true;
        }

        base.Dispose(disposing);
    }
}

```

NotFoundActionResult.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Threading;
using System.Threading.Tasks;
using System.Net.Http;
using System.Net;

namespace HurlingApi.Controllers
{
    public class NotFoundActionResult : IHttpActionResult
    {
        private readonly string _message;
        private readonly HttpRequestMessage _request;

        public NotFoundActionResult(HttpRequestMessage request, string message)
        {
            _message = message;
            _request = request;
        }

        public Task<HttpResponseMessage> ExecuteAsync(CancellationToken
cancellationTokentoken)
        {
            HttpResponseMessage response =
            _request.CreateResponse(HttpStatusCode.NotFound);
            response.Content = new StringContent(_message);
            return Task.FromResult(response);
        }
    }
}

```

PlayersController.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/players")]
    public class PlayersController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly PlayerDTOFactory _factory = new PlayerDTOFactory();

        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<PlayerDTO>> GetPlayers()
        {
            IEnumerable<Player> players = await _repository.Players().GetAllAsync();
            IEnumerable<PlayerDTO> playerDTOs = _factory.GetDTOCollection(players);
            IQueryable<PlayerDTO> oDataPlayerDTOs =
playerDTOs.AsQueryable<PlayerDTO>();
            return oDataPlayerDTOs;
        }

        [Route("id/{id:int}")]
        [HttpGet]
        public async Task<IHttpActionResult> GetPlayerById(int id)
        {
            Player player;

            //try to get requested player
            try { player = await _repository.Players().FindSingleAsync(p => p.Id ==
id); }

            catch (InvalidOperationException) { throw; }

            //if doesn't exist send not found response
            if (player == null)
            {
                return new NotFoundActionResult(Request, "Could not find player id=" +
id + ".");
            }
        }
    }
}
```

```

        var playerDTO = _factory.GetDTO(player);

        //send ok response
        return Ok(playerDTO);
    }

    [Route("id/{id:int}")]
    [HttpPut]
    public async Task<IHttpActionResult> EditPlayer([FromUri] int id, [FromBody]
PlayerDTO playerDTO)
    {
        //if id from URI matches Id from request body send bad request response
        if (id != playerDTO.Id)
        {
            return BadRequest("The id from URI: " + id + " doesn't match the Id
from " +
                                "request body: " + playerDTO.Id + "!");
        }

        //if model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        Player player;

        //try to get requested player
        try { player = await _repository.Players().FindSingleAsync(p => p.Id ==
id); }

        catch (InvalidOperationException) { throw; }

        //if doesn't exists send not found response
        if (player == null)
        {
            return new NotFoundActionResult(Request, "Could not find player id=" +
id + ".");
        }

        //position can't be changed
        if (player.PositionId != playerDTO.PositionId)
        {
            return new ConflictActionResult(Request, "Changing this player
position could break repository rules. " +
                "There is only one player per position allowed in a team. Ask
Martin.");
        }

        //if lastweekpoints are modified update overallpoints
        if (playerDTO.LastWeekPoints != player.LastWeekPoints)
        {
            playerDTO.OverallPoints += playerDTO.LastWeekPoints;

            //update points in all teams player is in
            foreach (var team in player.Teams.ToList())
            {
                team.OverAllPoints += playerDTO.OverallPoints;
                team.LastWeekPoints += playerDTO.LastWeekPoints;
            }
        }
    }

```

```

        //now playerDTO is ok, set player's properties
        player.FirstName = playerDTO.FirstName;
        player.LastName = playerDTO.LastName;
        player.GaaTeam = playerDTO.GaaTeam;
        player.LastWeekPoints = playerDTO.LastWeekPoints;
        player.OverallPoints = playerDTO.OverallPoints;
        player.Price = playerDTO.Price;
        player.Rating = playerDTO.Rating;
        player.Injured = playerDTO.Injured;
        player.PositionId = playerDTO.PositionId;

        //try to update repository
        try { int result = await _repository.Players().UpdateAsync(player); }
        catch (Exception) { throw; }

        //send no content response
        return Ok("Player Id=" + id + " was successfully updated.");
    }

    [Route("")]
    [HttpPost]
    public async Task<IHttpActionResult> PostPlayer([FromBody] PlayerDTO
playerDTO)
    {
        //if the model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        //find out if a position with given id exists in the repository
        bool exist = await _repository.Positions().ExistAsync(p => p.Id ==
playerDTO.PositionId);

