

BOOKING SYSTEM FOR DEMONSTRATION CAMERAS

OLUKAYODE NICHOLAS AKINYOKUN

(201378586)

This dissertation was submitted in part fulfilment of requirements for the
degree of MSc Advanced Computer Science

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF STRATHCLYDE

SEPTEMBER 2014

DECLARATION

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.

(please tick) Yes [] No []

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is **17700**

I confirm that I wish this to be assessed as a Type 1 2 3 4 5
Dissertation (please circle)

Signature:

Date:

ABSTRACT

Fundamentally, booking systems are required to have the functionality to manage requests made by different users to access certain resources within a timeslot, in a way that optimises the overall usage of these resources and without conflicts in allocated timeslots. In order to effectively coordinate access to these resources, there is always need to implement scheduling.

Currently, the two different approaches that have been explored by researchers to solve this problem of finding optimal methods to allocate constrained resources are booking and queuing. Booking typically allows a user to reserve a timeslot in advance and this has the advantage of providing the user with guaranteed access at a known time. On the other hand, with queuing, when users makes a request to access a resource, they are added to a queue and as soon as that resource becomes available, it is allocated to the user at the front of the queue based on a First-Come, First-Served scheduling policy. This helps to maximise resource utilisation because resources are allocated to waiting users as soon as they become available. However, while these scheduling approaches and their various hybrids have been implemented and described in resource scheduling research literatures, there has been no justification for the most suitable approach to scheduling and most importantly, little is known about the advantages and disadvantages of integrating both approaches.

To this end, this research aims to further investigate and explore how queue-based and booking-based scheduling can be effectively integrated into a hybrid approach to improve resource utilisation in the context of online booking systems, whilst considering the issues that may arise when a system uses both approaches for scheduling.

The method used to achieve this involved the implementation of an experimental web-based booking system that allow customers to book cameras for a particular timeslot. The system was primarily evaluated by using load tests to determine the problems that resulted from integrating booking and queuing. Based on the results of these tests, booking was found to be more efficient in optimising resource usage and decreasing waiting times, although this depended on a number of factors such as the scheduling algorithm, the number of concurrent users, and the usage duration of each virtual user.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor, Dr. John Wilson, for his support, guidance and encouragement in the course of completing this research project.

In addition, I would like to thank the staff of the Department of Computer and Information Sciences, University of Strathclyde for their support throughout my studies at the University.

TABLE OF CONTENTS

CONTENTS

CHAPTER 1: INTRODUCTION

1.1.	Research Context.....	1
1.2.	Problem Description.....	2
1.3.	Research Questions.....	2
1.4.	Research Objectives.....	2
1.5.	Structure of Dissertation.....	3

CHAPTER 2: LITERATURE REVIEW

2.1.	Scheduling.....	4
2.2.	Practical Approaches to Scheduling.....	4
2.2.1.	Booking Scheme.....	5
2.2.2.	Queuing Scheme.....	6
2.2.3.	Priority Queue.....	8
2.2.3.1.	Types of Priority Queue.....	8
2.2.4.	Combination of Booking and Queuing.....	9
2.3.	Slot Allotment Algorithm.....	10
2.4.	Scheduling Policies.....	11
2.4.1.	Non-preemptive Scheduling Policies.....	11
2.4.2.	Preemptive Scheduling Policies.....	12
2.5.	Performance Evaluation of Scheduling Policies.....	13

CHAPTER 3: SYSTEM DESIGN

3.1.	Functional Requirements.....	15
3.2.	Non-functional Requirements.....	16
3.3.	Business Logic of the Booking Functionality.....	16
3.4.	Business Logic of the Queuing Functionality.....	16
3.5.	Class Diagram.....	17
3.6.	Development Approach.....	18
3.7.	System Architecture.....	19
3.8.	System Development.....	20
3.8.1.	HyperText Markup Language (HTML) 5.....	20
3.8.2.	Cascading Style Sheet (CSS) 3.....	20
3.8.3.	jQuery.....	21
3.8.4.	HyperText Preprocessor (PHP).....	21
3.8.5.	MySQL.....	21
3.9.	Database Design.....	22
3.10.	User Interface Design.....	23
3.10.1.	Customer’s User Interface.....	24
3.10.2.	Administrator’s User Interface.....	27

CHAPTER 4: DETAILED DESIGN

4.1.	Back-End Development.....	30
4.2.	User Class.....	30

4.3.	Booking Class.....	33
4.3.1.	timeToUnix() Method.....	35
4.3.2.	validateTime() Method.....	37
4.3.3.	validateBooking() Method.....	38
4.3.4.	scheduleBooking() Method.....	39
4.4.	Queue Class.....	41
4.4.1.	How the Priority Queue Works.....	42

CHAPTER 5: VALIDATION

5.1.	Test Strategy.....	49
-------------	--------------------	----

CHAPTER 6: SYSTEM EVALUATION

6.1.	Performance Evaluation.....	53
6.1.1.	Load Test 1.....	54
6.1.2.	Load Test 2.....	55
6.1.3.	Summary of Results.....	58
6.2.	User Evaluation.....	59
6.2.1.	Questionnaire Design.....	60
6.2.1.1.	System Usability Scale.....	60
6.2.2.	Analysis and Results.....	61
6.2.3.	Observations.....	63

CHAPTER 7: CONCLUSION AND REFLECTIONS

7.1. Conclusion.....64

7.2. Limitations.....65

7.3. Learning Outcomes.....65

7.4. Suggestions for Future Work.....65

References.....67

Appendices.....71

Appendix A: System Documentation.....71

 Appendix A.1: Requirements.....71

 Appendix A.2: Database Setup.....71

 Appendix A.3: Configuration.....73

Appendix B: User Guide.....74

 Appendix B.1: Customer User Guide.....76

 Appendix B.2: Administrator User Guide.....81

Appendix C: Use Case Diagram.....86

Appendix D: Class Diagram.....88

Appendix E: Results of Load Tests.....89

Appendix F: Email Invitation.....91

Appendix G: Participant Information Sheet.....92

Appendix H: User Evaluation Questionnaire.....94

Appendix I: Database Attributes.....96

LIST OF FIGURES

Figure 3.1:	The booking system's architecture.....	20
Figure 3.2:	Overview of the database design.....	22
Figure 3.3:	Registration form.....	23
Figure 3.4:	Login form.....	24
Figure 3.5:	New booking form – (customer).....	24
Figure 3.6:	Manage booking table – (customer).....	25
Figure 3.7:	Queue form.....	26
Figure 3.8:	Demo countdown timer.....	26
Figure 3.9:	Waiting list page.....	27
Figure 3.10:	New booking form – (Admin).....	28
Figure 3.11:	Manage booking table – (Admin).....	28
Figure 3.12:	Manage users table.....	29
Figure 4.1:	<code>addNew</code> method.....	31
Figure 4.2:	<code>toggleUserStatus</code> method.....	31
Figure 4.3:	<code>checkAvailability</code> method.....	32
Figure 4.4:	<code>validateEmailUsername</code> method.....	32
Figure 4.5:	<code>updateStatus</code> method.....	33
Figure 4.6:	<code>signIn</code> and <code>signOut</code> method.....	33
Figure 4.7:	Booking form PHP script.....	34
Figure 4.8:	<code>EndTime</code> method.....	34

Figure 4.9:	Code snippet of the <code>addBooking</code> method.....	35
Figure 4.10:	<code>timeToUnix</code> method.....	36
Figure 4.11:	<code>validateTime</code> method.....	37
Figure 4.12:	<code>validateBooking</code> method.....	38
Figure 4.13:	Code snippet of <code>scheduleBooking</code> method.....	40
Figure 4.14:	<i>While Loop</i> for booking.....	40
Figure 4.15:	Restrictions on the queuing approach.....	42
Figure 4.16:	Code snippet of the queuing form PHP script.....	43
Figure 4.17:	<code>addQueue</code> method.....	43
Figure 4.18:	PHP script handling countdown timer.....	44
Figure 4.19:	<i>While Loop</i> for queuing approach.....	45
Figure 4.20:	<code>trimArray</code> method.....	45
Figure 4.21:	<code>QueueOptimization</code> method.....	46
Figure 4.22:	<code>getEstimatedPosition</code> method.....	46
Figure 4.23:	Estimated waiting time PHP script.....	47
Figure 4.24:	<code>getEstimatedWaitingTime</code> method.....	47
Figure 4.25:	<code>secondsToTime</code> method.....	48
Figure 6.1:	Response time and Waiting time vs number of concurrent users – (booking).....	54
Figure 6.2:	Throughput vs number of concurrent users – (booking).....	55
Figure 6.3:	Response time and Waiting time vs number of concurrent users – (queuing).....	56
Figure 6.4:	Throughput vs number of concurrent users – (queuing).....	57

Figure 6.5:	User-friendliness of the system – (queuing).....	63
Figure A.1:	Importing the database.....	72
Figure A.2:	Screenshot the database.....	72
Figure A.3:	Database parameters.....	73
Figure B.1:	Logging into the system.....	74
Figure B.2:	Email verification form.....	75
Figure B.3:	Reset password email.....	75
Figure B.4:	Reset password form.....	76
Figure B.5:	Customer sidebar menu.....	76
Figure B.6:	The booking form for customers.....	77
Figure B.7:	Booking notification email.....	78
Figure B.8:	List of available timeslots.....	78
Figure B.9:	Manage booking table – (customer).....	79
Figure B.10:	Queue form for customers.....	79
Figure B.11:	Demo page for customers.....	80
Figure B.12:	Waiting list page for customers.....	80
Figure B.13:	Customer’s account update form.....	81
Figure B.14:	Change password form.....	81
Figure B.15:	Administrator’s sidebar menu.....	82
Figure B.16:	Administrator’s dashboard.....	82

Figure B.17: Administrator’s booking form.....	83
Figure B.18: Manage booking table – (Administrator).....	84
Figure B.19: Manage waiting list table.....	84
Figure B.20: Manage user account table.....	85
Figure B.21: Account update form – (Administrator).....	85
Figure C.1: Use case diagram of the booking system.....	87
Figure D.1: Class diagram of the booking system.....	88
Figure H.1: User evaluation questionnaire.....	94
Figure H.2: User evaluation questionnaire (contd.).....	95

LIST OF TABLES

Table 5.1:	Result of testing <code>timeToUnix</code> method.....	50
Table 5.2:	Result of testing <code>scheduleBookingMethod</code>	50
Table 5.3:	Result of testing <code>addBooking</code> method.....	50
Table 5.4:	Result of testing <code>addQueue</code> method.....	51
Table 5.5:	Result of testing <code>cancelBooking</code> method.....	51
Table 6.1:	Responses to the System Usability Scale questionnaire.....	62
Table B.1:	Login details for the different types of users.....	74
Table E.1:	Result of the load test on the booking algorithm.....	89
Table E.2:	Result of the load test on the queuing algorithm.....	90
Table I.1:	Attributes of the user table.....	96
Table I.2:	Attributes of the product category table.....	96
Table I.3:	Attributes of the products table.....	97
Table I.4:	Attributes of the profile table.....	97
Table I.5:	Attributes of the password reset table.....	98
Table I.4:	Attributes of the queue table.....	98
Table I.7	Attributes of the booking table.....	99

Chapter 1

Introduction

1.1 Research Context

Typically, booking systems must have the functionality to cope with requests made by users to access certain resources within a specified timeslot. In order to effectively coordinate access to these resources for optimum resource utilisation, there is need to implement scheduling (Pinedo, 2012).

In essence, scheduling involves solving the problem of finding optimal methods to allocate some resources to certain entities (Lombardi and Milano, 2012). The problem of resource scheduling is very common, extending across different contexts such as communication bandwidth allocation, parallel processing, transportation, timetabling, appointment scheduling, project management, production scheduling, amongst others (Coley *et al*, 2012). In these contexts, there is always need to allocate different resources in time to entities (such as people, machines, etc.) that need to use them in a way that optimises the overall usage of the resources.

To address the problem of scheduling, a number of researchers such as Orduña and García-Zubia, (2011); Lowe and Orou, (2012) have investigated different approaches, notably booking and queuing to schedule access to limited resources in the context of online booking systems. However, while these scheduling approaches and their various extensions have been implemented, there has been no justification for the most suitable approach to scheduling.

In view of this research gap, this project focuses on evaluating the different approaches to implement scheduling in an online booking system and how these approaches can be effectively integrated to significantly improve the utilisation level of resources and decrease the waiting time of users. The project is made up of two main components: namely the software development component which describes the requirements specification and development tools used to implement a web-based booking system that allow customers to book demonstration cameras for a particular timeslot; and the last component, which involves evaluating the performance of the approaches used to implement scheduling in the booking system in order to determine the most suitable approach to scheduling.

1.2 Problem Description

Digital Barriers uses a number of demonstration video streams to demonstrate the technologies of its Tactical Visual Intelligence (TVI) cameras to potential customers. In order to assess the capabilities of the TVI cameras, customers need to book a timeslot so that they can take control of the PTZ (Pan, Tilt and Zoom) functionality of their preferred TVI camera within a specified timeslot.

However, as the current customer video demonstration capabilities are saturated, customers are required to compete against each other to gain exclusive access to these TVI cameras on an ad-hoc basis. This has caused problems during customer demonstrations where multiple people have tried to access the same camera and most importantly, it raises the question about how the usage of the cameras can be further optimised.

To resolve these issues, this project aims to develop an experimental web-based booking system that would allow potential customers to book one of the TVI cameras for a particular timeslot. During that timeslot, the system would ensure that each customer have exclusive access to the TVI camera.

1.3 Research Questions

This research aims to answer the following research questions:

1. What practical approaches can be used for scheduling and how can these approaches be applied in the context of an online booking system?
2. What functional requirements are suitable to demonstrate scheduling in a web-based booking system?
3. How might these requirements be developed to provide a technical solution that will allow customers book demonstration cameras for a particular timeslot.

1.4 Research Objectives

The objectives of carrying out this research are:

1. To understand the practical approaches that can be used for scheduling and how these approaches can be applied and integrated in the context of an online booking system.
2. To implement these practical approaches in a web-based booking system, identify their strengths and weaknesses and consequently, evaluate their performances to determine the most suitable approach to scheduling.

1.5 Structure of Dissertation

The rest of this dissertation is made up of six chapters, each with a number of sub-sections arranged in a logical and concise manner.

Chapter 2 provides a review of the research done so far with regards to scheduling in the context of online booking systems, as well as the current methodologies and conflicting evidence. It pulls together significant developments in this area and identifies major research themes.

In Chapter 3, the core components of the system architecture, including a high-level overview of the Graphical User Interface design and the overall database design were presented.

Chapter 4 describes the detailed design of the booking system and the decisions that were made during the software development process based on the relative merits and demerits of the alternatives considered.

Chapter 5 presents an explanation of the testing strategies that were used to validate the booking system.

Chapter 6 presents the outcomes of the performance and user evaluation of the booking system. It also explains the performance improvements that could be achieved by using either booking or queuing for scheduling.

Chapter 7 concludes the report with a set of observations made during the system development and evaluation stages. It also include suggestions for future work.

Chapter 2

Literature Review

This chapter presents a review of existing research works in resource scheduling. It begins by describing the practical approaches to scheduling in online booking systems and how these approaches can be integrated to improve the utilisation of resources and decrease the waiting times of customers. Afterward, an explanation of the types of scheduling policies was presented and finally, a list of metrics commonly used in evaluating the performance of the scheduling schemes were highlighted.

2.1 Scheduling

Traditionally, scheduling has always been used where resource demand and resource availability is predictable and constrained and consequently, there is need to allocate resources to users in an optimal way (Pinedo, 2012). However, with online booking systems, both the resource demand (that is, request for exclusive access to a resource) and the availability of the resource are typically less predictable (Coley *et al*, 2012).

In these scenarios, while resource demand will always vary due to the fluctuations arising from when users desire exclusive access to a resource and for how long that access is required, resource availability varies due to both fluctuations in how long a resource will be used, as well as other issues that might remove a resource from service (Lowe, 2012). Consequently, this inherent variation with resource demand and availability in the context of online booking systems has led researchers to explore different approaches to scheduling and what can be implemented practically to improve the utilisation of resources and decrease the waiting times experienced by users (Lowe, 2013).

2.2 Practical Approaches to Scheduling

According to Lowe, (2012); Maiti *et al*, (2013), there are two fundamental approaches, notably booking (otherwise known as calendar-booking) and queuing that can be used for resource scheduling in an online booking system. In essence, given a collection of user requests to access a certain resource, the main reason for implementing these schemes is to allow the system's scheduling server process each user's request to access specific resources, subject to a scheduling policy, in such a way that optimises the allocation and overall usage of each resource (Pinedo, 2012).

As highlighted by Orduña and García-Zubia, (2011); and Lowe, (2013), the choice of implementing either queuing or booking or a combination of both scheduling schemes in an online booking system depends on a number of factors, namely: the scheduling algorithm, the number of expected concurrent users; the number of available resources; the time each user needs to access the resource requested; and the access guarantees that is provided to users.

In certain situations where there are few users waiting to access a resource simultaneously and the time needed by a user to utilise that resource if granted access is relatively short, a booking scheme becomes unsuitable. Consequently, it becomes necessary to use a variant of the queuing scheme called priority queuing.

With this priority queue, user requests are assigned priority levels (such as high, medium, or low) as they join the queue and a request with the highest priority is served before requests with lower priorities. In cases where two requests have the same priority, they are served according to their order of arrival in the queue.

2.2.1 Booking Scheme

In a booking scheme, users are able to select in advance a timeslot in which they would like to access the resource and subsequently make a reservation that will provide guaranteed access to that resource. They return to utilise the resource at the time specified on their reservation and are given exclusive access to the resource requested (Orduña and García-Zubia, 2011).

Typically, booking schemes are more commonly used where the available resources are limited and the demand from users is relatively high; thus, having a confirmed timeslot to utilise an already reserved resource is desirable (Lowe, 2012).

As highlighted by Lowe and Orou, (2012), the basic logic for resource allocation based on the implementation of a booking scheme occur as follows:

1. A user attempts to log into the system and is provided with a list of resources that can be accessed.
2. The user selects a particular resource from this list and then is presented with the current status of the resource and given the option to make a reservation that will provide guaranteed access

to the resource based on the time specified by the user, if the resource is currently not in use by another user.

3. If the user decides to reserve the resource, a reservation web page will appear with the available timeslots that can be booked. This page will also show the maximum allowed time a user can utilise the selected resource.
4. Once the user selects a preferred timeslot to access the resource, the system verifies that the particular timeslot has not been taken by another user before it stores the reservation in the scheduling database. If the timeslot is available, the system prompts the user to make the reservation. Thereafter, a confirmation message appears, and an automated email message generated by the system is sent to the user with the resource reservation details.
5. The resource reservation page will then make a coloured highlighting of all timeslots that has been reserved by users to access each resource.

In general, the main disadvantage of scheduling resources using a booking scheme is that users will typically reserve a session to utilise a resource but will very commonly not make use of the whole session. The unused component of the reservation made is then left unutilised, thereby resulting in under-utilisation of resources (Lombardi and Milano, 2012).

2.2.2 Queuing Scheme

Queuing schemes, on the other hand are used when there is a relatively significant number of resources and users can only be allocated just one of them, at any point in time. More specifically, Orduña and García-Zubia, (2011) highlighted that by using the queuing scheme to schedule access to resources, all requests made by users to access a particular resource have to be placed sequentially in a queue in the order that they arrived and then processed successively one after another on a first-come, first-served basis. On the whole, users are only provided access to the first available resource that meets their specific request (Lowe, 2013).

According to Lowe and Orou, (2012), resource scheduling and allocation to users based on the queuing scheme occur as follows:

1. A user logs into a reservation system and is provided with the list of resources they have permission to access.

2. The user selects a particular resource from the list of available resources and then is presented with the current status of the resource and given the option to queue for access, if the resource is currently in use by another user.
3. If the resource is currently in use and the user decides to queue to access the resource, the request is placed in the queue by the system's scheduling server. Consequently, the user is presented with a screen that contains information indicating which resource they have requested access to, an estimate of how long they have to wait in the queue before they can access the resource, their current position in the queue, as well as the maximum time allowed for them to access this resource when it becomes free.
4. When any resource becomes available from the 'pool of resources', the system's scheduling server will 'scan' through the queue looking for the next user's request made to access the resource based on a predefined scheduling policy such as the First-Come-First-Served policy. The system will also make sure that the time requested by the user to access the resource will not affect the next booking made for this resource by another user, if applicable. This allocation strategy according to Lowe, (2012) ensures that a resource will only be allocated to a user in the queue when there is a guarantee that they will have exclusive access to the resource based on the duration they have specified.
5. Once the system finds a request in the queue that conforms to the conditions specified in (4), the user that made such request is granted access to utilise the resource.

One significant advantage of implementing the queuing scheme in an online booking system, as identified by Lowe, (2013) is that as soon as a resource becomes available, it will be allocated to a waiting user in the system, thereby maximising resource utilisation, as compared to the booking scheme. However, there is no guarantee that users will have access to the resources desired at the specified time (Lowe, 2013). As a matter of fact, since the time each user will spend on the queue to access a particular resource cannot be estimated accurately, there is no assurance that a user will eventually get access to a resource, even after spending significant time on the waiting list. In addition to this, other problems associated with the queuing scheme, as highlighted by Maiti, (2011) include:

1. Loss of priority in the queue due to a slow internet connection.
2. The queuing scheme is meant to work on a "first-come-first-served" basis. Consequently, there is need for the system's scheduling server to keep track and assign a priority level to the user

request that arrived first in the queue, as well as other requests that arrived later. However, assigning and maintaining such priorities could be sometimes inefficient, especially when the internet connection is slow and noisy.

2.2.3 Priority Queue

In priority queuing, user requests are assigned a priority as they join the queue and requests with higher priority are processed before those with lower priority (Stanford, *et al* 2013).

Earlier works on queuing by Li *et al*, (2008) discusses a variant of queuing called priority queuing which processes user requests for immediate access to a specified resource based on the user priority level, with the request from users in the higher priority category being processed first.

This form of queuing was described and illustrated by the authors with the design and pilot implementation of a scheduling system for shared online laboratory resources which allow users to make reservations to conduct experiments within a timeslot. The scheduling system also included a functionality to coordinate the reservations made by geographically dispersed users in a way that avoids time conflicts caused by differences in time zones between the users. Although the system provides integrated functionality for handling various priority levels in user requests, the system was not flexible enough to accommodate scenarios wherein a user fails to utilise the reserved timeslot.

