Student D
BSc (Hons) Computing (Information Systems Development) Year 4
Matric No: 20081xxxx

Honours Project Final Report

Project Title: Development and Test of a Room Booking Application for BlackBerry Playbook Tablet

Project Supervisor: Brian Hainey
Second Marker: Richard Foley

Submitted for the Degree of
BSc in Computing (Information Systems Development), 2011-2012

"Except where explicitly stated all work in this document is my own"

Signed:______________________ Date:___________

Abstract

Tablet computers are a rapidly expanding category of mobile computing, challenging user reliance on other mobile computing devices, such as laptops, by offering novel methods of usability and a wide range of interesting applications. The use of tablet computers in industry is starting to rise, with tablet applications being used to solve various technical problems and improve "traditional" methods. A large facilities management company is interested in exploring the capabilities and benefits of tablet computers for use within their company in order to improve one such "traditional" method. The aim of this project was to determine whether or not it is possible to create a room booking application for BlackBerry PlayBook tablet which effectively replaces the traditional room booking method currently in use within the company.

In order to understand the problem domain and to highlight known issues regarding the applications of the project, a literature investigation was undertaken. This involved a review of HCI issues regarding tablet and touch based devices. In addition to this, a technical review was undertaken in order to determine the best technologies for fulfilling the project implementation. Following this, interviews were conducted with Company X, which highlighted their interests in the application and outlined the basic project requirements.

Using a develop and test approach supported by human computer interaction (HCI) principles, a client-server application was developed which allows users to book an available room via BlackBerry PlayBook tablet. The project involved the development of a front-end BlackBerry PlayBook application with a supporting WCF service application which allows for "point-to-point" communication via "RESTful" services. Following the application implementation, qualitative evaluation methods were used to assess the effectiveness of the project and provide project validation.

The results of the testing and evaluation indicated that the project was successful in addressing the issue of improving a "traditional" method. In addition to this, the project also highlighted a potential development method, and web communication architecture which may prove beneficial to other software developers investigating the development of an application in this relatively new domain.

Acknowledgements

I would like to thank my supervisor, Brian Hainey, for his valuable support and advice throughout this project.

I would also like to thank xxxxx xxxx and the staff of xxxxx xxxxxxx, for allowing me to undertake this project and for providing continued inspiration, which helped improve my technical skills.

Finally, I would like to thank my friends and family for their continued support throughout my academic career.

Contents

Table of Figures

Table of Appendices

1 Introduction

The introduction section will introduce the topic area as well provide a basic overview of the aims of objectives of the project.

1.1 Background

Tablet computers are a rapidly expanding category of mobile computing, challenging user reliance on other mobile computing devices, such as laptops, by offering novel ways of usability with a wide range of interesting applications. Since the arrival of the iPad, there has been a rising interest in modern tablet computers, or "Post-pc" devices, with many manufacturers beginning to offer unique tablet computer solutions. A large investment bank, Morgan Stanley, believed that in 2011, combined shipments of smartphones and tablets would overtake those of personal computers (The Economist, 2011). It is also expected that this growth will directly affect the Laptop PC segment (eTForecasts, 2011). A study published by Forrester Research indicates that tablet sales will grow from 10.3 million in 2010, to 44 million by 2015, which will overtake laptop sales by approximately 5 million units (Indvik, 2011).

With the rise in popularity of tablet computers, we are beginning to see tablet applications being used to replace their traditional counterpart. Graham (2010) details an application currently available for tablets which removes the traditional cash register from the workplace, allowing for various functions such as stock checking, report generating and payment processing. Many companies are also starting to use tablet computers to reduce paper usage within the office (CBC, 2010). There are many more applications which are taking over from their traditional counterpart such as workshop manuals, books, medical directories and patient access, and inventory management (Smithie, 2011). An example of this is the use of tablet computers in aviation. Apple iPad's are being adopted as an alternative to the required reference material required by pilots (Murphy, 2011). The use of the iPad removes the need for 40lbs of required in-flight reference materials, in favour of electronic editions, and allows the pilots to have much quicker access to the information.

Paper use for information storage is highly inefficient. Paper archives are costly to maintain, are poorly searchable and are hard to access from a distance (Andersen, 2008). Ashby (2011), states that office workers spend up to 40% of their time looking for hard copy information, 22.5% of documents are lost or misplaced, 30% of documents contain obsolete information and 80% of documents are never referenced again. There are many digital solutions available which can reduce the use of paper for information storage within a business environment. One solution to this problem is the tablet computer. Tablet computers offer the portability of mobile applications with the function of a larger computer. This is supported by Andersen (2008) who states that replacement for paper as portability will come driven by mobile communications. The author then states that information access and communication aspects of electronic note-taking, will overpower the portability and simplicity issues of paper use. Pachner (2011) also states that "The advance in electronic communication is not just one of the factors influencing demand for printing-paper grades, it is the factor".

A large facilities management company (Company X) is interested in exploring the capabilities of tablet computers, which may be used within their company to improve current “traditional” systems. Since the company is a member of the BlackBerry Alliance, they have selected the BlackBerry PlayBook as their preferred tablet computer as it is compatible with the existing server architecture (BlackBerry Enterprise Server) and other company devices. The company is interested in testing out the functionality of the PlayBook tablet, by creating a client-server application which replaces current room booking methods at the company reception. Currently, the process of booking a room involves the receptionist searching for available rooms, offering the choices to the inquirer, and reserving the room based on their selection. Since the company has a single receptionist, the process of booking a room may prove difficult at times as the receptionist may be dealing with visitors, other employees or handling phone calls.

1.2 Project Outline

This project involved the development and test of a BlackBerry PlayBook application and Windows Communication Foundation (WCF) service application, which allows staff and visitors to book available rooms. The room booking application consists of a front-end application, created for BlackBerry Playbook with which the users will interact, and a WCF service application back-end which will handle data transactions and methods.

1.3 Research Question

Can a room booking application be developed for BlackBerry PlayBook tablet which will improve existing room booking methods, and provide improved access and usability for visitors and employees?

1.4 Aims and Objectives

The aim of this project was to determine whether or not it is possible to create a room booking application for BlackBerry PlayBook tablet which would improve access and usability of an existing room booking process used within a large corporation. Based on the aims of the project, the following objectives were identified which provided a clear working methodology for the project. Objectives have been defined for both the literature review and for the primary method.

1.4.1 Objectives of the Literature Review

By creating specific objectives for the literature review, it was expected that many of the key issues, potential problems and possible solutions related to the subject area would be identified. The objectives met through the literature review are:

1. **Overview of the use of tablet computers and its applications**

   The purpose of this objective was to understand the growing tablet computing market and how tablets are being used to improve traditional tasks. As tablet computers become more popular, so does their presence in everyday life. There are various applications available for tablet devices which essentially eradicate the need for its traditional counterpart. An investigation of these applications helped provide an understanding of the novel uses of tablet computers, justifying the relevance of this project.

2. **Investigate the usability of touch screen devices**

   This research objective aided in the understanding of how mobile devices are used and provided guidance when designing the user interface components of the application. This section covered the research of mobile devices such as smartphones and tablet computers, and focussed on identifying any key issues which could affect the design and development of the application.

3. **Technical Review**

   Since this is a develop and test project, it is important to have a sound understanding of the application domain prior to beginning development. This literature objective involved the analysis of software development methodologies and software development technologies in order to identify the methods and technologies best suited for the requirements of the project.

1.4.2 Objectives of the Primary Method

The following objectives had been identified in order to aid in the completion of the primary method.

1. **Capture the necessary requirements in order to determine the required functionality of the application**

   This objective involved capturing all of the necessary requirements in order to effectively plan the development, and in order to determine the functionality required for the application to meet the quality standards expected.

2. **Conduct practical experiments in order to determine PlayBook capabilities and to obtain reusable code**

   Practical experiments were conducted prior to the project start in order to prove concepts and determine the capabilities of software solutions and the PlayBook tablet. These experiments will provided reusable code components, which were be refined and included in the final build. An example of this is the jqGrid which was incorporated in the early PlayBook iterations.

3. **Develop and test the room booking PlayBook application and WCF service application**

This is the main project method, which was conducted as a series of application iterations. The development and testing of the application was supported by the evidence of the literature review.

4. **Evaluate the final application and discuss the findings**
   This is the final stage of the application which involved assessing the outcome of the project and discussing the findings of the project in relation to the original research question and hypotheses.

1.5 Hypotheses

Based on the literature review and practical experiments conducted, the following hypotheses had been created.

1. A room booking application on a tablet computer will benefit the employees of Company X by consolidating the process of room booking within one application whilst eradicating the need to consult a third party.

This was evaluated by HCI questionnaire, where by a sample of users consisting of students were presented with screenshots and an explanation of the application and were asked to rate the application in terms of design and usability.

2. Users will be more inclined to use the “View map of rooms” booking method as opposed to the “View list of rooms” method due to the high level of interaction and novelty.

This was also evaluated under the HCI questionnaire, where by the students were asked if they preferred the original “list” view or the final “map” view when viewing available rooms.

1.6 Report Structure

This section details the remaining sections of this report. Each section will include a brief overview of the section as well as highlight the main issues identified.

Literature Review

This section details the literature investigation which was undertaken in order to gain a better understanding of the problem domain. The main areas of investigation were an overview of the use of tablet computers and its counterparts, an investigation of the usability of touch screen devices and a technical review. By completing the literature review, several issues were highlighted which influenced the problem analysis and implementation of the main methods. In addition to this, the literature review also indicated the most suitable development technologies and planning methods which ensured the projects successful completion.

Problem and Systems Analysis

This section details the problem and systems analysis process which was undertaken in order to develop the requirements of the project and learn more about the potential issues which could affect the project outcome. The problem and systems analysis consisted of several interviews with Company X, which highlighted their interest in the project and also aided in the requirements capture for the project.

Design and Implementation

The design and implementation section details the main methods and development which was conducted throughout this project. As the design and implementation was completed over several iterations of two main project applications, this section details all of the implementations and the significant findings of each which influenced the later iterations. Included in this section are the testing procedures which were used to test the project against the captured requirements.

Evaluation

The evaluation section details the methods which were employed in order to evaluate and test the project. In order to test the hypotheses and also highlight any specific weaknesses of the application, an HCI questionnaire was completed by several students. This section details the testing and evaluation procedures which were used to assess the project. In addition to this, an application "walkthrough" is given, which highlights the basic functionality of the application regarding a typical "booking" procedure.

Discussion and Conclusions

The final section, discussion and conclusions, discusses the general findings of the project and the outcome of the project evaluation procedure in relation to the research question and hypotheses. Included in this section is a discussion of the limitations of the project and future work, which indicates potential future developments which may be influenced by this project.

2 Literature Review

The literature review will provide the knowledge required for successful completion of the project, highlighting the key issues, potential problems and possible solutions to many of project elements. As stated in section 1.4.1, the following objectives aim to focus the literature review around the problem area:

- Overview of the use of tablet computers and its applications
- Investigate the usability of touch screen devices
- Technical review

2.1 Overview of the use of tablet computers and its applications

Since the iPad was introduced in 2010, there has been a rising interest in tablet computers. Tablet computers are not a new concept however, as there have been examples as early as the 1980s. These tablet computers typically featured pen based input, such as that found on the Microsoft Tablet PC. As well as featuring resistive touch screens, these tablet computers featured modified versions of desktop operating systems, such as Microsoft Windows XP (Tablet PC Edition). Since then, tablet computers have evolved into what has become known as the “Post-pc” device.

Tablet computers in the “Post-pc” era are quite different from their earlier counterparts. Most of the tablet computers available today feature large capacitive touchscreens and usually run an augmented form of mobile operating system, typically that of smartphones such as Android and iOS. This has resulted in tablets sharing some of the key characteristics of smartphone devices such as mobility, touch optimisation, power consumption and usage patterns (Morgan Stanley, 2011). However, as similar as tablets and smartphones may be, there are definitive differences which must be considered during development (see section 2.2).

As tablet computers become more popular, we are beginning to see them being used to improve “traditional” methods and tasks. There are numerous industries which are beginning to incorporate tablet computers into everyday life in order to improve the lives of employees and customers. Murphy (2011) discusses the use of table computers within the aviation industry to reduce the need for 40lbs of required in-flight documents as well as providing faster and more efficient document traversing. Another interesting application of tablet computers is in the restaurant industry. Patrizio (2011) details a system currently in use in the restaurant industry where paper menus are replaced with tablet computers. This system eradicated the need for a waiter to take your order, and instead everything was processed via wireless communication. This removal of a third party, the waiter in the case of the restaurant application, is similar in nature to this project in that the application will remove the need for a third party (the receptionist). An article written by Graham (2011) further supports this motivation by stating that you can look and see what you want instead of written descriptions, more information can be included and the advantage to the guest is speed of service. Although it is not directly related to the project application, the benefits of the restaurant ordering application described above are clear and consistent with the goals of this project.

2.2 Investigate the Usability of Touch Screen Devices

Since the application focuses heavily on touch screen interaction and user centred design, it is important to understand HCI principles and heuristics in order to develop an interface which is highly useable. There are a variety of usability factors which must be considered when approaching the development of a touch screen application. The aim of this literature objective is to identify and assess key heuristic and usability issues concerned with touch screen devices and their usability.

Touch Screen Interaction

Modern tablet computers and mobile devices feature multi-touch capacitive touchscreens which are responsive to user based touch via single and multiple-finger input (Han, 2006). Multi-touch technology allows for complex, gesture based input which can in turn allow for more complex user interfaces. There are a variety of touch gestures supported on most tablet devices. The core gestures for most touch commands are given as tap, double tap, drag, flick, pinch, spread, press, press and tap, press and drag, and rotate (Wroblewski et al., 2010).

While these user-based gestures can be used in a variety of ways to create interesting usability, the main focus of the application will be on interaction simplicity through minimalism. Nielsen (1994), states that every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. A benefit to simplistic design, as given by Chang et al. (2007), is that simplicity can help reduce the cost and features incorporated, while encouraging innovations in methods of accessing this information. This is supported by Berkun (1999), who states that simplicity does not mean a lack of functionality, it means a fast initial learning curve and consideration for the number of concepts the user needs to understand. Since the application will be introduced to the users without any formal training or previous interaction, minimising any learning curves which may appear is imperative in ensuring the application is accepted by the end users. This is supported by the Ten Usability Heuristics (Nielsen, 1994), particularly the heuristic "recognition rather than recall" which states that the users memory load should be minimised by making all objects, actions and options visible.

As the application will be located in a company headquarters, there will be a variety of users accessing the application, each with different abilities and as such the application must cater to them all. Microsoft define the concept of accessibility as ensuring that programs and functionality are easily available to the widest range of users, including those who have disabilities and impairments. As previously mentioned, tablet computers feature a wide range of command input gestures, but it more is important to design an application which can accommodate those with disabilities. For this reason it is expected that the application will largely utilise the simple tap, and double tap methods as defined by Wroblewski et al. (2010). This is supported by Saffer (2010) who states that anything beyond basic movements that use single fingers or the hand as a single entity may be challenging for some disabled users.

Touch Screen Usability Issues

When developing applications for touch screen devices, one of the main factors which must be taken into consideration is accuracy of input. Adams and Hooten (2010), state that precise selection is difficult in touch-based devices due to the size of the human finger. This is due to the changing contact area during finger movements. In order to ensure maximum usability of the planned buttons and small controls of the application, particular attention must be paid to the size of the buttons and control which will be implemented. This is supported by Lee and Zhai (2009) who found that the performance of finger operated touch screen buttons deteriorated when the size of the button falls below a fraction of the finger width. This can have an adverse effect on the application as buttons which are too small may frustrate the user and ultimately deter use of the application.

Screen size is an important factor to take into consideration regarding usability. Tablet computers boast larger screen sizes typically ranging from 5” to 10”. The device appointed for this project features a 7” screen with a resolution of 1024 x 600 (BlackBerry) which as described by Nielsen (2011), when discussing their similarity to 10” inch tablets and 3.5” mobile phones, are described as a “bit of both”. This may present a scalability issue should the application need to be optimized for a new device.

As the PlayBook tablet features a screen size of 7”, the usability of the tablet is not affected by size, as it would be on smaller platforms such as smartphones, but instead presents certain interaction issues which may be relevant to the development of the room booking application. Nielsen and Norman (2010), state that one of the issues with using larger screens is the opportunity for accidental selection and triggering of actions. They state that this happens on larger screens because the same hands which are necessary to hold the device can accidentally touch the screen. This may be even more of an issue with the smaller, 7” screen found on the BlackBerry PlayBook device as it may prove awkward to hold depending on user hand size.

2.3 Technical review

The objective of this section of the literature review is to identify the key issues stemming from the technical aspect of the project.

Investigate Agile Software Development Methodologies

Software development methodologies or life cycles can be defined as a framework which is used to structure, plan, and control the process of developing information systems (CMS, 2008). Sommerville (2011) states that there are four fundamental activities, present in all software processes. These are given as software specification, software design and implementation, software validation and software evolution. Each of these activities is comprised of many sub-activities which are typically represented in each of the life cycle models. These steps are conception, requirements gathering, design, coding, testing, release, maintenance and retirement (Dooley, 2011).

There are a variety of modern software development life cycles available which are optimised for different project types. It is important to identify a suitable life cycle so as to ensure the project aims and objectives can be catered to whilst ensuring maximum productivity. Boehm and Turner (2003), state that plan-driven methods are based on strong engineering principles placing emphasis on well-defined and documented processes. An example of software systems which may utilise this approach is aerospace and government systems (Sommerville, 2011). Since the project involves the development of a simple application, a lifecycle which accommodates the scope of the project is Agile software development. Agile software development is targeted at small software projects with a small number of developers (Dooley, 2011).

Agile software development is a software development lifecycle typically suited to projects which involve the customer and accommodate changing requirements (Sommerville, 2011). Agile methods were created in 2001, as a reaction to traditional software development methods, as an alternative to document driven, heavyweight software development processes (Cohen et al., 2003). This was known as the Agile Manifesto and consisted of the following:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

-Beck et al, 2001

This is further reinforced by the following principles behind the Agile Manifesto:

- Our highest priority is to satisfy the customer
- Welcome changing requirements
- Deliver working software frequently
- Business people and developers must work together
- Build projects around motivated individuals
- The most efficient and effective method of information is face to face conversation
- Working software is the primary measure of progress
- Agile processes promote sustainable development
- Continuous attention to technical excellence and good design enhances agility
- Simplicity—the art of maximizing the amount of work not done – is essential
- The best architectures, requirements and design emerge from self-organising teams
- At regular intervals, the team reflects on how to become more effective.

-Beck et al, 2001

The creation of the agile manifesto characterised the basic fundamental values of agile methods whilst distinguishing itself from other traditional software development lifecycles (Cohen et al., 2003). There are various methods of Agile software development available, such as Extreme Programming (XP), Scrum, Crystal family of methodologies, Rational Unified Process etc. each presenting advantages and disadvantages (Abrahamsson et al., 2002). A survey was conducted in 2010 in order to determine the most popular agile methods. The survey revealed that the three most popular methods were Scrum (58%), XP(4%) and Scrum/XP Hybrid (17%) (Szoke, 2011).

There are many benefits to adopting a Scrum framework. One major benefit is the use of User stories to determine functionality. User stories are described as “units of customer-visible functionality” (Power, 2011). Since the project revolves around the creation of a software application with the primary concern of satisfying the end user, user stories may provide an advantage in capturing the desired functionality. User stories are a simple method of capturing user functionality as Cohn (2004) describes, you “simply write down what you want to do, and then you do it”. This will be beneficial as it will allow for the creation of useable content without spending too much time planning the development.

Another significant benefit to the use of the Scrum framework is that it caters to small teams (Janoff, Rising 2000). Although there are several specific roles to fill in the Scrum development framework, these roles can be catered to by involving the stakeholders (Company X), which will give them a larger presence in the project. This will allow them to express their requirements more effectively as well as define the needs of the users who will be using the application.

PlayBook Development Technologies

The completion of this objective will provide a fundamental understanding of the BlackBerry PlayBook platform and the various development methods available. There are three major development methods available for the PlayBook Tablet OS. These are BlackBerry WebWorks SDK for Tablet OS, BlackBerry Tablet OS SDK for Adobe AIR and BlackBerry Native SDK for Tablet OS. As well as these options, there is a runtime available for Android applications. This allows developers to port Android applications to the PlayBook tablet, essentially offering a fourth method of development by using the Android SDK.

The BlackBerry WebWorks SDK for Tablet OS allows for the creation of rich internet applications based on popular web technologies such as HTML5, CSS and JavaScript (BlackBerry 2011). Web applications are described as websites that are specifically optimised for use on a smart device (Stark, Jepson 2012). This contrasts with native applications, which are typically installed on the device and have access to native device resource. There are many development benefits to using web technologies in mobile devices. Wroblewski (2011) states that one of the main benefits to web based mobile apps is accessibility as all major mobile operating systems support web applications. This is supported by Pimmel (2011) who states that the idea of the mobile web is “Design and develop once, deploy everywhere”. One obvious disadvantage to using web applications is limited access to hardware features (Stark, Jepson 2012).

Developing mobile web applications is largely based on technology preference. However, there are advantages and disadvantages to using the various technologies. The BlackBerry WebWorks SDK and BlackBerry Tablet OS SDK for Adobe AIR accommodate the use of HTML5, with the WebWorks SDK placing particular emphasis on it. HTML5 is the fifth major revision of the core mark-up language of the web, HTML (W3C 2011), and supports most of the features of HTML 4.01 (Freeman, Robson 2011). It is expected that almost all smartphones will support HTML5 by 2013 (Whitney 2011). This growth, as described by Whitney, is due to a need for rich applications to run cross platform. This is not an issue for this project as it involves the development of a standalone application for one platform only. It is, however, a benefit as it would allow the simple migration of the application should the stakeholder change device.

HTML5 hosts many features which are aimed at making the web a better place for desktop-style web applications (Mikkonen, Taivalsaari 2011). These features are given as the following:

1. Offline Applications: HTML5 will allow web applications to run offline.
2. Local Storage: HTML5 will offer a local storage solution that behaves like a simple key-value database.
3. Canvas API: The Canvas API is a 2d drawing canvas for producing graphical content in browser.
4. Built-in audio and video support: New audio and video tags allow media files to be played without external plug-ins.
5. Asynchronous script loading: HTML5 supports synchronous loading of scripts using async attribute of script tags.
6. Drag-and-drop support: Allows for drag and drop interaction.
7. Content menus: Programmatic control of browser menus.
8. Cross-document messaging: Safe communication of web pages.
9. Editable web pages: Allows for client-side, in-browser rich text page edits.
10. HTML5 no longer concerns itself solely on web content standardisation, but also focuses on standardisation of interactive functionality

The use of HTML5 in creating rich web applications can be augmented with the use of JavaScript. JavaScript is defined as the most ubiquitous programming language in history (Flanagan 2011). This is due to “overwhelming majority” of websites utilising JavaScript across a wide range of platforms including PC’s, smartphones and tablets. One significant benefit in the use of JavaScript is the variety of libraries and plugins available which can enhance elements of the web application. Examples of this are the use of the JQuery, jQgrid and Rafael.js libraries when conducting the initial software experiments (section 3.4).

Furthering the use of JavaScript is the use of Ajax (Asynchronous JavaScript and XML). Ajax is a client-side scripting technology which makes web applications more responsive (Deitel, Deitel 2008), and turns browsers into application platforms that closely mirror desktop applications (Holdener 2008). The typical web application model works by sending interface triggered HTTP requests to a web server, the server processes the data, and returns an HTML page to the client (Garrett 2005). Garrett states that one of the key benefits in the use of Ajax is the elimination of “the start-stop-start” nature of interaction between client and server. Since the application will rely on the transmission of data between client and server, Ajax holds many benefits. As the project involves the deployment of a WCF service application, Ajax may be used as it is one of the key technologies used in the parsing of HTTP requests and serialisation of XML or JSON data when using WCF services.

The second method of application development for BlackBerry PlayBook is the BlackBerry Tablet OS SDK for Adobe AIR. Adobe AIR is a cross-platform runtime that allows the development of applications using technologies such as HTML5, Ajax, Adobe Flash and Adobe Flex to develop rich internet applications (Young, Givems, Gianninas 2009). This is similar to the WebWorks SDK in that you can use familiar web technologies for development, but where it differs is in its cross platform support and support of Adobe technologies. Particular attention will be paid to Adobe Flex, as it is a mobile development framework and the most appropriate based on the project.

Adobe Flex is an open source framework which allows for the development of mobile applications for popular mobile operating systems as well as browser and desktop applications. Flex applications typically consists of code written in two languages, ActionScript and MXML (Noble et al. 2010). ActionScript is a scripting language, but is object oriented in nature (David 2011). MXML is a mark-up language, similar to HTML. Although ActionScript is similar to JavaScript in the sense that they are both implementations of ECMAScript (ECMA-262 specification), ActionScript is different from JavaScript in a few ways. ActionScript is an object oriented language, and focuses around the programming of data as objects, and JavaScript is a weakly typed scripting language, although it does in some cases support object-oriented development.

Since Flex cannot be used out with Adobe environments, this gives HTML5 an advantage as it can be used in any web environment. Furthermore, HTML5 can be developed and tested using the most basic of editors, whereas Flex development requires Adobe development environments or plugins. In 2011, Adobe reported that they will no longer support flash development for mobile browsers and will instead focus on development around HTML5 technologies (Arthur 2011). Although this does not directly affect the development of an application using the Adobe AIR runtime, it further reinforces the prevalence of HTML5 as the future of mobile web development.

The third method of development available for BlackBerry PlayBook tablet is the BlackBerry Native SDK for Tablet OS. This is a powerful application development framework which allows for the creation of high performance applications using C and C++ APIs which expose most of the hardware driven features of the PlayBook (BlackBerry 2011). Since the project application does not require any hardware accelerated features, there is little benefit over using one of the alternative development methods.

REST vs. SOAP

One of the main issues when researching the WCF application was the argument of Representational State Transfer (REST) vs. Simple Object Access Protocol (SOAP). These are two of the main web communication standards supported in WCF and both are used widely within the WCF community. However, although both offer similar functionality, both appeared to be very different.

REST is defined as an architectural style for designing networked applications (Elkstein 2008). SOAP is a protocol specification for exchanging data between two endpoints (Flanders 2009). Although both focus around the simple exchange of data, both do it in a very different manner. The main focus of RESTful services, as given by Spies (2008) is the simple point-to-point communication over HTTP. SOAP also relies on HTTP communication but does so with differing communication principles. The following points as described by Chappell (2009), indicates the basic communication principles of both REST and SOAP:

REST

- Focussed on accessing named resources (URLs)
- Every application exposes its resources through the same interface.

SOAP

- Focussed on accessing named operations
- Different applications expose different interfaces.

As defined above, both communicate very differently with applications. REST typically involves accessing a URL resource whereas SOAP involves accessing named operations. In a RESTful service, resources are exposed via URLs. For Example, the following URL will retrieve the "part" with id equal to 00345 (Costello, unknown):

http://www.parts-depot.com/parts/getPart?id=00345

In contrast to this, SOAP typically involves sending a payload which adheres to a SOAP schema (anonymous, 2005). For this reason, REST is generally regarded as the "simpler" method. This is due to it "shedding" some of the heavyweight requirements needed in a SOAP application (Rubio, 2005). However, SOAP is structurally more complex than REST for a reason, resilience (Anonymous, 2005). SOAP payloads can typically include several characteristics that are not available in REST such as priority, expiration, security credentials, routing information and transactional behaviours. This does not mean that SOAP is a more "secure" web service however, as both REST and SOAP utilised Secure Socket Layers (SSL) (Flanders, 2009). What can be said though is that SOAP allows for more secure "end-to-end" security, which is not an option for RESTful services based on its "point-to-point" architecture style. In general, the application areas of each are given as follows (Rozlog, 2010)

REST

- **Areas of limited bandwidth and resources**
  The return structure of REST can be in any format. Any browser or web technology can be used because REST uses standard GET, PUT, POST and DELETE verbs which make Ajax a suited technology.
- **Requirement for totally stateless operations**
  If the requirement is for CRUD (Create, Read, Update, and Delete) operations, REST is the best fit.

- **Caching situations**
  The information can be cached because of the totally stateless operation of the REST approach.

SOAP

- **Asynchronous processing and invocation**
  If the application needs a guaranteed level of reliability and security.
- **Requirement for formal contracts**
  If rigid specification requirements are required.
- **Stateful operations**
  If the application needs contextual information and conservational state management.

As indicated above, REST is particularly suited for Ajax requests and for CRUD operations. Since the application will involve the development of a "front-end" application using popular web technologies, such as JavaScript, Ajax requests will ultimately go "hand-in-hand" with a REST architecture style. In addition to this, the project will require database persistence and as such the support for stateless operations (CRUD) will be significantly beneficial.

With regards to mobile application development, Cox et al. (2011) believe that REST is the most suitable mobile web communication method as it offers developer productivity, performance, bandwidth efficiency, security and robustness. In a study by Aijaz et al. (2009), it was discovered that in the performance of SOAP and REST mobile web servers, REST was the recommended solution. This was evaluated by studying the performance metrics of server utilization, waiting-time, throughput and queue-length. An additional study by Al Shawan et al. (2010) supported REST as the most beneficial communication method as opposed to SOAP. This study involved the evaluation of distributed SOAP and RESTful mobile web services, and ultimately discovered that REST has several performance benefits over SOAP. One of the interesting findings of this study was that RESTful services employ a loosely coupled relation between server and client due to the uniform interface (as mentioned above), which adds balance towards using it for distributed mobile web services. This is particularly beneficial towards this project as there may eventually be a desire to distribute the application to other web formats (section 3.3.2).

2.4 Summary of the Literature Review

The conducting of a literature review has uncovered many issues and potential solutions regarding the design and development of an application for tablet computer devices. The following major issues have been identified through the conducting of the literature review.

1. Learn about similar tablet applications and their benefits
2. Identify how users interact with touch screen devices
3. Identify the key usability design issues
4. Identify the most appropriate lifecycle for a project of this nature
5. Assess software technologies and identify the most appropriate based on its features and the requirements of the project

3 Problem and Systems Analysis

In order to gain a better understanding of the development processes and problem area in general, a problem and systems analysis took place in order to identify any key issues relevant to the project or the steps involved. Whilst this was essentially an on-going process throughout the project, an initial problem and systems analysis took place prior to the main undertaking of the project. This was complimented by evidence gathered from the literature review. The following tasks were completed prior to the main project development in order to advance the knowledge of the problem domain:

1. **Evaluation of company systems**
   This task involved assessing the company systems in order to determine the best technologies for the application development in order to ensure compatibility. In addition to this it served as a way to investigate potential solutions which may be suitable for the project.
2. **Practical experimentation**
   This task involved the completion of a series of practical experiments based around what was known of the project at the time. These experiments were used to test technical solutions.
3. **Lifecycle and requirements capture**
   This task involved selecting an adequate development lifecycle for the project and also identified the requirements of the project.

3.1 Evaluation of Company Systems

As this project was being developed in conjunction with a large Company (Company X) and would eventually be deployed within the Company, an initial analysis of the company systems took place. This was completed in order to gain a better understanding of the possible solutions and tools which were available for use and also to ensure that the technologies used within the project were fully compatible with the existing technology of Company X. The systems analysis took place in a single interview prior to the requirements interviews given in the following subsection. This meeting involved an hour long Q and A with a software developer of Company X, which identified many of the potential solutions and technologies available for the application development. One of the major topics discussed was the deployment of the application. Based on this discussion, the following deployment topology diagram was created.

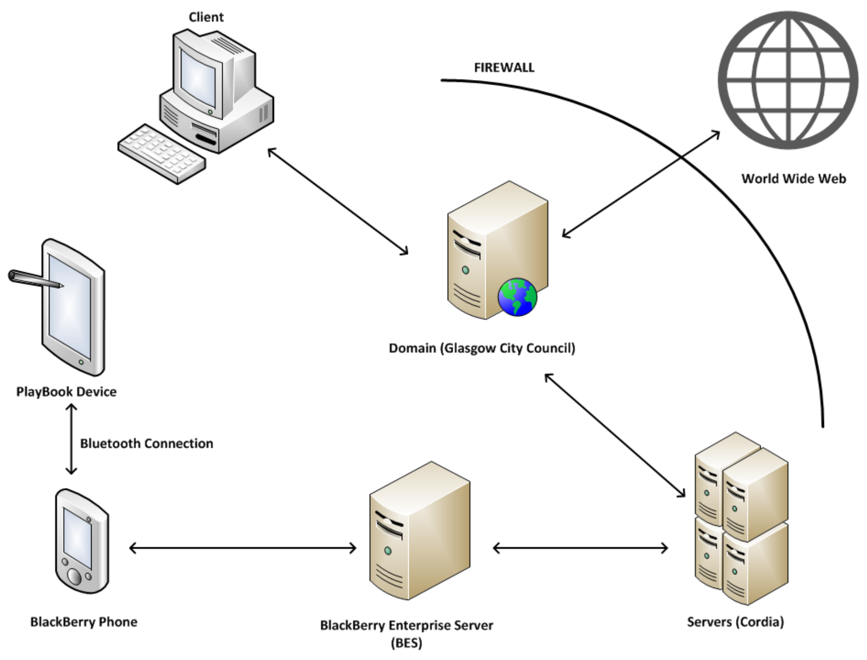


Figure 1 – Company X Deployment Topology

The above topology indicates the deployment scenario of the PlayBook application. As indicated above, the PlayBook application must connect through a BlackBerry smartphone. This was identified as a result of practical experiment 1 (section 3.2), which was conducted prior to this meeting. The "back-end" service (WCF and corresponding database) would be hosted on the servers of Company X.

In addition to this, the meeting highlighted some of the potential tools and solutions which were available for development. It was discovered that company favoured both .NET and Java frameworks and did not have a preference as to which was used. The .NET framework was selected due to the authors familiarity with it having completed some C# win forms applications and an MVC application. In addition to this, the company indicated their preferred database technology as SQL Server 2008. By selecting SQL Server 2008 as the database technology of choice, it would eradicate any persistence problem which may have been met with the introduction of a new database technology.

3.2 Design and Implementation

Prior to the final design and implementation period of the project, a series of practical experiments were conducted which helped prove many key concepts and demonstrate the possibilities of the PlayBook device and its development technologies. These experiments were conducted in the period leading up to the final project undertaking, in the period of September 2011 to December 2011. The experiments served mainly as a way to test the development technologies as well as their integration with the supporting technologies within Company X.

The practical experiments did not follow any schedule, and were largely conducted to further the author's knowledge, as well as prove many of the key points of interest for the project and also for Company X. Prior to conducting these experiments, very little was known about the project other than it would feature a "booking" application and a "sign in" application. By conducting these experiments it was expected that some key concepts and issues may be identified at the same time gaining some reusable code components.

The following is a list of the practical experiments conducted, as well as a short explanation of why it was selected and how it influenced the project. Since the practical experiments were not conducted on a schedule as such, they will be presented in the order in which they were completed. The code listings for the practical experiments can be found in appendix A.

1. **Simple test application consisting of HTML5, CSS and JavaScript deployed to PlayBook.**

   The first experiment was conducted to demonstrate the use of web technologies using the BlackBerry WebWorks SDK. It served as a quick assessment of how easy it was to create a sample application and deploy it to the PlayBook device. The web app was essentially a "hello world" app, and due to its simplicity did not really test any of the capabilities of using web technologies. Instead, this experiment served to test the process of converting web pages or a website into a PlayBook application, and deploying the application to simulator and eventually the physical device.

   It was found that the process of converting an application to a PlayBook format using the WebWorks SDK was a relatively simple process, as was deploying the converted application to the PlayBook simulator. The only major issue that was found when packaging the application was that certain errors could occur within the required "config.xml" file. It took many attempts to package the file due to errors occurring. The error messages presented were quite verbose, although they were also quite unclear to someone unfamiliar with the technologies. Seeking support was difficult also, as the PlayBook and its technologies are quite new and have a relatively limited user base. From this experiment a default "config.xml" file was created, that was an error free "boilerplate" file which served throughout the project until the final builds. The config.xml file is a document that contains the WebWorks application namespace, the name of the application, application permissions, the start page, and the icons and defines the author elements (RIM 2012).

   Another problem was highlighted here involving the deployment of the application to the physical device. The deployment to the simulator is quite an easy process, and was met with minimum errors or failure. Deploying the application to the PlayBook device was a problematic process which involved the process of "signing applications". The signing of applications is a lengthy process which involves applying for application "keys", so that an application can be deployed on the physical PlayBook device. It was decided that for the majority of the project, the PlayBook simulator would be sufficient, as it offers the exact same debugging functionality as the PlayBook device but reduces the process of loading the application on the device. It would have been preferable to deploy the application on the physical device, as it would be a more accurately represents the key design and interaction concepts.

2. **Use JavaScript to create HTTP requests.**

   This experiment demonstrated the use of HTTP requests in order to retrieve information from a web service. This experiment consisted of two parts; Create HTTP requests to retrieve a hard coded array and modify the HTTP request to fetch data from a web service (within Company X). This functionality was then tested on the PlayBook and confirmed the possibility of data transfer.

   As mentioned in section 3.3.2, Project X, which is a currently on-going project within Company X involves the development of “legs”, each of which has a BlackBerry front end and a Web front end. This is largely due to the need to share much of the information across the various applications within the company. Before the requirements meetings were conducted, Company X expressed a desire to have client-server communications. This would be integrated with the central repositories of Project X in order to provide data to the devices. On the advice of a company X software developer, XMLHttp requests were researched and a small experiment was conducted. This experiment took a few weeks to prove, largely due to unfamiliarity with the process of creating XMLHttp requests. The experiment was essentially a very simple header request, where the header content was specified as both “XML” and “JSON” and the output was displayed on page in its “raw” format. The experiment served as the basis for experiment number 3. Since one of the main concepts of WCF service applications is HTTP requests for information, this experiment essentially proved that requests to the current data sets in Company X was possible.

3. **Retrieve JSON data from a web service and display the data in an HTML table.**

   Expanding on the previous experiment, this experiment looked at furthering the HTTP requests and displaying them is a format understandable to users. One of the main issues when getting data from web services is in the presentation of data. XML and JSON in their “raw” format are unreadable to untrained persons, as such it is important to correctly display or “serialise” the data so that it is understandable for users.

   This experiment was conducted both using a hard coded array of JSON objects, and by performing HTTP requests in order to use the “live” data. The live data was test data that was available on the database in Company X. The JSON objects were in the following format:

```
{"addressLine1":"075 Ingram Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama, Oprah","id":"52"}
```

The above format is a direct copy of the HTTP request content from the database, and was hard coded as it is impossible; due to restrictions on the Company X web service, to perform HTTP requests from an unauthorised client. In the JavaScript, there are two functions; function MockData(), and function GetHTTPData(). Depending on the test location, the mock function could simply be “swapped” out with the “real” function by changing the “onclick” attribute of the following element:

```
<input type="button" value="Click me!" onclick="MockHTTPData()" />
```

This experiment influences a couple of aspects of this project. Most namely, the early “search for rooms” feature and in application layering. The inclusion of the search bar, which allowed for data to be searched and displayed, was later found to be a functional requirement and as such this experiment provided reusable code. Regarding application layers, this experiment catalysed by the use of “mocks” in past modules such as the Software Processes and Practices module, reinforced the benefit of having interchangeable data resources (section 4.3.2). Not only did the use of a “mock” allow for the continuous development of the application from locations outwith Company X, but it also allowed code to be perfected and tested before being used with real data.