        //if doesn't exists send bad request response
        if (!exist)
        {
            return BadRequest("Position with Id=" + playerDTO.PositionId + "
doesn't exist in the repository.");
        }

        //playerDTO is ok, make new player
        var player = _factory.GetModel(playerDTO);
        player.Id = 0;
        player.LastWeekPoints = 0;
        player.OverallPoints = 0;

        //try to insert player into repository
        try { int result = await _repository.Players().InsertAsync(player); }
        catch (Exception) { throw; }

        //InsertAsync(player) created new id, so playerDTO must reflect that
        playerDTO = _factory.GetDTO(player);

        //send created at route response
        return Created<PlayerDTO>(Request.RequestUri + "/id/" +
playerDTO.Id.ToString(), playerDTO);
    }

```

```

[Route("id/{id:int}")]
[HttpDelete]
public async Task<IHttpActionResult> DeletePlayer([FromUri] int id)
{
    Player player;

    //try to get a player with given id
    try { player = await _repository.Players().FindSingleAsync(p => p.Id ==
id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exists send not found response
    if (player == null)
    {
        return new NotFoundActionResult(Request, "Could not find player id=" +
id + ".");
    }

    //try to delete the player
    try { int result = await _repository.Players().RemoveAsync(player); }
    catch (Exception)
    {
        return new ConflictActionResult(Request, "Deleting player with Id=" +
id + " would break referential integrity " +
        "of the repository. You must first manually delete
this player from all the " +
        "teams he is in. Ask Martin to add automate
cascade deleting feature.");
    }

    //send ok response
    return Ok("Player Id=" + id + " deleted.");
}

protected override void Dispose(bool disposing)
{
    if (!_disposed)
    {
        if (disposing)
        {
            _repository.Dispose();
        }

        // release any unmanaged objects
        // set object references to null

        _disposed = true;
    }

    base.Dispose(disposing);
}
}
}

```


PositionsController.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/positions")]
    public class PositionsController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly PositionDTOFactory _factory = new PositionDTOFactory();
        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<PositionDTO>> GetPostions()
        {
            IEnumerable<Position> positions = await
            _repository.Positions().GetAllAsync();
            IEnumerable<PositionDTO> positionDTOS =
            _factory.GetDTOCollection(positions);
            IQueryable<PositionDTO> oDataPositionDTOS =
            positionDTOS.AsQueryable<PositionDTO>();
            return oDataPositionDTOS;
        }

        [Route("id/{id:int}")]
        [HttpGet]
        public async Task<IHttpActionResult> GetPositionById(int id)
        {
            Position position;
            //try to get requested position
            try { position = await _repository.Positions().FindSingleAsync(p => p.Id
            == id); }
            catch (InvalidOperationException) { throw; }
            //if doesn't exist send not found response
            if (position == null)
            {
                return new NotFoundActionResult(Request, "Could not find position id="
            + id + ".");
            }

            var positionDTO = _factory.GetDTO(position);
            //send ok response
            return Ok(positionDTO);
        }
    }
}
```

```

[Route("name/{name}")]
[HttpGet]
public async Task<IHttpActionResult> GetPositionByName([FromUri] string name)
{
    Position position;

    //try to get requested position
    try { position = await _repository.Positions().FindSingleAsync(p => p.Name
== name); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (position == null)
    {
        return new NotFoundActionResult(Request, "Could not find position
name=" + name + ".");
    }

    var positionDTO = _factory.GetDTO(position);
    //send ok response
    return Ok(positionDTO);
}

[Route("id/{id:int}")]
[HttpPut]
public async Task<IHttpActionResult> EditPosition([FromUri] int id, [FromBody]
PositionDTO positionDTO)
{
    //if id from URI matches Id from request body send bad request response
    if (id != positionDTO.Id)
    {
        return BadRequest("The id from URI: " + id + " doesn't match " +
"the Id from request body: " + positionDTO.Id + "!");
    }

    //if model state is not valid send bad request response
    if (!ModelState.IsValid) { return BadRequest(ModelState); }

    Position position;

    //try to get requested position
    try { position = await _repository.Positions().FindSingleAsync(p => p.Id
== id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exists send not found response
    if (position == null)
    {
        return new NotFoundActionResult(Request, "Could not find position id="
+ id + ".");
    }

    Position position1;

    //try to get a position with same name
    try { position1 = await _repository.Positions().FindSingleAsync(p =>
p.Name == positionDTO.Name); }
    catch (InvalidOperationException) { throw; }