Similarly, the functionality of priority queuing was further analysed in a pilot study carried out by Stanford, *et al* (2013). This study was based on the underlying assumptions that classes of users have fixed priorities, and that no user from a low-priority class should be granted exclusive access to a resource while there are requests to access that particular resource from users with higher priority in the queue. From the works of Stanford *et al* (2013), priority queuing was presented as a significant improvement to the basic queuing model.

2.2.3.1 Types of Priority Queue

In essence, there are two different types of priority queue: the non-preemptive priority queue and the preemptive priority queue (Kihlwan, 2012). The non-preemptive and preemptive priority queues are both extreme cases with respect to the preemption condition, as they “never” or “always” allow preemption, respectively when a request to access a resource arrives in the queue from a user with a higher priority during the service of the request from a lower priority user (Kihlwan, 2012).

More specifically, in the case of the preemptive priority queue, during the service of a user's request, the scheduling server "continuously" checks the queue for the presence of requests to access a resource from users with higher priorities than the current user being serviced, to determine whether or not to preempt the service of this user. If there exists a request from a user with higher priority in the queue, the low priority request currently being processed will be preempted (Harchol-Balter, 2013). In many practical situations, however, it may be counter-productive and undesirable to continuously review the system state, as this can incur considerable processing time, especially when the information on the queue is decentralised and distributed over a network (Kihlwan, 2012).

On the other hand, with non-preemptive priority queues, the scheduling server "never" reviews the system state while processing a user's request to access a particular resource. This implies that users with high priorities will have to wait for some time until the service of a low priority user is completed. As highlighted by Harchol-Balter (2013), this can lead to severe degradation of the quality of service experienced by high-priority users, especially when the processing time of a low-priority user request is relatively slow.

2.2.4 Combination of Booking and Queuing

For optimum utilisation of resources, an efficient scheduling scheme is essential. However, current approaches to scheduling are typically based on either booking or queuing (Orduña and Garcia-Zubia, 2011), though rarely both and each of these have advantages and disadvantages. In the light of this, Lowe, (2013) emphasised that a scheme that merges these two approaches successfully can potentially benefit from the advantages of both whilst compensating for their respective limitations.

By taking into account the benefits and shortcomings of the aforementioned approaches to resource scheduling, several researchers have also explored the potentials of a hybrid approach that attempts to integrate booking and queuing in an online booking system. One of the first to consider this approach was Li, *et al* (2008), who discussed the functionality of a system that allow users to book resources ahead of time, whilst still permitting queued users to have exclusive access to resources during periods when there were no bookings. From a practical perspective, this approach seemed to ameliorate the limitations of booking and queuing respectively, however, there was no suitable description of the implementation of the system combining both booking and queuing neither was there any attempt made to evaluate the resulting system.

Similarly, Maiti (2011) described another hybrid approach that was referred to as “slotted queuing” which allowed a limited number of users to reserve a particular resource for the same timeslot, after which the users will have to queue to utilise the resource one after the other within the reserved timeslot. Again, a prototype implementation was described but there was no evaluation of the effect of combining the two approaches on the system performance, neither was there an investigation into the issues that may arise when a single resource is accessed by users through both approaches.

More recently, Lowe and Orou, (2012) implemented a hybrid approach similar to Li, *et al* (2008) and Maiti (2011) in the context of managing access to remote laboratory resources. This time the implications of merging booking and queuing to schedule resources were extensively discussed. In essence, after a detailed analysis was carried out to assess the extent to which the booking and queuing algorithms interacted when used for resource scheduling, it was found that there would often be significant periods of time a number of users were waiting to gain exclusive access to a resource, but the resources that were not currently in use at that particular moment were not allocated by the system because there was a pending booking made by another user that has failed to show up. In order to resolve this issue, Lowe, (2013) recommended some techniques that can be implemented to manage resource utilisation effectively, namely:

1. If a user does not utilise a resource at the commencement of a reserved session, the reservation should be immediately cancelled, leaving the entire timeslot available to a different user who might want to use the resource.
2. The implementation of an allocation strategy that will allow the system’s scheduling server to dynamically allocate a timeslot shorter than the default access time to queued users in circumstances where this will optimise the utilisation of available resources. Overall, this will help to reduce the size of queues, queue waiting lists, as well as the waiting times experienced by users.

2.3 Slot Allotment Algorithm

In an online booking system, there is need to implement a slot allotment algorithm that will allocate timeslots to users and subsequently, coordinate user requests to access specific resources during that timeslot (Maiti and Tripathy, 2011).

Essentially, the slot allotment algorithm is designed to be interactive by giving the user a full picture of the booking status of a particular resource for a couple of days (say n). To accomplish this, it uses a calendar object to calculate all the days from the next day to the n th day, where n is an integer value specified by the administrator. This calendar object is displayed in a tabular layout where the headers represent the days and the rows represent the timeslots available in each day. The duration of these timeslots is decided and set by the system administrator.

Inside each slot, the user requests are maintained by priority queuing and by keeping the maximum number of reserved sessions in a slot limited to a small number, the queuing technique become more effective in managing resource utilisation.

The system finds out if any booking has been made in each timeslot and then, it calculates the total number of bookings. It is only when the number of bookings is not equal to the maximum number allowed that a timeslot is made available for booking.

2.4 Scheduling Policies

In order to schedule resources in a way that optimises their overall usage, the scheduling server in an online booking system must be implemented to follow a certain scheduling policy (Harchol-Balter, 2013).

Accordingly, these scheduling policies can be categorised based on whether the policy is preemptive or non-preemptive. A policy is *preemptive* if the processing of a user's request to access a resource can be stopped partway through its execution. On the other hand, a scheduling policy is considered *non-preemptive* if a user's request is always processed completely. Moreover, scheduling policies can be differentiated further based on whether the policies assume knowledge of the duration specified by a user to utilise a particular resource.

2.4.1 Non-preemptive Scheduling Policies

As highlighted by Harchol-Balter (2013), there are two types of non-preemptive scheduling policies, namely:

1. Non-preemptive scheduling policies that do not make use of the duration specified by a user to utilise a resource. Examples include:

- a. First-Come-First-Served (FCFS): When the scheduling server becomes free, it chooses to process the user request at the head of the queue. Although, this is a relatively simple policy to implement (Harchol-Balter, 2013), waiting time depends on the arrival order of requests made by users in the queue, and this can potentially increase the waiting times experienced by users with requests that arrived much later.
 - b. Non-Preemptive Last-Come-First-Served (NLCFS): Every time the scheduling server becomes free, it chooses the last user request that arrives in the queue and processes that request. This implies that the user with the shortest waiting time on the queue is served first.
 - c. RANDOM: When the server frees up, it chooses a random user request in the queue and processes it.
2. Non-preemptive scheduling policies that makes use of the duration specified by a user to utilise a resource. An example of this is the Shortest-Job-First (SJF) policy. With the Shortest-Job-First (SJF) policy, whenever the scheduling server is free, it chooses to process the user's request with the smallest duration. By so doing, it achieves the best response time (Harchol-Balter, 2013).

2.4.2 Preemptive Scheduling Policies

According to Harchol-Balter (2013), there are two types of preemptive scheduling policies, namely:

1. Preemptive scheduling policies that do not make use of the duration specified by a user to utilise a resource. A typical example is the Preemptive-Last-Come-First-Served (PLCFS) scheduling policy.

In essence, with the Preemptive-Last-Come-First-Served policy, whenever a new user request to access a resource enters the queue, it immediately pre-empts the request currently being processed by the scheduling server. It is only when the new request has been completely processed that the scheduling server resumes the processing of the preempted request. Therefore, with this scheduling policy, each user request creates two preemptions – one when it arrives on the queue and one when it has been completed processed by the scheduling server.
2. Preemptive policies that makes use of the duration specified by a user to utilise a resource. An example is the Preemptive-Shortest-Job-First (PSJF) scheduling policy.

The Preemptive-Shortest-Job-First policy operates similarly to the Shortest-Job-First policy, except that the size-based priorities are enforced pre-emptively (Harchol-Balter, 2013). Thus, at any moment in time, the user request that is currently being processed by the scheduling server is the request that has been specified by the user to utilise the resource if granted, within a short period of time. A preemption only occurs when a new request arrives whose specified usage time is smaller compared of the request being processed. In general, the Preemptive-Shortest-Job-First scheduling policy minimises the average waiting time that users might experience in the queue.

2.5 Performance Evaluation of Scheduling Schemes

According to Wei, *et al* (2012), the following metrics are commonly used in evaluating the performance of the approaches used for scheduling, namely:

1. Waiting time. This refers to the time that elapses between when a user submits a request to access a resource and when that request is processed by the scheduling server. Typically, the average waiting time (usually in minutes) among all processed user requests is usually measured to reflect the “efficiency” of the approach used for scheduling.
2. Server utilisation rate. This represents the ratio of the utilised scheduling server CPU processing time compared with the total CPU processing time available. Usually, this metric refers to an average value over a specified period of time.
3. Response Time: The time it takes a request to fully load. From the time the request is initiated until the time it is complete. This measure represents the average response time at a certain minute of the test and it generally indicates the performance level of the entire system under test (web server + DB).

By and large, despite the emerging research on scheduling schemes, and the growing recognition of the importance of effective resource scheduling for the optimum utilisation of resources, there has not been a detailed analysis of how the different practical approaches to scheduling discussed in previous sections can improve user experiences and resource usage. To this end, there is need to further investigate how

queue-based and booking-based approaches to scheduling can be effectively integrated to improve and effectively manage resource utilisation in the context of online booking systems.

In order to accomplish this, the next chapter goes on to describe the functional requirements and core components of the system architecture, as well as a high level overview of the overall database design of the booking system that will be developed to assess the performances of booking and queuing and the benefits and issues that might arise when they are integrated in a single system.

Chapter 3

System Design

This section describes the functional requirements of the booking system, the system architecture, the database design, as well as the methodology that has been used for the software development. It also highlights the technologies that have been used in developing the application, with an explanation of why they are appropriate.

3.1 Functional Requirements

Basically, the functional requirements is made up of the functions of the booking system. The short list of functional requirements specified by Tony Jasnosz of Digital Barriers was reviewed and after careful consideration of what can be realistically achieved within the time allocated to the project, the requirements were refined. In general, the booking system would:

1. Allow users create an account and subsequently modify the account information, if required.
2. Allow a customer to book a surveillance camera for a 30-minute or 60-minute demonstration between 9:00 AM and 5:00 PM.
3. Allow customers to amend and cancel existing bookings.
4. Allow customers to view a 30-minute or 60-minute demonstration between 9:00 AM and 5:00 PM.
5. Allow an administrator to book a camera on behalf of a customer. This booking would be recorded automatically in the customer's booking history.
6. Verify the email address of customers who have just created a new account.
7. List available timeslots in a day so that a customer can choose another timeslot, if the most preferred timeslot has been booked by another customer.
8. Send booking confirmation emails to customers after each booking. These emails will include the booking reference, the name of the customer, the reserved timeslot (e.g. 10:30 AM – 11:00 AM), the date, as well as the name of the camera that has been booked.
9. Manage bookings made for a particular camera using priority queuing, in a way that seeks to optimise the overall usage of the camera and also act fairly to the customers.

The use case diagram (shown in Appendix C) illustrates the functional requirements of the booking system and the users (notably the administrator and customers) that will interact with it.

3.2 Non-Functional Requirements

The non-functional requirements highlight the characteristics of the booking system and these include:

1. Authentication of users (customers and the system administrator) at log in.
2. Implementation of a simple and intuitive user interface.
3. Scalability, as the booking system is experimental and therefore has been designed in a way that would allow for changes to be incorporated easily.

3.3 Business Logic of the Booking Functionality

For a customer to make a booking, the customer have to first log into the system. Upon login, the customer selects a camera, chooses a particular date and time he/she wants to access the camera, as well as how long access to the camera is needed. Thereafter, the customer submits the booking.

Once the customer submits the booking, the system checks if the timeslot (that is, the date and time) selected by the customer is available. In the context of this booking system, a timeslot is available when it has not been booked by another customer. Moreover, if the timeslot is available, the system saves the booking and sends an automated booking confirmation email containing the booking details to the customer. This automated booking confirmation email will include the booking reference, the name of the customer, the date of the demonstration, the start and end times of the demonstration, the booking duration (which could be for 30 or 60 minutes), as well as the name and model of the camera that has been booked by the customer.

On the other hand, if the timeslot selected by the customer is unavailable, the system will prompt the customer to select another timeslot. If the customer agrees to choose another timeslot, the system would generate a list of all reserved and available timeslots for the rest of that day. This is necessary to allow the customer to make an informed decision regarding the best available time he/she would like to have access to the camera. Once the customer selects the timeslot, the system saves the booking details in the database and sends the customer an email confirming that the booking has been made.

3.4 Business Logic of the Queuing Functionality

In order to access a camera using the queuing approach, a customer will have to first log into the system. Upon login, the customer selects a camera and then specifies how long he/she would like to access the camera. This duration can either be for 30 or 60 minutes, respectively.

Once the customer submits the request, the system first checks if the timeslot selected by the customer is available and then, it checks if the timeslot would not conflict with a timeslot that has been booked, as a customer can only be granted access to a particular camera for the duration specified if there would not be a conflict with already reserved timeslots. If these conditions evaluate to true, the system saves the details and starts a countdown timer. This countdown timer has been implemented as an alternative to a demonstration video stream due to some limitations that would be explained in the next chapter.

Moreover, if the timeslot selected by the customer has been booked by another customer, the system will prompt the customer to join a queue. If the customer decides to join the queue, the system will automatically select the nearest available timeslot for the customer from the list of all timeslots available for that day. In addition, the system will display how long the customer would have to wait on the queue to access the camera and the number of customers on the queue.

As soon as the camera selected by the customer becomes available, the system will 'scan' through the queue looking for the next customer that has requested to access that camera, using a First-Come-First-Served scheduling policy. Once the system finds that customer, the customer is granted access to utilise the camera for the specified duration.

3.5 Class Diagram

The booking system is made up of four main classes namely, the user class, the product class, the booking class and lastly, the queue class. While the user class handles basic system functionalities such as the validation of email addresses, user login, password reset, and sending of notification emails etc., the booking and queue classes handle camera reservations made by each customer, respectively. All the product class does is to list the available cameras in the booking form.

The class diagram (shown in Appendix D) depicts all these classes, as well as the relationship that exist between each of them. Each class in the class diagram is drawn using three-part rectangles with the class's name at the top, attributes in the middle, and methods at the bottom. The attributes of a class and their values define the state of each object that is created from the class, while the methods specify how an object acts and reacts, in terms of state changes and interactions with other objects in the booking system.

The class diagram also contains additional information regarding the visibility (that is, information hiding) of attributes and methods in the booking system. More specifically, private attributes (indicated with a -) are hidden from other classes while public methods (indicated with a +) are not hidden from other classes in the Booking system.

When two or more classes share a relationship, a line is drawn between each class and labelled with the name of the relationship. For example, the Customer and Booking classes are associated with one another whenever a customer makes a booking. Thus, a line labelled “makes” connects the Customer and Booking class, representing exactly how the two classes are related to each other.

In addition, these relationships have multiplicity, which indicates how an instance of a class can be associated with the instances of other classes. Numbers are placed on the association path to denote the minimum and maximum instances that can be related through the association in the format “minimum number . . maximum number”. Moreover, the concept of Inheritance is shown with a solid line and a hollow arrow pointing at the User superclass from the Customer and Administrator subclasses.

3.6 Development Approach

The Iterative and Incremental software development model has been used as the methodology to guide the process of developing the booking system.

Essentially, Incremental development is an approach to agile software development, where various parts of the system are developed in successive increments at different times, and then integrated as they are completed to make up a complete, stable and robust system (Cockburn, 2008).

With Iterative development, the software is developed incrementally through several iterations which subsequently provides the opportunity to make necessary improvements on the system based on key issues identified in previous deliverables (Cockburn, 2008). As highlighted by Sommerville, (2010), while neither approach precedes, requires or implies the other, it is always best to integrate both.

In the light of this, it is important to emphasise that the Iterative and Incremental software development model has been selected instead of the standard waterfall model because:

1. The processes of requirements specification, system design, programming, and testing are interleaved, compared to the waterfall model where each phase has to be completed before the next phase.
2. Iterative and Incremental software development model offers the flexibility to incorporate changes into the software requirements specification and the software functionality during the development cycle, if necessary. This is not possible with the waterfall model since the functional requirements have to be well understood and unlikely to change radically during system development.
3. There is great support for code refactoring.
4. Continuous testing and refinement of the evolving system at the end of each iteration is easier.
5. Issues identified from each increment can be resolved during the next iteration.

3.7 System Architecture

With regards to the fact that the booking system is a web application involving a client-server architecture and requiring user interaction through a series of Graphical User Interfaces, the Model-View-Controller architectural pattern was chosen and implemented as the system architecture.

The Model-View-Controller pattern is made up of the following components:

1. The View is the presentation layer. It is made of web pages that display information related to the services available on the booking system. In addition, it allows data entry and manipulation by the user.
2. The Controller coordinates the booking system's functionality via the booking and queuing algorithms implemented with PHP. It also handles event handling and transfers data entered by each user between the View and Model.
3. The Model is made up of the MySQL database where information is stored and retrieved. This information is then passed back to the Controller for processing, and later on to the user through the View.

The schematic diagram of the booking system's architecture is shown below:

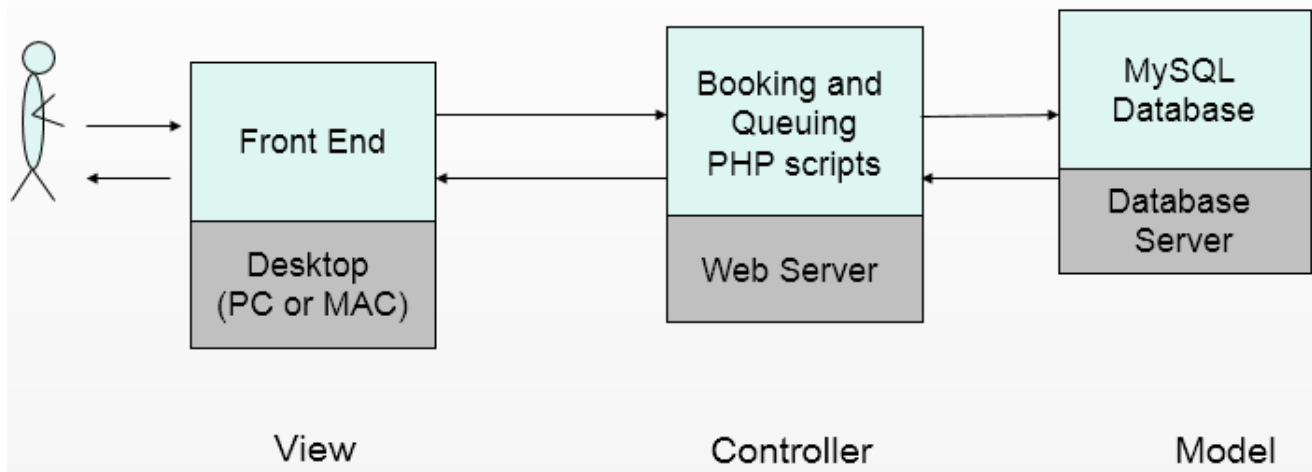


Figure 3.1: The booking system's architecture

In general, the decision to implement the Model-View-Controller architecture for the booking system made because it allows separation of concerns between the data and code, which consequently provides benefits such as data integrity, reusability and maintainability. In addition, it makes it relatively easier for any of the core modules of the booking system to be upgraded or replaced independently as the system's functional requirements changes.

3.8 System Development

The following technologies have been used to develop the booking system:

3.8.1 HyperText Markup Language (HTML) 5

This has been used to define the structure of the web pages. Essentially, HTML 5 was used because it includes new markup and semantics that supports intelligent web forms and further enhances the accessibility of web applications.

3.8.2 Cascading Style Sheet (CSS) 3

This is the style sheet language that has been used to describe the style, design, and presentation of the web pages.

CSS 3 is split into logical and user-friendly modules, with the most important CSS 3 modules being backgrounds and borders, text effects, animations, and layouts. These modules contain new features and capabilities, as well as a set of customisable functions that allows for greater flexibility and control in the specification of presentation semantics in the booking system.

In general, CSS 3 was selected because of the styling effects it generates in web browsers which further enhances the content accessibility of web pages, whilst reducing the complexity and repetition in formatting content. Consequently, this means reduced server requests and load times for the Booking system.

3.8.3 jQuery

This cross-platform JavaScript library allows client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the web content that is displayed. Its syntax is designed to make it easier to manipulate Document Object Model (DOM) elements, create animations, and handle events.

jQuery has been used in this booking system to implement CSS manipulations, AJAX calls, user input validation and the countdown timer.

3.8.4 Hypertext Preprocessor (PHP)

In the booking system, object-oriented PHP has been used as the server-side scripting language to generate dynamic and interactive web pages, execute the scheduling algorithms to process each customer's bookings, commits queries to the database, amongst others.

Essentially, PHP was chosen over ASP.NET because it has built-in support for the MySQL[®] Relational Database Management System (RDBMS).

3.8.5 MySQL

In essence, MySQL database has been used to store user accounts and the bookings made by customers, amongst others. The reasons for choosing MySQL over other Relational Database Management Systems such as PostgreSQL include:

- The database administration be handled by phpMyAdmin.
- MySQL is easy to configure.
- It includes multiple storage engines, allowing one to choose the one that is most effective (e.g. InnoDB) for each table in the booking system.
- It supports the `mysqli` extension, which is a relational database driver used in the PHP programming language to provide an interface with MySQL databases. This allows for object-oriented programming and the use of prepared statements.

