Managing desk space allocation and usage within Modern Office Environments

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF ENGINEERING in Computer Science

in

The Queen's University of Belfast

by

Saoirse McCann

9TH May 2017

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC3002 – COMPUTER SCIENCE PROJECT**

**Dissertation Cover Sheet**

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Saoirse McCann Student Number: 40104396

Project Title: Managing desk space allocation and usage within Modern Office Environments

Supervisor: Dr Barry McCollum

**Declaration of Academic Integrity**

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.**

*Student’s signature* *Date of submission*

Acknowledgements

First, to my supervisor, Dr Barry McCollum, thank you for all of the support and guidance you have provided throughout this project and for answering my endless questions.

To all the lecturers at Queens’ who have taught me over the past four years and to all of my teachers at Dominican College Fortwilliam, thank you for passing on your knowledge and for your enthusiasm for teaching.

To my family and friends, thank you for your patience, and for encouraging me to keep going and never give up.

Abstract

The objective is to implement a solution to aid a company in managing resource allocation within a modern office environment. The concept of agile working was one of the main motivations and influences for the end solution, allowing employees to choose how, when and where they work. The end solution provides an Executive Officer Dashboard displaying key performance indicators for the company as a whole, allowing the office space to be analysed and managed in order to maximise the spatial utilisation of the company as a whole.

Table of contents

[Acknowledgements iii](#_Toc482052394)

[Abstract iii](#_Toc482052395)

[Table of contents iv](#_Toc482052396)

[Table of Figures vii](#_Toc482052397)

[Chapter One 1](#_Toc482052398)

[Introduction and Problem Specification 1](#_Toc482052399)

[1.1 Introduction 1](#_Toc482052400)

[1.2 Background to the Problem 2](#_Toc482052401)

[1.2.1 Current trends 2](#_Toc482052402)

[1.2.2 The Problem 2](#_Toc482052403)

[1.2.3 Civica’s Approach 2](#_Toc482052404)

[1.3 Research for the Solution 3](#_Toc482052405)

[1.3.1 Agile working 3](#_Toc482052406)

[1.3.2 Pareto’s 80-20 Rule 4](#_Toc482052407)

[1.4 Analysing Space Utilisation 4](#_Toc482052408)

[1.4.1 Frequency rate 5](#_Toc482052409)

[1.4.2 Occupancy rate 6](#_Toc482052410)

[1.4.3 Utilisation rate 7](#_Toc482052411)

[Chapter 2 8](#_Toc482052412)

[System Requirements Specification 8](#_Toc482052413)

[2.1 SOLUTION OVERVIEW 8](#_Toc482052414)

[2.1.1 Introduction 8](#_Toc482052415)

[2.1.2 End users 8](#_Toc482052416)

[2.1.3 Executive Officer Dashboard 9](#_Toc482052417)

[2.1.4 Resource Allocation 9](#_Toc482052418)

[2.2 ASSUMPTIONS 9](#_Toc482052419)

[2.3 Functional Requirements 10](#_Toc482052420)

[2.4 Non-Functional Requirements 11](#_Toc482052421)

[Chapter 3 12](#_Toc482052422)

[Design 12](#_Toc482052423)

[3.1 SYSTEM ARCHITECTURE 12](#_Toc482052424)

[3.2 Presentation Layer 13](#_Toc482052425)

[3.3 Domain/Business Layer 13](#_Toc482052426)

[3.4 Data Layer 13](#_Toc482052427)

[3.5 USER INTERFACE 13](#_Toc482052428)

[3.5.1 Design Overview 14](#_Toc482052429)

[3.5.2 User Interface Design Guidelines 14](#_Toc482052430)

[3.5.3 Screen UI Mock-Ups 14](#_Toc482052431)

[Chapter 4 15](#_Toc482052432)

[Implementation and Testing 15](#_Toc482052433)

[4.1 IMPLEMENTATION 15](#_Toc482052434)

[4.1.1 Server 15](#_Toc482052435)

[4.1.2 Languages 16](#_Toc482052436)

[4.1.3 Libraries 16](#_Toc482052437)

[4.1.4 Coding Conventions 17](#_Toc482052438)

[4.1.5 Source Control 18](#_Toc482052439)

[4.1.6 Development 19](#_Toc482052440)

[DATABASE 19](#_Toc482052441)

[MODELS 21](#_Toc482052442)

[SERVICE 22](#_Toc482052443)

[CONTROLLERS 23](#_Toc482052444)

[VIEWS 24](#_Toc482052445)

[4.2 TESTING 25](#_Toc482052446)

[4.2.1 Testing Strategy 25](#_Toc482052447)

[4.2.3 Manual Testing 25](#_Toc482052448)

[4.2.2 Automated Testing 26](#_Toc482052449)

[Chapter 5 27](#_Toc482052450)

[System Evaluation 27](#_Toc482052451)

[5.1 System Evaluation Approach 27](#_Toc482052452)

[5.2 Evaluation of Functional Requirements 28](#_Toc482052453)

[5.3 Non-functional Requirements Evaluation 32](#_Toc482052454)

[5.3 Usability questionnaire 33](#_Toc482052455)

[Chapter 6 35](#_Toc482052456)

[Conclusion 35](#_Toc482052457)

[6.1 Strengths 35](#_Toc482052458)

[6.2 Weaknesses 35](#_Toc482052459)

[6.3 Conclusion 36](#_Toc482052460)

[6.4 Future Recommendations 37](#_Toc482052461)

[Appendices 40](#_Toc482052462)

[Appendix A – User Interface Design Guidelines 40](#_Toc482052463)

[Appendix B – Screen Mock-Ups 41](#_Toc482052464)

[Appendix C – Acceptance Tests 49](#_Toc482052465)

[Appendix D - Usability Questionnaire 53](#_Toc482052466)

[Appendix E – User Manual 58](#_Toc482052467)

Table of Figures

[Figure 1: Satisfaction in tech companies 3](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050719)

[Figure 2: Agile working in tech companies 3](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050720)

[Figure 3: Architecture overview 12](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050721)

[Figure 4: Architecture Design 13](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050722)

[Figure 5: Productivity Power Tool 17](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050723)

[Figure 6: GitHub Screenshot 18](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050724)

[Figure 7: Entity models 19](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050725)

[Figure 8: DBContext class 19](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050726)

[Figure 9: Migrations folder 19](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050727)

[Figure 10: Entity-Framework diagram for the end system 20](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050728)

[Figure 10: Entity-Framework diagram for the end system 20](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050729)

[Figure 11: Team entity model 21](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050730)

[Figure 13: Database tables 21](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050731)

[Figure 12: User entity model 21](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050732)

[Figure 14: View model 21](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050733)

[Figure 15: Service method to calculate resource frequency rate 22](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050734)

[Figure 16: Get method for weekly overview generating data for the dashboard 23](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050735)

[Figure 17: Bundle config class 24](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050736)

[Figure 18: Bundles being rendered in layout view 24](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050737)

[Figure 19: TDD approach 25](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050738)

[Figure 21: Execution unit tests 26](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050739)

[Figure 20: Unit test 26](file:///D:\gitlab\BMC001\Dissertation\Disssertation.docx#_Toc482050740)

Chapter One

Introduction and Problem Specification

* 1. Introduction

The objective of the project at hand is to develop a system to aid in the management of resources within a modern office environment. In order to do so, the first port of call was to analyse the trends of current layouts, practices and concepts adapted in today’s office workspaces.

Currently, companies face major overheads for office space, resources, licenses, etc. Many companies now have open-plan workspaces for their employees in order to allow the company to expand. The paradigm shift from private individual offices to open-plan workspaces was driven by financial advantages, the promise of increased productivity and the potential for better interaction between employees.

With an open-plan layout companies can make better use of space, allowing for more employees in one building and reducing the overheads of paying for extra space. In environments where teams have to interact with one another on a continual basis, then an open-plan layout could increase productivity and allows colleagues to approach one another without the need to arrange a meeting.

More importantly, however, is the gradual shift into the concept of agile working. Agile working allows employees to choose when, where and how they work. Companies are becoming more flexible, allowing employees perks such as the ability to work off-site and moving away from the structured 9 to 5 working day.

The problem area that will be discussed and evaluated is how best to allocate and manage the work space within a company.

* 1. Background to the Problem

1.2.1 Current trends

For the most part, the concept of desk ownership still applies to the majority of companies. This approach is straightforward for the company as they simply allocate one desk for every employee, and no further planning or analysis is required. Employees prefer to have their own personal desk space that they can take ownership of, personalise and become accustomed to. This approach, however, is inefficient for the company, as resources become underutilised with no mechanism to utilise a vacant desk as it still technically has an “*owner*”.

1.2.2 The Problem

With employees working off-site, on annual leave or off sick, a number of resources can be left vacant. If each employee has a personal desk, then these resources cannot be utilised when required as they are still technically “*occupied*” by said employee, even when the resource is not in use.

It is becoming more common for companies to move away from the normal resource allocation process of providing each employee with a desk. The more efficient agile approach is to allocate resources when requested, which ensures no resources are not utilised when they could be, and allows employees to request the type resource they require.

The problem at hand is how best to allocate and manage the work space within a company. A company’s workspace can be categorised into resources, such as desks, private offices, meeting rooms, lunch areas, etc.

* + 1. Civica’s Approach

Civica is a technological company with a workforce of over 3,500 employees. Their current approach to resource allocation is a flag scheme. They operate the same scheme across all of their offices worldwide. The scheme involves an employee placing a flag on their desk if it is available to be used as a hot desk. For example, if an employee only worked Monday through to Thursday, then every Friday they would place the flag on their desk indicating that it is available for anyone to utilise. The issue with this scheme is that many people feel a sense of ownership over “*their*” desk and would therefore prefer no one other than themselves occupy it.