```

```

        //if doesn't exists send not found response
        if (position == null)
        {
            return new NotFoundActionResult(Request, "Could not find position id="
+ id + ".");
        }

        Position position1;

        //try to get a position with same name
        try { position1 = await _repository.Positions().FindSingleAsync(p =>
p.Name == positionDTO.Name); }
        catch (InvalidOperationException) { throw; }

        //if that exist and if it is different that one we are editing send bad
request response
        if (position1 != null && position1.Id != id)
        {
            return new ConflictActionResult(Request, "There is already a position
with Name:" + positionDTO.Name + " in " +
            "the repository! We allow only unique position
names.");
        }

        //positionDTO is ok, update the position
        position.Name = positionDTO.Name;

        //try to update the position in the repository
        try { int result = await _repository.Positions().UpdateAsync(position); }
        catch (Exception) { throw; }

        //send no content response
        return Ok("Position Id=" + id + " was successfully updated.");
    }

    [Route("")]
    [HttpPost]
    public async Task<IHttpActionResult> PostPosition([FromBody] PositionDTO
positionDTO)
    {
        //if model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        Position position;

        //try to get a position with same name
        try { position = await _repository.Positions().FindSingleAsync(p => p.Name
== positionDTO.Name); }
        catch (InvalidOperationException) { throw; }

        //if exists send bad request response
        if (position != null)
        {
            return new ConflictActionResult(Request, "There is already a position
with Name:" + positionDTO.Name + " in " +
            "the repository! We allow only unique position
names.");
        }
    }

```

```

        position = _factory.GetModel(positionDTO);

        //try to insert the position into the repository
        try { int result = await _repository.Positions().InsertAsync(position); }
        catch (Exception) { throw; }

        //InsertAsync(position) created new id, so positionDTO must reflect that
        positionDTO = _factory.GetDTO(position);

        //send created at route response
        return Created<PositionDTO>(Request.RequestUri + "/id/" +
positionDTO.Id.ToString(), positionDTO);
    }

    [Route("id/{id:int}")]
    [HttpDelete]
    public async Task<IHttpActionResult> DeletePosition([FromUri] int id)
    {
        Position position;

        //try to get requested position
        try { position = await _repository.Positions().FindSingleAsync(p => p.Id
== id); }
        catch (InvalidOperationException) { throw; }

        //if doesn't exists send not found response
        if (position == null)
        {
            return new NotFoundActionResult(Request, "Could not find position id="
+ id + ".");
        }

        //check if any player references this position
        bool exist = await _repository.Players().ExistAsync(p => p.PositionId ==
id);

        //if exists send bad request response
        if (exist)
        {
            return new ConflictActionResult(Request, "Can't delete this position,
because there are " +
                                "still some players referencing the position!");
        }

        //try to remove the position from the repository
        try { int result = await _repository.Positions().RemoveAsync(position); }
        catch (Exception) { throw; }

        //send ok response
        return Ok("Position Id=" + id + " deleted.");
    }

    protected override void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {

```

```

        _repository.Dispose();
    }

    // release any unmanaged objects
    // set object references to null

    _disposed = true;
}

base.Dispose(disposing);
}
}
}

```

TeamsController.cs

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/teams")]
    public class TeamsController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly TeamDTOFactory _teamFactory = new TeamDTOFactory();
        private readonly PlayerDTOFactory _playerFactory = new PlayerDTOFactory();

        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<TeamDTO>> GetTeams()
        {
            IEnumerable<Team> teams = await _repository.Teams().GetAllAsync();
            IEnumerable<TeamDTO> teamDTOs = _teamFactory.GetDTOCollection(teams);
            IQueryable<TeamDTO> oDataTeamDTOs = teamDTOs.AsQueryable();
            return oDataTeamDTOs;
        }
    }
}

```

