Ticketing System for Liberty Stadium

401312 - Mathew Collins

May 15, 2011
Abstract

The aim of the project was to create a reliable and easy to use ticketing system for use by Swansea City FC and Ospreys Rugby Union at the Liberty Stadium. The major goals of the project were to: a) Create a system which makes it easier for customers to purchase the tickets that they want. b) To create a more efficient and modern system for providing entry to the ground on a match day, which will reduce costs on a match-by-match basis. c) To provide a fairer system for distributing tickets for away matches. This dissertation discusses the design and implementation of the project and what has been achieved.
Contents

1 Introduction 3

2 Background information 5
  2.1 Overview of the system currently in use . . . . . . . . . . . . . 5
  2.2 Drawbacks to current system and ideas to improve . . . . . . . 6
  2.3 Technical Challenges - Interface Design, GUI, Database Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.4 Examples of other good systems . . . . . . . . . . . . . . . . . 10

3 System specification and design 12
  3.1 Software used . . . . . . . . . . . . . . . . . . . . . . . . . . 12

4 System implementation 15
  4.1 The liberty stadium ticket office server . . . . . . . . . . . . . 15
    4.1.1 Server SOAP web service . . . . . . . . . . . . . . . . . 15
    4.1.2 Database . . . . . . . . . . . . . . . . . . . . . . . . . 16
  4.2 Ticket office client . . . . . . . . . . . . . . . . . . . . . . . . 23
    4.2.1 Home Screen . . . . . . . . . . . . . . . . . . . . . . . . 23
    4.2.2 Seating Plan . . . . . . . . . . . . . . . . . . . . . . . . 25
    4.2.3 Block Plan . . . . . . . . . . . . . . . . . . . . . . . . . 27
    4.2.4 Away tickets . . . . . . . . . . . . . . . . . . . . . . . . 31
    4.2.5 Basket . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
    4.2.6 Administration Panel . . . . . . . . . . . . . . . . . . . 33
  4.3 Website . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
    4.3.1 Log in screen and storing state . . . . . . . . . . . . . . 36
    4.3.2 Matches Attended . . . . . . . . . . . . . . . . . . . . . . 38
    4.3.3 Edit Details . . . . . . . . . . . . . . . . . . . . . . . . . 38
  4.4 Away ticket allocation . . . . . . . . . . . . . . . . . . . . . . . 40
5 Security and testing

5.1 Security
5.1.1 Authentication
5.1.2 SQL Injection Attack
5.1.3 Buffer Overflow Attacks

5.2 Testing
5.2.1 White Box Testing
5.2.2 Black Box Testing

6 Using the system

6.1 Installation
6.2 User guide

7 Conclusion

7.1 Evaluation
7.2 Possible changes and improvements
7.3 Further Work
7.3.1 Easier access to the stadium
7.3.2 The website
7.4 Improvements by the company
7.5 Acknowledgements
Chapter 1

Introduction

Figure 1.1: The Liberty Stadium.

As a season ticket holder at Swansea City football club, and a frequent visitor to the Liberty Stadium for rugby matches, I have often dealt with the ticket office there, and also purchased tickets for matches in a variety of ways, including online, over the phone and in person. I believe that there are significant improvements that could be made to the current system which would benefit the stadium management company, saving them time and money, as
well as the customers, the fans, who would find it easier to purchase and use their tickets, leading to an altogether more enjoyable match day experience, and, hopefully, more frequent visits to the stadium.

The Liberty Stadium (Figure 1) was opened in 2005, and is home to Swansea City Football Club and Ospreys (formerly Neath-Swansea Ospreys) Rugby Union. The stadium is owned partially by the two clubs, and partially by Swansea Council, and together the three form the Stadium Management Company (known as Stadco), which, as the name suggests, is responsible for the day-to-day management and running of the stadium.

The system can also be considered to be a critical one, as the failure of it would result in significant financial loss for the company, and, in the case of the turnstile software failing, potentially thousands of angry fans stuck outside the ground, many of whom may decide not to return - against this would have implications for the company, and probably the software developers too. Therefore, the system must be created using well established principles of software engineering that guarantee a high degree of reliability.

The main aims of the project are to create a reliable and easy to use system which will simplify the purchasing of tickets. A more efficient and modern entry system to the ground, which will reduce costs on a match-by-match basis. And to provide a fairer system for distributing away tickets.
Chapter 2

Background information

2.1 Overview of the system currently in use

There are two ways for customers to attend matches at the Liberty Stadium - either by buying a season ticket, which entitles them to attend every match for the next nine months, or by purchasing tickets on a match-by-match basis. Season tickets are usually bought in the summer, however they are available to buy throughout the season, with a discount offered for the matches that have already taken place.

Match tickets are available to purchase online, via telephone or in person. Ticket sales for the two teams differ significantly, with 40% of tickets for the Ospreys being bought online, compared to only 3% for Swansea City. Telephone sales account for around 10% for each club, with the remained bought from the ticket office in person, either in advance or on the day of the game.

Season ticket holders are given a voucher booklet and card, which must be brought to every game. The card is shown to the turnstile operator, but kept by its owner, and the appropriate voucher is handed over in order to gain entry.

In addition to selling tickets for upcoming home matches, the ticket office also handles distribution of tickets for matches which are to be playing by either club away from home. Season ticket holders are given priority when purchasing away tickets, however for many matches demand outstrips supply, and additional conditions must be met in order to be eligible for a ticket - usually this is determined by how many previous away matches have been
attended, so a season ticket holder who has travelled to three away matches will be given priority over someone who has been to only one. When purchasing tickets for these matches, away ticket stubs must be produced at the ticket office, which are then punch-holed and given back to the customer.

The current system is provided for the company ProVenue, who have a base program and modify it according to the needs of the individual clubs. The main ticket office software is provided, while additional software, such as online facilities, are provided at an additional cost, as well as a premium for each ticket purchased. TicketMaster also provide a similar Windows-based service to some football clubs, including Arsenal and Manchester United.

2.2 Drawbacks to current system and ideas to improve

The current online ticketing system is poor, with no provision to choose where to sit, no ground seating plan (one is available on the official website, however it doesn’t show any seat numbers, and is effectively useless). There is also a processing fee for each ticket purchased, and no option to buy online and collect at the game - tickets must be posted out at extra cost to the customer. In addition to this, many important items are ambiguous or badly worded, as can be seen from Figure 2. For example, if you were purchasing children’s tickets for the corner of the east stand, you would select ”West COR” and then ”INETU16”, which give very little indication as to what they actually mean (presumably INETU16 stands for ”in Education, Training or Under 16”, however the meaning is clearly not obvious).

The telephone system is a premium rate number, and whenever I have tried to use it, it seems to be rarely answered. From speaking to other fans, this appears to be the case for many other people too. By improving the online system and providing automatic ticketing booths (figure 4) in various locations around the city, the telephone system could stay as it is.

Buying tickets in person is by far the easiest way to purchase tickets, however many fans live outside of the area, or cannot get down to the stadium to purchase tickets in advance due to work. This results in large queues at the ticket office on a match day. With improved ways of buying elsewhere, hopefully a larger number of people would buy in advance, easing the congestion. There are many issues regarding the sale of away tickets, especially
for Swans matches, and it’s something that is regularly brought up during meetings of the supporters trust. As stated above, ticket stubs from previous away matches are required for some more high-profile matches. This is far from an ideal situation, for a number of reasons. Firstly it requires all fans who would be interested in a ticket to keep track of their tickets for other matches, many fans who have been to several away games might lose their ticket stubs, or perhaps were not given one to begin with. In addition to this, fans with surplus stubs, or who do not wish to travel, can give theirs to
friends who do not have any, and would not otherwise be eligible.