* 1. Research for the Solution

1.3.1 Agile working

With advances in the reliability and functionality of remote access software it is becoming more common for employees to work off-site. Different industries require employees to be on customer sites or out of the office for meetings.

For this reason, it is becoming less common for all employees to be in the office every day. Hence, let us consider each employee of the company as a “visitor”, who has the option to use office space if required. If space is required then the most appropriate resource for the “visit” can simply be booked, e.g. a hot desk.

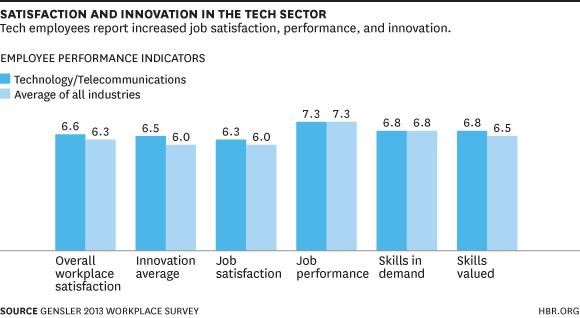
An article in the Hazard Business Review by Diane Hoskins states that overall performance and attitudes to the working environment improve when employees have free reign of the space where they work. The study stated this approach worked particularly well in technology companies.

Figure 1: Satisfaction in tech companies

The reason why these technology companies noticed an increase in productivity and the satisfaction of the workforce were due to the principles of agile working. Agile working gives employees a choice of where, when and how to work. A number of different space options are provided, such as meeting rooms, hot desks, team workspaces, etc. Agile working gives the workforce the freedom and flexibility to pick the best working environment for them and their task.

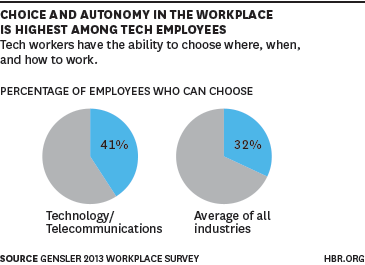
An agile workspace has more of a focus on the actual space and how it is used. Emphasise is put on business and employee requirements as opposed having departments which may be underutilised taking up a large number of resources. The concept of desk-ownership is therefore deprecated under an agile allocation system.

Figure 2: Agile working in tech companies

1.3.2 Pareto’s 80-20 Rule

The Pareto principle states that roughly 80% of outputs come from 20% of the inputs.

Applying Pareto’s rule to the issue of space allocation within an office would indicate that 80% of space is underutilised. In order to improve this ratio and to understand how best to manage it, a mechanism to analyse the space and how it is used is essential. Employing Pareto’s rule to space allocation suggests that there is no need to have space for 100% of a workforce. However, it is difficult to say the exact percentage of the workforce a company should provide space for. All companies have different working environments, practices and needs; they all have a workforce and all employees take time off work.

* 1. Analysing Space Utilisation

Space utilisation is a measure of how and how well space is being utilised.

Frequency, occupancy and utilisation rates are key performance indicators which measure how efficiently a space is being used. In this evaluation, space will be considered a resource. Space encompasses the following: desks, meeting rooms, lunch areas, etc.

Calculating the frequency, occupancy and the utilisation rates for a company can help in many ways. These measurements are vital when a company is seeking to expand. The indicators allow a clear insight into which resources are being underutilised, allowing the possibility of converting them into workspaces for new employees and other space options which are in higher demand. In turn, this will have fiscal benefits for the company allowing them to expand, while staying at the same location with no additional overhead for a larger office. Overall, if a company has a thorough analysis of their space utilisation, it provides endless possibilities for the company to improve and manage their workspace.

1.4.1 Frequency rate

The frequency rate represents the percentage of time space is used compared to its availability.

Example:

A company has a working day from 9am to 5pm. The table below shows when a resource was in use for one working week, Monday through to Friday.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY |
| 9am - 10am | **VACANT** | **UTILISED** | **VACANT** | **UTILISED** | **VACANT** |
| 10am - 11am | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** | **VACANT** |
| 11am – 12pm | **UTILISED** | **VACANT** | **UTILISED** | **VACANT** | **UTILISED** |
| 12pm - 1pm | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** |
| 1pm – 2pm | **VACANT** | **UTILISED** | **UTILISED** | **UTILISED** | **VACANT** |
| 2pm – 3pm | **UTILISED** | **UTILISED** | **UTILISED** | **VACANT** | **UTILISED** |
| 3pm – 4pm | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** |
| 4pm – 5pm | **VACANT** | **UTILISED** | **UTILISED** | **UTILISED** | **UTILISED** |
| Hours in use | 5 | 7 | 7 | 6 | 5 |

HOURS IN USE

Then the sum of all of the hours the resource was utilised needs to be calculated.

Total hours in user =

(5 +7 + 7 + 6 + 5) = **30 hours**

HOURS AVAILABLE

Total availability is the total amount of hours the resource is available to be used, 8 hours a day for 5 days.

Total availability: 5 \* 8 = **40 hours**

%

1.4.2 Occupancy rate

The occupancy rate represents how full the space is compared to its capacity.

Example:

Following on from the same scenario as above, the table shows how many people occupied the resource when it was in use.

|  |  |
| --- | --- |
| CAPACITY | 20 |
|  | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY |
| 9am - 10am | **0** | **20** | **0** | **20** | **0** |
| 10am - 11am | **10** | **20** | **20** | **15** | **0** |
| 11am – 12pm | **10** | **0** | **18** | **0** | **19** |
| 12pm - 1pm | **15** | **20** | **20** | **10** | **20** |
| 1pm – 2pm | **0** | **20** | **17** | **10** | **0** |
| 2pm – 3pm | **15** | **5** | **8** | **0** | **20** |
| 3pm – 4pm | **20** | **13** | **15** | **15** | **16** |
| 4pm – 5pm | **0** | **7** | **20** | **15** | **16** |
| HOURS IN USE | **30** |

HOURS IN USE

Add up all of the hours the resource was utilised (where the occupants is greater than 0).

Hours in use: **30**

NUMBER OF PEOPLE OCCUPAYING THE SPACE

Add up the number of occupants for when the resource is in use.

Total number of occupants: **469**

%

1.4.3 Utilisation rate

The utilisation rate represents how well the space is utilised.

Example:

Following on for the same example as above, now that the frequency and occupancy rates have been obtained, we can calculate the utilisation rate of the resource.

|  |  |
| --- | --- |
| CAPACITY | 20 |
|  | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY |
| 9am - 10am | **0** | **20** | **0** | **20** | **0** |
| 10am - 11am | **10** | **20** | **20** | **15** | **0** |
| 11am – 12pm | **10** | **0** | **18** | **0** | **19** |
| 12pm - 1pm | **15** | **20** | **20** | **10** | **20** |
| 1pm – 2pm | **0** | **20** | **17** | **10** | **0** |
| 2pm – 3pm | **15** | **5** | **8** | **0** | **20** |
| 3pm – 4pm | **20** | **13** | **15** | **15** | **16** |
| 4pm – 5pm | **0** | **7** | **20** | **15** | **16** |
| HOURS IN USE | **30** |

OCCUPANCY RATE

How full the space is compared to its capacity.

Occupancy rate: **78%**

FREQUENCY RATE

Percentage of time space was in use compared to its availability.

Frequency rate: **75%**

%

Chapter 2

System Requirements Specification

2.1 SOLUTION OVERVIEW

2.1.1 Introduction

The goal is to develop a software solution which will aid in the planning and management of resources within a modern office environment.

Research has shown that technology companies appear to be most flexible and are more likely to adopt an agile working environment. For this reason the prototype solution was developed with a technology company in mind. However, this was mainly to aid with test data. The end solution should be versatile and adaptable to all office environments.

Research shows that employees perform better when they can control and choose their working environment. This was one of the main motivations for the system, a clear focus on allowing employees to choose the most appropriate space for them to carry out their task, as opposed to simply automatically assigning the next available space. There should be no concept of desk ownership; if a resource is required it must be booked.

2.1.2 End users

The system will be designed with two end users in mind:

1. Administrators – all end users who are added to the system as administrators will have access to the Executive Officer Dashboard and the resource booking functionality. Administrators will also be able to add, update and remove users, teams, resources and time slots.
2. Non-Administrators – all end users who do not have administrative rights will only have access to the resource booking functionality.

2.1.3 Executive Officer Dashboard

The system must have an intuitive user experience design. Information relating to employees, teams, resources and time slots should be easy and straightforward to add to the system. Once all of the information has been entered the system will provide an Executive Officer Dashboard that will display any relevant Key Performance Indicators that the company requires to effectively manage their resource allocation. The dashboard will display the frequency, occupancy and utilisation rates in the form of charts and graphs which will aid the end user in making decisions on how best to manage and allocate their resources to make the workspace more cost efficient. The Key Performance Indicators should be displayed per week, for the overall company and for each resource.

2.1.4 Resource Allocation

Beyond providing a dashboard to aid in analysis the spatial allocation measurements for the company, the system must also manage the booking of resources. Users should have access to functionality to select where to work. On top of this, functionality to automatically assign a workspace to a user should be implemented to provide the user with a simple one-step solution to booking a workspace.

When booking a resource, consideration will have to be taken into the schedule of each resource to ensure it is not double booked or booked beyond its capacity. All employees should have no more than one resource assigned at one time.