```

[Route("id/{id:int}")]
[HttpGet]
public async Task<IHttpActionResult> GetTeamById([FromUri] int id)
{
    Team team;
    //try to get requested team
    try { team = await _repository.Teams().FindSingleAsync(t => t.Id == id); }
    catch (InvalidOperationException) { throw; }
    //if doesn't exist send not found response
    if (team == null)
    {
        return new NotFoundActionResult(Request, "Could not find team id=" +
id + ".");
    }

    var teamDTO = _teamFactory.GetDTO(team);
    //send ok response
    return Ok(teamDTO);
}

[Route("name/{name}")]
[HttpGet]
public async Task<IHttpActionResult> GetTeamByName([FromUri] string name)
{
    Team team;

    //try to get requested team
    try { team = await _repository.Teams().FindSingleAsync(t => t.Name ==
name); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (team == null)
    {
        return new NotFoundActionResult(Request, "Could not find team name=" +
name + ".");
    }

    var teamDTO = _teamFactory.GetDTO(team);
    //send ok response
    return Ok(teamDTO);
}

[Route("id/{id:int}/players")]
[HttpGet]
public async Task<IQueryable<PlayerDTO>> GetTeamPlayers([FromUri] int id)
{
    Team team;

    //try to get requested team
    try { team = await _repository.Teams().FindSingleAsync(t => t.Id == id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (team == null)
    {
        throw new ObjectNotFoundException("Could not find team Id=" + team.Id
+ ".");
    }
}

```

```

        IEnumerable<PlayerDTO> playerDTOs =
            _playerFactory.GetDTOCollection(team.Players);
        IQueryable<PlayerDTO> oDataPlayerDTOs = playerDTOs.AsQueryable();
        return oDataPlayerDTOs;
    }

    [Route("id/{id:int}")]
    [HttpPut]
    public async Task<IHttpActionResult> EditTeam([FromUri] int id, [FromBody]
TeamDTO teamDTO)
    {
        //if id from URI matches Id from request body send bad request response
        if (id != teamDTO.Id)
        {
            return BadRequest("The id from URI: " + id + " doesn't match " +
                "the Id from request body: " + teamDTO.Id + "!");
        }

        //if model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        Team team;
        //try to get requested team
        try { team = await _repository.Teams().FindSingleAsync(t => t.Id == id); }
        catch (InvalidOperationException) { throw; }

        //if doesn't exists send not found response
        if (team == null)
        {
            return new NotFoundActionResult(Request, "Could not find team Id=" +
id + ".");
        }

        Team team1;

        //try to get a team with same name
        try { team1 = await _repository.Teams().FindSingleAsync(t => t.Name ==
teamDTO.Name); }
        catch (InvalidOperationException) { throw; }

        //if exists and if it is different that one we are editing throw bad
request response
        if (team1 != null && team1.Id != id)
        {
            return new ConflictActionResult(Request, "There is already a team with
name:" + teamDTO.Name + " in " +
                "the repository! We allow only unique team names.");
        }

        //teams can't be moved between users
        if (teamDTO.UserId != team.UserId)
        {
            return new ConflictActionResult(Request, "You cannot change UserID.
It's not allowed to move " +
                "teams between users. Ask
Martin.");
        }

        bool exist;

```

```

        //find out if the league referenced by this team LeagueId exists
        exist = await _repository.Leagues().ExistAsync(l => l.Id ==
teamDTO.LeagueId);

        //if doesn't exist send bad request response
        if (!exist)
        {
            return new NotFoundActionResult(Request, "Could not find league Id=" +
teamDTO.LeagueId + ".");
        }

        //update the team's properties
        team.Name = teamDTO.Name;
        team.OverAllPoints = teamDTO.OverAllPoints;
        team.LastWeekPoints = teamDTO.LastWeekPoints;
        team.Budget = teamDTO.Budget;
        team.LeagueId = teamDTO.LeagueId;
        team.UserId = teamDTO.UserId;

        //try to update the team in the repository
        try { int result = await _repository.Teams().UpdateAsync(team); }
        catch (Exception) { throw; }

        //send no content response
        return Ok("Team Id=" + id + " was successfully updated.");
    }

    [Route("id/{teamId:int}/player/id/{playerId:int}")]
    [HttpPut]
    public async Task<IHttpActionResult> PostTeamPlayer([FromUri] int teamId,
[FromUri] int playerId)
    {
        Team team;
        Player player;

        //try to get requested team and player
        try
        {
            team = await _repository.Teams().FindSingleAsync(t => t.Id == teamId);
            player = await _repository.Players().FindSingleAsync(p => p.Id ==
playerId);
        }
        catch (InvalidOperationException) { throw; }