4. **Display retrieved JSON data in jqGrid.**

This experiment was an expansion on the previous experiment and looked at the use of jqGrid for displaying retrieved information. The previous experiment involved the process of displaying understandable data to the user. During the initial research process, various plugins were researched which may have simplified processes whilst providing more functionality. Naturally, as the project used JavaScript, JQuery was researched (and later implemented). JQuery is a JavaScript library which was found to greatly reduce JavaScript workload and provide many plugins which offered unique and interesting functionality. One of the plugins which was researched was jqGrid. This is a plugin which allows for grids to be created from various content feeds and provided advanced functionality, compared with the simple tables from the previous experiments.

The experiment involved the creation of a jqGrid using a hard coded array of JSON objects. This was later expanded (in the main implementation) to create a grid from a JSON feed using the JQuery function .getJSON, which is an Ajax “GET” function which allows JSON data to be collected from the WCF service. The experiment was primarily used to test the plugins suitability; however, viewing of a similar application created by Company X on the PlayBook highlighted a potential usability problem when developing the user interaction. When demoing the grid at a meeting, an example of a similar grid, using jqGrid, was given by Company X using the BlackBerry Playbook. When trying to access sub-layers of data, normally you would “double-click”, or in the case of the touch device “double-tap”. However, this gesture was not possible on the PlayBook and only a single tap was a valid gesture. This was noted and was to be researched prior to implementing the grid in the main application, but changes in the requirements resulting in the problem being avoided.

5. **Create a sample user interface of room map, with interaction.**

This experiment demonstrated the use of a basic user interface for the “map view” of the application, and how users may interact with it. Initially, it was assumed that the application would feature a “grid view” (as indicated by previous experiments), which would display a list of the available rooms which users could then click on and make a booking. When researching some usability features and interesting applications of touch based devices and applications, it was suggested that perhaps users could interact with the rooms in a novel way which would utilise the touch benefits of the PlayBook. This idea was presented to Company X, who found it interesting and expressed a desire to pursue it. It was decided that the “map view”, would be a bird’s eye view of the office floor plan, which would have “highlighted” rooms, which were the meetings rooms which were available for booking.

The initial research for this experiment concluded that the HTML5 canvas element would be the logical choice; however, there seemed to be a lack of support for unique interaction using canvas. Further research yielded a JavaScript library called “Raphaël.js”, which is a JavaScript library to simplify working with vectors on the web. The experiment involved the creation of the “map view” using Raphaël.js, and a “mock” booking interaction was created to simulate a basic room booking. As the project was in the very early stages, not much though was given to persistence of bookings, or even to providing booking information. The main concern was in the actual PlayBook application and how it would be presented to the user. Basically, this experiment had an example map interface, which was created by drawing a series of rectangles which represented rooms, each having a timeout animation. The timeout animation turned the room from green, which indicated the room was available, to red, which indicated that the room was now booked.

```
var room2 = paper.rect(250,0,250,100);
            room2.attr({fill: '#64FE2E', "stroke-width": 3});
            room2.node.onclick = function(){
                        room2.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                        });
            }
```

Figure 2 – Raphaël.js Code Snippet

Although this would serve as a very crude example of the room booking procedure (by comparison to the final build), it demonstrated some sample touch applications of the PlayBook device.

6. **Create a basic WCF service application and simple web page in order to collect and display JSON data.**

   The final practical experiment which was conducted before the main implementation was the creation of a WCF service application and a simple web page which would collect and display JSON data. This experiment was not finished prior to the main implementation and instead sort of "bled" over into the first iteration of the application. However, as WCF was a new concept, it provided the basis for learning the main concepts of WCF prior to creating the first major WCF application.

   One of the main concepts which had been identified in light of this experiment is the concept of Simple Object Access Protocol (SOAP) vs. Representational State Transfer (REST). WCF SOAP and WCF REST both provided many benefits to a project of this nature, and as such an expansion of the technical review (section 2.3) was undertaken in order to determine the advantages and disadvantages of each, ultimately concluding with the selection of REST as the most suitable for this project.

3.3 Lifecycle and Requirements

3.3.1 Lifecycle

The software development process selected for this project was Scrum. Scrum is an agile framework for completing complex projects and can be used to cater to small development teams (Janoff, Rising 2000). This development process was selected based on the results of the literature review (section 2) which was conducted in order to determine the most suitable development processes based on the known requirements of the project at the time. One of the main benefits to using a Scrum framework, as previously mentioned is that it caters to small project and development teams. This was particularly suitable to this project as it is an individual undertaking.

The Scrum development framework involves several specific roles which make up "Scrum teams". Scrum teams typically consist of an agile development team, supported by two specific roles, the Scrum master and the product owner (Cohn 2012). The Scrum master is essentially a "Coach" who supports and facilitates productivity. The product owner represents the business and its interests, also representing the users and stakeholder's interest in the project. As this project was being conducted in collaboration with Company x, the Scrum roles were allocated as follows:

1. The Agile Development Team: Michael Meina (Myself).
2. The Scrum Master: xxxxx xxxx (Junior Developer).

3. The Product Owner: yyyyyy yyyyy (Development Manager).

The Scrum development process revolves around the development of multiple iterations of a product, which is executed during “Sprint cycles”. Sprint cycles are typically completed every 30 days, after which a working increment of the software is released. However, due to the short timescale of the project, the sprint cycles were 21 day cycles which culminated with a Sprint meeting with the Scrum Master and Product Owner.

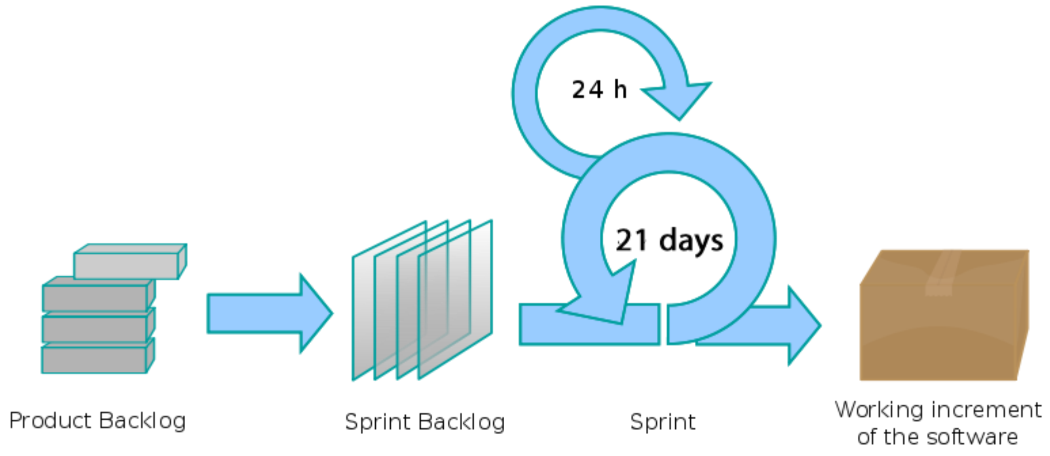


Figure 3 – Sprint Cycle Diagram

Sprint Iterations

The original sprints as given in the interim report were loosely followed for the project lifecycle, with the exception of the final sprint which was altered to allow time for the final project proceedings. These dates were:

- 3rd February 2012
- 24th February 2012
- 16th March 2012
- 6th April 2012

The actual dates of the completion of the sprints were:

- 3rd February 2012
- 24th February 2012
- 16th March 2012
- 30th March 2012

The final sprint was adjusted to a 2 week sprint in order to allow time to evaluate the project. During the sprints, several iterations of each application were created. These were not the deliverable iterations, but were instead smaller sections of significant work. At the end of each sprint cycle, a “snapshot” of both iterations (the deliverable) was presented to the company for evaluation. The following diagram illustrates the completion of the iterations in approximate relation to the sprints.

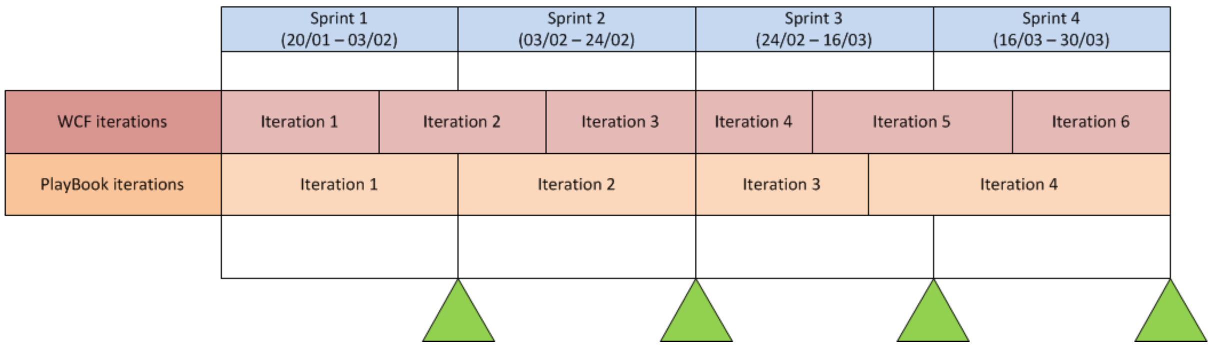


Figure 4 – Iteration Implementation Diagram

As indicated by the diagram above, several iterations of each application was developed. The green triangles indicate "snapshots" of the application which was presented as a deliverable to Company X. At first it was assumed that the iterations of the application would be completed in sync with the conclusion of each sprint cycle, as suggested in the original Gantt chart of the project proposal. However, due to changing requirements and other issues which affected development, the project iterations were completed as indicated above. This was not detrimental however, as each "snapshot" deliverable included more of the required functionality.

Each sprint cycle was concluded with a stand-up meeting which involved a brief demonstration and review of the application. Typically these meetings consisted of the development team, the Scrum master and the product owner. However, on a couple of occasions, particularly during the "middle" of the project, the Product owner was unavailable. On these occasions, the Scrum master would fulfil this role also. The stand-up meetings typically consisted of the following procedures:

1. **A discussion of the user stories implemented**
   A brief discussion of the user stories which were implemented with this deliverable.
2. **A short demonstration of the application**
   The application is demonstrated by the development team bringing any new features or issues to notice.
3. **A code review of all code created**
   The Scrum master talks through the code with the development team, highlighting any obvious flaws and suggesting improvements.
4. **Quality assurance and acceptance testing**
   Due to the lengthy process of quality assurance and acceptance testing, this was completed every two sprints as the project was relatively small in nature. The quality assurance and acceptance testing involved the testing of the application in order to ensure that the application met the expected requirements as defined by the user

stories. This process was typically completed by the Scrum master acting as product owner.

The main objective of the stand-up meetings, in addition to presenting the deliverables was to develop more user stories and move more tasks from the product backlog into the development stage. The stand-up meetings usually concluded with a discussion about the future paths of the applications and a development of the appropriate user stories.

3.3.2 Requirements

One of the main benefits to using a Scrum framework, as indicated by the literature review (section 2.3) is the use of "user stories". User stories are a form of requirements capture which can be described as "units of customer-visible functionality" (Power 2011). The use of user stories was a major benefit complimented by the Scrum framework. Cohn (2004) describes the method of using user stories as "simply write down what you want to do, and you do it". This was found to be a simple yet powerful method of requirements capture, by comparison to past methods, as it identified the key aspects of functionality required by the user and outlined the specific criteria required in order to get that feature working. The following section details the requirements capture procedure undertaken throughout the project.

Initial Requirements Capture

Prior to the user stories being created, two meetings were held with Company x, where the core functionality and requirements were captured. These meetings were conducted by the Product Owner, and involved the Scrum master and the development team. The meetings were generally informal but were very focussed on defining the requirements of the project.

Meeting one

The first meeting conducted involved a general discussion about the application and where the project was situated in relation to a larger project which was currently being developed within Company X. This larger project has many "legs", as described by the Scrum master, one of which would be the applications developed as part of this project. The "legs" of the project are essentially separate, smaller projects which share the same "body" (central system). This project will be referred to as "Project X".

Project X is a large project currently being developed in Company X which involves a revision of many of the systems and tools used within the company. Company X's main interest in this project is to assess and evaluate the feasibility of tablet computers as a form of information display. The BlackBerry PlayBook was selected as the company is a member of the BlackBerry alliance, and all existing devices on the current system are BlackBerry based. This would essentially eradicate problems which may have arisen when introducing a new device or technology as the PlayBook device can utilise the current systems.

The main points raised at this meeting are given as follows.

1. **The development of a PlayBook application will assess the suitability of displaying visual information in a new or novel way.**
   As mentioned in the project introduction, the development of this application will be used to assess the suitability of tablet computers and interaction as a new way of displaying and interacting with information. Company X are currently undergoing a large revision of software and hardware systems and are looking to investigate how effective this may be with regards to certain solutions they have within the company.
2. **In Project X, all "legs" must have a BlackBerry front-end, and a web-front end.**
   Since the company is a member of the BlackBerry Alliance, and all existing devices and systems are BlackBerry integrated, one of the requirements of Project X is that all "legs" must have a BlackBerry front end and a web front end. BlackBerry have an SDK available called WebWorks SDK, which allows for the creation of mobile applications using popular web technologies such as HTML5, XHTML, CSS, JavaScript, PHP etc. This further reinforces the decision to use WebWorks SDK, as all technology which will be developed for the BlackBerry device can be simply "ported" to a browser based application. This is identified in the literature review, as Pimmel (2011) states that the idea of mobile web is "design and develop once, deploy anywhere".
3. **A room booking application will be developed to test the suitability of both the device and the interaction method.**
   The development of this application will aim to assess and evaluate the device itself as a platform for development. The company are interested in a "trial run" of the PlayBook before they invest in it as a new platform for Project X.
4. **The PlayBook must be within Bluetooth range of a BlackBerry smartphone on the BES.**
   Prior to the projects scheduled start, many small test applications were developed to test the device and certain development concepts which would be relevant to the project. One of the tests which were conducted was the development of a simple XMLHttp request application, where a user could input a name into a search bar, and request information from Company X database via JavaScript XMLHttp request. When this application was deployed to the PlayBook there were errors connecting the device to the BES. This problem was solved by routing the device (via Bluetooth) though a BlackBerry smartphone which was connected to the BES.

Meeting two

This meeting was held shortly after meeting one and involved the discussion of specific features and functionality required of the application for Company X. The meeting was very productive and yielded many requirements which formed the basis for development. Present at the meeting was the Product owner, the Scrum master and the development team (the author).The product owner represented the interests of both the business and its users. Firstly, the product owner highlighted the general functionality and interests in the project and later exposed themselves to questioning by the development team in order to further the requirements and develop new ones. The questioning was particularly effective as it allowed for the user and business interests to be elaborated on. This was beneficial as project requirements, based on past experiences and research, are prone to changing, as such, this questioning allowed for the requirements to be developed to a point of near absolute clarity. It must be noted, however, that these were only initial requirements which were developed further after these initial meetings.

The following is a list of the features and functionality decided in the meeting. Each will include a short explanation of the feature or functionality as it was defined at the time.

1. **Maintain a list of rooms**
   As the project will contain a variety of rooms available for booking, each with specific equipment or facilities, it is important to be able to maintain these rooms in order to inform the user of changes to the room. This feature would be a developer feature as the room list will be maintained by either the in-house software development team or the technical support team.
2. **Search for available rooms**
   Users must be able to search for available rooms by either inputting specific search criteria, or by viewing the room's availability. This could be implemented by having a tabular view of the available rooms, or by having a map or calendar view (see point 9).
3. **Book a room**
   Users must be able to search for rooms with specific features, and book the room. Room bookings will only be available between 8am and 6pm, and can only be booked in 15 minute increments. Room bookings must have a booking name, which would typically be an employee name, a start time, and an end time.
4. **Room booking confirmation**
   When a user books a room, a clear confirmation or denial of the booking must occur to indicate that the room booking has been confirmed or has been denied.
5. **15 minute refresh function**
   Since the bookings have a 15 minute increment, a 15 minute refresh function will allow for the room status to be updated according to its availability.
6. **Define booking length**
   Users must be allowed to define how long they wish to book a room for. This will be done by allowing them to input a start time, and an end time for the booking.
7. **Cancel bookings**
   Users must be able to cancel bookings. One way this may be achieved is by assigning a unique id, which may allow users to view their own bookings and cancel them.
8. **Modify bookings**
   Users must be allowed to modify their bookings if possible. An example of this not being allowed is if a user has a booking for an hour that they wish to extend, but there is a booking which exists immediately after the original booking. In this case the modifications of bookings would not be allowed.

9. **Calendar view**
   As mentioned in point 2, the users must be able to search and view specific rooms. One way this may be implemented is by using a calendar to display the bookings that exist, so as the user knows when the room is available for booking.
10. **Outlook API**
    One of the interests in the application is in “tying” it in with Microsoft Outlook. Currently, the company uses Microsoft Outlook to book rooms. This is done via the receptionist. The application may be integrated with the current Outlook system. This will be particularly beneficial should the unique booking id or reference be implemented, as the user could be emailed the id or reference to prevent them from the impracticalities of having a code displayed on screen.
11. **Recurrences**
    The current room booking method within the company allows for recurring bookings to be made. Users may be allowed to make recurring bookings which would remove the need for them to make repetitive bookings.

These requirements, although not complete, provided the basic functional requirements of the project.

Final Requirements

After gathering the initial requirements detailed in the previous section, user stories were created to define the requirements in greater detail.

The user stories were created by the Scrum master and the development team and were created as they were required. Since the project utilised a Scrum framework, several iterations of the application(s) were planned, and as such, not all user stories were required to be fully developed. User stories were usually created during the stand-up meetings (section 3.3). In the stand-up meetings, the Scrum master and the development team would develop the user stories from the existing requirements and add or remove certain features where appropriate. Typically, the user’s stories take the “formal” user story format, given as:

“As a (role), I want (something), so that (benefit)” – Cohn, 2008.

All of the user stories were created in the above form, on user story cards. These cards are simple cards which contain the user story in the “formal” format, and in our case also contained time estimations and acceptance criteria. The user stories typically contained time estimations for each task required which was provided by the author. At each sprint meeting, these time estimations were reflected on and used to identify specific weaknesses in the author’s abilities and also to promote the ability to accurately estimate times. This is continued in section 5, evaluation. An example of a user story card is given as follows (search rooms example):

Search
As a user I want to be able to search availability of meeting rooms So that I can see which rooms are available to make a booking

Time estimate: 15 hours.

Typically these cards are physical cards, which are usually pinned to a large board (named a task board) in an office or development space to visually manage the requirements of a project. However, this project utilised a free online tool called Trello (Trello.com), which is an online collaboration tool for managing work and processes. In Trello, the management of the user stories was the exact same as using physical cards and task boards. One of the major benefits to using Trello, as opposed to physical cards and task boards, was that it is a web based tool and allowed for multiple users to collaborate on the same project. The Trello task board created for this project had the development team, Scrum master, Product owner and the Software development manager of Company X as active members who could log on in from most web based devices and view the current status of the project. This is more beneficial compared with its physical counterpart as it allows for all interested parties to access the project from almost anywhere, whereas the physical version would require everyone to be present in a single location. Screenshots of the Trello task board can be viewed in appendix B.

The user stories in this project only use “as a user” and “as a developer” as the only actors in the system are users and developers. When writing the user story, the description is broken into three separate parts. This is to clearly split the user story into three main parts, “role”, “something” (a function usually) and “benefit” (the result of the function). Not all of the initial requirements were developed into user stories. This was largely based on time constraints and/or relevance. The following is an example of the user stories which were developed during the entire project lifecycle. The full user stories, both original and appended can be viewed in appendix C. Throughout the project, changes in the requirements were prone to occurring, as such the user stories will be reflected on later in the report and will be compared with the originals (see conclusion section 6.2.1).

Maintain Rooms
As a developer I want a list of rooms and their availability to be accessible So that I can retrieve and update this information.
Acceptance Criteria 1: Does the system allow me to retrieve the status of all rooms? Acceptance Criteria 2: Does the system allow me to update the status of a room?
Tasks - Create WCF project and learn the basics - Build rooms model - Build test web app - Tie web app and WCF service together - Create retrieve status method - Create the update room method

Figure 5 – Example User Story (Maintain Rooms)

4 Design and Implementation

4.1 Design

User Interface Design

As the project involved the development of a "front-end" application which would be used by the staff of company X, it was important that the application had a clear and consistent design in order to prevent usability issues. This was particularly important as it is expected that the application would be deployed without any training being issued. In order to prevent any design issues from occurring and also identifies any known issues regarding tablet and touch screen devices, a literature objective was undertaken in order to advance knowledge of the problem domain. Literature review section 2.2 highlighted many issues which had to be considered when designing the application interface such as gestural input and accuracy of input. However, one of the main interests regarding the application interface design as mentioned in the initial literature review was a focus on simplicity. As mentioned previously, one of the main issues when deploying the application is deploying it in a fully usable and understandable state, by removing the learning curve often associated with the introduction of a new software solution. This is supported by Berkun (1999) who states that simplicity does not mean lack of functionality but means a fast initial learning curve. Another benefit to simplicity is that it encourages innovative methods of accessing information (Chang et al. 2007). This proved particularly true regarding the "datepicker" and "timepicker" described in section 4.2.1, which simplified the process of inputting dates and times and presented it in an innovative method.

The main design of the application was based around the requirements identified in the user stories. The user interface elements were largely designed around satisfying the acceptance criteria of the user stories, whilst aiming to incorporate the design issues identified in the literature review as well as any other design issues which were identified as the development continued. In certain circumstances, some informal user interface prototypes were created. An example of this is demonstrated in practical experiment 5 (section 3.2). This practical experiment was conducted in order to assess the feasibility of a "map view" selection method.

Initial Interface Design

As the PlayBook application ultimately yielded two separate applications in terms of the envisaged functionality, two different user interfaces were designed. The first user interface revolved around a "grid" view and also around the "search" user story. Both of which were later abandoned. The following screenshots indicate the basic user interface which was designed based on the functionality initially required.

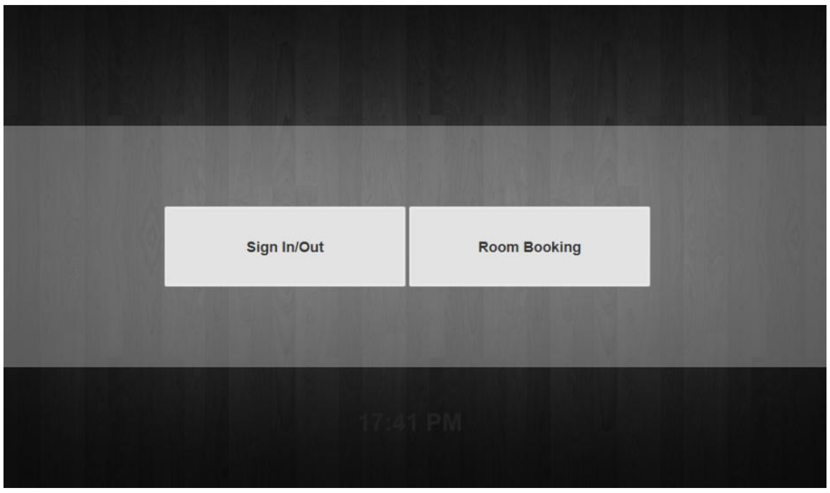
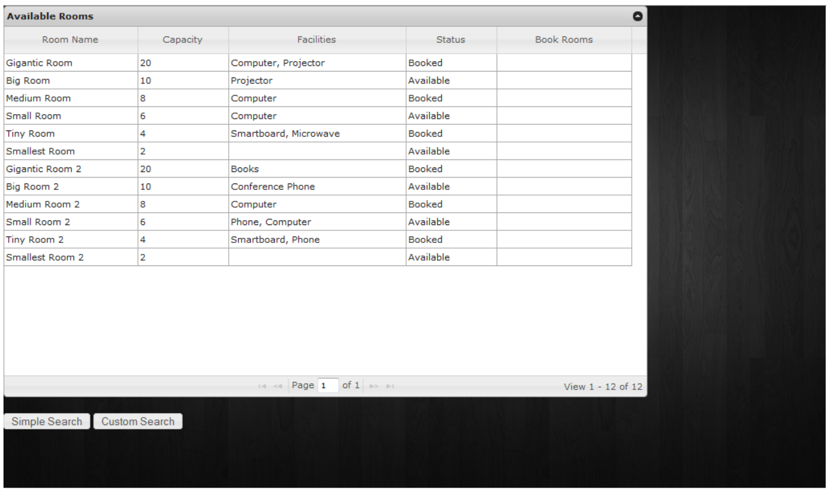


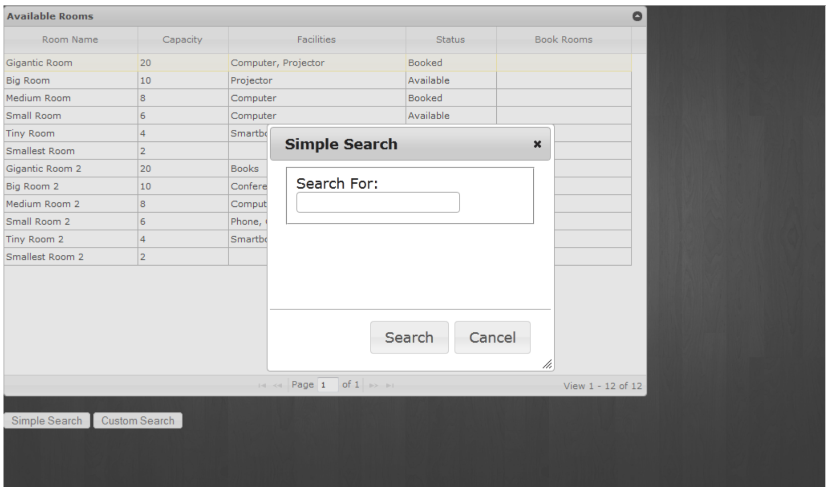
Figure 6 – Original Home Screen Design



Available Rooms

Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room	10	Projector	Available	
Medium Room	8	Computer	Booked	
Small Room	6	Computer	Available	
Tiny Room	4	Smartboard, Microwave	Booked	
Smallest Room	2		Available	
Gigantic Room 2	20	Books	Booked	
Big Room 2	10	Conference Phone	Available	
Medium Room 2	8	Computer	Booked	
Small Room 2	6	Phone, Computer	Available	
Tiny Room 2	4	Smartboard, Phone	Booked	
Smallest Room 2	2		Available	

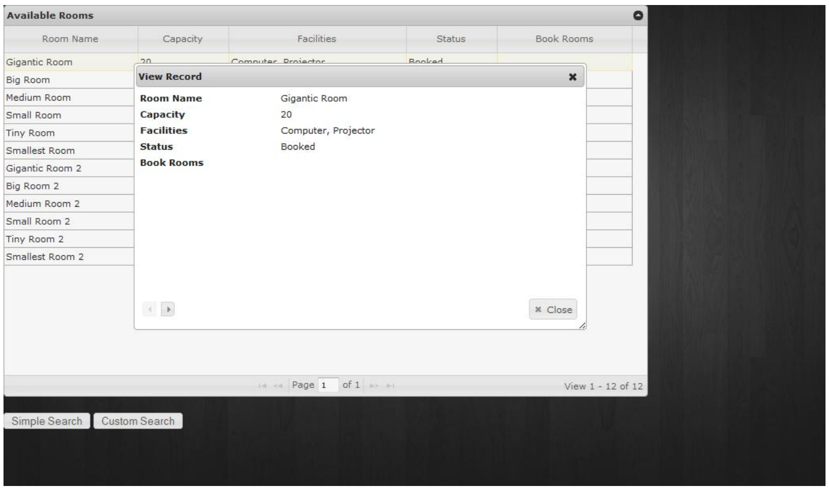
Figure 7 – “List View” Page



Available Rooms

Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room	10	Projector	Available	
Medium Room	8	Computer	Booked	
Small Room	6	Computer	Available	
Tiny Room	4	Smartbo		
Smallest Room	2			
Gigantic Room 2	20	Books		
Big Room 2	10	Confere		
Medium Room 2	8	Comput		
Small Room 2	6	Phone, (		
Tiny Room 2	4	Smartbo		
Smallest Room 2	2			

Figure 8 – Simple Search Box


Available Rooms

Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room				
Medium Room				
Small Room				
Tiny Room				
Smallest Room				
Gigantic Room 2				
Big Room 2				
Medium Room 2				
Small Room 2				
Tiny Room 2				
Smallest Room 2				

View Record

Room Name	Gigantic Room
Capacity	20
Facilities	Computer, Projector
Status	Booked
Book Rooms	

Page 1 of 1 View 1 - 12 of 12

Figure 9 – Room Information

The above screenshots indicate the basic user interface which was designed around the requirements of PlayBook application iteration 1 and 2. Although this was in very rough form, it satisfied the basic design issues such as displaying information clearly. Certain key issues identified in the literature review were addressed also. Most notably this was accuracy of input. As the PlayBook has a relatively small screen, it was important to incorporate large and usable buttons. This can be seen in figure X and figure x, where the buttons were enlarged in order to offer a higher degree of usability. In addition to this, the fonts were enlarged make the functionality clearer. The use of jqGrid was initially tested for both usability and design in practical experiment 4, which was completed prior to the main project development. The design issues highlighted here were carried through to the “final” screen design.

Final Interface Design

As indicated by the implementation iterations (section 4.2), the application interface had a major revision following the change of requirements. The main changes here which impacted the design of the application were the focus on the “map view” and “calendar view”. This ultimately resulted in the abandonment of the “Search” user story which included both requirements to search for rooms and view a list of rooms. The “search” user story was abandoned in favour of more intuitive methods of viewing the information. The “search for available rooms” function was removed as the database was simplified to simply contain a room and a roomid. The “view list of available rooms” view was removed as the rooms no longer had room criteria upon which to search for and also as rooms and bookings were able to be viewed without inputting criteria using the “map view” and “calendar view” pages. The following screenshots represent the final application interface designs of the “major” interface elements. These screenshots can be viewed in more detail in appendix D.

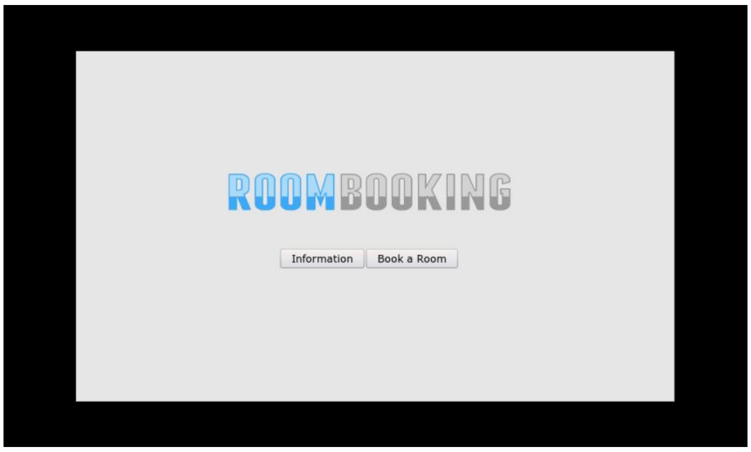


Figure 10 – Final Home Screen Design

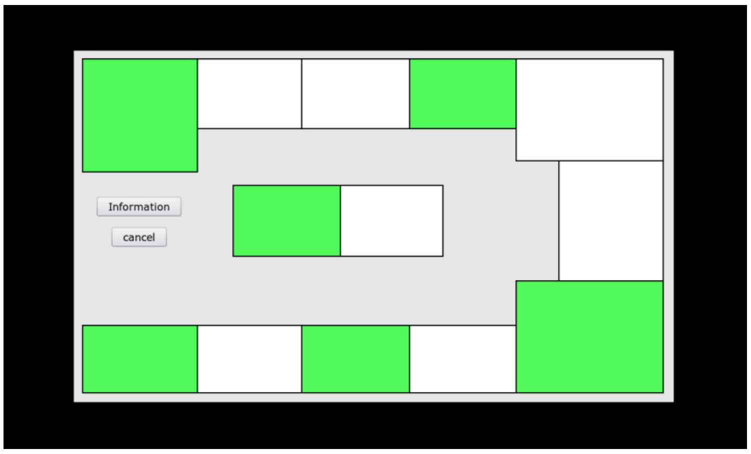
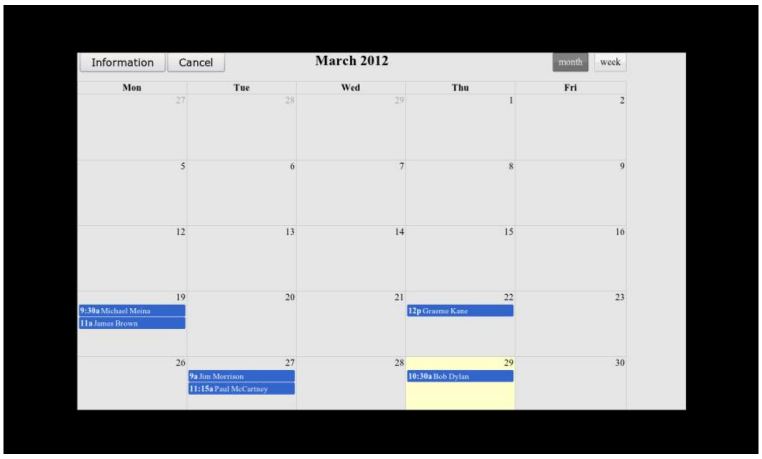


Figure 11 – "Map View" Page



March 2012

Mon	Tue	Wed	Thu	Fri
27	28	29	1	2
5	6	7	8	9
12	13	14	15	16
19 **9:30a** Michael Meina **11a** James Brown	20	21	22 **12p** Graeme Kane	23
26	27 **9a** Jim Morrison **11:15a** Paul McCartney	28	29 **10:30a** Bob Dylan	30

Figure 12 – "Calendar View" Page

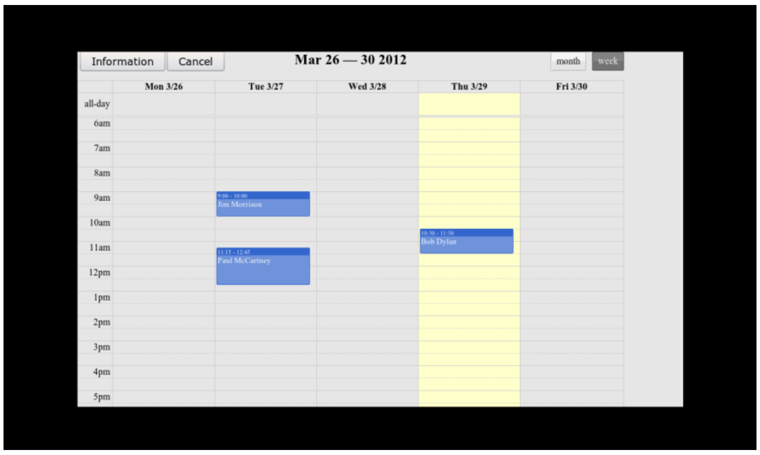


Figure 13 – “Calendar Day View” Page

The above screenshots indicate the “final” application interface design. In addition to this there are screenshots of every aspect of functionality which can be viewed in appendix D. Similar to the first interface design; this interface design was based around the requirements identified and also around the usability issues identified in the literature review. As mentioned, the main focus was on simplicity in order to provide an easier usability experience. This was achieved by incorporating only the single “tap” input gesture and also by keeping a very simple and clean colour scheme. The use of the single “tap” eliminates any confusion as to the input gestures required and makes the application very approachable. Other design elements which were incorporated were the large buttons and large fonts. As indicated in the literature review, button size can often have a detrimental effect to the usability of the application as the users may become frustrated. In order to prevent this issue from occurring, larger buttons were incorporated with a larger font in order to provide clarity for most users.

WCF design

The architecture style selected for the WCF service application of this project was Representational State Transfer (REST). REST or “RESTful” services are an architectural style for designing “point-to-point” client-server communication technologies (section 2.6). REST services typically involve exposing data resources via URL, which can be accessed via Ajax commands such as GET, POST, PUT and DELETE. In addition to this CRUD (Create, Read, Update and Delete) operations can be accessed. The following diagram indicates the basic WCF service implemented in this project:

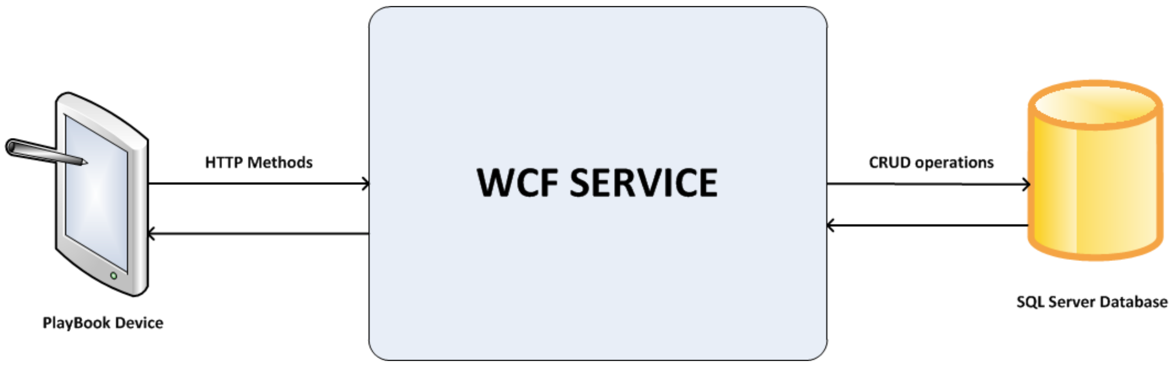


Figure 14 – Basic WCF Service Diagram

4.2 Implementation

This section will detail the development of the applications of the project. The implementation section contains three main sections; Practical Experiments, Implementation Iterations and Main Implementation.

The first section, Practical Experiments will detail the various code experiments which were created prior to the main implementation in order to gain a further understanding of the problem domain. The second section, Implementation Iterations will detail each of the iterations which were undertaken throughout the project. The final section, Main Implementation, will detail the “final” implementation of the project and will detail the project development in greater detail, incorporating lessons learned from the iterations.

Implementation Iterations

As mentioned previously, in section 3.3, the project utilised a Scrum framework and as such revolved around the creation of multiple iterations of working software. Several iterations of each application were created. This section will detail the development processes used to develop the applications as well as discuss any issues which occurred and the solutions implemented to resolve them. The multiple iterations of each application formed the basis for “snapshot” deliverables which were a deliverable of the project applications at the end of each sprint cycle (section 3.3.1). Each application will be presented in a dedicated section, which will detail each of the application iterations in the order in which they were completed. This will illustrate the evolution of each application in its own context for clarity. The code listings for the BlackBerry PlayBook application iterations can be found in Appendix E.

4.2.1 BlackBerry PlayBook Application

This section will detail the various iterations of the BlackBerry PlayBook application developed during the project lifecycle and will detail the significant discoveries and solutions recognised in each revision of the PlayBook application.