2.2 ASSUMPTIONS

Due to the time constraints that have been set for developing the solution the following assumptions have been made:

* A prototype solution will be developed for one office consisting of a single one level building, for an IT company.
* When an employee is off work on annual leave or working from home they do not need to book any resources for said days.
* It is the responsibility of each user to book their resources for the upcoming week.
* Archiving of the “Booking” table in the database will not be considered during the development of the prototype.
* Users only have to book resources for Monday – Friday.

2.3 Functional Requirements

The functional requirements state the functionality that should be implemented and available at the end of the system. Each requirement was carefully considered against what was required for the end solution and what was achievable in the time frame provided.

1. Administrators shall have access to functionality to add new users, teams, resources and time slots to the system
2. Administrators shall have access to give users administrative rights for the system
3. Accounts shall be automatically created for each user when they are added to the system
4. Information regarding existing users, teams, resources and time slots should be available to the administrator to update
5. Information regarding existing users, teams, resources and time slots should be available to the administrator to delete
6. Administrators shall have access to functionality to add members to teams
7. Administrators shall have access to an executive officer dashboard which will provide information of measurements of utilisation
8. Analytics for the key performance indicators on the executive officer dashboard shall be displayed in the form of charts and tables
9. Each user should have access to an allocation timetable displaying all resources they have booked for the current week and the following week
10. A user shall have access to functionality to delete a resource booking which is displayed in their allocation timetable
11. A user shall have access to functionality to book a resource for themselves for a single time slot
12. A user shall have access to functionality to block book a resource for themselves for multiple time slots across a single or multiple days.
13. A user shall have access to functionality to book a resource for a group booking; including add attendees and teams to the booking
14. The system shall allow users to add attendees to an existing group booking
15. Group bookings must be accepted by each users before it is added to their allocation timetable
16. The system shall prevent users from booking multiple resources for the same time period
17. The system shall keep a timetable of allocation for each resource
18. The system shall prevent a resource being double booked
19. The system shall prevent a resource from being over booked
20. The system should provide functionality to allow a user to automatically assign themselves a resource for the entire week

2.4 Non-Functional Requirements

1. The system should be mobile compatible
2. Modified data on the system should be updated in the database within 2 seconds
3. The system should allow the user to easily switch between booking a desk, room or other resource
4. The UI should be intuitive and user friendly, requiring minimal training for the end user

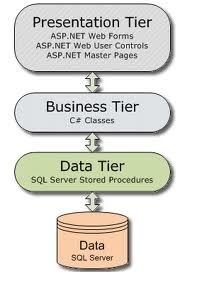
Chapter 3

Design

This chapter provides an overview of the overall architecture of the end solution. An insight into how each layer was implemented will be discussed along with reasoning as to why each decision was made.

3.1 SYSTEM ARCHITECTURE

Figure 3: Architecture overview

A three tiered architecture approach was selected as it allows for separation of concerns. During the course of research into different architecture approaches, it was noted that a vast amount of support and guidance was available for implementing an n-tiered architecture, especially with ASP.NET MVC.

The end solution, therefore consists of a presentation layer, a business/domain layer and a data layer.

The separation into layers results in a more practical approach which is easier to maintain and extend, resulting in a flexible, manageable and extendable end solution.

Having all of the business logic in a separate layer allows the service methods to be updated as business rules change. Hence, only the project containing the domain layer would require an update, and no changes are required to the data layer or the presentation layer. Within the solution, each layer was created as a separate project to further separate all of the areas of concern.

3.2 Presentation Layer

The presentation layer consists of an ASP.NET MVC application containing a user interface which the end user interacts with. Using MVC provides a solid platform for separation of concerns; it is also easy to integrate with other frameworks and libraries such as JQuery and Highcharts. The MVC framework provides a solid structure which makes unit testing more manageable, providing an extra layer of protection.

3.3 Domain/Business Layer

The domain layer consists of C# classes each of which comprise of a number of service methods containing the business rules and logic for the system. The business rules are separated out into components making the solution more manageable. This means, the is a User Service containing all of the service methods required to get, manipulate, delete and add users to the system. The same pattern will follow for teams, resources, time slots, bookings, accounts and rate calculations. The domain layer communicates with the data layer to retrieve the data to then filter and manipulate it into the required results and then send them to the presentation layer to be displayed for the end user.

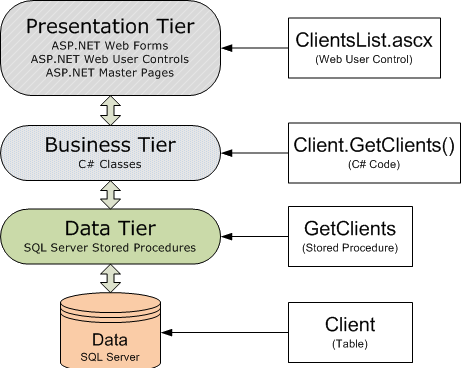
3.4 Data Layer

Figure 4: Architecture Design

The data layer follows an entity framework code-first migrations approach. This approach was selected due to its flexibility. It allows the database structure to be changed and updated as new requirements are added. Also, when more thought and understanding is put into a requirement it provides a clearer view of what the structure should be. This results in a change of the current database structure. The data layer is used to interact with the database in order to access, manipulate and add data.

3.5 USER INTERFACE

There were two main endpoints in mind for this solution. Ideally, there would be two end solutions, a phone application and a web application. The phone app would simply allow users to book a resource and view their allocation. And the web app would allow full use of the system, including access to the executive officer dashboard.

Although, due to time constraints it was decided to design one end solution that would cater for both end points. For this reason, during the design phase it was decided that the solution should be responsive and compatible to use on both mobile phones, computer, tablets, etc.

3.5.1 Design Overview

An important objective is for the end system to have a clean and professional layout. Consistency and ease of use are vital for the solution to be user friendly and intuitive. The application UI consists of Twitter Bootstrap 3. Twitter Bootstrap provides a solid structure which allows the system to be responsive, and therefore compatible with different hardware devices and browsers.

3.5.2 User Interface Design Guidelines

During the design phase, a brief list of design guidelines was created. The design guidelines contain an overview of how UI components should be implemented and displayed on screen. Having design guidelines in place aided in keeping the user interface consistent across the whole solution, which makes the system easier to use and in turn increases the user experience. Refer to appendix A to view the list of design guidelines.

3.5.3 Screen UI Mock-Ups

Screen mock-ups were also designed before any implementation started. This helped to visualize what the end product should look like and also provided a clear goal to work towards. Although producing the mock-ups was initially very time consuming and a tedious task, in the long run, it saved time during implementation as the design decisions had already been made. At times, the implementation did stray away from the design mock-ups, but the overall, concept and functionality remained the same. Refer to appendix B to view the initial UI wireframes.

Chapter 4

Implementation and Testing

This chapter provides a detailed description of the implementation approach and the testing strategies adopted in producing the end product. All of the technology choices, libraries and frameworks used were chosen carefully and a rationale for each choice is provided. This chapter also includes an insight into how testing was carried out to ensure the system meets the requirements and that the functionality is performing as expected.

4.1 IMPLEMENTATION

4.1.1 Server

The end solution is comprised of one operating system and two servers, a database server and a web server.

When researching for a database server the main requirement was for platform that could support complex SQL queries and more importantly store data in a relational way. Only open source options were considered. In the end Microsoft SQL Server was chosen due to the amount of support, information and guidance available. The SQL Server Management Studio meant that all data in the database could easily be viewed and analysed during development, helping to enhance performance and provide a vital insight into the data while debugging. MySQL, CSQL and PostgreSQL were also considered as potential database servers.

Internet Information Services (ISS) was chosen for the web server, which is hosted on Microsoft server (the operating server). Apache and nGinx were also considered as potential web servers but were not suitable for dot net applications without using a port for the dot net framework such as Mono for Linux. For this reason, ISS was selected as the best approach due to its ease to integrate with the solution framework and it’s compatible with most browsers and operating systems, which is one of the non-functional requirements for the solution.

4.1.2 Languages

All backend development was completed in C#. Due to using a code-first migration approach to create the database, no SQL code was required for the database creation. However, SQL was still used throughout the project to query the database. LINQ statements were used to write query expressions to retrieve, manipulate, delete and add data to the database. HTML, CSS and JavaScript were used in the views for the UI.

Visual Studio 2012 was used as the Integrated Development Environment (IDE) for all development on the end solution.

4.1.3 Libraries

A number of libraries were included in the end solution to aid in the development of the system and to enhance the end product. All of the libraries used were open-source.

|  |  |  |
| --- | --- | --- |
| **Library** | **Version** | **How it was used** |
| Highcharts | Highcharts JS v4.0.1 | Highcharts is a JavaScript library which provides an easy framework to display interactive charts on a web page. The Highcharts library was used to display all of the charts for the dashboard views. |
| Bootstrap | Twitter Bootstrap v3.3.7 | Bootstrap is a framework which contains HTML and CSS design templates. Overall, the templates and interactive components which were readily available saved time and provided optimum results to the look and navigation of the end system. |
| JQuery | JQuery JavaScript v1.12.4 | JQuery is a javaScript library which simplifies writing javaScript code. The use of JQuery throughout the project saved time, enabling Ajax interaction and event handling to be implemented with as much ease as possible. |

4.1.4 Coding Conventions

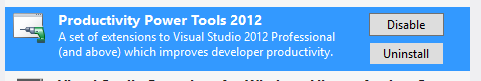
A series of coding conventions were followed throughout the project to ensure consistency and maintain coding standards. The *Productivity Power Tool 2012* was added as an extension tool to the project. This tool has a number of shortcuts to improve developer productivity and helps maintain readable and maintainable code.