        //if doesn't exist send not found response
        if (team == null)
        {
            return new NotFoundActionResult(Request, "Could not find team Id=" +
teamId + ".");
        }

        if (player == null)
        {
            return new NotFoundActionResult(Request, "Could not find player Id=" +
playerId + ".");
        }
    }

```



```

        //find out if there is this player already in the team
        bool playerAlreadyInThisTeam = team.Players.Any(p => p.Id == playerId);

        if (playerAlreadyInThisTeam)
        {
            return new ConflictActionResult(Request, "Player with id=" + playerId
+ " is in this team already!");
        }
        //find out if there is a player with same field position (we allow only
one player per position)
        bool positionAlreadyInThisTeam = team.Players.Any(p => p.PositionId ==
player.PositionId);

        if (positionAlreadyInThisTeam)
        {
            return new ConflictActionResult(Request, "There is already a player
with the same position in this team!");
        }
        //find out if team has budget to buy this player
        if (team.Budget < player.Price)
        {
            return new ConflictActionResult(Request, "Your this team budget=" +
team.Budget + " is not big enough " +
                                " to cover this player price=" +
player.Price + ".");
        }

        //decrease the team budget
        team.Budget -= player.Price;

        //add player to this team
        team.Players.Add(player);

        //try to save changes in the repository
        try { int result = await _repository.Teams().SaveChangesAsync(); }
        catch (Exception) { throw; }

        //send ok response
        return Ok("Player with Id=" + playerId + " was added to team Id=" + teamId
+ ".");
    }

    [Route("")]
    [HttpPost]
    public async Task<IHttpActionResult> PostTeam([FromBody] TeamDTO teamDTO)
    {
        //if model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        //find out if a team with same name exist
        bool exist = await _repository.Teams().ExistAsync(t => t.Name ==
teamDTO.Name);

        //if exist send bad request response
        if (exist)
        {
            return new ConflictActionResult(Request, "There is already an team
with name:" + teamDTO.Name + " in " +
                                "the repository!. We allow only unique team
names.");
        }
    }

```

```

        //find out if user the team is referencing exists
        exist = await _repository.Users().ExistAsync(u => u.Id == teamDTO.UserId);

        //check if doesn't exist send bad request response
        if (!exist)
        {
            return new NotFoundActionResult(Request, "Could not find user with
Id=" + teamDTO.UserId + ".");
        }

        //find out if team with same userId exists
        exist = await _repository.Teams().ExistAsync(t => t.UserId ==
teamDTO.UserId);

        //if exists send bad request response
        if (exist)
        {
            return new ConflictActionResult(Request, "User with id=" +
teamDTO.UserId + " already has " +
"a team! We allow only one team per user.");
        }

        //find out if the league this team is referencing exist
        exist = await _repository.Leagues().ExistAsync(l => l.Id ==
teamDTO.LeagueId);

        //check if doesn't exist send bad request response
        if (!exist)
        {
            return new NotFoundActionResult(Request, "Could not find league Id=" +
teamDTO.LeagueId + ".");
        }

        var team = _teamFactory.GetModel(teamDTO);

        //new team points to 0
        team.OverAllPoints = 0;
        team.LastWeekPoints = 0;

        //try to insert the team into the repository
        try { int result = await _repository.Teams().InsertAsync(team); }
        catch (Exception) { throw; }

        //InsertAsync(team) created new id, so teamDTO must reflect that
        teamDTO = _teamFactory.GetDTO(team);

        //send created at route response
        return Created<TeamDTO>(Request.RequestUri + "/id/" +
teamDTO.Id.ToString(), teamDTO);
    }

```

```

[Route("id/{id:int}")]
[HttpDelete]
public async Task<IHttpActionResult> DeleteTeam([FromUri] int id)
{
    Team team;

    //try to get requested team
    try { team = await _repository.Teams().FindSingleAsync(t => t.Id == id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (team == null)
    {
        return new NotFoundActionResult(Request, "Could not find team Id=" +
id + ".");
    }

    //try to remove the team from the repository
    try { int result = await _repository.Teams().RemoveAsync(team); }
    catch (Exception)
    {
        return new ConflictActionResult(Request, "Deleting team with Id=" + id
+ " would break referential " +
        "integrity of the repository. First manually remove all
players " +
        "from this team. Ask Martin for automatic cascade
removal functionality.");
    }