PlayBook Application Iteration 1

The first PlayBook application iteration involved the creation of a simple page which contained a “jqGrid”. This was completed shortly after the successful completion of the first WCF service application iteration. The page contained a jqGrid element which displayed the data as a result of the JSON data received from the WCF service application. At this stage, the grid only displayed the data from the WCF service application in a simple manner, and did not allow for the manipulation of this data. This iteration was created to satisfy the “Build search results screen” task of the “Search” user story.

This application iteration identified a problem when using jqGrid with the WCF service application data feed. The issue was to do with the display of the data in the JSON format returned from the service. As indicated in section 3.2, an initial experiment using jqGrid was conducted in order to test the suitability of the plugin. However, the data which was largely being experimented with was a “hard-coded” array of JSON elements. By transferring to the use of “live” data, the issue of cross domain scripting had arisen. This was due to the use of Ajax in retrieving the data from the WCF service application. Typically, browsers do not allow cross domain Ajax scripting as a security policy. This was a problem as the application relied heavily on sending and receiving data on a cross domain model.

The cross domain problem was alleviated from the jqGrid page by using a JSONP “callback” parameter in the Ajax url as follows:

```
url: "http://localhost:6188/RoomBookingService.svc/GetRooms?callback=?"
```

This solution essentially permits cross domain Ajax requests by telling the jQuery to replace the last “?” with a generated method parameter. This exposed the JSON data to the web application and allowed the data to be formatted into a jgGrid (Appendix E).

PlayBook Application Iteration 2

The second iteration of the PlayBook application expanded on the last iteration and added more functionality to the existing PlayBook application. The main features which were added to this iteration were the “search” features. In addition to this, the jqGrid was developed further to include the “onclick” feature when a room is clicked, which displays the information of the room in a separate modal form.

This iteration originally intended to have a “Simple Search” and a “Custom Search” feature. The “Simple Search” feature was to be a simple text input field, in which a user could enter any search criteria and search for matching data. The “Custom Search” feature was to be similar to the advanced searches available on popular sites, in that it would allow for multiple search criteria. After much experimentation, it was decided that the application “search” feature would consist of a search input field, and a drop down menu which would allow the user to specify what they would like to search for. For example, a user could enter “Computer” and set the drop down menu to “Facilities”. This allowed the user to view all facilities with a computer (see Appendix C).

In addition to this, there were refinements to the jqGrid implementation. These were relatively simple additions such as the “onclick” function, which when a cell is clicked, it will display the information of the selected room object. As well as this, certain usability features were added, such as the “sorting” feature of jqGrid, which allowed the data columns to be sorted by clicking the title cell.

PlayBook Application Iteration 3

Following the completion of the second application iteration, a sprint meeting was held with the Scrum Master at Company X where the application was reviewed thus far. This meeting involved the discussion of the requirements and the user stories and involved a brief demonstration of the application. Based on the application demonstration and the desire to include the “map view”, it was decided that the application would instead focus on booking rooms from the “map view” as opposed to the previously defined “search rooms” page. This ultimately resulted in a change to the user stories.

The original intention was to implement the “map view” page as it was completed earlier in the practical experiment 5 (section 3.2). However, based on the envisaged functionality a new approach was implemented. This was due to the future implementation of a “status” method, which would eradicate the need for a “timeout animation” as defined in the practical experiment. However, the process of drawing the map was simplified in order to allow for maximum modification. For both the practical experiment and iteration four, the maps which were drawn would be mock-ups of the floor plan of Company X which would be used to demonstrate functionality. This will eventually be modified to be an accurate representation of the floor plan of company x.

The “map view” was drawn as a scalable vector graphics (SVG) image in Adobe Illustrator, and exported as a web-enabled SVG file. This eliminated much of the labour which would be required to draw the room objects using HTML5 canvas or similar graphics elements, as it removed the need to explicitly “plot” the coordinates of the shape. The exported SVG file was simply opened in the web browser and the svg section was copied and pasted into the HTML of the “map view page”. This was a relatively straight forward process which required no extensive modification.

As well as the “map view” page, a “main page” was created. This was the main entry point of the application which is shown at application start up. This page features an “Information” button and a “Book a Room” button. The “Information” button enables a popup form, which displays the application instructions. The “Book a Room” button simply links the user to the “Map View” page.

This iteration resulted in the development of a “fuller” application, in that the application now no longer revolved around the testing of concepts and technologies, but instead involved the creation of the main page and the various other pages which were required in the application scenario. The following pages were created as a result:

- Main Page: This is the main page of the application which is shown at application start up. This page contains two buttons, a “Book Room” button and an “Information” button.
- Room Booking Map: This is the “map view” page, which displays a map of available “rooms”.

The implementation of the above “pages” implemented some of the features indicated in two of the main user stories. These were “Search” and “Map and Calendar View”.

PlayBook Application Iteration 4

This was the final iteration of the PlayBook application, which added more functionality to the previous application iteration in order to provide a final working prototype which was integrated with the WCF service application and displayed the basic functionality as defined in the user stories.

The main additions to this iteration were the “Calendar View” and “Booking” functionality. Originally, as indicated in the interim report, the booking was to be contained on a separate page or a “checkout” page. This was originally implemented into the project, but was eventually changed in order to minimise the amount of pages required to complete a process.

This iteration saw the implementation of the “Calendar View” page, which was implemented as a way to display the existing bookings for a particular room. The calendar was implemented using a jQuery plugin called “FullCalendar”, which provides a full size calendar which supports Ajax requests in order to populate the calendar with “events”. The calendar was implemented very easily but provided problems when trying to populate the calendar with actual data. Based on research conducted, the most feasible way of getting the live JSON data into the calendar was by using the “getJSON” function of jQuery. However, in order to collect the data, the correct “room” values had to be passed from the map view page, into the booking page. For instance, if a user clicks “Room 1” (which is not identified by number but instead by location on the map) the data for room 1 must be loaded on the next page. This was achieved by inserting a URL parameter for each room as follows:

```
$("#room2click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=2";
   });
```

This allowed the room id to be passed between the pages. This value was collected in the calendar view page by using the following JavaScript function:

```
function getUrlVars() {
      var vars = {};
      var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi,
function (m, key, value) {
         vars[key] = value;
      });
      return vars;
   }
```

This function basically “unwraps” the URL room value, by removing the surrounding URL characters and returning the integer value, in the above example, “2” would be returned. In regards to the calendar, this URL value would be submitted as part of an Ajax “getJSON” function, where the Ajax method would “GET” all bookings for room id “2”. Due to a variety of technical issues and time constraints, this feature was never fully implemented. However, the “Bookings” were simulated on the calendar page by using a “hard-coded” array of booking events, which used actual booking values as given in the database set (Appendix D).

The booking functionality was added to this page by adding a dialog form to the “onclick” event of the calendar. When a user clicks anywhere on the calendar body, i.e. the “date” cells, a popup dialog form will ask the user if they wish to book a room. When the user clicks book a room another dialog box is presented which contains input fields for “Booking Name”, “Date”, “Booking Start” and “Booking End”. Originally, this page also contained a “Room Id” input field where the user could input the room in which they wished to book. This was ultimately removed and the room id is dynamically inserted from the “getUrlVars()” function as given above. When the user populates the form with the required values, the data is serialised in JSON format in order to be submitted to the WCF service. This is done in two steps. In the first step, the values of the input fields are “inserted” into a JSON string as follows.

```
var data = { "BookingEnd": "\/Date(" + dateTimeEnd + ")/", "BookingID": 1,
"BookingName": $("#empname").val(),"BookingStart": "\/Date(" + dateTimeStart + ")/",
"RoomID": first };
```

The second step involves “Stringifying” the JSON data using the “JSON.stringify()” function. This is basically the conversion of the manual JSON string into a valid JSON string which will be valid for submission to the WCF service. After the data is correctly serialised, the booking is submitted via an Ajax “POST” request. If the booking is successful, a dialog form will indicate that it was successful, and if the booking failed, a dialog box will indicate that the booking has failed.

After the booking form was successfully completed, the booking form was augmented to offer a higher degree of usability. In order to test the booking Ajax function, the data was simply typed into each of the input fields. However, in a real situation this would not be acceptable. On the PlayBook tablet, when a user clicks an input field, an onscreen keyboard appears which allows the user to input information. This was problematic in two ways. The first problem was that the users were required to input the exact format as required by the web service. The second problem was that the appearance of the onscreen keyboard often resulted in certain page elements being obscured. When testing in the PlayBook application, the keyboard was often found to cover the input field, resulting in the user being unable to see what is being typed. This problem was solved by implementing two jQuery UI input features. These were “Datepicker” and “Timepicker”. Instead of the user having to type in the dates and times manually, all the user has to do is select the input field, and a “Datepicker” or a

“Timepicker” will appear. The “Datepicker” is simply a small calendar which appears and allows the user to select a date. After the date is selected, it is formatted into the correct format as required by the WCF service. The “Timepicker” is similar to the “Datepicker” however, the “Timepicker” allowed the user to specify a booking time. Originally, this was to be done with “sliders” which allowed the user to select the time they wanted by sliding a “slider” which incremented the time. This was found to be a laborious and somewhat confusing implementation and was altered to be similar to the date picker by containing “time cells” which represented the time values as follows:

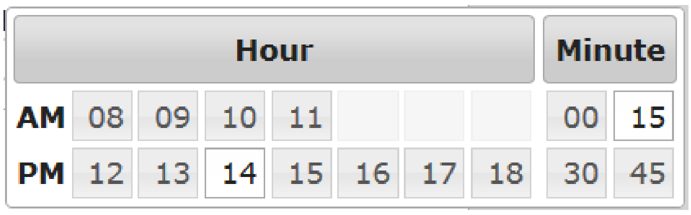


Figure 15 – Timepicker

With regards to the original requirements (section 3.3.2), the rooms were allowed to be booked in 15 minute increments. As such, the minutes were only allowed to be selected in 15 minute gaps. The hours represented the office opening times which are 08:00am to 19:00. Using these input methods reduced the labour required to enter the times manually and ultimately simplified the process of adding these values. This was in line with one of the main design motivations identified in literature review section 2.2, simplicity.

Another issue was identified when creating this application iteration. This was an error when submitting dates to the database. Originally, the date would be submitted to the WCF service in the format “yyyy-MM-dd hh:mm:ss.sss”. This date format was valid, in that the WCF service would accept it and post the information into the database without error. However, inspection of the inserted values in the database showed that the dates being inserted were “1970-01-01 00:00:00.000”. Research into the issue indicated that the date required parsing before being submitted into the database. Several failed attempts at resolving this issue were attempted in the WCF service application using the C# “DateTime.Parse” method. This allowed the DateTime value to be passed into the service as a string and would be serialised as a valid DateTime for the SQL insertion. The WCF solution failed in multiple attempts.

The problem was eventually resolved by converting the DateTimes to millisecond datetimes, which was successfully tested as a valid DateTime using Fiddler2. The milliseconds datetimes were created by implementing a method to calculate the milliseconds from the value the user specified. This function is as follows:

```
var d = $("#date").val();
var dateParts = new Date((Number(d.split("-")[0])), (Number(d.split("-")[1]) - 1),
(Number(d.split("-")[2])));
```

```
var dateis = dateParts.getTime();

var timeEnd = $("#endtime").val();
 var time1 = ((Number(timeEnd.split(':')[0]) * 60) * 60 + Number(timeEnd.split(':')[1]) * 60)
* 1000;

var timeStart = $("#starttime").val();
var time2 = ((Number(timeStart.split(':')[0]) * 60) * 60 + Number(timeStart.split(':')[1]) * 60)
* 1000;

var dateTimeEnd = dateis + time1;
var dateTimeStart = dateis + time2;
```

The above function basically "collects" the date value based on the inputs of the user. These values are then "split" to remove the dashes(-) and colons(:) from the format input from the date and time pickers. After the numbers have been separated, the datetimes are calculated by calculating the milliseconds. For instance, in the "var timeStart" section above, a value such as 09:30 may be input. This number is then split into two numbers, an hours number "9" and a minutes number "30". The above calculation multiplies the hours number by 60 to get the minutes, and then again by 60 to get the seconds of the minutes. This is then added to the minutes section, which is multiplied by 60 in order to get the seconds in the minutes. The result of the addition is then multiplied by 1000 to get the millisecond time format. The date is calculated by utilising the "getTime()" JavaScript function which calculates the milliseconds time automatically. This is not an option for the "times" as they require manual conversion. After the times are calculated, the "date" millisecond value and the "time" milliseconds values are added together to form the "start time" and the "end time". This method, when inserted into the JSON string and passed into the web service inserts the correct booking times.

4.2.2 WCF Service Application

This section will detail each of the iteration of the WCF service application which was completed throughout the project lifecycle. Each of the iteration descriptions will include a brief description of the work completed in the iteration, as well as highlight any of the major issues or technical solutions identified in the iteration. The code listings for the WCF service application iterations can be found in Appendix F.

WCF Service Application Iteration 1

The first iteration of the WCF service application involved the creation of a basic WCF service which contained a "hard-coded" list of "room objects". The main objective of this iteration was to learn the basics of WCF services in order to satisfy the "Maintain Rooms" user story task "Create WCF project and learn the basics" (Appendix C). This WCF service application was very simple in nature and did not contain any layers of abstraction. The functionality of this application was very simple, it allowed for a collection of “room objects” to be returned from the service as an array of JSON objects as follows:

[{"Capacity":20,"Facilities":"Computer, Projector","RoomId":1,"RoomName":"Gigantic Room","Status":false},{"Capacity":10,"Facilities":"Projector","RoomId":2, "RoomName":"Big Room","Status":true}]

The room objects were stored in a C# static class which was called “RoomRepository” (appendix ?). In addition to the ability to retrieve objects, the ability to create room objects was implemented. This was done by inserting the values into the browser URL which would then be interpreted by the service URI string. The WCF service would then accept the values and insert the room object into the room repository as a new “Room” item using the “SaveRooms” method in the service class (svc.cs). The URI string for inserting the room values are as follows (Appendix F):

> http://localhost:6188/RoomBookingService.svc/
> SaveRooms?roomid={roomid}&roomname={roomname}&capacity={capacity}&facilities={facilities}&status={status}

The above method is a crude way of inserting data into the service application. To insert a room in the above format, the values represented in the curly brackets would be replaced by the value type as indicated in the “Room” class (Appendix F). For example, the following would be used to insert a room (indicated in red):

> http://localhost:6188/RoomBookingService.svc/
> SaveRooms?roomid={9}&roomname={“Large Room”}&capacity={16}&facilities={“Smart Board, Whiteboard”}&status={false}

At this point in the project, there was originally to be a “Room Status” attribute, which would indicate the current status of a booked room. This was given as a Boolean value, with true representing “Available” and false representing “Booked”. This was later removed from the application and was proposed to be implemented in a later iteration.

This iteration of the application may be considered one of the most important of the project as it proved a few of the concepts which were the basis for the project. These were the ability to “get” data from the web service, and the ability to “insert” data into the service. This proved the concept of “booking a room” as it was proven that objects can be retrieved from a repository of sorts, and that objects can be committed to a repository.

WCF Service Application Iteration 2

Following the first WCF service application, the second iteration aimed to build on the existing features of the first. After the development of the first application, meetings were conducted in order to discuss the next step regarding the development of the service application. There was a desire from all parties to see the room data persisted to a database or related data source.

This application iteration involved the implementation of database storage to the service application by modifying the existing application code in order to accommodate the database of choice, in this case, SQL Server 2008. The database at this stage consisted of a single table called “Rooms”, which was hosted locally. The database connection was implemented into the existing “RoomBookingService.svc.cs” file (Appendix F). In this iteration of the application, only the “GetRoom method” was implemented. This was achieved by connecting the database source using an SQLConnection string. This was declared within the method, which was later discovered to be poor practice. The method worked by opening the connection to the database and performing a simple SQLCommand (“Select * from Rooms). The results were returned to the WCF service application and were interpreted by the SQLDataReader, which formatted the data as a “Room” object as defined by the “Room” class.

Furthering the development of the last iteration, this iteration also proved a significant concept and a new solution to storing and accessing the information required of the project. This application iteration proved the concept of using SQL Server Database the form of object persistence for the application. However, only the “GetRooms” method was proved. At this stage further work was required to implement the “Booking” method, which would serve as one of the most important criteria for the successful completion of this project.

WCF Service Application Iteration 3

The third iteration of the WCF service application was a direct expansion from the last iteration. This application iteration involved the implementation of the “Booking” method using SQL Server databases.

Implementing the “Booking” method using SQL had proven to be one of the most difficult development tasks in this project and the issue carried across for several application iterations. In this iteration, the WCF service application was modified to include a new “Booking” method, which was similar to the “GetRooms” method as discussed in the previous section (Appendix F). Both methods use the same basic approach such as defining a connection string (or using a globally declared connection string as was utilised in later iterations) and opening and closing the connection string. The main difference, however, was that the “Booking” method involved the procedure of passing values into an SQL query, which would then be sent to the database.

This iteration was semi-successful, as the concept of inserting values into the database was proven. However, this was achieved by using a Query with “hard-coded” values as follows:

```
INSERT INTO Bookings(booking_id, room_id, emp_name, date, start_time, end_time)
VALUES (3, 2, 'Michael', '2012-02-30', '12:00:00', '13:00:00’)
```

In regard to the concept of sending data to the database, this iteration proved the concept of inserting data into the database. In order for this to become a functional web service, the “insert” query had to be developed in order to allow values to be passed from the front-end application and be successfully submitted to the database.

WCF Service Application Iteration 4

Following the difficulties which were presented from the last iteration, namely regarding the process of passing values into the WCF service application and having them inserted into the database, this application iteration assessed a potential alternative solution to the WCF service application as used thus far.

The solution researched as an alternative to the WCF service application was the use of a WCF data service using OData. WCF data services differ from WCF service applications in a number of ways. The most notable difference which uniquely defines WCF data services is that the WCF data service is used to expose a data model as a RESTful web service (Ukleja 2009). WCF data services expose the data from the data model as “raw” data sources. The implementation of this method allowed for JSON formatted (from the PlayBook application) data to be send into the database with minimum effort. This approach involved the generation of a data model, which is based on the SQL Server database. In the WCF data service application, the data was exposed by configuring an access rule similar to the following:

```
config.SetEntitySetAccessRule("Rooms", EntitySetRights.AllRead);
config.SetEntitySetAccessRule("Bookings", EntitySetRights.All);
```

The “EntitySetRights” attribute sets the level of access to the data. In the above example, the “Rooms” table was set to “AllRead”, which allowed all read operations to be performed, but none of the write options to be performed. The “Bookings” table, as indicated above was set to “All”, which allowed both read and write operations to be performed. This data was accessible in a similar fashion to the WCF service application. The WCF service application exposed the data through “endpoints” such as the following “GetRooms” example:

http://localhost:6188/RoomBookingService.svc/getrooms

The WCF data service service exposed the data in a similar manner, but exposed entire tables and instead relied on the front-end to sort the data which was required. The data was exposed (in the case of the “Room” table) using the following URL:

http://localhost:6239/RoomBookingDataService.svc/Rooms

Unlike the WCF service application, which returned the data in JSON format as defined in the service operations, the WCF data service exposed the data as XML or atom publishing format. However, the data could be inserted in JSON format by specifying the Ajax data type as JSON in the web application.

After this concept was proved, it was decided that the WCF data service would be the best way to continue forward with the project. However, when the initial testing involving the PlayBook application and the service was conducted, a problem was discovered regarding cross domain scripting. It was found that the service was inaccessible from the Ajax application as it was hosted on a separate domain. The problem was eventually alleviated by hosting the application in the same domain. This was not an acceptable solution however, as the application would inevitably be implemented in a cross domain scenario. Following this, it was decided to try again with the WCF service application and try and implement the “Booking” method as originally intended.

WCF Service Application Iteration 5

This application iteration essentially “followed-on” from application iteration three, as the results of application iteration four required a return to the original problem. The main problem which was occurring was in the process of submitting values to the WCF service application and passing them to a database as a “Booking” object. Many attempts to resolve the problem yielded no results; however, after vigorous research the problem was eventually solved.

The main problem with the original attempts at implementing the feature was that the WCF service application was not really receiving any data. When submitting data to the WCF service application, an Ajax “POST” method was used to submit the data as a JSON string. The WCF application would successfully receive the information, but no procedures were in place to deal with it. In order to successfully “use” the data that had been sent, the WCF service application had to be expanded to handle the booking request. This was achieved successfully by changing the data received to an instance of the “Booking” class. Originally, the operation contract defined the value types of the values being submitted, in a similar fashion to the very first iteration. By changing the expected values to a Booking class object (void CreateBooking(Booking booking)), and by inserting the values (defined in the data contract class) into the “Insert” command, the booking feature was eventually implemented as follows:

```
public void CreateBooking(Booking booking)
   {
      Booking newbooking = new Booking();
      sql = new SqlConnection("Data Source=comp;Initial Catalog=BookingDB;Integrated Security=True");
      sql.Open();

      string command =
         ("INSERT INTO Bookings( BookingName, BookingStart, BookingEnd, RoomID ) " +
          "VALUES ("
            + "'" + booking.BookingName + "'" + ", "
            + "'" + booking.BookingStart  + "'" + ", "
            + "'" + booking.BookingEnd + "'" + ", "
                 + booking.RoomID + ")");

      SqlCommand cmd = new SqlCommand(command, sql);
      cmd.ExecuteNonQuery();
   }

   public void Close()
   {
      sql.Close();
   }
```

Eventually, based on advice received, the above query would be parameterised, which would ultimately deter SQL injection attacks. At first, the idea of SQL injection attacks was a non-issue, as it was assumed that the PlayBook code would be inaccessible to anyone other than developers for Company X. However, regarding the original specification, one of the requirements of Project X, is the prospect of web applications (section 3.2).This in turn means that the application could eventually be ported to the web, where SQL injection attacks become more of a serious issue, especially where a large company database is concerned.

WCF Service Application Iteration 6

The final iteration of the WCF service application involved the implementation of layers of abstraction and a general improvement of all existing code. Based on the research conducted and the advice received, the final stage in making the WCF service application would involve the inclusion of layered architecture. In the case of this project, the WCF service application was expanded to include two new "Model" classes, which separated the data access logic from the business logic. A "model" class was created for each Class of the application. This resulted in a "RoomsModel" and a "BookingModel" being created. Each of the model classes contained all of the data access logic required of the application. The main reason for the layering of the architecture is in the interest of changing datasets without having to change a lot of the code. As indicated by the early requirements, one of the interests in this project was to investigate the Microsoft Outlook API. Company X was interested in using Microsoft outlook in place of the SQL server database. By layering the application, the implementation of new data source would simply be a case of "tearing-out" the current data source in favour of a new source.

Other advances to the service application in this iteration, as previously mentioned were the parameterisation of queries in order to deter SQL injection attacks. Although this is not an immediate issue considering the application is only accessible from the PlayBook device, the company expressed the potential for the application to be ported to web. As such, the parameterisation of SQL queries will prevent any injection attacks from occurring when conducting Ajax "POST" methods. Only the "CreateBooking" method was parameterised, as it is currently the only method which allows data to be submitted to the WCF service. All other methods utilise the "WebGet" operation contract attribute, which is purely a "GET" method and can only result in the retrieval of data.

4.3 Final Implementation

This section details the main implementation of the application after the completion of the iterations. As detailed in the previous section, the application was completed by undertaking several iterations of each application element, each iteration advancing on the last. This section details the "final" implementation of each application and will provide greater detail of the development and workings of the final prototypes.

4.3.1 BlackBerry PlayBook Application

The final implementation of the BlackBerry PlayBook application resulted in the successful development of a “front-end” application, which when coupled with the successful development of a WCF service application, allowed a user to successfully book a room. The above section details all of the iterations of the BlackBerry PlayBook application which were completed throughout the project. These iterations resulted in two different applications being created. The first of the applications was focussed around a “grid” view of the rooms which could be booked, and the second application focussed around the “map” and “calendar” views.

Application Data

The main data format used in the application is JavaScript Object Notation (JSON). JSON is a lightweight data-interchange format (json.org). One of the major benefits to using JSON was that it is easy for humans to read and to write. This proved true, particularly regarding the way the “Booking” data was formatted for submission to the WCF service. The very first data exchange test (practical experiment 2) revolved around the use of both XML and JSON data, and resulted in the selection of JSON data. JSON appeared the better choice due to its support within Ajax and WCF, two of the main development technologies. The selection of JSON proved to be highly beneficial as when the cross domain issue had been highlighted, the use of the “getJSON” method in jQuery allowed for the cross domain retrieval of the JSON data. In addition to this, as mentioned, the use of JSON greatly simplified what had become the “manual” processing of a booking particularly when testing the application using fiddler. This was achievable as JSON is a highly legible data format which does not require deserialization in order to understand. The following is an example of a JSON booking string.

```
'{"BookingEnd":"\/Date(1332420656202+0000)\/","BookingID":1,"BookingName":"client sent","BookingStart":"\/Date(1332334256202+0000)\/","RoomID":2}'
```

Ajax Requests

In order to retrieve the application data from the WCF service, Asynchronous JavaScript and XML (Ajax) was used. Ajax allows for data to be sent and retrieved from a web service asynchronously, without reloading the page (w3schools). This was selected based on the initial research conducted and the practical experiments conducted prior to the main project development. The main Ajax requests used in the PlayBook application are “GET” and “POST”.

In the early iterations of the project application, the room data was displayed on a “grid” provided by jqGrid. In order to populate this grid, the data had to be retrieved from the WCF service. This was achieved by using Ajax “GET” requests. The “GET” request allowed the room data to be retrieved by specifying a URL, in this case the URL of the WCF service, which would return the data as requested by the Ajax method. Further into the iterations, certain problems presented in the display and retrieval of the JSON data. These problems were eventually solved by using the jQuery function “getJSON”, which is an Ajax “GET” request that allows for a “callback” parameter to be submitted to the service in order to retrieve the JSON data (see PlayBook iteration 1).

In addition to the “GET” method detailed above, the other Ajax request used in the application was the Ajax “POST” request. The “POST” request was used for all of the “booking” functions in the application. The Ajax POST method works by sending data to the web service via a URL. The following is an example of a “POST” request as utilised in the application.

```
$.ajax({
        type: "POST",
        url: " /RoomBookingServices/RoomBookingService.svc/createbooking",
        dataType: "json",
        data: booking,
        contentType: "application/json; charset=utf-8"
```

The above Ajax request specified the request type as “POST” and specifies the URL in which to submit the request, in this case the URL of the web service. In addition to this, the datatype and content type are specified as JSON, and the data itself is specified. The data in this case is “booking”, which represents the “Stringified” JSON as constructed after the form is populated (see PlayBook Iteration 1).

4.3.2 WCF Service Application

The final implementation of the WCF service provided most of the main functionality as required by the project. When coupled with the BlackBerry PlayBook application, users were able to successfully create a booking for a specific room and have the booking persisted onto a SQL server database. As was the case with the BlackBerry PlayBook application, this application was created by completing several iterations of the application. For the most part, these application iterations revolved around the creation of a WCF service application, with one iteration being dedicated to the exploration of a WCF data service using OData (see WCF iteration 4).

The final WCF service application provided functions for getting data from the database, and inserting data into the database. The only piece of functionality which was originally intended but excluded in the interest of time was a “getById” method. This was to be implemented around the time the FullCalendar was implemented, and would involve “getting” the bookings of that specific room via roomid. This is a relatively simple process which would involve the expansion of the existing “getbookings” method to accept an Ajax parameter which would be used to return all bookings belonging to the specific roomid. The room id, as mentioned, is passed between pages as a URL parameter which can be retrieved using a simple JavaScript function as detailed in PlayBook Iteration 4.

The final methods which exist in the WCF service application are “GetRooms”, “GetBookings” and “CreateBookings”. Both of the “Get” methods retrieve the data from the SQL server database by performing a simple “Select * from [table]” query. This would be modified to include a parameter in place of the “*” should the “getById” method be eventually implemented. The “CreateBooking” method inserts data into the database by using an “Insert into” query. This query was later expanded to “check” for existing bookings by modifying the insert query to the following:

```
"IF NOT EXISTS ( SELECT * FROM Bookings WHERE BookingStart = @BookingStart
AND BookingEnd = @BookingEnd AND RoomID= @RoomID )
                    INSERT INTO Bookings ( BookingName, BookingStart, BookingEnd,
RoomID ) VALUES ( @BookingName, @BookingStart, @BookingEnd, @RoomID )"
```

This query essentially “checks” the database to see if an identical record exists and will either insert or abort the operation as required. If the record already exists, the user will be alerted in the front end application with the “error” form as described.

As mentioned in the WCF service application iteration 6 description previously, the query was parameterised to prevent SQL injection attacks. This is a crucial security measure based on the possibility of the application being ported to other web based devices and even desktop browsers. The application data will be stored on the database of Company X and as such it is particularly relevant in order to prevent the loss or theft of data.

The WCF service application employed a layered architecture. Since the application would eventually have the SQL server database replaced with an alternative data source, in the case of this project Microsoft Outlook, it is particularly important to abstract the application layers in order to aid in the easy modification of layers. Based on this, the SQL methods were contained in a data access layer, called the “room model” and “booking model” within the SQL service application. Each “model” contained all of the data access logic for the data transactions. This layer was accessed from the “RoomBookingService.svc.cs” class, which contained the main method logic. Each method created a new instance of the model class and passed in the method parameters as defined in the entity classes. The following diagram indicates the communication of the WCF service application in the context of the entire project (Figure 16).

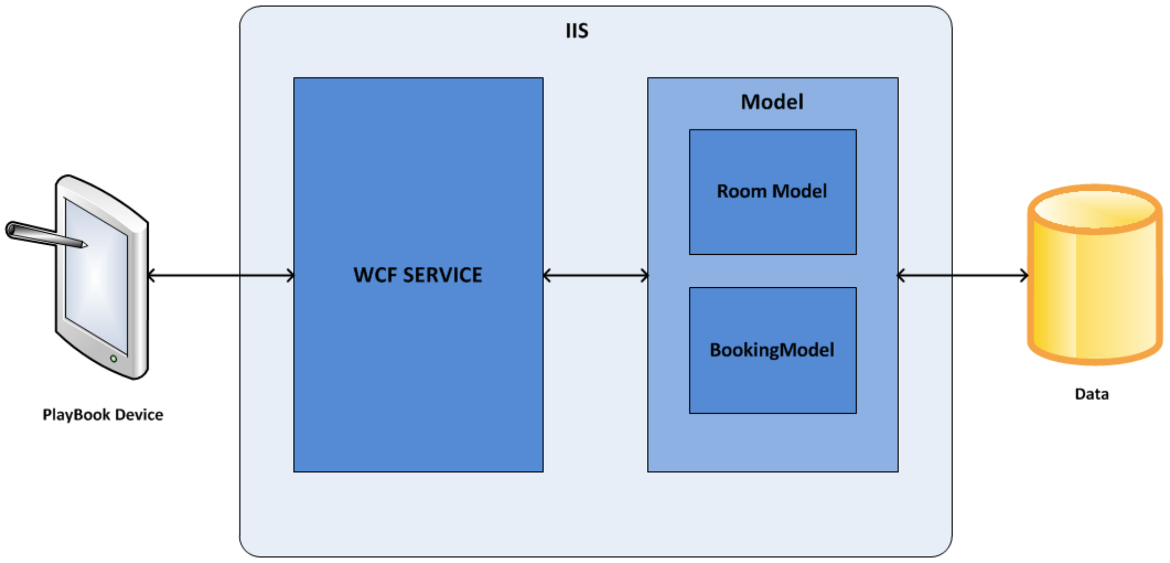


Figure 16 – WCF Service Application Architecture

The above diagram shows the working of the basic communication of the applications. The PlayBook application submits requests to the WCF service application, which then passes the information to a model which inserts the data into a database. As indicated in the diagram, the WCF service application is hosted in IIS. The following diagram indicates in more detail, the communication of the application in terms of what information is passed between the layers (Figure 17).

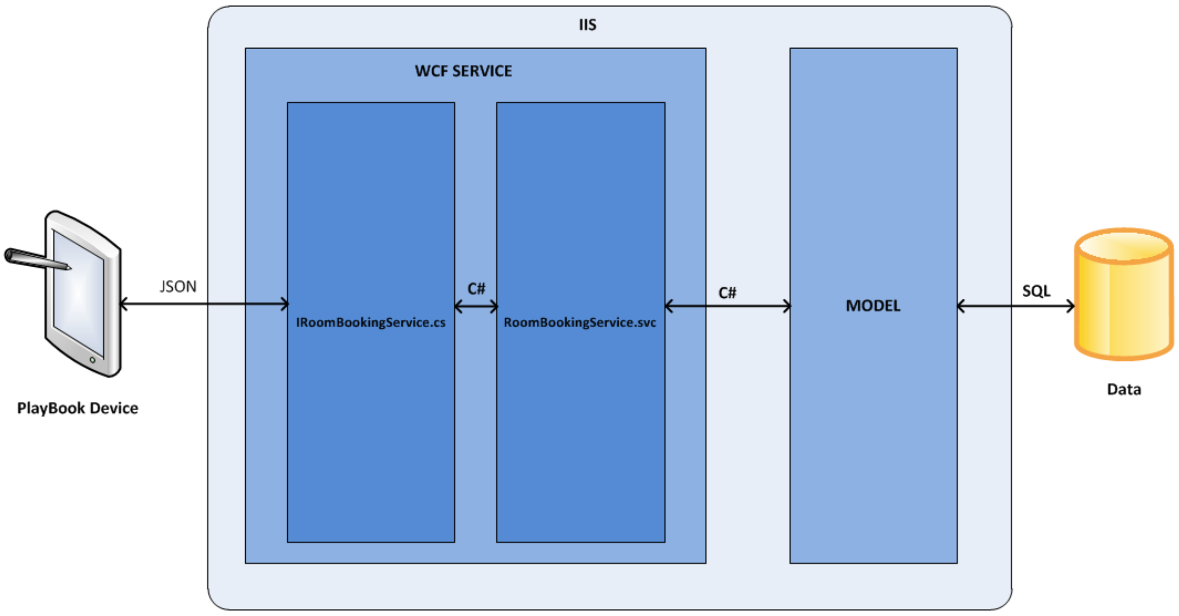


Figure 17 – Detailed WCF Service Application Architecture

This diagram indicates, in more detail, the communication of the layers of the application. In terms of a “booking” transaction, this can be described as follows:

1. An Ajax “POST” function submits a JSON string to the WCF service application.
2. This is received by a “WebInvoke” method in the WCF service interface class. The methods in the WCF interface class are either “WebGet” or “WebInvoke”. “WebGet” is used when a “Get” method is required, such as the returning of room information. “WebInvoke” is used for all other Ajax functions such as “POST”, “PUT” and “DELETE”.
3. The JSON values are passed to the RoomBookingService.svc as C# value types. This is handled transparently by the service.
4. A new instance of the appropriate model is created and the method parameters are passed as C# to be inserted into a parameterised query.
5. The values are inserted into the query, and the query is executed.
6. The user is alerted of its success or failure in the front end application.

The following activity diagram illustrates the implementation of the booking feature with regards to the overall communication of the application (Figure 18).

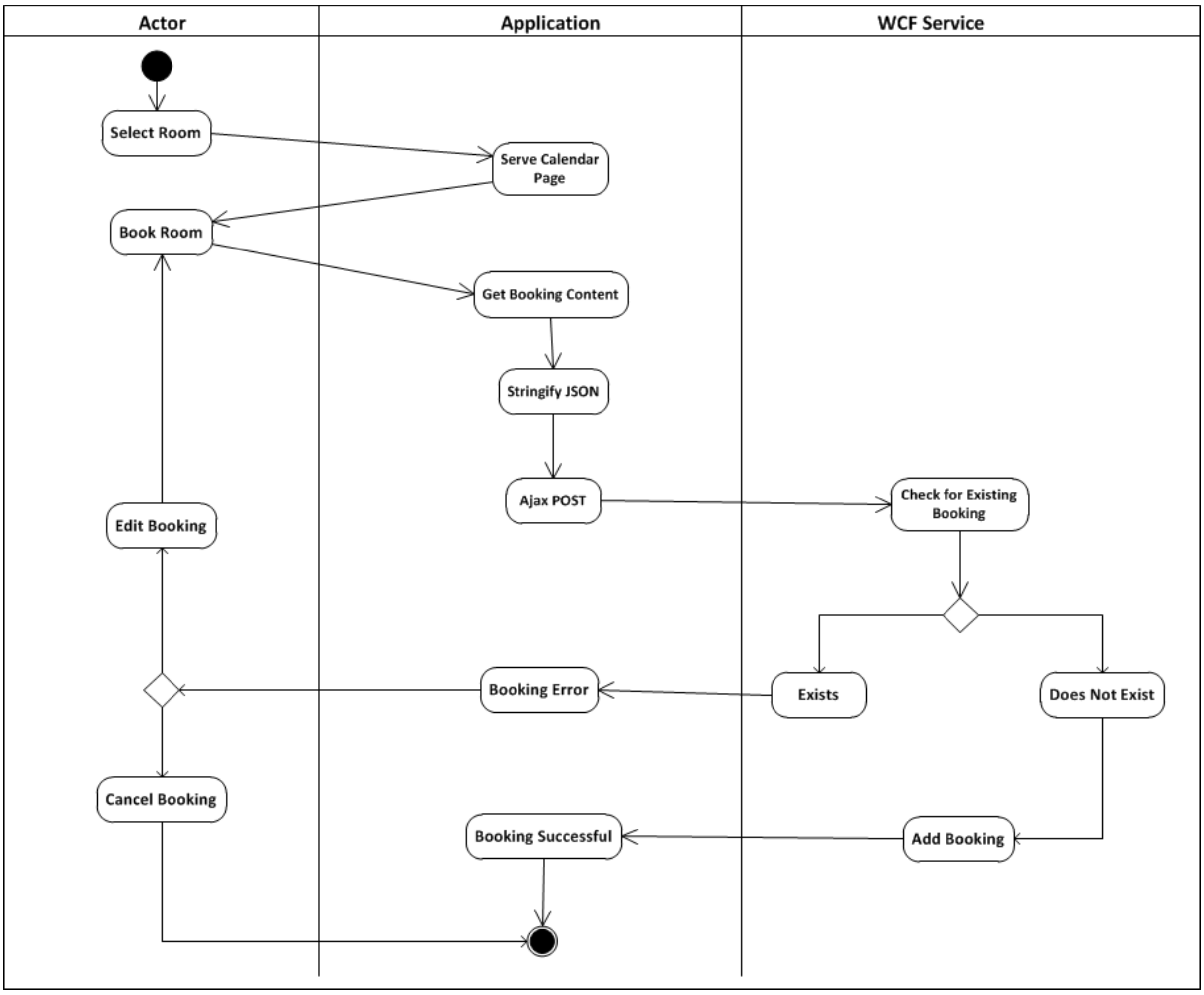


Figure 18 – Room Booking Activity Diagram

4.3.3 SQL Server Database

The database used to store all information regarding the application bookings and room objects was a SQL Server 2008 database. This was selected as it is the preferred database technology employed by company X. This in turn meant that the application technology simulated that of Company X closely, and also meant that the database could be incorporated easily in company X when required.