A system whereby every away ticket sold is allocated to a fan in the database would eliminate these two problems, and provide additional benefits. If, for example, there are 2,500 tickets for a certain away game, and the ticket office suspects that demand is more than that, a limit on previous away matches attended, or perhaps through a points system, could be set. Currently, other than guesswork, there is no way for the ticket office staff to know what number of previous away games would be suitable, as they keep no proper records of tickets sold and to whom. Under the proposed system, the program could be queried as to the correct number, and could say that, for example, there are 2,350 fans who have attended at least five away games. These fans could then be contacted - automatically via email if desirable - to let them know that they are eligible for a ticket, and asking them to respond by a certain date. When that date passes the people with the next amount of away matches attended could be contacted, and so forth until all tickets have been sold.
One of the major challenges involved in this project is designing the stadium seating plan which would be used in the online ticket sales and in the ticket booths. There is currently a stadium plan available online (Figure 4), but it does not show individual seats or rows, so this would have to be done from scratch unless one could be obtained from Stadco. Also if this program is to be reusable it will have to be possible for other stadium plans to be created somehow, and it is also possible that the current seating plan could be altered in future, due to expansion, a change in the area designated for away fans and segregation, or other circumstances. Flexibility is desirable but may not be possible due to the specific nature of the task.

Another major technical challenge is to ensure that the different components can communicate effectively and efficiently. The website will rely on the main program to provide it with details of which seats are available for which match. Likewise the system running the turnstiles will need to know which tickets to let into the ground - again this information will be provided...
by the master program. Communication failure would be a serious problem for the company, especially on a match day as potentially thousands of people would be stuck outside the ground with no way of gaining access.

Testing the program for correctness, efficiency and robustness will also provide a challenge. If some errant data is entered somewhere then the system needs to be able to cope with it without crashing. Ensuring that the program is correct is also vital, as it will hold thousands of records of customers, and for the same reason efficiency is important.

2.4 Examples of other good systems

Most major sports clubs in the UK now offer online facilities for purchasing match tickets. Clubs such as Manchester United (Figure 5), Arsenal and Sheffield United have their own systems which offer the customer far greater choice in terms of their purchase, including allowing them to choose the exact seats that they want, and a much slicker user interface. However many clubs, including Chelsea, use the same system that is used by the Swans and Ospreys, so there is certainly a possibility for addition use with other clubs. The Manchester United website also allows users to see the view from their chosen seat when purchasing tickets.

In addition to superior online facilities, the majority of stadia built in the last ten years now operate using a barcode system with their tickets, requiring only that tickets are swiped under a barcode reader to gain entry to the ground, thus eliminating the need for turnstile operators, as well as allowed season ticket holders to use a credit card like card for entry, rather than having a whole voucher booklet. Some clubs take this even further, with all food and drink sales done via prepaid cards, as well as issuing non-season ticket holding home fans with cards which are used in the same way as season tickets, where the supporter will purchase their ticket in the usual way, but will no receive a ticket, but use their supporters card to gain entry to the ground. In addition to the above points, the country’s top clubs, such as Manchester United and Chelsea, use a point system to allocate away tickets. Frequent supporters away from home are rewarded with a number of points for each match attended, the number of points depending on the status of the fixture - a midweek trip to Newcastle (a round trip of around 600 miles) would earn substantially more than travelling to Tottenham (10 miles away) on a Saturday for Chelsea fans. Future away tickets are then
Figure 2.4: Manchester United’s online ticketing system allows users to select which block of seating they want to sit in, and provides an image displaying what the view is like from that block. [?] offered to fans with the most points first, ensuring that regular travellers are rewarded. Regarding the ticket machines that could be placed around the city, many institutions, such as cinemas, train stations and airports already make use of machines where the customer enters their details and then a ticket is printed. This idea would work in exactly the same way.
Chapter 3
System specification and design

3.1 Software used

As previously stated, the system can be considered a critical system, which makes Java an undesirable option for implementing it. However, as it is only a financially critical system and not a safety critical one, I using Java as the basis of the main program. Java is my strongest programming language, and I felt at the beginning of the project that I would be better able to produce a working and function system with Java, rather than having to learn a new programming language first, which would have taken up time that could have been used for implementation. I do understand that in the real world a system of this sort would potentially be more suited to C++ or a similar language, however in a real world system there would likely have been a team of developers working on it, rather than just myself, and they would likely have been working on it full-time, while I have obviously had other aspects of my degree to complete a the same time.

The server is written in Java and deployed as an XML web service.

The main application is backed up by an MYSQL database, which I am more familiar with than Oracle, and is easier to integrate with Java in my experience. It can also be integrated with C-sharp, but an additional plugin was required to make that work.

I had originally intended to use php as the basis of the web interface, with perhaps a flash, html5 or similar application to host the seating plan. The reason I wanted to use html5 was to make the site accessible to users of ipads, ihpones, etc. and to make it usable with the next generation of
similar devices. I also considered using an embedded java applet, but eventually decided on C# and asp to create the website. Using asp combines the features of a website with that of an application, where the C# code handles the behind the scenes interaction with the database. I have some limited experience of using this method before and found it easy to use, understand and very practical. In particular some of the tools offered by asp, such as gridview and detailsview, are an excellent way of displaying information on a website without too much effort on the part of the programmer. As the website mostly displays data, with only one section for editing, there was little point in combining the website with the web service - rather it was created as a standalone application that can connect directly to the database.
Figure 3.1: A diagram showing the relationship between the various parts of the system and the end users.
Chapter 4

System implementation

4.1 The liberty stadium ticket office server

4.1.1 Server SOAP web service

The server runs on a glassfish server and accesses a MySQL database running on wampserver. These are currently running on the same machine, and although it is possible for them to run on separate ones, this is likely to be the case in the real world. The server connects to the database using the Java mysql connector from the standard library, and uses a class, DMBS.java, which was used in our second year group project, and provides methods for connecting to, accessing, getting and sending queries to the database. This class was written by Paul John Evans, and I am using it as it provides all functionality that I require, and I saw no reason to write a new class to perform exactly the same functions when this one was already available, with Javadoc documentation. I did make some small changes to the class, particularly the disconnect method, which failed to close the connection to the database properly. This wasn’t a problem in a standalone application, but due to the nature of web services, where the services is accessed periodically by the client, creating a new connection to the database each time, it’s imperative that the client closes the connections, otherwise the database becomes overloaded and refuses to accept any more, rendering the program useless.

Using Web Services also means that the client could be written in C-sharp, or another language that differs from the server, as web services are deployed as extensible mark-up language (XML) solutions, which can then be interpreted by other languages which also support web services.
The class *LibertyServer.java* contains a list of methods which can be called by the Client, and which return some form of data to it. These in turn call equivalent methods in *DBManager.java*, which performs the actual work by connecting to the database using an instance of the aforementioned *DMBS.java*.

A list of some of the methods provided by LibertyServer.java is provided below.

- List getAllCustomers()
- Customer getCustomerByID(int customerID)
- Customer getCustomerByName(String forenmae, String surname)
- List getAllMatches()
- Match getMatchByID(int matchID)
- Match getSeatsForMatch(int matchID)
- boolean updateCustomerInfo(int customerID, Customer customer)
- boolean addNewCustomer(Customer customer)
- boolean addNewMatch(Match match)

Every current method provided by the server is of this type, and any of them that involve writing to the database are synchronised in their critical blocks, to prevent incorrect information being returned by another method, and more importantly, to prevent multiple calls to update the same information at the same time. The write methods all return a boolean which indicates whether or not the write was successful.

