CM0343 – Large Individual Project

Olympic Diving Results Information System

Interim Report

Matthew Aish (0918770)

14/14/2012

Supervisor: Dr. J. Shao

Moderator: Prof. D. Marshall

Abstract

This report details the current progress of my project- designing and implementing an Olympic Diving Results Information System. The system is a dynamic web based application, powered by the Olympic Data Feed: sets of XML messages that are processed live to a database to reflect competition standings and statistics. Building upon existing solutions, the application will include the integration of user interaction features such as score prediction. At this point in the project, I have developed requirements, analysed this to form a design and implemented a very basic prototype to demonstrate the feasibility of the proposed solution. Conclusions show that this is an approach that can be developed into a full application that will meet the aims, objectives and requirements set.

Contents

Introduction

Overview

Beginning in July 2011, through to the end of the Games of the XXX Olympiad, I worked for LOCOG (London Organising Committee of the Olympic and Paralympic Games) where my role was a Test Analyst and Deputy Results Manager for Diving. A large portion of this role involved testing the various components of the Diving results systems, including:

- The LOCOG games-time website – a web based application providing live results for 109 million unique users (Balfour, 2012) during the games.
- CIS (Commentator Information System) – A system providing live results and statistics to commentators and journalists through touch-screen technology.
- Info -a web application acting as the information hub of the games for accredited media, sports officials and athletes.

The driving force behind these applications is an XML feed known as ODF (Olympic Data Feed). XML messages are sent to these applications by timing and scoring systems located on venue. These messages include data such as participant information, start lists and medallists, but are also used during competition to reflect the current rankings and statistics.

My aim for this project is to implement my own Results Information System for Diving, powered by ODF. The main differing aspect between my project and the real systems implemented at the Olympic Games is that I will be focussing on integrating user interaction (e.g. Allowing users to predict scores) in to the system, where as the real systems were purely designed for information absorption.

Existing Applications

Atos Commentator Information System

(More information at uk.atos.net/en-uk/olympic_games/what_we_deliver/information_diffusion_systems/default.htm)

Atos' CIS was used at the Olympic Games to provide broadcasters and journalists with real time results and statistics. Unfortunately, due to the private nature of this product I am unable to provide any screenshots of it in use. For diving, users are presented with a schedule of the day's events. They then select one and can view either the start list or results for the event. A table is used to display the current rankings and dive scores as the event progresses, and above, 2 boxes are used to display athlete data- one to present the athlete before a dive, and the second to show the athletes detailed scores after a dive.

Positives

- Results update in real time (seconds behind the real event).
- Simple navigation- users simply click on an event and can see all relevant data on one screen.

Negatives

- Detailed scoring is only shown when a result is given- after this the breakdown of each judge's score is unavailable.
- CIS will only show results for events on a single day at a time- information about previous days or future events is unavailable.
- When an athlete receives a score, the box their presentation data was in is empty but is still visible.
- No user interaction.

Locog Games Time Website

(See london2012.com/diving)

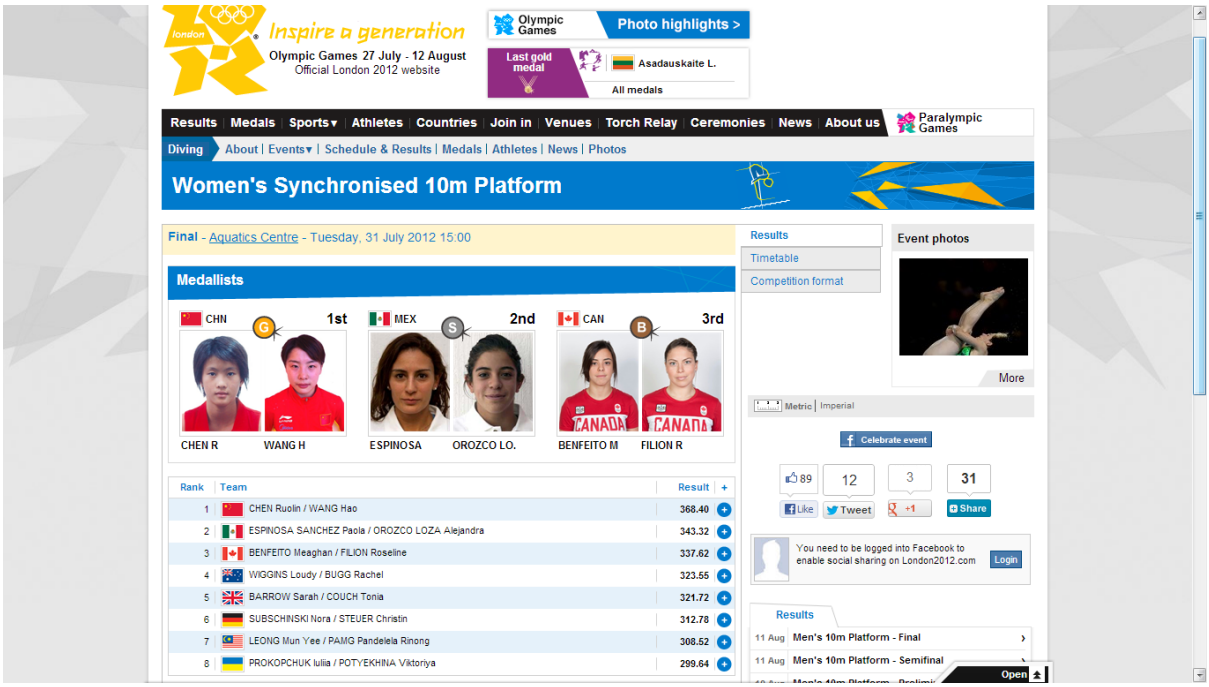
The LOCOG games time website was used at the Olympic Games for users across the world to visit and get the latest news and events. The specific part to focus on in relation to this project is the live results aspect. Similarly to CIS, results are updated live, however, the website allows information to be seen about all events, rather than just those on that day.

Positives

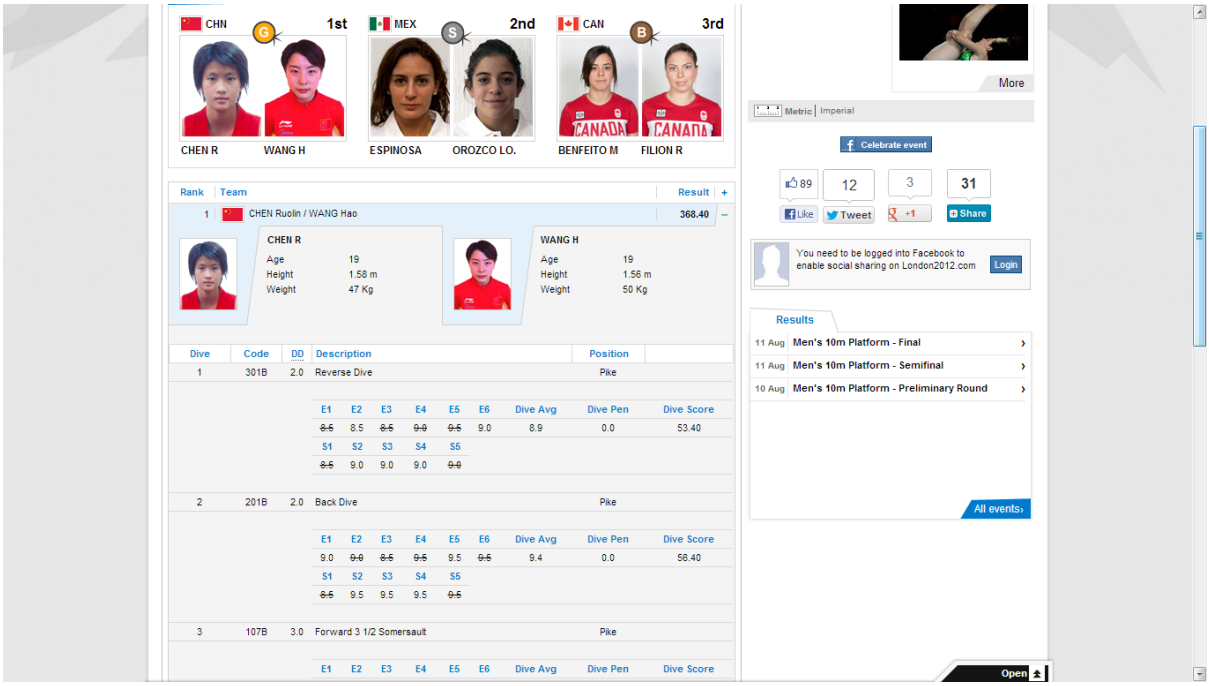
- Can look up all events.
- Detailed information (such as breakdown of scores) is available, but is hidden in expansions, keeping the layout clear but not losing the extra data.

Negatives

- The only user interaction is the ability to share the page via social networking websites.



Results Page



Expansion to view score breakdown

Fantasy Premier League

(See fantasy.premierleague.com)

Fantasy premier league is an online game based on the English Football Premier League. It allows users to pick a team of players, who are then awarded points as football matches progress depending on what happens (points for goals scored, assists etc.). Users can see the match scores updating as they see their own fantasy team points updating. Scores can be compared with other users in fantasy leagues.

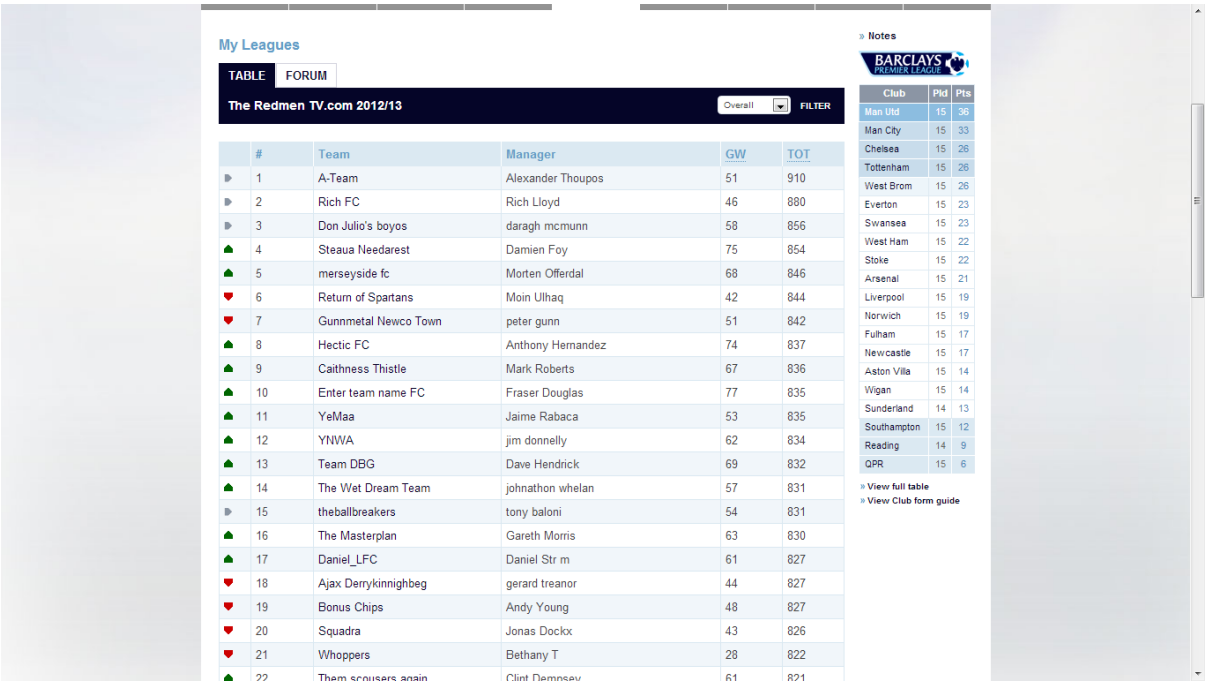
Positives

- Can see current scores and fantasy points updating together.
- Can participate in leagues to see how your points compare to others.

Negatives

- The actual results are placed underneath the fantasy scores so are far down the page. The two entities are quite separate and could be integrated together more successfully.

Points are awarded for each player's performance



Users can compete in leagues to compare their points to others

Deliverables

I have set myself the following deliverables for this project:

- Implement a dummy version of the Diving Olympic Data Feed so that messages can be sent and received, and then processed. As the actual feed is no longer in operation, this will simulate the sending of messages by selecting XML files from test data.

- Parse ODF messages and store the data contained in a database, to allow for meaningful use in the main application.
- Provide a main application interface for users to view results information, including:
    - Schedule information
    - Participant information (entries, team composition, biographical data etc.)
    - Start lists and results
    - Medal winners and final rankings
- Provide results information (previous diver scores, current standings etc.) in real-time as ODF messages are received during simulation of competition.
- Allow users to interact with the system during competition by predicting scores and medallists.

Audience

The audience for the resulting application of this project focuses on sports fans interested in following live results of sporting events. In this case, I am using Olympic Diving as an example due to my experience at LOCOG (my experience being my knowledge of the competition format and rules and knowledge of the data feed) in order to simplify the implementation of what could otherwise become a massively complex project. However, I believe that the idea could be applied to many other sports.

Live results, scores and statistics are hugely popular feeds of information, with thousands of web and mobile applications dedicated to them. However, these applications mainly allow users to absorb information- there is a lack of user interaction. I intend my application to appeal to current users of sports results applications, but to demonstrate how user interaction can be increased, thus also appealing to new users and providing a greater experience for existing ones.

Such an application could be developed to provide results for many different sports- in this case particularly other judged sports similar to diving such as Gymnastics or Synchronised Swimming. The provision of live results for these sports could help to increase coverage for their respective events, but along with the inclusion of greater user interaction, could also help to promote interest. Sports like Gymnastics and Synchronised Swimming often suffer in popularity outside of the Olympic Games. As is evident, systems like this are implemented for large scale events such as the Olympic Games, but greater user interaction could add a new dimension to those systems already in place. Similar systems could also be implemented on a smaller event scale by their respective organisers, thus giving greater accessibility to live information for their sporting events.

Approach

In terms of my approach to this project, I think the best way to subdivide it is into four main areas:

- Simulating the ODF message feed.
- Processing messages and inserting the data into a database.
- Creating a dynamic user interface.
- Integrating user interaction into the user interface.