The database design for this project is very simple. It consists of a “Room” table, which holds all information of the “Room” objects, and also a “Booking” table, which stores all information of bookings made. The database schema is as follows:

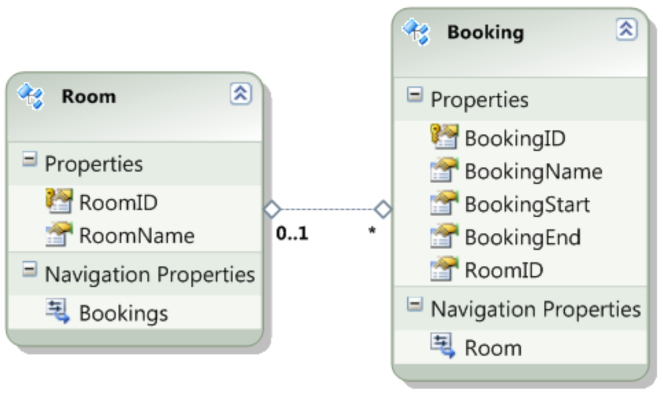


Figure 19 – Database Schema

The database has a one to many relationship, as one room can have many bookings. Each table has a unique identifier (RoomID and BookingID). The booking table has a foreign key value “RoomID”. The value types of the table attributes are as follows:

Room

- RoomID: int
- RoomName: nvchar(50)

Booking

- BookingID: int
- BookingName: nvchar(50)
- BookingStart: datetime
- BookingEnd: datetime
- RoomID: int

4.4 Testing

This section will detail the technical testing procedures which were used in order to successfully test the implementation of the project applications. This was completed in addition to the testing conducted during the stand-up meetings at the end of each sprint. The main forms of testing which took place were systems testing and quality assurance and acceptance testing.

In order to test the application as accurately to the deployment scenario as possible, the deployment conditions as specified by Company X were simulated. This involved publishing the WCF service application and hosting it within IIS as a “web application”. The PlayBook application was published to the application simulator and interacted with the WCF web application using a specified IP and port number. The final testing stages conducted late in the project involved setting up the project and running it within Company X. As the exact conditions of deployment in company x had already been simulated and employed throughout the application development, no errors were met when testing in Company X and demonstrating the application to the Scrum master and product owner.

When testing the WCF service application, the main tool which was used to test the functionality of the application was Fiddler. This is a free web debugging tool which logs all HTTP traffic. Using fiddler allowed for the inspection of traffic when "calling" the WCF service application. One of the main benefits to using Fiddler is that it allows for the "fiddling" of incoming or outgoing data (Fiddler 2012). This allowed the data requests to be altered in order to confirm functionality or errors (fault injection). Fiddler allowed for the simulation of Ajax requests without having to use the application. This was particularly useful as in certain circumstance the Ajax requests of the PlayBook application were behind the development of the WCF service application.

The Fiddler tests involved "composing" HTTP requests in order to test the functionality of the service. Since the WCF service application revolves around "GET" and "POST" Ajax methods, these were the main types of HTTP requests tested. The Fiddler tests work by calling a URL (WCF service URI) using one of the HTTP request types such as GET or POST. Fiddler also allows for custom headers in order to test the collection of data formats. POST requests allow for a request body to be submitted. This was useful in testing the "CreateBooking" method as it allowed for a JSON string to be submitted to the service in order to test both the JSON strings format validity and also the SQL insert command as specified in the application. This was particularly useful when checking the JSON strings as created by the application. As mentioned in PlayBook Iteration 3, there was a DateTime serialisation error in which the date being submitted to the service was in the wrong format. This involved a lot of modification to the jQuery function for getting dates. Instead of having to constantly modify and deploy the application in order to test, the JSON string was output to an alert box, and copied and pasted into the Fiddler request body panel in order to test the validity of the string. The Fiddler tests were an on-going process throughout the project and were continually used as a means to validate the functionality of the WCF service application. Included in Appendix G is the Fiddler tests conducted for the final implementation. The previous fiddler tests were identical and yielded identical results.

The main testing procedures conducted with regards to the BlackBerry PlayBook application was usability testing and system testing. As the PlayBook application was almost fully reliant on the WCF service application, system testing formed the main basis of testing the functionality of the PlayBook application. In addition to this general usability tests were conducted in order to test the usability of the application and its interface. The systems testing conducted involved both the PlayBook application and WCF service application. Test cases were used to define the tests and the criteria of the tests. The test cases were based on the acceptance criteria of the user stories (Appendix C). The PlayBook system testing was originally intended to be conducted across both the simulator and physical device. However, there were various technical issues which prevented the testing on the physical device. The main issues for this were due to the complicated deployment procedure as identified in practical experiment 1, and also due to limited access to the device. Although the simulator represents the PlayBook device software identically, it lacks the physical input, which could drastically change the outcome of the tests. The test documents can be viewed in Appendix H.

5 Evaluation

The evaluation section details the various evaluation techniques undertaken in order to test the hypotheses given in section 1.5. There were two major forms of evaluation undertaken in the project. These were Scrum meetings and usability evaluation through questionnaire. Following a discussion of each technique, an application “walkthrough” will outline the basic functionality of the application by demonstrating the process of booking a room.

5.1 Scrum Meetings

The project lifecycle selected for this project was a Scrum lifecycle. Throughout the lifecycle several “sprint meetings” took place which served as the main method of product evaluation throughout the project. Sprint meetings consisted of the application development team (the author), the Scrum master (junior developer of company x) and the product owner (development manager). The sprint meetings consisted of the following:

1. A discussion of the user stories implemented
2. A short demonstration of the application thus far
3. A code review
4. Quality assurance (QA) and product owner (PO) acceptance testing

The main stages regarding the evaluation of the application were code review and QA/ PO acceptance testing. The code reviews typically involved sitting down with the Scrum master and going through the code, analysing it for any obvious improvements or errors. Although this was very informal, it proved to be a very fun experience of evaluation which promoted the discussion of coding and development in general. The code reviews highlighted some specific issues with earlier iterations of the application. An example of this is the parameterisation of the WCF service application “Insert” query (section 4.2.2, WCF iteration 5). Originally, this query revolved around the direct coding of the parameters into the SQL string.

In addition to the code review, QA/PO acceptance testing took place in order to confirm and evaluate the delivered functionality of the application. As mentioned in section 3.3.2, due to the lengthy process of QA/PO acceptance and the limited availability of the Scrum team members, QA/PO acceptance testing took place on the second sprint meeting and the final sprint meeting only. The QA/PO acceptance tests were very similar to the system tests (Appendix H), in that they tested the functionality as defined by the user stories. These tests consisted of the validation and verification of the functionality as defined by the acceptance criteria of each user story. The QA/PO acceptance testing was quite an informal evaluation method, as the product simply “verified” the functionality against the expected acceptance criteria documented in each user story. As the system tests were created as a means for personal evaluation of the acceptance criteria, the system tests can be viewed as the outcome of the QA/PO acceptance tests (Appendix H).

As mentioned previously in section 3.3.3, the user stories involved creating time estimates for each of the tasks involved in each user story. This served two purposes, to identify weaknesses and to promote better time management. After the completion of each user story task, the actual time required to complete the task was compared to the estimate provided. In general, the actual times taken to complete tasks were greater than the estimations provided. This was not viewed negatively however, as it identified specific weaknesses and showed that there is no room for complacency regarding time management. The time estimations can be viewed on the user stories in Appendix C.

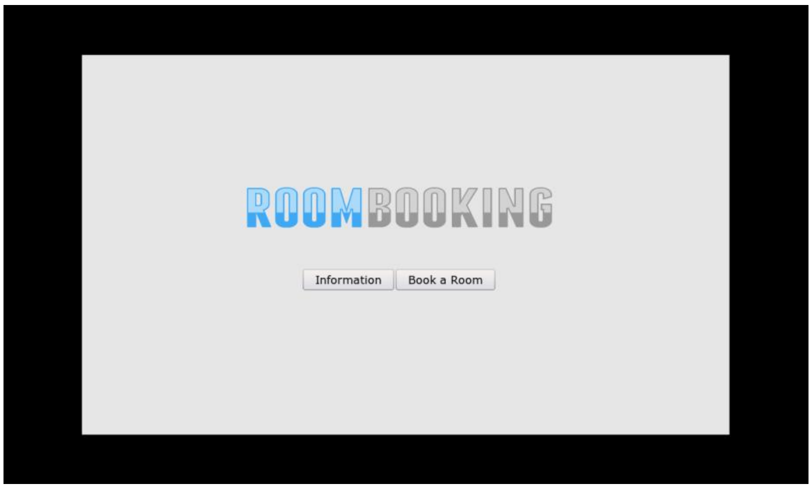
5.2 HCI Questionnaire

The originally intended method of usability testing, as defined in the interim report was to conduct the tests using the observation and interviewing method. However, this was found to be a difficult arrangement. This was largely due to the technical issues surrounding the deployment of the application to the PlayBook device as detailed in section 3.2. As an alternative to the observation and interviewing method, an HCI Questionnaire was created which was presented to students as a method of evaluating the project. The questionnaire consisted of 8 questions which aimed to assess the usability of the product and the user's preferences. As the application was unable to be presented to the users, the application screenshots were presented on an iPad tablet computer. The users were then "guided" through a sample booking using the steps provided in the application walkthrough (section 5.3) and asked to fill out the HCI questionnaire (Appendix I). The user group was a relatively small set; however, the completion of the HCI questionnaire did provide some valuable feedback on the project and also identified specific issues which could serve as the basis for future development. The HCI questionnaire template and the scanned results of the completed HCI questionnaires can be viewed in Appendix I and J.

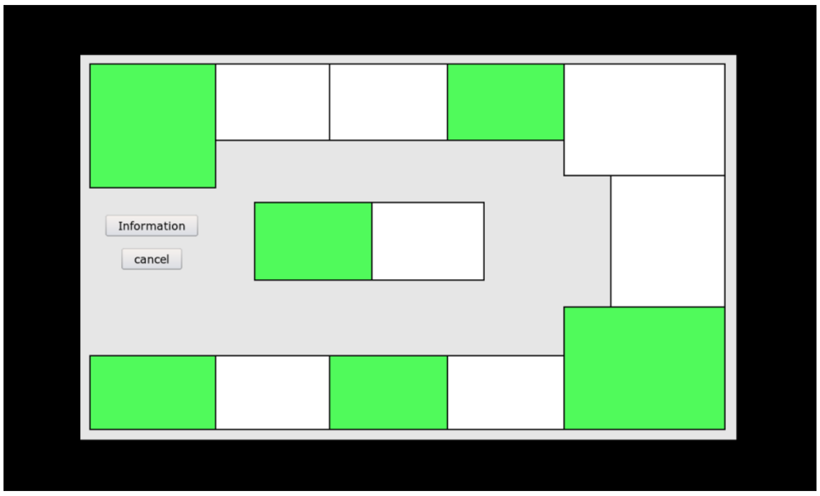
5.3 Application Walkthrough

This section will provide a walkthrough of the final application following the completion of the implementation. The walkthrough will detail a typical "booking" and will be illustrated with screenshots of the final application, which can also be found in Appendix D.

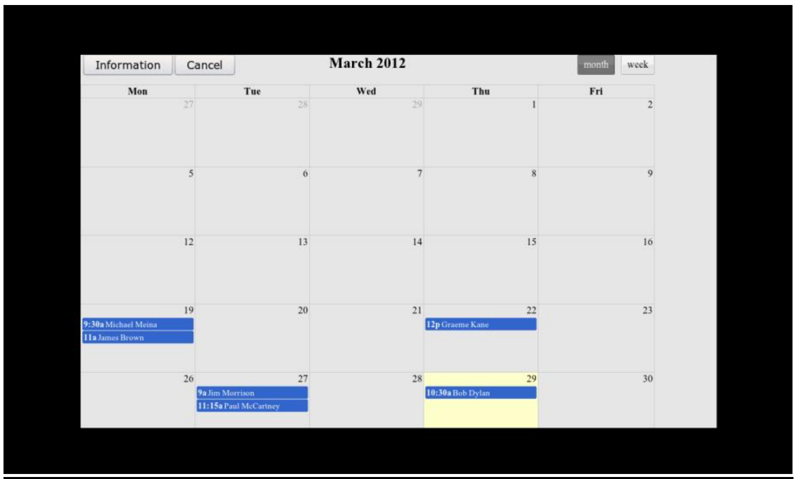
1. The user approaches the device and initiates the application from ready state.

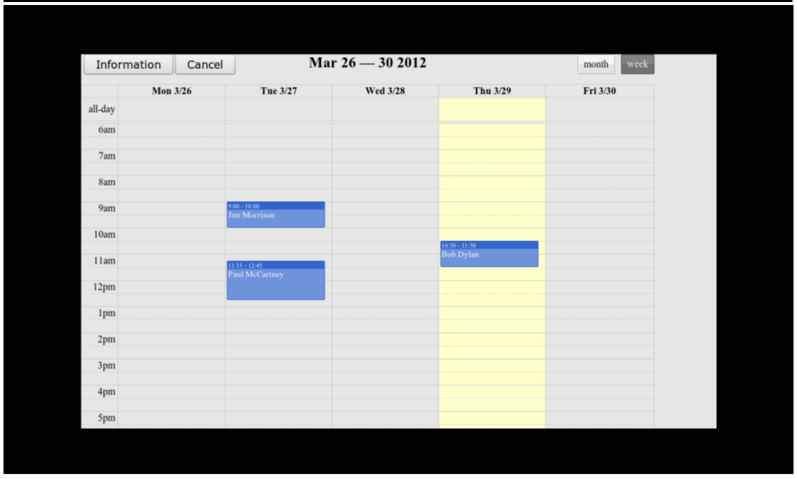


2. The user selects the “Book a room” button and is transferred to the “Map View” page.



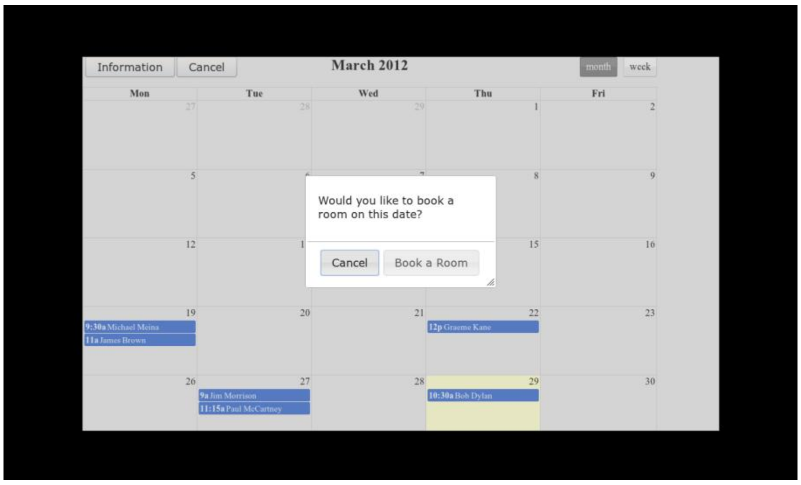
3. The user selects a “room” and proceeds to the “calendar view” page.



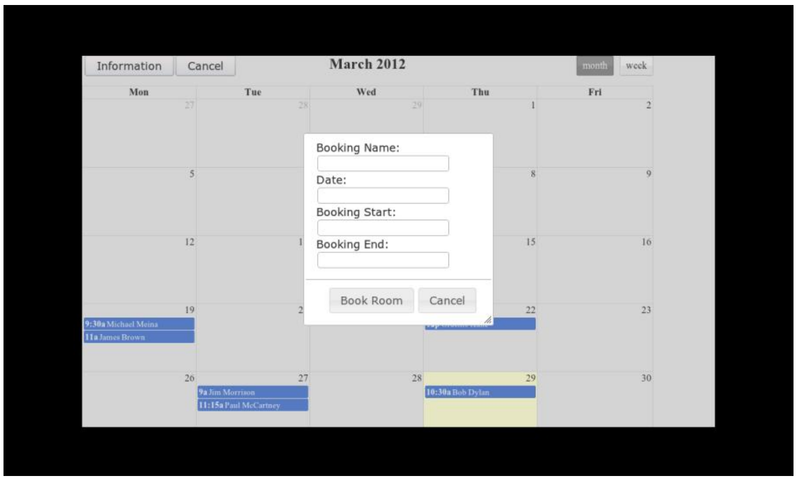


4. The calendar view page displays a calendar with “bookings” in order to indicate when the room is unavailable (see above).

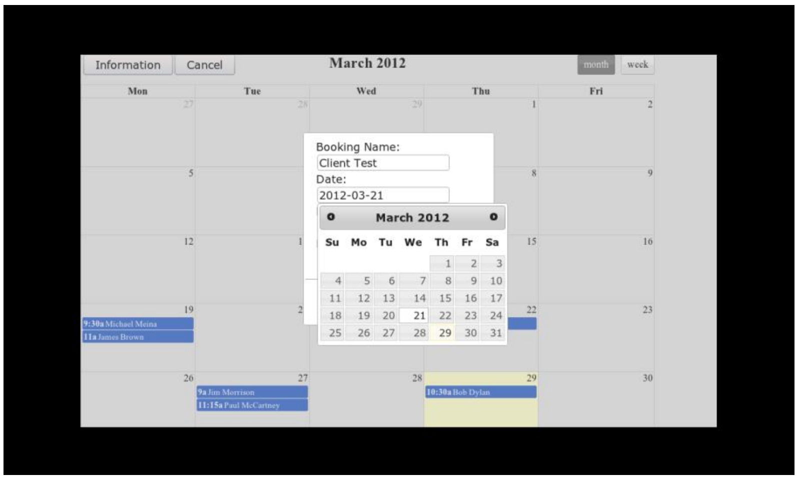
5. The user clicks a calendar cell, and is asked if they would like to book a room. If the book room button is clicked, a form will appear which allows the user to enter booking information.



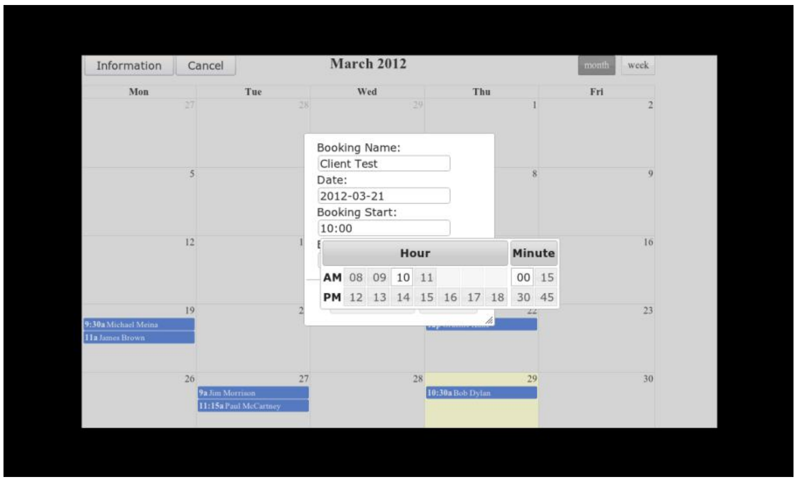
Would you like to book a room on this date?



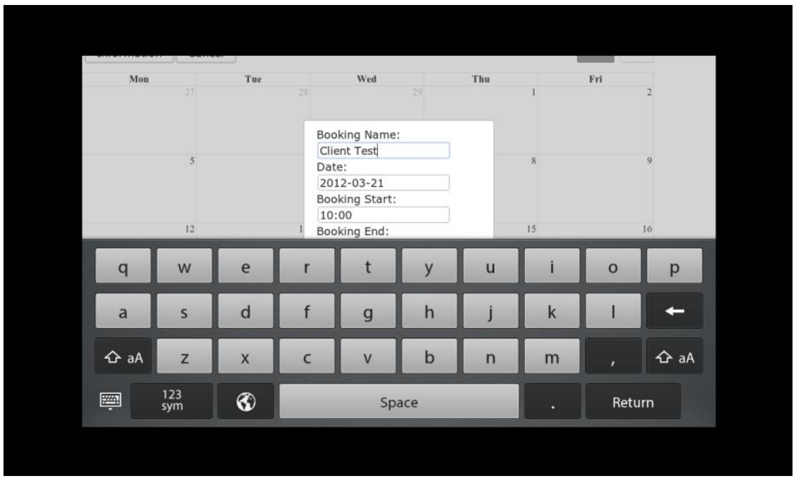
6. The user enters the booking information in the form and clicks the book room function.



Booking Name: Client Test
Date: 2012-03-21

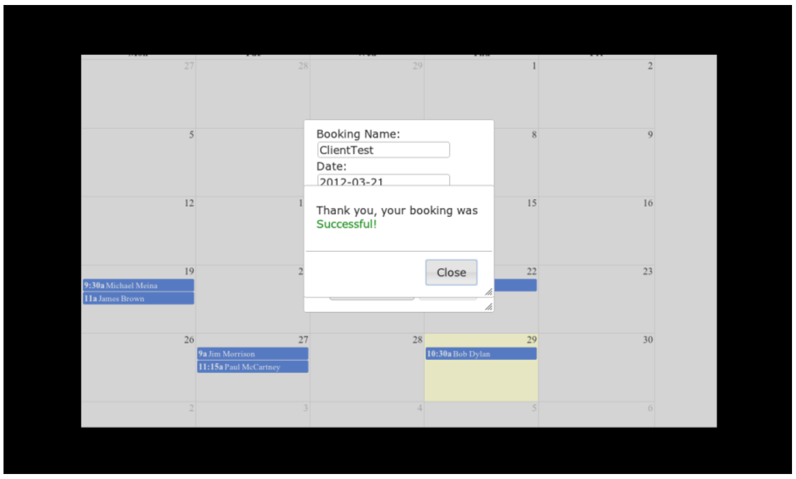


Booking Name:
Client Test
Date:
2012-03-21
Booking Start:
10:00



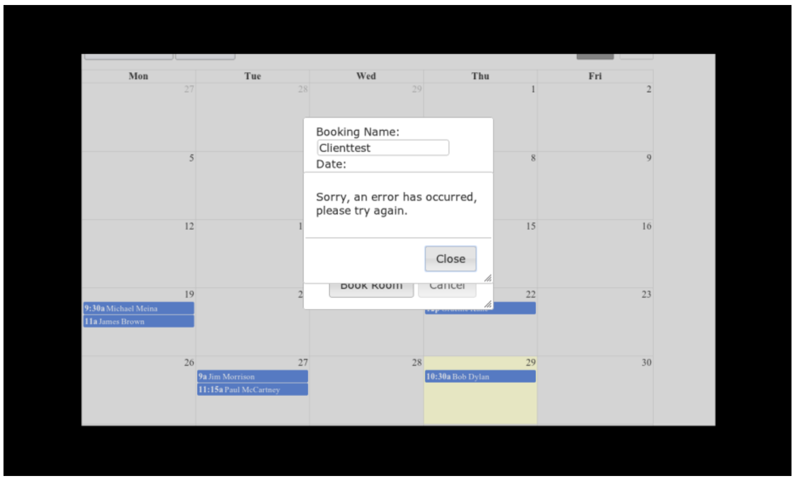
Booking Name:
Client Test
Date:
2012-03-21
Booking Start:
10:00
Booking End:

7. A confirmation form will appear which will indicate if the room was booked successfully, or if there was a booking error.



Booking Name:
ClientTest
Date:
2012-03-21

Thank you, your booking was Successful!



Booking Name:
Clienttest
Date:

Sorry, an error has occurred, please try again.

Success: "Thank you, your booking was Successful!"
Error: "Sorry, an error has occurred, please try again."

8. The user is returned to the start of the application.

6 Discussion and Conclusions

6.1 Project Overview

As tablet computing solutions become more prevalent in everyday life, we are beginning to see them used in a variety of interesting environments in order to improve “traditional” processes. The purpose of this development and test project was to develop a “room booking” application in order to evaluate the use of tablet computers in a novel situation and also to evaluate the development process of an application on the chosen solution (BlackBerry PlayBook). Based on the initial literature review and research stage, the following research question was created which provided direction and motivation for the project.

“Can a room booking application be developed for BlackBerry PlayBook tablet which will improve existing room booking methods, and provide improved access and usability for visitors and employees?”

The main project implementation involved the development and test of a client-server solution consisting of a “room booking” application for BlackBerry PlayBook tablet and a WCF service application. The BlackBerry PlayBook application is a “front-end” application which allows users to create a booking for a selected room. The PlayBook application was created using the BlackBerry WebWorks SDK, which allows for tablet applications to be created using popular web technologies and standards such as HTML5, JavaScript and CSS. The WCF service application incorporates a “RESTful” architecture style which focuses on the transmission of data using HTTP standards such as GET, PUT, POST and DELETE.

Using a Scrum framework supported by user stories, several iterations of the PlayBook application and WCF service application were created. Each iteration of the applications aimed to expand on the previous iteration by including more functionality required by the user stories.

In order to assess the application with regards to the original motivations of the project, two hypotheses were created which would be used to evaluate the project. The hypotheses are given as follows:

1. A room booking application on a tablet computer will benefit the employees of Company X by consolidating the process of room booking within one application whilst eradicating the need to consult a third party.
2. Users will be more inclined to use the “View map of rooms” booking method as opposed to the “View list of rooms” method due to the high level of interaction and novelty.

The main methods of testing and evaluating the project were systems testing, product owner acceptance testing and HCI questionnaire. Following the completion of the development and evaluation it was found that both hypotheses were proven to be true. In addition to this the evaluation methods highlighted specific weaknesses of the developed application, which may serve as precedence for the continuation of the study.

6.2 Discussion of Results

6.2.1 Comparison with Original Envisaged Functionality

The final PlayBook application differed from the envisaged functionality as defined in the interim report. Many of the features which were originally intended to be included were changed based on changing requirements or in order to simplify processes. The following is a brief summary of the envisaged functionality for "booking a room" as it was defined in the interim report:

a. The user approaches the device and initiates the application from ready state.
b. The user selects either "View list of available rooms" or "View map of available rooms".
c. View list of available rooms populates a table on screen which indicates the available rooms and details the room specifications such as capacity and facilities. This page also allows users to search for rooms matching specific criteria and will display the refreshed list in the table.
d. View map of available rooms will render an on screen "map" of the available rooms. Available rooms will be indicated by the colour green, and unavailable rooms will be indicated by the colour red.
e. A user selects a room (in both scenarios) and proceeds to the "Checkout" screen.
f. The "Checkout" screen allows the user to specify the length of time they wish to book the room for, and input their name in order to verify recurring or future bookings.
g. A confirmation screen confirms the transaction and can be closed by the user, or is timed out upon which the user is returned to the application home screen.

The most notable changes to the original functionality is the removal of the "View list of available rooms", the inclusion of the "calendar" view and the removal of the "checkout" screen in place of the booking form triggered by the "onclick" function of the calendar cells (PlayBook Iteration 3). The final functionality of the "booking a room" procedure is described as follows:

a. The user approaches the device and initiates the application from ready state.
b. The user selects the "Book a room" button and is transferred to the "Map view" page.
c. The user can view a "map" of available rooms, with green rooms indicating the meeting rooms.
d. The user selects a "room" and proceeds to the "calendar view" page.
e. The calendar view page displays a calendar with "bookings" in order to highlight when the room is booked.
f. The user clicks a calendar cell, and is asked if they would like to book a room. If the book room button is clicked, a form will appear which allows the user to enter booking information.
g. The user enters the booking information in the form and clicks the book room function.

h. A confirmation form will appear which will indicate if the room was booked successfully, or if there was a booking error.
i. The user is returned to the start of the application.

In comparison to the original functionality intended, the final application retains much of the same features and functionality, such as “map view”, but is augmented with new features such as the “calendar” view. The main difference from the original intention is the consolidation of the booking procedure and confirmation within the same page. This was done to reduce the amount of “pages” in the application, but ultimately resulted in some pages, namely the “calendar view” page containing large amounts of code.

As mentioned in section 3.3.2 (requirements), the project was prone to changing requirements. This resulted in application features being created which were eventually abandoned. One such example of this is the original “list” view for the rooms. This was later deprecated in favour of the “map view”, which turned out to be the preferred method of room viewing for the users. This was foreseen by Company X, who explicitly stated a requirement to incorporate the “map view” page. As a result, certain user stories, which were originally expected to be implemented, were removed from the product implementation. The main user story which was abandoned was the “search” user story. Since the “map view” allowed users to view currently available rooms, and as the “rooms” were reduced to a simple room name and room id, it was decided by Company X that a search function would be unnecessary. Detailed in Appendix C, is the user stories which were implemented in the application. This appendix contains both the “old” user stories, and the “new” user stories. The latter of which contains the completed time estimations of the tasks.

6.2.2 Product Owner and Systems Testing

One of the main methods of evaluating the research question indicated above is through the completion of product owner (PO) and systems testing. Throughout the project, PO acceptance testing was completed by the stakeholders in order to validate the functionality of the application against the acceptance criteria defined in the user stories. Although the PO acceptance testing was largely informal, the same test criterion was selected for the system testing. The system testing was comprised of several test cases which exercised the application in order to test the functionality and validate the acceptance criteria defined in the user stories (Appendix D). The results of the test cases showed that the implementation successfully encapsulated the requirements of the user stories. These results coupled with the results of the HCI questionnaire proved that an application can be created for a tablet computer which essentially replaces a “traditional” method. As previously mentioned, one of the main interest of this project, particularly regarding Company X’s interest in the project is the development procedure regarding the BlackBerry PlayBook application. The successful implementation of the selected user stories and the successful outcome of the test cases provide evidence that an application can be developed with minimal issues on the platform of choice. As given in the implementation section 4.2, there were several complications regarding the application development process. However, many of these mistakes largely occurred due to inexperience and unfamiliarity with the development technologies.

6.2.3 HCI Questionnaire

In order to evaluate the hypotheses identified for the project, an HCI questionnaire was created in order to assess the effectiveness of the project and highlight any weaknesses of the application. The HCI questionnaire involved a small sample of students, who were presented with a "walkthrough" of the application (section 5.2), following which they were asked to complete a short questionnaire regarding the features and functionality of the application. As mentioned previously, there were various technical difficulties which prevented the demonstration of the actual PlayBook application. As such, the application walkthrough utilised the collection of screenshots of the application and was presented to the users via iPad. In order to test the hypotheses, the first two questions of the HCI questionnaire aimed to directly address the hypotheses by asking the following questions:

1. If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?
2. When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms?

In addition to this, the HCI questionnaire aimed to investigate the effectiveness of the implementation.

The HCI evaluation results were generally positive regarding the application. However, certain issues were highlighted regarding the presentation of certain elements of the application, most notably regarding the "calendar view". With regards to hypothesis one, the HCI evaluation results indicated that the users would find a room booking application to be beneficial. In addition to this the second question, which was created to address hypothesis two, indicated that a "map" view would be the preferred method of viewing available rooms. Although the HCI questionnaire was conducted with a very small sample of students, these results may differ should more HCI questionnaires be complete. However, the positive feedback received from the HCI questionnaires supports the overall justification for the project as it was found that the user would find a room booking application beneficial.

6.3 Project Limitations and Future Work

Although the overall outcome of the project can be regarded as successful, there were several limitations which prevented the desired level of success envisaged in the project initiation stage. The main limitations of the project were the lack of "fuller" user evaluation and the technical issues which surfaced during implementation. As identified above, the HCI evaluation contained a small sample of users. The limited feedback gathered therefore may be subject to change if a larger user sample is introduced. Future evaluations of the application therefore, would typically involve a larger user sample which would provide more reliable evaluation results. In addition to this, the evaluation lacked the physical deployment of the application to the tablet device. This was due to various technical issues regarding the authorisation and deployment of the application (practical experiment 1, section 3.2). Had the physical device been used to demonstrate the application, and had the users actually had the chance to interact with the application, this may have ultimately affected the evaluation results. As identified in the implementation section, there were several limitations regarding the technologies implemented for use in this project. The most notable example of this is regarding the implementation of the “calendar” events. The original intention was to have a method for retrieving and updating the calendar events via an Ajax “getJSON” request (PlayBook application iteration 4, section 4.2.1). This was unachievable due to a lack of technical understanding regarding FullCalendar and due to time constraints.

As identified in the initial research and literature review, there were several technologies suitable for a project of this nature. Future work related to the project area could consist of the development and test of a similar application using the other available technologies and comparing them with the selected technologies of this project. In addition to this, a comparison may be made with other tablet devices available, using the technologies developed in this project. Since the project application involved the development of a front-end application using popular web technologies, the application essentially provides a multitude of platforms for expansion. As mentioned previously in the report, one of the main issues identified was the suitability of REST vs. SOAP for the web communication aspect of the project. A comparison of REST vs. SOAP with regards to the application developed in this project could prove an interesting expansion of the work developed in this project.

6.4 Conclusion

This project was conducted in order to investigate the growing use of tablet computers as an innovative solution for improving “traditional” methods. By developing a BlackBerry PlayBook tablet application and supporting web service application, and by conducting a user centred evaluation, it was found that the application would improve the process of booking a room compared with the “traditional” method currently in use within a large corporation. The main motivation behind this project was to investigate the use of tablet computers in improving “traditional” methods. As indicated in the project introduction, there are many innovative uses of tablet computers in industry today. The successful execution and generally positive feedback regarding the application may provide useful to parties interested in exploring the capabilities of tablet computers in order to improve processes or solve unique problems.

As tablet computing is a relatively new domain, there is a somewhat lack of documentation regarding the investigation and development of tablet computer applications. This was particularly evident when conducting the preliminary project investigations and the literature review. For this reason it is expected that one of the main project benefactors would be software developers interested in the development of tablet and other touch screen devices. Since the project also involved the development of a web service which served most of the application functionality for processes such as “booking a room”, it is expected that the project outcome may also provide valuable to developers investigating web communication technologies.

7 References

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J., (2002), Agile Software Development Methods– ESPOO 2002, Vtt Publications 478, pp. 167-168.

Aijaz, F., Muzzamil, A.C. & Walke, B., (2009), Performance Comparison of a SOAP and REST Mobile Web Service, Third International Conference on Open-Source Systems and Technologies 2009, pp.1-8.

AlShahwan, F., Moessner, K. & Carrez, F., (2010), Evaluation of Distributed SOAP and RESTful MobileWeb Services, International Journal on Advances in Networks and Services, 3 (3 & 4). 447 - 461. ISSN 1942-2644

Andersen, E. (2008), "Time to get serious about the paperless office", Ubiquity, Vol. 2008, no. April, pp. 1-2.

Anonymous, (2005), “SOAP vs. REST [XML/HTTP]: The Web Services debate.”, 03/03/2012.

Anonymous, (2011), "Beyond the PC", The Economist, Vol. 401, no. 8754, pp. 3.

Arthur, C., (2011), “Adobe kills mobile Flash, giving Steve Jobs the last laugh.”, 20/01/2012.

Ashby, L. (2011), "Extension's Progress in the Paperless Revolution: Balancing Digital and Paper", Journal of Extension, Vol. 49, no. 1, pp. 1-5.

Beck, K. et al., (2001), “Manifesto for Agile Software Development”, 20/01/2012.

Berkun, S.,(1999), “The Importance of Simplicity”, 25/01/2012.

BlackBerry. (unknown), “Native SDK Documentation”, 21/01/2012.

Boehm, B. & Turner, R. (2003), Observations on Balancing Discipline and Agility – ADC '03 Proceedings of the Conference on Agile Development, IEEE.

CBC News, (2010), “Tablet computers: Will devices like the iPad replace paper?”, 20/01/2012.

Centers for Medicare and Medicade Services (CMS)., (2008), “Selecting a Development Approach”, 18/01/2012.

Chang, A., Gouldstone, J., Zigelbaum, J. & Ishii, H. (2007), Simplicity in Interaction Design - First International Conference on Tangible and Embedded Interaction, ACM, pp.135-138.

Chappell, D., (2009), “SOAP vs. REST: Compliments or Competitors?”, 04/03/2012.

Cohen, D., Lindvall, M. & Costa, P., (2003), Agile Software Development: A DACS State-of-the-Art Report, Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland, .

Cohn, M. (2004), User Stories Applied For Agile Software Development, Addison-Wesley.

Cox, J., (2011), “SOAP vs. REST For Mobile Services”, 04/03/2012.

David, M. (2011), Flash Mobile: Developing Android and iOS Applications, Elsevier.

Deitel, P.J. & Deitel H.M, (2008), Ajax, Rich Internet Applications and Web Development for Programmers, Deitel.

Dooley, J. (2011), Software Development and Professional Practice, Apress.

Elkstein, M., (2008), “Learn REST: A Tutorial”, 28/02/2012.

eTForecasts, (2011), “Worldwide PC” ,10/30/2011.

Flanagan, D. (2011), JavaScript: The Definitive Guide: Second Edition, O’Reilly Media, Inc.

Flanders, J., (2009), “More On REST”, 28/02/2012.

Freeman, E. & Robson, E., (2011), Head First HTML5 Programming, O’Reilly Media, Inc.

Garrett, J.J., (2005), “Ajax: A New Approach to Web Applications”, 18/01/2012.

Graham, F. (2011), “Tablet time: Tablet computers take on the workplace”, 10/30/2011.

Graham, J. (2011), “IPad’s increasingly crop up on restaurant menus for ordering”, 20/01/2012.

Han, J., (2006), “Multi-Touch Interaction Research”, 20/01/2012.

Holdener, A.T, (2008), Ajax: The Definitive Guide, O’Reilly Media, Inc.

Hooten, E.R. & Adams, J.A., (2010). Comparing Input Error for Mouse and Touch Input – 2010 IEEE International Conference on Systems, Man and Cybernetics (SMC), IEEE, pp.2853-2858.

Indvik, L., (2011), “Forrester: Tablet Sales Will Eclipse Laptop Sales by 2015”, 20/01/2012.

Lee, S. & Zhai, S., (2009), The Performance of Touch Screen Soft Buttons – CHI 2009, ACM, pp.309-318.

Microsoft Dev Center., (unknown), “Accessibility”, 22/01/2012.

Mikkonen, T. & Taivalsaari, A. (2011), The Web as an Application Platform: The Saga Continues – 37th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, pp.170-174.

Morgan Stanley., (2011), “Tablet Demand and Disruption- Mobile Users Come of Age”, 20/01/2012.

Murphy, K., (2011), “The Paperless Cockpit”, 10/01/2012.

Nielsen, J. & Norman, D.A., (09-10/2010), “Gestural Interfaces: A Step Backwards In Usability”, Interactions Magazine.

Nielsen, J., (1994), “Ten Usability Heuristics”, 22/01/2012.

Nielsen, J., (2011), “Kindle Fire Usability Findings”, 22/01/2012.

Noble, J., Anderson, T., Braithwaite, G., Casario, M. & Tretola, R. (2010), Flex 4 Cookbook, O’Reilly Media, Inc.

Pachner, J. (2011), "Paper's Closing Chapter", *Canadian Business*, Vol. 84, no. 8, pp. 44.

Patrizio, A. (2011), “9 Things You Haven’t Tried With Your Tablet, Yet”, 20/01/2012.

Pimmel, K., (2011), “Making It a Mobile Web App”, 16/01/2012.

Power, K., (2011), Using Silent Grouping to Size User Stories, Agile Processes in Software Engineering and Extreme Programming, pp.60-72.

RIM, (2012), “Creating a BlackBerry WebWorks configuration document”, 26/02/2012.

Rising, L. & Janoff, N.S. (2000), "The Scrum software development process for small teams", Software, IEEE, Vol. 17, no. 4, pp. 26-32.

