

UNIVERSITY OF LEEDS

FINAL YEAR PROJECT

Mid Project Report

User Adaptive System to Support Automated Medical Appointment Scheduling of Outpatients

Author:

Andrew MUNRO

Supervisor:

Dr. Raymond KWAN

*A final year project report submitted in fulfilment of the requirements
for the degree of Computer Science*

in the

School of Computing

May 2014

UNIVERSITY OF LEEDS

Abstract

Faculty of Engineering

School of Computing

Computer Science

User Adaptive System to Support Automated Medical Appointment Scheduling of Outpatients

by Andrew MUNRO

This project aims to allow more communication with outpatients in regards to their appointments. It also aims to tackle various problems such as attendance, human opportunity costs and patient satisfaction. This will be achieved using emerging technology to bridge the current communication gap between patients and the scheduling systems used.

Contents

Abstract	ii
Contents	iii
List of Figures	vii
Abbreviations	viii
1 Project Overview	1
1.1 Project Aim	1
1.2 Objectives	1
1.3 Minimum Requirements	2
1.4 Extensions	2
1.5 Deliverables	3
1.6 Relevance to Degree Modules	3
2 Project Plan	4
2.1 Schedule	4
2.2 Background Research	4
2.3 Design	4
2.3.1 Methodologies	4
2.3.2 Flexibility	6
2.3.3 Scalability	6
2.3.4 Problems	6
2.4 Implementation	7
2.5 Evaluation	7
2.6 Conclusion	7
3 Problem Description and Background Research	8
3.1 Problem Description	8
3.2 Wasted Resources	8
3.2.1 Patient No-Shows	9
3.2.2 Cancellations	10
3.2.3 Managing Appointments	11
3.3 Patient Satisfaction and Experience	12
3.3.1 Patient Requirements	12
3.3.2 Waiting times	13
3.3.3 Patient Participation	13
3.4 Choose and Book - An existing online medical appointment service	14
3.5 Conclusion	14
4 System Design and Development	16
4.1 Introduction	16
4.2 User Requirements	16
4.2.1 Login System	16

4.2.2	List of Appointments	16
4.2.3	Appointment Information	17
4.2.4	Map	17
4.2.5	Scheduling Appointments	17
4.2.6	Calendar Synchronisation	18
4.2.7	Appointment Notifications	18
4.2.8	List of Notifications	18
4.3	System Requirements	18
4.3.1	Cross-Platform	19
4.3.2	Representational State Transfer (Restful) Web Services	19
4.3.3	One Development Language	19
4.4	Development and Technology Used	19
4.4.1	Mono and Xamarin	19
4.4.2	Communication	20
4.4.2.1	Push Communication	21
4.4.2.2	Direct Communication	22
4.4.2.3	Asp.Net	23
4.4.3	Data Storage	23
4.4.3.1	Entity Framework	24
4.4.4	Hosting	24
5	Implementation	25
5.1	Introduction	25
5.1.1	Architecture and Design Patterns	25
5.1.1.1	Model-View-Controller (MVC)	25
5.1.1.2	Component Pattern	27
5.2	Project Setup	27
5.2.1	Emulating Requests	27
5.2.2	Organising Code	28
5.3	Server Structure	29
5.3.1	Authentication	29
5.3.2	Routing	30
5.3.3	Request Methods	30
5.3.4	Data Storage and Retrieval	31
5.3.4.1	Data Models	31
5.3.4.2	Data Context	32
5.3.4.3	LINQ	32
5.3.4.4	Migrations	33
5.4	Client Structure	35
5.4.1	Client Core Library	35
5.4.1.1	Creating Requests	35
5.4.1.2	Request Response	36
5.4.1.3	Asynchronous Requests	36
5.4.2	Android Application	37
5.4.2.1	Activities	37
5.4.2.2	Intents	37
5.4.2.3	Resources	37

5.4.2.4	Layouts	38
5.4.2.5	Services	38
5.5	Login and Authentication Feature	38
5.5.1	Login Request	40
5.5.2	Registration Request	40
5.5.3	Incorrect Input	40
5.5.4	Conclusion	41
5.6	Home Screen Feature	41
5.6.1	Tab Activity	41
5.6.2	Fragments	41
5.6.3	List Fragments	42
5.6.4	Retrieving Appointments	42
5.6.4.1	Server Side Request	42
5.6.4.2	Appointment Model	42
5.6.5	Action Bar	43
5.7	Appointment Information Feature	43
5.7.1	Doctor's Image	44
5.7.2	Buttons	44
5.7.3	Action Bar	44
5.7.4	Dialer	45
5.8	Calendar Feature	45
5.8.1	Calendar Querying	45
5.8.2	Issues	46
5.9	Location and Map Feature	46
5.9.1	Google Maps API	46
5.9.2	Placing a marker	47
5.10	Appointment Scheduling Feature	47
5.10.1	Date Picker	47
5.10.2	Example Scheduler	47
5.10.3	Confirming the Appointment	48
5.10.4	Time issue	48
5.11	Notification Feature	49
5.11.1	GCM Service	49
5.11.2	Sending of GCM messages	50
5.11.3	Observer	50
6	Testing and Evaluation	51
6.1	Introduction	51
6.2	Testing	51
6.2.1	Unit Testing	51
6.2.2	Functional Testing	51
6.2.2.1	Testing Methodology	52
6.2.2.2	Results	52
6.2.3	Further Testing	52
6.2.3.1	Load testing	53
6.2.3.2	Concurrency Issues	53
6.2.4	Conclusion	54

6.3	Evaluation	54
6.3.1	Evaluation of Methodology	54
6.3.2	Controlled Trials	55
6.3.3	User Observation	55
6.3.4	User Questionnaire	56
6.3.5	Further Evaluation	57
6.3.5.1	Training	57
6.3.5.2	Comparison with Phone Appointment Booking	57
6.3.6	conclusion	57
7	Project Conclusion	59
7.1	Summary	59
7.2	Limitations	59
7.3	Future Extensions	59
7.4	Final Conclusion	59
A	Personal Reflection	60
B	External Resources and Materials	61
C	How ethical issues are addressed	62
C.1	Introduction	62
C.2	Project background	62
C.2.1	Personal Data	62
C.2.2	Employee Downsizing	62
C.3	Testing and Evaluation	63
C.3.1	User Based Assessments	63
D	Regression Test	64
E	User Evaluation	65
E.1	Introduction	65
E.2	Task List	65
F	User Evaluation Questionnaire	66
G	Project Resources	69
G.1	Code and Mobile Application Download	69
G.2	Mobile Application Screens	69
	Bibliography	73

List of Figures

2.1	Gantt Chart showing the schedule of the project	5
2.2	Table showing the schedule of the project	6
3.1	Factors contributing to non-attendance according to a patient questionnaire - [2] .	9
4.1	Diagram showing client architecture and business logic being reused across multiple platforms	20
4.2	Google Cloud Messaging - [15]	22
4.3	RESTful Web Services cross-platform communication - [16]	23
4.4	Example of an Asp.Net web service definition	24
5.1	Model View Controller Diagram - [17]	26
5.2	Screenshot showing the use of Postman	28
5.3	Example of a controller code showing authorised and routed requests	30
5.4	Example of a binding model associated with the register account request	31
5.5	Example of an Entity Framework Data Model	33
5.6	The Final Entity Relationship Diagram	34
5.7	Example of the Entity Framework Data Context	34
5.8	Creation of a client request	36
5.9	Example of the client sending a request	36
5.10	Screenshot showing the use of the Xamarin Layout Designer	39
6.1	Graph to show the pass rate of each feature category.	53
6.2	Graph to Show the Average User Ratings Collected from the User Questionnaire	56
G.1	Login Screen (Left), Home Screen (Right)	70
G.2	Appointment Information Screen (Left), Appointment Time Picker Screen (Right)	71
G.3	Conflicting Appointment Screen (Left), Confirm Appointment Time Screen (Right)	72

Abbreviations

NHS	N ational H ealth S ervice
APP	Software APP lication
API	A lication P rogramming I nterface
MVC	M odel V iew C ontroller
XML	E xtensible M arkup L anguage
JSON	J ava S cript O bject N otation language
SQL	S tructured Q uery L anguage
LINQ	L anguage I Ntegrated Q uery
ER	E ntity R elationship
GCM	G oogle C loud M essaging

Chapter 1: Project Overview

1.1 Project Aim

The aim of this project is to develop a system that allows more communication with outpatients in regards to their appointments, in an attempt to:

- reduce the amount of resources wasted in the event of no-shows and cancellations
- reducing the human cost that is required to manage appointments
- increase the user experience when making and managing an appointment

This project will not aim to implement any scheduling algorithms, but rather create a system that will support current algorithms and possibly shape future algorithms. For this project, I will use a preordained scenario to show off the different features of the system.

It will also not aim to replace the current scheduling system entirely, but act as an alternative to allow willing patients more direct control and easier access to information regarding their appointments.

1.2 Objectives

The objectives of this project is to:

- Collect relevant background data about the problem domain
- Identify requirements necessary to address the problem
- Design a server-client system that implements these requirements
- Prototype a Server that communicates directly with multiple patients and a predetermined scheduling algorithm
- Prototype a Client Application (Smart-phone Application) that allows patients to interface with the server
- Test the systems functionality based on usability and performance

- Evaluate the success of the system in regards to improving user satisfaction, reducing human management resources and reducing appointment wastage

1.3 Minimum Requirements

The minimum requirements of this project is to:

- A working prototype smart-phone application that:
 - Connects to a prototype server
 - Allows the user to view information on their scheduled medical appointments
 - Gives the user information such as location and map instructions, doctor's name, any prerequisite tasks the user must undertake prior to the appointment
 - Reminds the user about the appointment
 - Receives appointment updates from the server
- A working prototype server application that:
 - Connects to multiple clients
 - Interfaces with a dumb appointment scheduler
 - Notifies clients of changes to the schedule
 - Uses client information to optimise the scheduling process
 - Offers cancelled appointments to other clients

1.4 Extensions

The possible extensions are:

- Design a website interface for the system
- Investigate and test security issues
- App gives location information and integrates with google maps
- Additional App or website to be used by doctors

1.5 Deliverables

The deliverables are:

- Server application
- Client application
- Report and evaluation results

1.6 Relevance to Degree Modules

This project uses knowledge and techniques gained in modules studied as part of my Computer Science degree. The most relevant modules are typically 'Software Systems Engineering', 'Distributed Systems', 'User Adaptive Systems' and 'Human Computer Interaction'.

Chapter 2: Project Plan

2.1 Schedule

The schedule was designed as a Gantt chart, in order to highlight key points in the project and ensure that they were completed on time and according to the set deadlines. This schedule is subject to change as the project progresses, as many unforeseen issues may arise. I have tried to estimate tasks based on difficulty, assigning more time to key points where it is likely that problems will occur (such as the implementation phase).

A table was also created to compare objectives with deliverable deadlines.

The project has been split four main components, consisting of research, design, implementation and evaluation. These areas are then sub-divided into appropriate sections, targeting individual parts of the system, individual objectives and individual deliverables.

The following sections will explain these components, what parts of the project they will address and how they will be executed successfully.

2.2 Background Research

The research component will aim to get a better understanding of the problem, how I can address my aims and objectives and highlighting the current research in the field. This research will aid my own project by highlighting areas that need to be addressed, outlining unforeseen problems and finding technologies that I might be useful to my implementation.

2.3 Design

The design component will aim to design the entire system, addressing all objectives whilst making the system both flexible and scalable.

2.3.1 Methodologies

I will choose a software engineering methodology that best fits the development style of the project, explaining how it works and how it is executed successfully. I will also identify the programming languages I will be using, any programming patterns that are useful in the design of the system and Technologies I will use.