3.9 Database Design

The database stores all the relevant information such as user accounts, customer bookings, and product information amongst others. It is made up of the following tables:

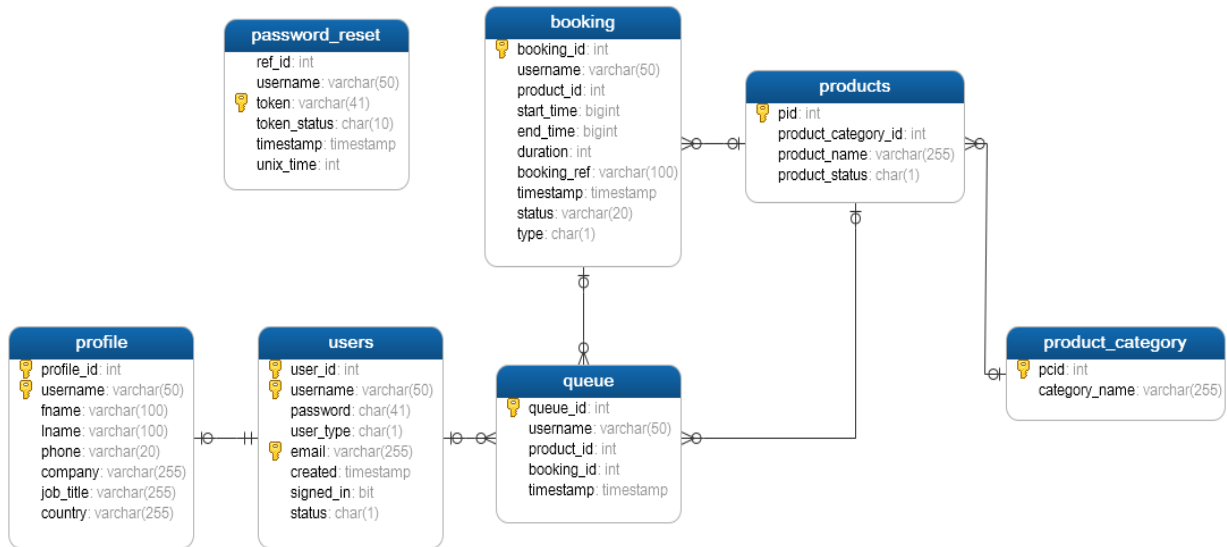


Figure 3.2: Overview of the database design

The purpose of each table in the database is given as follows:

1. **Users Table**: This table keeps record of the basic information needed by each user (either a customer or an administrator) with an existing account to log into the booking system.
2. **Profile Table**: This table keeps record of all other information provided by users during registration.
3. **Products Table**: This table stores information about the cameras and hardware encoders available for booking.
4. **Products Category Table**: This table stores basic information about the category that each camera or hardware encoder belongs to.
5. **Booking Table**: This table stores the details of all bookings made by customers.
6. **Queue Table**: This table keeps record of the priority queue that is created dynamically once a customer makes a booking.
7. **Password Reset Table**: This table stores the relevant information needed by customers to reset their password.

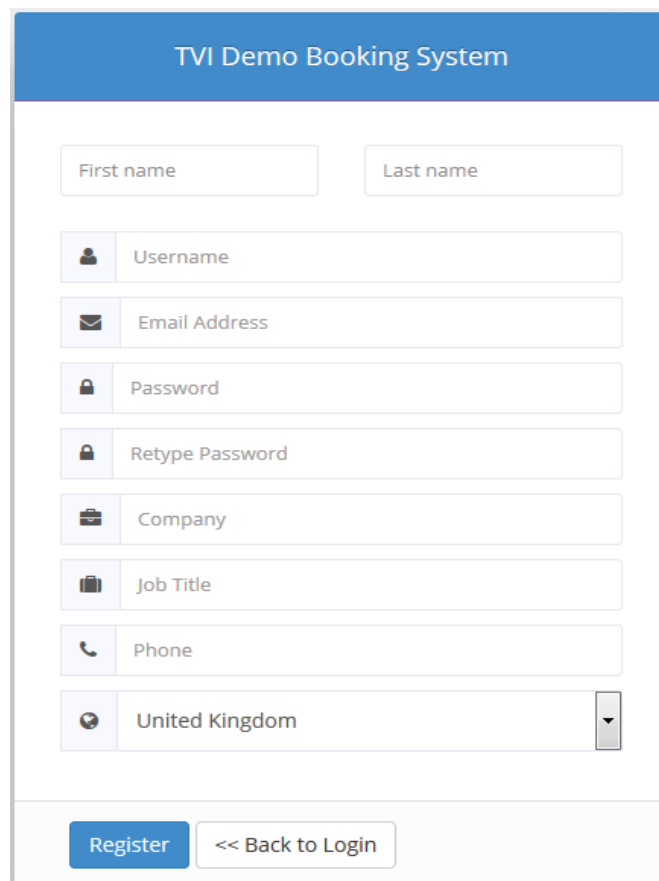
A more detailed explanation of the function of each attribute in the database is available in Appendix I.

3.10 User Interface Design

Bootstrap has been used to design the User Interface of the booking system. This framework was selected because it contains a set of style sheets that provides basic style definitions for all key HTML components. In addition, it includes several jQuery plugins which provides additional User Interface elements such as dialog boxes, tooltips, and date pickers, amongst others. All these features combined helps to improve the usability of the booking system.

Moreover, as the system is meant to be used by the administrator and customers, the design of the User Interface has been divided into two, to cater for the needs of each type of user.

In order to use the system, a user (which could either be an administrator or a customer) is required to create an account by using the registration form. See Figure 3.3 for the registration form.



The registration form is titled "TVI Demo Booking System" and contains the following fields and buttons:

- First name
- Last name
- Username
- Email Address
- Password
- Retype Password
- Company
- Job Title
- Phone
- Country (Dropdown menu showing "United Kingdom")
- Register button
- << Back to Login button

Figure 3.3: Registration form

After registration, the user proceeds to login into the system using the login form shown on the next page:

The image shows a login form for a system titled "TVI Demo Booking System". The form is contained within a white box with a blue header. The header text is "TVI Demo Booking System". Below the header, the text "Log In" is displayed. There are two input fields: "Email Address" with a person icon and "Password" with a lock icon. A link "Forgot your password?" is located below the password field. At the bottom right, there are two buttons: "Register" and "Log In".

Figure 3.4: Login form

Once a user submits the email address and password, the PHP script that processes the log in credentials determines if the user is a customer or an administrator and then, based on the result of this evaluation, it loads the appropriate sidebar for the user in the system's home page.

By default, all new registered users are classified as customers. Therefore, in order for a user to become an administrator, the 'user_type' attribute of the user would have to be changed in the database from 'C' which stands for customer to 'A' – an Administrator. Once an administrator is set, each user can then be changed from being a customer to an administrator or vice versa in the "Manage Users" table of the administrator's section of the website.

3.10.1 Customer's User Interface

This has been designed to allow customers to make bookings, view past and current bookings, amend and cancel existing bookings, and if necessary update their account information.

In order to book a particular camera, a customer would have to fill in the booking form (shown below in Figure 3.5). This form allows a customer to select the desired camera, the date of demonstration, the preferred timeslot (e.g. 9:00 AM to 10:00 AM), as well as the duration of the demonstration – which could either be for 30 minutes or 60 minutes.

New Booking

Product Category: --Choose One--

Product: --Choose One--

Date: Select Date

Start Time: 09:00 AM

Duration: 30 Minutes

Book Now!

Figure 3.5: New booking form – (Customer)

Once a customer makes a booking, the booking is saved in the database. If at a later date, the customer wishes to amend or cancel the booking, the customer would have to use the “Manage Booking” table to effect these changes. See Figure 3.6 for the “Manage Booking” table.

Manage Booking

3 Records found

5 records per page Search:

SN	Product	Date	From	To	
1	TVI RDK-U	01-09-2014	9:00 AM	9:30 AM	Amend Cancel
2	TVI MC350 (Minicam)	01-09-2014	2:00 PM	3:00 PM	Amend Cancel
3	TVI R500 (Tri-Star)	02-09-2014	3:00 PM	3:30 PM	Amend Cancel

Figure 3.6: Manage booking table – (Customer)

Moreover, for a customer to use the queuing functionality, a customer would have to complete the queue form. See Figure 3.7 on the next page.

View Demonstration Stream

Product: --Choose One--

Duration: 30 Minutes

View Demo

Figure 3.7: Queue form

This form allows the customer to specify the camera's demonstration video stream he/she would like to access and how long this access would be required. The major differences between the queue form and the booking form is that the queue form does not allow a customer to select the date and start time of a demonstration. These restrictions have been enforced in order to allow the system optimise the usage of cameras allocated to customers. Thus, the date and start time of a demonstration would be automatically determined using the Unix timestamp – to be described in the next chapter.

If the timeslot selected by the customer is available, a countdown timer would start counting down based on the duration that has been selected by the customer. This timer has been implemented to simulate the operation of a TVI demonstration video stream, as it was not possible to connect the booking system's backend to a TVI server that contains real demonstration video streams due to limited support from Digital Barriers. See Figure 3.8 for the countdown timer.

View Demonstration Stream

Demonstration video stream of TVI WMV (Polecat):

0 29 17
Hours Minutes Seconds

Stop Demo

Figure 3.8: Demo Countdown timer

On the other hand, if the timeslot selected by the customer has already been booked, the system would prompt the customer to join the waiting list of customers expecting to access the same camera. As soon as the customer joins the queue, the system will display how long the customer would have to wait on the queue to gain access to the camera, as well as the total number of customers already on the queue. See Figure 3.9 for a screenshot of the waiting list page.

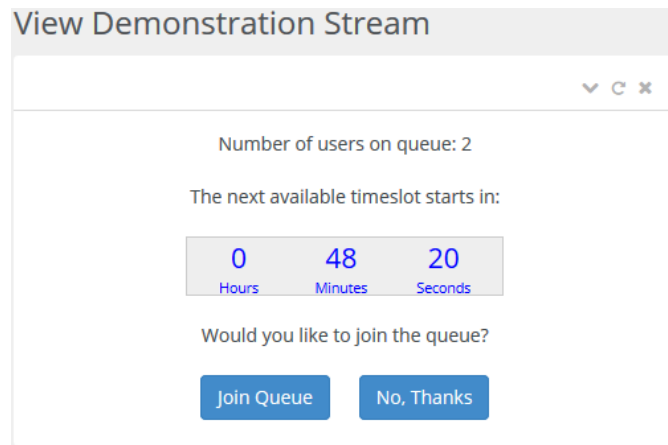


Figure 3.9: Waiting list page

See Appendix B.1 for more explanation and screenshots of the customer’s User Interface.

3.10.2 Administrator’s User Interface

The administrator’s User Interface is made up of all the features that the administrator would need to manage the booking system such as booking a camera on behalf of a particular customer, managing customer accounts, managing the waiting list for each camera, amongst others.

In order to allow an administrator to book cameras for customers, the booking form was slightly modified to include a drop-down list that shows the name of all the customers that have created an account in the system. See Figure 3.10 for the modified booking form.

New Booking

Product Category: --Choose One--

Product: --Choose One--

Select Customer: --Choose One--

Date: Select Date

Start Time: 09:00 AM

Duration: 30 Minutes

Book Now!

Figure 3.10: New booking form – (Admin)

With this modification, all the administrator has to do to make a booking for a customer is to select the name of the customer from the drop-down list and then fill in all the basic details normally requested from customers such as the product, start time, date etc. This booking would be recorded automatically in the customer’s booking history.

The administrator also have the capability to cancel the bookings that have been made by customers. This can be achieved by clicking on the “Cancel” button in the “Manage Booking” table. See Figure 3.11 for the “Manage Booking” table.

Manage Booking

3 Records found

5 records per page Search:

SN	Full Name	Product	Date	From	To	
1	Jones Lazlow	TVI RDK-U	02-09-2014	9:00 AM	10:00 AM	Cancel
2	Percival Donald	TVI WMV (Polecat)	01-09-2014	10:00 AM	10:30 AM	Cancel
3	Phillips Trevor	TVI MC350 (Minicam)	03-09-2014	11:00 AM	12:00 PM	Cancel

Figure 3.11: Manage booking table – (Admin)

Moreover, the administrator is able to update and delete customer accounts. To accomplish any of these, the administrator would have to choose the customer’s account from the list generated in the “Manage Users” table and then, click on either the “Edit” link to update the account or the “Delete” link to delete the account from the system. See Figure 3.12 for the “Manage Users” table.

List of Customers

13 Records found ▼ ☰

5 records per page Search:

EDIT	Full Name	Email Address	Phone	Company	Country	Status	Delete
EDIT	Lukens Rickie	rickie.lukens@life.invader.com	123-555-0160	Life Invader Inc	United States	Inactive	Delete
EDIT	Weston Devin	devin.weston@dwestholdings.com	346-555-0176	Devin Weston Holdings Inc	United States Minor Outlying Islands	Inactive	Delete
EDIT	Norris Jay	jay.norris@lifeinvader.com	310-555-0156	Life Invader Inc	Azerbaijan	Inactive	Delete

Figure 3.12: Manage users table.

See Appendix B.2 for more explanation and screenshots of the administrator’s User Interface.

Chapter 4

Detailed Design

This section discusses the implementation of the booking system. It includes key decisions that were made regarding the approaches used in developing the back-end of the system, some of the challenges encountered, and how these challenges were overcome.

4.1 Back-End Development

The back-end of the booking system has been developed to conform to the principles of Object-Oriented Programming instead of structured programming because of the benefits of inheritance, modularity and code reuse.

In the light of this, all the functionalities of the system were implemented separately as reusable methods in the user, booking and queue classes, respectively. The following sections contain a detailed description of each of those classes and their main methods.

4.2 User Class

The user class contains all the methods that handle the basic functionalities of the booking system. In order to access the booking system for the first time, a customer is required to create an account, as described in the System Design chapter.

Once the customer submits the registration form, a PHP script embedded within the registration form calls the `addNew()` method to process all the data supplied. See Figure 4.1 for the `addNew()` method.

```

function addNew()
{
    $this->toggleUserStatus();
    $this->password = $this->hashPassword($this->password);
    $result_validate = $this->validateEmailUsername();
    if($result_validate == ""){
        $link = dbconnect();
        $query = "insert into users (username, password, user_type, email, status)
        values ('$this->username', '$this->password', '$this->user_type', '$this->email', 'I')";

        $query_result = mysqli_query($link, $query);
        if($query_result){
            $query = mysqli_query($link, "insert into profile (username, fname, lname, phone, company, job_title, country)
            values ('$this->username', '$this->fname', '$this->lname', '$this->phone', '$this->company', '$this->job_title',
            '$this->country')");
            self::sendMail('registration');
            $result = "Successful";
        }else
        {
            $result = "Failed";
        }
        dbdisconnect($link);
        return $result;
    }
}

```

Figure 4.1: addNew method

The first step in this data processing involves setting the customer's status to 'I - Inactive' using the toggleUserStatus() method, shown below.

```

function toggleUserStatus()
{
    if($this->status == 'A') // A means Active
    {
        $this->status = 'I'; // I means Inactive
    }else
    {
        $this->status = 'A';
    }
}

```

Figure 4.2: toggleUserStatus method

Thereafter, the addNew() method calls the hashPassword() method to make an MD5 encryption of the customer's password. Although other cryptographic hash functions such as SHA-1 and SHA-256 could have been used, MD5 was deemed appropriate because it was well suited to the needs of the booking system. Moreover, since the primary objective of this research is to evaluate the efficiency of the practical approaches to scheduling in an online booking system, it was resolved that the system had to meet minimum security requirements, and consequently the fact that MD5 is susceptible to brute force attack, amongst other vulnerabilities was downplayed.

After the password has been encrypted, the `addNew()` method calls the `checkAvailability()` and `validateEmailUsername()` methods (See Figure 4.3 and 4.4) to check if the customer's username and email address does not already exist in the database. If these validation checks are passed, the `addNew()` method proceeds to save the new customer's data in the database.

```
function checkAvailability($value, $column)
{
    $link = dbconnect();
    $query = mysqli_query($link, "select * from users where $column = '$value'");
    if(($query) && (mysqli_num_rows($query) > 0))
    {
        $result = true;
    }else
    {
        $result = false;
    }
    dbdisconnect($link);
    return $result;
}
```

Figure 4.3: `checkAvailability` method

```
function validateEmailUsername()
{
    $result = "";
    $result = $this->checkAvailability($this->username, "username");
    if($result == true)
    {
        $result = "$this->username is associated with another account";
        return $result;
    }else
    {
        $result = $this->checkAvailability($this->email, "email");
        if($result == true)
        {
            $result = " $this->email is associated with another account";
            return $result;
        }
    }
    return $result;
}
```

Figure 4.4: `validateEmailUsername` method

As soon as the data is stored in the database, the `addNew()` method automatically sends an account activation email to the customer. It is important to send this email because it helps to verify the customer's email address. Once the customer clicks on the link embedded in the email, the `updateStatus()` method (shown below) updates the customer's status in the database to 'A – Active'.

```

function update_status($status)
{
    $link = dbconnect();
    $qresult = mysqli_query($link, "update users set status = '$status' where email = '{$this->email}'");
    dbdisconnect($link);
}

```

Figure 4.5: updateStatus method

When the customer tries to log in, the login page form handler first creates a new user object before calling the `xTrim()` method to sanitise the email address and password entered by the customer, to prevent cross-site scripting.

After the customer's email address and password (hereinafter referred to as login credentials) have been sanitised, the `login()` method establishes a connection with the database in order to verify that the customer's login credentials exist in the users table of the database. If the customer's login credentials are found, a new session will be created.

Subsequently, the `login()` method will call the `signIn()` method (shown in figure 4.6) to assign a '1' to the 'signed_in' attribute of the user object. This attribute has been included in the database to help the administrator identify customers that are currently logged in to their accounts. Accordingly, once customers log off their accounts, the `signOut()` method resets the 'signed_in' attribute's value to '0'. See Figure 4.6 for the `signOut()` method. Finally, the customer will be directed to the home page.

```

function signIn()
{
    $this->signed_in = 1;
    $link = dbconnect();
    $qresult = mysqli_query($link, "update users set signed_in = '{$this->signed_in}'
                                where email = '{$this->email}'");
    dbdisconnect($link);
}

function signOut()
{
    $link = dbconnect();
    $qresult = mysqli_query($link, "update users set signed_in = 0
                                where username = '{$this->username}'");
    dbdisconnect($link);
}

```

Figure 4.6: signIn and signOut method

4.3 Booking Class

This is the most important class in the booking system as it contains the methods that allow customers to make, amend and cancel bookings. In addition, it includes some methods that are used to enable the queuing functionality of the system.

In order to book a timeslot, a customer is required to complete an online booking form that allows him/her to choose a date from a calendar, as well as the camera type and the most preferred timeslot, which could be for 30 minutes or 60 minutes. Once the customer submits the booking form, a PHP script embedded within the booking form creates a new booking object to hold the data entered by the customer. This PHP script (shown in figure 4.7) proceeds to concatenate the date and time selected by the customer before calling the `timeToUnix()` method (described in subsection 4.3.1) to convert the concatenated date and time into a Unix timestamp.

```
$dummy = $xBooking->booking_date." ".$xBooking->booking_time;
$xBooking->start_time = $xBooking->timeToUnix($dummy, 'U');
$xBooking->duration = $xBooking->booking_duration * 60;
$xBooking->EndTime();
$xBooking->username = $_SESSION['current_user'];

$checkTime = $xBooking->validateTime();
if(($xBooking->validateBooking() == "0") && ($checkTime['status'] == 1))
{
    $result = $xBooking->addBooking($_SESSION['current_user_email']);
    if($result == "Successful")
    {
        $message = "<div class='alert-success'> Booking Successful, Booking Ref: $xBooking->booking_ref</div>";
    }else
    {
        $message = "<div class='alert-warning'> Oops.. Invalid booking check your form and try again.</div>";
    }
}
```

Figure 4.7: Booking form PHP script

Thereafter, the PHP script converts the duration selected by the customer to seconds, as this new value is required by the `EndTime()` method to calculate when the customer's timeslot will end. See Figure 4.8 for the `EndTime()` method.

```
public function EndTime()
{
    $this->end_time = $this->start_time + $this->duration;
}
```

Figure 4.8: EndTime method

The next step involves calling the `validateTime()` method to check if the time selected by the customer is valid. In essence, the booking system has been programmed to accept only timeslots in the hours between 9:00AM and 5:00PM, as these are standard business hours.

If the time is valid, the PHP script proceeds to call the `validateBooking()` method to check whether the timeslot selected by the customer is available. A timeslot is available when it has not been booked by another customer. If the timeslot is available, the PHP script calls the `addBooking()` method (shown below) to save the booking details in the database. Afterward, the `addQueue()` method is called to add the customer to a priority queue. The need to add the customer to a queue will be justified in section 4.4.1

```
public function addBooking($user_email)
{
    $link;
    $link = dbconnect();
    $query = "insert into booking (username, product_id, start_time, end_time, duration, type)
values ('$this->username', '$this->product_id', '$this->start_time', '$this->end_time',
        '$this->duration', '$this->type')";
    $query_result = mysqli_query($link, $query);
    if($query_result)
    {
        $this->booking_id = mysqli_insert_id($link);
        $this->booking_ref = self::encrypt($this->booking_id);
        $queryUpdateRef = mysqli_query($link, "update booking set booking_ref = '$this->booking_ref'
            where booking_id = '$this->booking_id'");
        //add booking to a priority queue
        $xQueue = new Queue(); // create a new queue object
        $xQueue = self::MakeQueue();
        $xQueue->addQueue();
    }
}
```

Figure 4.9: Code snippet of the `addBooking` method

Upon successful completion of the booking, the `sendMail()` method is invoked to send a booking confirmation email to the customer. However, if the timeslot is not available, the `scheduleBooking()` method is called to display a list of other available timeslots for that day.

The following sub-sections contain detailed descriptions of the main methods that are used for booking, notably `timeToUnix()`, `validateTime()`, `validateBooking()`, and `scheduleBooking()`.

4.3.1 `timeToUnix()` Method

Basically, the `timeToUnix()` method is used to convert a particular date and time to the Unix timestamp and vice versa. The need to include this method in the booking class was motivated by the fact that the

booking system had to be capable of manipulating the dates and times selected by different customers to avoid conflicts in the timeslots allocated to them – and this could only be achieved if it was possible to convert the dates and times to integers.

There are different ways to convert dates and times to integers in PHP. One of such ways is to use the PHP *idate()* function, which formats a local date and/or time as an integer. However, as this function accepts just one character in the format parameter and may sometimes return fewer digits than expected, a decision was made to convert the dates and times to the Unix timestamp. Besides, since the Unix timestamp is an accurate integer, arithmetic operations such as the addition and/or subtraction of two different dates for instance, will be easier to perform.