### 4.1.2 Database

One of the first priorities while designing the system was to create a suitable database structure, as the rest of the systems depend entirely on it. The database is a MySQL 3.1 database running on wampserver, and can be accessed only by the server and website - the client applications must go through the server to access the information in the database. Wampserver also comes with a phpmyadmin installation, which provides a web interface
for interacting with the database, thus the end user doesn’t require a vast knowledge of SQL to use it. It is possible that the ticket office management could use phpmyadmin to keep track of the database and create backups of it periodically. The tables and fields in the database are outlined below, along with a description where the meaning of the field may not be obvious.

The first three tables define the structure of the stadium and how the seats and blocks are distributed throughout the stands. These are essentially static and won’t change unless there is reconstruction or expansion of some kind at the stadium. Although they can be modified through the database, there is no programatic method for them to be changed as the structure is intended to be static.

### Stands

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>StandName</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>int</td>
<td>How many rows are in the block</td>
</tr>
</tbody>
</table>

### Blocks

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlockID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>StandID</td>
<td>int</td>
<td>Stands.StandID</td>
</tr>
<tr>
<td>BlockNumber</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>int</td>
<td>Categories.CategoryID</td>
</tr>
</tbody>
</table>

### Seats

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeatID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>BlockID</td>
<td>int</td>
<td>Blocks.BlockID</td>
</tr>
<tr>
<td>SeatNumber</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td>int</td>
<td>0 if not actual seat, otherwise 1</td>
</tr>
</tbody>
</table>

### Categories

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CategoryID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>CategoryName</td>
<td>varchar</td>
<td></td>
</tr>
</tbody>
</table>
Seats are included in the database even if there is no actual seat present, this is in order to maintain the structure of the blocks for the seating plan. Seats typically don’t exist in the corners of the stadium where there are more seats in the higher rows than towards the bottom, and also where space has been left for entrances, exits and camera positions. There are over 20,000 entries in the seats table, so listing them all would take up far too much room and is not really necessary anyway.

The seats each have a unique seat identification number, as well as referencing which block the seat belongs to. The blocks work in a similar way, with a reference to the stand they belong in. To find out which stand a seat is in, we must use a query covering all three tables, however the alternative of also storing the stand ID number in the seats table is undesirable, as this would take the table out of third normal form, and present potential consistency problems when the table is updated.

Each block is also allocated a category from the categories table, which represents the cost of tickets for seats in that particular block. The three categories are hard coded, premium seats, on the side of the pitch near the half way line, are the most expensive, then the rest of the tickets on the side are category 2, while the seats behind the goal are in category 3. Prices in these blocks change depending on the category of the match, and that is defined elsewhere.

The next set of tables relate to the matches, opponents and the different type of matches that are held at the ground.
### Opponents

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpponentID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Name</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Distance</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>GroundName</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>int</td>
<td>MatchTypes.TypeID</td>
</tr>
</tbody>
</table>

### Matches

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>OpponentID</td>
<td>int</td>
<td>Opponents.OpponentID</td>
</tr>
<tr>
<td>MatchCategory</td>
<td>int</td>
<td>Categories.CategoryID</td>
</tr>
<tr>
<td>Date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>int</td>
<td>MatchTypes.TypeID</td>
</tr>
<tr>
<td>HomeGame</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>TVGame</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>CupGame</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>Allocation</td>
<td>int</td>
<td>Away only</td>
</tr>
<tr>
<td>Threshold</td>
<td>int</td>
<td>Away only</td>
</tr>
</tbody>
</table>

### MatchTypes

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Name</td>
<td>varchar</td>
<td></td>
</tr>
</tbody>
</table>

### MatchCategories

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CategoryNumber</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>CategoryBlock</td>
<td>int</td>
<td>Categories.CategoryID</td>
</tr>
<tr>
<td>Price</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

Each team and match is associated with a certain match type. The types table by default contains only Swansea City F.C. and Ospreys Rugby, however if in future more teams were to play the ground, they could be added to the MatchTypes table. They would then be displayed alongside the original two entries throughout the program. The allocation and threshold columns of the matches table are only used for away matches and will be ignored if the game is a home one. Allocation holds the amount of tickets that have been given for the game, while threshold is the currently points total needed for a customer to purchase a ticket for the game.
The match categories table holds information about the price of tickets for each category, based on the match type. There are three hard coded categories, 1, 2 and 3, and each holds details for the price for each block type. So for example, for a match classed as category 1, the price for a premium ticket (block category 1) is £30, while for a standard side ticket (block category 2) it is £25.

The next set of tables deals with customers and their information, then the tickets that they have purchased.

### Customers
- **CustomerID** int Primary Key
- **Surname** varchar
- **Forename** varchar
- **Middlename** varchar
- **HouseNumber** int
- **Street** varchar
- **Area** varchar
- **City** varchar
- **PostCode** varchar
- **Email** varchar
- **Telehpone** int
- **DOB** date

### SeasonTickets
- **Season** int Primary Key
- **SeatID** int Seats.SeatID Primary Key
- **CustomerID** int Customers.CustomerID
- **MatchType** int MatchTypes.TypeID Primary Key

### TicketsSold
- **SeatID** int Seats.SeatID Primary Key
- **CustomerID** int Customers.CustomerID
- **MatchID** int Matches.MatchID Primary Key

The season tickets and tickets sold tables are similar, but hold slightly different types of information. The primary keys are defined in a way that
allows each seat to only be sold one per match (in the case of tickets sold) and once per season per match type (for season tickets) - the same seat can be sold once for a Swansea City season ticket, and once again for an Ospreys match. There is nothing in the database stopping tickets from being purchased for a seat that is already held by a season ticket holder, however this is done programatically, where seats that are taken by season ticket holders are not available for purchase. A possible weakness with this is that if a match were changed in the database to a different type, there is no protection against seats being sold in both tables. This however is a minor concern as it cannot be done programatically, and there is no reason to do it, as the sports are entirely separate and no team is an opponent to both the Swans and the Ospreys.

The staff table stores details of each member of staff and the passwords associated with their accounts, as well as their access level - 1 is a staff member, 2 is an administrator.

### Staff

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaffID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Username</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td>int</td>
<td>Defines access level of the staff member</td>
</tr>
</tbody>
</table>

### Database Normalisation

One of the most important aspects of database design is proper normalisation. Database normalisation is the process of converting or ensuring that a database meets certain relational requirements, and is necessary in order to avoid what are known as update anomalies, as well as preserving the integrity of the data, avoid repetition and increase the ease of expanding the database. There are many different normal forms, each providing a greater degree of normalisation, starting with first normal form, and ending with seventh normal form.

The higher formal forms, above third normal form, are generally not needed for small to medium sized databases, of which mine is one. They deal explicitly with tables that have multiple candidate primary keys, while each table in my database is pretty unambiguous in terms of what to choose as a
primary key. For those reasons, the database has been (mostly) normalised into third normal form (with one exception, more on that later), and I have mostly ignored the normal forms above third normal form.

“A relvar [relational variable] is in third normal form (3NF) if and only if, for all time, each tuple [table] consists of a primary key value that identifies some entity, together with a set of zero of more mutually independent attribute values that describe that entity in some way.”

So, looking at the matches table as an example, we can see that this table does indeed meet all of the criteria outlined above.

Normalisation - Addresses

In a theoretical sense, an address normalised to third normal form would comprise one field only, however in a practical sense this is very difficult to achieve. Firstly, containing the whole address as one value presents issues with searching, while storing just the post code and a house number may be slightly better, it still requires us to have another table which holds a list of all possible postcodes and their corresponding street names and locations (in fact, since multiple post codes often refer to the same street, an additional streets table might also be necessary). Obviously for a database which will see a large amount of local addresses, a few from further afield in the UK, and only a small amount internationally, there seems to be very little point in going to such lengths, especially as the postcodes table would require constant updating.