I plan to take a waterfall approach to development on each of these four areas- designing and implementing one before moving on to the next. I think this is the most sensible approach to take to

implementation, as each area is largely dependent on the success of its predecessor. For example, until I have implemented the sending of XML messages, it is difficult to know exactly how to process them and insert the data into a database. Until the database is complete, it will be difficult to know how to build the user interface. And without a baseline user interface, it will be impossible to incorporate the user interaction features. I therefore see these four areas as the places throughout the project timeline where there is the greatest likelihood that the project direction may change (for example, needing to change my approach to implementation, new ideas for the use of the application etc.). By completing these core elements before beginning work on the next, I believe I will minimise the risk that the direction will change drastically, resulting in completing redundant work, and ultimately minimise risk of project failure. A detailed Gantt chart with a breakdown of tasks and their estimated durations can be found in Appendix A.

Background

Diving

For those unfamiliar with the sport, it may be useful to understand the basic rules of diving in order to fully appreciate and understand the design and implementation of this project. A basic introduction to diving (from www.london2012.com) follows, and for those interested, a complete set of rules and regulations can be found on the website of FINA- the international swimming federation (www.fina.org).

Divers submit in advance the dives which they will perform. The more difficult a dive, the higher the potential score if it is executed correctly: judges award a score out of 10 for each dive, which is multiplied by the dive's degree of difficulty. In the Synchronised Diving events, pairs of athletes dive in tandem and are assessed for their execution and synchronisation by separate groups of judges – a judge never assesses both execution and synchronisation. The higher the judges' score, the higher the diver/team is placed.

Judges (seven for the individual events and 11 for the Synchronised) assess all phases of the dive, including the level of synchronisation. Each judge awards a score out of 10. Divers may score zero on a dive for double-bouncing on the end of the board, performing a dive other than the one they stated or taking too long to dive. Divers also have marks taken off for restarting a dive or armstand.

Dives are divided into four stages – the starting position, the take-off, the dive itself and the entry into the water. Each of these is judged against a strict set of criteria and the winning divers perform difficult and ambitious dives where every part is as near to perfection as possible. Divers aim to enter the water vertically with as little splash as possible. If you see a diver make no splash at all, you can expect very high marks. In the Synchronised events, the two divers must dive in perfect harmony with each other. (LOCOG, 2012)

Results Information Systems

Results information systems are key components of today's modern, data driven sports events. Sports data is constantly changing and customers need to know about these changes immediately. Results information systems collect, store, process and send this data. The scale of these systems varies drastically depending on the event in question. Smaller events may contract a single company,

purely to provide reliable timing and scoring and statistics, where as an event such as the Olympic Games requires multiple companies providing a wide variety of systems that encompass timing and scoring, live results, weather conditions, media conferences and much more.

Results information systems and services are most often provided by specialist companies to sporting event organisers. For example:

- Swiss Timing (www.swisstiming.com): Specialising in timing, scoring, data handling and broadcast solutions to major sports events throughout the world. They provide systems such as venue results systems, accreditation management systems, central database systems storing schedules, participants, start and results lists and medallists as well as sub systems providing clients with an interface to access to this data

- Deltatre (www.deltatre.com): Deltatre offer many different systems to cover all areas of a sporting event including online digital media solutions, on-venue and broadcast systems as well as backstage event management implementations.

In terms of this project, I believe that it may be able to provide a basis for future results information systems and could be developed in to a viable business solution. If this project is successful, it will hopefully demonstrate the possibilities and benefits of integrating live results and user score prediction. The use of ODF is purely as an example due to my experience with it, but we could see similar XML feeds being applied across many other sporting events, with such applications as this one at the front end for users.

Olympic Data Feed

ODF messages form the foundation of this project. They are sent from results systems based on the venue to consuming systems. There are different types of messages to represent different parts of competition data. Some are sent prior to competition, and others are sent during the competition to reflect the live standings of the event. The messages are then forwarded to central systems off the venue, where they are then forwarded to in house systems as well as external clients.

The table below gives a basic explanation of the types of messages that will be used in this project, the data contained in them and when they are triggered/ sent. Detailed documentation as well as sample data can be found at http://odf.olympictech.org. Samples of the messages below can be found in appendix B.

Message Type	Data Contained	Trigger
DT_SCHEDULE / DT_SCHEDULE UPDATE	Schedule information for events, including start dates, start times etc. A flag is used to indicate the status of the event (e.g. scheduled, in progress, official)	Sent once prior to competition. Schedule updates are then sent whenever there is a change.
DT_PARTIC/ DT_PARTIC_UPDATE	Information about all participants in the discipline (names, country, date of birth etc.)	Sent once prior to competition. Participant updates are then sent whenever there is a change.
DT_PARTIC_TEAMS/ DT_PARTIC_TEAMS_UPDATE	Contains names and lists of athletes that form each team.	Sent once prior to competition. Updates are then sent

		whenever there is a change.
DT_START_LIST	Contains the start list for an event including dive lists for each competitor.	Sent when the start order has been drawn, and then again when all dive lists have been submitted.
DT_RT_RESULT	Sends the current ranking and score information for a competitor, and sets the current and previous athlete flags.	Is sent on presentation of an athlete (i.e. the athlete is about to dive) to send current dive information, and sent when the dive is complete to send scores, rank and total.
DT_MEDALLISTS	List the medallists of an event	Sent when results are official.

Project Scope

After careful consideration of my aims and the background, it is important to define the scope of this project.

My aim is to make a working prototype and not a feature-packed application that is ready to be used in the real world. Being based on the Olympic Games, the application has the potential to take an extremely long amount of time, which would be impractical given the time restrictions on this project and that it is only myself working on it. It took over a year to simply test the real results applications, so it would be impossible and impractical to create a complete application here. However, I need to define a scope that still demonstrates that the aims of the project can be met.

I believe a suitable scope for the project is as follows:

- To implement the system using synchronised events only. There are less competitors and phases so this will make it quicker to implement. There is a slightly different structure between synchronised and individual ODF messages, so to implement everything would require a large amount of work which is unnecessary for this prototype.
- Results are the most important aspect of the data. Therefore it is not completely necessary to implement some items I specified in the initial plan. For example: Entry lists, biographies, judge lists. Although this data is important, its implementation would be long and superfluous in this case.
- The key part of the project I want to focus on is creating a form of live results feed that a user can interact with during competition. It is important for me to keep this as the main aim and to not go on a tangent of developing a huge results information system with large quantities of data. The feed contains a large amount of data, which all has value, however I will only be taking the most important data in to my own database for the purpose of this project.
- I will only focus on happy path scenarios. Introducing unhappy scenarios into events such as late changes in start lists, withdrawals, disqualifications, event postponement etc. will massively over complicate the proposed system. These kinds of scenarios are very unlikely in real events, and as stated earlier, took over a year to fully test for the Olympic Games. It

would be impossible to complete a system that took all of these eventualities into account given the time and resources available.

Design

Changes in Approach

Following the definition of the project scope, and given chance to reflect on my approach, I have decided that the approach I described in the introduction and initial plan is not suitable.

I think that the waterfall like approach is not the best way to develop this application, and that it would be much more sensible to work iteratively. I now plan to begin by creating a very simple prototype to test the feasibility of my design, and if this is successful, to then iteratively develop each aspect of it. Although I was correct in identifying that there are four main components to implementation, these are all heavily interlinked. Therefore it is much more suitable to develop each in parallel, allowing me to test and make necessary changes before the project moves too far ahead. If changes need to be made to the fundamental elements of the application, then it will pose a much greater difficulty to do this when some have already been completed. I believe this new approach will minimise the risk of project failure, and will help me to identify risks and changes that need to be made much quicker, with a much smaller impact on the overall project.

Upon reviewing my project plan, I think that I took a naive approach in the beginning. Immediately, I think much more time needs to be spent on the basic design. A detailed design and plan is crucial to a successful project, and I feel that I definitely need to scope more time on this. Although this does leave me with less time for implementation, I believe that with a well thought plan I will not need as much time for this as I am less likely to make mistakes.

I have decided that my aim for the end of this report is to have designed and implemented the basic prototype, after I have completed the basic design steps for the whole application. This will allow me to test the feasibility of my design and give plenty of time to make adjustments to the project if need be. If the prototype is successful, then this will provide me with a strong platform to build the whole application in the New Year.

Adjustments to the project plan Gantt chart can be found in Appendix C.

Requirements

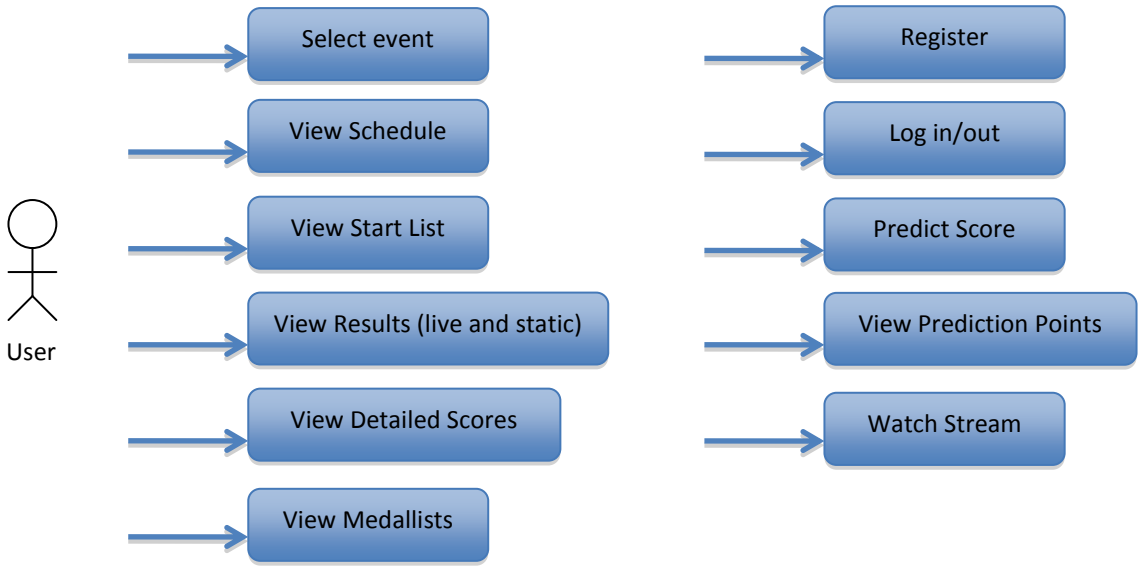
From my aims, and given the background research, I have set the following requirements for my system:

- The application must provide an interface to simulate the sending of ODF messages.
- The application must process the ODF messages needed and store the relevant data.
- The application must provide the ability to view data for all synchronised events at any time.
- The application must be able to retrieve data stored from ODF messages.
- The application must generate and display schedule information.
- The application must generate and display start lists, including dive lists.
- The application must generate and display live results, updating when a change is made to the results data.

- The application must display medallists when available.
- On request, the application must display detailed scoring information.
- The application must allow users to register, sign in and sign out.
- The application must allow signed in users to predict scores as competition progresses.
- The application must award a points score for correct predictions and track a total for the event.
- The application must integrate streaming video (I have decided to include this so that users can fully interact with the system whilst still being able to watch the event).
- The application must be able to show users results of their prediction (i.e. a breakdown of the points gained).
- The application must be able to show a user their ranking in a league table compared to other users.

Requirements Analysis

From the requirements stated above, I have developed the uses cases shown below for a user faced with the front-end of the application. Detailed descriptions of each use case follows.



Select Event:
A user should be able to select one of the diving events (in this case a synchronised event) to view its current state i.e. the start list, results or medallists depending on how far the event has progressed

View Schedule:
A user should be able to view a schedule of the events including basic information such as start times, end times, and if the event is currently running.

View Start List:
A user should be able to view the start list for an event and view basic information such as the teams and start order and the dives they will be performing.

View Results (Live and Static):

During competition, a user should be able to view live results to show the current standings. The screen should then update as the competition progresses. When complete, a user should be able to view a static screen containing the complete results for that event

View Detailed Scores:
Often in diving, results are displayed as overall totals or totals for each dive. A user should be able to find these totals and view a breakdown of each total by each judges score and any penalties that were awarded.

View Medallists:
A user should be able to view the medal winners for an event.

Register:
In order to use prediction features, a user will be able to register- creating a username and password.

Log In/Out:
As users can have a user account, the user should be able to log in and out when desired.

Predict Score:
During competition, a user should be able to input a score prediction that they would award for a given dive.

View Prediction Points:
A user should be able to view the "points" they have attained by correctly predicting the scores awarded by the judges and which athletes will win medals.

Watch Stream:
A user should be able to view a stream of the event/phase in question when it is in progress.

Basic System Components

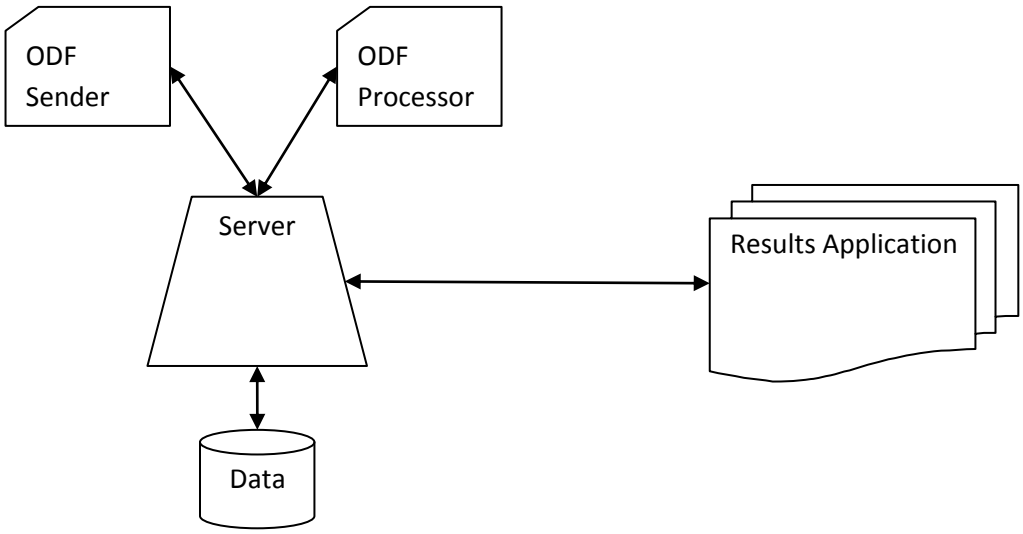
I now need to design the basic components of the system and how they will interact, as well as the technologies I will use.

Architecture

I believe that I have three architectures to choose from:

Option 1

The ODF Sender sends messages and stores them on a server. The message is then retrieved and processed by the ODF Processor and data is inserted in to the database. The results application then reads from the database in order to display information.