To convert a specific date and time to the Unix timestamp, a method from within the booking class has to return a value and an argument indicated as 'U' to the `timeToUnix()` method (shown in figure 1.9). Once this is done, the `timeToUnix()` method creates a new date object and then it converts the current date and time to the Unix timestamp.

```
public function timeToUnix($xdate, $returnType='U')
//H for human readable date and time, U for Unix timestamp
{
    date_default_timezone_set('Europe/London');
    $result = "";
    if($returnType == 'H')
    {
        $result = date('d-m-Y ', $xdate);
    }else
    {
        $date = new DateTime($xdate); // format: DD/MM/YYYY
        $result = $date->format('U');
    }
    return $result;
} }
```

Figure 4.10 `timeToUnix` method

On the other hand, if a method wants the `timeToUnix()` method to convert an existing Unix timestamp to human readable date, that method will return a value and an argument indicated as 'H'. Once the `timeToUnix()` method gets the argument, it will format the date using the PHP *date()* function and then return a formatted string representing the date to the calling method. For the *date()* function to work properly, the default time zone used by all *date()* and *time()* functions in the booking system had to be

set to a particular time zone – in this case, 'Europe/London'. This is because the *date()* function is time zone dependent.

4.3.2 validateTime() Method

The `validateTime()` method is used to check if the time selected by a customer is valid or not.

For this method to work properly, the opening time and closing time had to be defined as constants, with values of '09:00:00' and '17:00:00' respectively. Once a customer selects a date from the calendar, the `validateTime()` method concatenates the date with the predefined opening and closing times and stores the new values in two variables (called `$openingTime` and `$closingTime`). The values in these variables are then converted to the Unix timestamp, using the `timeToUnix()` method described above. See figure 4.11 for the `validateTime()` method.

```
public function validateTime()
{
    $result = "";
    $response = array();

    $selectedDate = self::timeToUnix($this->start_time, 'H');
    $closingTime = "$selectedDate 17:00:00";
    $openingTime = "$selectedDate 09:00:00";
    $closingTimeUnix = self::timeToUnix($closingTime, 'U');
    $openingTimeUnix = self::timeToUnix($openingTime, 'U');
    $now = time(); //returns current Unix timestamp
    $currentDate = splitDateTime($now, 'date');
    if($this->start_time < $openingTimeUnix) //check if time is too early
    {
        $result = "selected time is too early";
    }
    if($this->start_time > $closingTimeUnix) //check if time is too late
    {
        $result = "we have closed for today";
    }
    $response['msg'] = $result;
    if($result == "")
    {
        $response['status'] = 1;
    }else{
        $response['status'] = 0;
    }
    return $response;
}
```

Figure 4.11 `validateTime` method

After this conversion, the PHP *time()* function is used to generate the current Unix timestamp. There are different ways to generate the current Unix timestamp in PHP. One way to do this is to use the `$_SERVER['REQUEST_TIME']` variable which returns the Unix timestamp with microsecond precision at the start of an HTTP request. Another method is to include 'U' as an argument in the format parameter string of a *date()* function. However, since there is no need for microsecond precision in the booking system and the *date()* function is time zone dependent, the *time()* function was chosen.

In general, to check if the time selected by the customer is valid (that is, not earlier than 9:00AM or later than 05:00PM), the Unix values of the opening and closing times are compared with the customer's start time. If the time selected by the customer is valid, the `validateTime()` function returns a 1 and vice versa.

4.3.3 `validateBooking()` Method

Every time a customer tries to make a booking, the system uses the `validateBooking()` method to check the database to confirm whether the timeslot selected by the customer has not been booked by another customer.

```
public function validateBooking() {
    $link = dbconnect();
    $result = "";
    $query = "select * from booking where start_time <= '{$this->start_time}'
and end_time > '{$this->start_time}' and product_id = '{$this->product_id}'";
    $qresult = mysqli_query($link, $query);
    dbdisconnect($link);
    if($qresult)
    {
        if(mysqli_num_rows($qresult) == "0")
        {
            $result = "0";
            return $result;
        }else
        {
            $result = "1";
            return $result;
        }
    }
}
```

Figure 4.12 `validateBooking` method

If the timeslot has not been booked, the method returns a '0' and the `addBooking()` method is called to save the booking in the database. However, if the timeslot has been booked, the `validateBooking()` method returns a '1' and calls the `scheduleBooking()` method, described in section 4.3.4

At the initial stage of development, the `validateBooking()` method was implemented by using a SQL SELECT statement that compared the start and end times of bookings already stored in the database with the current customer's start and end times, to prevent the timeslots allocated to customers from overlapping. However, after testing this method, it was discovered that the SQL statement was not efficient enough to determine which timeslots were available and vice versa. Consequently, this allowed different customers to book the same timeslot.

To resolve this issue, the product IDs were included in the SQL statement. This was a good choice as the product IDs uniquely identified cameras that have been reserved and those that are currently available in the database. With this solution, the `validateBooking()` method was able to prevent timeslots allocated to customers from conflicting with each other.

4.3.4 `scheduleBooking()` Method

The `scheduleBooking()` method is responsible for generating the list of all the remaining timeslots available in a day. In addition, it manages the scheduling processes of the cameras, including tracking the status of each camera and assigning them to customers.

By default, the method takes a 'B' argument and works as the scheduler that handles customer bookings. Moreover, in order to take advantage of code reusability, the method has also been implemented as the scheduler that handles the priority queue, discussed later in section 4.4. In this case, it takes a 'Q' argument.

```

public function scheduleBooking($type = 'B')
{
    $link;
    $result = "";
    $list[][] = array();
    $step = $this->duration;
    $xdate_startTime = "09:00:00";
    $xdate_endTime = " 05:00:00 PM";
    $xdate = self::timeToUnix($this->start_time, "H");
    $fullDateTimeStart = $xdate.$xdate_startTime;
    $fullDateTimeEnd = $xdate.$xdate_endTime;
    $unixfullDateTimeStart = self::timeToUnix($fullDateTimeStart, "U");
    $unixfullDateTimeEnd = self::timeToUnix($fullDateTimeEnd, "U");
    $count = 0;
}

```

Figure 4.13: Code snippet of the `scheduleBooking` method

To implement the `scheduleBooking()` method, a multi-dimensional array had to be created first. This array is essential because it is used to store the list of all the reserved and available timeslots for a particular day.

After creating the array, the date selected by the customer was appended to the predefined start and end times. The values resulting from this concatenation was then converted to the Unix timestamp using the `timeToUnix()` method, as they will be used in the *while loop* to generate the list of available timeslots.

The last step involves executing a *while loop*. More specifically, as long as the *while loop* test condition evaluate to true, a list containing all the timeslots available for the rest of that day, as well as the ones that have been reserved will be generated dynamically. In addition, the list also contains a new booking object that was created because the initial booking object that was passed into the `scheduleBooking()` method had already expired.

```

while($unixfullDateTimeStart < $unixfullDateTimeEnd)
{
    $checkBookingTime = new Booking();
    $checkBookingTime->start_time = $unixfullDateTimeStart;
    $checkBookingTime->product_id = $this->product_id;
    $status = $checkBookingTime->validateBooking();
    $statusTime = $checkBookingTime->validateTime();
    if($statusTime['status'] == '1') // 1 = unavailable timeslot
    {
        //return start times
        $list[$count][0] = date('d-m-Y g:iA', $unixfullDateTimeStart);
        $list[$count][1] = $status; //shows whether timeslot is reserved or available
        $list[$count][2] = $unixfullDateTimeStart;
    }
    //increments the start times based on the duration selected
    $unixfullDateTimeStart = $unixfullDateTimeStart + $step;
    $count ++;
}
return $list; }

```

Figure 4.14: *While loop* for booking

4.4 Queue Class

The queue class contains methods that support priority queuing, which has been implemented as an alternative to booking in the booking system.

In order to include priority queuing in the booking system, some assumptions were made regarding the business logic of the system after reviewing the requirements specification obtained from Digital Barriers. The first assumption that was made is that the booking approach will be used by customers to reserve timeslots and during these timeslots, a salesman will be required to demonstrate the capabilities of a particular camera to them in person. On the other hand, the queuing approach will be used by customers that want to view a demonstration video stream that would allow them to test the camera's functionalities by taking control of the PTZ (Pan, Tilt and Zoom) control of the camera remotely.

Based on these assumptions, the booking and queuing functionalities were initially implemented separately such that a customer will have the choice to either book a timeslot or join the waiting list to gain access to a specific camera's demonstration video stream. However, after testing this, it was discovered that the timeslots that had just been allocated to customers (using the queuing approach) did not fit between timeslots that were booked earlier and this caused conflicting timeslots.

Thus, to overcome this challenge, the booking and queuing functionalities had to be integrated. The first step towards achieving this was to normalise the booking and queuing tables in the database to ensure that information about requests made by different customers to view demonstration video streams of all cameras can be stored in the booking table. With these information, the `scheduleBooking()` method was able to easily check the availability of cameras that have been requested by each customer before granting access – to avoid conflicting timeslots.

In addition, the queuing functionality had to be programmed to accept only timeslots in the hours between 9:00AM and 5:00PM on weekdays. The following code snippet shows how this restriction was enforced.

```
$timestamp = time();
$dw = date( "w", $timestamp); // (Sunday = 0 and Saturday = 6)
if($dw == 0 or $dw == 6)
{
    $officeStatus = "CLOSED";
}
$today = timeToUnix($timestamp, 'H'); // get date from timestamp
$todayFullDateStart = $today." 09:00:00";
$todayFullDateEnd = $today." 17:00:00";
$todayFullDateStartUnix = timeToUnix($todayFullDateStart, 'U');
$todayFullDateEndUnix = timeToUnix($todayFullDateEnd, 'U');
if(($timestamp < $todayFullDateStartUnix) or ($timestamp > $todayFullDateEndUnix))
{
    $officeStatus = "CLOSED";
}
```

Figure 4.15: Restrictions on the queuing approach

4.4.1 How the Priority Queue Works

Once a customer clicks on the 'View Demo' button in the queue form, a PHP script embedded in this form creates a new booking object to hold the product and duration entered by the customer.

After this, the PHP script (shown in Figure 4.16) stores the current Unix timestamp obtained from the PHP `time()` function in a variable, converts the duration selected by the customer to seconds and then it calls the `EndTime()` method to calculate when the customer's timeslot will end.

```

$t=time(); // current Unix timestamp
$xBooking->start_time = $t;
$xBooking->duration = $xBooking->booking_duration * 60;
$xBooking->EndTime();
$xBooking->username = $_SESSION['current_user'];
$checkTime = $xBooking->validateTime();
if(($xBooking->validateBooking() == "0")
{
    $result = $xBooking->addBooking($_SESSION['current_user_email']);
}
else
{
    $list = $xBooking->scheduleBooking('Q');
    if(count($list) > 0)
    {
        $list = TrimArray($list);
        if(!empty($list[1]))
        {
            $xBooking->start_time = $list[1][2];
            $xBooking->QueueOptimization();
            $xQueue = new Queue();
            $waitingTime = $xQueue->getEstimatedWaitingTime($xBooking->start_time);
            ../
            $waiting = $xQueue->getEstimatedPosition($xBooking->start_time, $xBooking->product_id);
            ../
        }
    }
}

```

Figure 4.16: Code snippet of the queuing form PHP script

Now, with all these parameters, the PHP script proceeds to call the `validateBooking()` method to check if the timeslot that will be allocated to the customer has not been booked (or queued for) by another customer. If the timeslot is available, the PHP script calls the `addBooking()` method to save details in the booking table, just like it is done when a customer books a timeslot.

In addition to this, immediately after the booking details have been saved in the booking table, a newly created queue object calls on the `addQueue()` method (see Figure 4.17) to save the details in the queue table. The `addQueue()` method helps to further ensure that the timeslots allocated to customers through the queuing functionality do not coincide with the timeslots already reserved by customers who made use of the booking functionality.

```

function addQueue()
{
    $link;
    $link =dbconnect();
    $qresult = mysqli_query($link, "insert into queue (username, product_id, booking_id)
    values ('$this->username', '$this->product_id', '$this->booking_id')");
}

```

Figure 4.17: `addQueue` method

Afterward, a countdown timer which represents a demonstration video stream begins to countdown. Due to the fact that there was limited support from Digital Barriers, it was not possible to interface the booking system's backend to the TVI server that contains the demonstration video stream. Therefore this had to be simulated by using a jQuery countdown plugin developed by Wood, (2014) to count down in hours, minutes and seconds, depending on the duration chosen by the customer. See Figure 4.18 for the jQuery code snippet handling the countdown timer.

```
$result = $xBooking->addBooking($_SESSION['current_user_email']);
if($result == "Successful")
{
    $demoActive = true;
    $end_time = $xBooking->end_time;
    $dt = new DateTime("@$end_time");
    $end_time = $dt->format('Y, m, d, H, i, s');
    $sTA = explode(',', $end_time);
    $sTA[1] = $sTA[1]-1;
    $sTA[1] = "0".$sTA[1];
    $sTA[3] = $sTA[3]+1;

    $message = "<div id='defaultCountdown'></div>";
?>

<script language="JavaScript">
$(function () {
var TimeDown = new Date();
TimeDown = new Date(<?php echo "$sTA[0]".", ". "$sTA[1]".", ". "$sTA[2]".",
    ". "$sTA[3]".", ". "$sTA[4]"; ?> ); //('Y, m, d, H, i, s');
$('#defaultCountdown').countdown({until: TimeDown, format: 'dHMS'});
$('#year').text(TimeDown.getFullYear());
});
</script>
```

Figure 4.18: PHP script handling countdown timer

Moreover, if the timeslot is not available, the `scheduleBooking()` method's code segment for handling the queuing functionality (shown below in Figure 4.19) will generate a multi-dimensional array that contains a list of all the reserved and available timeslots from the current time of the day till the closing time.

```

if($type == 'Q')
{
    $now = time(); //current Unix timestamp
    while($now < $unixfullDateTimeEnd)
    {
        $checkBookingTime = new Booking();
        $checkBookingTime->start_time = $unixfullDateTimeStart;
        $checkBookingTime->product_id = $this->product_id;
        $checkBookingTime->start_time = $now;
        $status = $checkBookingTime->validateBooking();
        $statusTime = $checkBookingTime->validateTime();
        if($status == 0)
        {
            $list[$count][0] = date('d-m-Y H:i:s', $now);
            $list[$count][1] = $status;
            $list[$count][2] = $now;
        }
        $now = $now + $step;
        $count ++;
    }
}

```

Figure 4.19: *While loop* for queuing approach

Since the queuing functionality is meant to automatically allocate a timeslot to each customer based on the duration selected, the next step will involve calling the `trimArray()` method to remove the empty arrays containing reserved timeslots from the list. See Figure 4.20 for the `trimArray()` method.

```

function TrimArray($sourceArray)
{
    $destination = array();
    foreach($sourceArray as $row)
    {
        if(!empty($row))
        {
            array_push($destination, $row);
        }
    }
    return $destination;
}

```

Figure 4.20: `TrimArray` method

Once the empty arrays have been removed, the `QueueOptimisation()` method (shown in Figure 4.21) will automatically select the closest available timeslot (if any) for the customer.

```

public function QueueOptimization()
{
    $link = "";
    $link = dbconnect();
    $query = "select max(end_time) as end_time from booking
where end_time <= $this->start_time and product_id = $this->product_id";
    $qresult = mysqli_query($link, $query);
    if($qresult)
    {
        $dummyArr = mysqli_fetch_assoc($qresult);
        $this->start_time = $dummyArr['end_time'];
        $this->EndTime();
    }
}

```

Figure 4.21: QueueOptimisation method

Thereafter, the system calls the `getEstimatedPosition()` method to display the number of customers currently on the queue (if applicable). See Figure 4.22 for the `getEstimatedPosition` method.

```

function getEstimatedPosition($startTime, $product_id)
{
    $result = "";
    $now = time();
    $link = "";
    $link = dbconnect();
    $query = "select count(*) as waiting from booking where
end_time > '$now' and start_time < '$startTime' and product_id = '$product_id'";
    $qresult = mysqli_query($link, $query);
    if($qresult)
    {
        $result = (mysqli_fetch_assoc($qresult));
        $result = $result['waiting'];
        return $result;
    }
}

```

Figure 4.22: `getEstimatedPosition` method

Furthermore, the system calls the `getEstimatedWaitingTime()` method to display how long the customer will have to wait on the queue if he/she decides to join the queue. In order to display an accurate wait time, a jQuery code (shown below in Figure 4.23) similar to the one used to generate the countdown timer described earlier was implemented.

```

$xQueue = new Queue();
$waitingTime = $xQueue->getEstimatedWaitingTime($xBooking->start_time);
if($waitingTime)
{
    if($waitingTime < 0){
        header("location: queue.php"); }

    $startTime = $xBooking->start_time;
    $dt = new DateTime("@$startTime");
    $startTime = $dt->format('Y, m, d, H, i, s');
    $sTA = explode(',', $startTime);
    $sTA[1] = $sTA[1]-1;
    $sTA[1] = "0".$sTA[1];
    $sTA[3] = $sTA[3]+1;
    $message = "Next Available time is in $waitingTime <br>
    <div id='defaultCountdown'></div> <br> Would you like to join the queue? ";
?>

<script language="JavaScript">
$(function () {
var TimeDown = new Date();
TimeDown = new Date(<?php echo "$sTA[0].", ". "$sTA[1].", ". "$sTA[2].",
". "$sTA[3].", ". "$sTA[4]"; ?> ); //('Y, m, d, H, i, s');
$('#defaultCountdown').countdown({until: TimeDown, format: 'dHMS'});
$('#year').text(TimeDown.getFullYear());
});
</script>

```

Figure 4.23: Estimated waiting time PHP script

```

function getEstimatedWaitingTime($startTime)
{
    $result;
    $now = time();
    $resultTime = $startTime - $now;
    $result = secondsToTime($resultTime);
    return $result;
}

```

Figure 4.24: getEstimatedWaitingTime method

After this, the estimated waiting time was calculated by simply converting the difference between the current Unix timestamp and the start time of a customer to seconds using the `secondsToTime()` method. See Figure 4.25 for the `secondsToTime()` method.

```

function secondsToTime($ss)
{
    $m = ""; $h = ""; $d = ""; $M = "";
    $s = $ss % 60;
    if((floor(($ss%3600)/60)>0))
    {
        $m = floor(($ss%3600)/60);
        $m = $m." minutes:";
    }
    if(floor(($ss % 86400) / 3600)>0)
    {
        $h = floor(($ss % 86400) / 3600);
        $h = $h." hours:";
    }
    if(floor(($ss % 2592000) / 86400)>0)
    {
        $d = floor(($ss % 2592000) / 86400);
        $d = $d. " days:";
    }
    if(floor($ss / 2592000)>0)
    {
        $M = floor($ss / 2592000);
        $M = $M." months:";
    }
    $result = "$M $d $h $m $s seconds";
    return $result;
}

```

Figure 4.25: secondsToTime method

Finally, if the customer eventually decides to join the queue, the `addBooking()` and `addQueue()` methods are invoked to save the details in the booking and queue tables, respectively.

The source code of the booking system have been included in a CD attached to this report. In general, having discussed the detailed design of the booking system's back-end, the next chapter goes on to describe the testing process that was carried out to validate the booking system.

Chapter 5

Validation

This chapter describes the type of tests that were carried out to verify that the booking system conforms to the functional requirements specified in the System Design chapter.

5.1 Test Strategy

Prior to testing the booking system, a test plan was created to identify the scope, approach, and schedule of intended test activities. This test plan defined the series of tests that was conducted to verify that the booking system satisfies the specified functional requirements.

After this, a decision was made to manually test the main methods the booking system uses for booking and queuing using white-box testing techniques. It is important to note that although validation could have been carried out with automated testing tools, manual testing was chosen because it is more effective in identifying usability issues and trivial bugs that cannot be easily discovered by running automated test scripts. Moreover, while white-box and black-box testing techniques are important and can be used to identify different types of bugs in a software, white-box testing was deemed appropriate and selected because it provides a degree of sophistication that is not available with black-box testing; which is the opportunity to manipulate the internal logic and structure of the source code that makes up a software to determine if the software works as expected rather than finding errors in the external behaviour of the software.

In order to test the main methods of the booking system (which includes the `timeToUnix()`, `scheduleBooking()`, `addBooking()`, `addQueue()` and `cancelBooking()` methods, respectively), test cases were defined based on the specified functional requirements. Each test case contained the name of the method being tested, the feature it is being tested for, the conditions before and after testing the method, the expected result, as well as the actual result of the test. Afterward, each of these methods were run with predetermined input values and checked to make sure that the actual outputs produced are similar to the predicted outputs.

The first step towards testing the booking system was to ensure that the `timeToUnix()` method can convert the dates and time entered by a customer to the Unix timestamp and vice versa. This test case is called UNIT01 and is shown below in Table 5.1.

Test ID	UNIT 01
Unit to Test	Method: <code>timeToUnix</code>
Objective	To test if a specific date and time can be converted to the Unix timestamp
Pre-condition	Date and time exists in standard format
Post-condition	Date and time is converted to the Unix timestamp
Expected Result	Date and time is converted to the Unix timestamp
Actual Result	Success

Table 5.1: Result of testing `timeToUnix` method

The second unit test, UNIT 02 confirms if the scheduler can generate the list of timeslots available in a particular day.

Test ID	UNIT 02
Unit to Test	Method: <code>scheduleBooking</code>
Objective	To test if the list of available timeslots can be generated by the scheduler
Pre-condition	Available timeslots are not shown on the web page
Post-condition	Available timeslots are shown on the web page
Expected Result	Available timeslots are shown on the web page
Actual Result	Success

Table 5.2: Result of testing `scheduleBooking` method

The next logical step was to test whether the booking made by a customer is stored in the database through the `addBooking()` method. This unit test, called UNIT 03 is shown in Table 5.3.

Test ID	UNIT 03
Unit to Test	Method: <code>addBooking</code>
Objective	To test if a booking can be stored in the database
Pre-condition	Booking details does not exist in the database
Post-condition	Booking details are stored in the database
Expected Result	Booking details are stored in the database
Actual Result	Success

Table 5.3: Result of testing `addBooking` method

After verifying that the `addBooking()` method functions properly, the `addQueue()` method was tested to check if the booking made by a customer will be added to the priority queue. The test case is shown in Table 5.4

Test ID	UNIT 04
Unit to Test	Method: <code>addQueue</code>
Objective	To test whether a new booking is added to the priority queue
Pre-condition	Booking details are not included in the queue
Post-condition	Booking details are included in the queue
Expected Result	Booking details are included in the queue
Actual Result	Success

Table 5.4: Result of testing `addQueue` method

The last unit test, shown in Table 5.5 checks whether bookings made by customers can be deleted from the database using the `cancelBooking()` method.

Test ID	UNIT 05
Unit to Test	Method: <code>cancelBooking</code>
Objective	To test whether a booking can be removed from the database
Pre-condition	Booking details exist in the database
Post-condition	Booking details does not exist in the database
Expected Result	Booking details does not exist in the database
Actual Result	Success

Table 5.5: Result of testing `cancelBooking` method

Based on the results of the unit tests conducted, it was discovered that all the test cases executed achieved 100% method and statement coverage, respectively.