Storing the whole address for each customer is therefore probably the most practical approach, and what I have decided on doing in this case. While from a database point of view one or more fields could be updated independently of each other, creating a possible update anomaly, this is not going to be possible from the software, which is where the vast majority of the updating is going to be coming from. For small-to-medium sized databases, some tables that are not fully normalised are generally going to be OK, as long as they are properly maintained.
4.2 Ticket office client

The ticket office client is a Java Application which is designed to run on the computers of the ticket office, and to be used by the ticket office staff when customers come to buy tickets in person, or over the phone. It is comprised of several major parts which are detailed below.

4.2.1 Home Screen

The home screen, which can be seen on the next page in figure 4.4, is presented to the user when they start the client application. The client requires that the server is running when it loads up - if the server is not running, an error message will appear, and the client will not open. Before the staff member can do anything, they are required to log on, which they can do by using the log in box at the top left of the screen. Once logged on, the user is presented with a list of upcoming matches. Below that is the match information panel, which gives summary information, such as date and tickets sold, for a selected match. When a match is selected, the information will be displayed below, and the user can access further details either by pressing the view match details button, or by double clicking on it. At the bottom right of the screen is the basket, which will always be empty upon login. If the user is an administrator, they will also see the administration panel button.

The list of upcoming matches can also be filtered by type using the checkboxes above the panel. It has been designed in such a way that if additional types were added to the database in future, these would also appear here automatically.
Figure 4.1: The seating plan I created.
4.2.2 Seating Plan

One of the most challenging aspects of the design for the project was the seating plan for home matches, which provides details for each seat in the ground, whether it has been sold, occupied by a season ticket holder, designated for away fans or allocated for segregation.

Each seat that can be sold is represented by a simple JLabel, coloured according to its status for the match in question. Additionally, each seat belongs to a particular block in the stadium itself, and this format is also represented in the program, with blocks being simple JPanels that just hold the JLabel elements that represent the seats.

As there are 18,750 seats in the stadium (the additional 1,770 are part of the executive boxes, hospitality packages, or are disabled seats, and are dealt with separately) there’s obviously going to some questions regarding performance if we wish to display them all at once. Not only would we have to have to display nearly 20,000 icons, we’d need to access the database to determine the status of each seat for a particular match. This could be done with only a small amount of queries, but would still require either if-then-else statements or a case statement to determine the colour of the seat.

Alternative approaches include using threads to display all of the seats. This, however, I discounted due to the issues arising from using threads with java swing - which is used to display the GUI. Swing is not considered to be thread safe, according to the Java API,

“Once a Swing component has been realised, all code that might affect or depend on the state of that component should be executed in the event dispatching thread.”

citejava

Further to the above, threads can also be difficult to use and make programs harder to debug, since the flow of execution may be different each time it is run when the operating system decides the order of thread execution.
Figure 4.2: The seating plan I created.
There are a few exceptions to the rule, but in general using threads with swing is not considered to be good practice by Java developers. Although it would be possible to use threads to construct the seat elements before we display them, we are likely going to be limited to a four core machine, which means that we could divide the seats up into four equal parts of around 5,000. This is possible, however there is still a slight delay in displaying the seats in this case. On a computer with only two cores, we would only see a 2x increase in performance.

Eventually I decided against displaying all the seats at once, and settled on a screen which shows the blocks only, colour coded in relation to the seats in the blocks. Red denotes no availability, orange limited availability (more than 67% of tickets sold), yellow medium availability (between 33% and 67% sold) and green denotes plentiful availability (less than 33% sold). Blocks marked in grey are not available for sale through the ticket office. These have either been allocated to the away team (in football matches only), or are the middle blocks of the West Stand upper tier, where the blocks are reserved for the directors, premier club members (long-term debenture holders), and hospitality guests, which are all handled separately. Blocks in black are being used for segregation and are also unavailable.

When a block is selected, the seating plan for that particular block now pops up, this has several advantages over the method of displaying all of the seats at once. In addition to the performance issues already outlined, it also means that the seats can be represented by larger images than would be the case in the full-stadium plan, facilitating the ease of purchasing tickets.

Creating the seating plan

The seating plan is represented by two JPanel’s, one for the upper tier and one for the lower tier. Both panels use a GridBagLayout, and the blocks are added to the appropriate layout in order after being retrieved from the database. Originally they were sized according to the number of rows in each block, however I thought that having each block the same sized looked far better. The eight upper tier blocks that are on the corners of their stands also merge into the blocks on the other stand, and although it would have been possible to show them in that way, I decided against it as it didn’t look particularly good, and you couldn’t tell that there were actually two blocks there instead of one. Customers will still know that the block is in the corner, facing the pitch diagonally, as they will be able to view a picture of the view.
from the block if selected.

### 4.2.3 Block Plan

The block plan displays the seats for a particular block, as well as an image displaying an approximate view of the ground taken from that particular block, or one nearby. Each seat is represented by a JPanel, and is colour coded depending on its status for that particular game chosen. Green represents seats that are yet to be sold for this match, red seats are those which are taken by season ticket holders, and seats coloured in blue have been purchased already for that particular game. Some areas of the seating plan have been greyed out, and these are the seats which are set to invalid in the database, as they are not in fact actual seats.

When a staff member wishes to purchase some tickets on behalf of a customer, there are two ways to do it: they can either select individual seats by clicking on them, or click and drag across a row to select several seats. Selecting seats will change their colour to cyan in the block view. Seats that are occupied or invalid cannot be selected, and it is also not possible to click and drag for seats at an arbitrary location in the block, this can only be done for seats on the same row. You can, however, select seats on the different rows, but it must be done separately.

Once the seats have been selected they can be added to the shopping basket by clicking on the add button, which will ask the staff member to confirm their reservation, along with a price total for the seats. These seats are now set to reserved in the database (CustomerID=0) in the tickets sold table. The staff member now has ten minutes in which to purchase them before they are set back to being available again. This is done to ensure that while the customer is purchasing additional tickets, their original seats are not sold in the meantime. Ten minutes seemed like more than enough time to select additional tickets for more matches.

**Creating the block plan**

The seating plan for each block was meticulously reproduced from the seating plan that is available on the Swansea City website, showing only block numbers and amount of seats in each block. Although there was no way of knowing exactly which seat numbers are not in use, the plan is still very accurate, with the correct seats set to invalid. This was fairly straightforward
Figure 4.3: One of 76 block plans, recreated from the seating plan on the Swansea City website.
for the blocks along the side of the pitch and behind the goal, as the only
invalid seats here are where there are entrances, exits and camera positions.

The difficulty came in the corners, where the blocks do not have the same
amount of seats at the top and bottom of their rows, so care had to taken
in order to ensure that the correct seats were set to invalid. The picture
in figure (4.2) shows one example of such a block, with the rows getting
narrower towards the bottom.