Rozlog, M., (2010), “REST and SOAP: When Should I Use Each (or Both)?”, 07/03/2012.

Rubio, D., (2005), “REST: Simplicity in Web Services design”, 03/03/2012.

Saffer, D., (2008), Designing Gestural Interfaces, O’Reilly Media, Inc.

Smithie, A.J. (2011), “Tablet Time in the Workplace*”,* 10/30/2011.

Sommerville, I. (2011), Software Engineering: Ninth Edition, Addison-Wesley.

Spies, B., (2008), “Web Services, Part 1: SOAP vs. REST”, 29/02/2012.

Stark, J. & Jepson, B., (2012), Building Android Apps with HTML, CSS, and JavaScript, Second Edition, O’Reilly Media, Inc.

Szoke, A., (2011), A Feature Partitioning Method for Distributed Agile Release Planning, Agile Processes in Software Engineering and Extreme Programming, pp.27-42.

Ukleja, J., (2009), “WCF Data Service vs. WCF RIA Services”, 22/02/2012.

W3C., (2011), “HTML5 differences from HTML4”, 20/01/2012.

W3schools, (unknown), “AJAX Tutorial”, 25/02/2012.

Whitney, L., (2011), “HTML5-enabled phones to hit 1 billion sales in 2013”, 11/01/2012.

Wroblewski, L., (2011), “UX Benefits to Building Mobile Web Apps”, 13/01/2012.

Wroblewski, L., Villamor, C. & Willis, D., (2010), “Touch Gesture Referencing Guide”, 13/01/2012.

Young, S.T., Givens, M.T. & Gianninas, D. (2008), Adobe AIR Programming Unleashed, Sams.

8 Appendices

Appendix A – Practical Experiment Code Listings

1. JavaScript HTTP Request

```
<link rel="stylesheet" type="text/css" href="rcss.css" />
<meta charset="utf-8" />
<title>Receptionist</title>

<script type="text/javascript">
var xmlHttp = null;

function GetHTTPData()
{
   var url= document.getElementById("urlText").value;


   xmlHttp = new XMLHttpRequest();
   xmlHttp.onreadystatechange = ProcessRequest;
   xmlHttp.open( "GET", "HTTP://JSONDoc.txt" +
document.getElementById("urlText").value + ",0,0,0", true );
   xmlHttp.onreadystatechange = handlerFunction;
   xmlHttp.send( null );
}


function ProcessRequest()
{
   if ( xmlHttp.readyState == 4 && xmlHttp.status == 200 )
   {
         document.getElementById("result").innerHTML = xmlHttp.responseText;
   } else {
         document.getElementById("result").innerHTML = "Not found";
   }
}


// This is whats in the file> {"jsonobj" : [{"addressLine1":"075 Ingram
Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama,
Oprah","id":"52"},{"addressLine1":"98 Waterside Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama, Oprah","id":"78"}]};
function handlerFunction(){
         if(xmlHttp.readystate == 4){
```

```
			var json = xmlHttpRequest.responseText;
			//var JSONString = [{"addressLine1":"075 Ingram
Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama,
Oprah","id":"52"},{"addressLine1":"98 Waterside Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama, Oprah","id":"78"}]
			var myobjects = eval("(" + json + ")");
			var addressLine1 = myobjects.addressLine1;
			var employeeName = myobjects.employeeName;

			window.alert(addressLine1 + employeeName);

	}

}

</script>
</head>
<body>

<form>
<input type="text" name="href" id="urlText">
<br />
<input type="button" value="Click me!" onclick="GetHTTPData()" />
<button  onclick="location.href='HomePage.html'" class="button">Home</button>
</form>
<p id="result">awaiting result</p>
  </body>
</html>
```

2. **Retrieve JSON data from a web service and display the data in an HTML table.**

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="rcss.css" />
<meta charset="utf-8" />
<title>Receptionist</title>

<script type="text/javascript">
var xmlHttp = null;

function MockHTTPData()

{
   var myJson = [
            {"addressLine1":"075 Ingram Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama, Oprah","id":"52"},
            {"addressLine1":"98 Waterside Street","addressLine2":"Govan,
Glasgow","currentAbsence":"No","employeeName":"Obama, Oprah","id":"78"},
            {"addressLine1":"508 Havelock Street","addressLine2":"Clydebank,
Glasgow","currentAbsence":"No","employeeName":"Smith, James","id":"74"},
            {"addressLine1":"968 Ingram Street","addressLine2":"Bearsden,
Glasgow","currentAbsence":"No","employeeName":"Thompson, John","id":"39"},
            {"addressLine1":"96 Waterside Street","addressLine2":"Milngavie,
Glasgow","currentAbsence":"No","employeeName":"Timney, Stephen","id":"100"}
   ]
   ProcessJSON(myJson);
}

function GetHTTPData()
{
   if ( document.getElementById("urlText").value == ""){

            var url= "http://localhost:64420/search/2100000a,0,0";
    } else {
            var url= "http://localhost:64420/search/2100000a,0," +
document.getElementById("urlText").value;
   }
   xmlHttp = new XMLHttpRequest();
   xmlHttp.onreadystatechange = ProcessRequest;
   xmlHttp.open( "GET", url, true );
   xmlHttp.setRequestHeader("Accept", "Application/json");
  xmlHttp.send( null )
}

function ProcessRequest()
{
   if ( xmlHttp.readyState == 4 && xmlHttp.status == 200 )
   {
      if ( xmlHttp.responseText == "Not found" )
      {
         document.getElementById("result").value = "Not found";
      }
      else
      {
         var myJSONObject = eval ( "(" + xmlHttp.responseText + ")" );
         //No parsing necessary with JSON!
               ProcessJSON(myJSONObject);
      }
   }
}

function ProcessJSON(json){
document.getElementById( "table" ).innerHTML =
"<tr><td>#</td><td>id</td><td>Name</td><td>address Line</td></tr>"
               for(var i = 0; i < json.length; i++) {
                  document.getElementById( "table" ).innerHTML =
document.getElementById( "table" ).innerHTML + "<tr><td>" + (i + 1) + "</td><td>" +
json[i].id  + "</td><td>" + json[i].employeeName  + "</td><td>" + json[i].addressLine1  +
"</td></tr>"
               }
}

</script>
</head>
<body>
<form>
<input type="text" name="href" id="urlText">
<br />
<input type="button" value="Click me!" onclick="MockHTTPData()" />
<button  onclick="location.href='HomePage.html'" class="button">Home</button>
</form>
<table border="1" id="table"
</table>
  </body>
</html>
```

3. Display retrieved JSON data in jqGrid

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="Libraries/jQueryThemes/css/trontastic/jquery-ui-
1.8.16.custom.css" />
<link rel="stylesheet" type="text/css" href="Libraries/css/ui.jqgrid.css" />
<meta charset="utf-8" />

<title>JSON Data Table</title>

<script type="text/javascript" src="Libraries/jquery-1.6.4.js"></script>
<script type="text/javascript" src="Libraries/js/i18n/grid.locale-en.js"></script>
<script type="text/javascript" src="Libraries/js/jquery.jqGrid.min.js"></script>

</head>

<body>
<table id="jsonData"></table>

<script type="text/javascript">
jQuery(document).ready(function(){
        jQuery("#jsonData").jqGrid({
                //url:'server.php?q=2',
                //datatype: "json",
                datatype: "clientSide",
                data: myJson,
                colNames:['Id','Name'],
                colModel:[
                                {name:'id', index:'RoomId', width:50, sorttype:"int"},
                                {name:'employeeName', index:'RoomName', width:200}

                        ],
                        pager: jQuery('#pager'),
        rowNum:10,
        viewrecords: true,
        caption: 'Grid demo',
        localReader : {
                                root: "rows",
                                page: "page",
                                total: "total",
                                records: "records",
                                        repeatitems: false,
                                        cell: "cell",
                                        id: "id",
                                        userdata: "userdata",
                                        subgrid: {root:"rows", repeatitems: true, cell:"cell"}
                              }
          });
});

var myJson =
[{"RoomId":1,"RoomName":"Dundas"},{"RoomId":2,"RoomName":"Dasds"},{"RoomId":2,
"RoomName":"Dasdfas"}

   ];
   for(var i=0;i<myJson.length;i++)
         jQuery("#jsonData").jqGrid('addRowData',i+1,myJson[i]);

</script>
  </body>
</html>
```

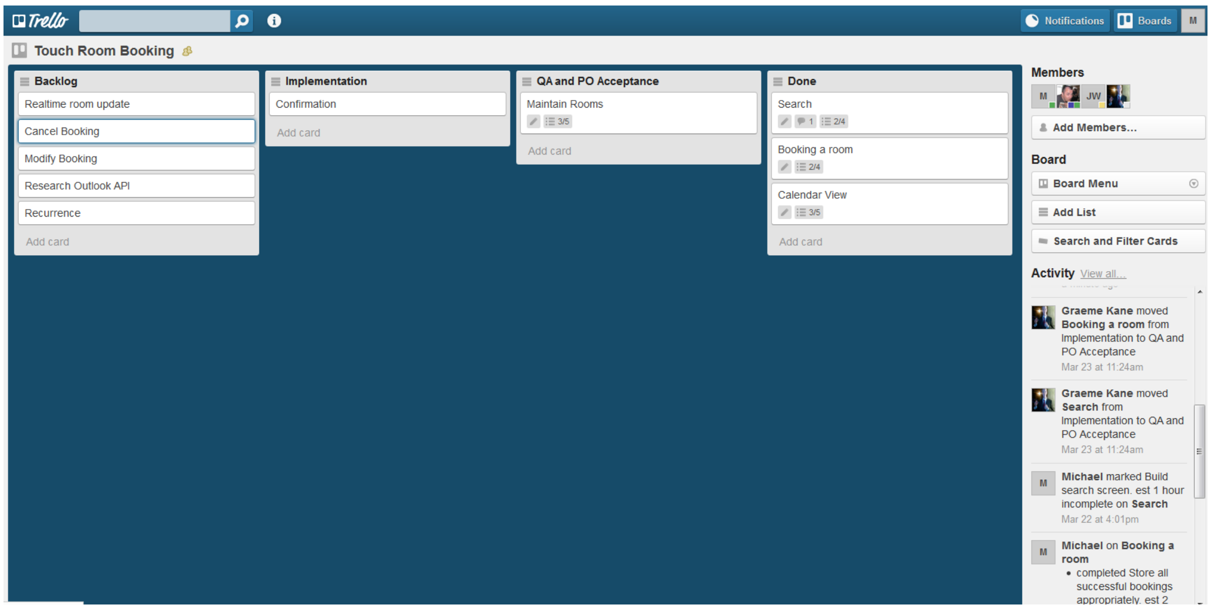
4. **Create a sample user interface of room map, with interaction.**

```
<!DOCTYPE html>
<html lang="en"><head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>Canvas Interface</title>


 <style type="text/css">
  #canvas_container
  {
          width: 1024px;
          border: 3px solid black;
          background-color: #D8D8D8;
          }


  </style>
</head>
<body>
<div id="canvas_container">

</div>

<script type="text/javascript" src="Lib/raphael-min.js"></script>
<script type="text/javascript">
window.onload = function(){
          var paper =new Raphael(document.getElementById('canvas_container'), 1024, 600);
          //top row left to right

                    var room1 = paper.rect(0,0,250,150);
                              room1.attr({fill: '#64FE2E', "stroke-width": 3});
                              room1.node.onclick = function(){
                                        room1.animate({fill: 'red'}, 5000, "elastic", function() {
                              this.attr({fill: '#64FE2E', "stroke-width": 3});
                                        });
                              }

                    var room2 = paper.rect(250,0,250,100);
                              room2.attr({fill: '#64FE2E', "stroke-width": 3});
                              room2.node.onclick = function(){
                                        room2.animate({fill: 'red'}, 5000, "elastic", function() {
                              this.attr({fill: '#64FE2E', "stroke-width": 3});
```

```
                });
            }
        var room3 = paper.rect(500,0,150,100);
            room3.attr({fill: '#64FE2E', "stroke-width": 3});
            room3.node.onclick = function(){
                room3.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                });
            }

        var room4 = paper.rect(724,0,300,150);
            room4.attr({fill: '#64FE2E', "stroke-width": 3});
            room4.node.onclick = function(){
                room4.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                });
            }

    //middle row l-r
        var room5 = paper.rect(0,200,200,150);
            room5.attr({fill: '#64FE2E', "stroke-width": 3});
            room5.node.onclick = function(){
                room5.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                });
            }

        var room6 = paper.rect(400,200,200,200);
            room6.attr({fill: '#64FE2E', "stroke-width": 3});
            room6.node.onclick = function(){
                room6.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                });
            }

        var room7 = paper.rect(724,200,300,200);
            room7.attr({fill: '#64FE2E', "stroke-width": 3});
            room7.node.onclick = function(){
                room7.animate({fill: 'red'}, 5000, "elastic", function() {
            this.attr({fill: '#64FE2E', "stroke-width": 3});
                });
            }
        //last row
                var room8 = paper.rect(0,400,250,150);
                        room8.attr({fill: '#64FE2E', "stroke-width": 3});
                        room8.node.onclick = function(){
                                room8.animate({fill: 'red'}, 5000, "elastic", function() {
                        this.attr({fill: '#64FE2E', "stroke-width": 3});
                                });
                        }

                var room9 = paper.rect(350,500,200,100);
                        room9.attr({fill: '#64FE2E', "stroke-width": 3});
                        room9.node.onclick = function(){
                                room9.animate({fill: 'red'}, 5000, "elastic", function() {
                        this.attr({fill: '#64FE2E', "stroke-width": 3});
                                });
                        }

                var room10 = paper.rect(754,500,300,100);
                        room10.attr({fill: '#64FE2E', "stroke-width": 3});
                        room10.node.onclick = function(){
                                room10.animate({fill: 'red'}, 5000, "elastic", function() {
                        this.attr({fill: '#64FE2E', "stroke-width": 3});
                                });
                        }

                var room11 = paper.rect(724,0,300,150);
                        room11.attr({fill: '#64FE2E', "stroke-width": 3});
                        room11.node.onclick = function(){
                                room11.animate({fill: 'red'}, 5000, "elastic", function() {
                        this.attr({fill: '#64FE2E', "stroke-width": 3});
                                });
                        }
}
</script>
</body>
</html>
```

Appendix B – Trello Task Board Screenshots

1. Trello task board



Touch Room Booking

Backlog
- Realtime room update
- Cancel Booking
- Modify Booking
- Research Outlook API
- Recurrence
- Add card

Implementation
- Confirmation
- Add card

QA and PO Acceptance
- Maintain Rooms 3/5
- Add card

Done
- Search 1 2/4
- Booking a room 2/4
- Calendar View 3/5
- Add card

Members

Add Members...

Board

Board Menu

Add List

Search and Filter Cards

Activity View all...

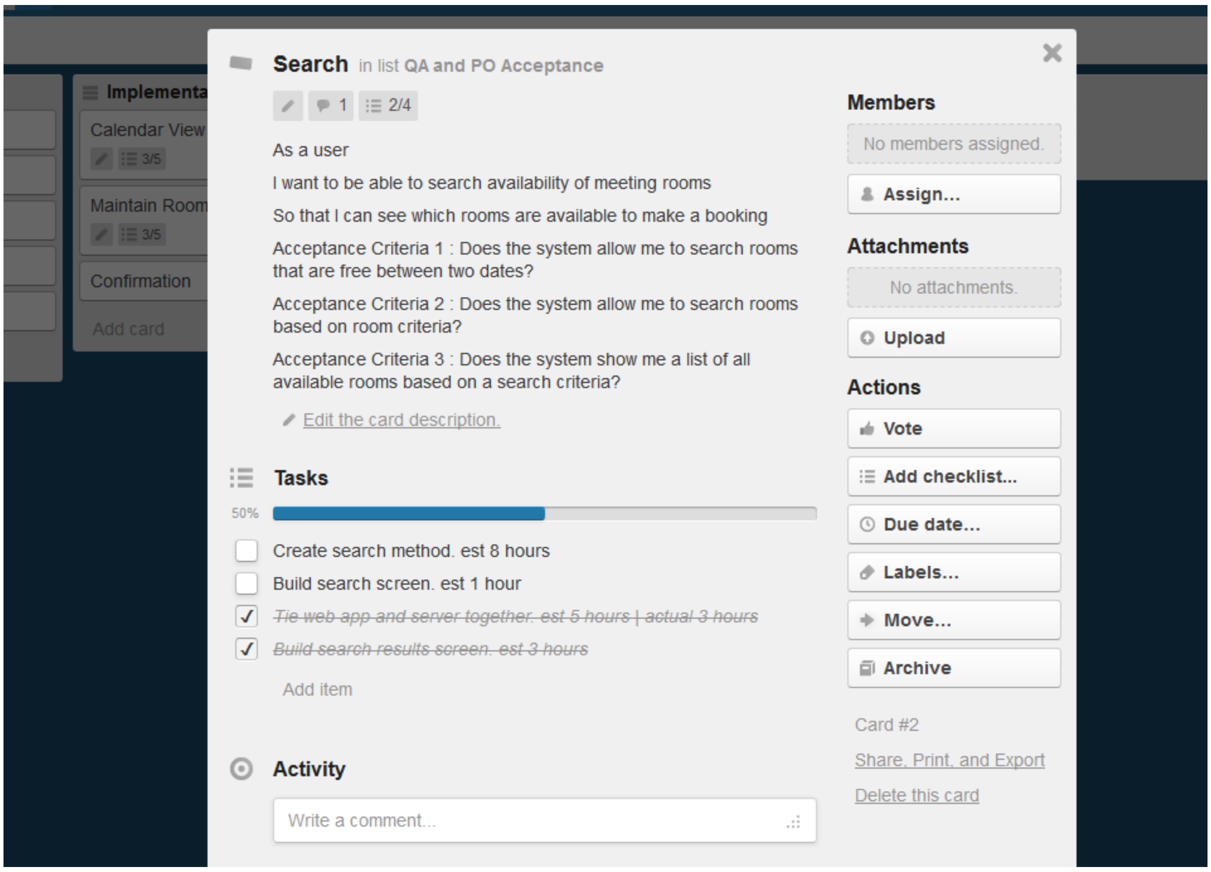
Graeme Kane moved **Booking a room** from Implementation to QA and PO Acceptance
Mar 23 at 11:24am

Graeme Kane moved **Search** from Implementation to QA and PO Acceptance
Mar 23 at 11:24am

Michael marked Build search screen. est 1 hour incomplete on **Search**
Mar 22 at 4:01pm

Michael on **Booking a room**
- completed Store all successful bookings appropriately. est 2

2. Example user story



Search in list **QA and PO Acceptance**

1 2/4

As a user

I want to be able to search availability of meeting rooms

So that I can see which rooms are available to make a booking

Acceptance Criteria 1 : Does the system allow me to search rooms that are free between two dates?

Acceptance Criteria 2 : Does the system allow me to search rooms based on room criteria?

Acceptance Criteria 3 : Does the system show me a list of all available rooms based on a search criteria?

Edit the card description.

Tasks

50%

- [ ] Create search method. est 8 hours
- [ ] Build search screen. est 1 hour
- [x] ~~Tie web app and server together. est 5 hours | actual 3 hours~~
- [x] ~~Build search results screen. est 3 hours~~

Add item

Activity

Write a comment...

Members

No members assigned.

Assign...

Attachments

No attachments.

Upload

Actions

Vote

Add checklist...

Due date...

Labels...

Move...

Archive

Card #2

Share, Print, and Export

Delete this card

Appendix C – User Stories

1. Original User Stories

Search

As a user
I want to be able to search availability of meeting rooms
So that I can see which rooms are available to make a booking

Acceptance Criteria 1: Does the system allow me to search rooms free between two dates?
Acceptance Criteria 2: Does the system allow me to search rooms based on room criteria?
Acceptance Criteria 3: Does the system dhow me a list of all available rooms based on search criteria?

Tasks
- Create search method (estimate 8 hours)
- Build search screen (estimate 1 hour)
- Tie web app and server together (estimate 5 hours)
- Build search results screen (estimate 3 hours)

Booking a Room

As a user
I want to be able to book out a room between two specified times
So that the room is marked as booked during the times given and cannot be booked by another user.

Acceptance Criteria 1: Does the system store a record of all bookings made?
Acceptance Criteria 2: Does the system confirm that the room has been successfully booked?
Acceptance Criteria 3: Does the system alert me if a booking already exists on the time I want to book and denies me the ability to book?

Tasks
- Create a method which accepts two times and a room id then returns whether or not this room can be booked. (estimate 10 hours)
- Create a screen which is shown when a room is clicked that accepts two times and confirms to the user if the selected room can be booked. (estimate 2 hours)
- Store all successful bookings appropriately. (estimate 2 hours)
- Create a means to show confirmation of a room booking. (estimate 1 hour)

Map and Calendar View
As a user I want to see a floor plan showing all rooms available for booking and a calendar for each room So that I am able to click any room to see a calendar of bookings for that room.
Acceptance Criteria 1: Can I view a graphic of all rooms and click specific rooms? Acceptance Criteria 2: Can I view a calendar view of any room with all bookings attached to that room?
Tasks - Create a floor plan of the rooms available to book (estimate 4 hours) - Allow the user to select any room (estimate 4 hours) - Create a calendar view (estimate 10 hours) - Create an item for each booking and draw appropriately on the calendar (estimate 10 hours) - Show the calendar whenever a room is selected with all bookings for selected rooms (estimate 4 hours)

Maintain Rooms
As a developer I want a list of rooms and their availability to be accessible So that I can retrieve and update this information.
Acceptance Criteria 1: Does the system allow me to retrieve the status of all rooms? Acceptance Criteria 2: Does the system allow me to update the status of a room?
Tasks - Create WCF project and learn the basics (estimate 6 hours) - Build rooms model (estimate 10 hours) - Build test web app (estimate 3 hours) - Create retrieve status method (estimate 6 hours) - Create the update room method (estimate 8 hours)

Confirmation
As a user I want to be notified when I make a booking So that I know that the booking was successful or denied.
Acceptance Criteria 1: Does the system show me if my booking was successful? Acceptance Criteria 2: Does the system show me if the booking was denied?
Tasks - Create a function which confirms or denies the room booking. (estimate 3 hours)

2. Appended User Stories

Booking a Room
As a user I want to be able to book out a room between two specified times So that the room is marked as booked during the times given and cannot be booked by another user.
Acceptance Criteria 1: Does the system store a record of all bookings made? Acceptance Criteria 2: Does the system confirm that the room has been successfully booked? Acceptance Criteria 3: Does the system alert me if a booking already exists on the time I want to book and denies me the ability to book?
Tasks - Create a method which accepts two times and a room id then returns whether or not this room can be booked. (estimate 10 hours, actual 15 hours) - Create a screen which is shown when a room is clicked that accepts two times and confirms to the user if the selected room can be booked. (estimate 2 hours, actual 8 hours) - Store all successful bookings appropriately. (estimate 2 hours, actual 15 hours) - Create a means to show confirmation of a room booking. (estimate 1 hour, actual 1 hour)

Map and Calendar View
As a user I want to see a floor plan showing all rooms available for booking and a calendar for each room So that I am able to click any room to see a calendar of bookings for that room.
Acceptance Criteria 1: Can I view a graphic of all rooms and click specific rooms? Acceptance Criteria 2: Can I view a calendar view of any room with all bookings attached to that room?
Tasks - Create a floor plan of the rooms available to book (estimate 4 hours, actual 2 hours) - Allow the user to select any room (estimate 4 hours, actual 5 hours) - Create a calendar view (estimate 10 hours, actual 6 hours) - Create an item for each booking and draw appropriately on the calendar (estimate 10 hours, actual 10) - Show the calendar whenever a room is selected with all bookings for selected rooms (estimate 4 hours, actual 6 hours)

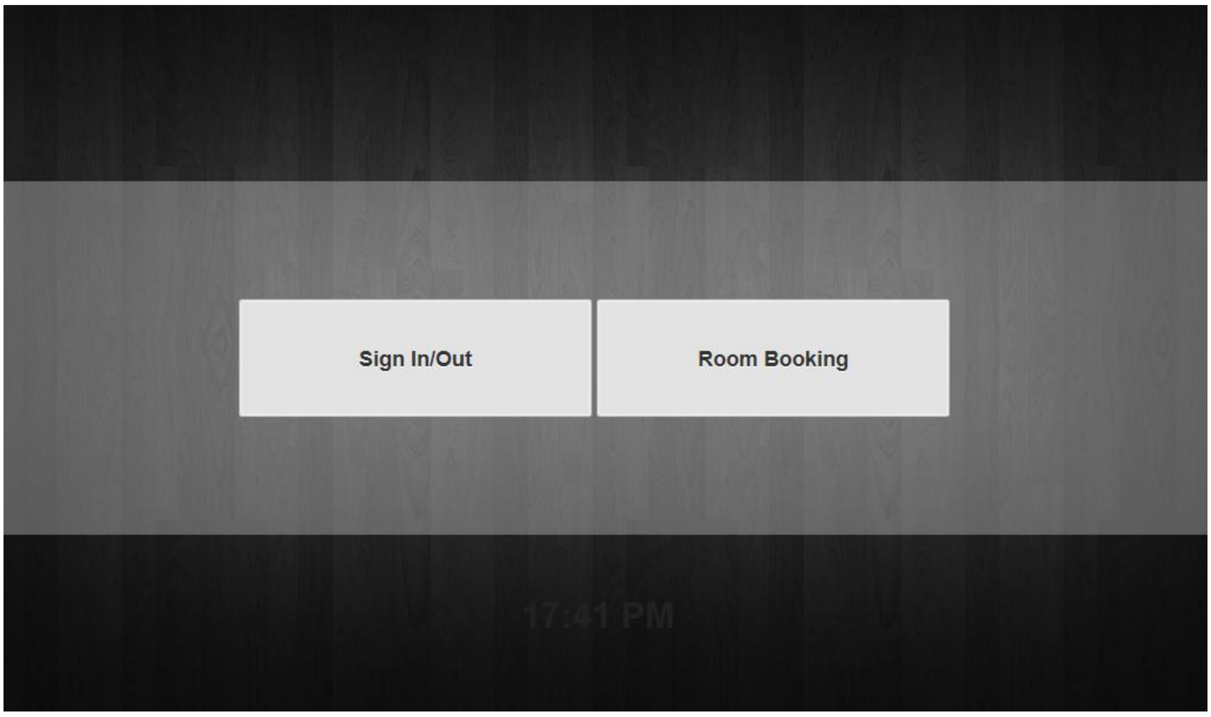
Maintain Rooms
As a developer I want a list of rooms and their availability to be accessible So that I can retrieve and update this information.
Acceptance Criteria 1: Does the system allow me to retrieve the status of all rooms? Acceptance Criteria 2: Does the system allow me to update the status of a room?
Tasks - Create WCF project and learn the basics (estimate 6 hours, actual 20 hours) - Build rooms model (estimate 10 hours, actual 14 hours) - Build test web app (estimate 3 hours, actual 6 hours) - Tie web app and WCF service together (estimate 5 hours, actual 6 hours) - Create retrieve status method (estimate 6 hours) (removed) - Create the update room method (estimate 8 hours) (removed)

Confirmation
As a user I want to be notified when I make a booking So that I know that the booking was successful or denied.
Acceptance Criteria 1: Does the system show me if my booking was successful? Acceptance Criteria 2: Does the system show me if the booking was denied?
Tasks - Create a function which confirms or denies the room booking. (estimate 3 hours, actual 3 hours)

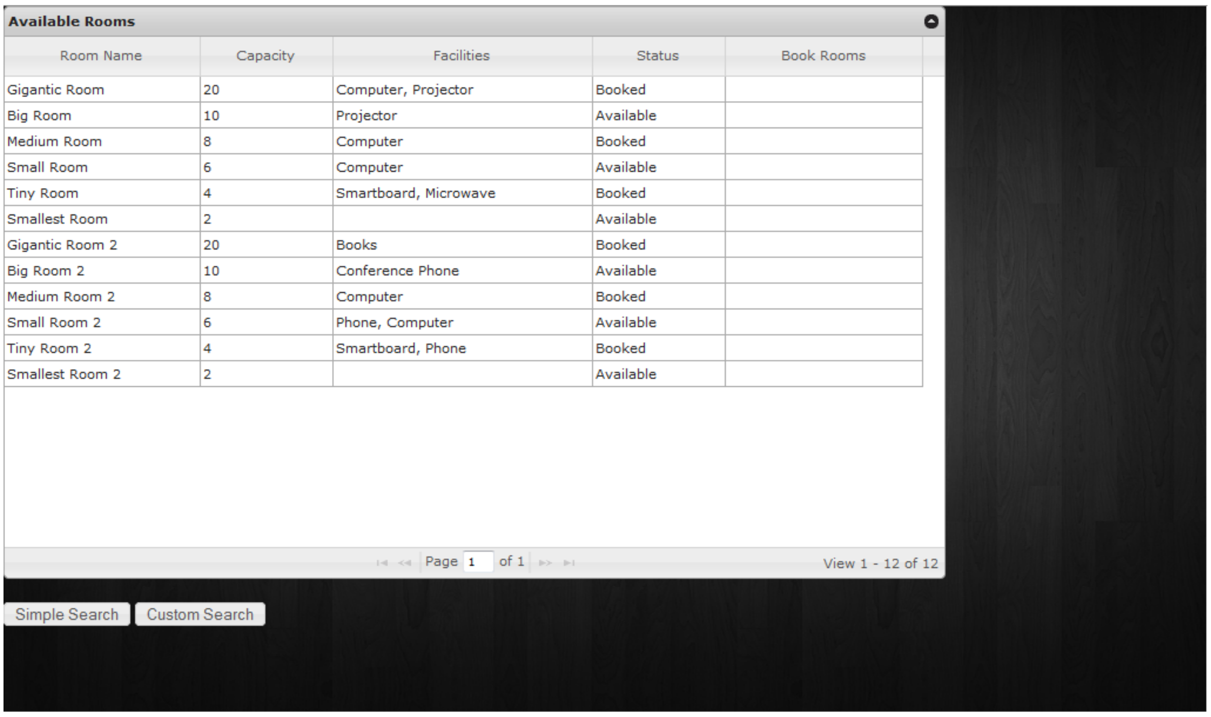
Appendix D – Application Screenshots

1. "Old" Application Screenshots

a. Main page



b. Grid view

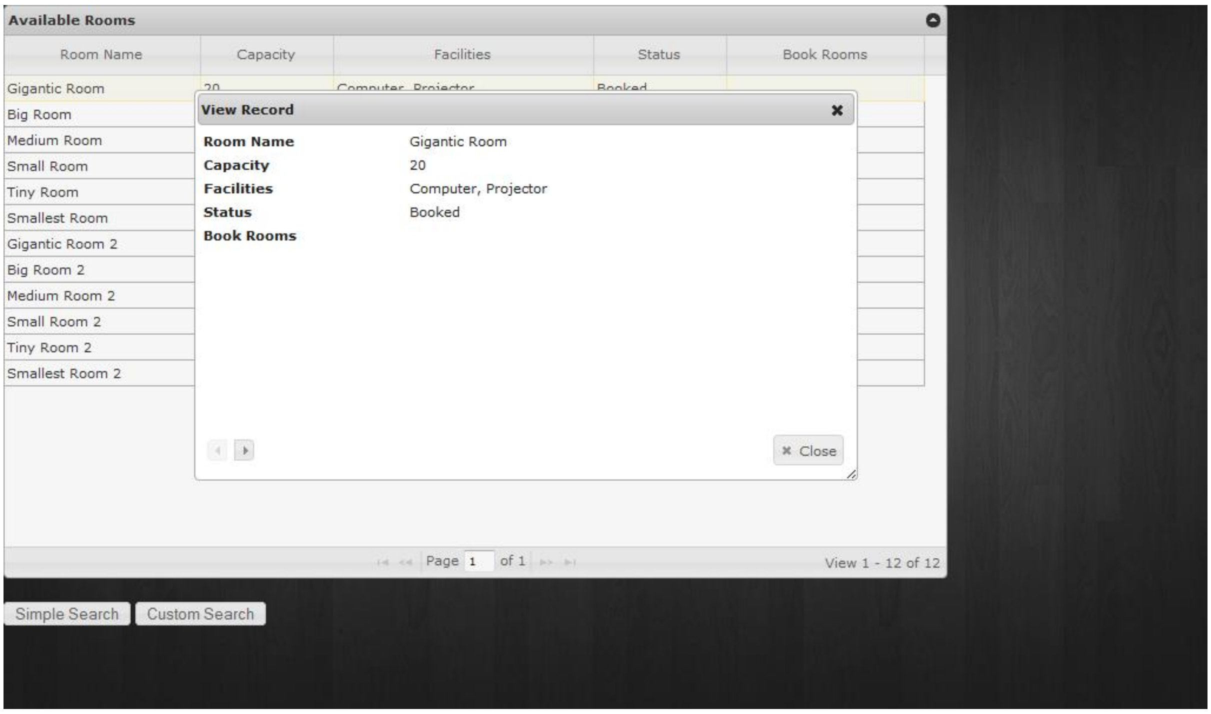


Available Rooms

Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room	10	Projector	Available	
Medium Room	8	Computer	Booked	
Small Room	6	Computer	Available	
Tiny Room	4	Smartboard, Microwave	Booked	
Smallest Room	2		Available	
Gigantic Room 2	20	Books	Booked	
Big Room 2	10	Conference Phone	Available	
Medium Room 2	8	Computer	Booked	
Small Room 2	6	Phone, Computer	Available	
Tiny Room 2	4	Smartboard, Phone	Booked	
Smallest Room 2	2		Available	

Page 1 of 1 View 1 - 12 of 12

c. Room information



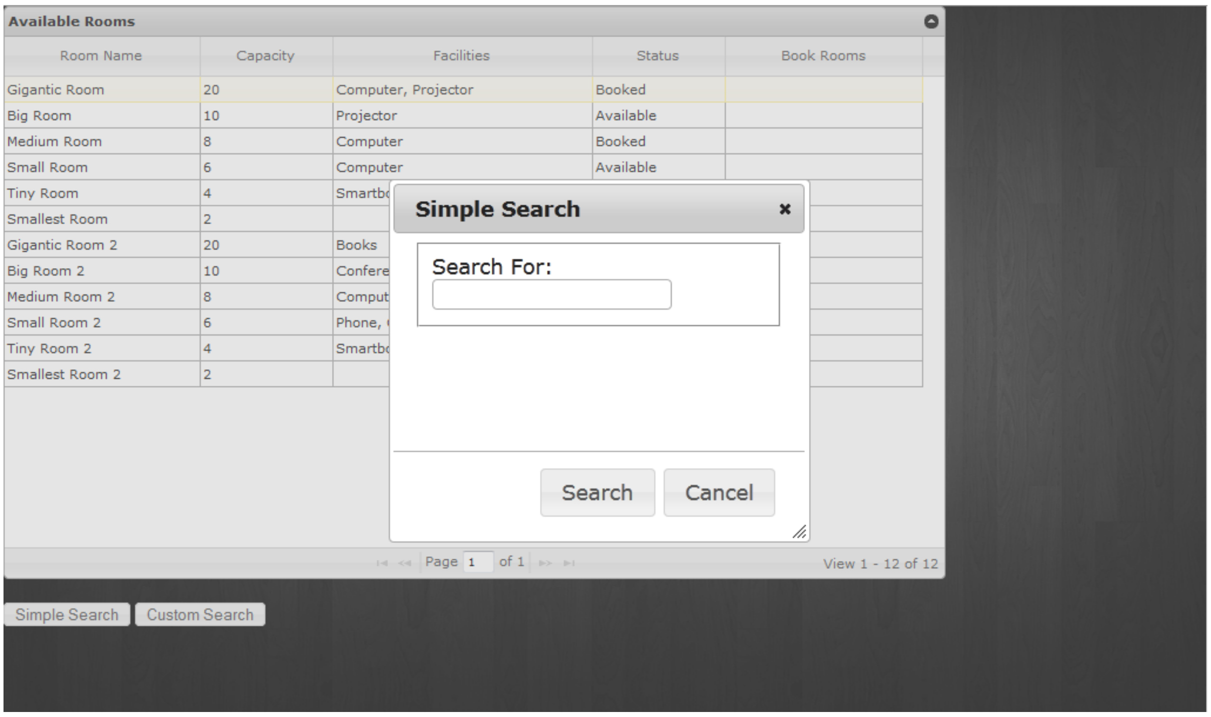
Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room				
Medium Room				
Small Room				
Tiny Room				
Smallest Room				
Gigantic Room 2				
Big Room 2				
Medium Room 2				
Small Room 2				
Tiny Room 2				
Smallest Room 2				

View Record

Room Name	Gigantic Room
Capacity	20
Facilities	Computer, Projector
Status	Booked
Book Rooms	

View 1 - 12 of 12

d. Simple search

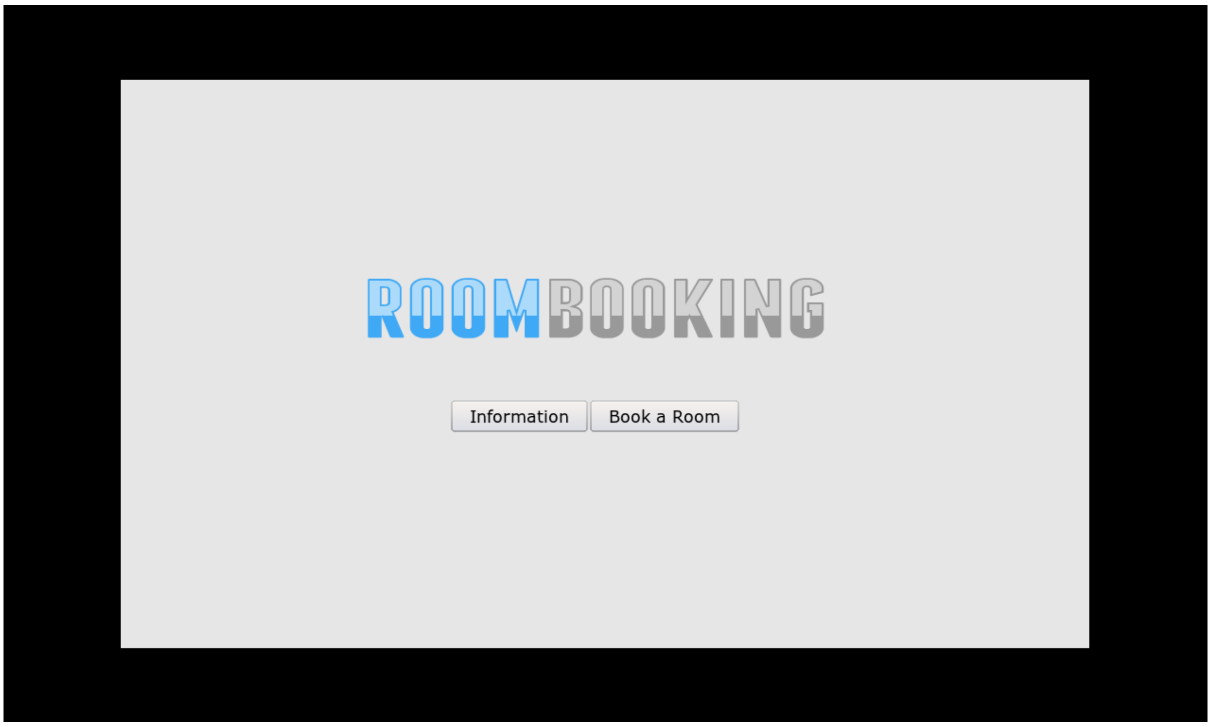


Room Name	Capacity	Facilities	Status	Book Rooms
Gigantic Room	20	Computer, Projector	Booked	
Big Room	10	Projector	Available	
Medium Room	8	Computer	Booked	
Small Room	6	Computer	Available	
Tiny Room	4	Smartb		
Smallest Room	2			
Gigantic Room 2	20	Books		
Big Room 2	10	Confere		
Medium Room 2	8	Comput		
Small Room 2	6	Phone,		
Tiny Room 2	4	Smartb		
Smallest Room 2	2			