After these unit tests, integration tests were conducted. During the integration testing stage, all the methods were combined and tested to evaluate the interaction between them. This was necessary to ensure that the new methods work properly when they are integrated with existing methods and

conformed to the functional requirements of the system. Similarly, regression tests were carried out throughout the development cycle to verify that new modifications (such as bug fixes or enhancements) that have been made to the code did not introduce faults in any previously-working code.

In general, the results obtained from the series of white-box tests that were conducted have showed that the booking system conforms to some of the functional requirements highlighted in the System Design chapter. However, it is important to emphasise that due to time constraints, these tests are by no means exhaustive. Moreover, if there were enough time allocated for the project, further tests would have been carried out to rigorously verify the booking system. In addition, good software engineering strategies, such as equivalence class partitioning and boundary value analysis would have been employed for writing test cases that will increase the chances of uncovering as many bugs as possible in the booking system.

Chapter 6

System Evaluation

This section discusses the evaluation of the booking system. It highlights the processes that have been followed in assessing the system's performance and usability.

6.1 Performance Evaluation

In order to identify an appropriate approach to scheduling in the booking system, the booking and queuing algorithms of the system had to be evaluated against a set of performance criteria, notably waiting time, response time, and throughput.

Here, the waiting time represents how long a customer's request will have to wait in a queue before being processed by the scheduler while the response time indicates the average time in milliseconds the scheduler spends in processing a request for service. Similarly, the throughput represents the number of requests that can be processed by the scheduler per unit of time. These parameters are essential, as the primary objective of the research is to evaluate the efficiency of the practical approaches to resource scheduling in online booking systems.

There are different load testing tools that can be used to measure and analyse the performance of the booking system. An example is the Apache JMeter which is an open-source software that is designed to load test different types of software – with a focus on web applications. In addition to the fact that the JMeter can be used to test the performance of both static and dynamic resources such as ASP.NET, PHP and Java files, etc., it can also be used to assess the overall performance of servers and networks under different workloads. However, having considered the fact that JMeter is limited in terms of reporting capabilities and that it does not perform all the actions supported by web browsers, such as executing the JavaScript found in HTML pages, BlazeMeter was chosen as the appropriate load testing tool to evaluate the performance of the booking system.

In essence, BlazeMeter is a cloud-based load testing platform that is used to perform load testing on mobile applications and websites. It is capable of emulating hundreds of concurrent virtual users to apply accurate workloads to any application. As BlazeMeter drives load against the application from load generators in the cloud, it captures the waiting and response times, throughput and processor utilisation

rate, etc. of business processes and transactions to determine whether the application can meet performance requirements.

Moreover, to determine the baseline performance of the algorithms that support the booking and queuing functionalities of the system, two load tests had to be executed and in each of these load tests, the number of concurrent virtual users trying to book or queue for a particular camera was set to 50. The number of concurrent users was limited to 50 because the free subscription to BlazeMeter that was used allows a developer to load test a website and/or mobile application with a maximum of 50 concurrent users.

6.1.1 Load Test 1

During the first load test, the booking functionality of the system was subjected to heavy workloads generated by 50 virtual users trying to book a specific camera at the same time. Here, workload refers to the stimulus applied to simulate a usage pattern and it includes the total number of concurrent users, data volumes, and transaction volumes, along with the transaction mix. The result obtained from the load test is presented in Appendix E.

Based on the results of this load test, it was discovered that as the number of users increase, the response time also increases. This clearly shows that the number of users trying to make a booking at the same time directly affects the speed at which each user is able to access and interact with the booking system. This trend is shown in the graph below:

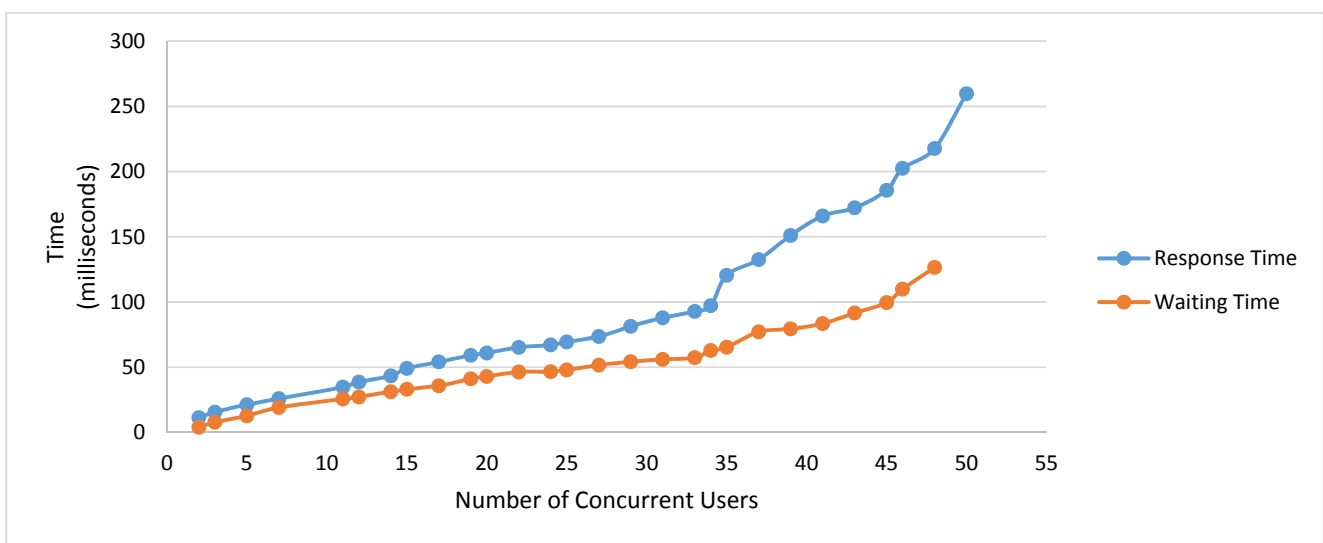


Figure 6.1: Response time and Waiting time vs number of concurrent users – (booking)

The response time peaks at 259.8 milliseconds when there are 50 concurrent users using the booking system. In general, with response time being an important parameter that affects the user experience of a website, a response time of 259.8 milliseconds was deemed appropriate, as this indicates that in ideal scenarios each user will not have to wait for more than 259.8 milliseconds before the scheduler processes their request.

Moreover, just like the response time of the booking functionality of the system has been observed to increase with larger number of concurrent users, the waiting time and the throughput of the system also increases in a similar way. See Figure 6.2 for the graph of throughput vs user load.

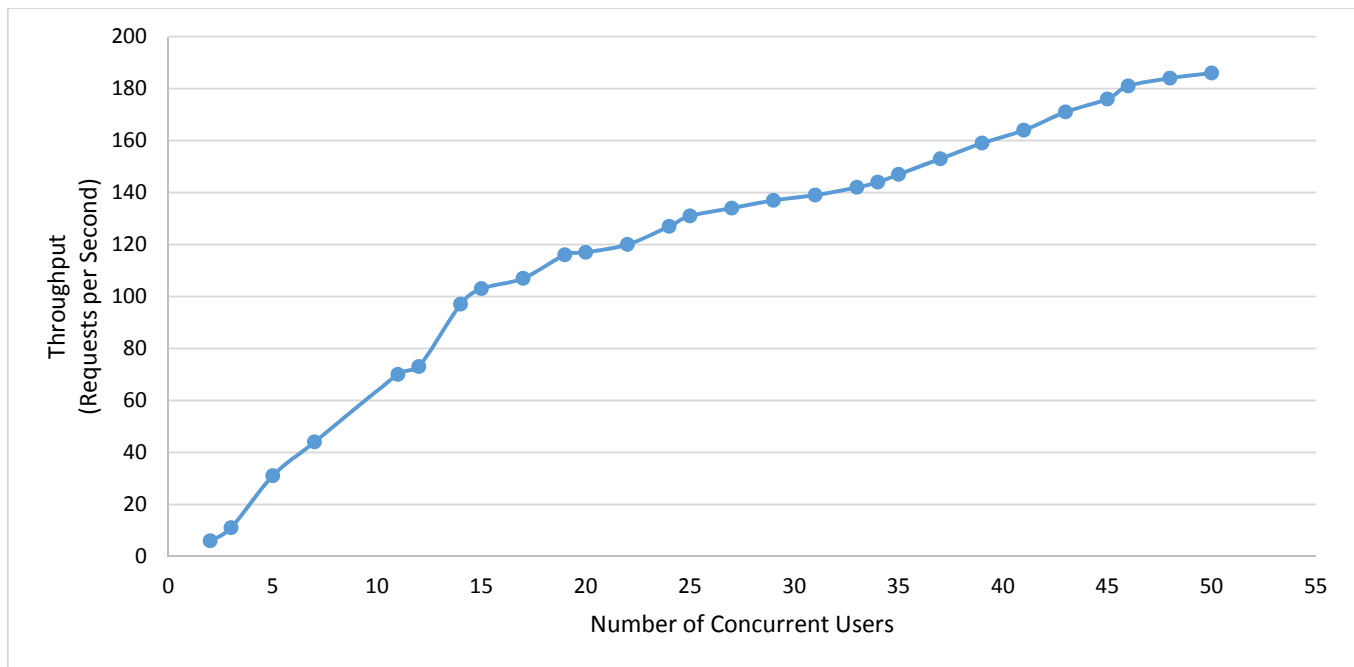


Figure 6.2: Throughput vs number of concurrent users – (booking)

6.1.2 Load Test 2

In this second load test, the queuing functionality of the system was also subjected to the same amount of workload generated by 50 virtual users, although this time the users were trying to queue to access a specific camera for 30 minutes. The result obtained from this load test is presented in Appendix E.

From the results of this second load test, it was discovered that as the number of user increases, the response time increases rapidly. This trend is depicted in the graph below:

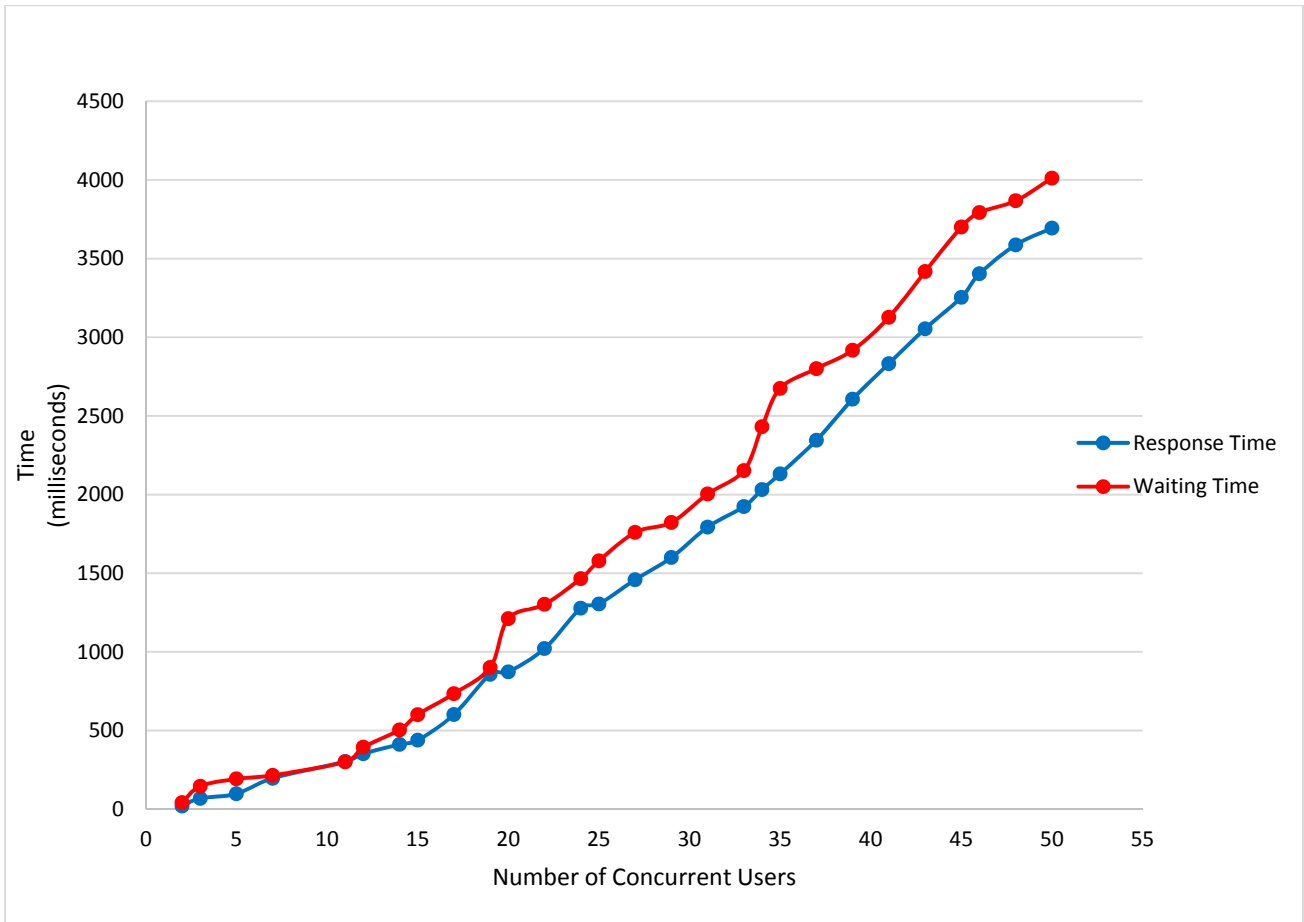


Figure 6.3: Response time and Waiting time vs number of concurrent users – (queuing)

By observing this graph, the response time starts with a gentle rise and linear growth for low to medium levels of workload. However, with an increase in the number of concurrent users trying to queue for the same camera, the number of requests processed by the scheduler increases and as a result, this causes the response times of the system to increase. The peak response time is 3693.87 milliseconds (compared to 259.8 milliseconds for booking) and this is recorded when the number of users increases to 50. At this point, there will be slight delays with regards to the time taken to receive full response from the server.

Similarly, as the scheduler becomes busier due to an increase in the number of requests that needs to be processed, the waiting time increases accordingly in a non-linear fashion. As a matter of fact, the busier the scheduler is, the more dramatic the response time will increase. This abrupt increase in the response time is caused by increases in the waiting time and this affects the user experience.

It is important to note that the sudden increase in response time for the queuing functionality can be partly attributed to the fact that for a particular user to be able queue for a camera, the system must first call the `timeToUnix()` method to convert the start and end times selected by the user to the Unix timestamp. Thereafter, the `validateBooking()` method is called to check if the time selected by the user has not been reserved by another user. If the time is available, the system calls the `addBooking()` and `addQueue()` methods to save the details in the database and include the user on the queue, respectively. After all these methods have been called, the system will now call the method that handles the countdown timer, to simulate a demonstration video stream. Indeed, all these method invocation adds to the time it takes the scheduler to process each user request and the response time is further affected when there are more users trying to queue for a particular camera at the same time.

Moreover, based on the results of the second load test, the rapid increase in response time also causes an increase in the throughput of the system. This trend is shown in the graph below:

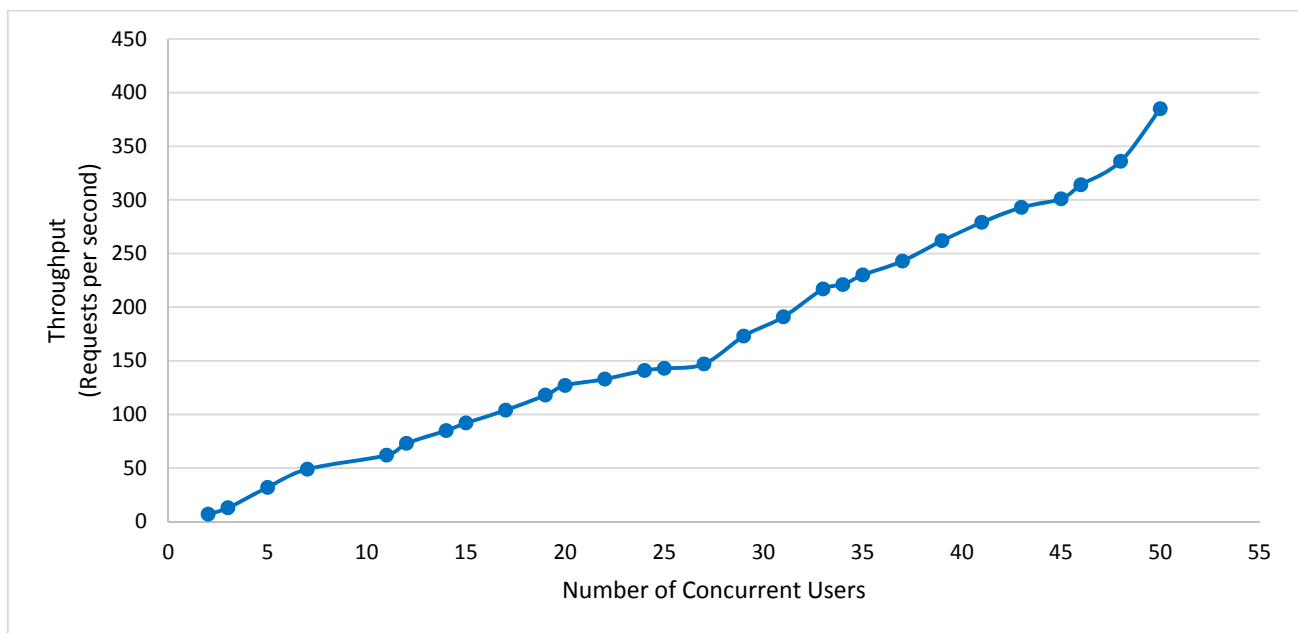


Figure 6.4: Throughput vs number of concurrent users – (queuing)

From this graph, it was observed that as the user load is increased, the throughput increases accordingly up until a certain point (when the number of users reaches 24) where the throughput levels out before increasing abruptly again. Although throughput rate will vary depending upon the number of concurrent users and the type of activity the user performs, the levelling out of the throughput indicates that the scheduler has reached its capacity to process customer's request, and is unable to scale further.

6.1.3 Summary of Results

Basically, booking and queuing have been implemented as two different and mutually exclusive approaches to scheduling in the booking system. From the load testing that was carried out, it has been discovered that booking is more efficient than queuing. This is because users are able to make bookings within a predefined timeslot that could either be of 30- or 60-minute duration. In addition, the booking functionality of the system is very scalable, as it can handle additional workload without adversely affecting the system performance.

On the other hand, the queuing functionality has increased waiting and response times because users are dynamically allocated a timeslot based on the remaining timeslots available for the rest of the day. And because the system has to invoke some methods to check the availability of each timeslot to accomplish this, this makes the queuing algorithm somewhat inefficient when compared to the booking algorithm.

The result of this performance evaluation correlates with the empirical evidences presented in Orduña and García-Zubia, (2011); and Lowe, (2013). According to Orduña and García-Zubia, (2011); and Lowe, (2013), the performance of the booking or queuing approach to scheduling depends mainly on a number of factors such as the scheduling algorithm, the number of concurrent users, the number of available resources, and the duration chosen by each user.

In the course of this performance evaluation, it has been observed that with the booking functionality, as the number of concurrent user increases, the response time increases faster, compared to the waiting time. On the contrary, if the queuing functionality is used and there are 15 concurrent users for example waiting to access a particular camera for 30 minutes, then the last user on the queue will have to wait for approximately 0.60 seconds, which is acceptable. However, if there are 50 concurrent users, then the last user will have to wait for approximately 4.0 seconds. While this waiting time may not have a fairly significant effect on the user experience, it can be inferred that when the number of concurrent users increases above 50, the waiting and response times will increase significantly and consequently, the user experience would be affected. This makes it evident that there can be complex interdependencies between booking and queuing that can affect the optimisation of resource usage when they are combined in a single system.

In general, both the booking and the queuing functionality of the system have been able to optimise the overall usage of the cameras and all of the key performance indicator values (such as the values for waiting time, response time and throughput) are within acceptable limits of the set performance thresholds. It is also important to emphasise that the CPU utilisation rate of the application server never exceeded 60 percent while performing the load tests.

6.2 User Evaluation

The primary aim of conducting user evaluation is to assess the usability of the booking system. To achieve this, ethical approval of studies had to be obtained first.

After this approval was obtained, five MSc students from the Department of Computer and Information Sciences (CIS) were recruited as participants via an email invitation (shown in Appendix F) to carry out the evaluation. The decision to limit the sample size for the usability test to five people was made on the basis of the findings and discussions in Turner *et al* (2006), which emphasised that most usability problems are detected with the first three to five users, and subsequently testing with more users is unlikely to reveal new usability issues. Moreover, given the importance of usability testing in software development, participant selection was limited to students from the CIS Department because the researcher believes that computer science students are more familiar with usability heuristics for user interface design and consequently, would be able to identify any usability problems in the booking system than other university students.

In line with the requirements of the University Ethics Committee, participants were advised prior to evaluating the system that all data would be anonymised and that they have the right to withdraw from the study at any time, without having to give a reason and without any consequences. They were also advised on how the data collected would be stored, processed and destroyed.

After satisfying these requirements, each participant was given a participant information sheet that contained a set of instructions (outlined in Appendix G) on how to use the booking system to perform some tasks. On completion of the tasks outlined in the information sheet, the participants were asked to fill out a questionnaire administered online through SurveyMonkey® (see Appendix H for the user evaluation questionnaire).

6.2.1 Questionnaire Design

The questionnaire includes two standard demographic questions (about age and gender), two open-ended questions about any problem encountered while a participant was using the system and any suggestions that could improve the system, as well as ten questions from the System Usability Scale (SUS), described in sub-section 6.2.1.1

In formulating the survey questions, the suitability of open- and close-ended questions were considered and a decision was made to limit the use of open-ended questions. More specifically, while open-ended questions give respondents the freedom to express their opinions, answers to open-ended questions are difficult to code and analyse, compared to close-ended questions. In view of this, more close-ended questions were included as they are particularly useful when trying to prove the statistical significance of a survey's results and most importantly, the fact that the responses obtained from close-ended questions can be used to categorise respondents into groups makes it easier to compare the responses.

Furthermore, the demographic questions were included in the questionnaire to help determine what factors may influence a participant's answers and opinions and also to cross-tabulate and compare the survey data across multiple demographics.