Details

The process of displaying the seats in the correct order is fairly straightfor-
ward. A list of seat numbers belonging to the block in question is returned
from the server, as well as the number of rows the block contains. From this
information a grid layout is created with the amount of rows in the block, and
JPanels representing each seat are them added to the display. To complicate
matters slightly, the seats in the four stands are obviously all facing towards
the pitch, and as such are represented on screen facing the pitch, too. The
position of the seat on the x and y axes can be determined fairly simply:

```java
int block_width = total seats / height of block;

// East Stand
x = seat number % block_width;
y = block_width - (seat number / block_width);

// West Stand
x = block_width - (seat number % block_width);
y = seat number / block_width;

// South Stand
x = seat number / block_width;
y = seat number % block_width;

// North Stand
x = (block_width) - seat number / block_width;
y = (block_width) - seat number % block_width;
```

After being colour coded and added to the block display, the seat panels
have a mouse listener added to them. A mouse listener is a java inter-
face which provides methods for dealing with mouse events - mouse entered, mouse exited, mouse released, mouse pressed and mouse clicked. For the block plan I’ve used the mouse entered and mouse exited events to keep track of where the cursor is on the screen. We’re only using this information in conjunction with the mouse pressed and mouse released buttons. When the mouse pressed event is triggered, meaning that the mouse’s left button has been pushed down, but not released, if the event occurs on a valid seat, that seat is added to a list of currently selected seats. Then, using the mouse entered and mouse exited events, any additional seats that the mouse enters before being released are also added to the list. At the end, we need to check that all seats that have been selected are valid, and this is done in two ways:

a) The row number of each seat is compared to the row number of each additional seat - if these all match, the selection is accepted, otherwise a warning dialog is displayed telling the user that only seats from the same row may be selected at the same time.

b) Each seat in the set is now checked to see whether it is a valid seat - that is, it is actually a seat, it is not held by a season ticket holder, and it has not been previously purchased.

c) Seats that have already been selected and added to the basket may also be selected again, this is allowed and the seats are simply not added to the basket if they are already in it.

If the selection matches the above criteria the seats are then added to a temporary basket. The staff member can now elect to add additional seats from the block to the basket, or purchase the ones that have been selected. Selected seats are highlighted in cyan on screen, and the dialog also informs the staff member, and therefore the customer, of their purchase amount.

4.2.4 Away tickets
Figure 4.4: The seating plan I created.
4.2.5 Basket

When a user has selected all of the seats and tickets that they wish to purchase, the staff member can click on the "go to basket" button in the bottom left hand corner of the screen, as shown in figure 4.5. This will take them to the basket view, where they’re presented with a list of all tickets that have been reserved (figure 4.6). The basket view panel is inside a scroll pane, in order to handle large orders, and is comprised of several objects of type TicketDetails, each of which represents one ticket.

![Basket](image)

Figure 4.5:

Existing Customer
Lookup Customer
Add New Customer

4.2.6 Administration Panel

Staff Editor
Match Editor
Opponent Editor

4.3 Website

The original plan was for the website to follow a similar plan to the ticket office client, where customers could themselves go, log on, and purchase tickets using the same seating plan as in the client. Due to time constraints I was unable to do this, but did still create a simple website where customers could go and check their past and future purchases, as well as update their contact information. While not doing everything I had originally envisaged,
and despite being very simple aesthetically, the website does do a job and displays the ticketing information quite clearly.

The website was created as an asp web application, backed by a C#
program which handled the interaction with the database. I also considered
4.3.1 Log in screen and storing state

When loading the website, customers are required to log in using their unique customer identification number and password before they can gain access to the other pages. The other two pages, customer.aspx and details.aspx have methods which run on page load, redirecting the user to the log in screen if they have not already logged in. As http connections are stateless, and we need to keep track of the current state of the user, a session variable is used to keep track of the customer’s ID number once they have logged in. If there is no activity from the user, their session will expire after 20 minutes, and they will be logged out.

Session variables were just one of several options that could have been used here, but I believe it to be the best one for this purpose. Cookies are
 Swansea City FC / Ospreys Rugby Ticket Information

Enter your details below to log in:

Customer ID Number: 1
Password: Collinx

Invalid username & password combination.
frequently misused in this way, but come with some limitations. They are stored on the end user’s computer, where they can be modified or deleted, and users can also turn cookies off in their browser settings - obviously not desirable for tracking log in details. Another practical client-side option is the viewstate variable, but again this can be modified by a malicious end user to give them potentially unauthorised access.

Session variables are stored on the server, and thus cannot be accessed by the end user, this increases security, which is always of paramount concern in any system which stores potentially sensitive information. The downside of using session variables comes with scaling the system. Currently the website, database and server application are all run from the same PC, but if the system were implemented in reality, it could be distributed over several computers in order to ease the bandwidth burden, in which case session variables tie the end user to a particular sever, and if the server were to go down, they would be forced to log back in again.

### 4.3.2 Matches Attended

Once logged-in the customer is redirected to a page showing details of their purchases. At the top of the screen there are buttons to log out and to edit their details, a brief information panel displaying the customer name, and their current points total, and below them the information is displayed. There are three components here - a list of season tickets purchased, then home tickets, and finally away tickets. Each section is displayed using an asp gridview component, and gives the customer basic information about their purchases, such as seat number and stand (if applicable), the date of the game and opponents name and the type of match being attended (i.e. Swansea City or Ospreys). The gridview components are backed by SQL data source components, which do the actual work of retrieving the information from the database. The SQL queries are fairly complex because they each take in details from several tables, but the only variable is the customer ID number, which is stored in the session variable.

### 4.3.3 Edit Details

The final section of the website is a page where customers can keep their information up to date. The page follows the format of the matches attended page with buttons and information at the top of the screen, and then another
Figure 4.13:

gridview displaying the customer’s personal information. If they wish to edit it, they can click on the edit link, which makes the fields editable. Keeping their information up to date is important for future communication from the

<table>
<thead>
<tr>
<th>season</th>
<th>typeName</th>
<th>seatNumber</th>
<th>standName</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Ospreys RU</td>
<td>N080</td>
<td>West Upper</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>typeName</th>
<th>Name</th>
<th>seatNumber</th>
<th>standName</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/05/2011 00:00:00</td>
<td>Swansea City FC</td>
<td>Nottingham Forest</td>
<td>R137</td>
<td>East Upper</td>
</tr>
<tr>
<td>26/05/2011 00:00:00</td>
<td>Ospreys RU</td>
<td>London Irish</td>
<td>G082</td>
<td>East Lower</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>typeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/05/2011 00:00:00</td>
<td>Northampton Town</td>
<td>Swansea City FC</td>
</tr>
<tr>
<td>01/09/2011 00:00:00</td>
<td>London Irish</td>
<td>Ospreys RU</td>
</tr>
<tr>
<td>05/12/2011 00:00:00</td>
<td>Doncaster Rovers</td>
<td>Swansea City FC</td>
</tr>
</tbody>
</table>
club, particularly if the customers want tickets to be posted out to them, but also for information on upcoming matches and when tickets are going on sale, as these can be communicated via email.

### 4.4 Away ticket allocation

Another of the my major objectives when starting this project was to create a fairer system of allocation of tickets for away matches. Currently for matches with sufficient demand, typically at least a few times a season, season ticket holders have to bring sufficient ticket stubs from previous away games to the ticket office, which are then punch-holed and given back to them. There is no record of who has purchased tickets, the process is time-consuming.
(not available online or over the telephone, for example) and can be easily circumvented by people borrowing ticket stubs from their friends. In order to address some of these problems, I have decided on a points-based system for away matches, similar to one that many clubs, particularly at the top level in the UK and on the continent already employ.

Ultimately I settled on a fairly simple system where away matches attended over the course of the past twelve months are taken into consideration. Tickets for future matches are not considered to be part of the total, matches only become valid after they have been played. I did initially intend to use something more complex, where recentness was also a factor, however in the end I deemed it an unnecessary addition since the one I’m going to use (detailed below) should still give a good approximation of loyalty. The formula for deciding on the amount of points allocated for each match is necessarily different between the two clubs, as they play in vastly different competitions - at least a few Ospreys matches are played in France and Italy each season, as well as Ireland, while at present Swansea City’s matches are all domestic, although there is potential for European competition sometime in the future.

When a new away game is added to the database by the ticket office manager, the allocation of tickets available is also stated. The software then returns the customers with the highest number of points, up to the amount of the allocation, and these customers can be notified by email within the program, or sent a letter via normal post. Usually they will be given a certain amount of time before they are offered to another group of customers with slightly fewer, and so on, provided there are any tickets still available.