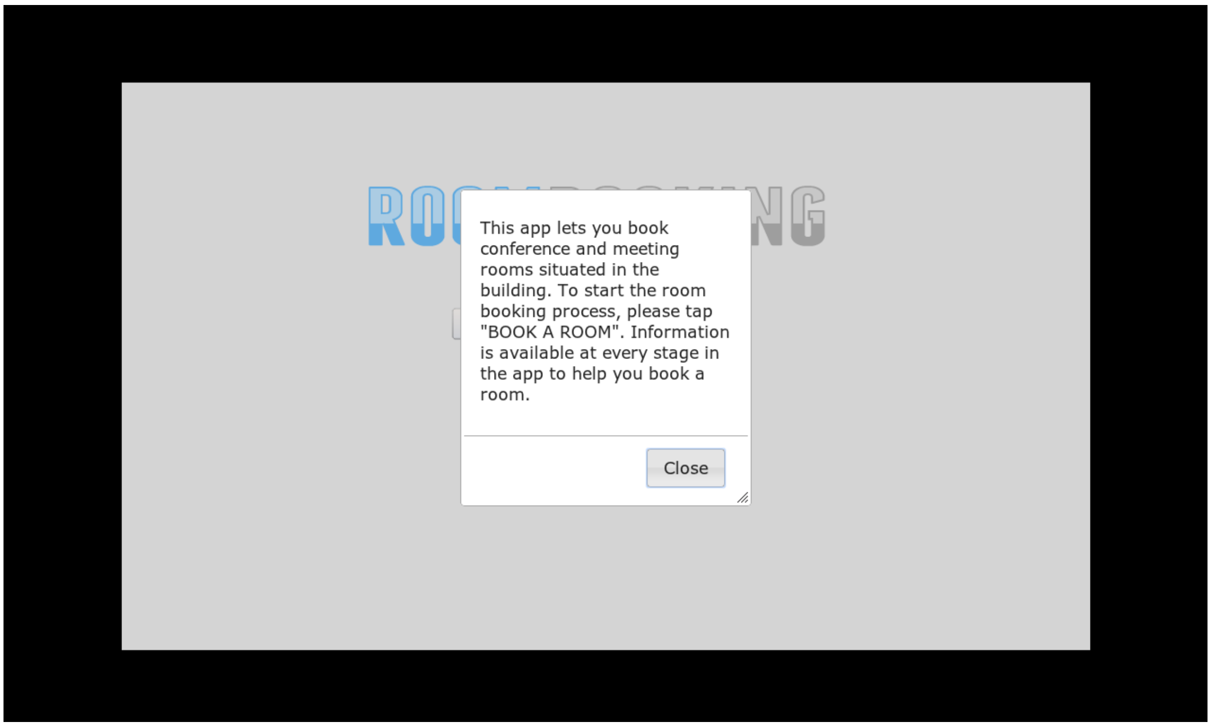
View 1 - 12 of 12

2. “Final” Application Screenshots

a. Main page

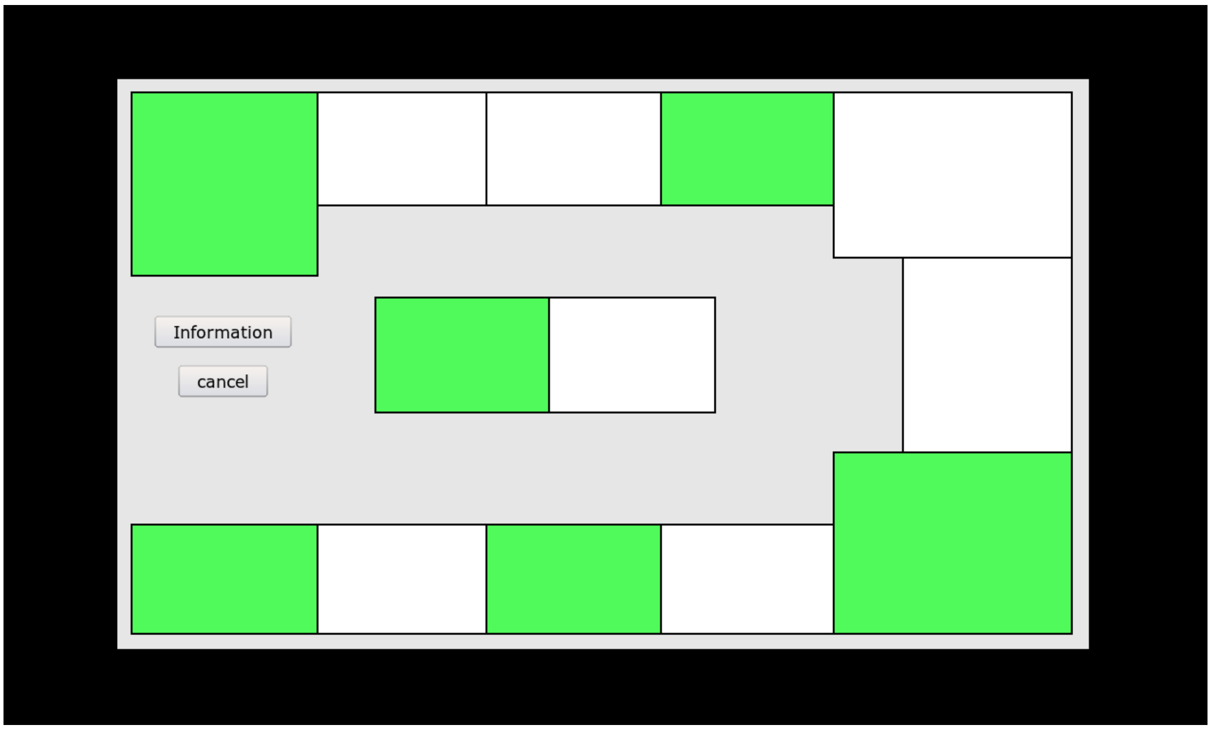


b. Main page information

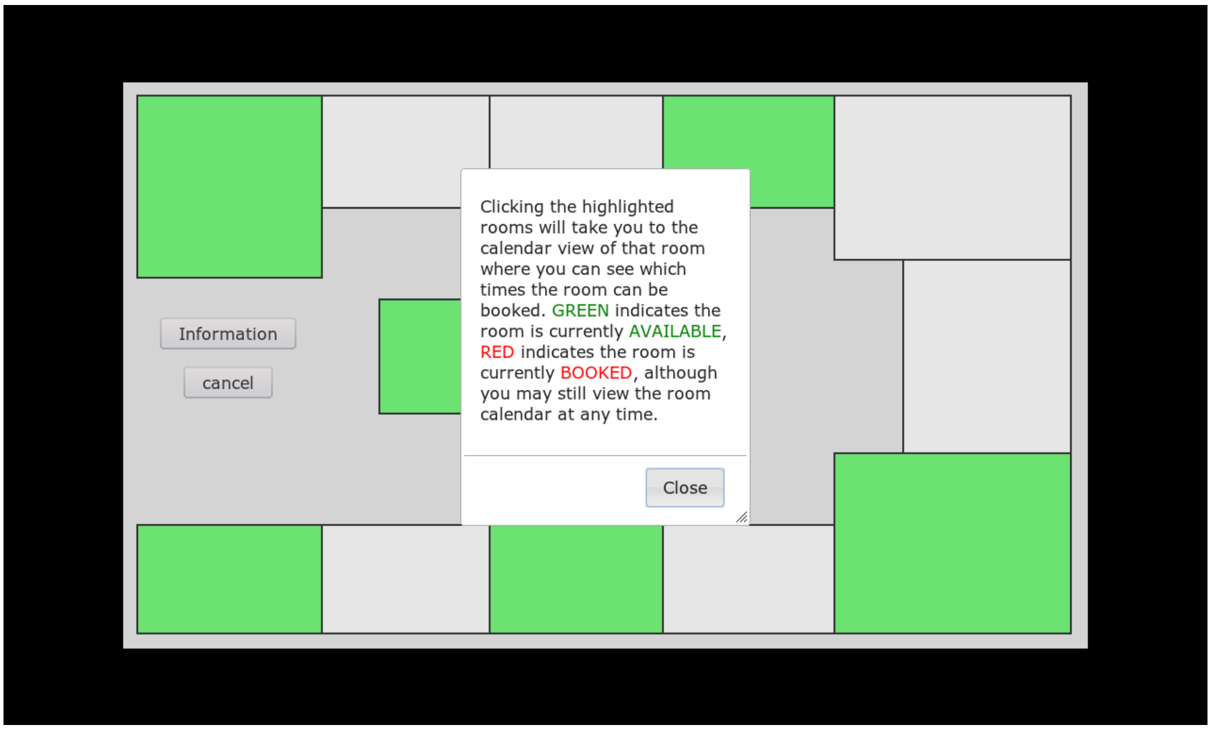


This app lets you book conference and meeting rooms situated in the building. To start the room booking process, please tap "BOOK A ROOM". Information is available at every stage in the app to help you book a room.

c. Map view

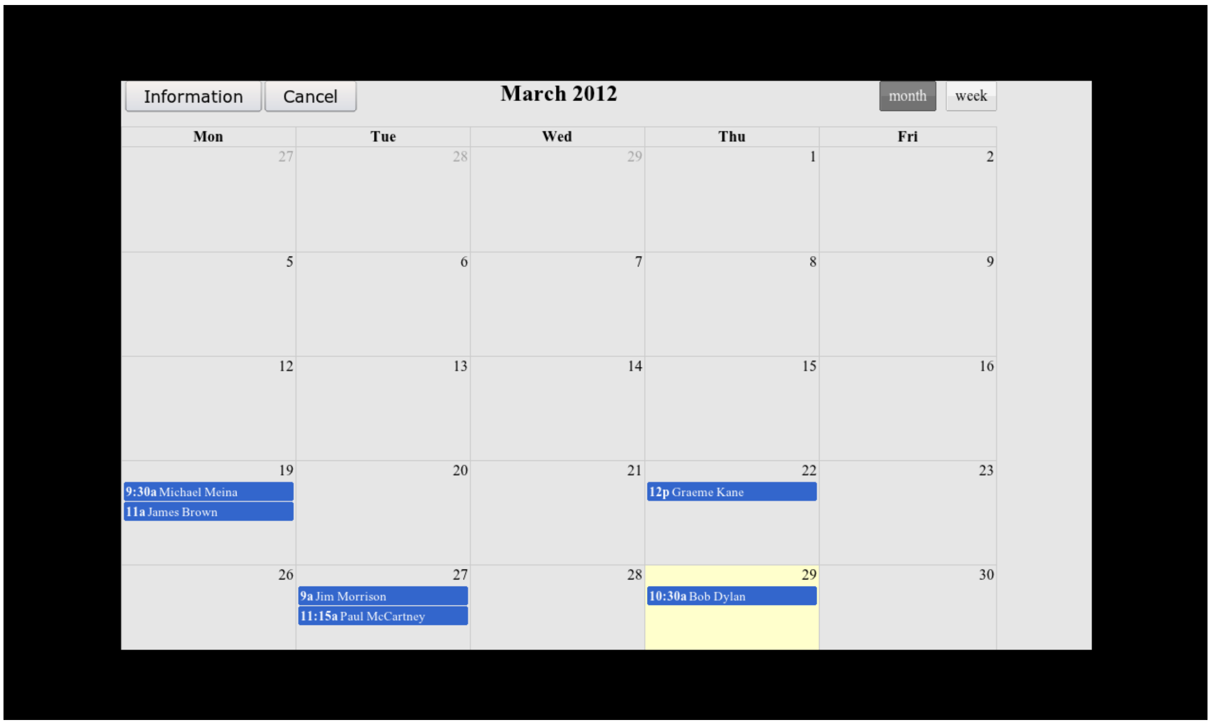


d. Map view information

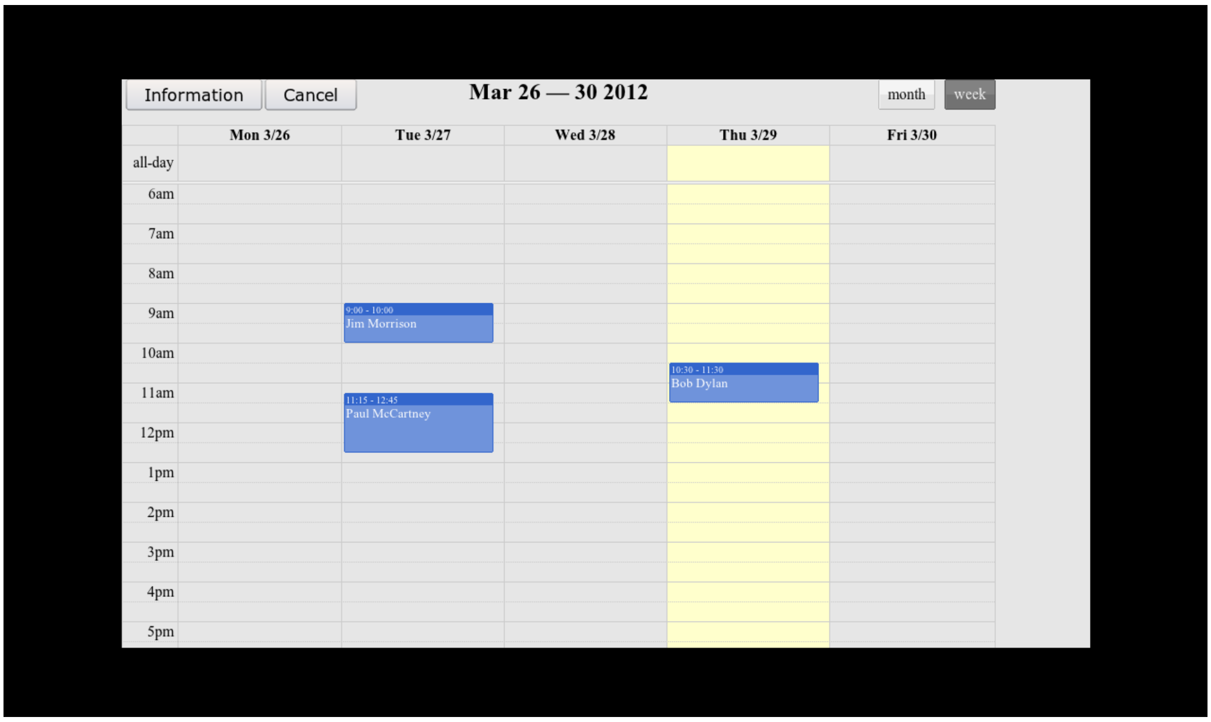


Clicking the highlighted rooms will take you to the calendar view of that room where you can see which times the room can be booked. GREEN indicates the room is currently AVAILABLE, RED indicates the room is currently BOOKED, although you may still view the room calendar at any time.

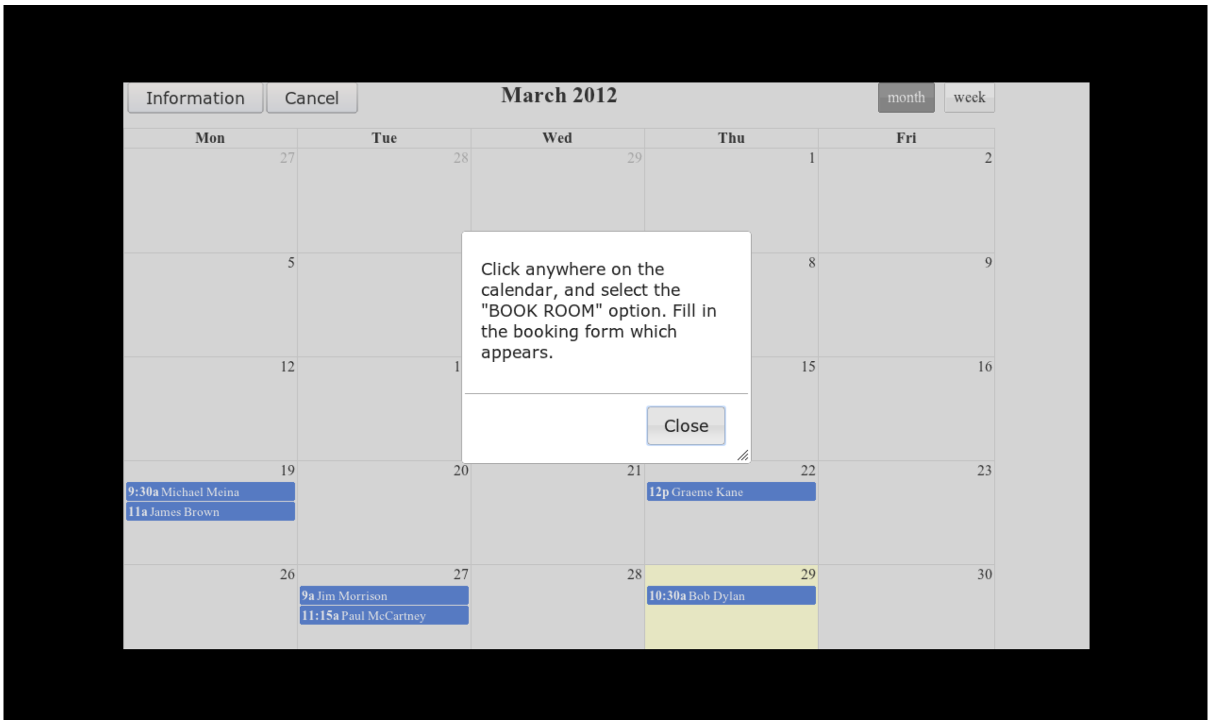
e. Calendar view main



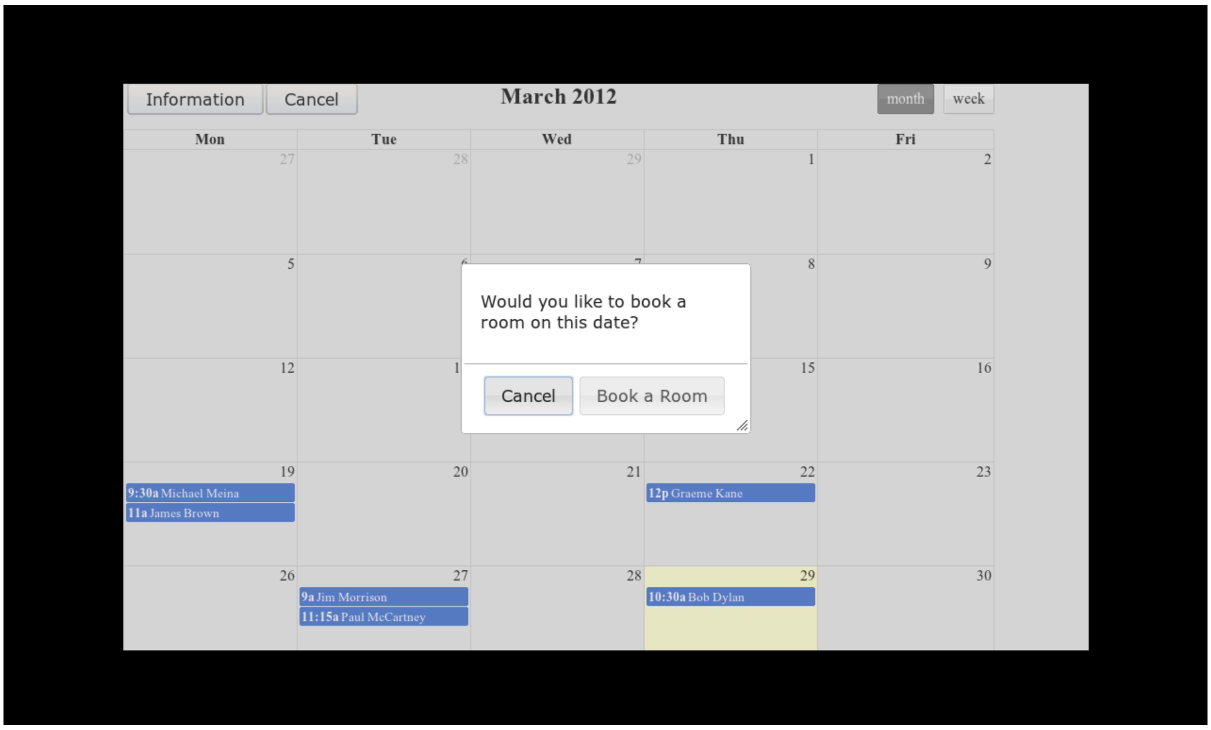
f. Calendar view day



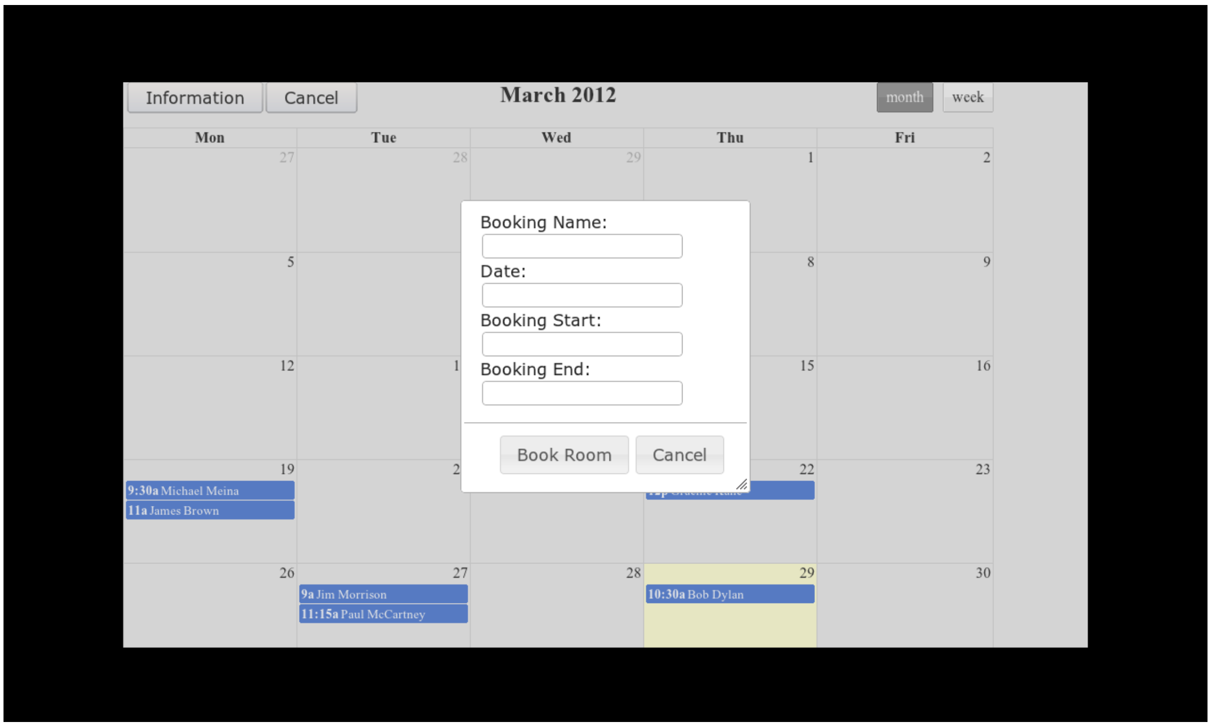
g. Calendar view information


Click anywhere on the calendar, and select the "BOOK ROOM" option. Fill in the booking form which appears.

h. Booking request confirmation


Would you like to book a room on this date?

i. Booking form



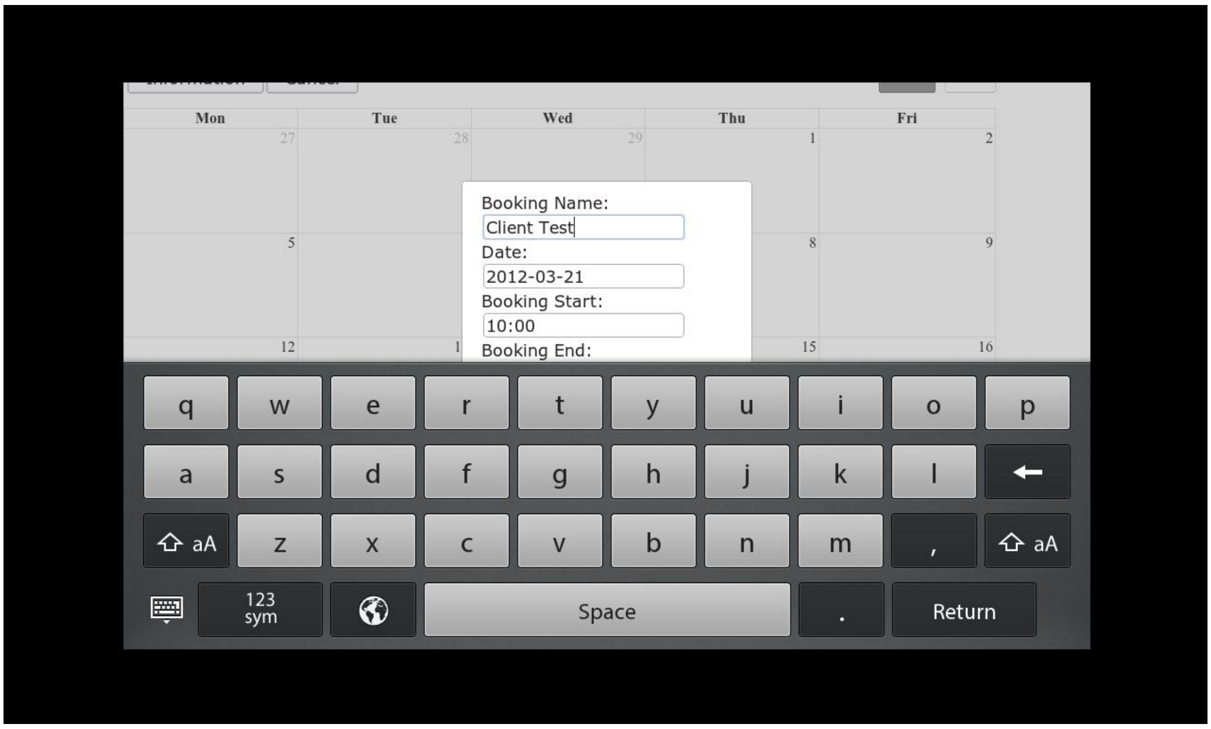
Booking Name:

Date:

Booking Start:

Booking End:

j. Keyboard input



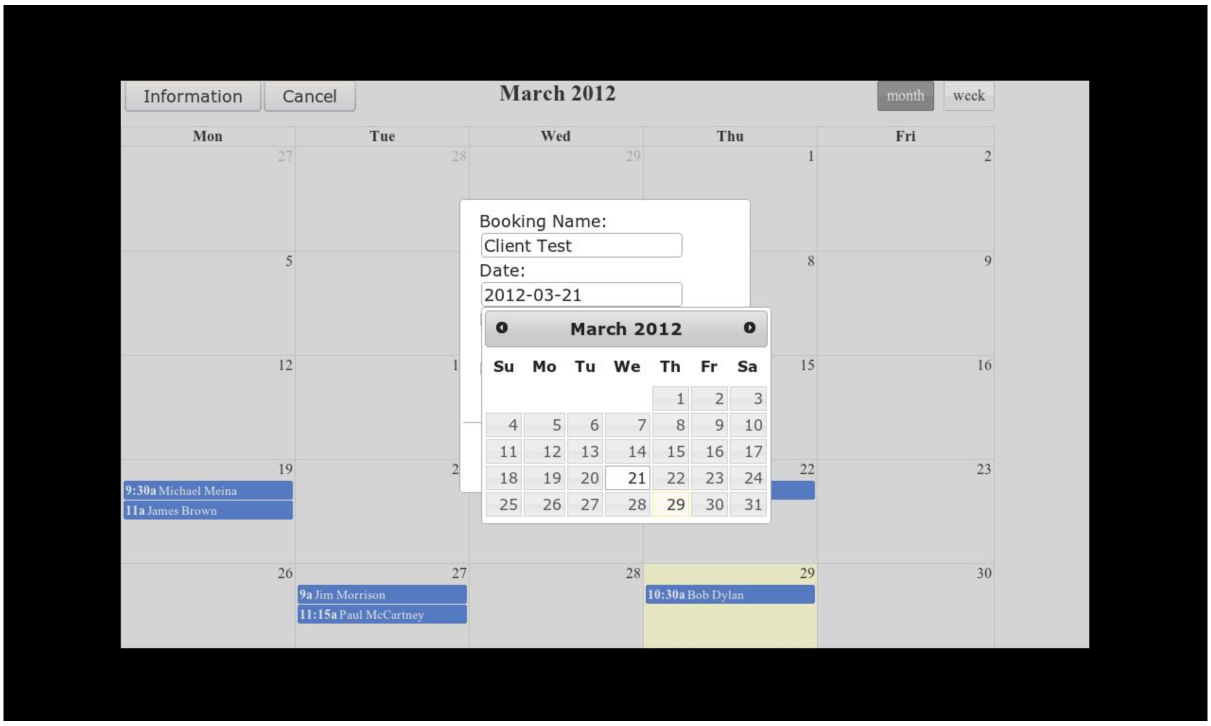
Booking Name: Client Test

Date: 2012-03-21

Booking Start: 10:00

Booking End:

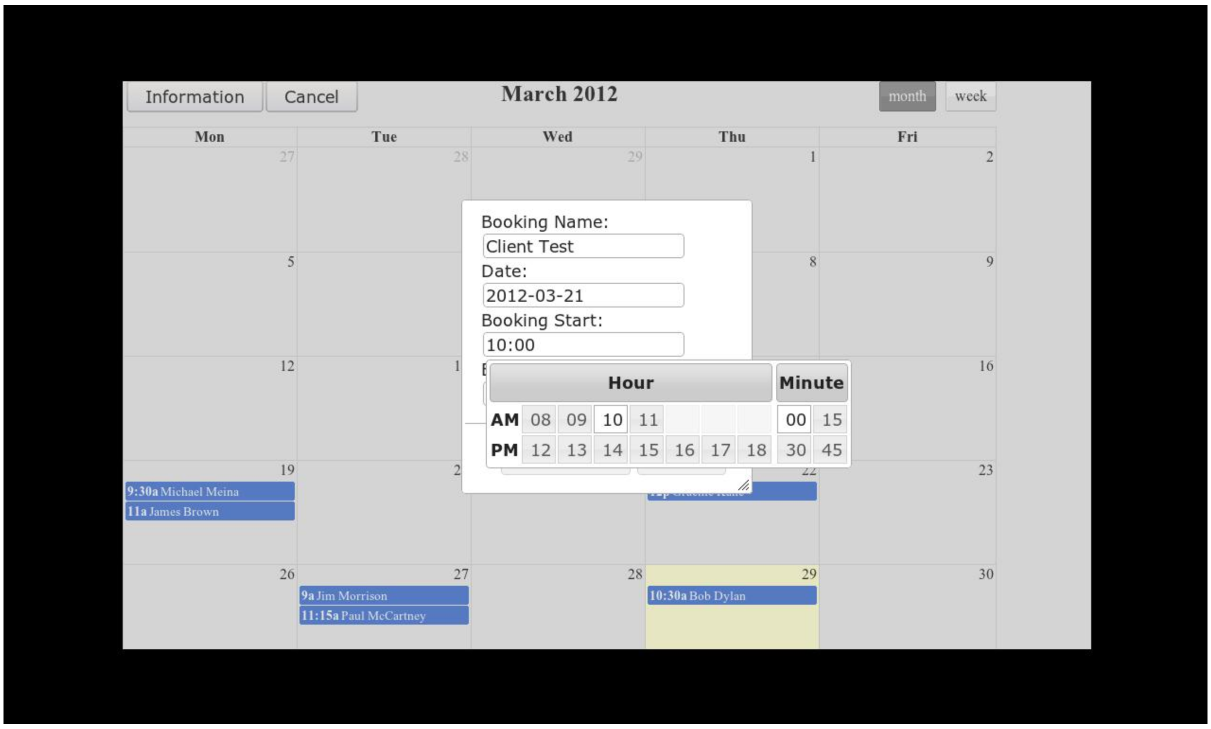
k. Datepicker



Booking Name:
Client Test
Date:
2012-03-21

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

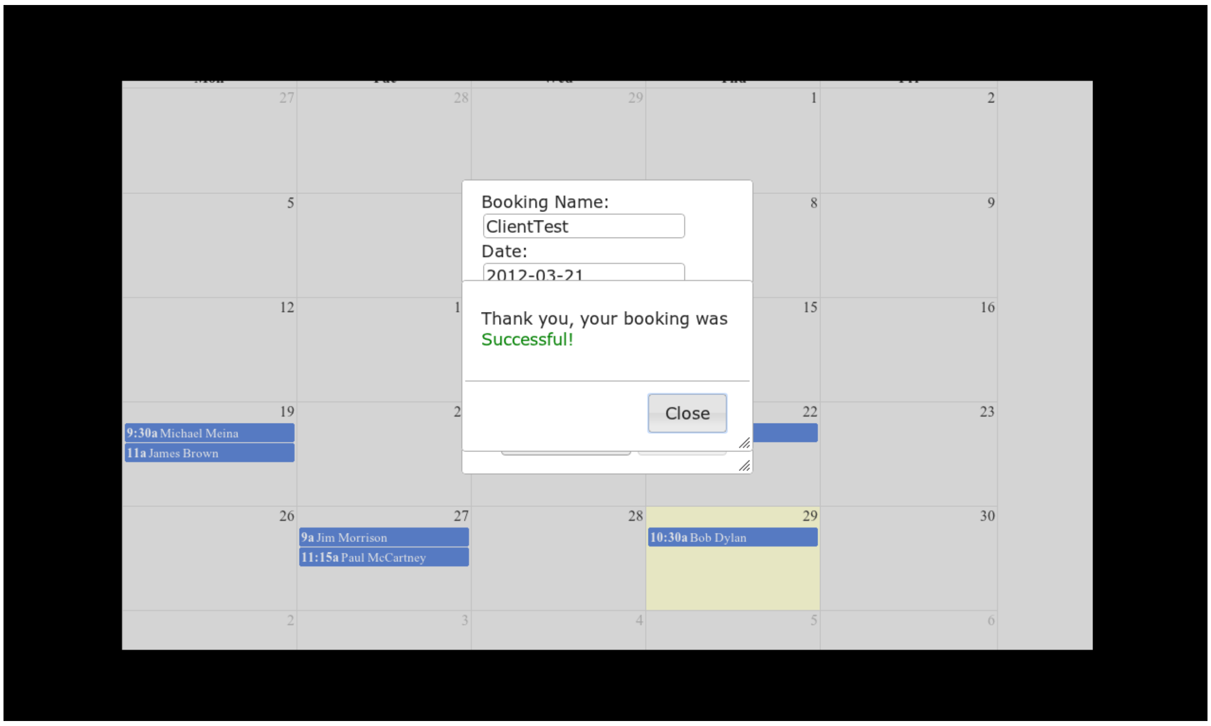
l. Timepicker



Booking Name:
Client Test
Date:
2012-03-21
Booking Start:
10:00

Hour								Minute	
AM	08	09	10	11				00	15
PM	12	13	14	15	16	17	18	30	45

m. Booking successful



Booking Name:

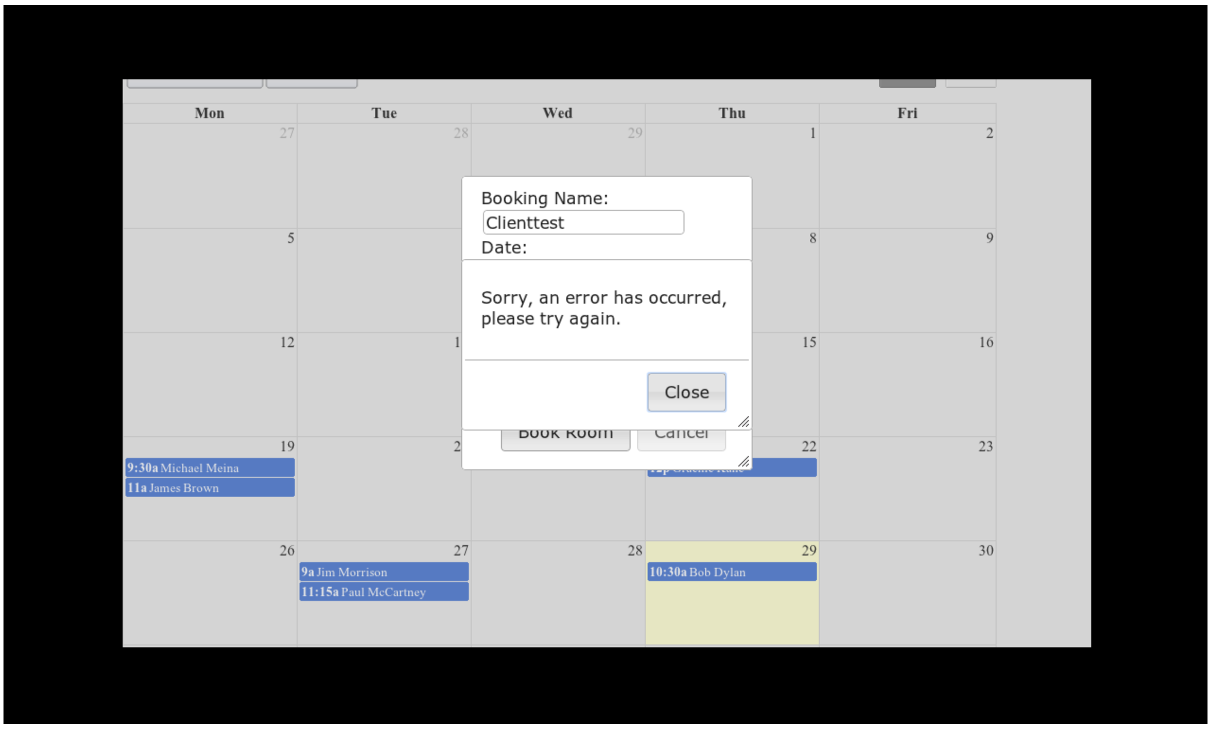
ClientTest

Date:

2012-03-21

Thank you, your booking was Successful!

n. Booking Failure



Booking Name:

Clienttest

Date:

Sorry, an error has occurred, please try again.