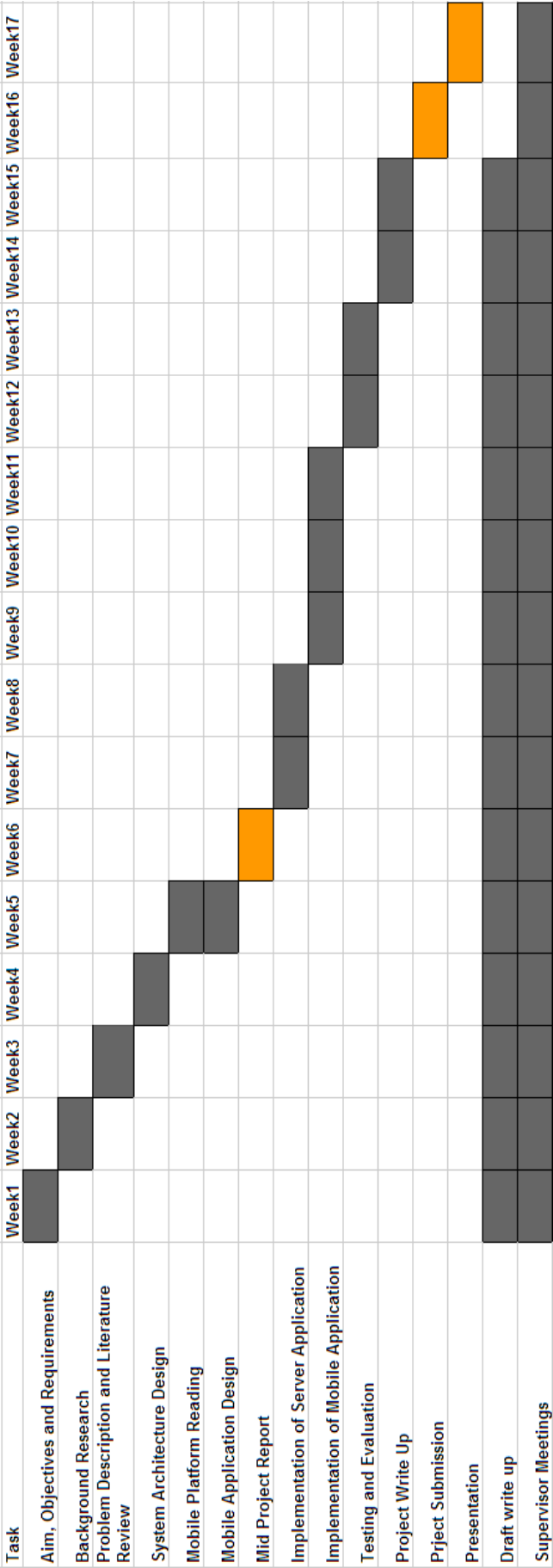


FIGURE 2.1: Gantt Chart showing the schedule of the project

Week	Activity (Start of Week)	Deliverables (Start of Week)
1	Investigating the problem	
2	Background Research	
3	Problem Description and Literature Review	Aim, Objectives and Requirements
4	System Architecture Design	Background Research Summary
5	Mobile Platform Reading	Design of System
6	Mobile Application Design	Summary of Mobile Technologies
7	Mid Project Report	App Wireframes
8		Mid Project Report
9	Implementation of Server Application	
10	Implementation of Mobile Application	Prototype Server
11		Prototype Mobile Application
12	Testing and Evaluation	Testing Designs
13		Testing Results
14	Project Write Up	Draft Project
15		Final Project
16		
17	Presentation	Presentation

FIGURE 2.2: Table showing the schedule of the project

2.3.2 Flexibility

Although the system is a prototype, I aim to create it such that it can be adapted in the future. The system will therefore be designed in a way such that additional platforms can be added easily. The system should also be designed in a way that it is feature independent, eliminating dependencies in the code. This will allow more features to be added easily in the future.

2.3.3 Scalability

The system will also be designed to be scalable. It must be able to scale with user demand, be fault tolerant and appear seamless to the end user, allowing for a smooth service at all times. I will talk about the various problems involved and how I will design the system to overcome them.

2.3.4 Problems

Many problems must be overcome in the implementation phase, such as data security, synchronising data and many more. These problems must be planned in advance, and so I will discuss in detail the problems that the system faces and my proposed solutions to them.

2.4 Implementation

The implementation of the system will occur in two separate platforms, the client and the server. These will sometimes overlap as some code will be shared between them, however for the most part, I will discuss issues unique to each platform separately.

For the server platform, I will discuss how I implemented the communication and data storage features and the overall structure of the server system.

For the client platform, I will discuss how the user interface is created, how it communicates with the server and any issues that I encountered.

2.5 Evaluation

The evaluation component will show how well my prototype system does in solving my aims and objectives identified in the project. I will analyse the results of user evaluation methods to evaluate the prototype, discussing the results and identifying key parts of my solution that need to be improved. I plan to evaluate on the following questions:

- How easy to use is the mobile application?
- Does the solution work? Does it fail occasionally?
- Is the solution missing key features?
- Would patients find the mobile application useful?
- Does the solution reduce wasted appointments?
- Does the solution reduce staff resources required for managing appointments?
- Does the solution increase the user experience when managing appointments?

2.6 Conclusion

From this evaluation, I will formulate a conclusion, reviewing my design choices, how accurate my expectations were and discussing future extensions that this project could inspire.

Chapter 3: Problem Description and Background Research

3.1 Problem Description

Medical appointment scheduling is a complex problem; patients often come with different backgrounds and personal schedules, requiring different treatment and different urgency, some even requiring support in getting to the appointments. Patients sometimes have a need to cancel their appointments or simply do not turn up, which can lead to a waste in resources if the appointment slot is not then assigned to another patient. Often, sessions can overrun, requiring more time per patient than is estimated, and so the following appointments are delayed. Clinics also reschedule appointments regularly, as new patients requiring urgent medical attention become a higher priority. This results in appointments being dynamic, often the time and date of the actual appointment is different from what was originally planned.

Dynamic scheduling leads to many issues. The problem of scheduling appointments becomes far more complex, which in turn requires more staff resources to manage the appointments.

Communication also becomes a problem, as patients need to be informed about all changes to the original schedule. Often, this results in a lower patient satisfaction and a higher chance of appointment cancellations.

These often contribute to longer waiting times and a lack of patient knowledge about their appointments, which can make no-shows and further last-minute cancellations more frequent. Often the clinic will not find a patient to take the free appointment slot, and these resources are wasted. In 2012, the Department of health announced that 1 in 10 NHS appointments are unfilled, costing the taxpayer up to £600 million a year.[?]

3.2 Wasted Resources

Many resources are wasted through appointment no-shows and cancellations. Research shows that the longer a patient must wait between making the appointment and the actual appointment date, the more likely it is that they will either cancel or not turn up[1]. Although we can see that

there is a relationship between the length of time that a patient must wait for an appointment and the cancellation risk, it is important to understand why.

The most common reasons why patients do not show up was collected through patient questionnaires as can be seen in the figure below.

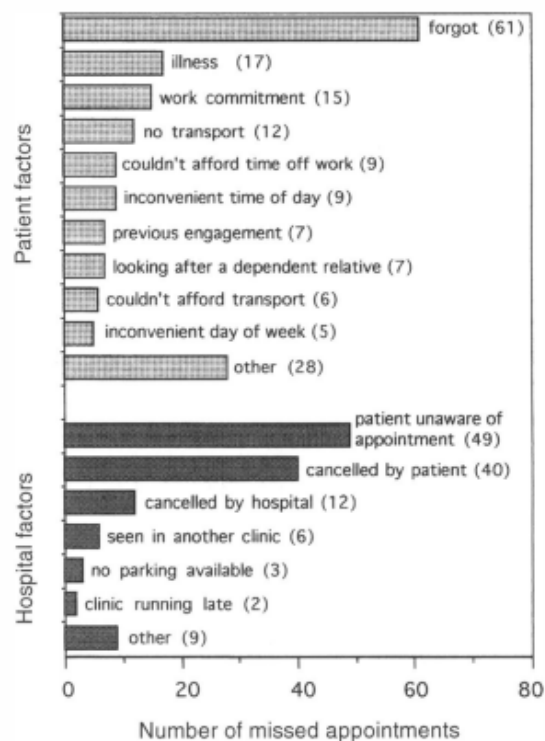


FIGURE 3.1: Factors contributing to non-attendance according to a patient questionnaire - [2]

This shows that the most common factor is either that the patient forgot or that they were improperly informed by the clinic.

3.2.1 Patient No-Shows

Patient no-shows (or non-attendance at outpatient clinics) are a big problem in appointment scheduling and is one of the largest contributors to wasted resources in the NHS. It is estimated that the financial cost of missed appointments contributes to a loss of £360 million per year[2]. Besides the financial costs, it also increases waiting times as that patient must be rescheduled for another appointment, effectively doubling the required resources per patient.

Because the most common factor contributing to no-shows is the patient forgetting to attend the appointment, it suggests a reminder system could be used effectively to reduce these numbers. Research has shown that telephone and postal reminders can help, but have not proved to be

cost effective in the past[3]. However, through recently emerging online devices and 'smart' technology, it is possible to provide a low-cost solution to this problem.

Another factor that can encourage non-attendance can be a lack of information known about the appointment. This can introduce a level of uncertainty within the patient, such as knowledge on how to get to their appointment, or fear about the dangerous/embarrassing factors involved in the appointment[4].

Research shows that the population that miss appointments are increasingly of a young demographic. Often they fail to understand why the appointments are important, and specifically why it is important to cancel the appointment rather than just not turn up.

Patients have given several reasons for no-shows in studies and questionnaires[5]:

- can't get time off work
- child-care
- lack of transportation or cost
- patient felt better or felt too worse to attend the appointment

For all these reasons, the no-show is preventable simply by increasing communication with the patient and allowing them either more information about the appointment or making it easier for them to either cancel/reschedule.

3.2.2 Cancellations

Often, clinics are required to cancel an outpatients appointment for a variety of reasons, usually due to a lack of resources such as staff or equipment.

Staff time is then wasted getting contact information for the patient and informing them of the cancellation. Time must also be spent corresponding with the patient and agreeing on a suitable replacement appointment.

If cancellations 'occur at the last minute', often the patient is not informed until they reach the hospital. This is often due to the clinic not wanting to bother wasting resources reaching the patient when it is unlikely that they will get a hold of them (the patient may already be in transit or indisposed). This leads to a lower overall user satisfaction as the patient wastes a trip to the hospital, only to find that they no longer have an appointment.

Although cancellations are not ideal, they are better than no-shows in that it is possible for the appointment to be offered to another patient. In some cases however, this is not attempted due to it being too costly and the additional complexity involved in the scheduling process.

By increasing communication with patients through emerging smart technology, it may be possible for these appointments to be quickly rescheduled whilst avoiding the additional costs, even for 'last-minute' cancellations.

3.2.3 Managing Appointments

Managing appointments is costly and a large proportion of the work is carried out by individual staff members, rather than an automated system. Several issues arise with this process:

- Appointments can only be managed within office hours (typically 9am - 5pm), however this depends on the clinic
- Patients can only make appointments over the phone and frequently have to queue to speak to a staff member
- Large proportions of staff members must be allocated to the appointments procedure which could be allocated elsewhere
- State of the system means that it is hard to analyse and therefore adapt to high demand

When making appointments, the patient is required to do so either in person or over the phone. This can only occur within office hours, which can conflict with the patients career or personal schedule, leading to a lower user satisfaction. Often patients will have to queue in order to speak to a staff member. When the patient is finally connected, there isn't time for the patient to explain their personal schedule and discuss conflicts, resulting in less user choice and flexibility.

The appointment system in many areas is also very labour-intensive, carried out by individual receptionists using spreadsheets and paper based systems[6]. This means that it is often very hard to analyse capacity and demand, identifying bottlenecks or methods to improve them and also makes it very hard to integrate with interactive technology such as smart phone devices and online appointments. Whilst some online systems do exist[7], they are time consuming, have poor functionality and tend to be only available on few devices[8].

These problems can be improved by creating an interactive online system that works on many devices. It could not replace the current system entirely, because not all patients will have

internet access or smart-devices, but it would provide benefits to patients with internet access such as:

- Easier accessibility to making appointments
- Possibly offer more appointment flexibility to the patient
- More information about the appointment
- Relieve demand on the staff that manage the system
- More analysis of appointment trends and offer insights into improvement

3.3 Patient Satisfaction and Experience

Maintaining a high patient satisfaction is the primary goal in appointment scheduling, ultimately because keeping the patients happy leads to less cancellations and less no-shows. This is not an easy task because the demand on the healthcare system is so great, and it is typically faced with many challenges.

3.3.1 Patient Requirements

Patients are given a level of responsibility that some may not be used to. For a general outpatient appointment, the NHS requires the patient to do a number of tasks to prepare for the appointment [9]:

- may be required not to eat/drink before the appointment
- may be required to bring samples of urine/stool or medicines
- may need to bring previous test results
- may need to take certain medicines at a certain time period prior to the appointment
- should bring maps and other information required for getting to their appointment

It has been seen that in previous research conducted on day surgery outpatients, the most likely cause of preventable appointment cancellations (5% of day surgery appointments) was due to inadequate preparation [10]. This shows that a large amount of patient cancellations occur simply because patients are expected to find out information about their appointment, transport options and other relevant factors.

3.3.2 Waiting times

Another factor that lowers patient satisfaction are waiting times that can occur when the schedule is either running late due to overrunning appointments, or when patients are grouped into time slots.

Patients are frequently grouped together into time slots to simplify the scheduling process (i.e the clinic will expect to have 10 appointments in one hour, so they ask all 10 patients to come at the same time and the appointments occur on a first come first serve basis).

Research suggests that because patients spend increasingly lengthy amounts of time waiting in the clinic for their appointment to start, they feel increasingly amounts of disrespect [5]. This is due to patients being 'left in the dark', with no indication on why their appointment is delayed and why they have to wait.

Through on-line applications and smart devices, we can inform patients about information related to their appointments, any disruptions in the regular service (waiting delays) and a more interactive system that would make the patient feel less disrespect. We will also be able to offer sooner appointments to patients as cancellations occur, which should reduce the waiting times overall.

We can also provide transport information, reminders on when they have to leave and any prerequisite requirements that the patient must undertake before leaving for their appointment (such as take medication or bring test results), reducing the likelihood of cancellations and no-shows as the patient is better prepared..