Advantages:

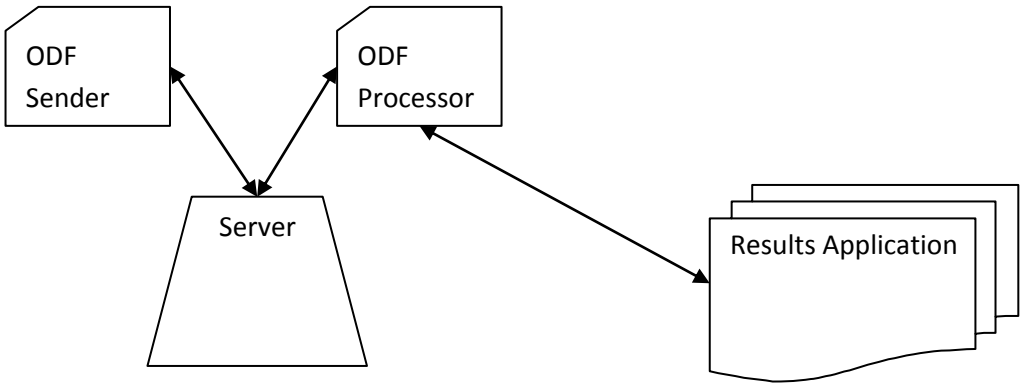
- The system makes logical sense- the data can be separately stored and managed and the components only manipulate this data

Disadvantages:

- The system is complex as it has many different components working in parallel

Option 2

The database is removed, and instead the data is fed directly to the results application by the ODF processor as messages are processed.



Advantages:

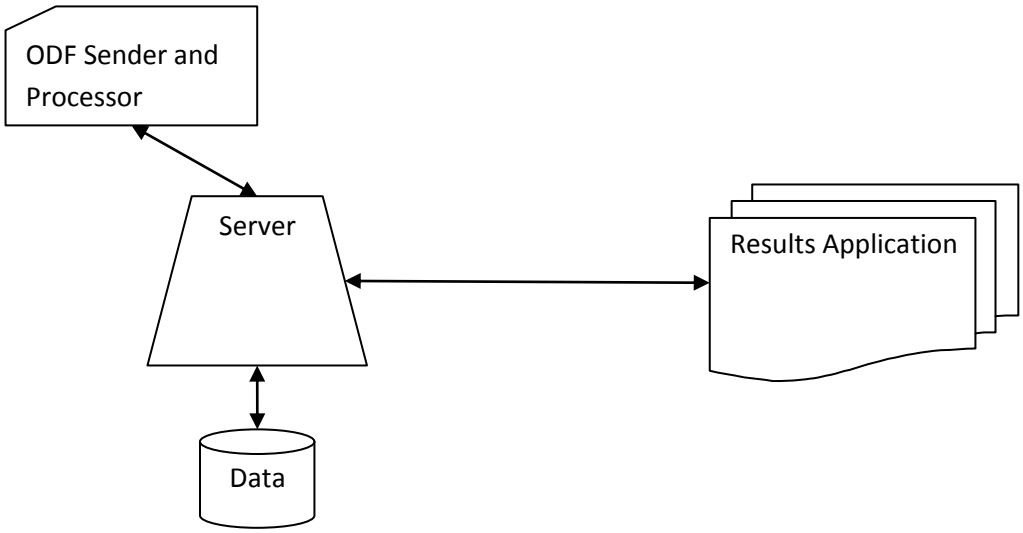
- We remove the need for a database so overhead is decreased

Disadvantages:

- The results application will need to be running in order for updates to be made- with a database this does not need to happen
- It will be difficult to store information about previous and upcoming events without a database
- Although the architecture becomes simpler, the implementation becomes much more complex

Option 3

The ODF Sender and Processor are combined, so that when messages are "sent" they are simply processed and the data is sent to the database.



Advantages:

- Combining the send and processor makes the architecture simpler and removes the need to use the server to communicate between them
- As I'm not using the real feed, it just needs to be simulated and this seems a simpler way to do so

Technologies

I will now look at the potential technologies for each element of the application.

ODF Sender and Processor

1. The sender and processor could be implemented using Java. I have implemented a small XML parser before using a text book so may be able to reuse elements of that program. It is also simple to implement a simple graphical user interface to select and send a file.
2. PHP has a lot of inbuilt XML support, and it also makes it simple to interact with a server as it is a server based language. However, the application would need to be run in a browser which may not be the simplest interface to use and implement.

3. JavaScript could also be used, which has the same benefits and disadvantages as PHP as it is also a browser based scripting language.

Database

1. MySQL is open source, can be run on the server and provides all the basic functionality I will need. Java, PHP and JavaScript can all connect and interact with these types of databases.
2. I could continue to work in Java. Although this would make it easier to communicate with the ODF Sender and Processor if that is in Java as well, it would mean implementing data structures from scratch. I feel this would be too complex and time consuming.

Results Application

1. If I continued with Java, and implemented everything in Java, it would again simplify interaction between the different components. However, I think this is unsuitable as implementing a complex GUI in Java will take a lot of time and learning.
2. PHP will allow me to perform queries on a MySQL database with relative ease and provides many options for processing the results.
3. JavaScript could also be used to access data, however this is bad practice as it is a client side scripting language. PHP is much more suitable as it is a server side scripting language. However, features of JavaScript can be combined with PHP, which may be useful for implementing the GUI.
4. AJAX can be used refresh the live results elements of pages without having to refresh the whole page.

Server

1. Creating a server using free software is relatively simple and can also easily be integrated with PHP and MySQL. Apache is a frequently used piece of free server software that matches this description.

Video Streaming

1. After searching, I have discovered that full streams of the Olympic events are available on YouTube, so it will be simple to embed this streaming media inside PHP pages. An example is here: http://www.youtube.com/watch?v=_lmT4WIK7G0
2. If the videos are removed from YouTube, for the purpose of this project, video clips can be stored on the server and then accessed by users using free software such as JWPlayer- a JavaScript streaming video player. (Avilable at http://www.longtailvideo.com/jw-player/)

Decision

I have decided that the best choice for this project is option 3. I will implement a small Java application to send and process ODF messages, reusing parts of a previous application I have worked with. This will then insert the data into a MySQL database running on an Apache server. The results application will then be browser based, and using PHP pages will access the database to generate pages. The pages will incorporate JavaScript to create a fully functional graphical user interface (yet

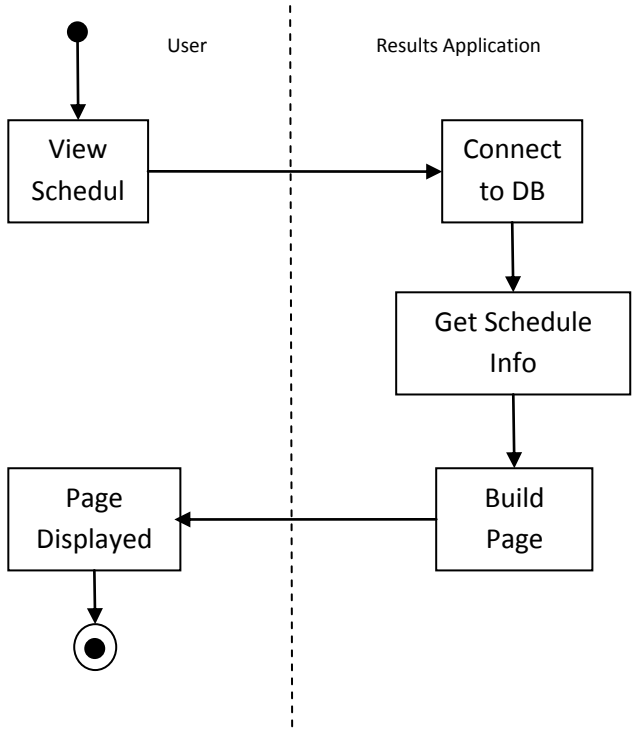
to be designed) and live pages will update asynchronously using AJAX. Streaming video will be embedded from YouTube.

System Flow

For each of the use cases described earlier, I have modelled the flow of the system between its components. The activity diagrams below show how the system will logically progress in order to achieve the results of each use case.

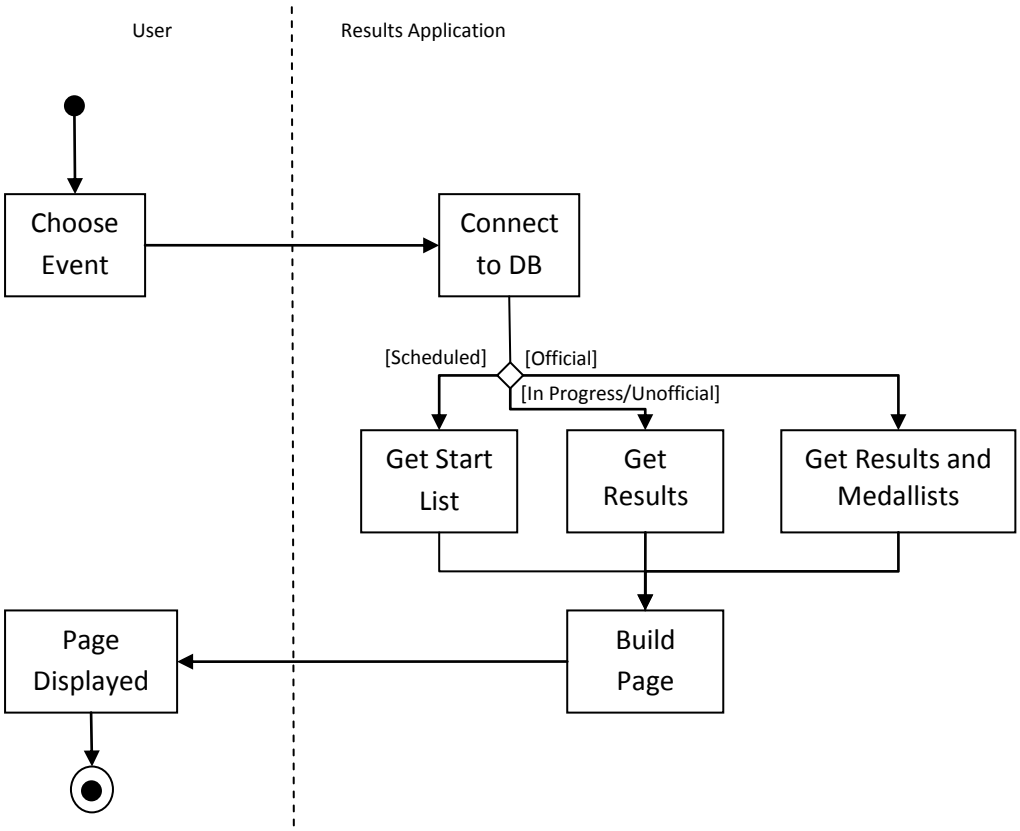
Get Schedule

The user chooses to view the schedule information. The application then connects to the database and retrieves the information for the events in question, and displays this to the user.



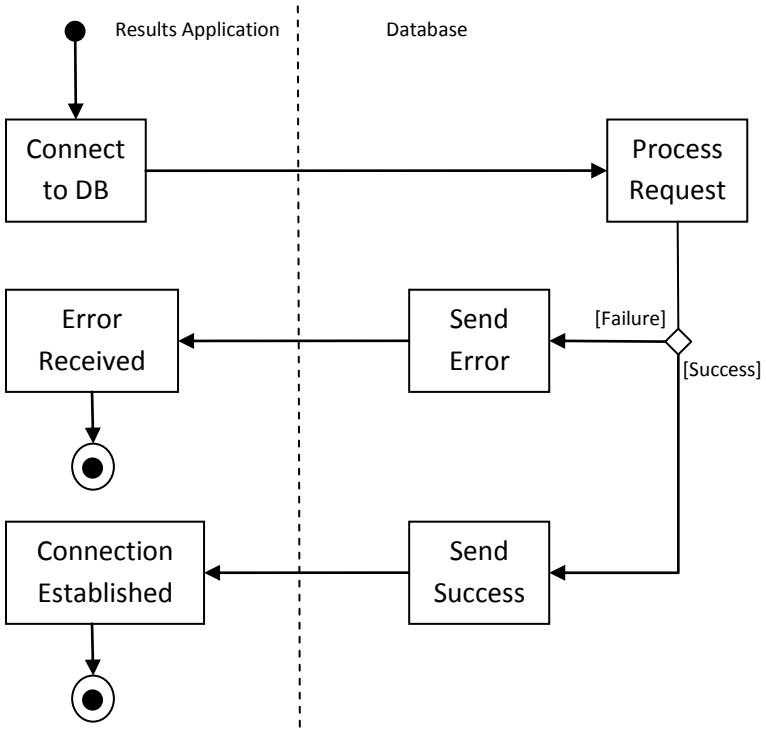
Select Event

A user chooses an event to view. The application then connects to the database and checks the schedule status of the event in question. If it is scheduled then it will show the start list, if it is in progress it will show the current live results, and if official it will show the final results along with the medallists. The page is then built and displayed to the user.



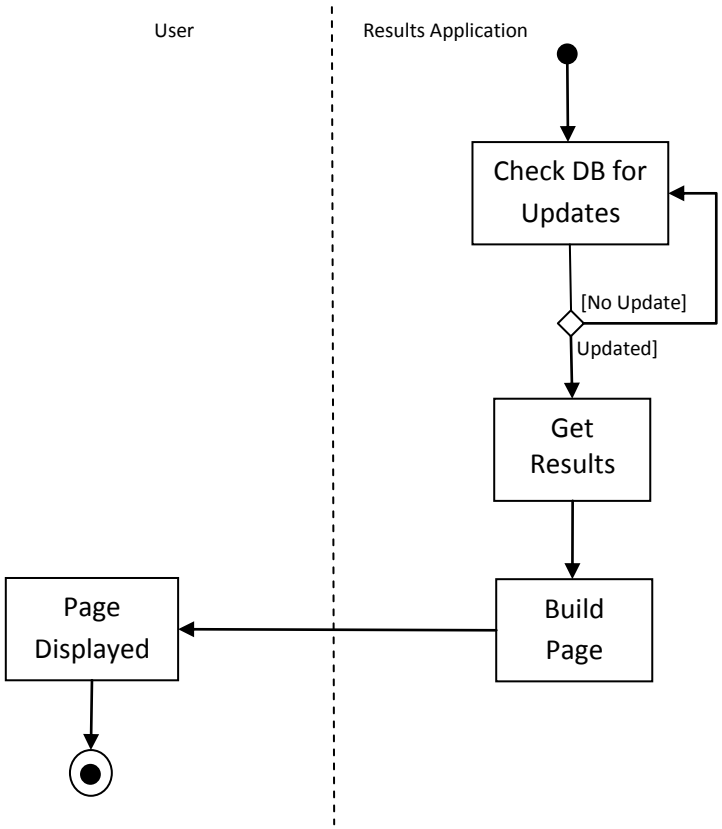
Connect to DB

The application requests to connect to the database. The request is received and either fails or succeeds, returning the outcome to the results application.