6.2.1.1 System Usability Scale

The System Usability Scale (SUS) is the most popular standardised usability questionnaire for measuring the perceived usability of a system (Sauro and Lewis, 2011). It consists of ten Likert items on a five-point scale ranging from *Strongly Disagree* to *Strongly Agree*.

The odd-numbered items are positively-worded while the even-numbered items are worded negatively. This positive and negative wording helps to minimize acquiescence and extreme response biases. In essence, while acquiescence bias occurs when respondents to a survey passively agree with all questions in the questionnaire, extreme response bias occurs when respondents select only the most extreme options or answers available (Vaerenbergh and Thomas, 2013). When these biases occur, the affected responses often lead to errors while analysing the questionnaire data and these errors can have a large impact on the validity of the survey.

It is important to note that while other questionnaires such as the Questionnaire for User Interaction Satisfaction (QUIS) and the Computer System Usability Questionnaire (CUSQ) could have been used to

assess the booking system's usability, the System Usability Scale was deemed appropriate and selected because it measures both learnability and usability, yields more reliable results across sample sizes and has been shown to detect differences in smaller sample sizes than other usability questionnaires (Sauro, 2011). After analysis, SUS questionnaire results yield a score between 0 and the 100th percentile and this represents a composite measure of the overall usability of the system being studied.

6.2.2 Analysis and Results

Before the questionnaires were analysed, all responses to the negatively-worded questions had to be reverse-scored with SPSS. Reverse-scoring the negatively-worded questions ensure that all of the questions – those that are originally negatively-worded and those that are positively-worded – are consistent with each other, in terms of what an “agree” or “disagree” imply, for instance. After this was done, the System Usability Scale score of each respondent was calculated using the scoring strategy highlighted in Brooke, (2013).

Based on this scoring strategy, each item can have a score that ranges from 0 to 4. While the score contribution of the positive worded items is obtained by subtracting 1 from the scale position, the score contribution of the negatively worded items is obtained by subtracting 5 from the scale position. The overall value of the SUS is then obtained by multiplying the sum of the scores by 2.5

With the entire sample size, all the completed questionnaires indicated that the usability of the booking system was satisfactory and in general, the average SUS score obtained from these questionnaires after converting the SUS score of each questionnaire to a percentile rank was a 68. See Table 6.1 for the results of the usability study.

Likert Items	Strongly Disagree	Disagree	Disagree Nor Agree	Agree	Strongly Agree	TOTAL
I think that I would like to use the system frequently	0	0	0	3	2	5
I found the system unnecessarily complex	3	1	0	1	0	5
I thought the system was easy to use	0	0	0	2	3	5
I think that I would need the support of a technical person to be able to use the system	3	2	0	0	0	5
I found the various functions in the system were well integrated	0	1	1	3	0	5
I thought there was too much inconsistency in the system	1	2	2	0	0	5
I would imagine that most people would learn to use the system very quickly	0	0	0	4	1	5
I found the system not very intuitive	1	2	1	1	0	5
I think the system response time is satisfactory	0	0	0	3	2	5
I needed to learn a lot of things before I could get going with the system	3	2	0	0	0	5

Table 6.1: Responses to the System Usability Scale questionnaire

Overall, how would you rate the user-friendliness of the system?

Answered: 5 Skipped: 0

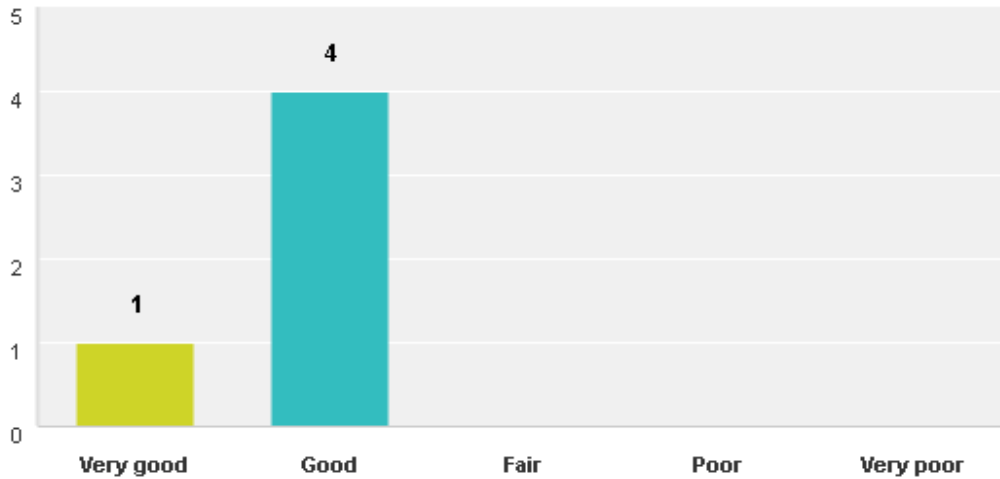


Figure 6.5: User-friendliness of the system

6.2.3 Observations

The feedback received from participants about the user interface design was positive. All participants were able to navigate within the booking system, and most commented that the navigation choices are logical and intuitive. In addition, the participants highlighted that information on the website was presented in a simple and logical order. However, three participants emphasised that the information density on the booking and queuing forms were relatively low and recommended that the additional form fields should be added to these forms.

In general, results gathered from the survey shows that the booking system is responsive, easy to use and has a consistent, clearly recognizable look and feel that will allow users to accomplish their tasks in a reasonable amount of time without unnecessary cognitive load – although there are still some improvements that can be made to the booking system to make it more user-friendly.

Chapter 7

Conclusion and Recommendations

This chapter concludes the dissertation by providing a summary of the project and the application developed. In addition, it discusses how the objectives of the research, outlined in chapter 1 were achieved. Finally, the chapter highlights the limitations on the project and some suggestions for future work.

7.1 Conclusion

The purpose of this project has been to identify an appropriate methodological approach to implement scheduling in an online booking system. To achieve this, the research involved reviewing existing works related to scheduling in order to acquire comprehensive understanding of the practical approaches that can be used to schedule resources in online booking systems.

From this background study, it was discovered that most of the scheduling systems typically supported either booking or queuing, though rarely both, for scheduling access to resources. Moreover, as there was no justification in the research literatures for using only booking or queuing for resource scheduling, the possibilities of integrating both approaches to leverage their benefits in a booking system was explored.

To achieve this, an experimental web-based booking system was developed. This booking system allows customers to book cameras using the booking functionality and also included the queuing functionality to allow a customer queue to access a particular camera demonstration video stream on an ad-hoc basis. Based on the results obtained from the unit and integration tests performed, it was discovered that both the booking and queuing functionalities of the system ensure that customers have exclusive access to the cameras requested within the timeslots specified, in a way that optimises the overall usage of each camera. In addition, usability testing was also performed to gather the data needed to evaluate how usable the system is and if it contains the required functionality.

Finally, in order to determine the most suitable approach to scheduling, the performances of the booking and queuing functionalities of the system were evaluated through load tests. Based on the results of the load tests, the queuing functionality was found to be somewhat inefficient when there are

more concurrent users trying to access the same camera, compared to the booking functionality. This makes the booking functionality more appropriate for scheduling in the context of this project. However, with certain modifications to the queuing algorithm, the efficiency of the queuing functionality could improve.

In general, the results of this research have shown that the booking and queuing approaches to scheduling have their relative merits and demerits and that they are typically suited to different usage scenarios. Moreover, the results have also highlighted that the choice (and performance) of any scheduling approach depends on a number of factors which includes the scheduling algorithm, the number of concurrent users, the available resources, and the usage duration of each user.

7.2 Limitations

In the course of developing the booking system, it was not possible to connect the booking system's backend to a TVI server that contains demonstration video streams. This is due to the fact that there was limited support from Digital Barriers. Therefore, in order to implement the queuing functionality of the system, a jQuery countdown timer had to be used to simulate the video stream.

7.3 Learning Outcomes

By completing this project, the researcher has been able to:

1. Acquire a comprehensive understanding of the practical approaches to scheduling in the context of online booking systems.
2. Identify an appropriate methodological approach to implement scheduling in an online booking system, whilst considering the strengths and weaknesses of this approach.
3. Gain some relevant hands-on experience in using Structured Query Language (SQL) and HyperText Preprocessor (PHP) to implement scheduling in a web application.

7.4 Suggestions for Future Work

Based on the limitations of the methods used in developing the booking system and the outcomes of this project, the following suggestions have been made as recommendations for further work:

1. The first and most important issue that needs to be addressed is the encryption of the data that is stored in the database. This is necessary to protect their confidentiality.

2. The booking algorithm of the system should be modified to limit the number of bookings that can be made by each customer for a particular camera model in any 24-hour period. This would help to further optimise the overall usage of the cameras.
3. There is need to have PHP test scripts that can be used to evaluate the performance of the booking and queuing functionalities of the booking system, respectively.
4. The booking functionality of the system can be extended by including the rating of customers, such that when a customer with a lower rating tries to make a booking at the same time as a customer with higher rating, the system would dynamically give priority to the customer with higher rating.
5. Due to time constraints, it was not possible to develop the mobile version of the booking system. With regards to this, it would be helpful if another version of the booking system could be developed and optimised for mobile devices and tablets.
6. Lastly, for the queuing functionality of the booking system to work as specified in the System Design chapter, it would be necessary to connect the booking system's back-end to a TVI server, which contains the list of demonstration video streams that are specific to each camera.

References

1. Bangor, A., Kortum, P. and Miller, J. (2009): Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4(3), pp.114-123. Available online: http://www.usabilityprofessionals.org/upa_publications/jus/2009may/JUS_Bangor_May2009.pdf. Last accessed 11 August 2013.
2. Brooke, J. (2013). SUS: A retrospective. *Journal of Usability Studies*, 8(2), pp.29-40. Available online: http://www.upassoc.org/upa_publications/jus/2013february/JUS_Brooke_February_2013.pdf. Last accessed 10 August 2014.
3. Cockburn, A. (2008). Using both Incremental and Iterative development". *CrossTalk: The Journal of Defense Software Engineering*, 21(5), pp. 27-30. Available online: <http://www.crosstalkonline.org/storage/issue-archives/2008/200805/200805-Cockburn.pdf>. Last accessed 26 July 2014.
4. Coley, C.T., Nessland, K.S., Leonhardt, T.F., Barry, C.J., Wilson M.F., and Nettuno, A.N. (2012). *Systems and methods for online scheduling of appointments and other resources*. United States Patent Number 008244566B1. Available online: <https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US8244566.pdf>. Last accessed 7 July 2014.
5. Gallardo, A., Richter, T., Debicki, P., Bellido, L., Mateos, V., and Villagra, V. (2011). A rig booking system for online laboratories. *Proceedings of the 2011 IEEE Global Engineering Education Conference (EDUCON), Amman, Jordan, 4-6 April, 2011*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5773206>. Last accessed 10 July 2014, pp. 643-648.
6. Harchol-Balter, M. (2013). *Performance modelling and design of computer systems: Queueing theory in action*. 1st Edition. New York: Cambridge University Press.
7. HP, (2010). *An Introduction to load testing for web applications*. Business whitepaper. Available online: <http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA1-3944EEW.pdf>. Last accessed 17 July 2014.
8. Kilhwan, K. (2012). T-preemptive priority queue and its application to the analysis of an opportunistic spectrum access in cognitive radio networks. *Journal of Computers and Operations*

- Research*, 39(7), pp. 1394-1401. Available online: <http://www.sciencedirect.com/science/article/pii/S0305054811002322#>. Last accessed 8 July 2014.
9. Li, Y., Esche, S.K., and Chassapis, C. (2008). A scheduling system for shared online laboratory resources, *Proceedings of the 38th IEEE Frontiers in Education Conference (FIE 2008), New York, USA, 22-25 October, 2008*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4720253>. Last accessed 7 July 2014, pp.T2B-1, T2B-6.
 10. Lombardi, M. and Milano, M. (2012). Optimal methods for resource allocation and scheduling: A Cross-Disciplinary Survey. *International Journal of Constraints*, 17(1), pp. 51-85. Available online: <http://link.springer.com/content/pdf/10.1007%2Fs10601-011-9115-6.pdf>. Last accessed 7 July 2014.
 11. Lowe, D. (2012). Impacts of scheduling algorithms on resource availability. *Proceedings of the Australian Society for Computers in Learning in Tertiary Education Refereed Conference Collection (Ascilite 2012), Wellington, New Zealand, 25-28 November, 2012*. Available online: http://www.ascilite.org/conferences/Wellington12/2012/images/custom/lowe,_david_-_impacts_of.pdf. Last accessed 8 July 2014, pp.575-579.
 12. Lowe, D. and Orou, N. (2012). Interdependence of booking and queuing in remote laboratory scheduling. *Proceedings of the 9th International Conference on Remote Engineering and Virtual Instrumentation (REV 2012), Bilbao, Spain, 4 - 6 July, 2012*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6293100&isnumber=6293092>. Last accessed 8 July 2014, pp.1-6.
 13. Lowe, D. (2013). Integrating reservations and queuing in remote laboratory scheduling. *IEEE Transactions on Learning Technologies*, 6(1), pp. 73-84. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6413147>. Last accessed 23 August 2014.
 14. Maiti, A. (2011). A hybrid algorithm for time scheduling in remotely triggered online laboratories. *Proceedings of the 2011 IEEE Global Engineering Education Conference (EDUCON), Amman, Jordan, 4-6 April, 2011*. Available

- online:<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5773256>. Last accessed 10 July 2014, pp. 921-926.
15. Maiti, A and Tripathy, B.K. (2011). An improved scheduling scheme for the management of online laboratories. *Proceedings of the 2011 IEEE Recent Advances in Intelligent Computational Systems (RAICS) Conference, Trivandrum, India, 22-24 September, 2011*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6069394>. Last accessed 11 July 2014, pp. 667-670.
 16. Maiti, A. and Maiti, C.K. (2013). Development of remote laboratories using cloud architecture with web instrumentation. *Proceedings of the 10th International Conference on Remote Engineering and Virtual Instrumentation (REV), Sydney, Australia, 6-8 February, 2013*. Available online at: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6502902>. Last accessed 8 July 2014, pp.1-4.
 17. Maiti, A., Maxwell, A.D. and Kist, A.A. (2013). An overview of system architectures for remote laboratories. *Proceedings of the 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bali, Indonesia, 26-29 August, 2013*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6654520>. Last accessed 5 July 2014, pp. 661-666.
 18. Orduña, P. and García-Zubia, J. (2011). Scheduling schemes among Internet Laboratories ecosystems. *Proceedings of the 8th International Conference on Remote Engineering and Virtual Instrumentation (REV 2011), Brasov, Romania, 28 June – 1 July, 2011*. Available online: http://www.morelab.deusto.es/publications/2011/pOrduna_rev2011b.pdf. Last accessed 24 August 2014, pp. 1-6.
 19. Pinedo, M.L. (2012). *Scheduling: theory, algorithms and systems*. 4th Edition. New York: Springer.
 20. Sauro, J. (2011). *A practical guide to the System Usability Scale: Background, benchmarks, and best practices*. 1st Edition. Denver, CO: Measuring Usability LLC.
 21. Sauro, J. and Lewis, J.R. (2011). When designing usability questionnaires, does It hurt to be positive? *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vancouver, Canada 7-12 May, 2011*. Available online:

- http://www.measuringusability.com/papers/sauro_lewisCHI2011.pdf. Last accessed 14 August 2014, pp.2215-2224.
22. Sommerville, I., (2010) *Software engineering*. 9th Edition. Boston: Pearson Education Inc.
 23. Stanford, D.A., Taylor, P. and Ziedins, I. (2013). Waiting time distributions in the accumulating priority queue. *Queueing Systems: Theory and Applications (QUES)*, 76(203), pp. 1-34. Available online: <http://link.springer.com/content/pdf/10.1007%2Fs11134-013-9382-6.pdf>. Last accessed 8 July 2014.
 24. Tullis, T.S. and Stetson, J.N. (2004). *A comparison of questionnaires for assessing website usability*. Boston: Fidelity Center for Applied Technology. Available: URL <http://home.comcast.net/~tomtullis/publications/UPA2004TullisStetson.pdf>. Last accessed 16 August 2014.
 25. Turner, C.W., Lewis, J.R., and Nielsen, J. (2006). 'Determining usability test sample size' in W. Karwowski (ed.), *International Encyclopedia of Ergonomics and Human Factors*, Second Edition, Volume 3, CRC Press, Boca Raton, USA, pp.3084-3088. Available online: http://drjim.0catch.com/2006_DeterminingUsabilityTestSampleSize.pdf. Last accessed 18 August 2014.
 26. Vaerenbergh, Y.V. and Thomas, T.D. (2013). Response styles in survey research: A literature review of antecedents, consequences, and remedies. *International Journal of Public Opinion Research*, 25(2), pp.195-217. Available online: <http://ijpor.oxfordjournals.org/content/25/2/195.full.pdf> Last accessed 19 August 2014.
 27. Wei, T., Dongxu, R., Zhiling, L., and Narayan, D. (2012). Adaptive metric-aware job scheduling for production supercomputers. *Proceedings of the 2012 41st International Conference on Parallel Processing Workshops (ICPPW), Pittsburgh, USA, 10-13 September, 2012*. Available online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6337469>. Last accessed 9 July 2014, pp. 107-115.
 28. Wood, K. (2014). *jQuery Countdown v2.0.1*. Australia: GitHub. Available: URL <http://keithwood.name/countdown.html>. Last accessed 18 August 2014.

Appendix A: System Documentation

This section describes the process of installing the booking system.

Appendix A.1: Requirements

Before the booking system can be installed, there is need to have a recent version of phpMyAdmin installed to handle the administration of the MySQL database. Since phpMyAdmin makes use of a web browser to perform database administration tasks, there is also need to have a web server (such as Apache Tomcat) to install the phpMyAdmin's files into.

Once the phpMyAdmin files have been installed, a web browser with cookies and JavaScript enabled will be required to access the phpMyAdmin's interface. In addition to this, there is need to have MySQL version 5.5 or newer, as well as PHP version 5.3.0 or newer, with session support, the Standard PHP Library (SPL) extension and JSON support.

The next step after these requirements have been met is to upload all the booking system files to the root directory of the hosting provider. These files are enclosed in a sub-folder called "book-tvi-demo" located in the installation folder of the CD attached to this report.

Appendix A.2: Database Setup

After uploading the system files to the hosting provider, the database have to be set up. This can be achieved by importing the database dump file (book-tvi-demo-db.sql) – also included in the installation folder of the CD, into phpMyAdmin.

Through phpMyAdmin, choose "book-tvi-demo-db.sql" from the installation folder as the file to import and then click Go, as shown below in Figure A.1. The database has been pre-populated with the names of some fictitious test users.

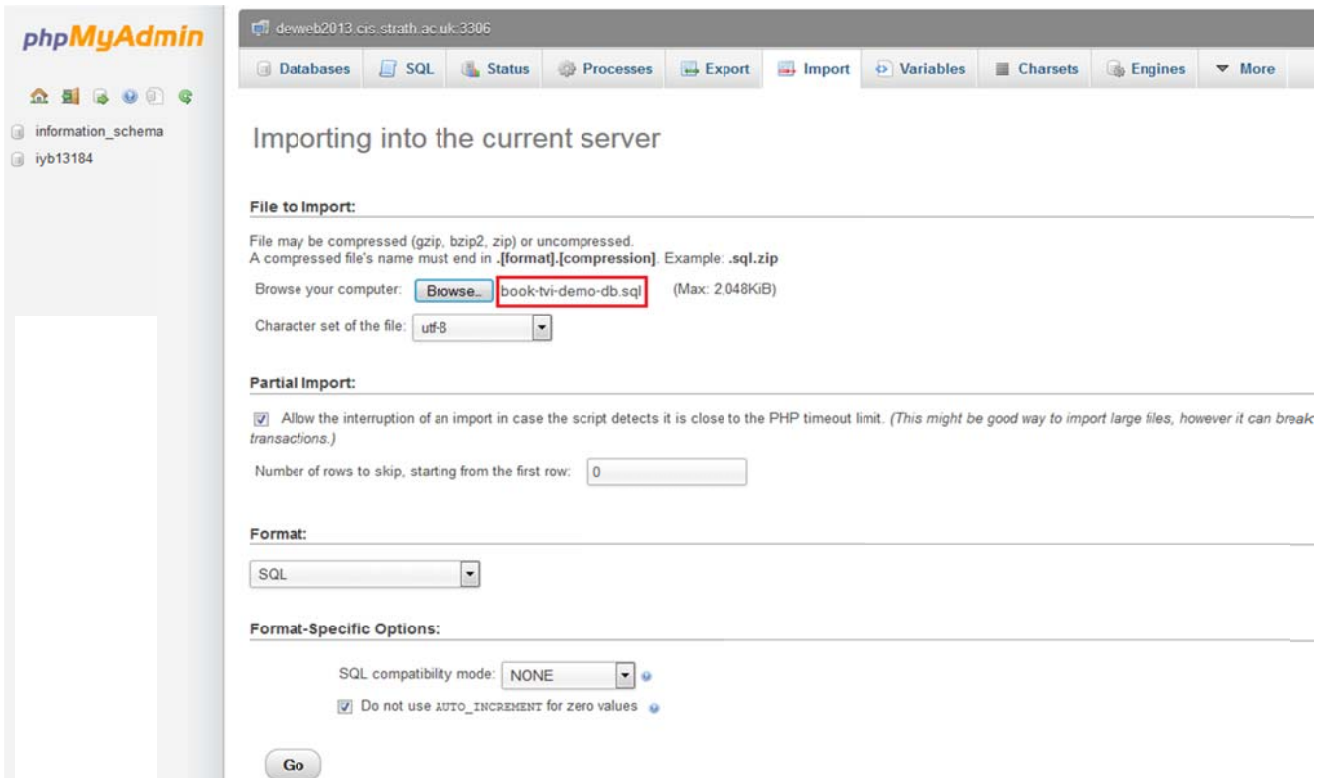


Figure A.1: Importing the database

Once the database has been successfully imported into phpMyAdmin, it should look like the screenshot shown below:

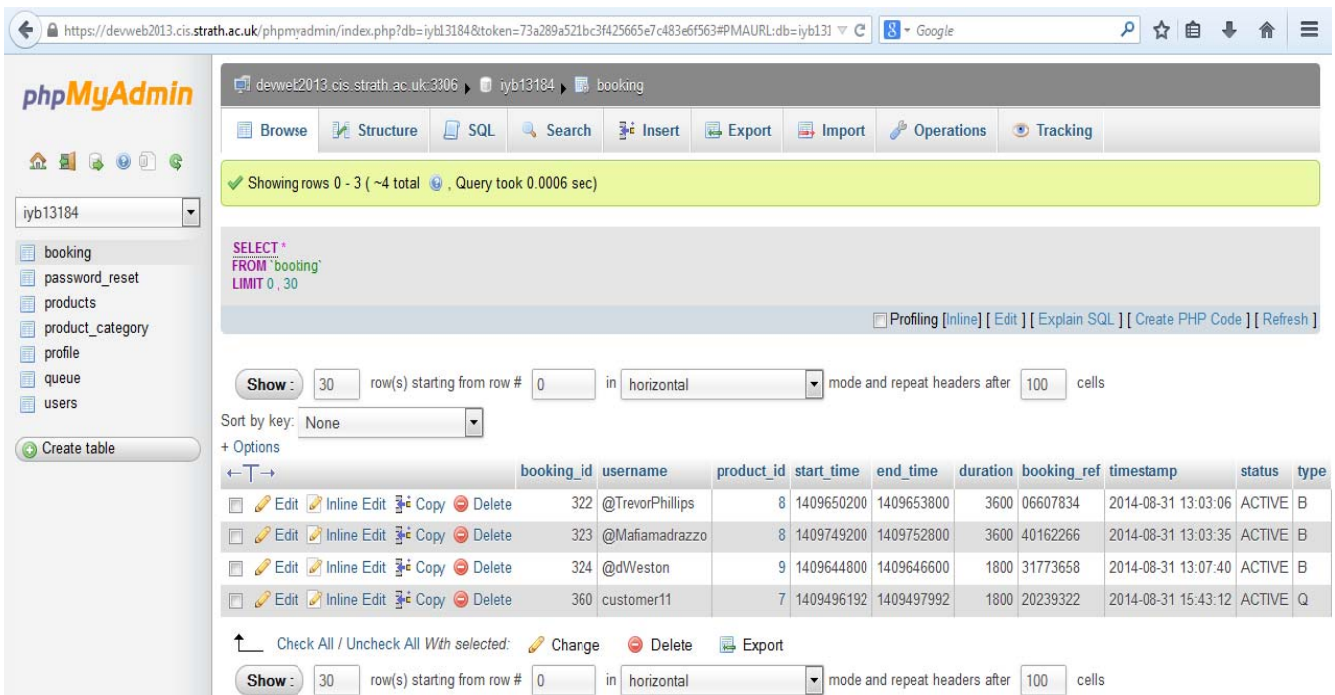


Figure A.2: Screenshot of the database

Appendix A.3: Configuration

Once the database has been set up, the next step is to configure the system files so that the right parameters are used to access the database and web server.