3.3.3 Patient Participation

Patient participation is no longer just a goal set by medical commissioners, but a legal obligation. The Health and Social Care Act 2012[11] introduced legislation that enables patients(and carers) to participate in the planning, managing and making decisions about their care and treatment.

The aim of this project targets this participation, engaging patients to have more control and access over their medical care. This system also has the ability to deliver personalised care plans to patients, which will increase the overall patient satisfaction.

Although this system does require patients to have access to the internet and know how to use it, patients without internet in today's world is an increasingly small demographic. The NHS

are also launching a program to help disadvantaged people learn how to access the internet and use medical services [12], to try and combat these issues.

3.4 Choose and Book - An existing online medical appointment service

An NHS service 'Choose and Book' was launched in 2006, aimed at providing patients with more choice through online appointments[13].

This system is similar to the project area in that it allows patients to create online outpatient appointments, having a choice over which clinic they go to and when they the appointment is booked for.

However, an independent survey of patient's experience using the service in 2008 showed that patients did not receive the degree of choice that the service was designed to deliver[14]. It has also been widely criticised as being time consuming, over complicated and . An article in 2012[8] shows that the system's popularity is diminishing.

Besides the clear flaws in the system such as ease of use and failing to offer more choice, it also fails to target significant areas of the problem description, such as electronic reminders, recycling unused appointments and general appointment information.

3.5 Conclusion

Attempts have been made in the past to simplify the appointment management process and take it online, however they fail to hit all of the objectives simply because the platforms were not ready. As smart-devices and their many applications are becoming increasingly popular, it opens up a new gateway to communicate directly with patients and receive quick response times. This makes it much easier to create a dynamic appointment schedule whilst maintaining a high user satisfaction level.

This project will therefore focus on improving the communication and interactivity between patients and the appointment scheduling system; so that more information is available to the patient, there is more chance of reusing free appointment slots, and less staff resources are used in managing appointments so they can be allocated to other areas.

The project will also look at making the appointment creation and rescheduling process easier for both patients and staff, requiring less management resources and offering more platform choices and flexibility.

Chapter 4: System Design and Development

4.1 Introduction

This chapter will focus on the requirements that the proposed solution must fulfil, outline the challenges involved and highlight the key technologies that will be used.

4.2 User Requirements

With the research showing areas that could be focused on to improve user satisfaction, a list of proposed user requirements was created to identify specific features that would improve the user experience. The following list shows the final user requirements and features for the proposed solution:

4.2.1 Login System

A login system used to hold the patients identity and allow them to easily manage their appointments on multiple devices. This will also act as a security mechanism to ensure that medical information is only revealed to the patient in question.

The ability to register a new account will also be available using the app, allowing patients to sign up easily.

4.2.2 List of Appointments

A list of the patients appointments which is kept up to date automatically when changes are made.

This will display very basic information about each appointment, such as when it occurs and the type of appointment it is. An image will also be available per appointment that corresponds to the clinic where that appointment occurs (a logo). This will act as a quick way that the patient can remember where the appointment is located without having to use the app further.

This information will all be retrieved securely from the server using the patients login credentials.

4.2.3 Appointment Information

Tapping on an appointment in the appointment list will bring up relevant information about that appointment.

Information about the doctor will be shown; including the doctors name, what medical field they are specialised in to and a portrait photograph. This will allow patients to know more information about the doctor prior to the appointment and be able to recognise the correct doctor when they arrive.

A contact number will also be supplied which will open the phone's dialer when tapped on. This will allow the patient to easily contact the medical clinic if there are any problems.

The location of the appointment will also be shown, stating the name of the location and opening the map if tapped on.

4.2.4 Map

A map will also be provided for every location, accessed through either the map button, or by tapping on a location name. The map will provide real-time location info through the Google Maps API, allowing the patient easy navigation options to find directions and arrive at the correct clinic on time.

4.2.5 Scheduling Appointments

Although appointments are initially set-up by a healthcare professional, patients will have limited control over the timing of the appointment.

The patient will be able to tap an existing appointment, and choose to schedule it (or re-schedule if it is already scheduled). They will be able to post a time showing when they want the appointment to occur.

The app will then either schedule the appointment if that time is available, or respond with three choices close to the appropriate time.

The patient will then be able to either confirm one of the choices, or alter their original time.

4.2.6 Calendar Synchronisation

The app will be able to access the patients phone calendar. When an appointment is scheduled, it will add the appointment information to the phones calendar. It will also remove the appointment information if the appointment is cancelled or rescheduled.

Having access to the phones calendar will also provide useful data for the scheduling algorithm to assign or suggest times appropriate to the user, however this will not be available in the prototype.

4.2.7 Appointment Notifications

When appointments are scheduled, rescheduled or cancelled, an automatic notification will be sent to the device informing the patient that their appointment information has been changed. This will prompt the patient to open the app and confirm that their appointment information is all correct.

Notifications will also allow live communication with the patient's device. Clinics will be able to post up to date information about appointments, such as reminding the patient of medication to be taken prior to the appointment, or simply telling them that they are running late.

Notifications will also be able to inform patients when an appointment is available sooner. This will likely occur when another patient cancels and the slot is free.

There are many use cases for notifications, and they will be developed so that any message can be sent to the device and link to a relevant appointment.

4.2.8 List of Notifications

A list of notifications is also required, so the user can keep track of recently occurring changes to their appointments, as well as being able to take action upon them if necessary.

Tapping on a notification will open an appointment information screen relevant to the related appointment.s

4.3 System Requirements

Alongside the user requirements, a list of system requirements was also created to support the user requirements and ensure the solution is flexible under possible future requirement changes. Here is the finalised list of system requirements:

4.3.1 Cross-Platform

Running on multiple devices is important so that the usability of the service is maximised across the entire patient demographic. Although this prototype will only target the Android platform, it will be designed such that it can be easily adapted to run on alternate devices such as iOS and web-browsers.

4.3.2 Representational State Transfer (Restful) Web Services

It is highly likely that the proposed solution will need to be integrated with other systems. Because of this, multiple endpoints will be available through restful web services, so that components of the system can be easily utilised by other systems with very few barriers to entry.

For example, many scheduling systems already exist, written in different programming languages and implemented in different ways. The system should be designed in a way that it can easily utilise these scheduling systems with minimal complications.

4.3.3 One Development Language

Very few devices share development languages, which can make it troublesome to offer cross platform support.

For the sake of simplicity and given the time requirements, this project will also aim to implement the server and client in the same programming language, C#.

This will reduce training costs required to build and maintain the system, as well as minimising rewritten code to support different devices.

4.4 Development and Technology Used

To fulfil all user and system requirements, various technologies will be used. Here, I will give a brief description of each technology and what role it will play in the proposed solution.

4.4.1 Mono and Xamarin

Mono is an open source implementation of the C# programming language that is cross-platform. This allows development of C# code that can be compiled and run on Windows, Linux and Mac systems.

Xamarin is a mobile app development framework built on Mono, targeting the Android, iOS and Windows Phone platforms. Also, because Xamarin is written using C#, it can integrate easily with the Windows SDK for the development of Windows Desktop Applications.

Xamarin allows business layer logic to be shared across the different platform implementations, essentially having one main code-base rather than a separate one for each platform.

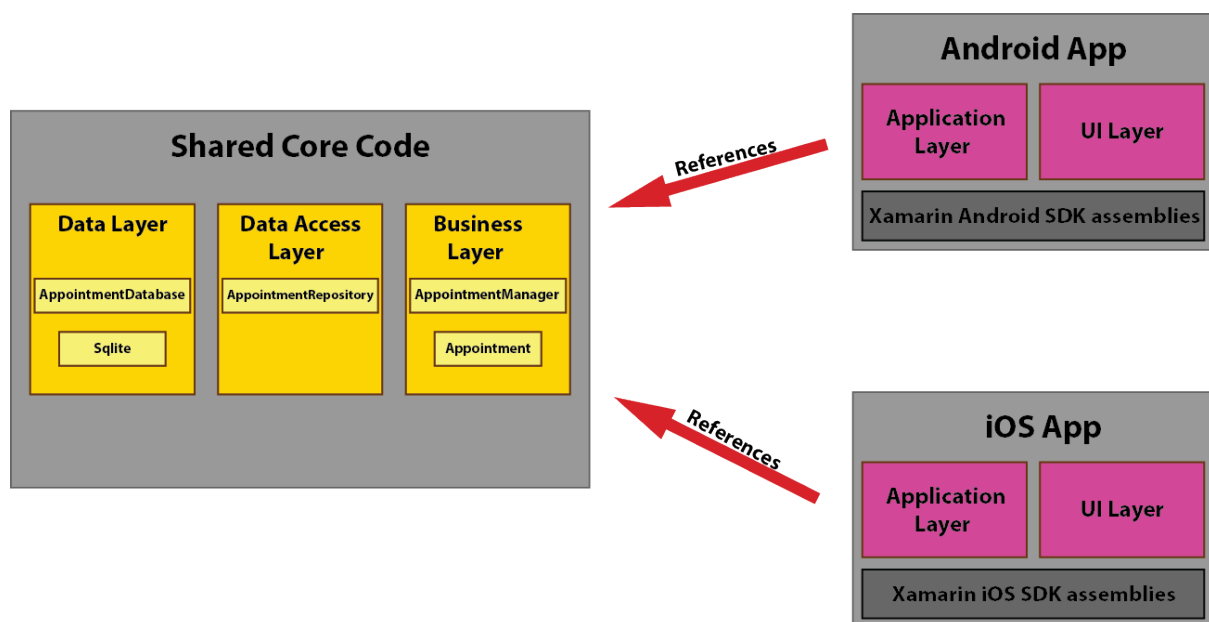


FIGURE 4.1: Diagram showing client architecture and business logic being reused across multiple platforms

The server application will therefore be written in C# with the possibility of being compiled using Mono, allowing it to run on either Windows or Linux server machines (although this is beyond the scope of this project). The Android App will also be written in C# using the Xamarin framework, allowing the code-base to be easily integrated with future platforms.

4.4.2 Communication

The system requires two types of communication based on the requirements, urgent and direct. It is very important to differentiate between these two when targeting mobile devices, due to the following issues:

- Bandwidth is limited
- Devices aren't always online
- Connections are unreliable

- Constant communication can cause excessive power consumption

Due to these issues, a constant connection to a remote server is not possible for long periods of time. This means that the communication must be separated into two separate components, each being useful for different scenarios.

4.4.2.1 Push Communication

Notification messages, also known as push notifications, are a way of sending a short notification message to a device. This is useful for sending urgent messages when the application is not being used, prompting the user for input.

There are however, limitations of this type of communication:

- Only small messages are allowed to be sent
- The server does not know when the message has been received
- Server to client messages only

Push communication can be seen as a 'answering machine service'. The server leaves a short message for the device to be received at some point in the future. If the device is offline, it will receive the message as soon as it comes online again, making it a good system for unreliable connections.

Push notifications should be kept short and only contain enough data to notify the application that it needs to connect to the server for more information.

Due to these limitations, the system will only use push notifications for the following scenarios:

- A sooner appointment is available for the patient
- An appointment has been cancelled and/or needs rescheduling
- Information about an appointment has been changed

'Google Cloud Messaging (GCM)' is the Android service for sending push notifications which this project will be using. It has a message limit of 4kb and is a free service, however it has a daily fair-usage limitation on how many notifications can be sent from a single application.

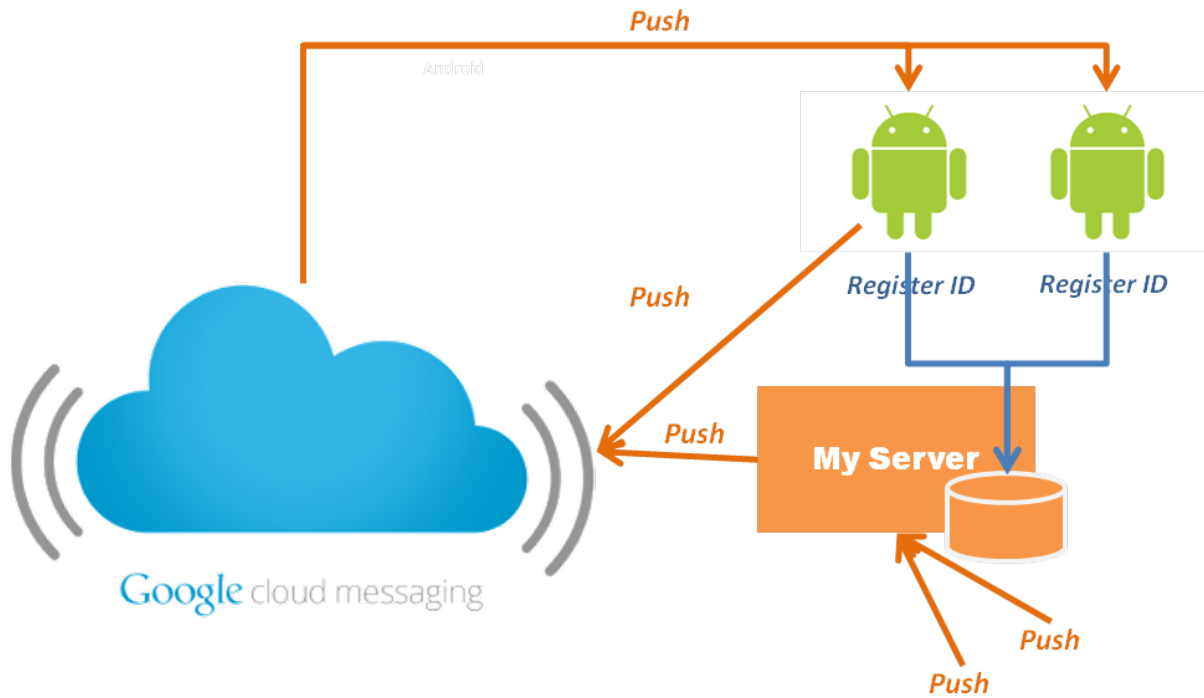


FIGURE 4.2: Google Cloud Messaging - [15]

By using this service, messages are queued and sent to the device as soon as it is available, prompting the user of some kind of notification relating to the application.