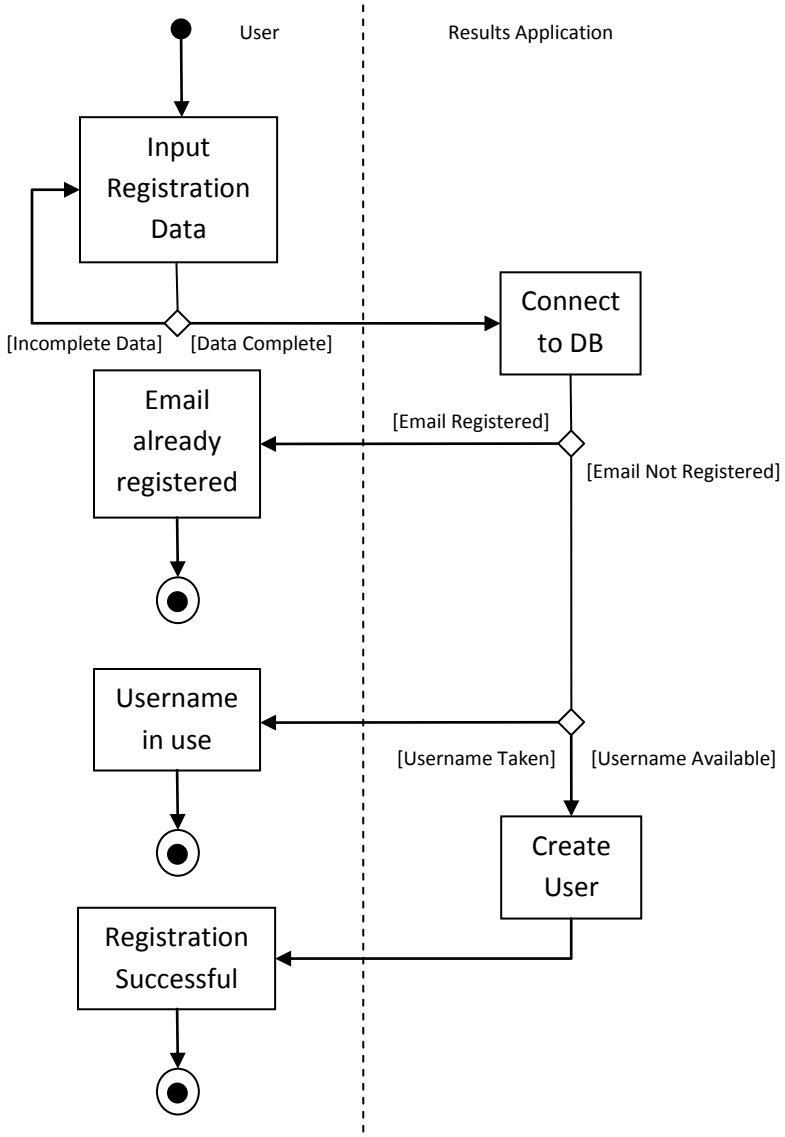
Update Live Results

The application regularly checks the database to see if there has been an update to the results. If there has been an update, the current standings are returned. The page is then built and displayed to the user.



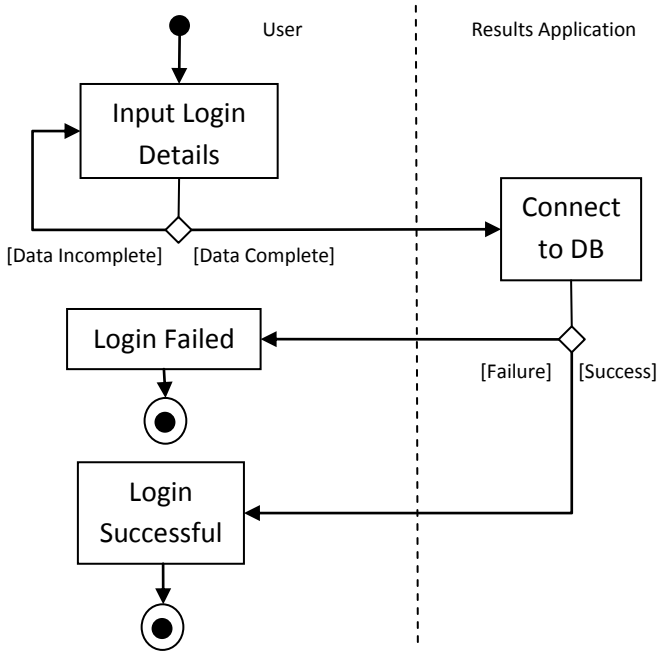
Register

A user inputs their registration information, and if the data is complete the application connects to the database. It then checks if the email address and username specified are already in use, and if not creates the user and notifies them that registration was successful.



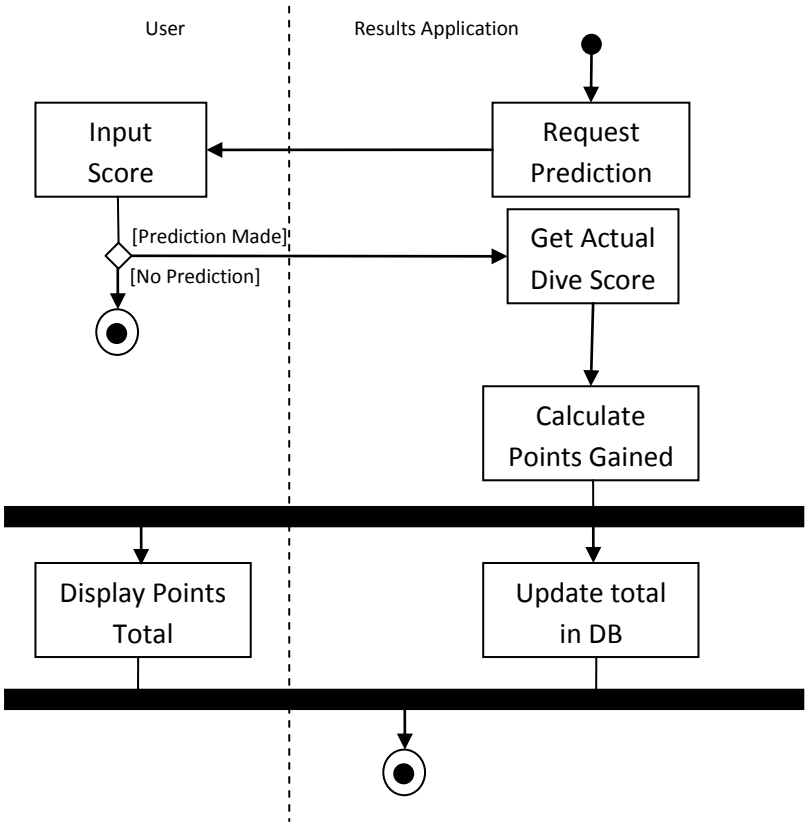
Login

The user inputs their login details. If the details are complete, the application connects to the database and validates the input credentials. The outcome is then returned to the user.



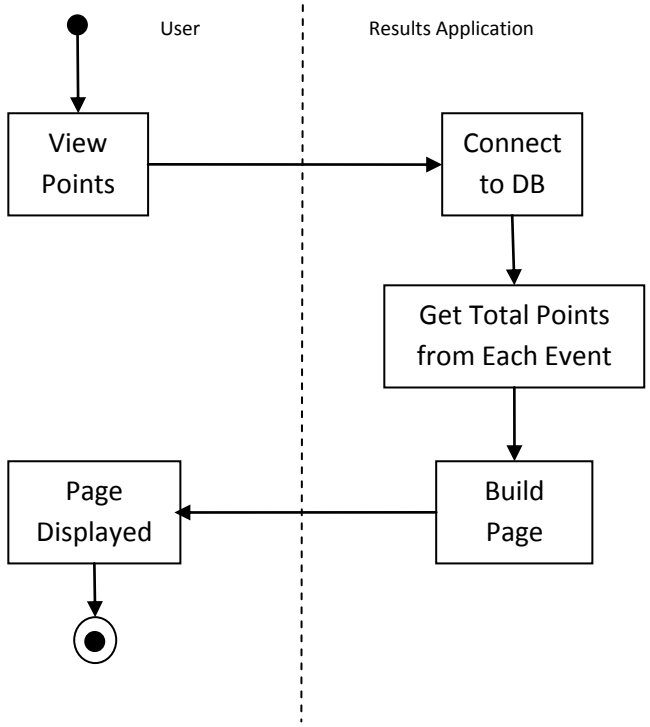
Predict Score

The application requests a score prediction. The user can choose not to, in which case nothing happens. If the user does input a prediction then this is compared to the actual scores and a points total is calculated. This is then displayed to the user and their total points are updated in the database.



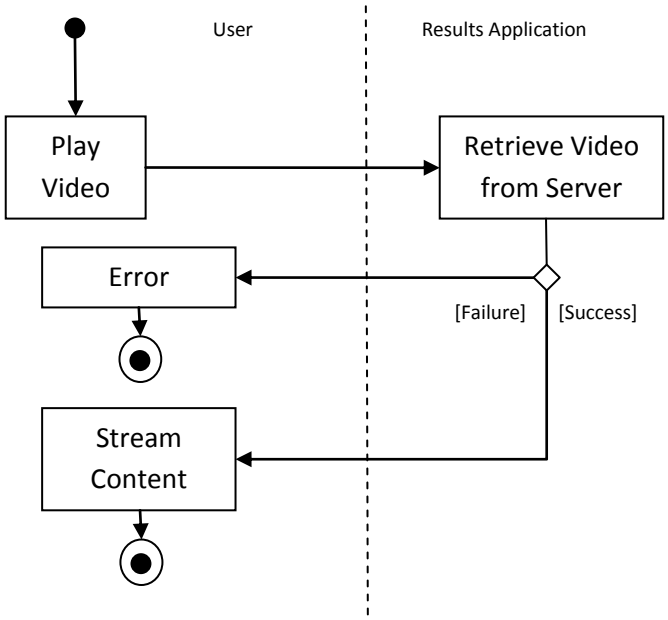
View Points Total

A user requests to view their total points. The application connects to the database and gets the total points gained in each event and returns this to the user.



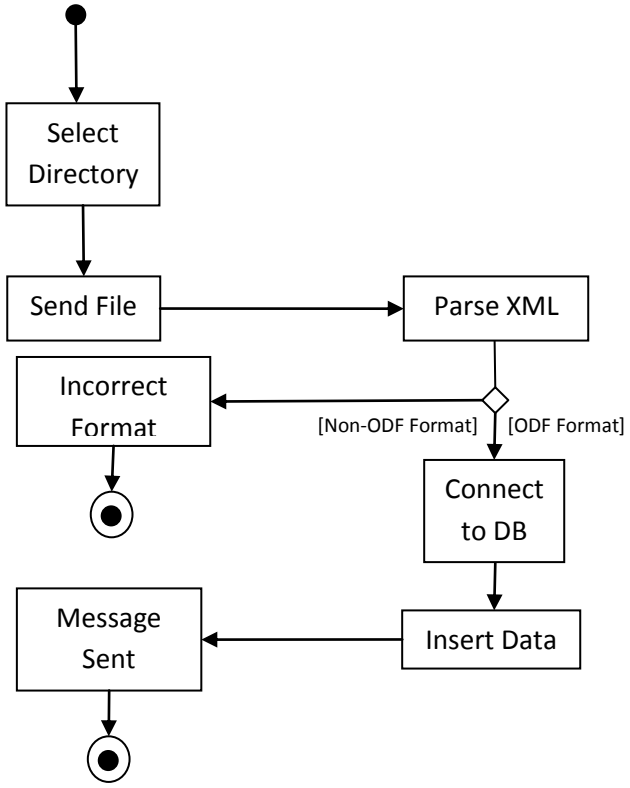
Watch Stream

The user chooses to play the live stream, the video is then retrieved and if this is successful, the content plays.



Sending ODF Messages

In order to send ODF messages, a directory containing the messages is selected. The file is then chosen and sent. The XML is parsed and if the message is the correct format the data is inserted into the database. Success or failure is returned back to the sender.