Figure 5: Productivity Power Tool

|  |  |
| --- | --- |
| **Coding convention** | **Example** |
| All class names should have upper-camel case naming convention | *ResourceController.cs* |
| All variables a lower-camel case naming convention | var *name* = “Testing”; |
| All parameters should have a lower-camel case naming convention | Index(Guid? *id*) |
| Where required, longer names should be used to favour readability | *CalculateFrequencyBetweenTwoDates* instead of CalculateFrequency |
| All migrations should follow an upper-camel case naming convention | *AddedUserTableToDatabase* |
| All interfaces should begin with an “I” and follow an upper-camel case naming convention | *IResourceController.cs* |
| Partial views should begin with a “\_” symbol and follow a lower-camel case naming convention | \_*resourceInformation.cshtml* |
| Where appropriate, region breaks should be used | *#region HelperMethods*  *#endregion* |

4.1.5 Source Control

GitHub was used for source control. GitHub was chosen as it integrates nicely with GitLab which was a requirement for all Queen’s students. The project was set up to facilitate both repositories meaning there were consistently two versions of the system that could be accessed at any time and were simultaneously updated anytime new code was pushed. GitHub was used on top of GitLab as GitHub provided issue tracking and a nice user-friendly interface showing the different branches which were active.

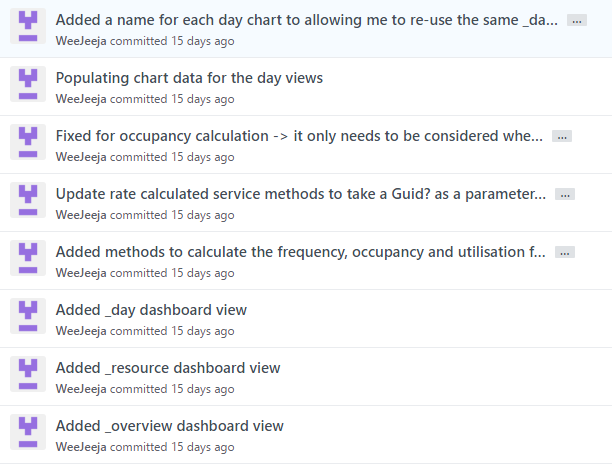
Although there is only one developer for this system, thought was still given to the practice of continuous integration. Commits were made regularly, and were as small as possible. Clear descriptions were added to each commit to allow reverting, if necessary, easier. Frequently integrating new functionality and updates into the master code base helped to keep the bug count to a minimum and made regression testing easier.

Figure 6: GitHub Screenshot

4.1.6 Development

Figure 7: Entity models

DATABASE

The first stage of development involved setting up the database. Entity Framework was used and a code-first migrations approach was implemented. A data model for each required table was created.

The DBContext was then added and contains a list of *DBSet* classes which each present an entity.

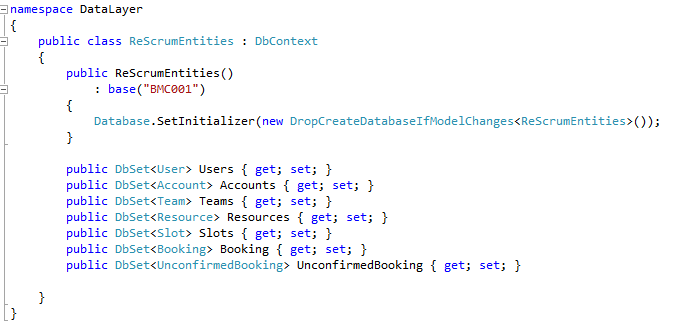


Figure 8: DBContext class

A migration was added anytime a data model class was updated or when the DBContext class was changed. Whenever the database is created all the migrations are executed and a table for each entity and all of the required attributes are added to the database.

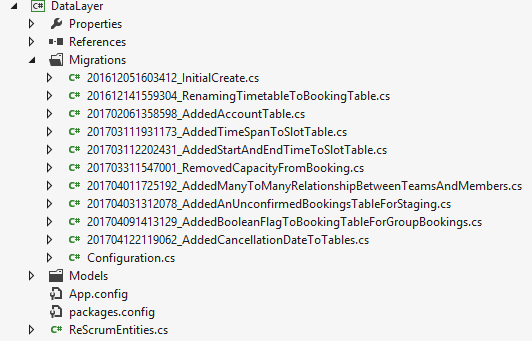


Figure 9: Migrations folder

All development was carried out in an iterative process. For instance, the first stage of the development plan was to capture all of the data relating to the company, e.g. their employees, resources, opening hours and teams. Therefore the users, resources, teams and time slots tables were required to be implemented at the beginning. The code first migrations approach facilitated an iterative development cycle as it allowed the database to be extended when required. Later, when the resource booking functionality had to be implemented, a migration was added to create the new Booking table. Figure 10 shows the final structure of the database. The E-R diagram displays all of the entities in the final system, the key fields and foreign keys for each entity and the relationships between the entities.

Figure 10: Entity-Framework diagram for the end system

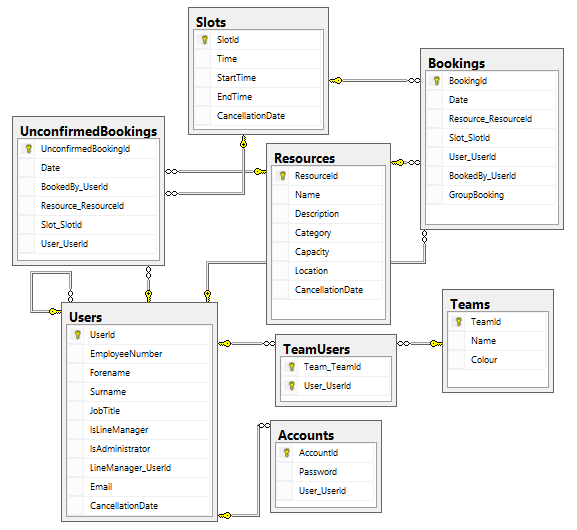
****

Figure 10: Entity-Framework diagram for the end system

MODELS

As mentioned above, entity models are used to represent each table in the database. Each entity model defines the type for each column and how it should be stored for the table it represents. Entity models also define how data is connected together, it states the foreign keys and the relationships. Refer to figure 10 to view the E-R diagram for the end system.

For example, the relationship between users and teams is many-to-many as each user can be a member of many teams and each team has many members. This relationship is set up in the entity models as shown in figures 11 and 12 with the use of the *virtual* keyword. Even though the DBContext does not contain a DBSet for *TeamUsers*, Entity Framework knows to create this linking table in order to manage the many-to-many relationship.

Figure 11: Team entity model

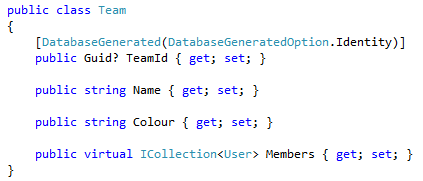
 

Figure 13: Database tables

Figure 12: User entity model



Wrapper models are used in the domain layer. Models contain domain specific properties and contain no knowledge of validation rules or storage procedures.

Figure 14: View model

View models are used in the presentation layer. Data annotations are used for validation, and attributes such as display names were also added. View models are used to state how the data should be presentated on screen and to add validation restrictions on data capture.

The distinction of models in each layer allows for better separation of concerns. For example, if only one model was used for every layer the display concerns, e.g. *DisplayName* attribute, would be missed with persistence concerns, e.g. *ForeignKey* attribute.

SERVICE

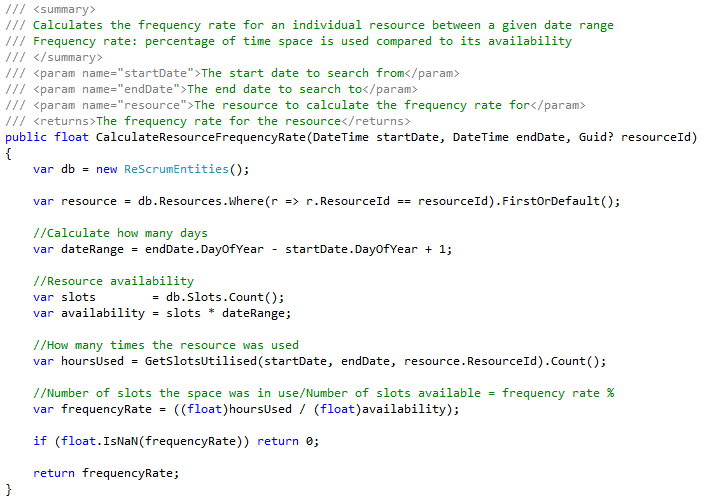
A service class was created for each component, e.g. *ResourceService*, *RateCalculationService*, in the domain layer. This approach is a more maintainable framework as it provides clear modules for logical functionality grouping. The domain layer has a reference to the data layer. Each service class contains methods which retrieve, modify and delete data from the database using LINQ statements. All of the business rules and calculations are performed in the service methods. The service methods provide the controllers on the presentation layer with a means to interact with the database, without having any knowledge or reference of it.

Figure 15: Service method to calculate resource frequency rate

CONTROLLERS

Once again, a controller for each component was created. The presentation layer has a reference to the domain layer. An instance to any of the required domain layer services was added to the controller to provide access to the service methods. GET methods were used to retrieve data to display for the user. POST methods handled validation errors and more often called service methods which modified, added or deleted data from the database.