Notifications are created by sending a RESTful web request to the google cloud messaging servers, specifying data such as the app id (id of the registered android app), device id (id of the registered android device), and any data to be sent.

4.4.2.2 Direct Communication

After a notification has been received, or simply through using the applications features, the device will require direct communication with the server. It will require communication to:

- Create or Reschedule an appointment.
- Request information about an appointment
- Request reminders about an appointment
- Requesting the database encryption key

All of these direct communication services will be done using restful web-service requests.

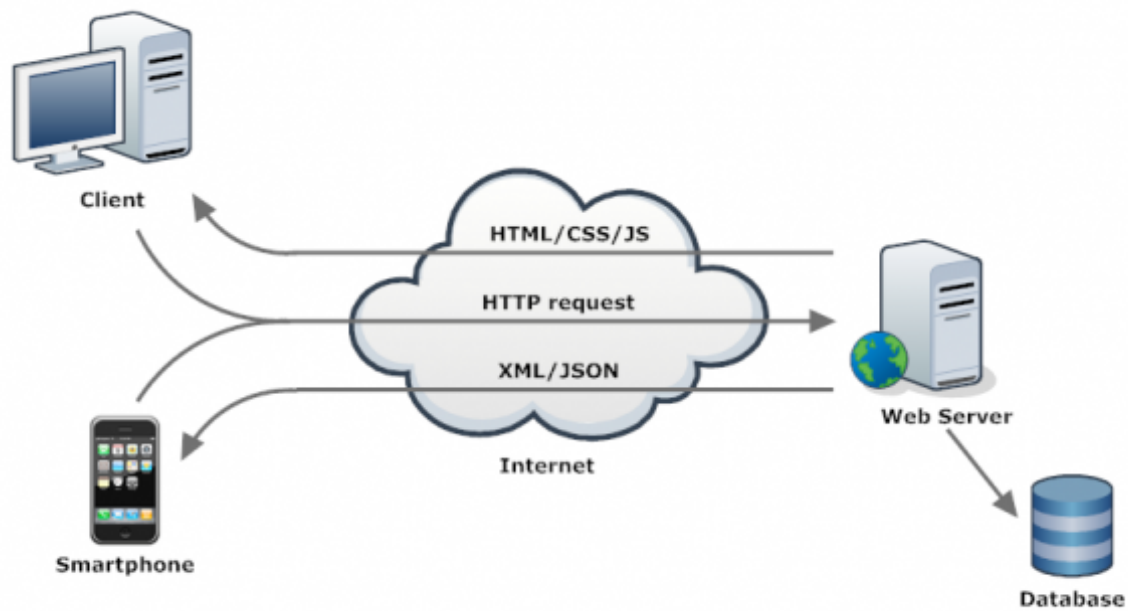


FIGURE 4.3: RESTful Web Services cross-platform communication - [16]

Rest requests can be consumed by almost any programming language and internet accessible device. Data is translated into a mark-up language (JSON will be used for this project), transported via web-services and then reassembled on the target device.

To service these requests, the system will use 'ASP.Net'.

4.4.2.3 Asp.Net

'Asp.Net' is a MVC based web application framework for web development in C#. It allows the binding of restful endpoints to business logic extremely easy, with built in authentication support and custom attributes to define specific restful behaviours.

The figure above shows a method that allows a smart-phone device ID to be registered to a patient's account. This particular method is for use in push notifications, so that the notifications server knows which device to send the patients notifications to. Asp.net will bind this method to the commented url above it by using the 'Route' attribute and the current controller that the method is defined in (the account controller).

4.4.3 Data Storage

Appointment data will be stored in an SQL database, requiring relational models to be created for the solution. The finalised models will be discussed in the implementation chapter.

```
// POST api/Account/DeviceID
[Route("DeviceID")]
public IHttpActionResult SetDeviceID(string deviceID)
{
    db.Patients.Find(User.Identity.Name).DeviceID = deviceID;
    db.SaveChanges();
    return Ok();
}
```

FIGURE 4.4: Example of an Asp.Net web service definition

To create and manage the database, the system will use C#'s 'Entity Framework' library, which allows for quick and easy development of database technology.

4.4.3.1 Entity Framework

'Entity framework (EF)' is an object-relational mapper that allows developers to work easily with relational data using domain specific objects. This eliminates the need for data access code such as SQL queries, and the database can be easily designed using standard C# code.

4.4.4 Hosting

The server application and database will require external server hosting to allow the android app communication with it. I will be using the 'Microsoft Azure' cloud platform to host the application as it allows easy integration with Microsoft frameworks such as 'Asp.Net' and C#.

Chapter 5: Implementation

5.1 Introduction

This chapter will discuss in detail, the implementation of the proposed Android client application, and the server application that accompanies it.

With the system already designed out and with a list of set features, I started exploring various coding patterns I could use to implement my design.

5.1.1 Architecture and Design Patterns

Design patterns are reusable solutions to commonly occurring problems in software design. The use of tried and tested design patterns is important in software because it stops programmers from making the same mistakes, and also speeds up development time as you no longer need to 'reinvent the wheel' with every software engineering project.

Picking the correct design pattern for the problem can prove troublesome, and choosing the wrong ones can lead to inefficient solutions, such as unnecessary duplication of code.

For this project, 'MVC' was used heavily in both the client and server side application.

5.1.1.1 Model-View-Controller (MVC)

Model-View-Controller is a well known software design pattern for implementing user interfaces. The design follows three interconnected parts:

- Model - holds the application data, and logic for accessing and manipulating this data.
- View - holds logic that displays the aesthetic view to the user. It can also contain mechanisms of receiving user input and passing that input along to it's controller.
- Controller - holds controller logic that accesses data from a model and passes it to a view. It can also update a models state with new information, and update the view as the model is updated.

Figure [5.1](#) shows the MVC architecture.

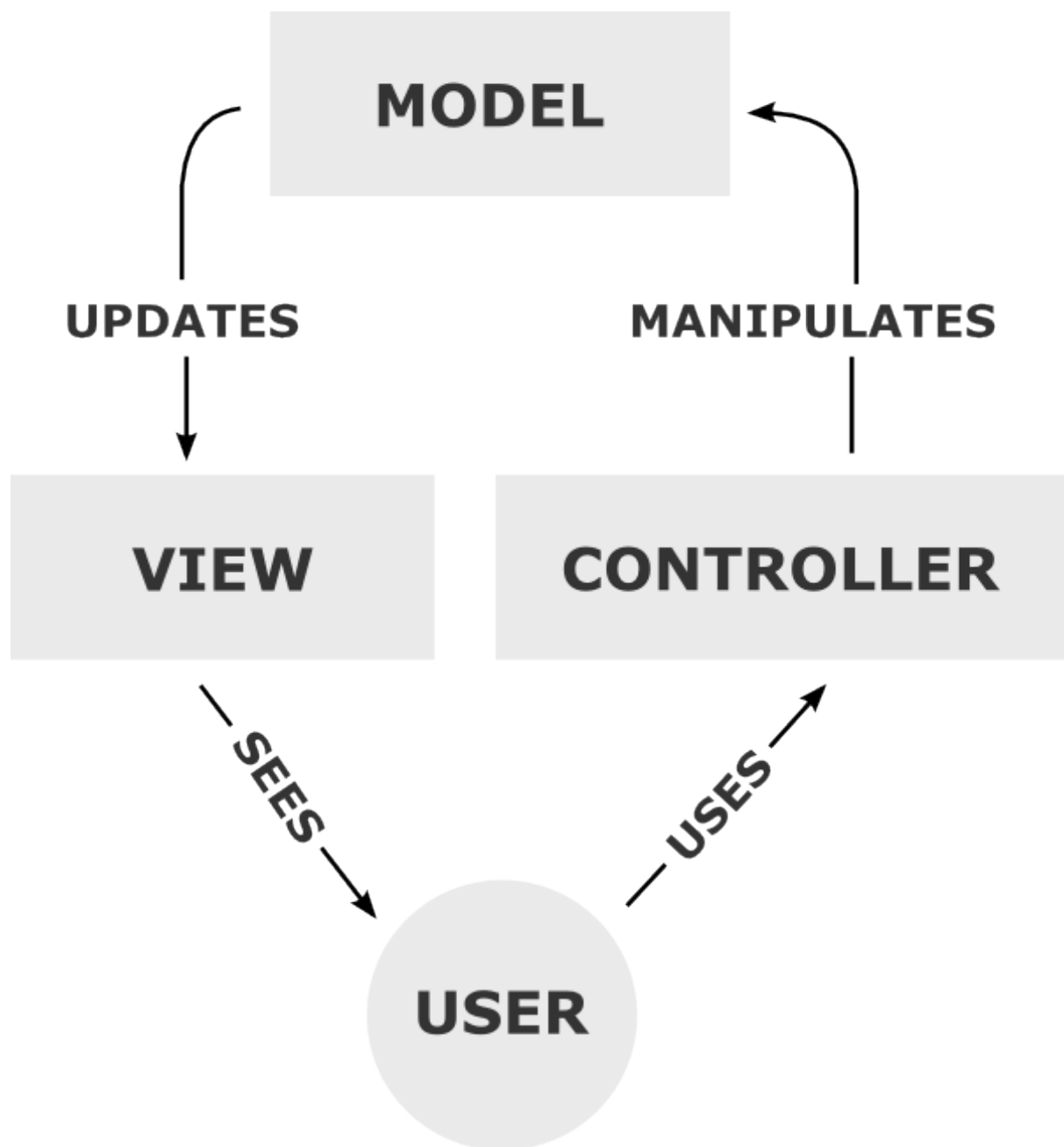


FIGURE 5.1: Model View Controller Diagram - [17]

The server application makes heavy use of certain aspects of MVC. Models are created for data storage and the transport of the objects from server to client. Controllers are also created to design restful web requests that the client can use to request the data models remotely. Views are not used in the project because no user interface is required for the web server.

The Android client application also uses this pattern, with the controllers requesting data from the remote server, using models to formulate the data into the right structure, and displaying the data in views.

5.1.1.2 Component Pattern

The component pattern is also used in the client application. The business logic is decoupled into separate components. A component is a typical manager class with helper functions that can be accessed anywhere in the applications code.

This allows the Android views to access different components and their functionality easily through a component manager.

5.2 Project Setup

A difficult part of the implementation was the development of both the client and the server at the same time, where features of the client are dependant on features of the server and vice versa.

For example, I could not implement the appointment information features of the client application until the server was able to send that information to it. I was also not able to test certain features of the server until the client had implemented ways of authenticating and sending the requests that the server would respond to.

5.2.1 Emulating Requests

To solve this problem, I used a tool called Postman to emulate the client requests to the server.

Postman is a simple web application that allows you to design RESTful web requests very easily and send them to a server. By doing this, I was able to design and test the requests I needed easily.

This meant I could focus solely on implementing the basic server features without having to develop the Android application alongside it. I later found that this was also an invaluable method of testing server features when problems occurred.