I also considered the possibility of introducing away season tickets. Similar to the season tickets for home matches, they would entitle supporters to tickets for every away game, and these tickets would be posted out to them as soon as they are received by the ticket office. This is a system that works well with some of the bigger clubs in the country, but when I asked the ticket office staff about it, they claimed that there was insufficient demand for it. Given that some away attendances for the Swans are often only a few hundred, particularly for midweek games, and that the new system will make it easier for regulars to purchase tickets anyway, I agree that it’s probably not something that needs to be implemented at the moment, although it could be a useful addition in future.

A list of opponents is kept in the database, along with the distance away the team is. The list can be maintained by the administrator in the case of
new opponents being played, or if a team happens to move to a new location which would require a different points total for the mileage.

4.4.1 Swansea City FC

The formula for allocation of points to a match is as follows:

- 1pt Base per match
- +1pt For every 50 miles away
- +2pt If the match is played in midweek
- +2pt If the match is on live television
- +2pt If the match is a major cup game (except Premiership clubs)
- +4pt If the match is a minor cup game
- +5pt If the match is a European game

Additionally, the ticket office manager is able to add or remove bonus points for each match prior to it being played, possibly to award bonus points as an incentive for more people to travel, or to decrease the points awarded for a hugely important game where demand is going to be very high in any case.

Applied to the matches this season, the following scores can be obtained for each match:
<table>
<thead>
<tr>
<th>Date</th>
<th>Opponent</th>
<th>Base</th>
<th>Dis.</th>
<th>MW</th>
<th>Cup</th>
<th>TV</th>
<th>Tot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug 7th</td>
<td>Hull City</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Aug 21st</td>
<td>Norwich City</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Aug 24th</td>
<td>Tranmere Rovers</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Sep 11th</td>
<td>Leeds United</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Sep 21st</td>
<td>Peterborough Utd</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Sep 25th</td>
<td>Nott’m Forest</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Sep 28th</td>
<td>Watford</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Oct 16th</td>
<td>Reading</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Oct 26th</td>
<td>Wigan Athletic</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Oct 30th</td>
<td>Crystal Palace</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Nov 7th</td>
<td>Cardiff City</td>
<td>1</td>
<td>0</td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Nov 20th</td>
<td>Doncaster Rovers</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Dec 4th</td>
<td>Ipswich Town</td>
<td>1</td>
<td>5</td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Dec 18th</td>
<td>Sheffield Utd</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Dec 26th</td>
<td>Queens Park R</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Jan 3rd</td>
<td>Leicester City</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Jan 22nd</td>
<td>Barnsley</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Feb 1st</td>
<td>Bristol City</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Feb 13th</td>
<td>Middlesbrough</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Feb 22nd</td>
<td>Coventry City</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Mar 5th</td>
<td>Scunthorpe Utd</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Mar 12th</td>
<td>Derby County</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Apr 2nd</td>
<td>Preston NE</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Apr 16th</td>
<td>Burnley</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Apr 23rd</td>
<td>Portsmouth</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Apr 30th</td>
<td>Millwall</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>May 12th</td>
<td>Nott’m Forest</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

The highest points total for a single match in this case is 9, given for the league cup game at Peterborough in September, while long trips to Norwich and Middlesbrough also score highly. The final away game, a playoff semifinal at Nottingham Forest, is worth nine points due to it being a midweek game live on television, however this total could be reduced slightly by the ticket office administration, as it was a high-demand match and tickets were sold out, even though they were only on sale for four days before the game, and didn’t reach general sale until May 11th, the day before.
4.4.2 Ospreys Rugby Union

As previously mentioned, computing a fair system for the Ospreys was more challenging than for Swansea City, as they regularly compete with teams from Italy in the Mangers league, and teams from France in the Heineken Cup, as well as Scottish, Irish and English teams throughout the season. Thus while distance is certainly a factor, it is more difficult to measure. Even though Dublin, the home of Irish side Leinster, is only 152 miles from Swansea, fans must either fly there or take a ferry, while London, around 180 miles away and the home of several English clubs, is much more straightforward to get to, either by car, train or coach.

In my opinion, fans that travel to Dublin should be awarded more points than those who travel to London, so the points system is slightly different here, using additional base points per country, rather than relying on distance. This is far from a foolproof method, but should ensure that fans are rewarded more for spending more time, money and effort on supporting the team. The Heineken Cup and Magners League are both major competitions and, as attendances are traditionally higher for the European cup matches, there are no extra points awarded for those games. There is a slightly different formula for computing the total for Welsh teams, and non-Welsh teams.

**Welsh teams**
- 1pt For the Welsh teams
- +1pt If the match is played in midweek
- +1pt If the match is on live television
- +2pt If the match is an LV Cup game

**Non-Welsh teams**
- 2pts For the English teams
- 3pts For the Scottish and Irish teams
- 4pts For the French teams
- 5pts For the Italian teams
- +2pt If the match is played in midweek
- +2pt If the match is on live television
- +2pt If the match is an LV Cup game

Applied to the matches this season, the following scores can be obtained for each match. It was difficult to find a definitive list of televised matches for the season, but as the list is only being used to demonstrate the formula used, there’s no particular need for 100% accuracy.
<table>
<thead>
<tr>
<th>Date</th>
<th>Opponent</th>
<th>Base</th>
<th>MW</th>
<th>TV</th>
<th>LV</th>
<th>Tot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 3rd</td>
<td>Ulster</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Sep 18th</td>
<td>Munster</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Oct 2nd</td>
<td>Llanelli</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Oct 9th</td>
<td>Toulon</td>
<td>4</td>
<td>2</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Oct 24th</td>
<td>Glasgow</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Nov 6th</td>
<td>Newcastle</td>
<td>2</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Nov 19th</td>
<td>Connacht</td>
<td>3</td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Dec 12th</td>
<td>Munster</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Dec 31st</td>
<td>Cardiff</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Jan 7th</td>
<td>Leinster</td>
<td>3</td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Jan 16th</td>
<td>London Irish</td>
<td>2</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Jan 29th</td>
<td>Cardiff</td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Feb 19th</td>
<td>Treviso</td>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Mar 26th</td>
<td>Edinburgh</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Apr 15th</td>
<td>Newport</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>May 6th</td>
<td>Aironi</td>
<td>5</td>
<td>2</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>May 14th</td>
<td>Munster</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Overall, I’m happy with the scheme, although I did change it several times with regards to the televised matches, particularly those in Wales, where attending a televised midweek game is still only going to require a drive of around an hour (to Newport) at most. By giving two additional points for televised and midweek matches these could be worth five points - more than travelling 350 miles to Newcastle for an LV Cup game. In the end, I opted to give only one additional point for televised and midweek matches against the other Welsh regions, giving a fairer reflection of the difficulty and cost of attending.

### 4.5 Integration

One of the major challenges encountered so far was how to pass complex objects between the server and client using Jax-WS. Because Web Services are language independent, their output is able to be read by any other language, which means that an output of strings or integers is straightforward, as are lists and arrays, as there are all general constructs that are used by most languages.

Originally I was returning simple types only; Strings and Lists (Jax-
WS automatically transforms Java.util.ArrayList into a language-independent list), then constructing the Customer (etc.) object on the client-side. After consulting with a textbook, Java Web Services: Up and Running[1], I discovered that it is possible to pass complex types using Web Services, using the -wsgen and -wsimport tools. Unfortunately this method is unavailable with the NetBeans IDE, and not using NetBeans causes a plethora of additional problems in implementing the Client side of the architecture.