Data Structures

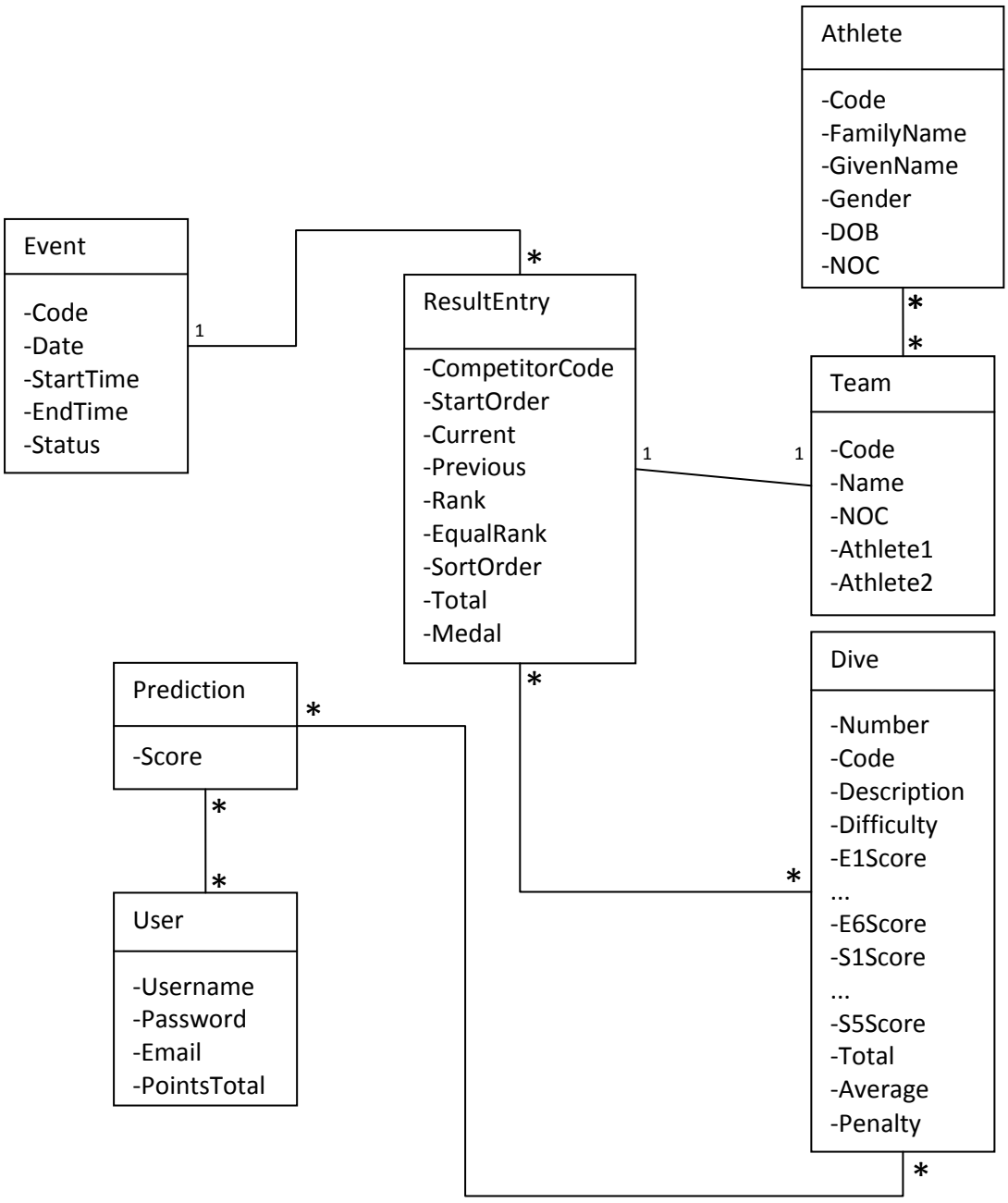
From the requirements, I have decided on the data I will need to obtain from the ODF messages and the data that will come from user input. The table below shows the data I will be storing in the database and its source:

Data	Description	Message
Event Code	A unique identifier for each event (e.g. DVM202101 – Men's 10m synchronised)	Initially populated from DT_SCHEDULE. Each message has an event identifier (DocCode) in the header to specify the event it relates to
Event Date	Date the event starts	DT_SCHEDULE
Event Start Time	Time the event starts	DT_SCHEDULE
Event End Time	Time the event ends	DT_SCHEDULE
Event Status	Flag to indicate if the event is schedule, in progress, official etc.	Initially from DT_SCHEDULE, updated by DT_SCHEDULE_UPDATE
Athlete Code	A unique identifier for each athlete	DT_PARTIC
Athlete Family Name	Family name of the athlete	DT_PARTIC

Athlete Given Name	Given name of the athlete	DT_PARTIC
Athlete Gender	Gender of the athlete	DT_PARTIC
Athlete Date of Birth	Date of birth of the athlete	DT_PARTIC
Athlete NOC	The National Olympic Committee that athlete represents (i.e. the nation they are competing for)	DT_PARTIC
Team Code	A unique identifier for each team that specifies the event they are competing in (e.g. DVM201CAN01 Canada's Men's 3m Synchronised Team.	DT_PARTIC_TEAMS
Team NOC	The National Olympic Committee the team represents	DT_PARTIC_TEAMS
Team Name	The name of the team (in this case the full name of the NOC)	DT_PARTIC_TEAMS
Team Athlete 1	The ID of the first athlete in the team	DT_PARTIC_TEAMS
Team Athlete 2	The ID of the second athlete in the team	DT_PARTIC_TEAMS
Result Competitor	The ID of the team a result relates to	DT_START_LIST, then updates from DT_RT_RESULT
Result Event	The ID of the event a result relates to	DT_START_LIST, then updates from DT_RT_RESULT
Start Order	The start order of the team in an event	DT_START_LIST
Current	Flag to indicate if the team is the current team diving	DT_RT_RESULT
Previous	Flag to indicate if the team was the last team to receive a score	DT_RT_RESULT
Rank	The current ranking of a team	DT_RT_RESULT
Equal Rank	A flag to indicate if the teams rank is equal (e.g. =2)	DT_RT_RESULT
Sort Order	The order that teams are sorted by (used in the case that ranks are equal so we can still sort all the teams)	DT_RT_RESULT
Total	The current total score of a team	DT_RT_RESULT
Medal	A flag set to indicate the medal, if any, associated with an team's rank at the end of competition	DT_MEDALLISTS
Dive Number	A number used to identify which number the dive is in a team's dive list	DT_START_LIST
Dive Code	A code used to identify the different dives (e.g. 407B)	DT_START_LIST
Dive Description	A text description of the dive	DT_START_LIST
Dive Difficulty	The difficulty rating of the dive	DT_START_LIST

Dive Scores (E1...E6, S1...S5)	The scores for each dive awarded by each judge (scores will be stored individually)	DT_RT_RESULT
Total	Total score for the dive	DT_RT_RESULT
Average	Average score for the dive	DT_RT_RESULT
Penalty	Penalty awarded, if any, for the dive	DT_RT_RESULT
Username	A users chosen username	Form input
Password	A users chosen password	Form input
Email Address	A users registered email address	Form input
Points Total	A users total points gained through predictions	Calculated by application
Predicted score	The predicted score by a user	Input by user
Actual score	Used to compare predicted scores for feedback	RT_RESULT (same as average score)

From this, I have modelled the following class diagram to show the relationship between the data.



When it comes to implementing these relationships in the database, it will be better to implement them using the relational model rather than an object-oriented model because:

- There is mostly numerical data (scores and ranks) and related data (names and descriptions) so storing this kind of data in tuples is a widely adopted practice.
- The database will be largely used for looking up values- using keys is a very efficient way to do this.
- The data is not really based on a real world object as it is mostly statistical and numerical
- Data persistence is not important as records will be accessed and changed frequently

So a relational model of this gives the following tables (Primary Keys underlined, Foreign Keys Bold):

Events (<u>Code</u>, Date, StartTime, EndTime, Status)

Athletes (Code, FamilyName, GivenName, Gender, DOB, NOC)

Teams (Code, NOC, Name, **Athlete1**, **Athlete2**)

Results (**Competitor, Event**, StartOrder, Current, Previous, Rank, EqualRank, SortOrder, Total, Medal)

Dives (**Competitor**, Number, Code, Description, Difficulty, E1Score....S5Score, Total, Average, Pentalty)

Users (Username, Password, Email, PointsTotal)

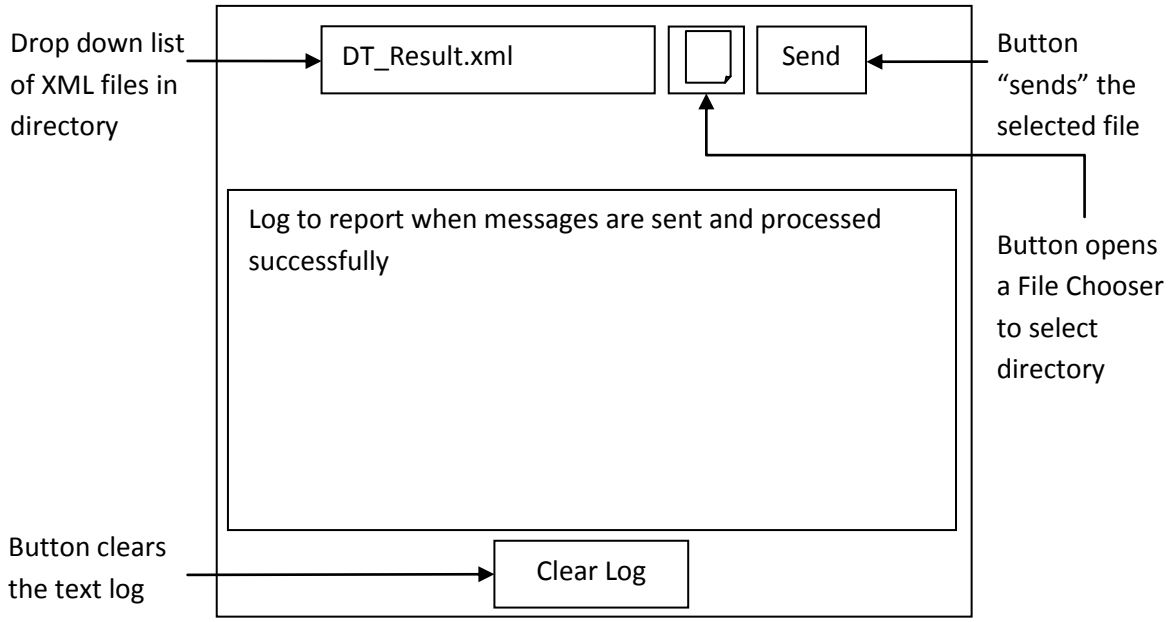
Predictions (**Username, Competitior, Number**, Score, Actual)

Basic Navigation and Layout

To help visualise the application, I will begin planning the basic layout and navigation of the user-facing front end. This will aid in creating the overall image of the application, and is a good place to start building the user interface. These plans are very basic, but provide a platform to begin full design and development of the user interface at a later date.

ODF Sender

The ODF Sender will be a very basic tool, simply to simulate sending messages by processing XML files and inserting data into the database. This part of the application will be a simple window with a drop down list to select files, a button to open a file chooser window and select a directory, and a button to send the selected file. There will also be a text log to show a history of sent files and if the sending was successful or not. There is a button to clear the text log if the sender desires.



Results Application Pages and Navigation

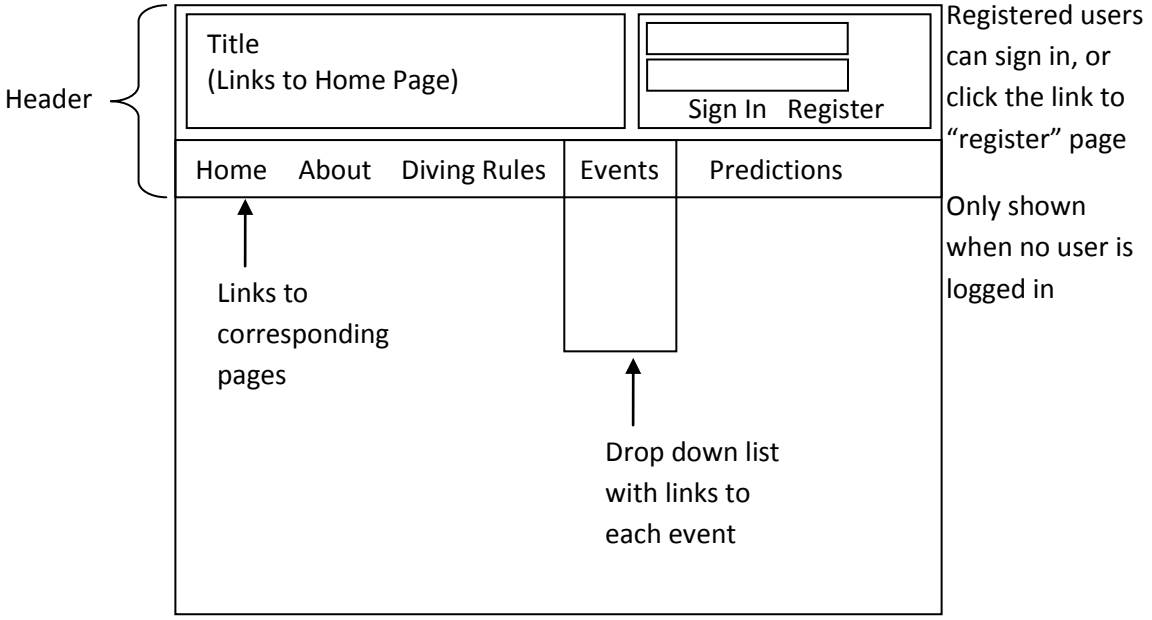
For the user-facing, web-based part of the results application, I believe I will need to include the following pages:

- A home page, to act as the index
- An “About” page, to describe the application and how prediction works
- A page describing the rules of diving for those unfamiliar
- A page for each event to display start lists, results and medallists
- A predictions page where users can view a breakdown of their predictions and points gained
- A page for users to register on

Each page will share a header that will allow access to all other pages, and will also include a section to log in/ register if the user is not logged in. A menu in the header will allow all the pages to be accessible from any other page.

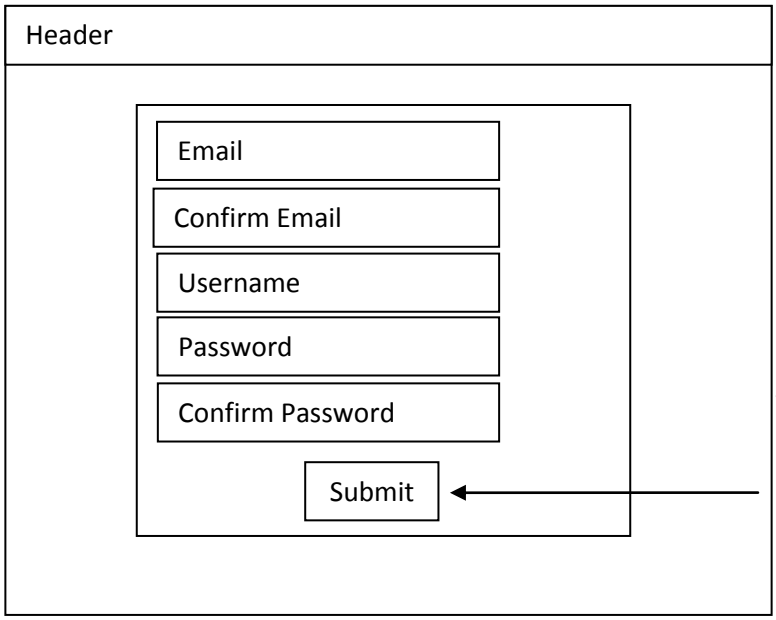
Header and Home Page Layout

The header features at the top of every page. It contains links to every other page- the links to event pages are under a drop down menu. The main content of the home page will be textual. The link to the register page is contained in the sign in box.