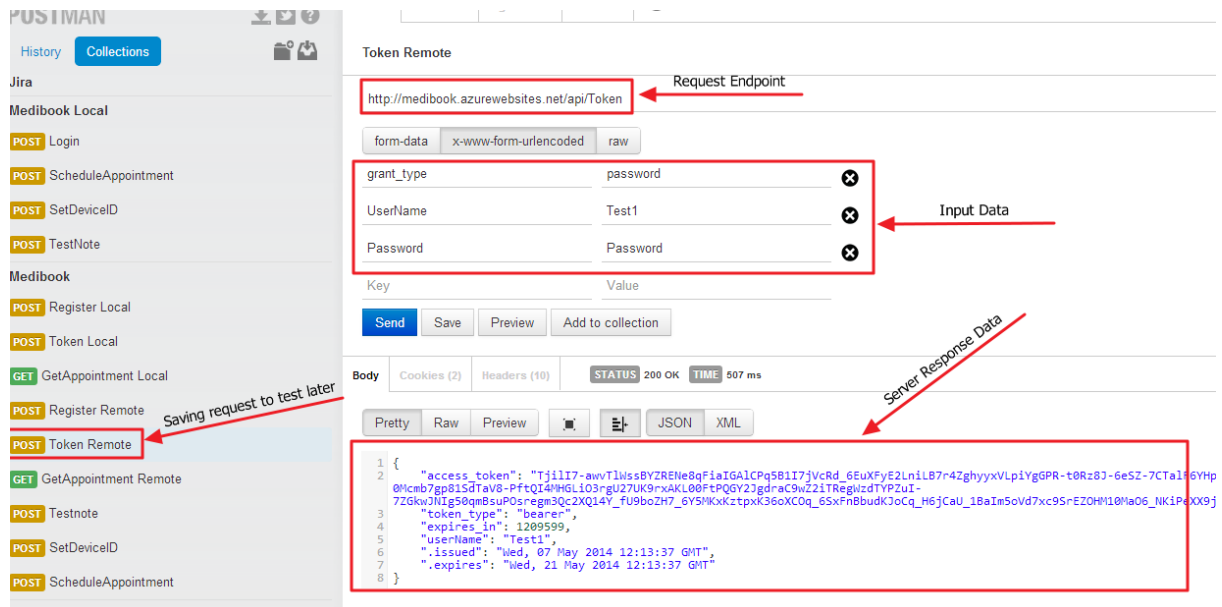


FIGURE 5.2: Screenshot showing the use of Postman

5.2.2 Organising Code

A feature in most modern programming languages (including C#) is the idea of modular design. This concept emphasises the separation of code functionality into interchangeable modules. These modules can then reference other modules easily, and makes sharing code simpler.

Early on in the implementation stages, I found that I was having to duplicate some code and data values in both the client and the server application. This caused many unnecessary issues. For example, I would sometimes change one value and forget to change the other.

To solve this problem, I structured all of my code into separate modules so that they could easily be shared with both my client codebase and my server codebase.

I organised the code into the following projects:

- Medibook.Client.Android - The Android application that focuses entirely on Android specific design. *References the Medibook.Client library.*
- Medibook.Client - The client library that contains all business logic that is shared with Android and IOS implementations. *References the Medibook.Shared library.*
- Medibook.Server - The server application. *References the Medibook.Shared library.*

- Medibook.Shared - The shared library that is shared between the Medibook.Client and Medibook.Server projects.
- Medibook.Testing - A unit-testing library that tests specific features of the client application and ensures they are working correctly.

With this modular setup, I was able to develop reusable code with no duplication, and allowing for a future iOS app to use the same business logic as the Android app.

With my projects setup, I began developing the basic infrastructure that both the server application and android application would use.

5.3 Server Structure

Using the Asp.Net framework, the majority of the server's structure was already pre-ordained with a strong emphasis on using the Model-View-Controller design pattern. The server was therefore structured into two components:

- Controllers - Containing the business logic and servicing all RESTful API request endpoints.
- Models - The data models for storing in the database and designing the structure of sent request data.

The server application did not have views because no interface was required for the prototype.

Most of the server implementation was done using controllers, as methods defined in a controller would become a web service endpoint which web-requests from the Android application could execute.

Figure 5.3 shows the basic setup of an Asp.Net controller, including routing and authorisation. Every request method in this controller requires an authorised token.

5.3.1 Authentication

Authentication is supported by Asp.Net, so it's implementation was not as tricky as first thought in my original design.

```

17 namespace MediBook.Server.Controllers
18 {
19     [Authorize]
20     [RoutePrefix("api/Appointment")]
21     public class AppointmentController : ApiController
22     {
23         private DataContext db = new DataContext();
24
25         // POST api/Appointment/ScheduleAppointment
26         [Route("ScheduleAppointment")]
27         [ResponseType(typeof(ScheduleResponse))]
28         public IHttpActionResult ScheduleAppointment(ScheduleAppointmentBinding model)
29     {
30         //Passing of the datetime object doesn't work, it's incorrectly serialized by

```

The image shows a code editor with C# code for `AppointmentController`. Red boxes highlight `[Authorize]`, `[RoutePrefix("api/Appointment")]`, and `[Route("ScheduleAppointment")]`. Red arrows point to these boxes with the following text: "Only authorized requests are allowed to execute methods in this controller" (pointing to `[Authorize]`), "Controller route prefix" (pointing to `[RoutePrefix("api/Appointment")]`), and "Request route for this method" (pointing to `[Route("ScheduleAppointment")]`).

FIGURE 5.3: Example of a controller code showing authorised and routed requests

By adding the `[Authorize]` attribute to a controller or individual request methods, Asp.Net recognises that this request (or all requests within the controller) requires authentication in order to access it.

To authorise requests, authorisation keys called 'Tokens' must be added to the requests authorisation header. If the authorisation header is missing or incorrect, the server will end the request and respond with the HTTP request code '401 Unauthorized'.

All requests sent to the server application require authentication, except the login and register requests, to ensure that patient data is kept secure.

5.3.2 Routing

When the server receives a request from the client, it will be in the form of a url. Asp.Net automatically parses this url into segments, in the form of 'controller/action'.

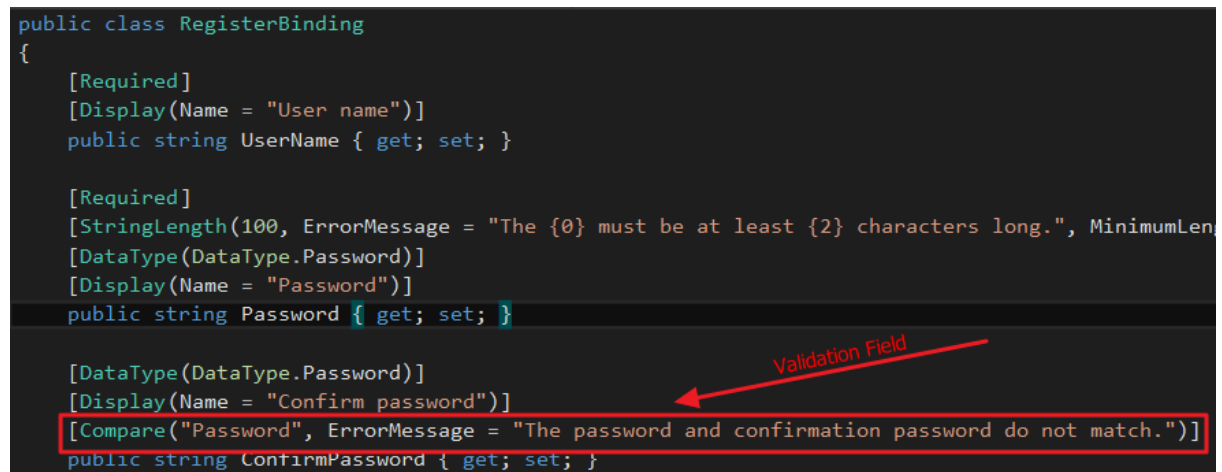
After the segments have been parsed, it finds the correct controller defined with the controller segment, then the correct action method with the action segment and executes it, returning the output to the sender.

As a working example, if the client sent a request of 'Appointment/Schedule' would tell Asp.Net to look in the Appointment controller for the Schedule request method.

5.3.3 Request Methods

After the request has been routed to the correct method, the data needs to be de-serialised into a binding model.

This binding model defines the structure of the request, specifying the data types for each input parameter. It also provides validation tools to ensure that the data is input correctly.



```
public class RegisterBinding
{
    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

The image shows a code editor with the above C# code. A red box highlights the `[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]` attribute. A red arrow labeled "Validation Field" points to this attribute.

FIGURE 5.4: Example of a binding model associated with the register account request

If the binding model is invalid for any reason, the server will end the request and respond with the HTTP status code '400 Bad request'. Error messages will also be returned if available, giving the request sender a reason as to why the request failed.

After the binding model is successfully validated, the request can be carried out the result is returned to the sender.

5.3.4 Data Storage and Retrieval

For data to be stored and retrieved easily, I used the Entity Framework library which integrates easily with ASP.Net and eliminates the need for most of the data access code.

Two methods are available for database development when using Entity Framework; code first and database first. Code first allows you to let Entity Framework create and setup the database as you write your data models. Database first allows you to use entity framework to generate data models from an already existing database. For quick development, the code first method was chosen for this project.

5.3.4.1 Data Models

Data models work the same way as binding models, in that you design properties with data types for the data to bind to.

You can even use data models as binding models, which allowed me to reuse the same data models to serialise and de-serialise the data when sending to and from the client. I found however, that this was not always a good solution, as some data models have extra information that the client doesn't need (for example, the password property for a user model should not be sent to and from the client, especially when it is a different user).

Entity Framework data models also allow you to assign attributes to the properties such as '[key]' to modify it's behaviour as a primary key when stored in the database.

The best and most useful feature in entity framework is the automatic handling of foreign keys. If in one data model, you wish to reference a different data model, you can set the data type of that property to the data model's type and Entity Framework will do the rest of the hard work. This eliminates the need for any sql queries to fetch other data models that correspond to the foreign keys.

All data models were first designed by hand using entity relationship diagrams, however as the project progressed, these models became dramatically different as new features were added.

5.3.4.2 Data Context

When using Entity Framework, you must also create a data context.

This data context maps all of the entities and relationships that are defined in each data model to a database using the 'DbSet' class, allowing you to insert, update and delete data easily.

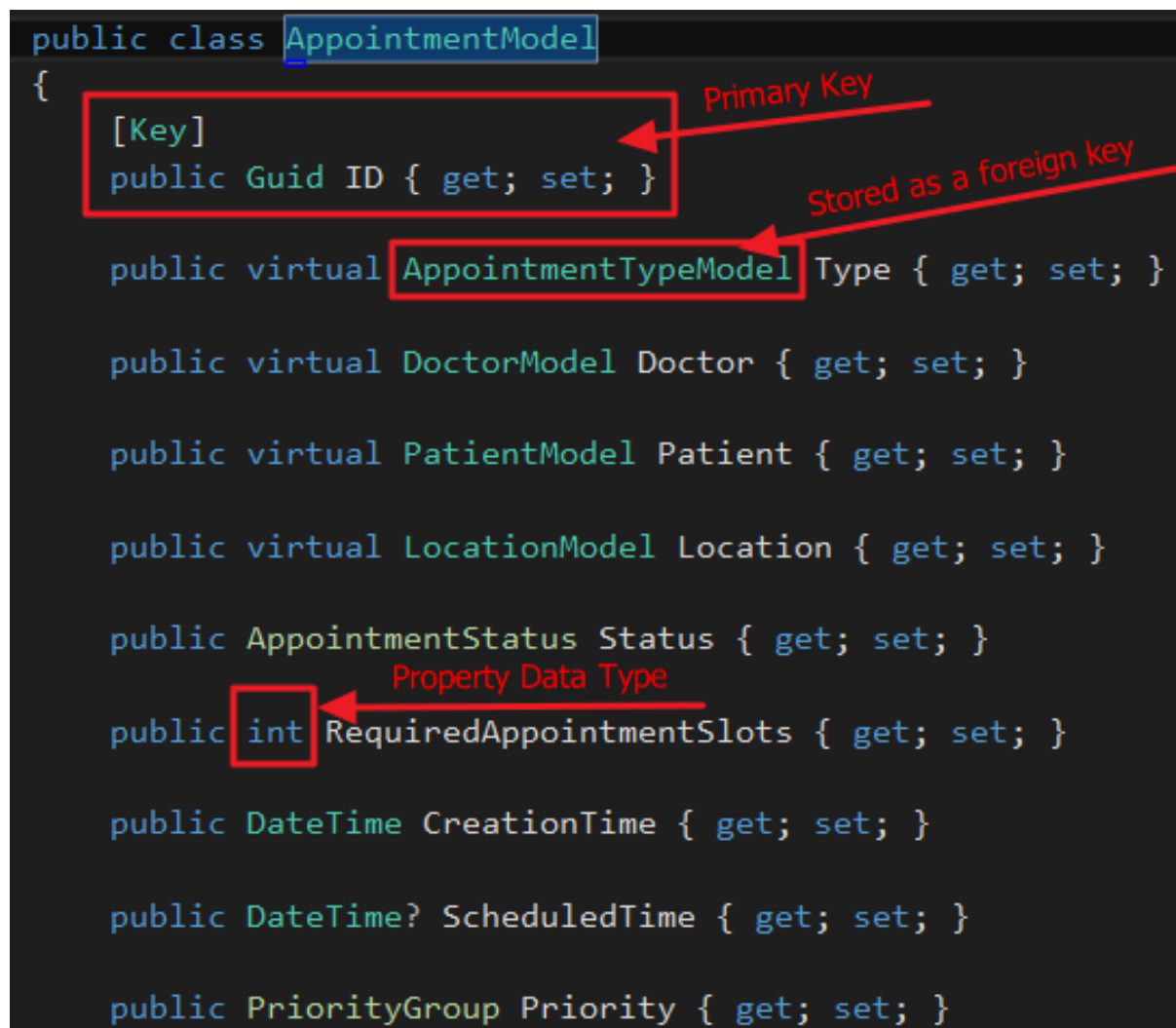
The data context also stores changes to the database transactionally, meaning that changes will only be written when calling the 'SaveChanges' method. This allows for changing the models with ease and not adding any performance issues as it writes unnecessary changes.

Using the data context, you can also query the data easily using LINQ.

5.3.4.3 LINQ

LINQ allows for the easy querying and updating data sets through C# code. It follows an SQL-like syntax uses standard, easily learned patterns.

Using LINQ allowed me to select data easily through code, removing the need for direct SQL queries which could over complicate the process and introduce security vulnerabilities.