    //send ok response
    return Ok("Team Id=" + id + " deleted.");
}

[Route("id/{teamId:int}/player/id/{playerId:int}")]
[HttpDelete]
public async Task<IHttpActionResult> DeletePlayerFromTeam([FromUri] int
teamId, [FromUri] int playerId)
{
    Team team;

    //try to get requested team
    try { team = await _repository.Teams().FindSingleAsync(t => t.Id ==
teamId); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (team == null)
    {
        return new NotFoundActionResult(Request, "Could not find team Id=" +
teamId + ".");
    }

    var player = team.Players.FirstOrDefault(p => p.Id == playerId);

    if (player == null)
    {
        return new NotFoundActionResult(Request, "Could not find player Id=" +
playerId + ".");
    }
}

```

```

        //return money back to team budget
        team.Budget += player.Price;

        //remove the player from the team
        team.Players.Remove(player);

        //try to save changes in the repository
        try { int result = await _repository.Teams().SaveChangesAsync(); }
        catch (Exception) { throw; }

        //send ok response
        return Ok("Player Id=" + playerId + " was removed from team Id=" + teamId
+ ".");
    }

    protected override void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _repository.Dispose();
            }

            // release any unmanaged objects
            // set object references to null

            _disposed = true;
        }

        base.Dispose(disposing);
    }
}

```

UsersController.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity.Core;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using HurlingApi.Models;
using System.Web.Http.Cors;

namespace HurlingApi.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/users")]
    public class UsersController : ApiController
    {
        private readonly IRepository _repository = new FantasyHurlingRepository();
        private readonly UserDTOFactory _factory = new UserDTOFactory();

        private bool _disposed;

        [Route("")]
        [HttpGet]
        public async Task<IQueryable<UserDTO>> GetUsers()
        {
            IEnumerable<User> users = await _repository.Users().GetAllAsync();
            IEnumerable<UserDTO> userDTOS = _factory.GetDTOCollection(users);
            IQueryable<UserDTO> oDataUserDTOS = userDTOS.AsQueryable<UserDTO>();
            return oDataUserDTOS;
        }

        [Route("id/{id:int}")]
        [HttpGet]
        public async Task<IHttpActionResult> GetUserById([FromUri] int id)
        {
            User user;
            //try to get requested user
            try { user = await _repository.Users().FindSingleAsync(u => u.Id == id); }
            catch (InvalidOperationException) { throw; }

            //if doesn't exist send not found response
            if (user == null)
            {
                return new NotFoundActionResult(Request, "Could not find user Id=" +
id + ".");
            }

            var userDTO = _factory.GetDTO(user);

            //send ok response
            return Ok(userDTO);
        }
    }
}
```

```

[Route("username/{username}")]
[HttpGet]
public async Task<IHttpActionResult> GetUserByUsername([FromUri] string
username)
{
    User user;
    //try to get requested user
    try { user = await _repository.Users().FindSingleAsync(u => u.Username ==
username); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (user == null)
    {
        return new NotFoundActionResult(Request, "Could not find user
Username=" + username + ".");
    }

    var userDTO = _factory.GetDTO(user);

    //send ok response
    return Ok(userDTO);
}

[Route("id/{id:int}")]
[HttpPut]
public async Task<IHttpActionResult> EditUser([FromUri] int id, [FromBody]
UserDTO userDTO)
{
    //if id from URI matches Id from request body send bad request response
    if (id != userDTO.Id)
    {
        return BadRequest("The id from URI: " + id + " doesn't match " +
"the Id from request body: " + userDTO.Id + "!");
    }

    //if model state is not valid send bad request response
    if (!ModelState.IsValid) { return BadRequest(ModelState); }

    User user;

    //try to get requested user
    try { user = await _repository.Users().FindSingleAsync(u => u.Id == id); }
    catch (InvalidOperationException) { throw; }

    //if doesn't exist send not found response
    if (user == null)
    {
        return new NotFoundActionResult(Request, "Could not find user Id=" +
id + ".");
    }

    User user1;

    // try to get user with same username
    try { user1 = await _repository.Users().FindSingleAsync(u => u.Username ==
userDTO.Username); }
    catch (InvalidOperationException) { throw; }