Where possible, methods were coded with a generic framework to allow future development to take place. More importantly, however, generic methods allow for re-use and decrease the amount of lines of code to implement. For example, the dashboard displays a weekly overview of the current week, last week and next week, and then has tabs for Monday through to Friday of each week displaying an overview for that day. The method to get the weekly overview is generic, the method can be used to get the overview of any week, so long as a date is given. Using the date parameter, the method then calculates the starting date for the week. The method then generates the data for the weekly overview and passes it into the controller, figure 16 displays this functionality. A similar approach is followed for the day, a date is passed in as a parameter and the method calculates which day it is.



Figure 16: Get method for weekly overview generating data for the dashboard

VIEWS

The razor view engine handles the display of the views on screen. Most views were strongly typed to allow access to the properties within the view model. When applicable, partial views were implemented. The use of partial views allowed the same code to be used in multiple views which significantly reduced the amount of duplicated code and the need for multiple views.

Scripts were rendered on the views to allow the JavaScript, bootstrap, Highcharts and JQuery libraries to be used. In order to eliminate the need to render all of the scripts for the libraries on every view, a number of bundles were added to the *BundleConfig* class. The bundles were then rendered on the “\_*layout*” partial view which was used as the layout for all of the other views. Overall, this approach saved time and was more efficient.

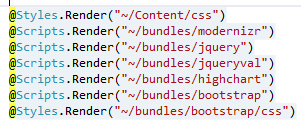
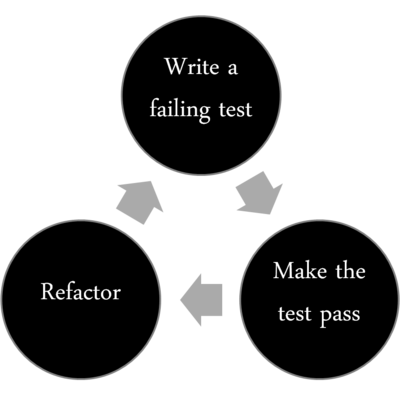


Figure 17: Bundle config class

Figure 18: Bundles being rendered in layout view

4.2 TESTING

4.2.1 Testing Strategy

It was apparent at the beginning of the project that following a Test Driven Development (TDD) approach was not feasible. TDD required the developer to write unit tests before implementing the code. After the test and code have been implemented, it is then time to refactor. TDD forces the developer to think ahead about what they are implementing and what variables, methods, services, etc. will be required to do so. This approach of making the developer think ahead produces cleaner code and overall results in fewer bugs due to high test coverage. Although, the benefits of TDD are clear, due to a lack of experience in following the TTD approach it was extremely time consuming. Writing unit tests was in itself a new technique to master and attempting the TTD approach simply overcomplicated matters.

Figure 19: TDD approach

To ensure the project progressed in the time frame required a combination to manual and automated testing was used.

4.2.3 Manual Testing

Manual testing involves physically using the system to check for defects. The tester plays the role of the end users, and is normally someone other than the developer. Manual testing should not be undervalued as automated tests cannot replace user intuition and they cannot speak for user experience

A test case template was created as a framework for the acceptance tests to follow. The purpose of the acceptance tests was to test the solution against the system requirements and assess whether the solution fulfils the requirement or not. Each test is given a requirement and a scenario to assess. The steps involved in creating the scenario, the expected and the actual outcomes are recorded on the test template.

Refer to appendix C to review the test case template that was used for all manual tests and for a sample of some of the tests that were performed.

4.2.2 Automated Testing

Overall, automated testing is less time consuming and less costly than manual testing. The main reason for this is that once created automated tests can be executed as many times and as frequently as necessary, at no additional cost.

Automated tests made the process of regression testing much easier. When new functionality was added to the system it was imperative to ensure everything that was previously functioning as expected still was. Unit tests were used for automated testing.

*NUnit v.3.6.1* was installed to write the unit tests, and the *NUnit 3 Test Adapter* tool was also installed to enable the execution of the NUnit tests on Visual Studio (IDE). Although the unit tests were time consuming to implement, they provided a solid basis for regression testing. When code was added or updated, all of the unit tests could automatically be executed and provided immediate feedback on what functionality, if any, was affected by the code change.

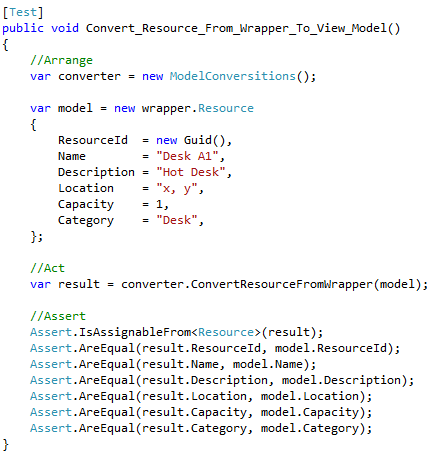
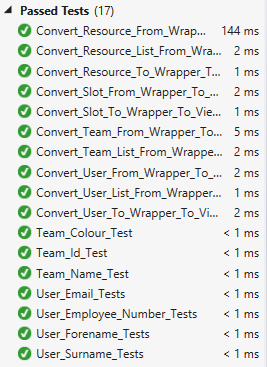
****Unit tests were implemented to test individual classes and methods in isolation. The *“Arrange–Act-Assert”* Pattern was followed for all unit tests. An example of the pattern can be seen in figure 20 below.

Figure 21: Execution unit tests

Figure 20: Unit test

Chapter 5

System Evaluation

5.1 System Evaluation Approach

The objective of the project was to develop a solution to aid a company in managing resource allocation within a modern office environment, which may follow practices such as agile working, allowing employees to choose where, when and how they work.

The functional and non-functional requirements have been evaluated against what was implemented. A usability questionnaire was also created to provide an unbiased evaluation of the system’s UI and how user-friendly and intuitive the overall end solution is.

The project developed allowed users to be added to the system as normal users or with administrative rights. Administrators are able to add, edit and delete resources, time slots, users and teams to the system. This approach means the end user has full control of how they want to manage their system. The time slots can be set to 1 hour slots, or 30 minutes. You can manage the utilisation of the company from 9am to 5pm or 24 hours a day. The company can select what works best for them. Resources can also be updated and deleted, as we expect the company to change their current resource structure once have access to their utilisation rates which will be provided in the executive officer dashboard.

An executive officer dashboard was created to allow the utilisation of the company’s resources as a whole to be analysed. A weekly overview of the company’s utilisation, frequency and occupancy rates for each week is available on the dashboard, along with a further breakdown of the utilisation of each resource. The level of breakdown into each resource provides the company with the means to analyse their resources and make changes if required. For example, if meeting room X with no projector is never used, but all other rooms with projectors have a high frequency rate, then it may be in the best interest of the company to add a projector to room X.

An overview of each day was also provided, this allows greater information for each day to be analysed. For example, a company would expect a higher utilisation rate at 5pm on a Monday compared to that of 5pm on a Friday, when the vast majority of employees will be leaving early in a flexible working environment.

All users can book resources for a single time slot, a block a time slots across a single day or multiple days, and add group bookings for multiple people. The primary focus when implementing the booking functionality was to allow employees to choose where they would sit. Research shows that employees are happier and are more productivity when they chose where they work, as they have the option to choose the best working environment for them (refer to section 1.3.1 for further information on this research). The system provides a clear overview of the resources a user has booked. Each user has access to a resource allocation timetable showing all of their bookings for the week, allowing them to clearly see where they will be working and make any changes required.

5.2 Evaluation of Functional Requirements

Functional requirements were created in the initial planning stage for the solution. A total of 20 requirements represent the functionality that should be implemented and available in the end system for it to be considered successful. Out of the 20 requirements, 18 were fully implemented and their functionality is successful. 1 of the requirements was partially implemented, and another 1 was not implemented. Time constraints was the main factor as to why these two requirements were not fully implemented.