```
public class AppointmentModel
{
    [Key]
    public Guid ID { get; set; }

    public virtual AppointmentTypeModel Type { get; set; }

    public virtual DoctorModel Doctor { get; set; }

    public virtual PatientModel Patient { get; set; }

    public virtual LocationModel Location { get; set; }

    public AppointmentStatus Status { get; set; }

    public int RequiredAppointmentSlots { get; set; }

    public DateTime CreationTime { get; set; }

    public DateTime? ScheduledTime { get; set; }

    public PriorityGroup Priority { get; set; }
}
```

The image shows a code editor with the following annotations:

- A red box around `[Key]` and `public Guid ID { get; set; }` with an arrow pointing to it labeled "Primary Key".
- A red box around `AppointmentTypeModel` with an arrow pointing to it labeled "Stored as a foreign key".
- A red box around `int` in `RequiredAppointmentSlots` with an arrow pointing to it labeled "Property Data Type".

FIGURE 5.5: Example of an Entity Framework Data Model

5.3.4.4 Migrations

After data models are created and added to the data context, Entity Framework allows you to migrate sample data when using the code first method. Migrations were therefore used to insert sample data into the database, such as appointments, patients, doctors, locations and any other information required for the prototype to function.

Besides adding data to a database, migrations also generate the SQL code necessary for making alterations to a database, using a version control system to allow different revisions of the database to be created. This allowed me to manage and update my remote database easily and not worry about adding new tables manually.

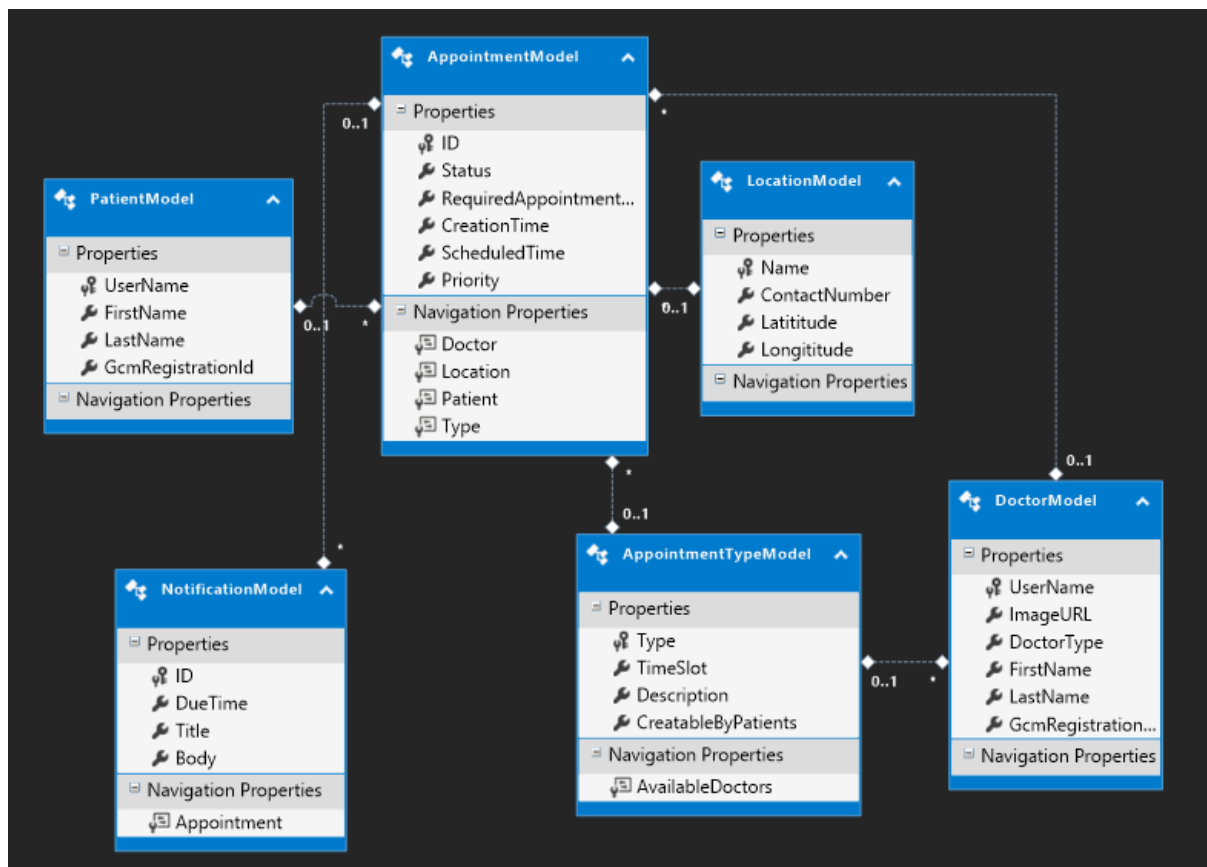


FIGURE 5.6: The Final Entity Relationship Diagram

```

public class DataContext : DbContext
{
    public DbSet<AppointmentModel> Appointments { get; set; }
    public DbSet<AppointmentTypeModel> AppointmentTypes { get; set; }
    public DbSet<DoctorModel> Doctors { get; set; }
    public DbSet<PatientModel> Patients { get; set; }
    public DbSet<LocationModel> Locations { get; set; }
    public DbSet<NotificationModel> Notifications { get; set; }

    public DataContext()
    {
        /*
        Example LINQ statement to select rows of data
        from the Patients table where the name is 'Andrew'
        */
        var patientsCalledAndrew = Patients.Where(patients => patients.FirstName == "Andrew");
    }
}

```

Represents an SQL Table in the Database

Example Selection Query

FIGURE 5.7: Example of the Entity Framework Data Context

5.4 Client Structure

The client application is composed of two separate code bases, the client core library and the Android application.

5.4.1 Client Core Library

The client core uses the component based design pattern and is responsible for the majority of the applications business logic. This includes:

- Sending and receiving requests from the server
- Storing data in the phone's memory and allowing multiple views to access it easily.
- Writing data to the phone's internal sqlite database
- Any other logic that isn't a platform specific feature

The client core is accessed via a static reference which means a component can be retrieved easily anywhere in the application. This turned out to be very useful for storing information that needed to be accessed by multiple parts of the client application.

5.4.1.1 Creating Requests

Besides storing data, the client core was also responsible for sending and receiving data from the server. To do this, the client needs to create a request and send it to the server so that it can receive data in return.

A client request is composed of three parts, the request url, the request parameters and the request method.

Figure 5.8 shows a request for confirming an appointment time with the server, specifying all three parts of the request.

Figure 5.8(a) shows that this particular request extends the 'AuthPostRequest' which is retrieves the authorisation key from the account component and adds it to the request header. It also sets the request method to 'POST'.

Figure 5.8(b) shows the request url, which it passes into the base constructor and is added to the request. This particular request targets the 'ConfirmSchedulingChoice' request method in the server's Appointment Controller.

```

namespace MediBook.Client.Core.Components.Appointment.Requests.Post
{
    public class AuthPostConfirmSchedulingChoice : AuthPostRequest
    {
        public AuthPostConfirmSchedulingChoice(Guid appointmentId, PossibleTime time)
            : base("Appointment/ConfirmSchedulingChoice")
        {
            this.Request.AddParameter("AppointmentId", appointmentId);
            this.Request.AddParameter("Time", time.Time);
            this.Request.AddParameter("AppointmentsToCancel", time.AppointmentsToCancel);
        }
    }
}

```

FIGURE 5.8: Creation of a client request

Figure 5.8(c) shows the parameters being added to the request. A parameter is a string key value pair which is added to the request so that it can be parsed by the server's request binding model.

5.4.1.2 Request Response

After the request has been created, the client executes it, sending it to the server and waiting for a response.

```

public async Task ConfirmSchedulingChoice(PossibleTime possibleTime)
{
    var request = new AuthPostConfirmSchedulingChoice(ActiveAppointment.ID, possibleTime);
    var response = await request.Execute<ConfirmSchedulingResponse>();

    if (response != null) throw new RequestException(response.Message);