```

```

        //if doesn't exist send not found response
        if (user == null)
        {
            return new NotFoundActionResult(Request, "Could not find user Id=" +
id + ".");
        }

        User user1;

        // try to get user with same username
        try { user1 = await _repository.Users().FindSingleAsync(u => u.Username ==
userDTO.Username); }
        catch (InvalidOperationException) { throw; }

        //if exists and if it is different that one we are editing send bad
request response
        if (user1 != null && user1.Id != id)
        {
            return new ConflictActionResult(Request, "There is already an user
with name:" + userDTO.Username + " in the " +
                "repository! We allow only unique usernames.");
        }

        //userDTO seems ok, update the user's properties
        user.Username = userDTO.Username;
        user.Password = userDTO.Password;
        user.Email = userDTO.Email;

        //try to update the user in the repository
        try { int result = await _repository.Users().UpdateAsync(user); }
        catch (Exception) { throw; }

        //send no content response
        return Ok("User Id=" + id + " was successfully updated.");
    }

    [Route("")]
    [HttpPost]
    public async Task<IHttpActionResult> PostUser([FromBody] UserDTO userDTO)
    {
        //if model state is not valid send bad request response
        if (!ModelState.IsValid) { return BadRequest(ModelState); }

        //find out if there is an user with the same username
        bool exist = await _repository.Users().ExistAsync(u => u.Username ==
userDTO.Username);

        //if exists send bad request response
        if (exist)
        {
            return new ConflictActionResult(Request, "There is already an user
with name:" + userDTO.Username + " in " +
                "the repository. We allow only unique
usernames.");
        }
    }

```

```

        var user = _factory.GetModel(userDTO);

        //try to insert the user into the repository
        try { int result = await _repository.Users().InsertAsync(user); }
        catch(Exception ) { throw; }

        //InsertAsync(user) created new id, so userDTO must reflect that
        userDTO = _factory.GetDTO(user);

        //send created at route response
        return Created<UserDTO>(Request.RequestUri + "/id/" +
userDTO.Id.ToString(), userDTO);
    }

    [Route("id/{id:int}")]
    [HttpDelete]
    public async Task<IHttpActionResult> DeleteUser([FromUri] int id)
    {
        User user;

        //try to get requested user
        try { user = await _repository.Users().FindSingleAsync(u => u.Id == id); }
        catch (InvalidOperationException) { throw; }

        //if doesn't exist send not found response
        if (user == null)
        {
            return new NotFoundActionResult(Request, "Could not find user id=" +
id + ".");
        }

        //find out if a the team referencing this user exist
        bool exist = await _repository.Teams().ExistAsync(t => t.UserId == id);

        //if exists send bad request response
        if (exist)
        {
            return new ConflictActionResult(Request, "Can't delete this user,
because some team still referencing the user!");
        }

        //try to remove the user
        try { int result = await _repository.Users().RemoveAsync(user); }
        catch(Exception) { throw; }

        //send ok response
        return Ok("User Id=" + id + " deleted.");
    }

    protected override void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _repository.Dispose();
            }
        }
    }

```



```
        // release any unmanaged objects
        // set object references to null
        _disposed = true;
    }
    base.Dispose(disposing);
}
}
```

Appendix D: Fantasy Hurling Jerseys





Bibliography

1. Industry Demographics - Fantasy Sports Trade Association. [Online] [Cited: November 21, 2014.] <http://www.fsta.org/?page=Demographics>.
2. Fantasy Premier League - The official fantasy football game of the Barclays Premier League. [Online] [Cited: November 21, 2014.] <http://fantasy.premierleague.com>.
3. Technologies, Market Share Statistics for Internet. Browser market share 2014. [Online] 2014. [Cited: 11 30, 2014.] <http://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qptimeframe=Y&qpcustomb=&qpcustomd=2>.
4. Jonson, Phil. Web developers less concerned about browser-compatibility, more concerned with HTML5. [Online] 2014. [Cited: 11 25, 2014.] <http://www.itworld.com/article/2693441/big-data/web-developers-less-concerned-about-browser-compatibility--more-concerned-with-html5.html>.
5. Gomez. Thriving in a multi-browser world. [Online] 2010. [Cited: 11 21, 2014.] http://www.cio.com.au/campaign/370042?content=%2Fwhitepaper%2F370263%2Fthriving-in-a-multi-browser-world-ensuring-optimal-website-performance-for-all-end-users%2F%3Ftype%3Dother%26arg%3D0%26location%3Dfeatured_list.
6. keynote.com. The Benefits of Third-Party Content Monitoring. [Online] 2014. [Cited: 11 23, 2014.] <http://www.keynote.com/resources/white-papers/benefits-of-3rd-party-content-monitoring>.
7. Kartik Bajaj, Karthik Pattabiraman, Ali Mesbah. Mining Questions Asked by Web Developers. [Online] 2013. [Cited: 11 15, 2014.] <http://salt.ece.ubc.ca/publications/docs/kartik-msr14.pdf>.
8. devrates.com. devrates projects list. [Online] 2013. [Cited: January 1, 2013.] <http://devrates.com/project/list>.
9. Pederson, Paul M. Sport Fandom in the Digital World. [book auth.] Ann Pegoraro. *Routledge Handbook of Sport Cummunication*. s.l. : Routledge.