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | IMPLEMENETED | COMMENT |
| R.1 - Administrators shall have access to functionality to add new users, teams, resources and time slots to the system | YES | Users, teams, resources and time slots can be added to the system and are successfully saved in the database. There is a many-to-many relationship between users and teams to represent the members of each team. |
| R.2 - Administrators shall have access to give users administrative rights for the system | YES | Users can be set as administrators when they are initially added to the system. Existing users can also be updated to have administrative rights, and they can also have their rights removed. |
| R.3 - Accounts shall be automatically created for each user when they are added to the system | YES | When users are added to the system a user account is automatically created and added to the database. |
| R.4 - Information regarding existing users, teams, resources and time slots should be available to the administrator to updated | YES | All information relating to users, teams, time slots and resources can be edited and is successfully updated in the database. The only field that cannot be updated is the Id for each record which uniquely represents it. |
| R.5 - Information regarding existing users, teams, resources and time slots should be available to the administrator to delete | YES | Teams can be successfully deleted from the system. However deleting time slots, users and resources affects the utilisation measurement rates due to the foreign keys in the booking table, meaning the booking they relate to would also have to be updated. To work around this a cancellation date was added to each of the tables. Any records with a cancelation date were not displayed on the system to interact with. However the records are still used in the rate calculations for the time periods before the cancellation date in order to return an accurate result. |
| R.6 - Administrators shall have access to functionality to add members to teams | YES | A user can be susccesfully added to a team, creating a record for said user and team in the team member table representing the many-to-many relationship in the database. |
| R.7 - Administrators shall have access to an executive officer dashboard which will provide information of measurements of utilisation | YES | Only users with administrative rights have access to the executive officer dashboard. |
| R.8 - Analytics for the key performance indicators on the executive officer dashboard shall be displayed in the form of charts and tables | YES | Charts and tables are displayed on the dashboard with accurate calculates for the frequency, occupancy and utilisation rates. |
| R.9 - Each user should have access to an allocation timetable displaying all resources they have booked for the current week and the following week | PARTIALLY | An allocation timetable for each user is available for the current week, but not for the week ahead. The logic to generate the allocation timetable is generic and takes two dates as parameters, and therefore can be reused for the timetable for the week ahead. However, due to time constraints, there was no time left to finish fully implementing this requirement. |
| R.10 - A user shall have access to functionality to delete a resource booking which is displayed in their allocation timetable | YES | Bookings can be successfully deleted from the database. |
| R.11 - A user shall have access to functionality to book a resource for themselves for a single time slot | YES | Booking is successfully added to the database with correct foreign keys for the user, time slot and resource. |
| R.12 - A user shall have access to functionality to block book a resource for themselves for multiple time slots across a single or multiple days. | YES | A booking for each time slots for every day in the date range selected is successfully added to the database. |
| R.13 - A user shall have access to functionality to book a resource for a group booking; including add attendees and teams to the booking | YES | Bookings are successfully added to the unconfirmed booking table for all attendees added to the booking. |
| R.14 - The system shall allow users to add attendees to an existing group booking | YES | A booking for the new attendee added is successfully added to the unconfirmed booking table. |
| R.15 - Group bookings must be accepted by each users before it is added to their allocation timetable | YES | Users have a list on their home page of unconfirmed bookings, when accepted they are successfully added to the booking table and deleted from the unconfirmed booking table in the database. |
| R.16 - The system shall prevent users from booking multiple resources for the same time period | YES | If a user attempts to book a resource for a time slot already occupied in their timetable, the system will update the existing booking with the new booking added. |
| R.17 - The system shall keep a timetable of allocation for each resource | YES | The booking table successfully tracks when each resource is booked and by whom. |
| R.18 - The system shall prevent a resource being double booked | YES | When creating a booking, the system will only display the resources that are available for the booking criteria entered, preventing the possibility of double booking. |
| R.19 - The system shall prevent a resource from being over booked | YES | When creating a group booking, if more attendees than the resource’s capacity are added, then a validation message is displayed informing the user to select a different resource. |
| R.20 – The system shall automatically email a user when they have been added to the system with their account details | NO | This functionality was not vital for the end solution and therefore was left until the end to be implemented. However, due to time constraints, this requirement was not implemented. |

5.3 Non-functional Requirements Evaluation

Non-functional requirements are used to measure the operation of the system as opposed to its functionality. Overall, due to the milted amount of time available to implement the end solution, the initial non-functional requirements were kept very basic. Out of the 4 requirements, 2 of them were considered to be successfully implemented. The other two were considered to be partially implemented as they were difficult to test accurately with the time and resources available.

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | IMPLEMENTED | COMMENT |
| N.1 - The system should be mobile compatible | **PARTIALLY** | For the most part, all pages are dynamically scalable and can be viewed on different screen dimensions. However, due to time constraints, no testing was carried out on a mobile device and therefore the success of this requirement cannot be fully analysed. |
| N.2 - Modified data on the system should be updated in the database within 2 seconds | **PARTIALLY** | All data that is updated on the system is successfully updated in the database. However, this requirement is difficult to test as it depends largely on network connectivity. |
| N.3 - The system should allow the user to easily switch between booking a desk, room or other resource | **YES** | When creating a booking all resources are displayed and the user can easily choose which one is most appropriate for their needs. |
| N.4 - The UI should be intuitive and user friendly, requiring minimal training for the end user | **YES** | This requirement was largely testing during the usability questionnaire, which concluded that 100% of testers found the system easy to use. |

5.3 Usability questionnaire

A questionnaire was used in order to evaluate how easy or difficult the system is to navigate and use. 10 testers exploratory tested the system and then filled out the questionnaire. This provided unbiased feedback and provided a way of evaluating how intuitive and user friendly the end system was. Each tester was given a brief overview of the concept of the system and were then left with a series of tasks to complete on the system.

The questionnaire consisted of 8 questions relating to the consistency, performance and usability of the system. Each tester had to *strongly agree*, *agree*, *neither agree/disagree*, *disagree* or *strongly disagree* with each statement. The questionnaire was kept basic and easy to complete in order find as many willing testers as possible.

The results showed that 100% of the testers like the fact that they get a choice in where they sit. After further discussion it was highlighted that 8 out of 10 of the testers would make their decision on where to work based on who would be working close to them, while the other two said it would have no influence on them.

70% of the testers agreed that the information displayed on the dashboard was useful, while the other 30% stated that they neither agreed nor disagreed. This outcome could be due to the fact that 3 out of 10 of the testers may not have fully understood the brief overview about the purpose of the system.

Overall, the statements relating to the usability of the system scored really well. 100% of the testers either agreed or strongly agreed that the system was easy to use, the error messaged made sense and that they now feel confident using the system after only a short period of time. 8 out of 10 of the testers strongly disagreed with the fact that the system was overly complex, and 100% also disagreed or strongly disagreed with the fact that a large amount of training is required in order to use the system.

90% of the testers stated that they agreed that the overall look of the system was consistent. Refer to appendix D to view the questionnaire template and see all of the ten completed questionnaires.

Chapter 6

Conclusion

6.1 Strengths

Using Entity Framework code-first migrations was an excellent decision for the solution as it allowed the database structure to be updated and changed throughout implementation. This approach to development combined with the overall three-tiered architecture of the end solution provides a more maintainable framework and allows the possibility of extending the project in the future.

During development, deep consideration was given into what was being implementation and what effects it would have on the final system. Every technical decision was carefully carried out. This resulted in the discovery that one of the requirements would have adverse effects on the system. If requirement R.5 was implemented as initial stated, to have the ability to delete users, resources and time slots, then this would have had a negative effect on the utilisation rates for any days with bookings linked to said user, resource or timeslot. For this reason, the code was carefully implemented and refactored to add a cancellation date to each of the tables. This allowed past calculations to still use the records required, and prevented the records being included in any future bookings.

Overall, the Executive Officer Dashboard provides a user friendly interface with vast amount of information of the spatial utilisation of the company. The key performance indicators are calculated for each day and week. Further analysis into each resource is also provided, showing the frequency, occupancy and utilisation rates for each resource and how they were utilised during the week. The information provided on the dashboard allows the end user to analyse their spatial utilisation for their company and make changes to improve it.

6.2 Weaknesses

The system relies heavily on all employees within a company consistently using it for it to return accurate utilisation calculations. There is nothing preventing employees from using resources without booking them, and also there is no mechanism to check that an employee is utilising the resource they have booked.

Although the system in scalable and dynamic, it would have been useful to have developed a separate mobile application which would only be used for booking resources and have no management dashboard. Although, this was not feasible in the strict time frame available.

Requirement R.20 was only partially implemented. This requirement was for the auto assign functionality, which would allow a user to click a button and be assigned a resource from 9am on Monday to 5pm on Friday, this part was implemented successfully. However, it would have been a better approach to select the resource closest to the employee’s line manager, instead of simply randomly selected a resource. The database does contains a line manager attribute for a user, so no changes would need to be carried out on the database if this was to be extended and implemented in the future.

6.3 Conclusion

The problem of resource allocation and spatial utilisation was well researched before development started. The research gathered was then used to produce a detailed list of functional requirements. The research discoveries led to one of the main focus points for the end system; allowing employees to choose where they work.

One of the main challenges with implementing the solution was a lack of developer experience. The amount of time that was required to understand and research frameworks and languages was severely underestimated. It was difficult learning how each component fitted together. The initial set-up of the project architecture and the research that was involved was very time consuming. Learning how to set up Entity Framework and how to follow the code-first migrations approach was an arduous task. In saying this, it is believed that the architecture, languages and frameworks were the right choices for the end solution.

The objective of the project was to develop a solution to aid a company in managing resource allocation within a modern office environment. Out of the 21 requirements initially stated, 17 were fully implemented, 2 were partially implemented and 1 was no implemented. Overall, the solution delivered was successfully. The end system is adaptable to function in any office environment, as it provides the end user with functionality to add resources in a manner which suits their company. The system fits in perfectly with the concept of agile working, which was one of the main inspirations for the project as a whole.

6.4 Future Recommendations

Although the end solution is considered to be a successful solution to the objective of managing resource allocation, there are still a number of improvements that could be made to the system, and a number of different approaches that could have been adopted instead of the solution developed.

The system could be stressed tested in order to verify how many users it can support at once during heavy network traffic. Without such testing, it is difficult to verify whether the system will be able to handle the demands of a whole company.

The solution provided relies on people using the system consistently and accurately. There is nothing to stop anyone from utilising a resource without booking it first. Calculations are also biased as to how resources are being used while they are booked. For example, a user may book a desk for the day, but the system does not provide any indication on how long the user actually spent at the desk. Users can book a resource and go for lunch, coffee breaks, toilet, etc. leaving the utilisation rates inaccurate for the company.

There are monitoring devices and apps that exist for tracking employees which could verify whether an employee is at their allocated resource or not. However, the vast opinion on them is that they are an invasion of personal privacy. Companies could also track when computers, laptops and phones are utilised, indicating that the desk or room they are in is being utilised. However this is still only a solution for resources with a technical device available, and there may be some resources, such as a team lunch area which may not have any of those devices.

References

**The first citation should be the URL to the software code repository which should contain the code and any other resource required to run the software.**

Famuyide, Stephanie, and Stephanie Famuyide. "15 Tips For Writing Better Requirements". *Business Analyst Learnings*. N.p., 2017. Web.