Appendix E – BlackBerry PlayBook Code Listings

1. **PlayBook Iteration 1**

   Main Page

```
<!DOCTYPE html>
<html lang="en"><head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>Main Page</title>
            <LINK href="lib/css/MainPageCSS.css" rel="stylesheet" type="text/css">
</head>
<body>
    <div id='buttonpos'>
    <span id='button'>
      <button type="button" class="buttondesign"
onclick="parent.location='SignInMain.html'">Sign In/Out</button>
      <button type="button" class="buttondesign"
onclick="parent.location='UIDEMOFINAL.html'">Room Booking</button>
      </span>
      </div>

            <div id='main'></div>
    <div id='clock'>
    <script type="text/javascript">
    //clock
        var currentTime = new Date()
        var hours = currentTime.getHours()
        var minutes = currentTime.getMinutes()
        if (minutes < 10) {
            minutes = "0" + minutes
        }
        document.write(hours + ":" + minutes + " ")
        if (hours > 11) {
            document.write("PM")
        } else {
            document.write("AM")
        }
    </script>
    </div>
</body>
</html>
```

jqGrid Page

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Room Booking</title>

   <!--main page css-->
   <Link href="lib/css/MainPageCSS.css" rel="stylesheet" type="text/css">
   <!--jquery component css(grid and ui)-->
   <link rel="stylesheet" type="text/css" href="lib/js/jqueryUI/css/smoothness/jquery-ui-
1.8.17.custom.css"/>
   <link rel="stylesheet" type="text/css" href="lib/css/ui.jqgrid.css"/>



   <!--javascript sources-->
   <script type="text/javascript" src="lib/js/jquery-1.7.1.js"></script>
   <script type="text/javascript" src="lib/js/jqueryUI/js/jquery-ui-
1.8.17.custom.min.js"></script>
   <script type="text/javascript" src="lib/js/jqGrid/js/i18n/grid.locale-en.js"></script>
   <script type="text/javascript" src="lib/js/jqGrid/js/jquery.jqGrid.min.js"></script
</head>
<body>

<!-- main test
<script type="text/javascript">
   $(document).ready(function () {
         $.getJSON("http://localhost:6188/RoomBookingService.svc/GetRooms?callback=?",
function (data) {
            $.each(data, function (key, val) {
               $("#rooms").append(val.RoomId + " ", val.RoomName + " ", val.Capacity);
            });
         });
      });
</script>
-->

<!-- jqGrid test -->

<script type="text/javascript">
```

```
//jqGrid
   $(document).ready(function () {
       jQuery("#roomtable").jqGrid({
           url: "http://localhost:6188/RoomBookingService.svc/GetRooms?callback=?",
           datatype: "json",
           colNames: ['Room Id', ' Room Name', 'Capacity', 'Facilities', 'Status', 'Book Rooms'],
           colModel: [
                                    { name: 'RoomId', index: 'RoomId', width: 75, sorttype: "int",
hidden: true },
                                    { name: 'RoomName', index: 'RoomName', width: 150 },
               { name: 'Capacity', index: 'Capacity', width: 100, sorttype: "int" },
               { name: 'Facilities', index: 'Facilities', width: 200 },
               { name: 'Status', index: 'Status', width: 100, formatter: statusFormat },
               { name: 'Book Room'}
                               ],
           pager: jQuery('#rooms'),
           rowNum: 20,
           viewrecords: true,
           sortorder: "desc",
           caption: 'Available Rooms',
           width: 800,
           height: 400,
           multipleSearch: true,
           loadonce: true, //enables sorting of rooms with below
           sortable:true,  //enables sorting of rooms

           //enables cell clicking modal forms for room booking

           onSelectRow: function (id) {
                  $(this).jqGrid('viewGridRow', id);
                  },
          /*
           function () {
           //booking selection form
               $("#bookRoom").dialog({
               height: 400,
               width: 350,
               modal: true,
               buttons: {
                  "Book":
                  //cancel room booking
                  "Cancel": function() {
                                              $(this).dialog("close");
```

```
                                        },
                  }
                });
              },

              */
                jsonReader: {
                    repeatitems: false,
                    id: "Id",
                    root: function (obj) { return obj; },
                    page: function (obj) { return 1; },
                    total: function (obj) { return 1; },
                    records: function (obj) { return obj.length; }
                },

            });
            //converts the Status from bool true to available, false to booked.

            function statusFormat(cellvalue, options, rowObject) {
                if (cellvalue == true) {
                    return "Available";
                }
                else {
                    return "Booked";
                }
            }

            //Grid search function
            grid.jqGrid('navGrid','#roomtable',{add:false,del:false,search:true,refresh:false});
                $("#simpleSearch").click(function(){
                    var text = $("#searchText").val();
                    var field = $("#fieldVal").val();
                    var postdata = grid.jqGrid('getGridParam','postData');
                    $.extend(postdata,{filters:'',searchField: field, searchOper: 'cn', searchString: text});
//search oper 'cn'= contains
                    grid.jqGrid('setGridParam', { search: text.length>0, postData: postdata });
                    grid.trigger("reloadGrid",[{page:1}]);
                    });
        });

</script>
<!--jqgrid table components-->
<div id="rooms"></div>
<table id="roomtable"></table>
```

```
<input type="button" id="simpleSearch" class="ui-state-default ui-corner-all" value="Simple
Search" />
<input type="text" id="searchText" />
<select id="fieldVal">
<option value="Room Name">Room Name</option>
<option value="Room Id">Room Id</option>
</select>

</body>

</html>
```

2. PlayBook Application Iteration 2

Modified jqGrid Page

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Room Booking</title>

   <!--main page css-->
   <Link href="lib/css/MainPageCSS.css" rel="stylesheet" type="text/css">

   <!--jquery component css(grid and ui)-->
   <link rel="stylesheet" type="text/css" href="lib/js/jqueryUI/css/smoothness/jquery-ui-
1.8.17.custom.css"/>
   <link rel="stylesheet" type="text/css" href="lib/css/ui.jqgrid.css"/>

   <!--javascript sources-->
   <script type="text/javascript" src="lib/js/jquery-1.7.1.js"></script>
   <script type="text/javascript" src="lib/js/jqueryUI/js/jquery-ui-
1.8.17.custom.min.js"></script>
   <script type="text/javascript" src="lib/js/jqGrid/js/i18n/grid.locale-en.js"></script>
   <script type="text/javascript" src="lib/js/jqGrid/js/jquery.jqGrid.min.js"></script>

</head>

<body>

<!-- main test
<script type="text/javascript">
   $(document).ready(function () {
         $.getJSON("http://localhost:6188/RoomBookingService.svc/GetRooms?callback=?",
function (data) {
            $.each(data, function (key, val) {
               $("#rooms").append(val.RoomId + " ", val.RoomName + " ", val.Capacity);
            });
         });
      });
</script>
-->

<!-- jqGrid test -->
```

```
<script type="text/javascript">
//jqGrid
   $(document).ready(function () {
      jQuery("#roomtable").jqGrid({
         url: "http://localhost:6188/RoomBookingService.svc/GetRooms?callback=?",
         datatype: "json",
         colNames: ['Room Id', ' Room Name', 'Capacity', 'Facilities', 'Status', 'Book Rooms'],
         colModel: [
                                    { name: 'RoomId', index: 'RoomId', width: 75, sorttype: "int",
hidden: true },
                                    { name: 'RoomName', index: 'RoomName', width: 150 },
            { name: 'Capacity', index: 'Capacity', width: 100, sorttype: "int" },
            { name: 'Facilities', index: 'Facilities', width: 200 },
            { name: 'Status', index: 'Status', width: 100, formatter: statusFormat },
            { name: 'Book Room'}
                           ],
         pager: jQuery('#rooms'),
         rowNum: 20,
         viewrecords: true,
         sortorder: "desc",
         caption: 'Available Rooms',
         width: 800,
         height: 400,
         multipleSearch: true,
         loadonce: true, //enables sorting of rooms with below
         sortable:true,  //enables sorting of rooms

         //enables cell clicking modal forms for room booking

         onSelectRow: function (id) {
               $(this).jqGrid('viewGridRow', id);
               },
        /*
         function () {
         //booking selection form
            $("#bookRoom").dialog({
            height: 400,
            width: 350,
            modal: true,
            buttons: {
               "Book":

               //cancel room booking
               "Cancel": function() {
```

```
                                        $(this).dialog("close");
                            },
                      }
                  });
                },

                */

                 jsonReader: {
                     repeatitems: false,
                     id: "Id",
                     root: function (obj) { return obj; },
                     page: function (obj) { return 1; },
                     total: function (obj) { return 1; },
                     records: function (obj) { return obj.length; }
                 },

              });

              //converts the Status from bool true to available, false to booked.

              function statusFormat(cellvalue, options, rowObject) {
                 if (cellvalue == true) {
                     return "Available";
                 }
                 else {
                     return "Booked";
                 }
              }
              //Grid search function
              grid.jqGrid('navGrid','#roomtable',{add:false,del:false,search:true,refresh:false});
                 $("#simpleSearch").click(function(){
                     var text = $("#searchText").val();
                     var field = $("#fieldVal").val();
                     var postdata = grid.jqGrid('getGridParam','postData');
                     $.extend(postdata,{filters:'',searchField: field, searchOper: 'cn', searchString: text});
//search oper 'cn'= contains
                     grid.jqGrid('setGridParam', { search: text.length>0, postData: postdata });
                     grid.trigger("reloadGrid",[{page:1}]);
                     });
    });
    });
</script>
<!--jqgrid table components-->
```

```
<div id="rooms"></div>
<table id="roomtable"></table>
<!--book room components-->
<div style="display:none" class="modalforms1">
<div id="bookRoom" title="Book This Room?">
   <form>
   <fieldset>
      <label for="roomname">Room Name:</label><p id="rnval"></p>
      <br />
      <label for="capacity">Capacity:</label><p id="capval"></p>
      <br />
      <label for="facilities">Facilities:</label><p id="facval"></p>
      <br />
      <label for="status">Status:</label><p id="statval"></p>
   </fieldset>
   </form>
</div>
</div>
<!--final book room form-->
<div style="display:none" class="modalforms2">
<div id="bookRoomFinal" title="Book Room">
   <form>
   <fieldset>
      <label for="roomdetails">Room Name:</label><p id="rn1"></p>
      <div id="minuteslider"></div><input type="text" id="minutes" style="border:0;
color:#000000; font-weight:bold;"/>
      <script type="text/javascript">
         $(function () {
            $("#minuteslider").slider({
               range: "max",
               min: 0,
               max: 60,
               values: 2,
               step: 15,
               slide: function (event, ui) {
                  $("#minutes").val(ui.value);
               }
            });
            $("#minutes").val($("#minuteslider").slider("value"));
         });
      </script>
      <br />

   </fieldset>
   </form>
</div>
</div>

<br />

<input type="button" id="simpleSearch" class="ui-state-default ui-corner-all" value="Simple
Search" />
<input type="button" id="customSearch" class="ui-state-default ui-corner-all"
value="Custom Search" />

<!--search modal forms -->
<script type="text/javascript">
   $("#simpleSearch").click(function () {
      $("#simpleSearchForm").dialog({
         height: 300,
         width: 350,
         modal: true,
         buttons: {
            "Search": function(){
            },

            "Cancel": function() {
                                          $(this).dialog("close");
                                    },

         }
      });
   });

   $("#customSearch").click(function () {
      $("#customSearchForm").dialog({
      height: 500,
         width: 400,
         modal: true,
         buttons: {
            "Search": function(){
            },
            "Cancel": function() {
                                          $(this).dialog("close");
                                    },
         }
      });
```

```
</script>

<input type="button" id="simpleSearch" class="ui-state-default ui-corner-all" value="Simple
Search" />
<input type="text" id="searchText" />
<select id="fieldVal">
<option value="Room Name">Room Name</option>
<option value="Room Id">Room Id</option>
</select>

</body>

</html>
```

3. PlayBook Application Iterations 3 and 4

Config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<widget xmlns=" http://www.w3.org/ns/widgets"
        xmlns:rim="http://www.blackberry.com/ns/widgets"
        version="2.0.0.0">

  <name>Room Booking</name>

  <description>
    A sample room blackberry PlayBook application to demonstrate the concept of a room
booking service in conjunction with wcf
    and sql server 2008.
  </description>
  <rim:orientation mode="auto"/>
  <rim:loadingScreen
                    onLocalPageLoad="true">
  <rim:transitionEffect type="zoomIn"/>
  </rim:loadingScreen>

  <icon src="appicon.png"/>

  <content src="RoomBookingMain.html"/>

  <feature id="blackberry.ui.dialog"/>
  <access uri="http://www.somedomain.com" subdomains="true">
    <feature id="blackberry.app.event"/>
    <feature id="blackberry.invoke"/>
  </access>

</widget>
```

CSS (Universal)

```
body
{
        height: 100%;
        margin: 0;
        padding: 0;
        background-color:#E6E6E6;
        background-size:1024px 600px;
        background-position:left top;
}

#logoimg
{
   background-position: center;
   margin-left: 250px;
   margin-right: auto;
   margin-top: 200px;
}

/*ROOM BOOKING MAIN */

/* buttons */
#info
{
  outline: 0;
  margin:0 4px 0 0;
  padding: .4em 1em;
  text-decoration:none !important;
  cursor:pointer;
  position: relative;
  text-align: center;
  margin-left: 350px;
  margin-right: auto;
  margin-top: 50px;
  zoom: 1;
  }

#book
{
  outline: 0;
  margin:0 4px 0 0;
  padding: .4em 1em;
```

```
   text-decoration:none !important;
   cursor:pointer;
   position: relative;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   zoom: 1;
   }



/*ROOM BOOKING MAP */


#info2
{
   outline: 0;
   margin:0 4px 0 0;
   padding: .4em 1em;
   text-decoration:none !important;
   cursor:pointer;
   position: relative;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   zoom: 1;
   }

#cancel
{
   outline: 0;
   margin:0 4px 0 0;
   padding: .4em 1em;
   text-decoration:none !important;
   cursor:pointer;
   position: relative;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   top: 20px;
   left: 25px;
   zoom: 1;
   }
```

```
#roomlist
{
   outline: 0;
   margin:0 4px 0 0;
   padding: .4em 1em;
   text-decoration:none !important;
   cursor:pointer;
   position: relative;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   top: 40px;
   left: 10px;
   zoom: 1;
   }


#rooms
{
    position: fixed;
    z-index: -100;
}

#buttons
 {
     position: fixed;
     top: 250px;
     left: 40px;
     z-index: 100;
 }

   /*CALENDAR PAGE*/
#calendar
{
    width: 924px;
    height: 400px;
    z-index: 150;
}

.calendarbuttons
 {
     position: absolute;
     top: 0px;
     left: 5px;
     z-index: 200;
 }

 #cancelcalendar
{
   outline: 0;
   margin:0 4px 0 0;
   padding: .4em 1em;
   text-decoration:none !important;
   cursor:pointer;
   position: static;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   zoom: 1;
   }

#info3
{
   outline: 0;
   margin:0 4px 0 0;
   padding: .4em 1em;
   text-decoration:none !important;
   cursor:pointer;
   position: relative;
   text-align: center;
   margin-left: auto;
   margin-right: auto;
   zoom: 1;
   }

}
```

RoomBookingMain.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Room Booking Application</title>

   <link rel="Stylesheet" href="lib/css/MainRBStyleSheet.css" type="text/css" />
   <link rel="Stylesheet" href="lib/js/jqueryUI/css/smoothness/jquery-ui-1.8.17.custom.css"
type="text/css" />

   <script src="lib/js/jquery-1.7.1.js" type="text/javascript"></script>
   <script src="lib/js/jqueryUI/js/jquery-ui-1.8.17.custom.min.js"
type="text/javascript"></script>

</head>
<body>

   <!-- Page Logo -->

   <div id="logoimg">
   <img src="lib/assets/rbpng.png" alt="Room Booking Logo"/>
   </div>

   <!--App Information Button & Script -->
   <input type="button" id="info" value="Information" class="ui-widget"/>

   <div id="information" style="display:none">
   <p>This app lets you book conference and meeting rooms situated in the building.
   To start the room booking process, please tap "BOOK A ROOM". Information is available
at every stage in the app to help you book a room.</p>
   </div>

   <script type="text/javascript">
      $(function () {
         $("#info").click(function () {
            $("#information").dialog({
               modal: true,
               buttons: {
                  "Close": function () {
                     $(this).dialog("close");
                  }
               }
            });
            $(".ui-dialog-titlebar").hide()
        });
    });
</script>

<!--Page Link Button & Script -->
<input type="button" id="book" value="Book a Room" class="ui-widget" />
<script type="text/javascript">
    $(function () {
        $("#book").click(function () {
            document.location.href = "RoomBookingMap.html";
        });
    });
</script>

</body>
</html>
```

RoomBookingMap.html

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Room Booking Map Selection</title>

   <link rel="Stylesheet" href="lib/css/MainRBStyleSheet.css" type="text/css" />
   <link rel="Stylesheet" href="lib/js/jqueryUI/css/smoothness/jquery-ui-1.8.17.custom.css"
type="text/css" />

   <script src="lib/js/jquery-1.7.1.js" type="text/javascript"></script>
   <script src="lib/js/jqueryUI/js/jquery-ui-1.8.17.custom.min.js"
type="text/javascript"></script>

</head>
<body>

<!-- ROOM BOOKING MAP SVG START -->

<div id="rooms">
<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
         width="1024px" height="600px" viewBox="0 0 1024 600" enable-background="new
0 0 1024 600" xml:space="preserve">

<rect x="15" y="14" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="196" height="193" id="room1click" />
<rect x="211" y="14" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="178" height="119"/>
<rect x="389" y="14" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="184" height="119"/>
<rect x="573" y="14" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="182" height="119" id="room2click"/>
<rect x="755" y="14" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="251" height="174"/>

<rect x="272" y="230" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="183" height="121" id="room3click"/>
<rect x="455" y="230" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="175" height="121"/>
```

```
<rect x="828" y="188" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="178" height="205"/>

<rect x="755" y="393" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="251" height="191" id="room4click"/>
<rect x="573" y="469" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="182" height="115"/>
<rect x="389" y="469" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="184" height="115" id="room5click"/>
<rect x="211" y="469" fill="#FFFFFF" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="178" height="115"/>
<rect x="15" y="469" fill="#50FA5B" stroke="#000000" stroke-width="2" stroke-
miterlimit="10" width="196" height="115" id="room6click"/>

</svg>
</div>

<script type="text/javascript">
//demo room for concept, top right corner
   $("#room1click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=1";
//has the room id tagged onto the url to load the bookings for the calendar
      });

   $("#room2click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=2";
   });

   $("#room3click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=3";
   });

   $("#room4click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=4";
   });

   $("#room5click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=5";
   });

   $("#room6click").click(function () {
      document.location.href = "RoomBookingCalendarView.html?roomid=6";
   });
```

```
</script>

<!-- ROOM BOOKING MAP SVG END -->

<!--App Information Button & Script -->
<div id="buttons">
    <input type="button" id="info2" value="Information" class="ui-widget" />
    <div id="information" style="display:none">
    <p>Clicking the highlighted rooms will take you to the calendar view of that room where
you can see which times the room can be booked.
    <span style="color:Green">GREEN</span> indicates the room is currently <span
style="color:Green">AVAILABLE</span>,
    <span style="color:Red">RED</span> indicates the room is currently <span
style="color:Red">BOOKED</span>, although you may still view the room calendar at any
time.
    Clicking the "ROOM LIST" button will show the rooms as a list of names.</p>
    </div>

    <script type="text/javascript">
        $(function () {
            $("#info2").click(function () {
                $("#information").dialog({
                    modal: true,
                    buttons: {
                        "Close": function () {
                            $(this).dialog("close");
                        }
                    }
                });
                $(".ui-dialog-titlebar").hide()
            });
        });
    </script>

    <br />

    <!--Page Link Button & Script -->
    <input type="button" id="cancel" value="cancel" class="ui-widget" />
    <script type="text/javascript">
        $(function () {
            $("#cancel").click(function () {
                document.location.href = "RoomBookingMain.html";
            });
        });
    </script>
    <br />
</div>
</body>
</html>
```

RoomBookingCalendarView.html

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Room Booking Calendar View</title>

   <link rel="Stylesheet" href="lib/css/MainRBStyleSheet.css" type="text/css" />
   <link rel="Stylesheet" href="lib/js/jqueryUI/css/smoothness/jquery-ui-1.8.17.custom.css"
type="text/css" />
   <link rel="Stylesheet" href="lib/js/fullcalendar-1.5.3/fullcalendar/fullcalendar.css"
type="text/css" />
   <link rel="Stylesheet" href="lib/js/jquery-ui-timepicker-0.2.9/jquery.ui.timepicker.css"
type="text/css" />

   <script src="lib/js/datajs-1.0.2.min.js" type="text/javascript"></script>
   <script src="lib/js/jquery-1.7.1.js" type="text/javascript"></script>
   <script src="lib/js/json2.js" type="text/javascript"></script>

   <script src="lib/js/jqueryUI/js/jquery-ui-1.8.17.custom.min.js"
type="text/javascript"></script>
   <script src="lib/js/fullcalendar-1.5.3/fullcalendar/fullcalendar.min.js"
type="text/javascript"></script>
   <script src="lib/js/jquery-ui-timepicker-0.2.9/jquery.ui.timepicker.js"
type="text/javascript"></script>

</head>
<body>

<!-- CALENDAR DIV -->
<div id="calendar"></div>
<div id="information" style="display:none">
<p>Would you like to book a room on this date?</p>
</div>

<!-- BOOKING FORM DIV -->
<div id="bookingform" style="display:none">

<label for="empname">Booking Name:</label>
<input type="text" name="empname" id="empname" class="text ui-widget-content ui-
corner-all" />
<br />
```

```
<label for="date">Date:</label>
<br />
<input type="text" name="date" id="date" class="text ui-widget-content ui-corner-all"
readonly="readonly"/>
            <!-- DATEPICKER SCRIPT -->
            <script type="text/javascript">
                $(function () {
                    $("#date").datepicker({
                        dateFormat: "yy-mm-dd"
                    });
                });
            </script>

<label for="starttime">Booking Start:</label>
<input type="text" name="starttime" id="starttime" class="text ui-widget-content ui-corner-
all"  readonly="readonly" />
            <!-- STARTTIME PICKER SCRIPT -->
            <script type="text/javascript">
             $("#starttime").timepicker({
                 hours: { starts: 8, ends: 18 },
                    minutes: { interval: 15 },
                    rows: 2
             });
            </script>

<label for="endtime">Booking End:</label>
<input type="text" name="endtime" id="endtime" class="text ui-widget-content ui-corner-
all" readonly="readonly"/>
            <!-- ENDTIME PICKER SCRIPT -->
            <script type="text/javascript">
              $("#endtime").timepicker({
                  hours: { starts: 8, ends: 18 },
                  minutes: { interval: 15 },
                  rows: 2
                });
            </script>

</div>

<!-- CALENDAR CODE -->
<script type="text/javascript">

//gets the roomid from the url
```

```
function getUrlVars() {
        var vars = {};
        var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi, function (m, key,
value) {
            vars[key] = value;
        });
        return vars;
    }

    var first = getUrlVars()["roomid"];


    $(document).ready(function () {

        //initiate calendar
        $("#calendar").fullCalendar({
            disableDragging: true,
            disableResizing: true,
            header: {
                left: '',
                center: 'title',
                right: 'month, agendaWeek'
            },

            editable: false,
            dayClick: function (date, allDay, jsEvent, view) {

                if (allDay) {

                    $("#information").dialog({
                        modal: true,
                        buttons: {
                            "Cancel": function () {
                                $(this).dialog("close");
                            },

                            "Book a Room": function () {
                                $("#bookingform").dialog({
                                    buttons: {
                                        "Book Room": function () {
```

```
//Converts the datetime concat format into milliseconds, which the database and WCF like!

// Takes the date from datepicker, splits it into three numerical values, reassembles that as a js
"Date" object
// gets the time in milliseconds. Months must have 1 subtracted as js months range 0-11 and
as such give us an incremented month.

var d = $("#date").val();
 var dateParts = new Date((Number(d.split("-")[0])), (Number(d.split("-")[1]) - 1),
(Number(d.split("-")[2])));
 var dateis = dateParts.getTime();

// Takes the time from the timepicker, splits it into the numerical values, and calculates the
milliseconds accordingly.
var timeEnd = $("#endtime").val();
var time1 = ((Number(timeEnd.split(':')[0]) * 60) * 60 + Number(timeEnd.split(':')[1]) * 60) *
1000;

 var timeStart = $("#starttime").val();
 var time2 = ((Number(timeStart.split(':')[0]) * 60) * 60 + Number(timeStart.split(':')[1]) *
60) * 1000;

 //Final datetime calculations.
 var dateTimeEnd = dateis + time1;
 var dateTimeStart = dateis + time2;

//data is serialised into the correct format here. RoomID first is the retrieved roomid from url.
var data = { "BookingEnd": "\/Date(" + dateTimeEnd + ")/", "BookingID": 1,
"BookingName": $("#empname").val(), "BookingStart": "\/Date(" + dateTimeStart + ")/",
"RoomID": first };

var booking = JSON.stringify(data);

                    $.ajax({
                     type: "POST",
                     url:"http://localhost:6188/RoomBookingService.svc/createbooking",
                        dataType: "json",
                        data: booking,
                        contentType: "application/json; charset=utf-8",
                        success: function () {
                            $("#calendarconfirm").dialog({
                                buttons: {
                                    "Close": function () {
```

```
                                    document.location.href = "RoomBookingMain.html";
                                }
                            }
                        });
                        //remove titlebar from dialog
                        $(".ui-dialog-titlebar").hide();
                    },

                    error: function () {
                        $("#calendarerror").dialog({
                            buttons: {
                                "Close": function () {
                                    $(this).dialog("close");
                                }
                            }
                        });
                        //remove titlebar from dialog
                        $(".ui-dialog-titlebar").hide();
                    }
                });

            },

            "Cancel": function () {
                $(".ui-dialog-content").dialog("close"); //closes all open dialogs
            }
        }
    });

    //remove titlebar from dialog
    $(".ui-dialog-titlebar").hide();
  }
 }
});

//remove titlebar from dialog
$(".ui-dialog-titlebar").hide();
}
},

//removes weekends from calendar
weekends: false,
events: [
{
```

```
            title: 'Michael Meina',
            start: new Date(1332149400000+0000),
            end: new Date(1332153000000 + 0000),
            allDay: false

          },
          {
            title: 'James Brown',
            start: new Date(1332154800000 + 0000),
            end: new Date(1332158400000 + 0000),
            allDay: false

          },
          {
            title: 'Graeme Kane',
            start: new Date(1332417600000 + 0000),
            end: new Date(1332424800000 + 0000),
            allDay: false
          },
          {
            title: 'Paul McCartney',
            start: new Date(1332843300000 + 0000),
            end: new Date(1332848700000 + 0000),
            allDay: false
          },
          {
            title: 'Jim Morrison',
            start: new Date(1332835200000 + 0000),
            end: new Date(1332838800000 + 0000),
            allDay: false
          },
          {
            title: 'Bob Dylan',
            start: new Date(1333013400000 + 0000),
            end: new Date(1333017000000 + 0000),
            allDay: false
          }
          ]
        });
    });

</script>

<div id="calendarerror" style="display:none">
```

```
<p>Sorry, an error has occurred, please try again.</p>
</div>

<div id="calendarconfirm" style="display:none">
<p>Thank you, your booking was <span style="color:Green">Successful!</span></p>
</div>

<!-- INFORMATION BUTTON -->
<div class="calendarbuttons">
   <input type="button" id="info3" value="Information" class="ui-widget" />
   <div id="pageinfo" style="display:none">
   <p>Click anywhere on the calendar, and select the "BOOK ROOM" option. Fill in the
booking form which appears.</p>
   </div>

   <script type="text/javascript">
      $(function () {
         $("#info3").click(function () {
            $("#pageinfo").dialog({
               modal: true,
               buttons: {
                  "Close": function () {
                     $(this).dialog("close");
                  }
               }
            });
            $(".ui-dialog-titlebar").hide()
         });
      });
   </script>

      <!--PAGE CANCEL TO MAP PAGE -->
   <input type="button" id="cancelcalendar" value="Cancel" class="ui-widget" />
   <script type="text/javascript">
      $(function () {
         $("#cancelcalendar").click(function () {
            document.location.href = "RoomBookingMap.html";
         });
      });
   </script>
</div>
</body>
</html>
```

Appendix F – WCF Service Application Code Listings

1. WCF Service Application Iteration 1

Room Repository

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

namespace RoomBookingServices
{
   public static class RoomRepository
       {
       public static List<Room> rooms = new List<Room>()
              {
new Room { RoomId = 1, RoomName="Gigantic Room", Capacity = 20, Facilities = "Computer, Projector", 
Status = false },
new Room { RoomId = 2, RoomName="Big Room", Capacity = 10, Facilities = "Projector", Status = true },
new Room { RoomId = 3, RoomName="Medium Room", Capacity = 8, Facilities = "Computer", Status = false 
},
new Room { RoomId = 4, RoomName="Small Room", Capacity = 6, Facilities = "Computer", Status = true },
new Room { RoomId = 5, RoomName="Tiny Room", Capacity = 4, Facilities = "Smartboard, Microwave", 
Status = false },
new Room { RoomId = 6, RoomName="Smallest Room", Capacity = 2, Status = true },
 new Room { RoomId = 7, RoomName="Gigantic Room 2", Capacity = 20, Facilities = "Books", Status = false 
},
new Room { RoomId = 8, RoomName="Big Room 2", Capacity = 10, Facilities = "Conference Phone", Status = 
true },
new Room { RoomId = 9, RoomName="Medium Room 2", Capacity = 8, Facilities = "Computer", Status = 
false },
new Room { RoomId = 10, RoomName="Small Room 2", Capacity = 6, Facilities = "Phone, Computer", Status 
= true },
new Room { RoomId = 11, RoomName="Tiny Room 2", Capacity = 4, Facilities = "Smartboard, Phone", Status 
= false },
new Room { RoomId = 12, RoomName="Smallest Room 2", Capacity = 2, Status = true }
              };
       }
}

// Status true indicated room is available, false indicated room is booked
```

IRoomBookingService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace RoomBookingServices
{
   // NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface
name "IRoomBookingJSONService" in both code and config file together.
   [ServiceContract]
   public interface IRoomBookingService
   {
      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "GetRooms")]

      Room[] GetRooms();

[OperationContract]
[WebInvoke(Method = "GET", BodyStyle = WebMessageBodyStyle.Bare, RequestFormat =
WebMessageFormat.Json,
ResponseFormat = WebMessageFormat.Json, UriTemplate =
"SaveRooms?roomid={roomid}&roomname={roomname}&capacity={capacity}&facilities={faciliti
es}&status={status}")]

      Room[] SaveRooms(int roomid, string roomname, int capacity, string facilities, bool status);

   }

   [DataContract]
   public class Room
   {
      [DataMember(Name="RoomId")]
      public int RoomId{ get; set; }

      [DataMember(Name = "RoomName")]
      public string RoomName { get; set; }

      [DataMember(Name = "Capacity")]
      public int Capacity { get; set; }

      [DataMember(Name = "Facilities")]
      public string Facilities { get; set; }
```

```
        [DataMember(Name = "Status")]
        public bool Status { get; set; }
    }
}
```

RoomBookingService.svc

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace RoomBookingServices
{
   public class RoomBookingService : IRoomBookingService
   {
         public Room[] GetRooms()
         {
            return RoomRepository.rooms.ToArray();
         }


         public Room[] SaveRooms(int roomid, string roomname, int capacity, string facilities, bool
status)
         {
            var room = new Room { RoomId = roomid, RoomName = roomname, Capacity = capacity,
Facilities = facilities, Status = status };
            RoomRepository.rooms.Add(room);

            return RoomRepository.rooms.ToArray();
         }


      }

   }
```

Web.Config

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <services>
      <service name="RoomBookingServices.RoomBookingService"
behaviorConfiguration="RoomBookingServiceBehaviour">
        <endpoint address="" binding="webHttpBinding"
bindingConfiguration="webHttpBindingWithJsonP"
contract="RoomBookingServices.IRoomBookingService"
behaviorConfiguration="webHttpBehavior">
        </endpoint>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="RoomBookingServiceBehaviour">
          <!-- To avoid disclosing metadata information, set the value below to false and remove the
metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes, set the value below to true.
Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="webHttpBehavior">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <webHttpBinding>
        <binding name="webHttpBindingWithJsonP" crossDomainScriptAccessEnabled="true" />
      </webHttpBinding>
    </bindings>
    <!--<serviceHostingEnvironment multipleSiteBindingsEnabled="true" />-->
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
  </system.webServer>
</configuration>
```

2. WCF Service Application Iterations 2 and 3

IRoomBookingServices.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;
using System.Data.Sql;

namespace RoomBookingServices
{
   // NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface
name "IRoomBookingJSONService" in both code and config file together.
   [ServiceContract]
   public interface IRoomBookingService
   {
      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "GetRooms")]

      List<Rooms> GetRooms();

      [OperationContract]
      [WebGet(BodyStyle = WebMessageBodyStyle.Bare, UriTemplate = "CreateBooking?")]

      void CreateBooking();
   }

   [DataContract]
   public class Rooms
   {

      [DataMember(Name="Room Id")]
      public int room_id { get; set; }

      [DataMember(Name = "Room Name")]
      public string room_name { get; set; }

      [DataMember(Name = "Room Facilities")]
      public string room_facilities { get; set; }

      [DataMember(Name = "Room Capacity")]
```

```
        public int room_capacity { get; set; }
    }
}
```

RoomBookingService.svc

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Configuration;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlTypes;

namespace RoomBookingServices
{
    public class RoomBookingService : IRoomBookingService
    {

        public List<Rooms> GetRooms()
        {
            List<Rooms> rooms = new List<Rooms>();

            using (SqlConnection objconn = new SqlConnection("Data Source=HAL;Initial
Catalog=RoomBookingDatabase;Integrated Security=True"))
            {
                if (objconn.State == ConnectionState.Closed)
                {
                    objconn.Open();
                }

                using (SqlCommand objcmd = new SqlCommand("Select * from Rooms", objconn))
                {
                    SqlDataReader reader = objcmd.ExecuteReader();
                    while (reader.Read())
                    {
                        Rooms roomObj = new Rooms();
                        roomObj.room_id = reader.GetInt32(0);
                        roomObj.room_name = reader["room_name"].ToString();
                        roomObj.room_facilities = reader["room_facilities"].ToString();
                        roomObj.room_capacity = reader.GetInt32(3);
                        rooms.Add(roomObj);
                    }
                }
            }

            return rooms;

        }

        public void CreateBooking()
        {
            SqlConnection objconn = new SqlConnection("Data Source=HAL;Initial
Catalog=RoomBookingDatabase;Integrated Security=True");
            {
                if (objconn.State == ConnectionState.Closed)
                {
                    objconn.Open();
                }

                SqlCommand objcmd = new SqlCommand("INSERT INTO Bookings(booking_id,
room_id, emp_name, date, start_time, end_time)  VALUES (3, 2, 'Michael', '2012-02-30', '12:00:00',
'13:00:00')", objconn);
                objcmd.ExecuteNonQuery();
                objconn.Close();

            }

        }

    }
```

Web.Config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0">
      <assemblies>
        <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
      </assemblies>
    </compilation>
  </system.web>
  <system.serviceModel>
    <services>
      <service name="RoomBookingServices.RoomBookingService"
behaviorConfiguration="RoomBookingServiceBehaviour">
        <endpoint address="" binding="webHttpBinding"
bindingConfiguration="webHttpBindingWithJsonP"
contract="RoomBookingServices.IRoomBookingService"
behaviorConfiguration="webHttpBehavior"></endpoint>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="RoomBookingServiceBehaviour">
          <!-- To avoid disclosing metadata information, set the value below to false and remove the
metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true" />
          <!-- To receive exception details in faults for debugging purposes, set the value below to true.
Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="webHttpBehavior">
          <webHttp />
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <webHttpBinding>
        <binding name="webHttpBindingWithJsonP" crossDomainScriptAccessEnabled="true" />
      </webHttpBinding>
    </bindings>
    <!--<serviceHostingEnvironment multipleSiteBindingsEnabled="true" />-->
  </system.serviceModel>
  <system.webServer>
```

```
    <modules runAllManagedModulesForAllRequests="true" />
  </system.webServer>
  <connectionStrings>
    <add name="RoomBookingDatabaseEntities"
connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=Sy
stem.Data.SqlClient;provider connection string=&quot;data source=HAL;initial
catalog=RoomBookingDatabase;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

3. WCF Service Application Iteration 4

RoomBookingDataService.svc

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Data.Services.Common;
using System.Linq;
using System.ServiceModel.Web;
using System.Web;
using DataServicesJSONP;

namespace RoomBookingService
{
   [JSONPSupportBehavior]
   public class RoomBookingDataService : DataService<RoomBookingDBEntities2>
   {

       public static void InitializeService(DataServiceConfiguration config)
       {
           config.UseVerboseErrors = true;
           config.SetEntitySetAccessRule("Rooms", EntitySetRights.AllRead);
           config.SetEntitySetAccessRule("Bookings", EntitySetRights.All);
           config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);
           config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
       }
   }
}
```

Web.Config

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0">
      <assemblies>
        <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
      </assemblies>
    </compilation>
  </system.web>
  <connectionStrings>
    <add name="RoomBookingDBEntities"
connectionString="metadata=res://*/RoomBookingModel.csdl|res://*/RoomBookingModel.ssdl|re
s://*/RoomBookingModel.msl;provider=System.Data.SqlClient;provider connection
string=&quot;data source=hal;initial catalog=RoomBookingDB;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
    <add name="RoomBookingDBEntities1"
connectionString="metadata=res://*/RBDataModel.csdl|res://*/RBDataModel.ssdl|res://*/RBData
Model.msl;provider=System.Data.SqlClient;provider connection string=&quot;data
source=hal;initial catalog=RoomBookingDB;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
    <add name="RoomBookingDBEntities2"
connectionString="metadata=res://*/RBModel.csdl|res://*/RBModel.ssdl|res://*/RBModel.msl;pr
ovider=System.Data.SqlClient;provider connection string=&quot;data source=hal;initial
catalog=RoomBookingDB;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
  </system.serviceModel>
</configuration>
```

4. WCF Service Application Iterations 5 and 6

IRoomBookingService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;
using System.Data.Sql;
using RoomBookingServices.Entities;

namespace RoomBookingServices
{
    [ServiceContract]
    public interface IRoomBookingService
    {
        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json, BodyStyle =
WebMessageBodyStyle.WrappedRequest, UriTemplate = "getrooms")]
        List<Room> GetRooms();

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json, BodyStyle =
WebMessageBodyStyle.WrappedRequest, UriTemplate = "getbookings")]
        List<Booking> GetBookings();


        [WebInvoke(Method = "POST", RequestFormat = WebMessageFormat.Json, UriTemplate =
"createbooking")]
        void CreateBooking(Booking booking);
    }

    [DataContract]
    public class Room
    {
        [DataMember(Name = "RoomID")]
        public int RoomID { get; set; }

        [DataMember(Name = "RoomName")]
        public string RoomName { get; set; }

    }
    [DataContract]
    public class Booking
    {

        [DataMember(Name = "BookingID")]
        public int BookingID { get; set; }

        [DataMember(Name="BookingName")]
        public string BookingName { get; set; }

        [DataMember(Name="BookingStart")]
        public DateTime BookingStart { get; set; }

        [DataMember(Name = "BookingEnd")]
        public DateTime BookingEnd { get; set; }