    ActiveAppointment.ScheduledTime = possibleTime.Time.ParseFromString();
    ActiveAppointment.Status = AppointmentStatus.Scheduled;
}

```

FIGURE 5.9: Example of the client sending a request

In figure 5.9, you can see that the execution method also specifies a 'ConfirmSchedulingResponse' parameter. This is a binding model that the client then uses to de-serialise the response data into a usable form. This is similar to the server's implementation of the binding models and allows the specification of data types for the response data.

5.4.1.3 Asynchronous Requests

One problem I ran into when implementing requests was that my client application would halt until it received a response from the server (or the connection timed out). To solve this issue,

I implemented asynchronous requests so that the program would continue functioning without waiting for a request to return its result first. This implementation uses the 'async await' feature in C#. When the program hits an await operator, the program will return to the request caller until a response is received. You can see this being used in figure 5.9.

This allowed me to effectively execute requests in the background without having an impact on the main application.

5.4.2 Android Application

The Android application, like the server application, uses the Model-View-Controller design pattern. The application structure was composed of a few base components; activities, intents, resources, layouts and services.

5.4.2.1 Activities

Android activities are the Android implementation of controllers. Usually tied to specific screens, they handle any logic required by the screen, including the creation of and switching to other activities if necessary.

Activities bind to an Android Layout and have several states that depend on if the activity is on the screen or not at the current time. For example, if the user presses the home button, the application is minimised and the activity is paused.

The Android client contained several activities, one for each screen available screen.

5.4.2.2 Intents

Android Intents are a description of a task to be performed. For example, when starting an activity, a 'StartActivity' intent must be created which takes the activity as a parameter. Intents are also useful for executing system tasks such as opening the phones dialer or calendar.

5.4.2.3 Resources

Android resources are all resources that the application uses, including pictures, layouts, sounds, constants and any other static data specific to the application.

As well as storing resources, the Android OS allows these resources to be easily accessed from within Activities.

The Android client uses resources to store all the layouts, fonts and images used in the application.

5.4.2.4 Layouts

Android layouts are views that define the visual structure for a user interface in the Android application.

Android provides an XML based API that corresponds to view classes and subclasses such as widgets that can be used in the user interface. It also allows you to attach properties to these widgets that set its position, id so the activity can find it, and many other variables that modify the widget's behaviour.

Xamarin also provides a drag and drop graphical user interface that allows for fast development of Android layouts for multiple devices.

As widgets are added to the design, the designer generates the XML source code for the layout. It also allows you to edit the layout code directly, re-rendering any changes you make. This made it extremely useful as you did not need to compile and run the Android app (a process that can take a long time) every time you make a change to the design.

5.4.2.5 Services

Android services allow tasks to run in the background. They run in their own independent life cycle, meaning that they continue to function even when the app isn't running.

Android services were used to process incoming notifications sent from the Google Cloud Messaging server. This allowed for the phone to receive messages whilst the app was not running and inform the user of changes to their appointments.

5.5 Login and Authentication Feature

With the basic infrastructure established, implementing the app authentication procedures was fairly simple.

To manage the authentication requests and logged in account, I created a component in the app core library which sends the login, registration and logout requests, as well as storing the current token used to sign authorised requests.

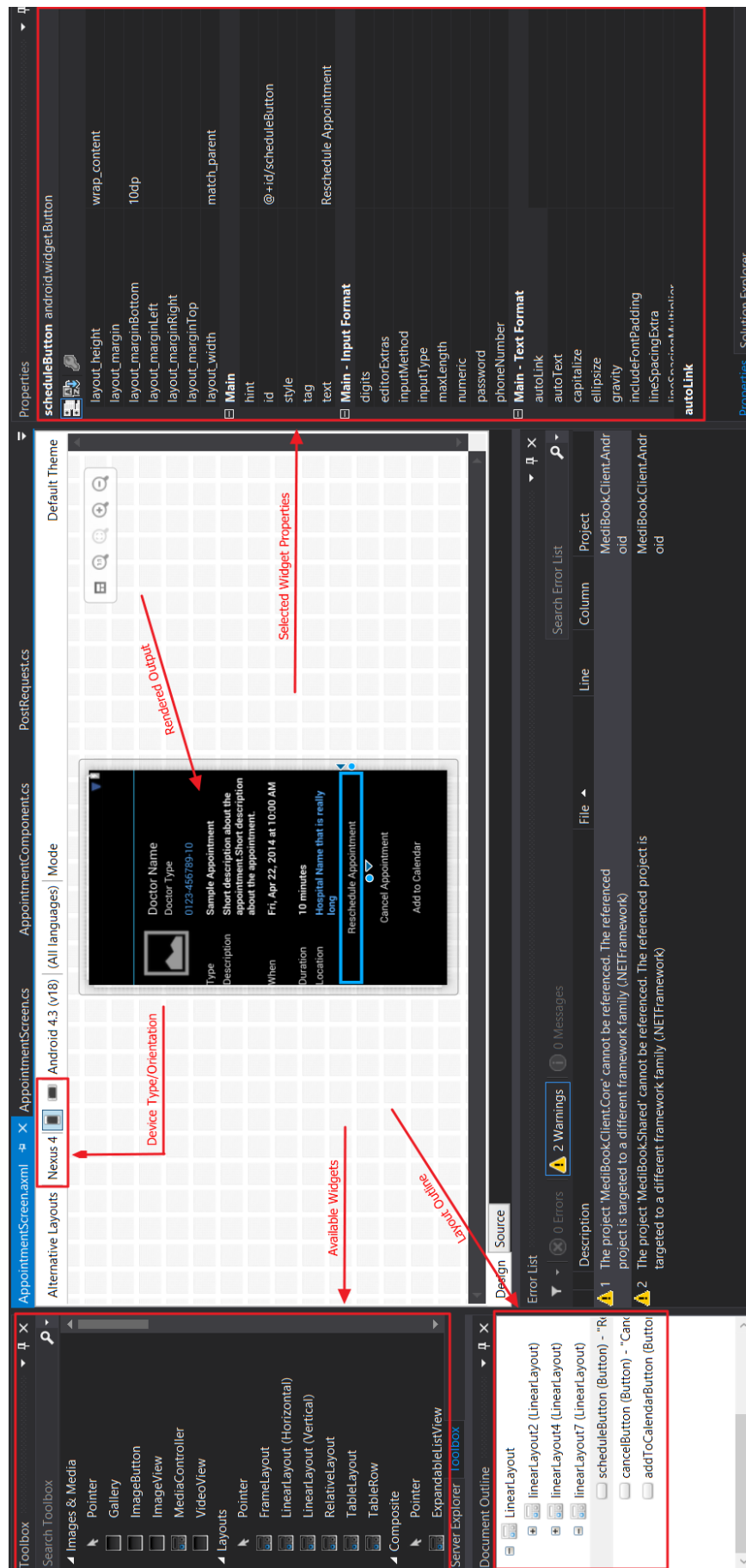


FIGURE 5.10: Screenshot showing the use of the Xamarin Layout Designer

The Android visual implementation of this feature was also very straight forward. I created an activity and set it as the main launcher activity. This tells the Android OS to open this particular activity first when starting the app.

The login screen has two text input widgets that allows users to input their login details. Two buttons were also added which linked to the activity and sent the login and registration requests.

5.5.1 Login Request

The login request sends the user name and password input via the HTTPS protocol ensuring it is sent securely. If the login is successful, the server will respond with an access token, which is then stored in the account component.

5.5.2 Registration Request

When creating a new account, the following information is needed as parameters in the request:

- UserName - unique user name required to log in with.
- Password - secret password required to log in with.
- AccountType - Either 'Patient' or 'Doctor'.
- FirstName
- LastName

Upon creating a new user account, the server will assign this account the role of either doctor or patient and add some example appointments to the account (for demonstration purposes). If the request is successful, the server responds the HTTP status code '200 OK'.

5.5.3 Incorrect Input

If either the login request or registration requests have incorrect input, the server will return the HTTP status code '401 Bad Request', also stating a reason as to why it failed such as 'Incorrect Password'. To give some feedback to the user, I needed to also implement an error text field in the layouts design which displayed when there was an error.

5.5.4 Conclusion

The login and authentication feature was surprisingly simple to implement due to the fact that Asp.Net has the majority of it built in. Combined with the role system, it made it very easy to identify users server side from the access token used to send the request, which became invaluable when deciding which appointments to send to the client.

I also found that because the login request was a prerequisite for any further action of the app, I added a progress dialog widget which shows whilst the app is waiting for a response from the server.

A dialog is a simple modal view, much like a pop up which displays data on top of the main layout. This simple visual feedback was effective at telling the user to be patient while the login request is in progress.

5.6 Home Screen Feature

For the home screen, two main features were required; the appointment list and the notification list. To allow for two separate lists on the same screen, the home screen was implemented as a tab activity.

5.6.1 Tab Activity

Tab activities allow easy navigation of two separate user interfaces and keep the design clear and separate. Navigation tabs are added to the top of the view in an action-bar to allow the user to easily switch between the different interfaces.

To embed an interface into a tab, fragments must be created to host these interfaces. In this case, list fragments were used to display and manage the lists. The home screen layout then had a fragment container, which hosted the selected tab's corresponding fragment within it.

5.6.2 Fragments

Fragments are child activities that represent a portion of the user interface. Instead of embedding a layout into an activity, you can embed the layout into a fragment, and then add that fragment to an activity. This results in the ability to have multiple screens per activity.

5.6.3 List Fragments

By using the list fragment, I was able to create a layout design for a single item and assign it to an adapter.

Android adapters act as bridges between the layout and data for the view. This is required for a list fragment so that it knows how to bind the data to the layout correctly, allowing for dynamic lists to be created easily. It also adds functionality such as the order in which items appear in the list, as well as touch event handling to determine which item was selected.

By keeping a reference of the appointment/notification and its index in the list, I was able to invoke a new activity with the relevant information needed when the user tapped on a list item.

5.6.4 Retrieving Appointments

Whilst implementing the client side, I used sample data to populate the data lists. Once the client implementation was complete, I started on implementing a request to fetch all of the appointments that are assigned to a patient.

5.6.4.1 Server Side Request

I started by creating a new controller in my server project to manage appointment requests, setting all requests in the controller to require authentication.

This allowed me to not only keep the information secure, but also fetch the patient's identity easily by using the Asp.Net property 'User.Identity' that is available in all authorised requests. The Asp.Net populates this by checking the used token against logged in sessions, retrieving the correct identity from the session.

5.6.4.2 Appointment Model

An appointment model was created based on my original ER Diagram and put in the shared project so that both the client and server could access it. This was to allow the server to store, retrieve and send the data to the client, as well as allowing the client to use the model to de-serialise the returned data from the request.

The model was then added to the data context and a new reference of the data context was created within the appointment controller.

With the patient's user name available, it was very easy therefore to retrieve all appointments linked with that user name, returning them in the request.

Figure 5.5 shows the appointment model and its properties. Although not all of the information was relevant for the list, this query sent of the entire appointment model so that it could later be used in the Appointment Information Screen, removing the need for a second query to be executed.

5.6.5 Action Bar

With the relevant appointments being successfully retrieved from the server, the appointments were stored in the appointment component and injected into the list adapter.

Later on in the implementation, I found that these appointments were becoming out of date. I found that a good solution to this was to create a button to manually fetch appointments from the server and repopulate the list.

The list view took up the entire screen and I did not want to remove focus from it as it was the main feature, so I created a second action bar above the tab bar.

To implement this, I utilised the 'App Compat' Android library which adds a top bar menu to all your activities. This made it very simple to add button tabs at the top of your screen, as well as displaying the name of the current screen to make navigation of the app easier.

A refresh button was added which, when pressed, would request the appointment data from the server again. A rotating progress icon was also added to the button to show whilst the request was in progress. Another solution to ensure appointments were kept up to date was the adding of the request to the 'OnActivityResumed' method, which executes whenever the activity becomes visible again. This resulted in the app refreshing the appointments whenever the home screen was re-opened.

A logout button was also added to the bar to allow the switching of accounts and returned the user to the login screen.

5.7 Appointment Information Feature

With the appointments already retrieved, implementing the appointment information feature was simple. When a appointment was tapped on the list, it would set the 'CurrentAppointmentOpen' property to the relevant appointment in the appointment component. It then opened a

new activity for the information screen.

To create the information screen, I used the layout designer to create a layout with the required widgets.

When the appointment information activity was started, it displayed the layout and retrieved the relevant appointment from the appointment component. It then assigned the text fields in the layout to the relevant information in appointment so that the information was specific to the requested appointment.

5.7.1 Doctor's Image

One technical challenge faced was providing the doctor's image for this feature. Images are normally added to the resources folder, however this would not be an effective solution because the app would then need to be updated whenever a new doctor is added.

Due to having a limitation on requests were only textual information could be sent, I added an image url to the doctor model. This url pointed to a web server hosting the image where by using a separate request, it could be retrieved.

Upon opening the activity, the image widget therefore contains a place holder image while it downloads the actual image onto the phone from the external server. This worked effectively and allowed doctor's portrait images to be changed remotely on the server without having to update the app.

5.7.2 Buttons

Several buttons were also created to allow the patient to easily schedule, re-schedule and cancel their appointments.

Upon pressing these buttons, the layout would invoke a method in the activity that carried out the functionality. The schedule button would create a new activity that took the user to the scheduling screen. The cancel button would send a cancel appointment request, sending the appointment id to the server in the request.

5.7.3 Action Bar

An action bar was also added to the appointment information screen using the same method as in the home screen. In the action bar, a map button was added that created a new activity for

the map view, as well as a back button that finished the current activity and returned to the home screen.

5.7.4 Dialer

By tapping the contact number text field, the activity was required to open the phones dialer with the same number.

This was relatively simple to implement. The text field's 'OnClick' property was utilised to invoke an 'OpenDialer' method in the activity. This method created an Android Intent that told the Android OS to open the dialer action of the phone. The 'SetData' field to set the number from the appointment so that the dialer had access to the relevant information

One issue I had was that in order for the dialer to parse the number correctly, the string value passed to it must be prefixed with 'tel:', which was missed in the first iteration of it's implementation.

5.8 Calendar Feature

The calendar feature added a button to the appointment information screen, allowing the user to add (or remove) the appointment details to their phone calendar.

Accessing the phone calendar was a complex process, requiring Android app permissions, content resolvers to retrieve the information and extensive reading of poor Android documentation.

5.8.1 Calendar Querying

Android is a privilege-separated operating system, requiring all applications to request privilege permissions in the applications manifest. Certain restricted features are not available until these privileges have been added, which prompts the user upon the install of the application of the restricted features it has access to.

The app manifest is essentially the Android application's configuration file, specifying which android sdk version it uses and any permissions it requires. The calendar access required the permissions 'READ_CALENDAR' and 'WRITE_CALENDAR', prompting the user that the application was able to read and write from the calendar.

Content resolvers are used to manage access to a structured set of data in Android. One problem that content resolvers solve is data access conflicts, when multiple Android applications are modifying the same data at the same time.

As an added feature, I wanted to query the the calendar to see if the appointment was already added. This would allow me to swap the 'Add to Calendar' button with a 'Remove from Calendar' button.

To do this, I needed to create a content resolver to fetch the correct calendar id, and then create a second content resolver to check if any entries existed in that calendar.

5.8.2 Issues

This feature started out relatively simple, but became increasingly problematic because Android has lots of deprecated methods of accessing calendar information. Deprecated methods are old ways implementing a feature, but are left in the Android sdk to preserve backwards compatibility.

One issue I encountered was that upon removing an appointment from the calendar, I found that it still existed when I queried it. After reading different versions of the same Android documentation, I found the most up to date implementation and discovered that when removing an entry from the calendar, the latest version of Android sets a 'removed' field boolean to true rather than deleting the entry. Although this was very simple to fix (by checking this 'removed' field), it took me many hours to find out why my entries were not being deleted.