10. Joris Drayer, Stephen L. Shapiro, Brendan Dwyer, Alan L. Morse, Joel White. The effects of fantasy football participation on NFL (National Football League) consumption: A qualitative analysis. *Sport Management Review*. 2010.
11. *Using Social Media to Build Community*. Kurt Komaromi, Scott Erickson. 2011, Vol. Competition Forum.
12. Podila. HTTP: The Protocol Every Web Developer Must Know. [Online] 2013. [Cited: April 2, 2015.] <http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>.
13. R., Fielding. Hypertext Transfer Protocol. [Online] September 1, 2004. [Cited: April 4, 2015.] <http://www.w3.org/Protocols/rfc2616/rfc2616>.
14. Berners-Lee, T. Uniform Resource Identifiers (URI): Generic Syntax. [Online] August 1998. [Cited: April 5, 2015.] Uniform Resource Identifiers (URI): Generic Syntax.
15. Model, Prototype. What is Prototype model- advantages, disadvantages and when to use it? *ISTQB EXAM CERTIFICATION*. [Online] [Cited: December 15, 2014.] <http://istqbexamcertification.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it>.
16. Microsoft. Storage documentation. [Online] [Cited: December 15, 2014.] <http://azure.microsoft.com/en-us/documentation/services/storage>.
17. —. Azure Regions. [Online] [Cited: December 14, 2014.] <http://azure.microsoft.com/en-us/regions>.
18. Anonymous. Web API. [Online] [Cited: December 15, 2014.] http://en.wikipedia.org/wiki/Web_API.
19. Node.js. [Online] [Cited: December 20, 2014.] <http://nodejs.org>.
20. Microsoft. ASP.NET. [Online] [Cited: December 15, 2014.] <http://www.asp.net/web-api>.
21. Wasson, Mike. Unit Testing Controllers in ASP.NET. *Web API 2*. [Online] 2014. [Cited: April 22, 2015.] <http://www.asp.net/web-api/overview/testing-and-debugging/unit-testing-controllers-in-web-api>.

22. monetate.com. Ecommerce Quarterly EQ2 2014. [Online] [Cited: April 1, 2014.] <http://www.monetate.com/research/#ufh-i-25962265-ecommerce-quarterly-eq2-2014>.

23. W3C. HTML5. *A vocabulary and associated APIs for HTML and XHTML*. [Online] October 28, 2014. [Cited: December 14, 2014.] <http://http://www.w3.org/TR/html5>.

24. —. Introduction to CSS3. *W3C Working Draft, 23 May 2001*. [Online] May 23, 2001. [Cited: December 14, 2014.] <http://www.w3.org/TR/2001/WD-css3-roadmap-20010523>.

25. Mozilla. Ajax. [Online] 2005. [Cited: December 24, 2014.] <https://developer.mozilla.org/en/docs/AJAX>.

26. jQuery Foundation. jQuery. [Online] [Cited: December 23, 2014.] <http://jquery.com>.

27. —. jQuery User Interface. [Online] [Cited: December 24, 2014.] <http://jqueryui.com/>.

28. Sublime Text. Sublime Text. [Online] [Cited: December 25, 2014.] <http://www.sublimetext.com>.

29. Appcelerator, Inc. Aptana. [Online] [Cited: December 14, 2014.] <http://www.aptana.com>.