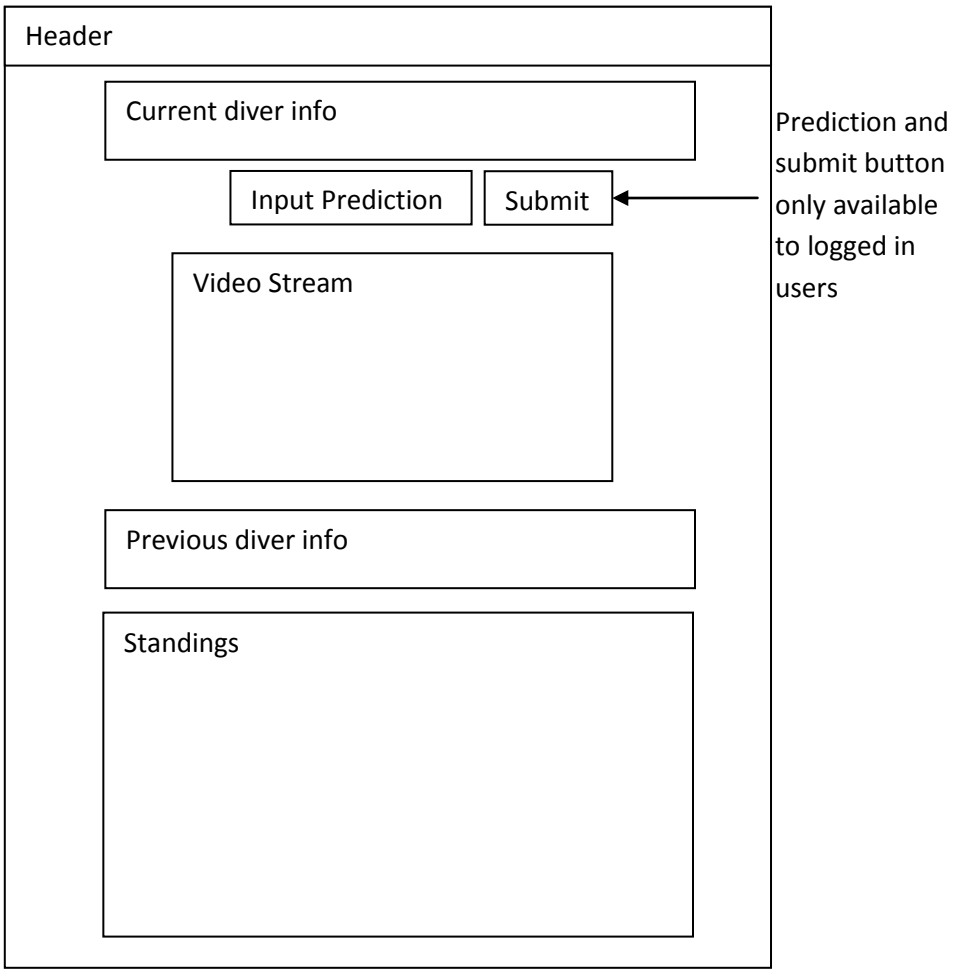
Register Page

The register page provides users with a familiar form to complete to allow them to register and use the prediction features.

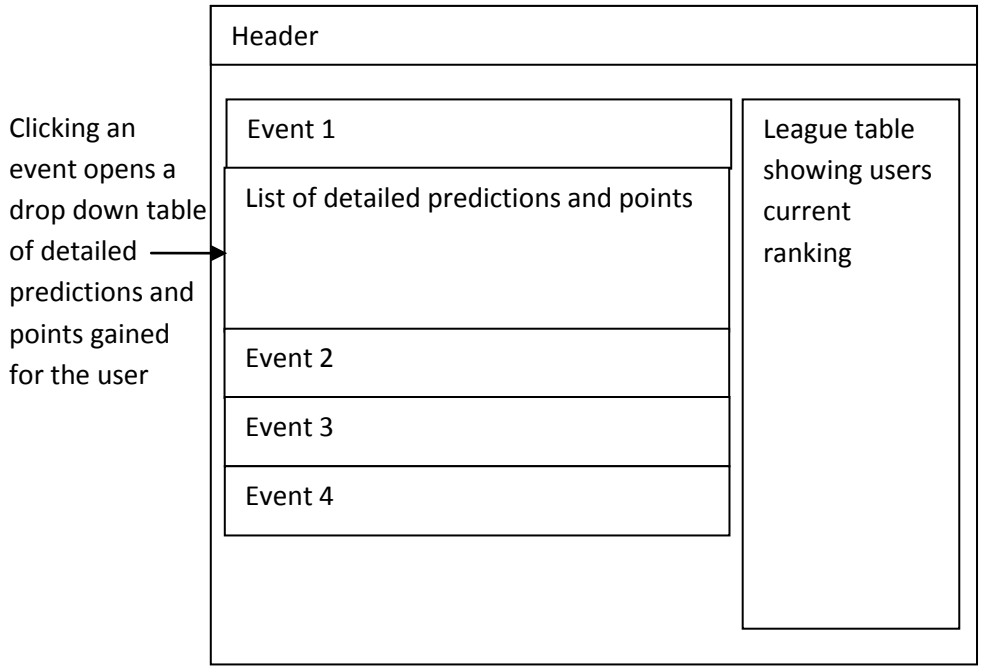


Event Page

The event pages contain the information for the current diver and the fields to predict scores at the top of the page. As the application aims to focus on user interaction I think that the top of the page is the area where this will be noticed the most and is therefore most encouraging for users to use. Below this is the streaming video of the event- this remains near the top of the page as users will not want to scroll in order to predict scores and still watch the event. Below this is the score information for the previous diver- so the user should be able to view these main 3 elements (and the most important elements) without scrolling. The user can then scroll down to see a detailed ranking standings table.


Header
Current diver info
Input Prediction
Submit
Video Stream
Previous diver info
Standings
Prediction and submit button only available to logged in users

Predictions Page



Other Pages

The “About” and “Diving Rules” pages will have the same structure as the home page, as they will only contain textual content in the main body.

Prototype

I am now going to implement a very basic prototype in order to test the feasibility of my design and as a basis to then continue the development into the complete application.

My aim for the prototype is to implement a basic PHP page that can display a start list to screen, with dive information for each team. This will require me to implement the ODF Sender, process DT_PARTIC, DT_PARTIC_TEAMS and DT_START_LIST messages, insert this data into the database and then retrieve and print this data to a browser using PHP.

I think this will be a suitable prototype because it tests communication between the main components of the application that are fundamental to its correct operation. Although this prototype will not include any of the prediction features and graphical user interface of the results application, I do not feel these features are completely necessary to include as they do not need as proof of functionality- we know a graphical user interface can be applied to a PHP page using CSS, and that PHP can capture form input from users. Once I can confirm that connecting to and querying the database works correctly, I am confident that these prediction features can be implemented and that stylistic elements can be applied. At this point, I think the functionality is the most important thing to focus on, because if it fails then the design may need a lot of readjustment.

ODF Sender

The ODF Sender is divided into 3 main functions. The message is sent from the GUI, then parsed into a tree of ODF Elements, which is then processed by the ODF Processor class to get the required data and is inserted into the database. ODF Element is my own class, which is comprised of the element name, and a vector containing ODF Attributes. Each attribute is a pair of strings- the name of the attribute and its value. The pseudo code and explanations below gives more detail on the implementation (complete code listings can be found in Appendix D):

ODF Sender

```
If (SendButton)
       Get Selected File
       Add Filename to Log
       Create SAXTreeViewer
       Tree = SAXTreeViewer.init (File)
       Create ODFProcessor
       ODFProcessor.Process (tree)
End if
```

If the send button is pressed, the application gets the file that is selected and adds its details to the log. A SAXTreeViewer is then created (more below about this class), and is initialized on the file. This returns a tree of OdfElements. An ODFProcessor is then created which processes the tree.

SAX Tree Viewer

```
StartElement()
       Get Current Tree Node
       Get Name of Element at Current Tree Node
       New ODFElement (Element Name)
       For each attribute
              New ODFAttribute
              Add attribute to ODFElement
       End For
       Create tree node containing ODFElement
       Add node to tree
End StartElement()

EndElement()
       Go back to parent node
End EndElement()
```

The SAXTreeViewer is a class created by Brett McLaughlin and is demonstrated in the book “Java & XML”. The original application uses SAX (Simple API for XML), a free Java library that provides XML parsing support (available at.saxproject.org). It allows us to instantiate a reader, which is then instructed to parse an XML document. The SAX parser has content handlers registered to it, which allow application code to be executed as the XML data is being parsed. The handlers that are of most interest here are the StartElement() and EndElement() handlers, which are executed whenever the parses encounters the start of an element and the corresponding ending. The original application builds a default tree model. When it sees the start of an element it creates a tree node containing the element name and adds sibling nodes for all of the attributes. When the tree model is created it is then turned into a JTree and displayed in a frame.

I have changed the application slightly. It creates the tree model, however, when the start of the element is found, it creates a new ODFElement from the element name, and then loops through the

array of attributes returned by the parser. In this loop it creates an ODFAttribute for each attribute, and adds it to the ODFElement. The ODF Element is then added to a tree node, which is added to the tree When the element ends, we move back up the tree so we can add the next element. Instead of creating a JTree, the tree model is returned so that it can be processed by ODFProcessor.

ODF Element

```
OdfElement (String Name, Vector<OdfAttribute> Attributes)
       AddAttribute(OdfAttribute)
              Add OdfAttribute to Vector
       End AddAttribute()

       GetElementName()
              Return name of Element
       End GetElementName()

       GetAttValue(SearchTerm)
              For every attribute in Vector
                     Get Attribute Name
                     If Name == SearchTerm
                            Get Attribute Value
                            Break For Loop
                     End If
              End For
              Return Attribute Value
       EndGettAttValue()
End OdfElement
```

The OdfElement class is my own class, used to store and access information about OdfElements. The Element is made up of a name and a Vector containing OdfAttributes. Its methods allow us to add an attribute to the vector, return the name of the element, and return the value of a given attribute. The latter is achieved by searching through the attributes, and if the name of an attribute matches the attribute we are searching for then the value of that attribute is returned.

ODF Attribute

```
OdfAttribute(String Name, String Value)
       GetAttributeName()
              Return Name
       End GetAttributeName()

       GetAttributeValue()
              Return Value
       End GetAttributeValue()
End OdfAttribute
```

The OdfAttribute class is simply a pair of strings- the name of the attribute and the value of the attribute. It has a method to return each of these values.

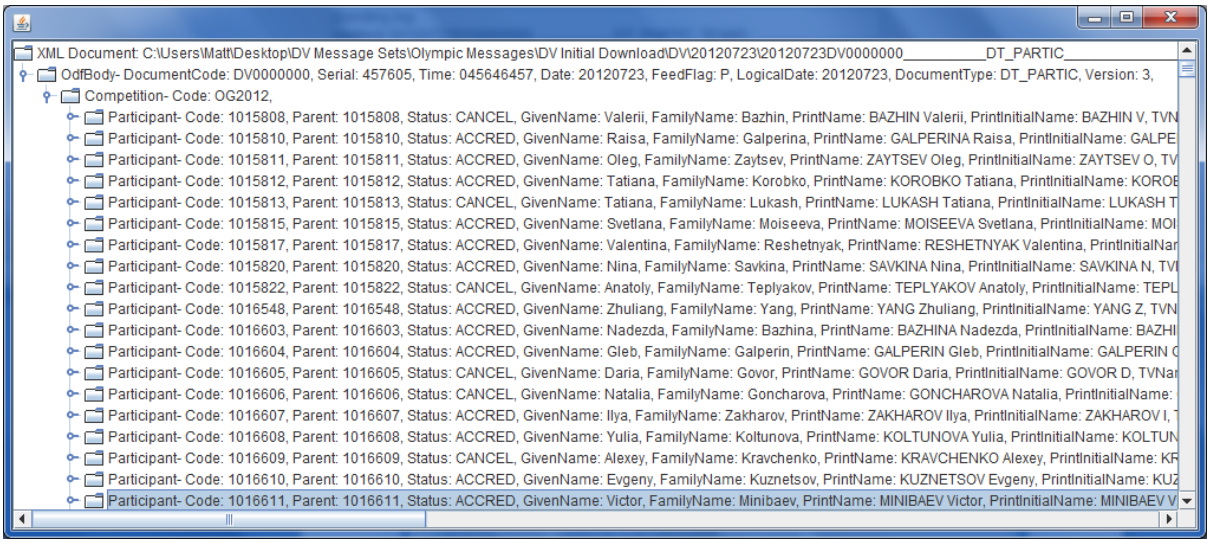
ODF Processor

```
Processor.init()
       Connect to Database
End init

Processor.process(tree)
       Set root node
       Go to ODFBody
       Get DocType and and DodCode
```

When the processor is initialised, it makes a connection to the database. The process method then takes the tree model supplied to it and gets the Document Type and Document Code from the ODFBody Element. This tells us what type of message it is and what event it relates to. The processor then checks the DocType, and executes different methods depending on which message type it is, as they all have a different structure.

```
If DocType == DT_PARTIC
	Go to competition element
	Go to participant element
	For all participants
		If participant is accredited and function is athlete
		Get id, family name, given name, noc, gender and dob
		Insert into athletes table
		End If
	End For
End If
```

If the message is DT_PARTIC, we navigate to the participant element, and then loop through all sister nodes on the tree. This allows is to loop through all the participants, where we can get their data (provided it is an accredited athlete, there is data for other "participants" in the message such as coaches and officials) and then insert it into the database using the connection that was made earlier in the class.



Graphical version of the tree structure created from DT_PARTIC

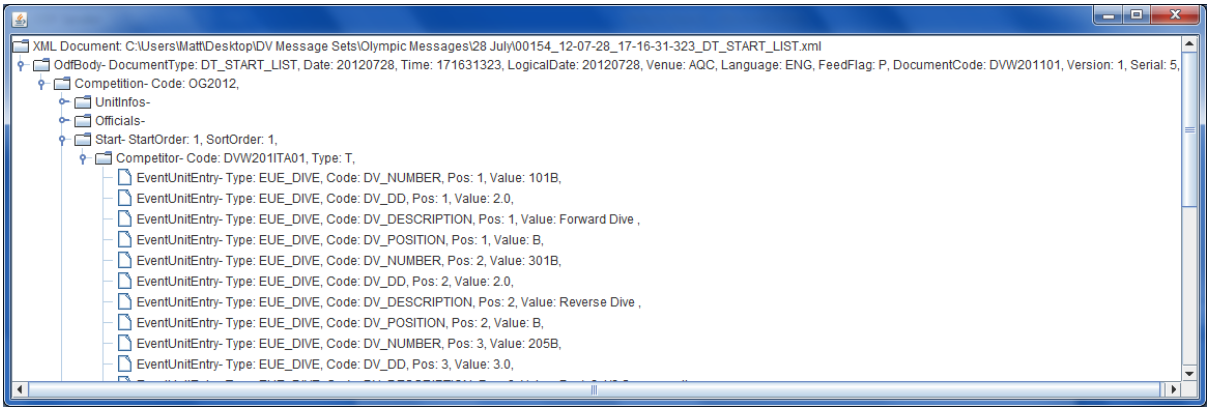
```
If DocType == DT_PARTIC_TEAMS
	Go to ODFBody element
	Go to Compeition element
	Go to Team element
	For all Teams
		Get code, organization and name
		Go to composition
		Go to athlete
		Get athlete 1 and athlete 2
		Insert data into teams table
	End For
End If
```

If the message is DT_PARTIC_TEAMS, we go to the team element and loop through all sibling nodes. We then get the team name, organisation and code, and then step down the tree to retrieve the athlete codes. This is then inserted in to the database.