The parameters that are used to connect to the database are found in the “helper.php” file (shown below) located in the “model” sub-folder of the “book-tvi-demo” folder.

```
    /**
define('HOST', 'devweb2013.cis.strath.ac.uk');
define('USER', 'iybl3184');
define('PASS', 'Cognoscenti19');
define('DBNAME', 'book-tvi-demo');
    /**/
function dbconnect()
{
    $link = mysqli_connect(HOST, USER, PASS);
    if(@mysqli_ping($link) == true)
    {
        mysqli_select_db($link, DBNAME);
        return $link;
    }
}
function dbdisconnect($link)
{
    mysqli_close($link);
}
function dbConnectionChecker($link)
{
    $result = @mysqli_ping($link);
    $result? $result = true : $result =false;
    return $result;
}
```

Figure A.3: Database parameters

These parameters include the server hostname, the username and password used to connect to the database and lastly, the name of the MySQL database.

The process of installing the booking system is completed once these parameters have been set. To view the booking system in your browser, use this base URL: ../book-tvi-demo/index.php. This should take you to the Log In page. If you are not able to view the system there may be an error in the server or database setup.

Appendix B: User Guide

This guide provides detailed information about how the booking system should be used by a user. The guide is accompanied by screenshots and some instructions on what data to enter and how to progress to successive pages within the system.

Given that there are two types of users (that is, customers and administrator) that can use the booking system, the user guide has been separated into two different guides, one for the customer and the other for the administrator. In order to allow these users to have easy access to the booking system, the system has been hosted on the CIS Department's web server. The URL is:

<https://devweb2013.cis.strath.ac.uk/~iyb13184/book-tvi-demo/index.php#>

Email	Password
customer@tvidemo.com	test123
admin@tvidemo.com	test435

Table B.1: Login details for the different types of users

Before customers and the administrator can use the booking system's functionalities, they are required to login to the system. To log in to the system, the user must enter an email address and password and then click Login, as shown in Figure B.1:

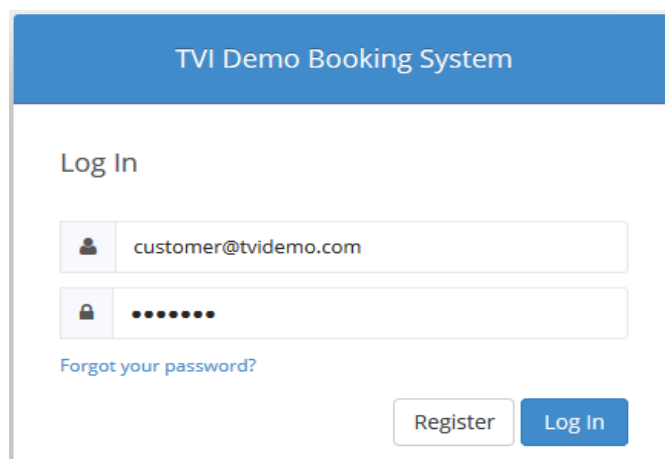
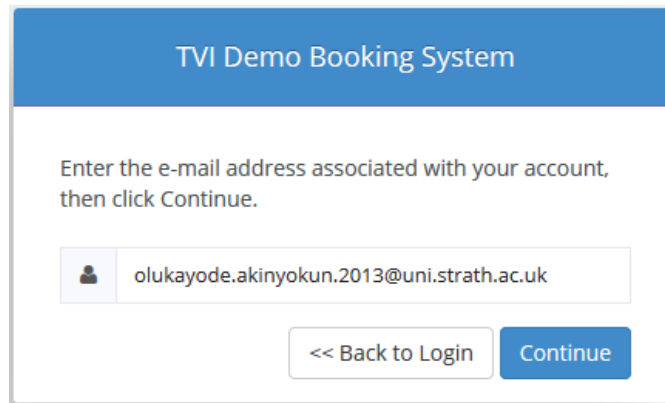


Figure B.1: Logging into the system

If a user forgets the password, the user would have to click on the “Forgot Password” link below the password field to change the password to a new one. More specifically, when the user clicks of the link, a new form (shown below in Figure B.2) appears prompting the user to enter his/her email address in order to proceed with the password reset process.



The screenshot shows a web form titled "TVI Demo Booking System". The form contains the instruction: "Enter the e-mail address associated with your account, then click Continue." Below this is a text input field containing the email address "olukayode.akinyokun.2013@uni.strath.ac.uk". At the bottom of the form are two buttons: "<< Back to Login" and "Continue".

Figure B.2: Email verification form

Once the user clicks “Continue”, the system checks the database to confirm that the password exists and thereafter it sends an email with a password reset link to the user. This link contains a token that expires after 24 hours. See Figure B.3 for the email that contains the password reset link.

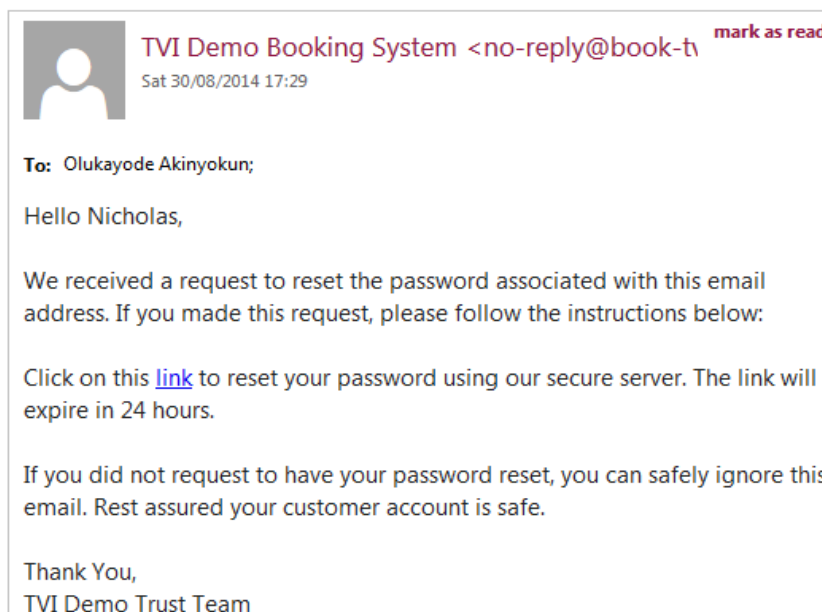
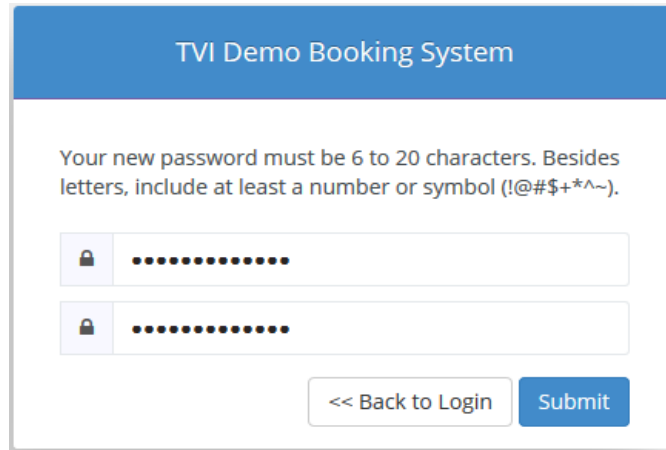


Figure B.3: Reset password email

When the user clicks on the link, the system redirects the user to a webpage (shown in Figure B.4) that allows the user to create a new password. The link embedded in the email contains a token that is used to verify that the message has not been received for more than 24 hours.



The screenshot shows a web form titled "TVI Demo Booking System". Below the title, there is a text instruction: "Your new password must be 6 to 20 characters. Besides letters, include at least a number or symbol (!@#\$+*^~).". There are two password input fields, each with a lock icon on the left and a series of dots representing the password. At the bottom of the form, there are two buttons: a white button with a left-pointing arrow and the text "<< Back to Login", and a blue button with the text "Submit".

Figure B.4: Reset password form

Appendix B.1: Customer User Guide

Once a customer submits the email address and password, the system checks the database to confirm that the customer's account exists. If this condition evaluates to true, the system loads the sidebar menu options meant for customers (see Figure B.5) and then, it goes on to display a booking form.

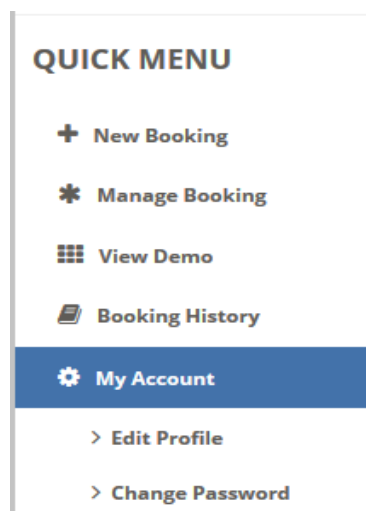


Figure B.5: Customer sidebar menu

For a customer to book a camera, the customer have to complete the booking form by specifying the category of the desired camera, the desired camera itself, the date of demonstration, the time the

customer wants the demonstration to start, as well as the duration of the demonstration. See Figure B.6 for the booking form.

The image shows a web form titled "New Booking" with a close button (x) and a refresh button (c). The form contains the following fields:

- Product Category: TVI Integrated Cameras
- Product: TVI MC350 (Minicam)
- Date: 01-09-2014
- Start Time: 11:00 AM
- Duration: 60 Minutes

A blue "Book Now!" button is located at the bottom of the form.

Figure B.6: The booking form for customers

Once the booking details have been saved in the database, the system sends an automated email to the customer to confirm that the booking has been made. This email contains the booking reference, the camera booked, date of demonstration, as well as the start and end times of the demonstration. See Figure B.7 for an example of the booking confirmation email.

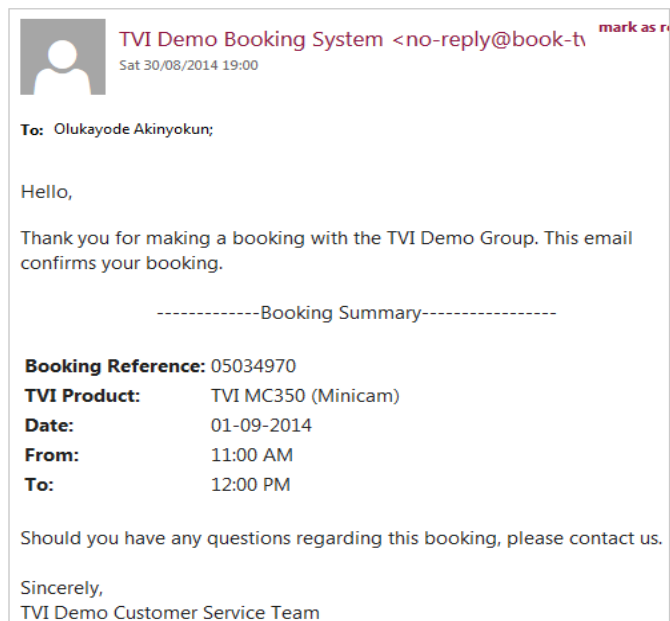


Figure B.7: Booking notification email

Under certain circumstances, the camera that a customer wants to book will not be available because another customer has already booked the camera for use during the same timeslot that the new customer wants. To solve this problem, the system would prompt the customer to select another timeslot from a list of all reserved and available timeslots that has been generated by the scheduler. See Figure B.8 for the list of available timeslots page.

Time Slots	Status
9:00AM	RESERVED
10:00AM	BOOK NOW
11:00AM	RESERVED
12:00PM	BOOK NOW
1:00PM	BOOK NOW
2:00PM	BOOK NOW
3:00PM	BOOK NOW
4:00PM	BOOK NOW

Figure B.8: List of available timeslots

Furthermore, the customer can manage all bookings that have been made by clicking on “Manage Booking” in the sidebar. The “Manage Booking” link generates a table (as shown below) that lists all the cameras that have been booked for demonstration at a later date, the date of the demonstration, as well as the start and end times of the demonstration.

Manage Booking

2 Records found

5 records per page Search:

SN	Product	Date	From	To	
1	TVI WMV (Polecat)	01-09-2014	11:00 AM	12:00 PM	Amend Cancel
2	TVI MC350 (Minicam)	03-09-2014	1:00 PM	1:30 PM	Amend Cancel

Figure B.9: Manage booking table – (customer)

For each booking listed in the table, there are two buttons (notably, the “Amend” and “Cancel” buttons) that a customer can use to amend and cancel a booking, respectively. The “Booking History” link on the sidebar functions in a similar like the “Manage Booking” link, although the only difference is the “Booking History” table generates all the bookings that have been made by a customer in the past. Aside from booking, a customer can also use the queuing functionality of the system to queue for access to a particular camera. In order to do this, the customer will have to select the desired camera and duration on the queue form, as shown below in Figure B.10.

View Demonstration Stream

Product:

Duration:

Figure B.10: Queue form for customers

Once the system processes the queue form and determines that the product selected by the customer is available for the entire length of the duration specified, a countdown timer (shown below) will set off. The customer can choose to stop the countdown timer before the time elapses.

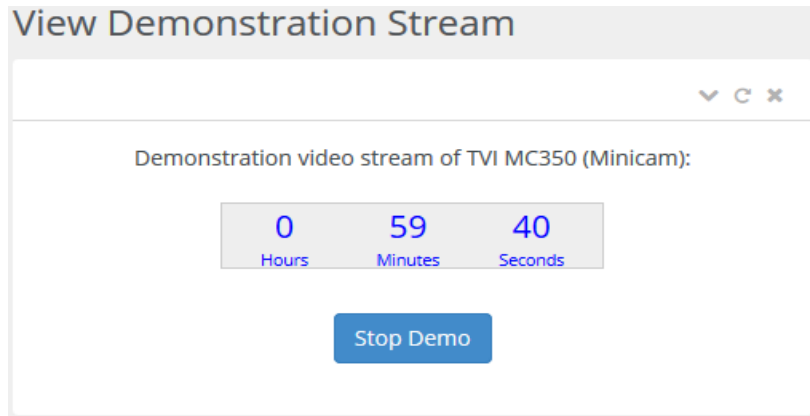


Figure B.11: Demo page for customers

On the other hand, if after processing the queue form, the system determines that the product and timeslot that would be allocated to the customer has already been booked, the scheduler will automatically select the nearest available timeslot for the customer and afterward, the system will prompt the customer to join the waiting list (which is essentially a priority queue) of customers expecting to access the same camera. If the customer decides to join the waiting list, the system would display an estimate of how long the customer would have to wait on the queue before gaining access to the desired camera. However, if the customer decides not to join the waiting list, the system will redirect the customer back to the queue form. See Figure B.12 for the waiting list page that shows the estimated wait time.

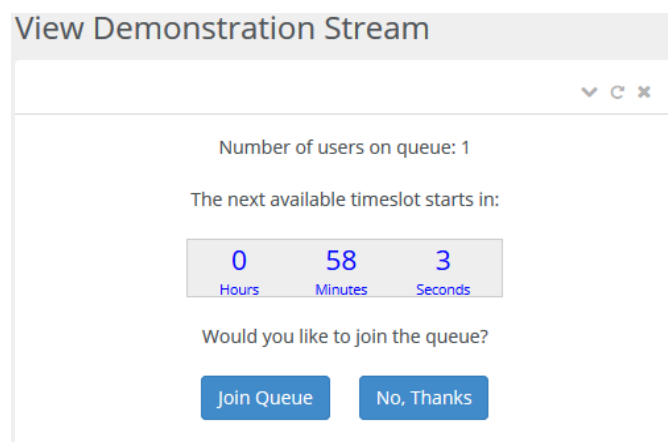


Figure B.12: Waiting list page for customers

Moreover, apart from booking and queuing for cameras, customers can update their profile information and change their passwords. These are achieved by clicking on the “Edit Profile” and “Change Password” menus, respectively. See Figure B.13 and B.14 for the forms used for changing passwords and updating customer’s profile information.

The screenshot shows a web form titled "Update Account Information". At the top right of the form area are three small icons: a downward arrow, a refresh symbol, and a close symbol. The form contains the following fields and values:

- Text input: "Doe"
- Text input: "John C."
- Text input with user icon: "customer11"
- Text input with email icon: "customer@tvidemo.com"
- Text input with briefcase icon: "Alpine Industries Ltd."
- Text input with briefcase icon: "Manager"
- Text input with phone icon: "+234-567-8766"
- Text input with globe icon: "Iceland" (with a dropdown arrow on the right)
- Blue button: "Save Details"

Figure B.13: Customer’s account update form

The screenshot shows a web form titled "Change Password". At the top right of the form area are three small icons: a downward arrow, a refresh symbol, and a close symbol. The form contains the following fields and values:

- Password input field with lock icon and masked characters: "....."
- Password input field with lock icon and masked characters: "....."
- Password input field with lock icon and masked characters: "....."
- Blue button: "Save Password"

Figure B.14: Change password form

Appendix B.2: Administrator User Guide

For an administrator to use the booking system, the administrator has to login. Once the administrator submits the email address and password, the system checks the database to confirm that the

administrator’s account exists. If this condition evaluates to true, the system loads the sidebar menu options meant for the system administrator (see Figure B.15) and then, it goes on to display the administrator’s dashboard (See Figure B.16)

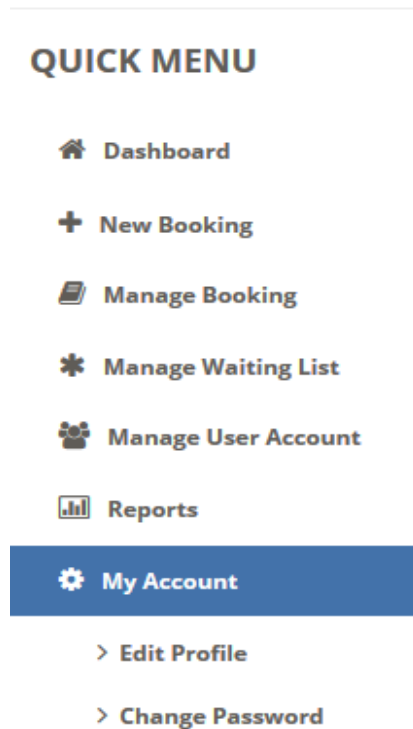


Figure B.15: Administrator sidebar menu

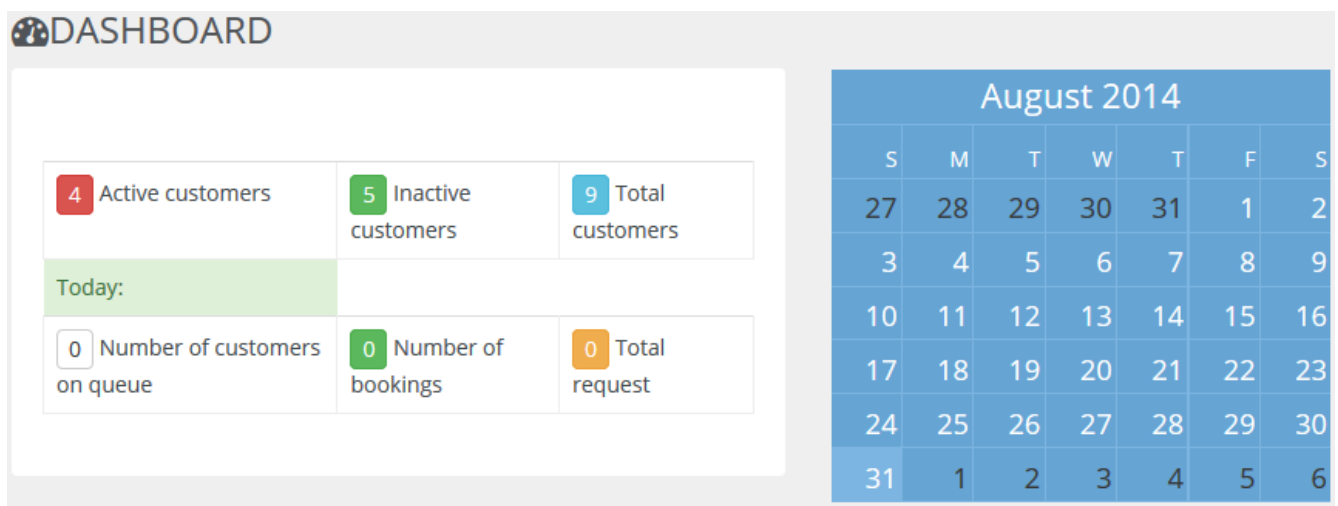


Figure B.16: Administrator dashboard

The administrator’s dashboard displays the total number of customer accounts (both active and inactive) in the database, as well as the total number of bookings and customers on the queue in a day.