        [DataMember(Name = "RoomID")]
        public int RoomID { get; set; }

    }
}
```

RoomBookingService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Configuration;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Activation;
using System.Text;
using System.Data;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using RoomBookingServices.Entities;
using RoomBookingServices.Model;

namespace RoomBookingServices
{
   [ServiceBehavior(IncludeExceptionDetailInFaults = true, InstanceContextMode =
InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Single)]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   public class RoomBookingService : IRoomBookingService
   {

      //retrieves all rooms
      public List<Room> GetRooms()
      {
         RoomsModel model = new RoomsModel();
         List<RoomEntity> roomEntities = model.GetRoom();
         List<Room> rooms = new List<Room>();

         foreach (var roomEntity in roomEntities)
         {
            Room room = new Room()
                        {
                           RoomID = roomEntity.RoomID,
                           RoomName = roomEntity.RoomName
                        };
            rooms.Add(room);
         }
         model.Close();
         return rooms;
```

```
        }

        //retrieves all bookings
        public List<Booking> GetBookings()
        {
            BookingsModel model = new BookingsModel();
            List<BookingEntity> bookingEntities = model.GetBooking();
            List<Booking> bookings = new List<Booking>();

            foreach (var bookingEntity in bookingEntities)
            {
                Booking booking = new Booking()
                                  {
                                      BookingName = bookingEntity.BookingName,
                                      BookingStart = bookingEntity.BookingStart,
                                      BookingEnd = bookingEntity.BookingEnd,
                                      RoomID = bookingEntity.RoomID
                                  };
                bookings.Add(booking);
            }

            model.Close();

            return bookings;
        }

        //Creates a new booking
        public void CreateBooking(Booking booking)
        {
            BookingEntity bookingEntity = new BookingEntity()
                                        {
                                            BookingName = booking.BookingName,
                                            BookingStart = booking.BookingStart,
                                            BookingEnd = booking.BookingEnd,
                                            RoomID = booking.RoomID
                                        };
            BookingsModel model = new BookingsModel();
            model.CreateBooking(bookingEntity);

        }
    }
}
```

BookingsModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.SqlClient;
using RoomBookingServices.Entities;
using System.Data;

namespace RoomBookingServices.Model
{
    public class BookingsModel
    {
        private SqlConnection sql;

        public BookingsModel()
        {
            sql = new SqlConnection("Data Source=hal;Initial Catalog=RoomBookingDB;Integrated
Security=True");
            sql.Open();
        }



        //gets all bookings from the database
        public List<BookingEntity> GetBooking()
        {
            List<BookingEntity> bookings = new List<BookingEntity>();

            using (SqlCommand objcmd = new SqlCommand("Select * from Bookings", sql))
            {
                SqlDataReader reader = objcmd.ExecuteReader();
                while (reader.Read())
                {
                    BookingEntity books = new BookingEntity();

                    //0-4 indicates the SQL database columns
                    books.BookingID = reader.GetInt32(0);
                    books.BookingName = reader.GetString(1);
                    books.BookingStart = reader.GetDateTime(2);
                    books.BookingEnd = reader.GetDateTime(3);
                    books.RoomID = reader.GetInt32(4);

                    bookings.Add(books);
                }
            }
            return bookings;
        }

        //Inserts a booking into the database
        public void CreateBooking(BookingEntity booking)
        {
            using (var conn = new SqlConnection("Data Source=hal;Initial
Catalog=RoomBookingDB;Integrated Security=True"))
            using (var cmd = conn.CreateCommand())
            {
                conn.Open();
                //modify to be where date
                cmd.CommandText =
                    @"IF NOT EXISTS ( SELECT * FROM Bookings WHERE BookingStart =
@BookingStart AND BookingEnd = @BookingEnd AND RoomID= @RoomID )
                        INSERT INTO Bookings ( BookingName, BookingStart, BookingEnd, RoomID )
VALUES ( @BookingName, @BookingStart, @BookingEnd, @RoomID )";

                cmd.Parameters.AddWithValue("@BookingName", booking.BookingName);
                cmd.Parameters.AddWithValue("@BookingStart", booking.BookingStart);
                cmd.Parameters.AddWithValue("@BookingEnd", booking.BookingEnd);
                cmd.Parameters.AddWithValue("@RoomID", booking.RoomID);
                cmd.ExecuteNonQuery();

                conn.Close();
            }
        }

        public void Close()
        {
            sql.Close();
        }
    }
}
```

RoomsModel.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using RoomBookingServices.Entities;
using System.Data.SqlClient;

namespace RoomBookingServices.Model
{
    public class RoomsModel
    {
        private SqlConnection sql;
        public RoomsModel()
        {
            sql = new SqlConnection("Data Source=hal;Initial Catalog=RoomBookingDB;Integrated
Security=True");
            sql.Open();
        }

        //returns all rooms from the database
        public List<RoomEntity> GetRoom()
        {
            List<RoomEntity> rooms = new List<RoomEntity>();

            using (SqlCommand objcmd = new SqlCommand("Select * from Rooms", sql))
            {
                SqlDataReader reader = objcmd.ExecuteReader();
                while (reader.Read())
                {
                    RoomEntity room = new RoomEntity();
                    room.RoomID = reader.GetInt32(0);
                    room.RoomName = reader.GetString(1);
                    rooms.Add(room);
                }
            }
            return rooms;
        }

        public void Close()
        {
            sql.Close();
        }
    }
}
```

BookingEntity.cs

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

namespace RoomBookingServices.Entities
{
    public class BookingEntity
    {
        public int BookingID { get; set; }

        public string BookingName { get; set; }

        public DateTime BookingStart { get; set; }

        public DateTime BookingEnd { get; set; }

        public int RoomID { get; set; }

    }
}
```

RoomEntity.cs

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

namespace RoomBookingServices.Entities
{

    public class RoomEntity
    {
        public int RoomID { get; set; }

        public string RoomName { get; set; }
    }
}
```

Web.Config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
        <!-- tracer for wcf service errors -->
  <system.diagnostics>
    <sources>
      <source name="System.ServiceModel" switchValue="Warning, ActivityTracing"
        propagateActivity="true">
        <listeners>
          <add type="System.Diagnostics.DefaultTraceListener" name="Default">
            <filter type="" />
          </add>
          <add name="ServiceModelTraceListener">
            <filter type="" />
          </add>
        </listeners>
      </source>
    </sources>
    <sharedListeners>
      <add initializeData="c:\users\michael\desktop\room booking
app2\roombookingservices\roombookingservices\web_tracelog.svclog"
        type="System.Diagnostics.XmlWriterTraceListener, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
        name="ServiceModelTraceListener" traceOutputOptions="Timestamp">
        <filter type="" />
      </add>
    </sharedListeners>
  </system.diagnostics>

  <system.web>
    <httpRuntime maxRequestLength="16384"/>
    <authentication mode="None"/>
    <compilation debug="true" targetFramework="4.0">
      <assemblies>
        <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"/>
      </assemblies>
    </compilation>
  </system.web>
  <system.serviceModel>
    <services>
      <service name="RoomBookingServices.RoomBookingService"
behaviorConfiguration="RoomBookingServiceBehaviour">
        <endpoint address="" binding="webHttpBinding"
bindingConfiguration="webHttpBindingWithJsonP"
```

```
contract="RoomBookingServices.IRoomBookingService"
behaviorConfiguration="webHttpBehavior">
          <identity>
            <servicePrincipalName value=""/>
          </identity>
        </endpoint>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="RoomBookingServiceBehaviour">
          <!-- To avoid disclosing metadata information,
          set the value below to false and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="true" />
          <!-- To receive exception details in faults for debugging purposes, set the value below to
true.
          Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="webHttpBehavior">
          <webHttp />
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <webHttpBinding>
        <binding name="webHttpBindingWithJsonP" crossDomainScriptAccessEnabled="true">
        </binding>
      </webHttpBinding>
    </bindings>
    <!--<serviceHostingEnvironment multipleSiteBindingsEnabled="true" />-->
    <!--<serviceHostingEnvironment aspNetCompatibilityEnabled="true"
multipleSiteBindingsEnabled="true" />-->
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
  </system.webServer>
  <connectionStrings>
    <add name="RoomBookingDatabaseEntities"
connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider
=System.Data.SqlClient;provider connection string=&quot;data source=HAL;initial
catalog=RoomBookingDatabase;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
    <add name="RoomBookingDatabaseEntities1"
connectionString="metadata=res://*/RoomBookingDB.csdl|res://*/RoomBookingDB.ssdl|res://*/
```

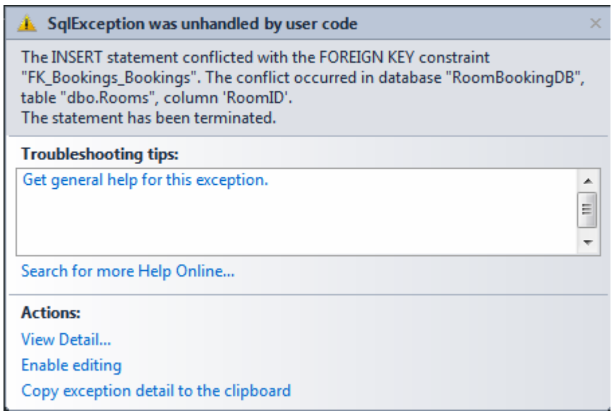
```
RoomBookingDB.msl;provider=System.Data.SqlClient;provider connection string=&quot;data
source=HAL;initial catalog=RoomBookingDatabase;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Appendix G – Fiddler Tests

The following is an example of the Fiddler tests conducted throughout the duration of the application development. Fiddler was used to test the functionality of the WCF service application by testing all outcomes of HTTP request which could be submitted to the service. Fault injection was used to confirm invalid JSON strings and identify unexpected behaviour. The test results are given as HTTP status codes. These are:

- 200: This indicates a successful request.
- 400: This indicates a bad request, typically received when invalid data is submitted.
- 404: Not found. Typically given when trying to access non-existent or inaccessible application areas (such as the root URIs).
- 405: Method not allowed. This is typically given when trying to submit a request which is not supported by the specific resource, such as the "POST" requests on get-only resources.

In certain circumstances, the Visual Studio IDE indicates certain errors, particularly regarding invalid formats. An example of an error given by the IDE is as follows:



SqlException was unhandled by user code

The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Bookings_Bookings". The conflict occurred in database "RoomBookingDB", table "dbo.Rooms", column 'RoomID'.
The statement has been terminated.

Troubleshooting tips:
Get general help for this exception.

Search for more Help Online...

Actions:
View Detail...
Enable editing
Copy exception detail to the clipboard

Included in the test cases is a short description of the test including changes made, the request body (data being submitted for testing), the request type and the request response.

Description	Request Body	Request Type	Response
Test RoomBookingService.svc (tests the root of WCF application)	None	GET	404
Test RoomBookingService.svc (tests the root of WCF application)	None	POST	404
Test /getrooms (tests generic getrooms method for "GET")	None	GET	200

Test /getrooms (tests generic getrooms method for “POST”)	None	POST	405
Test /getrooms (tests generic getrooms method for “DELETE”)	None	DELETE	405
Test /getbookings (tests generic getbookings method for “GET”)	None	GET	200
Test /getbookings (tests generic getbookings method for “POST”)	None	POST	405
Test /getbookings (tests generic getbookings method for “DELETE”)	None	DELETE	405
Test /createbooking (tests empty createbooking method for “GET”)	None	GET	405
Test /createbooking (tests empty createbooking method for “POST”)	None	POST	400
Test /createbooking (tests empty createbooking method for “DELETE”)	None	DELETE	405
Test /createbooking (tests example JSON string using “GET”)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(1335862800000)/", "RoomID":"6"}	GET	405
Test /createbooking (tests example JSON string using “POST”)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(1335862800000)/", "RoomID":"6"}	POST	200
Test /createbooking (tests example JSON string using “DELETE”)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(1335862800000)/", "RoomID":"6"}	DELETE	405
Test /createbooking (tests example JSON string using “POST” with empty date values)	{"BookingEnd":"/Date()/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date()/", "RoomID":"6"}	POST	400
Test /createbooking (tests example JSON string using “POST” with invalid date values)	{"BookingEnd":"/Date(client)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(client)/", "RoomID":"6"}	POST	400

Test /createbooking (tests example JSON string using “POST” with different date values)	{"BookingEnd":"/Date(2012-03-11)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(2012-03-11)/", "RoomID":"6"}	POST	200
Test /createbooking (tests example JSON string using “POST” with invlid date values)	{"BookingEnd":"/Date(11111111)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(11111111)/", "RoomID":"6"}	POST	200
Test /createbooking (tests example JSON string using “POST” with invalid BookingID value)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":I,"BookingName":"ClientTest", "BookingStart":"/Date(1335864600000)/", "RoomID":"6"}	POST	400
Test /createbooking (tests example JSON string using “POST” with invalid BookingName value)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":1,"BookingName": 12345, "BookingStart":"/Date(1335862800000)/", "RoomID":"6"}	POST	200
Test /createbooking (tests example JSON string using “POST” with invalid RoomID)	{"BookingEnd":"/Date(1335864600000)/", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"/Date(1335862800000)/", "RoomID": X}	POST	400
Test /createbooking (tests example JSON string using “POST” with invalid title names (added ‘s’ into names))	{"BookingsEnd":"/Date(1335864600000)/", "BookingsID":1,"BookingsName":"ClientTest", "BookingsStart":"/Date(1335862800000)/", "RoomsID":"6"}	POST	400
Test /createbooking (tests example JSON string using “POST” with invalid dates (removed date identifiers))	{"BookingEnd": ”(1335864600000)", "BookingID":1,"BookingName":"ClientTest", "BookingStart":"(1335862800000)", "RoomID":6}	POST	400
Test /createbooking (tests example JSON string using “POST” with invalid dates (dates as integers))	{"BookingsEnd":1335864600000, "BookingsID":1,"BookingsName":"ClientTest", "BookingsStart":1335862800000, "RoomsID":"6"}	POST	400

Appendix H – System Tests

The following list details the test cases which were completed as part of the project testing phase. Test cases were used to test specific functionality of the application when coupled with the WCF service application.

ID	Description	Input Data	Actions	Expected Result	Actual Result	Status
T1	The application starts successfully	None	Click Room Booking icon	Application loads without error and displays the main page	Application loads without error and displays the main page	Pass
T2	Selecting "information" displays the information form	None	Click information button	Information form displays without error and provides instructions	Information form displays without error and provides instructions	Pass
T3	Information form can be closed	None	Click close button on information form	Information form closes without error returning the user to the main page	Information form closes without error returning the user to the main page	Pass
T4	Selecting "book a room" takes the user to the next page	None	Click book a room button	The user is taken to the map view page	The user is taken to the map view page	Pass
T5	Selecting "information" displays the information form	None	Click information button on map page	Information form displays without error and provides instructions	Information form displays without error and provides instructions	Pass
T6	Selecting "cancel" returns the user to the start screen	None	Click close button on information form	Information form closes without error and returns the user to the	Information form closes without error and returns the user to the	Pass
T7	Select cancel button on map page returns the user to the main page	None	Click cancel button on map page	The user is returned to the main page	The user is returned to the main page	Pass
T8	Selecting a highlighted room takes users to the next page	None	Click any of the green "rooms" on the map	The user is taken to the calendar page	The user is taken to the calendar page	Pass
T9	Selecting un-highlighted rooms do nothing	None	Click any of the white "rooms" on the map	No action	No action	Pass
T10	Selecting "information" displays the information form	None	Click information button on calendar view page	Information form displays without error and provides instructions	Information form displays without error and provides instructions	Pass
T11	Selecting "cancel" returns the user to the previous screen	None	Click close button on information form	Information form closes without error returning the user to the map view page	Information form closes without error returning the user to the map view page	Pass
T12	Selecting "week" shows the weekly calendar view.	None	Click week button next to month button	The calendar week view is loaded and displays mock events	The calendar week view is loaded and displays mock events	Pass
T13	Clicking any of the date cells asks if the user wishes to book a room	None	Click anywhere on the calendar date cells	The first booking room form is displayed	The first booking room form is displayed	Pass
T14	Clicking "cancel" on this form cancels the forms display	None	Click the cancel button on the first booking room form	The booking room form closes without error returning the user to the calendar view page	The booking room form closes without error returning the user to the calendar view page	Pass
T15	Clicking "book a room" on this form transfers user to "booking form"	None	Click the book room button	The next booking room form containing input fields is shown	The next booking room form containing input fields is shown	Pass

T16	Clicking “cancel” returns user to calendar view	None	Click the cancel button on the second booking form	The booking room form is cancelled without error returning the user to the main calendar page	The booking room form is cancelled without error returning the user to the main calendar page	Pass
T17	Clicking “book a room” with no data results in error	None	Click the book room button ensuring all input fields are clear	An error message is displayed in a form	An error message is displayed in a form	Pass
T18	When “Booking name” input is selected, on screen keyboard shows	None	Click the input field for booking name	An onscreen keyboard appears which allows the user to input their name	An onscreen keyboard appears which allows the user to input their name	Pass
T19	Users can type their name in using the on screen keyboard	“Client Test”	Type in a name (mock name given)	The user can use the onscreen keyboard to enter their information	The user can use the onscreen keyboard to enter their information	Pass
T20	Keyboard can be “cancelled” by clicking anywhere outside the keyboard	None	When the keyboard is present, click anywhere outside of the keyboard area	The keyboard is closed without affecting the input field	The keyboard is closed without affecting the input field	Pass
T21	Clicking date input displays the datetime picker	None	Click the date input field	A datepicker is shown which allows users to search and select dates	A datepicker is shown which allows users to search and select dates	Pass
T22	Datetime picker allows for users to change months	None	Change the month displayed by clicking the arrows located at either side of the header bar	The month is changed and the dates are updated accordingly	The month is changed and the dates are updated accordingly	Pass
T23	Datetime picker allows for users to select any date	“13/04/2012” (2012-04-13)	Select the date given in the input data	The correct date can be selected and is displayed	The correct date can be selected and is displayed	Pass
T24	Datetime is displayed in the input container in a legible format	None	None	The date is displayed in an understandable format	The date is displayed in an understandable format	Pass
T25	Datetime picker can be closed by clicking anywhere outside datepicker	None	When the datetime picker is displayed, click anywhere outside the datetime picker area	The datetime picker is closed without error or affecting the input field	The datetime picker is closed without error or affecting the input field	Pass
T26	Clicking booking start input field displays a timepicker	None	Click the booking start input field	A timepicker is shown which allows users to input a time value	A timepicker is shown which allows users to input a time value	Pass
T27	Clicking booking end input field displays a timepicker	None	Click the booking start input field	A timepicker is shown which allows users to input a time value	A timepicker is shown which allows users to input a time value	Pass
T28	Timepicker allows users to select an hour value and a minute value	Booking Start- “10:00” Booking End- “11:30”	Input the booking start and booking end values as defined in the input data column using the timepicker	The correct dates are input into the respective input fields and is displayed correctly and in an understandable format	The correct dates are input into the respective input fields and is displayed correctly and in an understandable format	Pass
T29	Selected timepicker value is displayed in the input field in a legible format	None	None	The timepicker value is displayed in an understandable format	The timepicker value is displayed in an understandable format	Pass
T30	Timepicker can be closed by clicking anywhere outside timepicker	None	When the timepicker is displayed, click anywhere outside the datetime picker area	The datetime picker is closed without error or affecting the input field	The datetime picker is closed without error or affecting the input field	Pass

T31	When booking form is populated, user can book a room and is alerted of bookings status	Booking Name- “Client Test” Date- “2102-04-13” Booking Start- “10:00” Booking End- “11:30”	Populate the booking form with the values given in the input data column and click book room	The booking is submitted successfully and a “success” dialog is displayed	The booking is submitted successfully and a “success” dialog is displayed	Pass
T32	When booking is resubmitted, user is alerted of duplicate booking by way of an error.	Booking Name- “Client Test” Date- “2102-04-13” Booking Start- “10:00” Booking End- “11:30”	Populate the booking form with the values given in the input data column and click book room	The booking is submitted successfully but an “error” dialog is displayed	The booking is submitted successfully but an “error” dialog is displayed	Pass

The following table depicts the system tests in relation to the acceptance criteria defined in the user stories which were implemented. Each of the system test cases are plotted in relation to the acceptance criteria it fulfilled.

Booking a Room

Name	**Description**	**Test Cases**
Acceptance Criteria 1	Does the system store a record of all bookings made?	(untestable in system tests)
Acceptance Criteria 2	Does the system confirm if a room has been booked successfully?	T17, T31, T32
Acceptance Criteria 3	Does the system alert me if a booking already exists on the time I want to book and denies me the ability to book?	T32

Calendar View

Name	**Description**	**Test Cases**
Acceptance Criteria 1	Can I view a graphic of all rooms and click specific rooms?	T4, T8
Acceptance Criteria 2	Can I view a calendar view of any room with all bookings attached to that room?	T8, T12

Confirmation

Name	**Description**	**Test Cases**
Acceptance Criteria 1	Does the system show me if the booking was successful?	T30
Acceptance Criteria 2	Does the system show me if my booking was denied?	T31, T17

Appendix I – HCI Questionnaire Template

HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

When viewing available rooms, would you rather view rooms as a “list”, or view a “map” of available rooms? (View screenshots)

Does the ability to view a map of available room’s aid in the booking of a room?

Does the ability to view a calendar which indicates the room’s availability aid in the booking of a room?

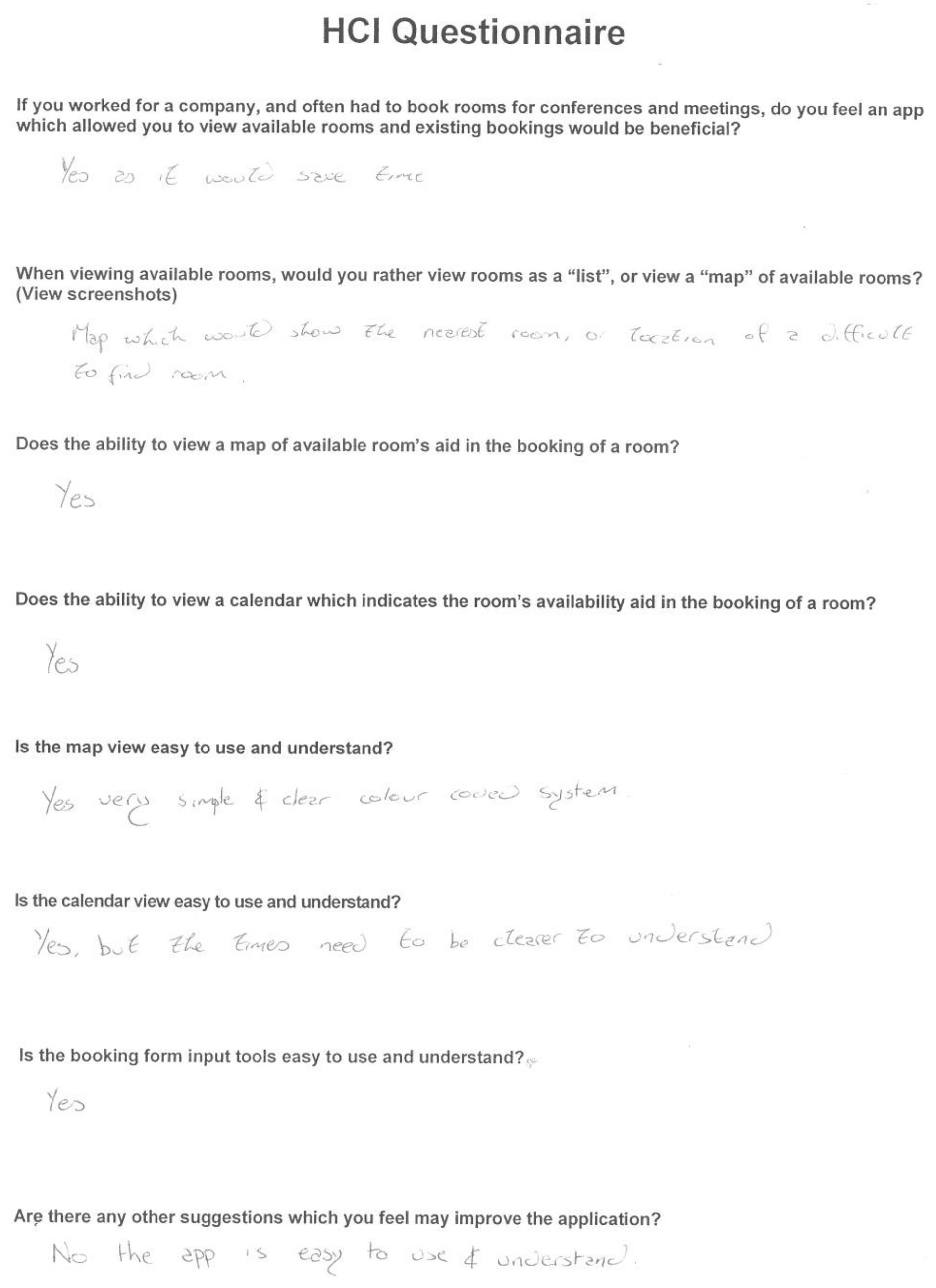
Is the map view easy to use and understand?

Is the calendar view easy to use and understand?

Is the booking form input tools easy to use and understand?

Are there any other suggestions which you feel may improve the application?

Appendix J – HCI Questionnaire Results



HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

Yes as it would save time

When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms? (View screenshots)

Map which would show the nearest room, or location of a difficult to find room.

Does the ability to view a map of available room's aid in the booking of a room?

Yes

Does the ability to view a calendar which indicates the room's availability aid in the booking of a room?

Yes

Is the map view easy to use and understand?

Yes very simple & clear colour coded system.

Is the calendar view easy to use and understand?

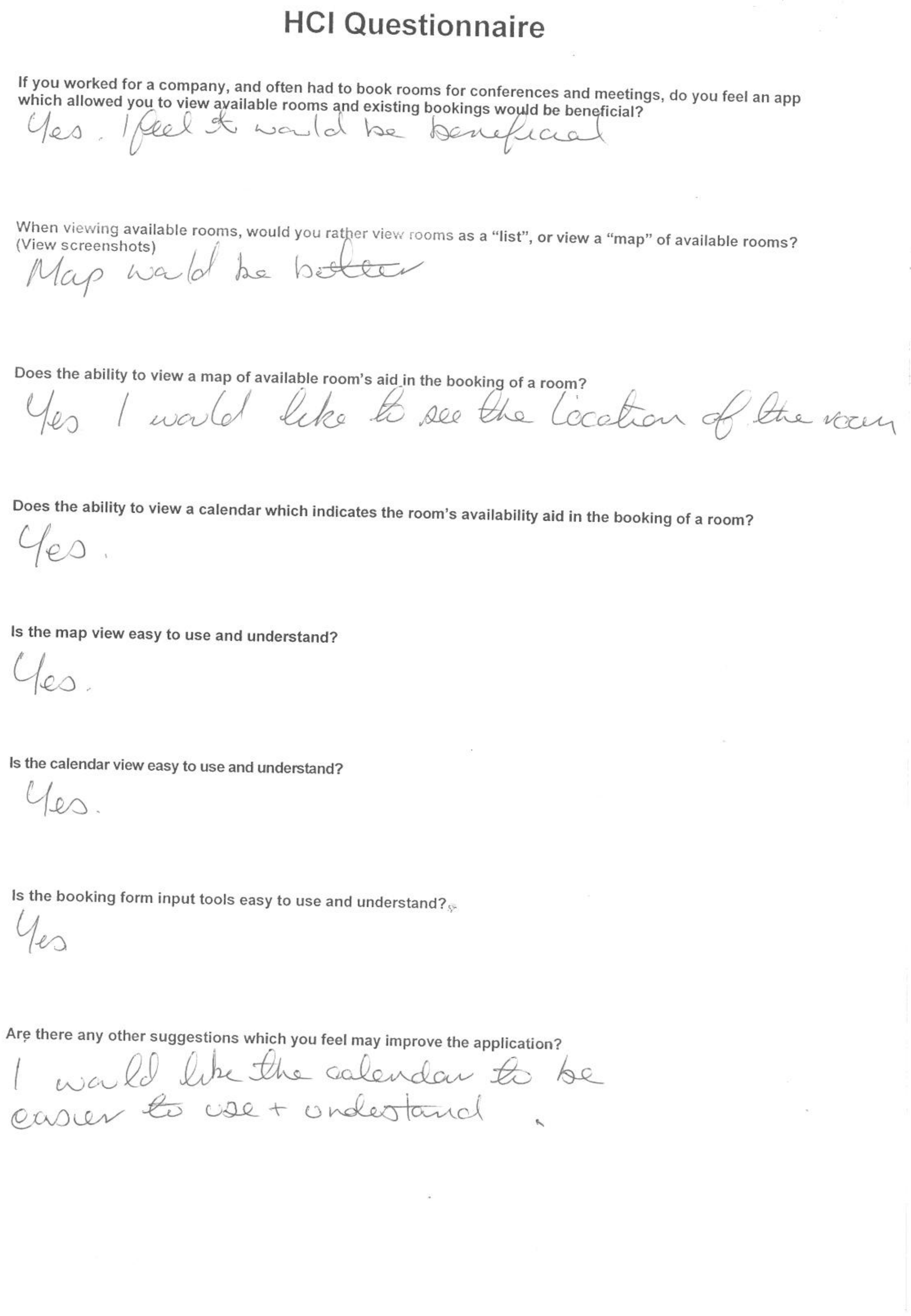
Yes, but the times need to be clearer to understand

Is the booking form input tools easy to use and understand?

Yes

Are there any other suggestions which you feel may improve the application?

No the app is easy to use & understand.



HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

Yes. I feel it would be beneficial

When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms? (View screenshots)

Map would be better

Does the ability to view a map of available room's aid in the booking of a room?

Yes I would like to see the location of the room

Does the ability to view a calendar which indicates the room's availability aid in the booking of a room?

Yes.

Is the map view easy to use and understand?

Yes.

Is the calendar view easy to use and understand?

Yes.

Is the booking form input tools easy to use and understand?

Yes

Are there any other suggestions which you feel may improve the application?

I would like the calendar to be easier to use + understand

HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

YES

When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms? (View screenshots)

MAP

Does the ability to view a map of available room's aid in the booking of a room?

YES

Does the ability to view a calendar which indicates the room's availability aid in the booking of a room?

YES

Is the map view easy to use and understand?

YES

Is the calendar view easy to use and understand?

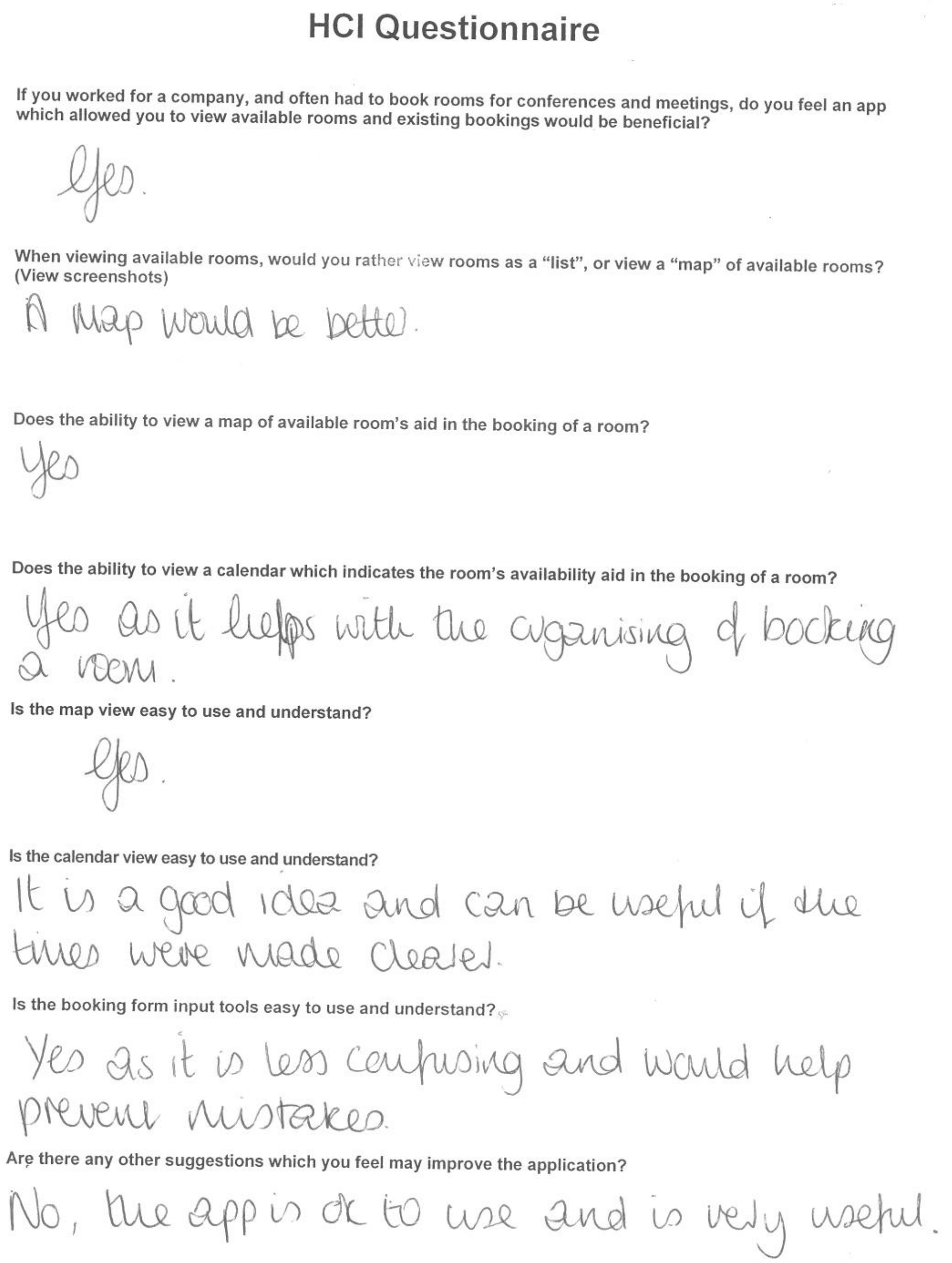
YES

Is the booking form input tools easy to use and understand?

YES

Are there any other suggestions which you feel may improve the application?

CLEARER TIMES



HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

Yes.

When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms? (View screenshots)

A map would be better.

Does the ability to view a map of available room's aid in the booking of a room?

Yes

Does the ability to view a calendar which indicates the room's availability aid in the booking of a room?

Yes as it helps with the organising of booking a room.

Is the map view easy to use and understand?

Yes.

Is the calendar view easy to use and understand?

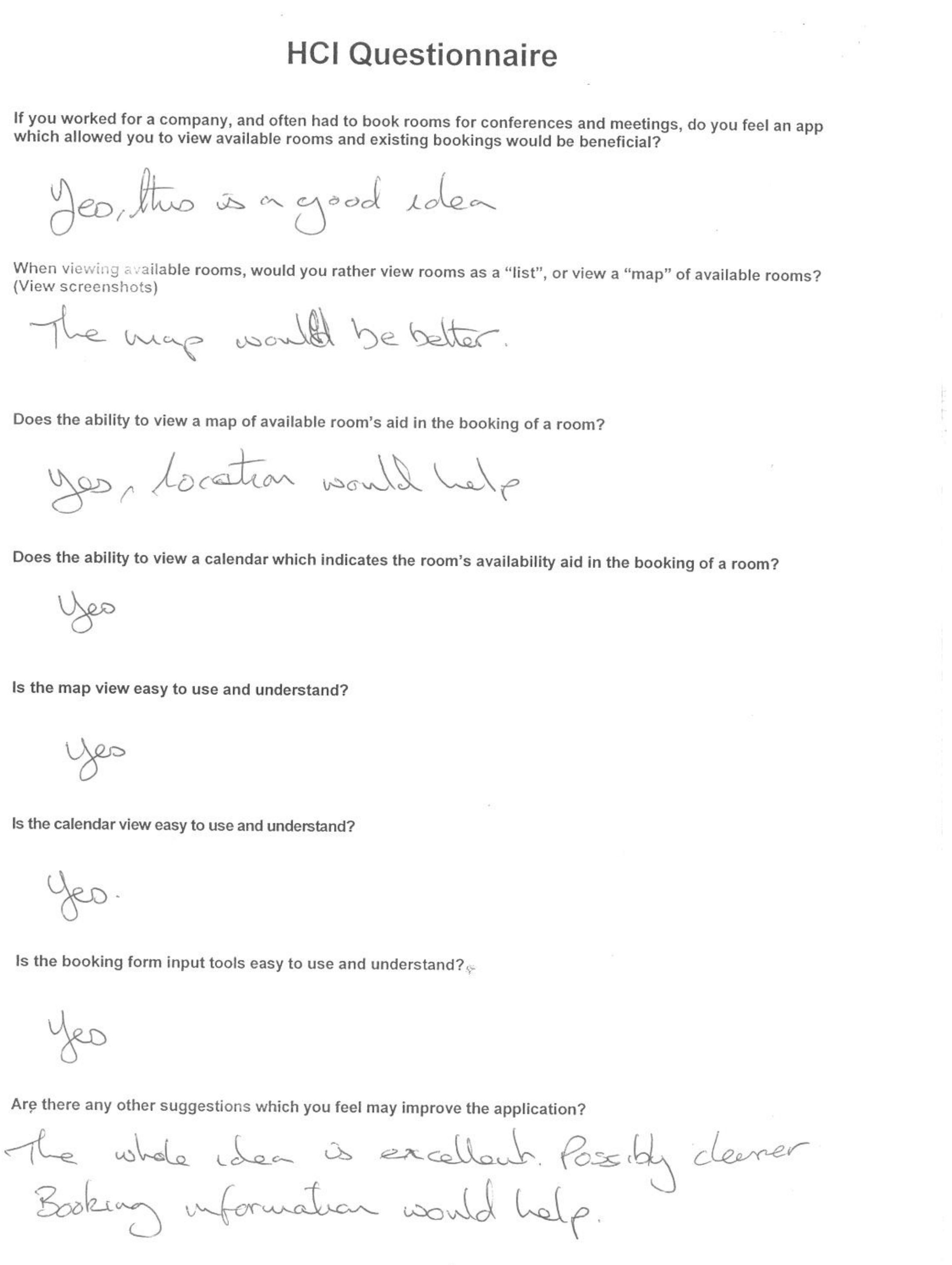
It is a good idea and can be useful if the times were made clearer.

Is the booking form input tools easy to use and understand?

Yes as it is less confusing and would help prevent mistakes.

Are there any other suggestions which you feel may improve the application?

No, the app is ok to use and is very useful.



HCI Questionnaire

If you worked for a company, and often had to book rooms for conferences and meetings, do you feel an app which allowed you to view available rooms and existing bookings would be beneficial?

Yes, this is a good idea

When viewing available rooms, would you rather view rooms as a "list", or view a "map" of available rooms? (View screenshots)

The map would be better.

Does the ability to view a map of available room's aid in the booking of a room?

Yes, location would help

Does the ability to view a calendar which indicates the room's availability aid in the booking of a room?

Yes

Is the map view easy to use and understand?

Yes

Is the calendar view easy to use and understand?

Yes.

Is the booking form input tools easy to use and understand?

Yes

Are there any other suggestions which you feel may improve the application?

The whole idea is excellent. Possibly clearer Booking information would help.