Graphical version of the tree structure created from DT_PARTIC_TEAMS

```
        If DocType == DT_START_LIST
                Go to UnitInfos Element
                For all sibling nodes
                        If Element name == Start
                                Get StartOrder
                                Go to Competitor Element
                                Get Team Code
                                Insert Start Order and Team Code into Results Table
                                Go to EventUnitEntry
                                For all eventunitentry
                                        Get Difficulty, Code, Number and Desc. for each
dive
                                        Insert Dive Information into Dives Table
                                End For
                        End If
                End For
        End If
```

If the message is a start list, we go to the UnitInfos element which has siblings which are Start elements. We loop through all the sibling nodes, and if it is a Start element then we get the start order. Next we go to the competitor element to get the team code- these 2 pieces of data can now be inserted in the results table (and we add the event code from the DocCode we got earlier). After this we get the dive information by going to EventUnitEntry and looping through all the sibling nodes, then inserting the data into the database.

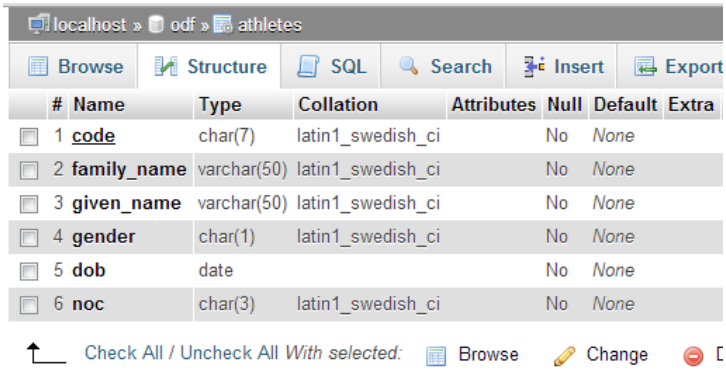


Graphical version of the tree structure created from DT_START_LIST

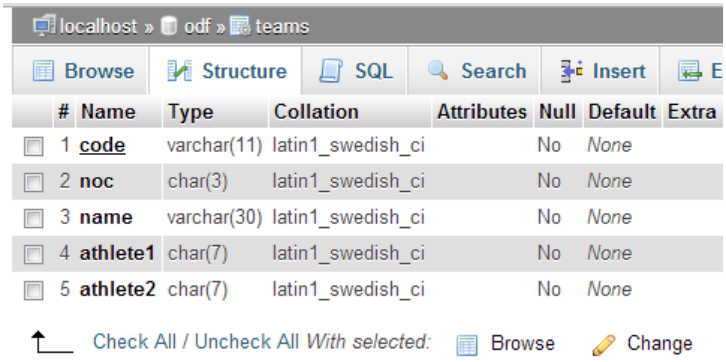
Database

In order to ease the management of my database, I installed PhpMyAdmin on the server, a free and open source tool written in PHP, intended to handle the administration of MySQL databases through a browser (available at phpmyadmin.net/home_page/index.php). The following screenshots show the partial implementation of the tables I defined earlier. These are the tables and fields needed for the prototype and will be added to later to create the full application.

Athletes



Teams



Results

localhost » odf » results

Browse | Structure | SQL | Search | Insert | Expo

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	event	varchar(9)	latin1_swedish_ci		No	None	
2	competitor	varchar(11)	latin1_swedish_ci		No	None	
3	start_order	int(2)			No	None	

Check All / Uncheck All *With selected:* Browse Change

Dives

localhost » odf » dives

Browse | Structure | SQL | Search | Insert | Export

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	competitor	varchar(11)	latin1_swedish_ci		No	None	
2	dive_number	int(1)			No	None	
3	dive_code	char(5)	latin1_swedish_ci		No	None	
4	description	text	latin1_swedish_ci		Yes	NULL	
5	difficulty	char(3)	latin1_swedish_ci		No	None	
6	E1	char(4)	latin1_swedish_ci		Yes	NULL	
7	E2	char(4)	latin1_swedish_ci		Yes	NULL	
8	E3	char(4)	latin1_swedish_ci		Yes	NULL	
9	E4	char(4)	latin1_swedish_ci		Yes	NULL	
10	E5	char(4)	latin1_swedish_ci		Yes	NULL	
11	E6	char(4)	latin1_swedish_ci		Yes	NULL	
12	S1	char(4)	latin1_swedish_ci		Yes	NULL	
13	S2	char(4)	latin1_swedish_ci		Yes	NULL	
14	S3	char(4)	latin1_swedish_ci		Yes	NULL	
15	S4	char(4)	latin1_swedish_ci		Yes	NULL	
16	S5	char(4)	latin1_swedish_ci		Yes	NULL	
17	pen	char(4)	latin1_swedish_ci		Yes	NULL	
18	avg	char(4)	latin1_swedish_ci		Yes	NULL	
19	score	char(6)	latin1_swedish_ci		Yes	NULL	

Check All / Uncheck All *With selected:* Browse Change Dr

PHP Page

The following pseudo code describes the implementation of the PHP page to generate a start list with dives in a browser window. Full code listings can be found in Appendix E.

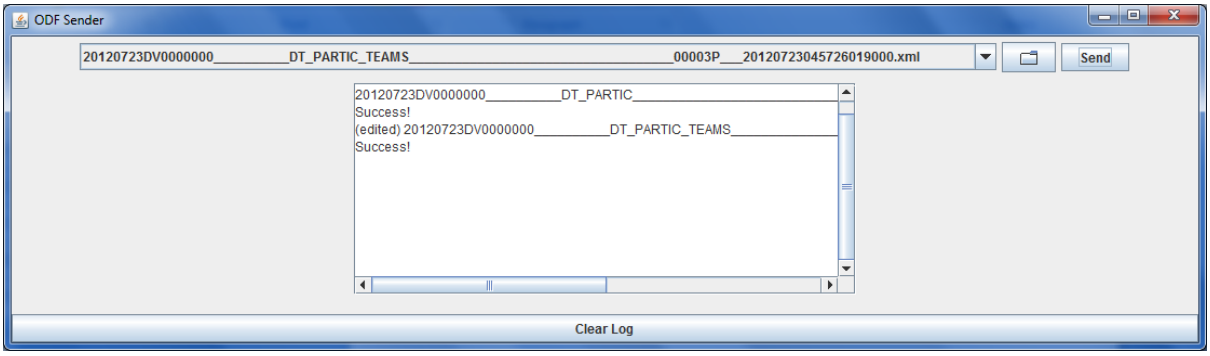
```
	Connect to database
	Select competitors from results where event = this event
	If rows returned == 0
		Print "no start list available"
	End If
	Else

		Select name, order, noc and athletes names from results where event =
this event
		While (rows returned)
			Print team information
			Select dive information from dives where competitor = this
competitor
		End while
			While (rows returned)
				Print dive information
			End while
	End else
```

The PHP page first connects to the database. We then select competitors from results which have the same event the page is for. If this returns 0 rows, then we know there is no start list sent yet. Otherwise we get the all team information and the athletes names for the event in question. We then use a while loop to go through every row of the tuples returned, echoing the data to the page. For each row, we then perform a query to return all of the dive information for that team, and echo this to screen as well.

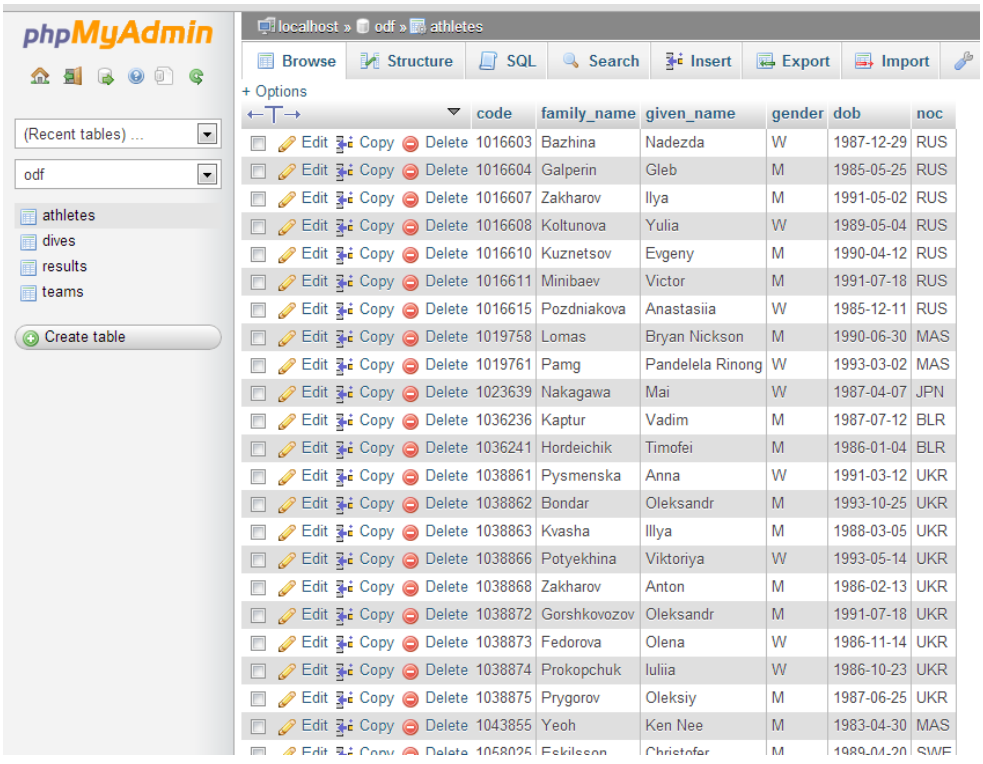
Results

The prototype is run and works correctly using the following steps:

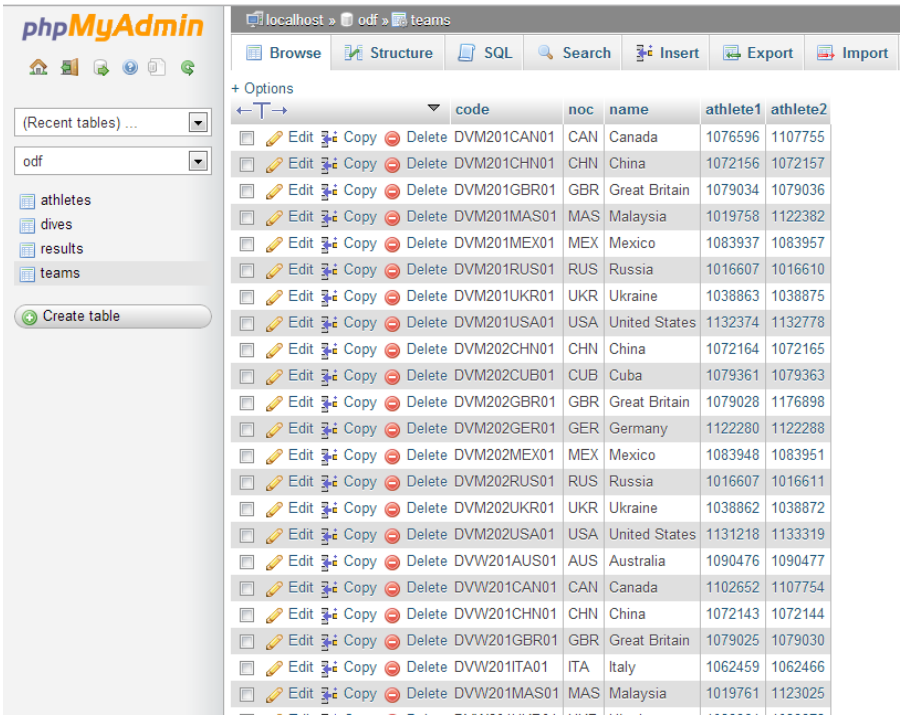
1. DT_PARTIC and DT_PARTIC_TEAMS are sent from ODF Sender



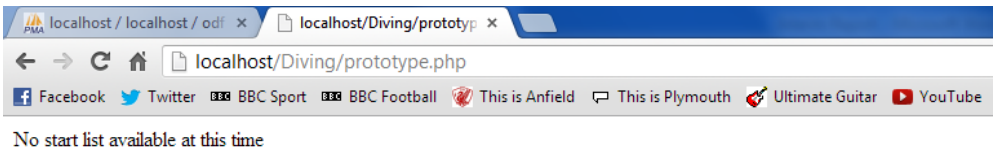
2. The data is populated in the database