If the administrator wants to book a camera for a customer, the administrator have to complete a modified version of the booking form which contains a drop-down list that shows all the registered customers. In this form, the administrator specifies the category of the desired camera, the desired camera itself, the customer, date of demonstration, the time the customer wants the demonstration to start, as well as the duration of the demonstration. See Figure B.17 for the administrator’s booking form.

The image shows a web form titled "New Booking" with a light gray header. Below the header, there are six rows of form fields, each with a label on the left and a control on the right. The controls are a mix of text boxes and dropdown menus. At the bottom of the form is a blue button with white text that says "Book Now!".

Product Category:	TVI Integrated Cameras
Product:	TVI WMV (Polecat)
Select Customer:	Phillips Trevor
Date:	01-09-2014
Start Time:	09:00 AM
Duration:	30 Minutes

Figure B.17: Administrator booking form

Once the booking details have been saved in the database, the system sends an automated email to the customer selected by the administrator to confirm that the booking has been made.

If the administrator wants to cancel a booking that has been made by a customer, the administrator clicks on the “Manage Booking” link on the sidebar. This link displays a table (shown below) that allows the administrator to cancel an existing booking.

Manage Booking						
3 Records found						
5 records per page		Search: <input type="text"/>				
SN	Full Name	Product	Date	From	To	
1	Phillips Trevor	TVI WMV (Polecat)	02-09-2014	10:30 AM	11:30 AM	Cancel
2	Madrazo Martin	TVI WMV (Polecat)	03-09-2014	2:00 PM	3:00 PM	Cancel
3	Weston Devin	TVI MC350 (Minicam)	02-09-2014	9:00 AM	9:30 AM	Cancel

Figure B.18: Manage booking table – (Administrator)

Besides making and cancelling bookings, the administrator can also remove the customers that are on the waiting list for a particular camera. This is achieved by using the “Manage Waiting List” table, shown below in Figure B.19

Manage Waiting List						
2 Records found						
5 records per page		Search: <input type="text"/>				
SN	Full Name	Product	Date	From	To	
1	Yetarian Simeon	TVI RDK-U	31-08-2014	1:41 PM	2:11 PM	Cancel
2	John C. Doe	TVI WMV (Polecat)	31-08-2014	1:41 PM	2:41 PM	Cancel

Figure B.19: Manage waiting list table

Moreover, if the administrator wants to update or delete a customer’s account, the administrator will have to click on the “Manage User Account” link on the sidebar menu. Upon clicking this link, a table that contains all user accounts (both customer and administrator) that have been created on the system is generated. From this table, the administrator can then proceed to click on the “EDIT” or “Delete” link to update and delete user accounts respectively. See Figure B.20 for the “Manage User Account” table.

Manage User Account

10 Records found

5 records per page Search:

EDIT	Full Name	Email Address	Phone	Company	Country	Status	Delete
EDIT	Akinyokun Nicholas	captain.nicholas@gmail.com	07035227990	University of Strathclyde	Nigeria	Active	
EDIT	Lukens Rickie	rickie.lukens@life.invader.com	123-555-0160	Life Invader Inc.	United States	Active	Delete
EDIT	Phillips Trevor	trevor.phillips@lifeInvader.com	425-555-0170	Trevor Phillips Enterprises	Canada	Active	Delete

Figure B.20: Manage user account table

If the administrator clicks on the “EDIT” link, a slightly modified version of the form customers use to update their account is displayed. This form (shown below) includes new form fields that the administrator can use to set the user type and status of a customer to “Administrator” and “Inactive” respectively.

Update Account Information

Rickie Lukens

@rickieLuck

rickie.lukens@life.invader.com

Life Invader Inc.

Software Developer

123-555-0160

United States

Customer

Active

[Save Details](#)

Figure B.21: Account update form – (Administrator)

Appendix C: Use Case Diagram

This section contains the use case diagram of the booking system, as shown in Figure C.1. The labeled stick figures on the diagram represent actors and they indicate the roles that a user can play while interacting with the booking system. Similarly, the scheduling subsystem of the booking system has been depicted by a rectangle with the <<actor>> notation.

The use cases are represented by an oval and a line drawn from an actor to a use case depicts an association. This association typically represents two-way communication between a use case and the actor that interacts with it.

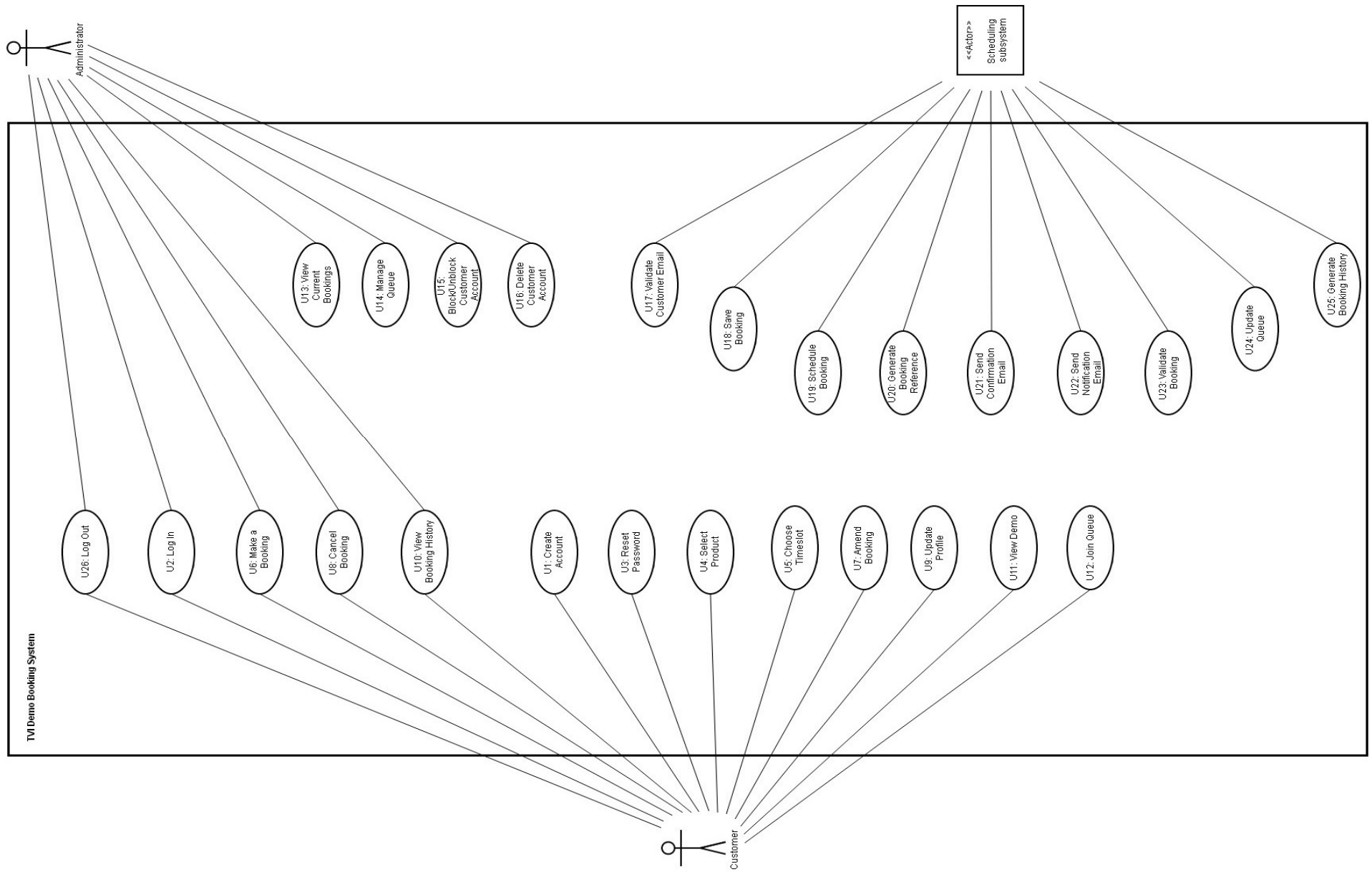


Figure C.1: Use case diagram of the booking system

Appendix D: Class Diagram

This section contains the class diagram of the booking system, as shown in Figure D.1:

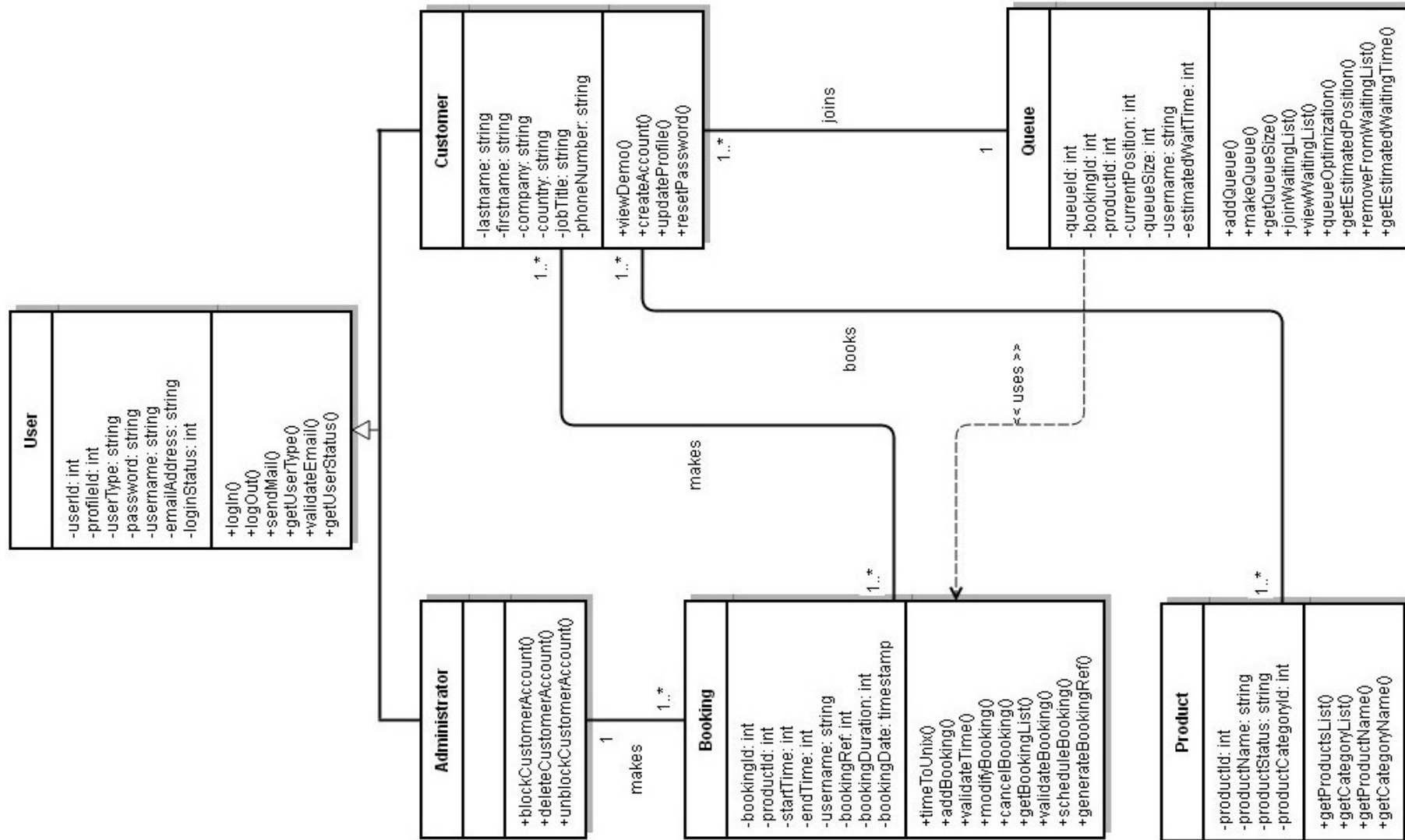


Figure D.1: Class diagram of the booking system

Appendix E: Results of Load Tests

This section contains the load tests that were used to assess the performance of the booking and queuing functionality of the system. The processes that were followed in performing these tests have been described in the Evaluation chapter.

Number of Users	Waiting Time (ms)	Response Time (ms)	Throughput (KB/s)
2	4.11	11.47	6
3	7.92	15.67	11
5	12.81	21.32	31
7	19.18	26	44
11	25.79	34.78	70
12	27.25	38.67	73
14	31.28	43.45	97
15	33.13	49.16	103
17	35.8	54.12	107
19	41.2	59.2	116
20	43	61	117
22	46.42	65.3	120
24	46.67	67.15	127
25	48	69.5	131
27	51.65	73.56	134
29	54.23	81.41	137
31	56.1	87.97	139
33	57.34	92.83	142
34	62.87	97.19	144
35	65.34	120.5	147
37	77.21	132.56	153
39	79.46	151.1	159
41	83.56	166	164
43	91.59	172.3	171
45	99.6	185.7	176
46	109.9	202.45	181
48	126.57	217.78	184
50	141.89	259.8	186

Table E.1: Result of the load test on the booking algorithm

Max Users	Waiting Time (ms)	Response Time (ms)	Throughput (KB/s)
2	42.28	20.48	7
3	145.76	68.67	13
5	193.11	97.94	32
7	214.87	196.52	49
11	301.11	302.86	62
12	393.79	352.32	73
14	503.98	412.1	85
15	601.23	438.33	92
17	734.75	601.96	104
19	901	857.34	118
20	1211.77	873.4	127
22	1302.84	1021.45	133
24	1466.02	1278.16	141
25	1578.31	1304.46	143
27	1760.11	1458.9	147
29	1823.3	1600	173
31	2004.35	1793.41	191
33	2152.56	1923.56	217
34	2431.19	2032.14	221
35	2675.4	2132.14	230
37	2800.92	2345.73	243
39	2917.56	2607.11	262
41	3127.17	2832.5	279
43	3417.63	3054.24	293
45	3701.24	3254.24	301
46	3793.11	3403.78	314
48	3867.9	3587.31	336
50	4012.14	3693.87	385

Table E.2: Result of the load test on the queuing algorithm

Appendix F: Email Invitation

This section presents the email invitation that was distributed to recruit participants for the booking system user evaluation.

SEEKING PARTICIPANTS IN A USABILITY STUDY

Hello,

I am looking for participants to evaluate the usability of a web application I have developed as part of my MSc research project.

The evaluation takes around 10 minutes to complete and will provide invaluable feedback that can be used to improve the usability of the system.

If you are interested in participating, your help would be much appreciated - please click on the attached information sheet to read the instructions on how to use the system.

Moreover, please note that participation is voluntary and that you are free to withdraw from the study at any time, without having to give a reason and without any consequences.

Thanks,
Nicholas

Appendix G: Participant Information Sheet

This section presents the information contained in the participant information sheet that was used by participants for assessing the usability of the system.

Participant Information Sheet

Researcher: Akinyokun Olukayode Nicholas
Researcher's email: olukayode.akinyokun.2013@uni.strath.ac.uk
University: University of Strathclyde
Department: Department of Computer and Information Sciences
Title of Study: User Evaluation of TVI Demo Booking System

1.1 What the study will involve

The purpose of this evaluation is to assess the usability of a booking system, developed as part of my MSc research project. The evaluation will take no longer than 10 minutes to complete and will involve following a set of instructions detailed below on how to use the system to perform some tasks.

After performing these tasks, you will be asked to fill out a questionnaire. This questionnaire is completely anonymous and all data gathered will be held confidentially and erased on completion of the project.

Please note that you have the right to withdraw if you wish and that participation is voluntary.

1.2 Instructions

1. Open your browser and go to:

<https://devweb2013.cis.strath.ac.uk/~iyb13184/book-tvi-demo/index.php#>

2. **Log In**

Proceed to login into the system with these credentials:

- Email: eval@tvidemo.com
- Password: test123

3. **Make a booking**

- Upon login, you'll see a form for making a new booking. Fill out all the form fields (they are mandatory)

- Then, click the "Book Now!" button

4. **Modify Booking**

- Next, click on the "Manage Booking" menu on the sidebar. You will see a record of the booking you have just made
- Click on the "Amend" button
- Change the date and time of the booking
- Click on the "Save" button to save these changes

5. **Cancel Booking**

- Click again on the "Manage Booking" menu
- This time, click on the "Cancel" button

6. **Log out**

Click on the "Log Out" button on the top-right corner of the screen.

Thank you for your time. Please click on this link to complete the questionnaire:

<https://www.surveymonkey.com/s/YDKJ8NT>

1.3 **Ethics Approval**

Ethical approval of studies has been obtained for this research project. If participants have any queries or concerns with regards to the ethics of this study, they should contact the researcher. Should participants wish to raise any ethical queries or concerns with a party other than the researcher, they should contact Prof. Ian Ruthven at: ian.ruthven@strath.ac.uk

Appendix H: User Evaluation Questionnaire

This section contains a screenshot of the user evaluation questionnaire that was administered to participants.

1. Please indicate your gender?

- Male
 Female

2. Which category below includes your age?

- 18-24
 25-34
 35-44
 45-54
 55+

3. Please indicate the level of your agreement with the following statements.

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree
I think that I would like to use the system frequently.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system unnecessarily complex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought the system was easy to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would need the support of a technical person to be able to use the system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the various functions in the system were well integrated.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought there was too much inconsistency in the system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I would imagine that most people would learn to use the system very quickly.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system not very intuitive.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think the system response time is satisfactory	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I needed to learn a lot of things before I could get going with the system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure H.1: User evaluation questionnaire

4. Overall, how would you rate the user-friendliness of the system?

- Very good
- Good
- Fair
- Poor
- Very poor

5. Did you encounter any problem while using the system?

- Yes.
- No.

If Yes, please state the nature of the problem?

6. Do you have suggestions that could improve the system?

Figure H.1: User evaluation questionnaire (contd.)

Appendix I: Database Attributes

This section describes the attributes that have been used in the database of the booking system.

Users Table

user_id	This is the primary key that uniquely identifies each user account. It auto increments by 1 once a new account has been created.
username	This attribute constitutes the unique username chosen by each user during registration.
password	This is the MD5 hash value of each user's password.
user_type	This specifies the category of user; which can either be a customer or an administrator.
email	This constitutes the email address of each user. The email address is verified once a user creates an account and subsequently, it is used alongside the password to log in
created	This represents the date and time a user account was created.
user_status	This indicates whether a user's account has been activated or not. It can either stores 'A' which means 'Active' when a user's email address has been verified or 'I' which stands for 'Inactive' if the email address has not been verified.
signed_in	This attribute indicates whether a user is currently signed in to their account or not. It is a Boolean data type and will hold only integers: 0 or 1, with 1 indicating that a user is currently signed in while 0 indicates otherwise.

Table I.1: Attributes of the user table

Products Category Table

pcid	This is the primary key that uniquely identifies the category of cameras.
category_name	This attributes represents the name of each camera category

Table I.2: Attributes of the products category table

Products Table

pid	This is the primary key that uniquely identifies each camera.
product_category_id	This attribute uniquely identifies a camera's category; which could either be TVI Integrated Cameras or TVI Hardware Encoders.
product_name	This is the name of each camera. An example is the Digital Barriers TVI MC350 'Minicam'
product_status	This attribute indicates whether a camera is available or not.

Table I.3: Attributes of the products table

Profile Table

profile_id	This is the primary key that uniquely identifies each user's profile. It auto increments by 1 once a new account has been created.
username	This attribute constitutes the unique username chosen by each user. Here, it is used to establish and enforce a link between the Users Table and the Profile Table.
fname	This is the first name of each user.
lname	This is the last name of each user.
phone	This constitutes a user's telephone number.
company	This is the name of the company that a customer is affiliated to.
job_title	This is the job title of a customer.
country	This is the country that a customer's company is located.

Table I.4: Attributes of the profile table

Password Reset Table

ref_id	This attribute is used as an alternate key to identify only one row in the table.
username	This attribute constitutes the unique username of each user. Here, it is used to establish and enforce a link between this Table and the Password Reset Table.
token	This attribute is used to verify that the password reset link sent via email to a user is valid and belongs to the user.
token_status	This attribute is used to check whether a token is valid or has already expired. In the booking system, any token generated when a user requests to reset his/her password automatically expires after 24 hours.
timestamp	This is the current system date and time returned by the MySQL <code>NOW()</code> function.
duration	This is total duration (in minutes) of the timeslot that has been booked by a user.

Table I.5: Attributes of the password reset table

Queue Table

queue_id	This attribute uniquely identifies each user's request on the priority queue.
username	This attribute constitutes the unique username chosen by each user. Here, it is used to establish and enforce a link between the Users Table and the Queue Table.
product_id	This constitutes a unique identifier of the camera that has been requested.
booking_id	This attribute uniquely identifies the bookings made by users. Here, it is used as a foreign key to link the Booking Table and the Queue Table.
timestamp	This is the current system date and time returned by the MySQL <code>NOW()</code> function.

Table I.6: Attributes of the queue table

Booking Table

booking_id	This is the primary key that uniquely identifies each booking made by users. It auto-increments by 1 for each new booking.
username	This attribute constitutes the unique username chosen by each user. Here, it is used to establish and enforce a link between the Users Table and the Booking Table.
product_id	This attribute uniquely identifies each camera that has been booked. Here, it is used to establish and enforce a link between the Products Table and the Booking Table.
start_time	This is the time that a booking made by a user is scheduled to start.
end_time	This is the time a user's booking will end.
duration	This is total duration (in minutes) of the timeslot that has been booked by a user.
booking_ref	This is a unique numeric string automatically assigned to each user's booking.
timestamp	This is the current system date and time returned by the MySQL <code>NOW()</code> function.
status	This attribute indicates whether a booking has not been cancelled.
type	This attribute indicates whether a camera has been reserved using the booking functionality – ('B') or the queuing functionality - ('Q')

Table I.7: Attributes of the booking table