With the use of the Jax-B framework I was able to create XML objects on the web service and unmarshall them into java classes on the client side. However my attempts to integrate this with my application were less successful, and despite devoting numerous hours in an attempt to get it to work, I was unsuccessful. I also looked at using spring web services, and alternative that offers similar functionality to Jax-B, but in the end decided against it.

Now only simple types and lists can be returned by the client, and this is not really a major problem as it does not affect the functionality of the program in any way - it just means a few additional calls to the server and returning results in object form rather than in list form. In future I will make sure I get Jax-B working before starting work on the rest of the project!
Chapter 5

Security and testing

5.1 Security

The nature of Web Services is that they are published at a location on the internet via a glassfish, tomcat or equivalent server, and then anyone can access them and implement a client that uses the services provided. Obviously this is undesirable for a service which has access to confidential information, so the server will have to run on a machine that is protected by a firewall. The firewall will allow access only to machines on the same network as the server, and then only through port 8080 (the port used by glassfish). As the website will also run on the same system, there will be no access issues for it, while anyone else wishing to access the server from outside will be unable to do so.

5.1.1 Authentication

Passwords are used for both staff members, in order for them to use the client software, as well as customers, who can use it to login on the website. Storing passwords in the database in plaintext is obviously insecure, as a hacker who gains access to the database would then be able to see everyone’s passwords, perhaps without staff even knowing about the intrusion. Ideally therefore, passwords would be encrypted by the client software and transferred to the server, which would not decrypt them, rather it would store the hashed version of the password in the database. That way, the hashes may be found by an attacker, but they would have no way of knowing the actual passwords, and therefore no way of logging on to the system. Any malicious changes
made could be detected and reversed, usually by restoring the database from a backup, which should be taken regularly.

Due to time constraints, the passwords are stored as plaintext in the database for this project - it is sufficient for the functionality to work, and I recognise that in a real-world application, this does not represent good practice for the reasons outlined above.

5.1.2 SQL Injection Attack

The most common form of attack on any application which uses a database is the SQL injection attack. Although the firewall should prevent any outside interference with the server, it’s still possibly that the attack could take place from inside the client, either accidentally or maliciously. SQL injection attacks exploit a vulnerability in encoding SQL statements as a succession of string concatenations. For example, the following SQL statement looks perfectly harmless:

```java
String tString = jTextArea.getText();
String SQL = "SELECT * from someTable WHERE password=" + tString + ";
```

However, what happens if the user enters the following?

' OR '1' = '1'

The SQL query now reads:

```sql
SELECT * from someTable WHERE password=' OR '1'='1';
```

Since 1 is always equal to 1, this query will always evaluate to true, and will return all results from the table, leading to the possibility of a huge security leak. Likewise if a user enters ' as their string, the rest of the line will be commented out by the SQL software and therefore ignored. Used in conjunction with an UPDATE or DELETE statement, this vulnerability exposes the potential for massive data loss, corruption or theft. SQL injection can also occur accidentally, if a staff member were to add a customer with the surname O’Halloran, for example, the MySQL database would interoperable the ' as the end of a string, and return an error as it doesn’t know what a ‘Halloran’ is. This can be countered by escaping the character with a backslash, but you cannot expect the staff to know or remember that whenever they do it.
Thankfully the solution is fairly straightforward - instead of using string concatenation, we can pass the arguments as additional SQL parameters, and the SQL software will now treat the parameter as a string only, meaning that it cannot be used to change the syntax of the query. A basic SQL parameter is shown below:

```java
String tString = jTextArea.getText();
PreparedStatement statement =
"SELECT * from someTable WHERE password=?;"
statement.setString(1,tString);
statement.execute();
```

The question mark symbol is a placeholder for a parameter, which is then added to the statement in the next line with the `setString(1,...)` method. This states that the first parameter for this statement is to be `tString`. Parameters can be of any type that is supported by SQL, and are used throughout the database connection code in order to avoid the SQL injection attack. I had to make some modifications to the `DBMS.java` class in order to achieve this, in particular modifying the `GetQuery` and `SetQuery` methods so that they accepted objects of type `PreparedStatement` rather than strings, as the previous class made no attempt to protect against SQL injection attacks.

The C# programming language uses a very similar technique. The parameters are known as annotations, are named rather than numbered, and come in the format `@annotationName`. The annotations form part of the query and work in exactly the same way as the parameters used in Java.

Annotations also prevent malicious code from being inserted through the website - SQL injection attacks are frequently used for the purpose of embedding cross-site scripting - and appropriately sized columns in the database truncate any data that is too large. Thus, Bob Laramee’s favourite trick of copying and pasting the source code from the BBC News website won’t result in a system failure.
5.1.3 Buffer Overflow Attacks

5.2 Testing

5.2.1 White Box Testing

5.2.2 Black Box Testing
Chapter 6

Using the system

In this section I have included a basic installation guide, and a short user guide for the system. Set-up requires the addition of several additional external services.

6.1 Installation

To install the Liberty Stadium ticketing system, you are required to download and install several additional applications, in order to provide all of the services available. The client program will not run without the server side being online, and the Liberty server will not be able to return any useful information without the database being installed - rendering the whole thing useless. The website can be installed separately and requires the ASP development server to run - a standard web server will not suffice as there is also C# backing code - but again, without the backing database the website will not allow anyone to log-in.

- Glassfish server version 3.x
- MySQL database version 3.x
- ASP development server

The glassfish server can be downloaded from http://glassfish.java.net and also comes bundled with some versions of the netbeans IDE. The client is configured to search for the server on localhost and port 8080, which is the default port used by glassfish and should not be used by any other application.
To deploy the server, go to http://localhost:4848 - the glassfish administration panel - and select applications and deploy. Use the dialog that pop up to select the LibertyStadiumServer.war file from the installation CD, and then click deploy. If the server has been deployed properly it will be now be shown on the list here, and you can access the wdsl file and the tester from here if you wish.

You will need an installation of a MySQL database. When developing the project, I used wampserver, which can be downloaded from http://www.wampserver.com/en/, and includes everything you need to run the database, including a console window and an installation of phpmyadmin, which can be used to manage the database. Wampserver also provides the apache framework that is needed for MySQL to run. An SQL file with the database structure and some test data has been provided, and is called LibertyDatabase.SQL. If you have installed wampserver, you can use the phpmyadmin interface, selecting the SQL command, and simply copy and paste the entirety of this file, which will set up the database. Alternatively you can also copy and paste the file into the MySQL console window. The default MySQL access of username: root and password: 123456 are used. If you wish to change these values, you can find them at the top of the DMBS.java file, located in the LibertyStadiumServer package. Note that if you do change the username or password, you will need to recompile and then redeploy the LibertyStadiumServer.war file for the changes to take effect.

For the website to run, you will require an installation of ASP development server, which comes bundled with any version Visual Studio. The website will almost certainly not work on a normal web server, as there is C# code involved, which handles the communication with the database and authentication.

6.2 User guide
Chapter 7

Conclusion

7.1 Evaluation

When I began the project, I think I seriously underestimated the amount of work that would have to be done in order to do everything that I'd wanted. A project of this size would usually be done by a team, and possibly without them doing any other work at the same time. As such, there were several things that I ultimately had to leave out, which I will cover in the next section. In the end I decided to concentrate on creating a working client application for the ticket office, ensuring that it was working correctly, giving it the majority of the features I had intended to in the first place, and making it look at least reasonably aesthetically pleasing.