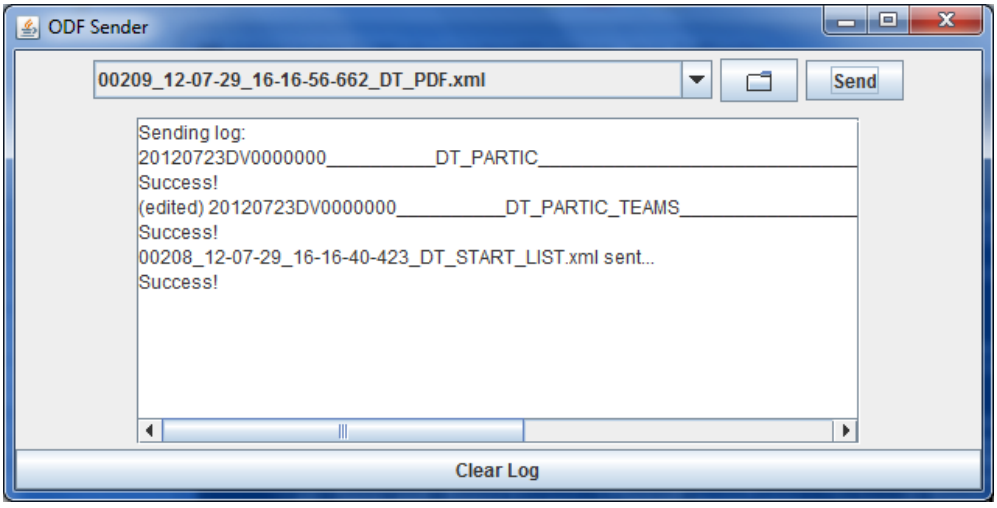
	code	family_name	given_name	gender	dob	noc
Edit Copy Delete	1016603	Bazhina	Nadezda	W	1987-12-29	RUS
Edit Copy Delete	1016604	Galperin	Gleb	M	1985-05-25	RUS
Edit Copy Delete	1016607	Zakharov	Ilya	M	1991-05-02	RUS
Edit Copy Delete	1016608	Koltunova	Yulia	W	1989-05-04	RUS
Edit Copy Delete	1016610	Kuznetsov	Evgeny	M	1990-04-12	RUS
Edit Copy Delete	1016611	Minibaev	Victor	M	1991-07-18	RUS
Edit Copy Delete	1016615	Pozdniakova	Anastasiia	W	1985-12-11	RUS
Edit Copy Delete	1019758	Lomas	Bryan Nickson	M	1990-06-30	MAS
Edit Copy Delete	1019761	Pamg	Pandelela Rinong	W	1993-03-02	MAS
Edit Copy Delete	1023639	Nakagawa	Mai	W	1987-04-07	JPN
Edit Copy Delete	1036236	Kaptur	Vadim	M	1987-07-12	BLR
Edit Copy Delete	1036241	Hordeichik	Timofei	M	1986-01-04	BLR
Edit Copy Delete	1038861	Pysmenska	Anna	W	1991-03-12	UKR
Edit Copy Delete	1038862	Bondar	Oleksandr	M	1993-10-25	UKR
Edit Copy Delete	1038863	Kvasha	Illya	M	1988-03-05	UKR
Edit Copy Delete	1038866	Potyekhina	Viktoriya	W	1993-05-14	UKR
Edit Copy Delete	1038868	Zakharov	Anton	M	1986-02-13	UKR
Edit Copy Delete	1038872	Gorshkovozov	Oleksandr	M	1991-07-18	UKR
Edit Copy Delete	1038873	Fedorova	Olena	W	1986-11-14	UKR
Edit Copy Delete	1038874	Prokopchuk	Iuliia	W	1986-10-23	UKR
Edit Copy Delete	1038875	Prygorov	Oleksiy	M	1987-06-25	UKR
Edit Copy Delete	1043855	Yeoh	Ken Nee	M	1983-04-30	MAS
Edit Copy Delete	1058025	Eskilsson	Christofer	M	1989-04-20	SWE



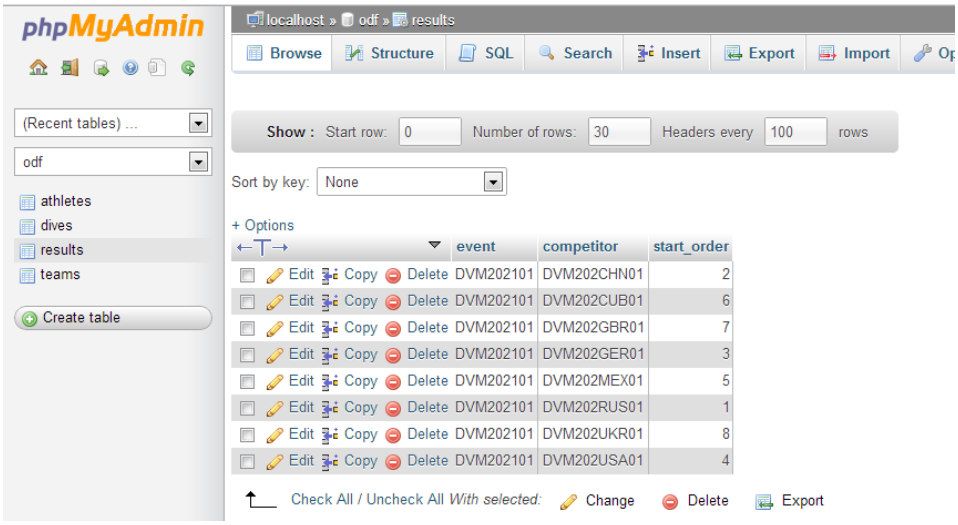
3. If we look at the PHP page, there is currently no start list

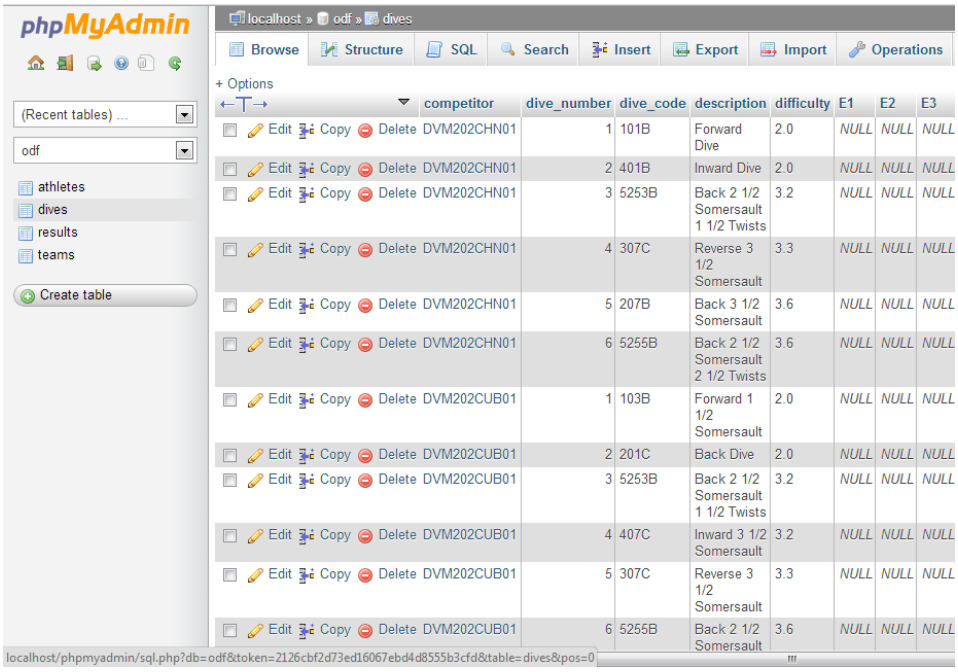


4. We send DT_START_LIST from ODF Sender

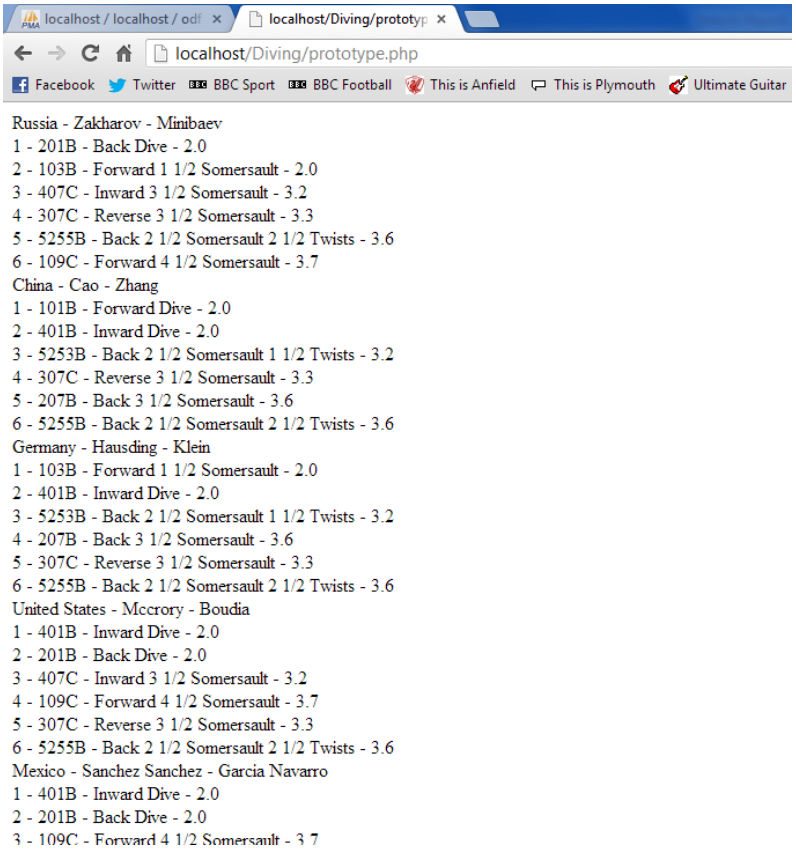


5. The data is populated in the database





6. The PHP Page now shows the start list and dive list for each team



Conclusions

Results

I believe that the project is currently on track to meet the aims deliverables I set out. So far I have managed to specify a basic design concept and the technologies I will use to achieve this, including specifying the intricacies of data flow through the system and a resulting user interface concept. After building a basic prototype based on this design, I believe that my design ideas have been justified as the prototype has been successful- meeting the aims I set for it and proving that the design is a feasible one. I think this prototype has provided a strong platform to build the full application from as I continue the design and implementation processes. However, it is important to acknowledge that my design and prototype are so far very basic, and represent a very small portion of the overall system. There is still a lot of more detailed design work to complete, and the final implementation will be far more complex than what I have achieved so far. I could easily run into a lot of problems in the future and have to make changes to the project, however, I feel that my work so far has definitely proven that the core functionality of my system is viable and I am confident this can be continued and adapted into a full system to meet my aims.

Next Steps

Now that the prototype has confirmed the viability of my design, the project can move forward into implementing the full system. This will involve iteratively developing the results application functionality, along with its graphical user interface and incorporating user prediction. Although this implementation will now make up the majority of the project, it is vital that I recognise the importance of design. Up to this point, I feel that the extra time spent on background and design is

largely what has lead to a successful prototype. I need to make sure that I continue to design the more detailed parts of the implementation, rather than going straight into building as I believe that good design will lead to a much more successful outcome. So immediately, the next step is to begin designing the more detailed elements of the application, such as detailed layouts, user interface, how the sure prediction will work etc. I will also need to complete implementing the database and message processing before I begin implementing the main application as the main application cannot function properly without complete data.

Project Approach Evaluation

In terms of my approach to managing the project, I feel that my initial approach was naive. Because of my experience at the Olympics with the real systems, I think I initially acted as if I knew exactly what to do for the project, and how to go about it so did not pay a lot of attention to how I would approach it. This was evidently a poor move and I soon changed my mind as I began designing. It became quickly apparent that I was allowing my past experience to drive the project rather than stopping and thinking about it myself. Although this was an early mistake, I feel I did well to rectify this quickly and change my approach. Once I had taken time to step back and think about the aims and the scope of the project I recognised the flaw in my approach, and I believe changing this has so far been successful. I now feel that I have a strong plan for the next steps of the project and have a much better whole picture of how the project will progress.

I would've liked to have added more features and implementation to my prototype, however, the scope I could set for it was greatly affected by the amount of time available. Because I needed to complete my basic design and background research before beginning the prototype, there was not a great amount of time left to complete it for this report. It would have been nice to include a small part of user interaction and a basic graphical user interface, however, I don't think this could have been achieved for this report. However, I do feel that building the basic prototype was successful in determining the feasibility of my design by showing that the core functionality will suffice to meet my aims.

Changes in Approach

As I mentioned above, I would've liked to have included more features in the prototype but did not have the necessary time to do so, so I am now going to re-assess my project plan again to ensure I have made the greatest use of the time available in the coming months.

Originally I planned not to continue working into the Christmas break, however, I feel it may be necessary to use some of this period towards this project. As I mentioned previously, I think it is extremely important to continue design work throughout the project, therefore I feel that I should allocate more time to detailed design. However, this is hard to fit into the project plan, so I think making more time available is the best solution. I've also noticed that at no point have I planned to get any user feedback. Considering a large bulk of the application is user-orineted, I am going to add a user test and heuristic evaluation into the development lifecycle to make sure my designs demonstrate good human computer interaction. These additions can be seen in the latest version of the project plan in Appendix F.

Bibliography

Atos, n.d. *Information Diffusion Systems.* [Online]
Available at: http://uk.atos.net/en-uk/olympic_games/what_we_deliver/information_diffusion_systems/default.htm
[Accessed 4 December 2012].

Balfour, A., 2012. *London 2012 Olympic and Parlympic Games-Time Digital Report.* [Online]
Available at: http://www.slideshare.net/balf/london-2012com-olympic-games-digital-round-up-13-august-2012
[Accessed 4 December 2012].

Deltatre, 2012. *Deltatre.* [Online]
Available at: www.deltatre.com
[Accessed 6 December 2012].

International Olympic Committee, 2012. *Diving - Sync. - Women - 3m - London 2012 Olympic Games.* [Online]
Available at: http://www.youtube.com/watch?v=_lmT4WlK7G0
[Accessed 28 October 2012].

International Olympic Committee, 2012. *Olympic Data Feed.* [Online]
Available at: http://odf.olympictech.org/
[Accessed 26 October 2012].

LOCOG, 2012. *Olympic Diving.* [Online]
Available at: http://www.london2012.com/diving/
[Accessed 2 December 2012].

LOCOG, 2012. *Olympic Diving - Information, History, Rules.* [Online]
Available at: http://www.london2012.com/diving/about/
[Accessed 4 December 2012].

Long Tail Video, 2012. *JW-Player Overview.* [Online]
Available at: http://www.longtailvideo.com/jw-player/
[Accessed 28 November 2012].

McLaughlin, B., 2001. *Java & XML.* 2nd ed. Sebastopol: O' Reilly.

Megginson, D., 2004. *SAX.* [Online]
Available at: http://www.saxproject.org/
[Accessed 12 November 2012].

phpMyAdmin, 2012. *phpMyAdmin.* [Online]
Available at: http://www.phpmyadmin.net/home_page/index.php
[Accessed 28 November 2012].

Premier League, 2012. *Fantasy Premier League.* [Online]
Available at: http://fantasy.premierleague.com/
[Accessed 3 December 2012].

Swiss Timing, 2012. *Swiss Timing.* [Online]
Available at: www.swisstiming.com
[Accessed 7 December 2012].