"What Is Executive Dashboard? - Definition From Whatis.Com". *SearchCIO*. N.p., 2017. Web.

"Writing Quality Requirements". *Processimpact.com*. N.p., 2017.

Thinking, Fresh. "Utilisation Analysis | Fresh Business Thinking". *Fresh Business Thinking*. N.p., 2017. Web.

<http://www.smg.ac.uk/documents/utilisation.pdf>

<http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

<http://www.requirements.com/Glossary/NonFunctionalRequirements/tabid/91/Default.aspx>

<http://softwareengineering.stackexchange.com/questions/162399/how-essential-is-it-to-make-a-service-layer>

<https://msdn.microsoft.com/en-us/library/ff648105.aspx>

<https://www.lucidchart.com/pages/er-diagrams>

<http://www.conceptdraw.com/examples/er-notation>

<http://survey.datassentialsurveys.com/help/Inquisite_ER_Diagram.htm>

<http://kc.litiumstudio.se/documentation/previous-versions/litium-studio-4-6/litium-studio-4-6-2/litium_studio_ecommerce/architecture-design/er-diagrams>

<http://www.tableau.com/sites/default/files/media/which_chart_v6_final_0.pdf>

<http://www.investopedia.com/terms/c/capacityutilizationrate.asp>

<http://www.businessballs.com/pareto-principle-80-20-rule.htm>

<https://en.wikipedia.org/wiki/Pareto_principle>

<http://ui-patterns.com/blog/The-8020-rule--the-Pareto-principle>

<http://www.archibus.com/ai/abizfiles/v23_help/archibus_help/user_en/Subsystems/webc/Content/web_user/space/planning/concepts/allocation.htm>

<https://hbr.org/2014/01/employees-perform-better-when-they-can-control-their-space>

<http://spendmatters.com/2011/01/05/six-strategies-for-it-cost-allocation/>

<http://www.flexibility.co.uk/flexwork/offices/facilities4.htm>

<http://staging.cl/wp-content/uploads/2014/07/Office-Space-How-much-is-enough.pdf>

<https://www.theregister.co.uk/2016/03/07/ibm_offshoring/>

<https://www.bloomberg.com/news/articles/2007-10-25/shaping-the-workforce-of-the-futurebusinessweek-business-news-stock-market-and-financial-advice>

<http://officeprinciples.com/agile-working/>

<https://forums.asp.net/t/1940843.aspx?Apply+3+layer+to+MVC+Application>

<http://stackoverflow.com/questions/24588838/entities-vs-domain-models-vs-view-models>

Appendices

Appendix A – User Interface Design Guidelines

|  |  |  |
| --- | --- | --- |
| **Component** | **Design guideline** | **Code reference** |
| Main headlines | All main headlines should use the heading 1 tag. | <h1> |
| Normal text | All test which is not a heading should be displayed in a paragraph tag | <p> |
| Table headings | All table headings should be in capitals and bold | <strong> |
| Tables | All tables should use the bootstrap table and hover classes | <table table-hover> |
| Button (go to create page) | All buttons that lead to a create page should use bootstraps primary button class | .btn-primary |
| Button (to delete a record) | All buttons that lead to a a record being deleted should use bootstraps danger button class | .btn-danger |
| Button (to go to new info page) | All buttons that navigate to a new page should use bootstraps information button class | .btn-info |
| Button (to go back) | All buttons that navigate back should use bootstraps default button class | .btn-default |
| Button (adds a new record) | All buttons that lead to a record being added should use bootstraps successful button class | .btn-success |

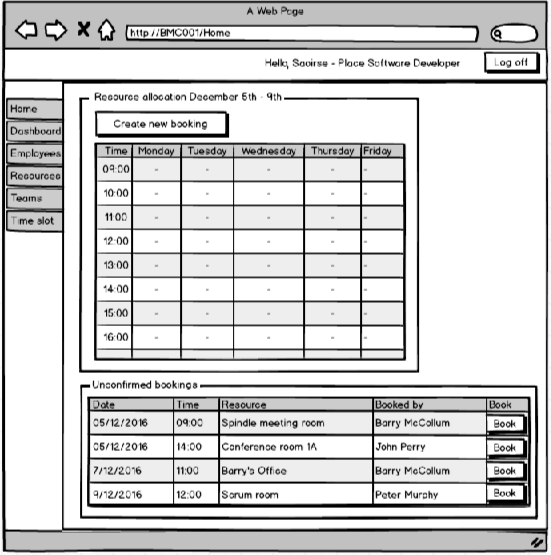
Appendix B – Screen Mock-Ups

Home page

The home screen will display an allocation timetable for the logged in user.

From this page, the user should be able to add a new booking by clicking on the “Create new booking” button, and also delete bookings by selected an individual entry in the timetable.

A list of unconfirmed group bookings should also be displayed which the user can either accept, which should add the booking into their timetable, or delete it, removing it from their list.



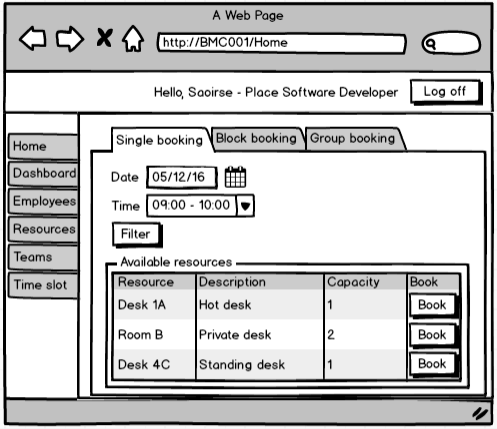
Only administrators will have access to this area

Single booking page

This page should be used to add a booking for a single time slot.

For example, a desk can be book for the 9am time slot on Monday 5th of December.

Clicking on the “Filter” button should return a list of resources that are available for the date and time entered.

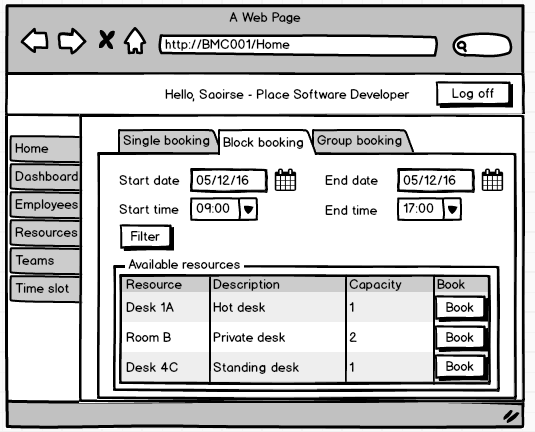
Clicking on the “Book” button should then create the booking.

Block booking page

This page should be used to add a booking for multiple time slots.

For example you can book at desk from 9am to 5pm for Monday 5th of December, or you can book a desk from 9am to 5pm from Monday 5th of December to Friday 9th of December.

Clicking on the “Filter” button should return a list of resources that are available for the duration of dates and times entered.

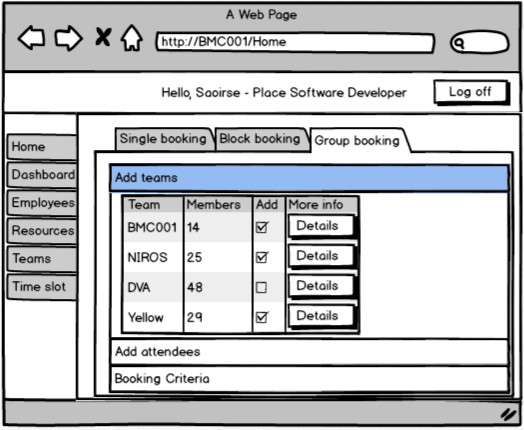
Clicking on the “Book” button should then create a booking for each time slot for each day in the date range.

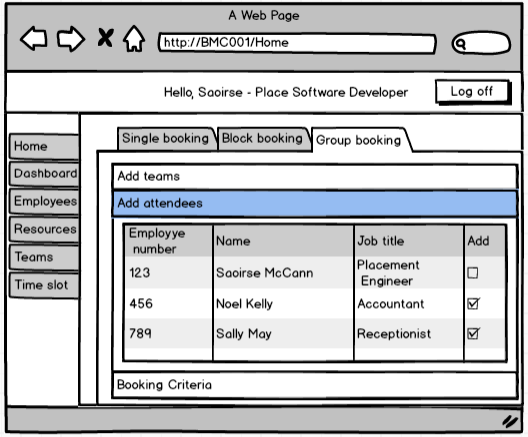
Group booking page

The group booking page should allow users to add a booking for multiple people, it should involve three steps

STEP 1: Select the teams that should be added to the booking.

During this step there should be a “Details” button for each team; this button should allow users to view a list of members in each team. It should open in a new tab.



STEP 2: Select the attendees that should be added to the booking.

During this step the user should have a list of all users on the system. A checkbox should be displayed beside each user allowing them to be added to the booking.

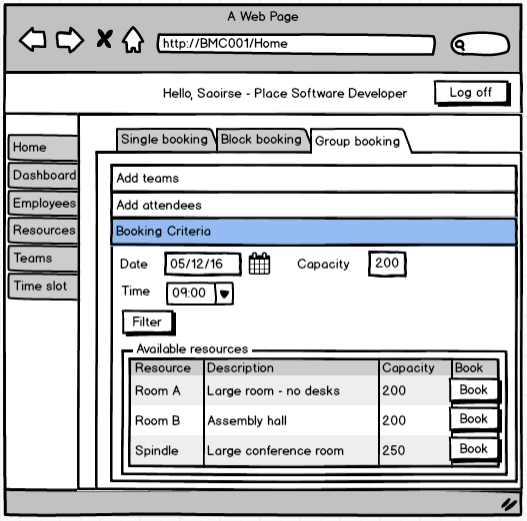
STEP 3: Enter the booking criteria.