I was extremely pleased with the final result, even though there was no time to get every little thing that I wanted implemented, what I did have worked extremely well, and I also received some very positive feedback at the project demonstration fair, particularly from the industrial visitors. Most seemed impressed about the extent and size of the system, and that it was seemingly bug-free, with one commenting that it was far more work that he usually saw from a third year project. I was also told at the beginning of the project that nobody expected me to produce a complete working system by the end of it, and while it may not be entirely complete, it offers most of the functionality that you would expect from such a system, and, hopefully, an easier way for the ticket office staff to buy tickets for customers.

I've also learned about using several new java packages and libraries, including mouse listeners, window listeners, focus listeners, more advanced
swing components, such as JTables and spinners, various layout managers, including the versatile GridBagLayout, and expanded on previous work involving web services and databases.

7.2 Possible changes and improvements

Within the framework of the project, there are several things that, if I had the opportunity to do again, might have turned out slightly differently. I would probably concentrate more on creating the website from the beginning, and then possibly making a client afterwards. One of the major objectives was to create an easier system for customers to purchase tickets, and while that is the case if they’re ordering in person or over the phone, and while I am very pleased with the client application, I didn’t create a website that offered the functionality that I had originally intended.

There are also some minor changes that I would like to make to the client. I would probably split the administration panel’s into three parts, rather than displaying all three at once. I think the application looks so much better with the image of the stadium as the background, rather than just having the greyness of these panels.

I would also create a class that extends JTextField to provide the functionality for the text areas I created. The fields where staff can enter details for a new customer, and those on the search bar, have focus listeners attached to them that trigged when the focus is lost, checks whether the box is empty or contains the default value, and then changes the colour of the text if necessary. Instead of having the parent component do the focus listening, I could have implemented them as a simple extension:

```java
public class FocusTextField extends JTextField, implements FocusListener {

    private String m_defaultText;

    public FocusTextField(String defaultText) {
        m_defaultText = defaultText;
        addFocusListener(this);
    }

    @Override
```
public void focusGained (FocusEvent e) {
    if (getText ().equals (m_defaultText)) {
        setText (""");
    }
    setForeground (Color.BLACK);
}

@Override
public void focusLost (FocusEvent e) {
    if (getText ().equals (m_defaultText)) {
        setForeground (Color.LIGHT_GRAY);
    }
    else if (getText ().equals ("")) {
        setForeground (Color.LIGHT_GRAY);
        setText (m_defaultText);
    }
}

The above class simply extends the JTextField class with an additional default value, then when focus is gained, it will empty the text box if the value is the default value. If focus is lost, it will change the colour to light grey if the text is the default text or if the field is blank. It will also change the text back to the default text if the field is blank. This would mean that I wouldn’t have to write any additional code each time I used this technique, which I did not do originally. I think I’m being overly critical here though, as when I first used the buttons I did not intend to reuse them. Although they didn’t provide any additional functionality, I really liked my changing text fields, as they avoided the need for labels and generally improved the overall appearance of the dialogs in which they were used. Given more time, or the opportunity to create the client again, I would certainly use them in other areas of the client, such as the log-in box, and in the administration panel.

7.3 Further Work

There are so many possibilities for further work on this project, many of which I had originally hoped to be able to implement. Particularly, although
I have implemented some of the points system for away matches, showing how many points the away match is worth, and how many points each customer has, as well as allowing for the current threshold of points required, time constraints mean that I was unable to build this into the system to allow only those with enough points to purchase tickets for these games - currently the default threshold is 0, meaning that anyone is able to purchase them.

7.3.1 Easier access to the stadium

Another one of my original ideas was to create an easier system for entry to the stadium, similar to other new sports venues across the United Kingdom and Europe. I have a very clear idea of how this could be done, which I have outlined here, but again, time constraints meant that I had to concentrate on some areas of my initial plan while others had to be sacrificed, and this was one that I decided to do without.

The idea is that instead of having a book of vouchers, one of which is handed in at the turnstile upon entry to the ground, season ticket holders would have a card only. These cards would come with a barcode, RFID tag or similar attached. A reader would be placed at each turnstile, and would then check with the central server if that person was allowed entry. If so, the turnstile would open, if not, they would be refused entry.

Likewise non-season ticket holders would each have a card, rather than tickets for individual matches. These would be very similar to the season ticket cards, except that they would not allow automatic entry to the ground, rather the customer would have to pre-purchase their tickets before the card would allow them entry. Making the system electronic also cuts down on the need for printing paper tickets, usually several thousand for each game, as well as postage costs. Non-season ticket holders could purchase their tickets through the website, and then not need to collect their tickets or have them posted out, again this would cut down on the costs involved for both the stadium management company and the customers themselves.

This system would certainly be a safety critical system, as allowing too many people to enter a section of the ground could have serious consequences. It would certainly lead to overcrowding, and in some cases could endanger safety as there could be a crush, or the section would be overfull in event of an evacuation. Each stand’s turnstiles would therefore necessarily require an accurate and synchronized counter to ensure that no more than the maximum amount of people are allowed into the ground.
The supporters’ cards and their related barcode/RFID tag could also be used in conjunction with the ticket office client - particularly on the basket screen where customer ID numbers have to be entered - the card(s) could just be scanned by a barcode reader or RFID reader.

### 7.3.2 The website

Another area for which I had an exciting vision at the start of the project was the website. As seen in the background section, the current website falls well below standard for a modern day sports venue which could be hosting top-flight European rugby and Premier League football in the near future. Only 3% of sales for Swansea City tickets came through the website the beginning of the 2010-2011 season, and this can be partly attributed to the poor state of the online system. The website would work in a very similar way to the client program used by the ticket office staff. There would obviously be some features that wouldn’t be present - there would be no administration panel, for example, and no mechanism to search through the customer database if you didn’t know your customer number (you wouldn’t have been able to log-on in any case). The seating plan would be present and customers would be able to choose their own seats, and there would also be an option for new customers to register before purchasing tickets.

When buying multiple tickets, the ticket office client allows the staff member to simply enter the customer numbers of several customers at the same time. This would only be possible online in two ways - either the customer purchasing the tickets must know the customer numbers and passwords of those he wishes to buy tickets for, or there could be an option in each customers settings to add ”trusted” customers, who are allowed to purchase tickets on their behalf. Since families and friends will naturally want to sit together, this is a much better option than forcing everyone to buy their tickets separately, and also provides a way for parents to monitor the accounts of their children.

The ”trusted customers” scheme could be created in the database with the addition of just one more table with two fields, both referencing Customers.CustomerID. The first column would indicate the number of the first customer, and the second would be the ID number of the customer they are trusting to buy tickets for them. The two fields together would provide the composite primary key.
7.4 Improvements by the company

Figure 7.1: The official ticketing website has been updated since my meeting with the ticket office manager.
At the beginning of the project, the website for purchasing tickets was not very informative or helpful. Since then, since my last meeting with the manager of the ticket office, the ticket sales website has been updated somewhat. Now instead of displaying confusing internal names for types of ticket sales, such as ”INETU16”, more sensible names, such as ”Student” and ”Senior Citizen” have been added. Likewise instead of ”WESUPP”, the actual names of the stands have been used.

The website also has a more professional look, and it appears that the charges for purchasing online have also been removed, which was something that I mentioned in my initial document that was contributing to the very small amount of tickets sold online, particularly for Swansea City. There is, however, still no seating plan, you are allocated a seat in the selected stand and there is no map to show you where the seat is.

It also appears that further improvements might be forthcoming. For the recent play off semi-final match against Nottingham Forest, season ticket holders were given first refusal on their seats, and the tickets could be reserved online. The system had a far more professional feel to it and seemed to work well, although there were initially issues with passwords, as the customer’s post code had been used, but was not always in the same format. The feedback from the ticket office suggested that around 60% of season ticket holders ordered their tickets for the game online, a significant increase on the 3% of customers that usually use it!

7.5 Acknowledgements