The user then enters the date and time that the booking is for, along with the capacity required for the booking.

The “Filter” button should then display a list of resources which are available for the date and time entered.

An error message should be displayed if the user clicks on the “Book” button for a resource which does not have the capacity to hold the number of attendees/team members previously selected.

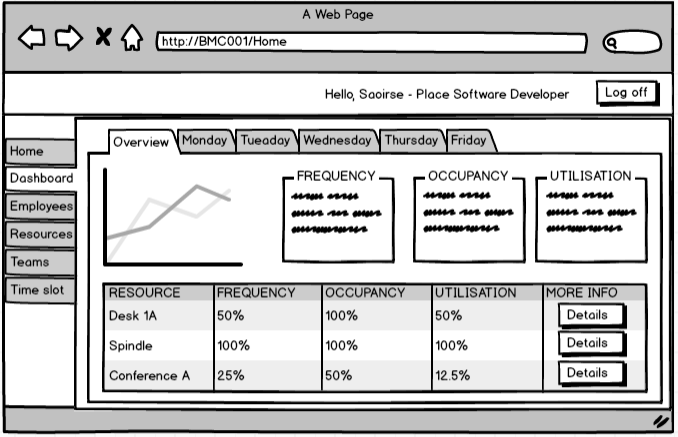
Clicking the “Book” button successfully should then add an unconfirmed booking for each attendee/team member included in the booking.



Dashboard overview page

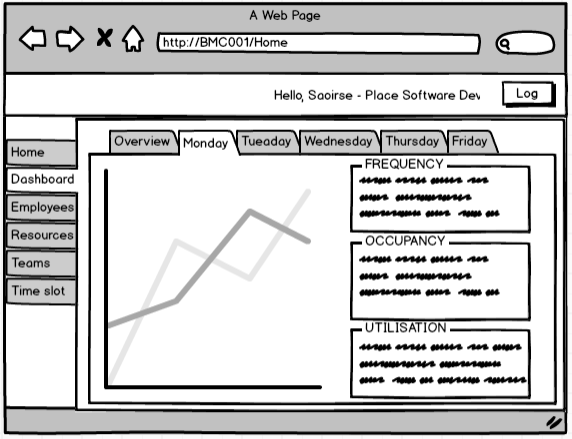
The Executive Officer Dashboard should contain a weekly overview showing the frequency, occupancy and the utilisation rates for the week. A graph should also be used to display this information.

The frequency, occupancy and utilisation rate should also be displayed for each resource. The “Details” button beside each resource should navigate to a resource overview page showing how that resource was utilised through the week



Dashboard day page

The Executive Officer Dashboard should contain a tab for each day of the week showing the frequency, occupancy and the utilisation rates for that day. A graph should also be used to display this information.



Appendix C – Acceptance Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | | 1. Administrators shall have access to functionality to add new users, teams, resources and time slots to the system | | | |
| **Scenario** | | Administrator adds new user to the system | | | |
| **Step #** | **Action** | | **Expected Outcome** | **Actual Outcome** | **Result** |
| 1 | Click on “Employees” tab | | The employee index page should be displayed with a table of all the current employees. “Add new employee” button should be displayed above the table. | The employee index page is displayed with a table of all the current employees. “Add new employee” button is displayed above the table. | **PASS** |
| 2 | Click on “”Add new employee” button | | The create employee page should be displayed with all of the fields to add an employee | The create employee page is displayed with all of the fields to add an employee | **PASS** |
| 3 | Enter employee details and click “Create” | | Should be redirected back to the index page and the new user should be displayed in the table of employees | Redirected back to the index page and the new user is displayed in the table of employees | **PASS** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | | 1. Administrators shall have access to functionality to add new users, teams, resources and time slots to the system | | | |
| **Scenario** | | Administrator adds new resource to the system | | | |
| **Step #** | **Action** | | **Expected Outcome** | **Actual Outcome** | **Result** |
| 1 | Click on “Resources” tab | | The resource index page should be displayed with a table of all the current resources. “Add new resource” button should be displayed above the table. | The resource index page is displayed with a table of all the current resources. “Add new resource” button is displayed above the table. | **PASS** |
| 2 | Click on “”Add new resource” button | | The create resource page should be displayed with all of the fields to add a resource | The create resource page is displayed with all of the fields to add a resource | **PASS** |
| 3 | Enter resource details and click “Create” | | Should be redirected back to the index page and the new user should be displayed in the table of resources | Redirected back to the index page and the new user is displayed in the table of resources | **PASS** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | | 1. Administrators shall have access to functionality to add members to teams | | | |
| **Scenario** | | Administrator adds an existing user to an existing team  Pre: there are users already in the system that are not members of the test team | | | |
| **Step #** | **Action** | | **Expected Outcome** | **Actual Outcome** | **Result** |
| 1 | Click on “Teams” tab | | The team index page should be displayed with a table of all the current teams. “Add members” button should be displayed beside each team in the table. | The team index page is displayed with a table of all the current teams. “Add members” button is displayed beside each team in the table. | **PASS** |
| 2 | Click on “Add members” button | | The add members page should be displayed with a list of all users who are not currently in the team, and an “Add member” button should be displayed beside each user | The add members page should is with a list of all users who are not currently in the team, and an “Add member” button is displayed beside each user | **PASS** |
| 3 | Click on the first “Add member” button | | Should be redirected back to the index page and a success message should be displayed at the top of the page stating that the user was added as a member for the team | Redirected back to the index page and a success message is displayed at the top of the page stating that the user was added as a member for the team | **PASS** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | | 1. Information regarding existing users, teams, resources and time slots should be available to the administrator to update | | | |
| **Scenario** | | Administrator updates team name  Pre: there is a team already is the system | | | |
| **Step #** | **Action** | | **Expected Outcome** | **Actual Outcome** | **Result** |
| 1 | Click on “Teams” tab | | The team index page should be displayed with a table of all the current teams. “Edit” button should be displayed beside each team in the table. | The team index page is displayed with a table of all the current teams. “Edit” button is displayed beside each team in the table. | **PASS** |
| 2 | Click on “Edit” button | | The edit page should displayed with all of the team’s details in an editable format. | The edit page is displayed with all of the team’s details in an editable format. | **PASS** |
| 3 | Edit the team name and click on the “Save” button | | Should be redirected back to the index page and team name should be updated in the index table. | Redirected back to the index page and team name is updated in the index table. | **PASS** |

Appendix D - Usability Questionnaire

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  |  | X |
| A lot of training is required in order to use the system |  | X |  |  |  |
| I now feel confident using the system |  |  |  | X |  |
| I thought the information provided on the dashboard was useful |  |  |  |  | X |
| I thought the error messages provided on the system made sense |  |  |  | X |  |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  | X |  |
| A lot of training is required in order to use the system | X |  |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  |  |  | X |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex |  | **X** |  |  |  |
| I thought the system was easy to use |  |  |  |  | **X** |
| I thought the overall look of the system was consistent |  |  |  | **X** |  |
| A lot of training is required in order to use the system |  | **X** |  |  |  |
| I now feel confident using the system |  |  |  | **X** |  |
| I thought the information provided on the dashboard was useful |  |  | **X** |  |  |
| I thought the error messages provided on the system made sense |  |  |  |  | **X** |
| I like the fact that I get a choice in where I work |  |  |  |  | **X** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  |  | X |
| A lot of training is required in order to use the system | X |  |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  |  | X |  |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  | X |  |
| A lot of training is required in order to use the system | X |  |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  |  |  | X |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  | X |  |
| I thought the overall look of the system was consistent |  |  |  | X |  |
| A lot of training is required in order to use the system |  | X |  |  |  |
| I now feel confident using the system |  |  |  | X |  |
| I thought the information provided on the dashboard was useful |  |  |  |  | X |
| I thought the error messages provided on the system made sense |  |  |  | X |  |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  |  | X |
| A lot of training is required in order to use the system |  | X |  |  |  |
| I now feel confident using the system |  |  |  | X |  |
| I thought the information provided on the dashboard was useful |  |  | X |  |  |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  |  | X |
| A lot of training is required in order to use the system | X |  |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  |  |  | X |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex |  | X |  |  |  |
| I thought the system was easy to use |  |  |  | X |  |
| I thought the overall look of the system was consistent |  |  |  |  |  |
| A lot of training is required in order to use the system |  | X |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  |  | X |  |
| I thought the error messages provided on the system made sense |  |  |  | X |  |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Strongly disagree** | **Disagree** | **Neither** | **Agree** | **Strongly agree** |
| I found the system unnecessarily complex | X |  |  |  |  |
| I thought the system was easy to use |  |  |  |  | X |
| I thought the overall look of the system was consistent |  |  |  | X |  |
| A lot of training is required in order to use the system | X |  |  |  |  |
| I now feel confident using the system |  |  |  |  | X |
| I thought the information provided on the dashboard was useful |  |  | X |  |  |
| I thought the error messages provided on the system made sense |  |  |  |  | X |
| I like the fact that I get a choice in where I work |  |  |  |  | X |

Appendix E – User Manual

**Appendices**

These should include as appropriate:

1. A User manual giving details on how to use the software, including details of input data, output formats and error messages.