Another issue I encountered was that upon cancelling an appointment, it was not removed from the calendar, resulting in it displaying an incorrect time. This was fixed by calling the 'RemoveFromCalendar' method when an appointment time is updated.

5.9 Location and Map Feature

The location and map feature aimed to prevent patients getting lost on their way to the appointment. By tapping on the location name or selecting the map button in the action bar of the appointment information screen, the app opened the map activity.

The map activity was very simple to implement. Android has a 'MapFragment' class designed for embedding 'Google Maps' directly into apps, providing a dynamic map and useful helper methods to manipulate it.

5.9.1 Google Maps API

To utilise the map fragment, Google requests that you provide your application API key in the app's manifest file. To obtain a key, you must register your app with google through the google

developer portal, a process that takes just a few minutes, is free of charge and allows you to access many useful API's.

After adding the API key provided, the map fragment was able to render a map within the app.

5.9.2 Placing a marker

After implementing the app map, the solution required a marker on the map displaying the location position and name. This required me to change the location data model, adding latitude and longitude properties to it so that the server could send the correct location.

Once the marker was set on the map, the map's settings were changed so that the map view would open with the marker centred.

5.10 Appointment Scheduling Feature

The main feature of the application was the ability to schedule and reschedule appointments. This was the hardest feature of the project aside from the initial design. This was primarily down to the complexity of the scheduling process, resulting in many obstacles that needed to be avoided. To begin implementing feature, I created several workflow diagrams to emulate the scheduling of an appointment by hand. Different scenarios were drafted, providing a set of clear instructions that the server and client carried out for each one. By modeling out the scheduling process by hand, it made the implementation much easier.

5.10.1 Date Picker

To start scheduling an appointment, a button was added to the appointment information screen that invoked a schedule appointment activity. This activity had three buttons in it's layout; a time picker button, a confirmation button and a cancel button.

When selecting the time picker, a dialog was created that showed a date and time slider. Care was taken to ensure that the user could not pick a time before the current date. After the user had inputted a time and date, the confirmation button would execute the request.

5.10.2 Example Scheduler

To emulate a real world scheduling algorithm, I implemented an 'Example Scheduler' program to find an appointment for the user. The aim of this program was to return an appointment time as close to the requested time as possible. The first obstacle to overcome was appointment conflicts.

To detect conflicts, the scheduler queries the appointment database for all appointments where the times overlap. It first calculates the time range of the requested time using the appointment duration. If any scheduled appointments overlap, the scheduler will flag the time as conflicting.

The scheduler deals with conflicts in two ways; using a priority based system to cancel other appointments or suggesting other available times. Every appointment is assigned a priority in the appointment model, based on how vital it is that this appointment is scheduled soon. This is used to compare how important appointments are relative to each other and whether an appointment can be rearranged or not.

If the conflicting appointments are less important than the requested appointment, the scheduler will cancel them and inform the patients and doctors by sending notifications to the devices. If the priorities are the same or the requested appointment is less important, the scheduler will find three possible times as close as possible to the requested time. It does this by incrementing the requested time hourly and repeating the conflict detection process again until it finds three possible appointments.

5.10.3 Confirming the Appointment

If the scheduler returns one result, the server will confirm the appointment and respond to the client that the appointment has been scheduled. Otherwise, it will return the three possible choices.

If three choices are received from the server by the client, the activity will start a new scheduling choice activity screen, which displays the times to the patient and asks them to choose one. Upon choosing a time, the client will send a confirmation request to the server to schedule the appointment.

5.10.4 Time issue

One issue I ran into when implementing this were time zones. The time calculated on the phone was in the time-zone 'British Summer Time (BST)', which was an hour ahead of the time calculated on the server 'Coordinated Universal Time (UTC)'. This caused many problems when sending and receiving times because they would become out of sync.

My first attempt to solve this was simple, change the time zone that the server is running on. I later found that this was not possible because the server was hosted on the Windows Azure

Cloud, designed to be run in any geographical location and therefore prohibiting any changing of the time-zone. The app had to be therefore designed to run in any time zone.

The second attempt to solve this utilised C#'s 'TimeZoneInfo' class which allows for the conversion of times to different timezones. I later found that due to the nature of the the Monodroid compiler that allowed C# to be transformed into Java based code, no regional information was available and the 'TimeZoneInfo' class was unable to function correctly.

I did not find a solution to this besides manually offsetting the between UTC and UTC + 1 when times are sent and received. This will not work forever, as when the phone's time zone changes from 'British Summer Time' back to 'Greenwich Mean Time (GMT)' the offset will still be in effect. However, as this was a prototype, this bug was not a priority.

5.11 Notification Feature

The last feature of the application implemented push notifications. The need for notifications was important for reminding patients about their appointments and providing live communication with them. Push notifications utilised the Google Cloud Messaging API to send the message and Android Services to process the messages as they are received.

Google Cloud Messaging was chosen to provide push notifications due to it's simplicity to implement, as well as being optimised for battery efficiency and poor bandwidth.

5.11.1 GCM Service

In order for the Android device to receive GCM messages, it must start a service that runs as a background process on the device. This service registers the application with the GCM Service that runs natively on all Android devices. Upon registration, a device registration id is provided which is used to determine which device to send the message payload to. This registration id must therefore be sent to the server and stored in the patients data model so that the server can successfully send messages to the correct device.

Besides device registration, the background service is also responsible for handling messages that are received from the GCM Server. Upon receiving a message, the 'OnMessage' method is triggered in the service with the resulting payload. This method adds the notification to the notification list in the home screen and shows creates a local notification to inform the user that they have a new message.

5.11.2 Sending of GCM messages

In order to create a GCM message, a rest request must be sent to the GCM servers. The request must contain the Android application's API key, the registration id of the patient's device and the message payload.

The API key is retrieved the same way as the Google Maps API key was retrieved, through the Google Developer Portal. This informs the GCM Service on the Android device which application the message is related to.

The registration id is sent via a web request from the Android app and the Account Component assigns this id to the relevant patient model. This means that no notifications can be sent to the device until it has retrieved this request from the Android client.

The message payload contains any data that is sent to the device, such as the notification message and the relevant appointment id. Different types of notifications are therefore created easily by specifying custom data in the payload.

5.11.3 Observer

Another important feature of sending notifications was the scheduling of them. Notifications need to be sent at specific times to remind patients about various aspects of the appointment. To do this, an Observer service was implemented on the Asp.Net server that runs in the background. This utilises the Timer method in the Observable class to execute notifications at a set time.

This allows for reminder notifications to be queued easily as an appointment is scheduled. The prototype automatically schedules a notification to be sent an hour prior to an appointment's start time.

Chapter 6: Testing and Evaluation

6.1 Introduction

This chapter will focus on the testing and evaluation of the solution implemented. In order to evaluate how suitable the solution is at fulfilling the requirements, user based assessments were performed to collect feedback.

6.2 Testing

Before the assessments could begin, the application needed to be tested thoroughly so that the it would perform correctly and the feedback from the assessments would be most valuable. The testing was split into two sections, unit testing and functional testing.

6.2.1 Unit Testing

Unit testing is a form of automated software testing that individual executes portions of the code and validates whether the output is desirable or not. Unit testing is very useful in iterative based development where testing needs to be repeated often to ensure older features do not get broken by newer ones. This is a common issue in software programming, especially when many developers are working on the same code base and don't fully understand the entirety of it, resulting in undesirable changes and side effects commonly occurring.

Although some unit tests were written for this project using the 'Visual Studio Unit Testing Framework', I decided that the time cost for implementing the tests was not worth the output, especially as the project had a fairly small code base and I was the sole developer. Most of the features were also created independently of each other, resulting in a smaller risk of them modifying each others functionality.

6.2.2 Functional Testing

Functional testing is a requirement in any software development project to ensure it is ready for release. Functional testing is the manual act of testing the functionality of an application, just like an end user would. This helps to identify functions of the application that are broken or poorly designed, ensuring that these functions do not make it to the end user.

6.2.2.1 Testing Methodology

In order to test the app's functions, an initial regression test was used. Regression testing is a form of functional software testing that seeks to uncover software bugs within certain features of a software application. An initial regression test includes all features implemented in the prototype. As extra features are added at a later date, new regression tests must be made for these features. Only once all features are tested and pass can the application be signed off for release.

To start regression testing, an extensive list of all the app's features was made. This test list comprised of the following information:

- Test Number - A unique number for easily identifying the test.
- Feature Category - The category that the feature belongs to.
- Feature Type - The type of feature it is.
- Action - A description of feature and its desired effect.
- Pass/Fail - Whether the test was successful or not.
- Comments - Reason as to why the test failed or any improvements that could be made.

With the test list formulated, each feature was tested multiple times to determine if they should pass or fail by comparing the test action to the actual result.

6.2.2.2 Results

Appendix C shows the results of the initial regression test. The initial results showed that although the majority of the app was working as intended, the notification feature category was completely broken and needed fixed prior to the user based assessments.

6.2.3 Further Testing

Although aspects of the mobile application were fully tested, most of the testing was focused on the user interface and testing of the server was ignored. Further testing must be done to analyse how the system would perform in a production environment. Due the the system being a prototype, it was not necessary for this project.

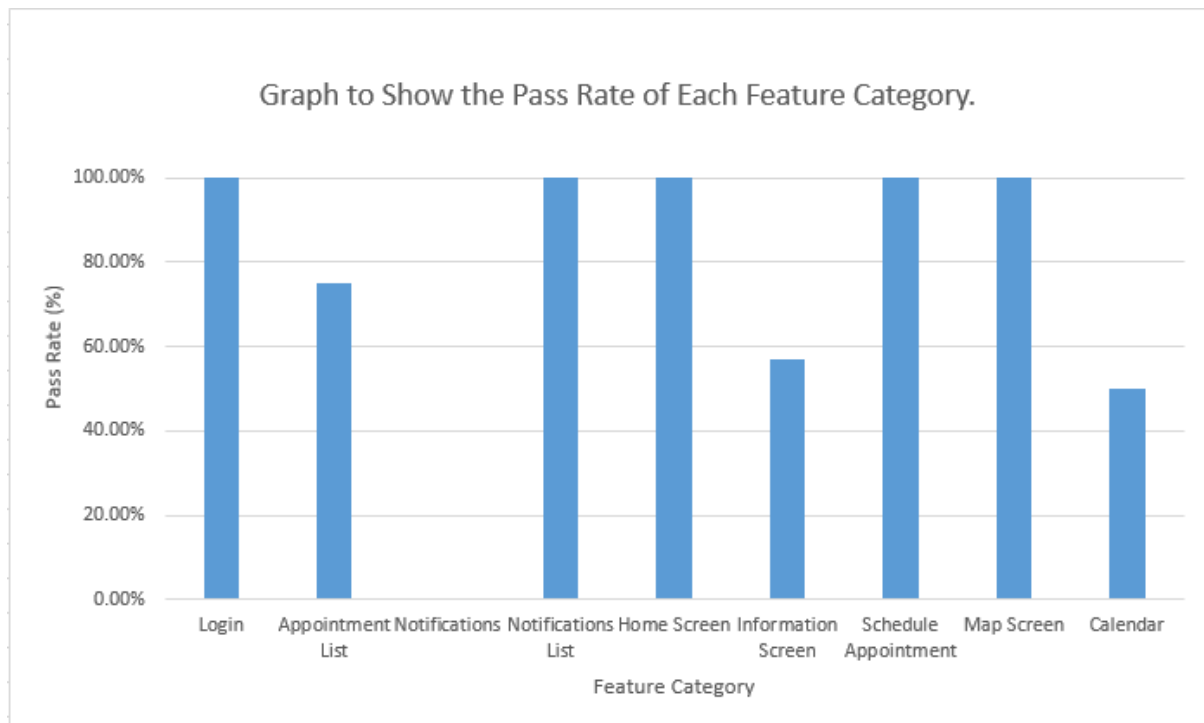


FIGURE 6.1: Graph to show the pass rate of each feature category.

6.2.3.1 Load testing

In order to assess how many concurrent users can successfully use the system before issues occur, load testing must be performed. This is a process where test accounts are created and accessed at the same time, which is repeated until performance or other issues begin to occur. This helps to prepare for user demand and outline efficiency issues or the hardware requirements for the system prior to it being launched.

6.2.3.2 Concurrency Issues

One hypothetical issue came to mind when implementing the prototype; what happens when two separate users schedule the same appointment at the same time. Although the system is designed to be flexible enough such that these issues would not have a fatal effect on the system, it would likely create an error that is not currently handled by the client application. Many issues like this could exist, and features would have to be tested thoroughly to discover them all.

6.2.4 Conclusion

With the test results collected, any failures were investigated and the specific action was re-implemented. Final regression tests were repeated until all failures were passed and the prototype was ready for evaluation.

Further performance testing and analysis of the applications resource usage could also have been performed, but slow/poor performance was not picked up by the functional testing and was therefore deemed unnecessary.

6.3 Evaluation

After testing was completed and the app was in a stable state, the evaluation could commence.

As stated in Section 2.5, I wanted to evaluate the solution based on the following questions:

- How easy to use is the mobile application?
- Does the solution work? Does it fail occasionally?
- Is the solution missing key features?
- Would patients find the mobile application useful?
- Does the solution reduce wasted appointments?
- Does the solution reduce staff resources required for managing appointments?
- Does the solution increase the user experience when managing appointments?

In order to evaluate these questions, three methods of evaluation were proposed, user observation, user feedback and trials in real hospital environments.

6.3.1 Evaluation of Methodology

Whilst designing and implementing the prototype, it became apparent that the proposed solution was not ideal. Firstly, the solution could wrongly encourage patients to reschedule appointments when it is unnecessary. This would, without a doubt put strain on the system, making it hard for medical staff to constantly adapt to changes in the schedule.

This is however a hypothetical problem. When patients are faced with the probable long waiting times that they would in an hospital environment, it would be undesirable for them to reschedule appointments often. Trials of the proposed system would have to be run to determine whether the additional strain would exist or not.

Another problem the solution faces is that it would not be able to replace the current system entirely, as not all patients have access to smart devices and internet access. The system would have to be integrated alongside other systems in place, which could prove challenging with clinics that still use paper based approaches or rely on staff members for appointment management. As the number of patients without internet access is a growing minority, this may not be as problematic as first thought. Also, as the solution is designed to be platform independent, web interfaces could be set up in clinics and hospitals to accommodate for this, or even dedicated staff members to manage appointments through the system on a patient's behalf.

6.3.2 Controlled Trials

In order to evaluate the effects of the solution on staff resources and appointment wastage, trials would need to be carried out in real clinics and hospitals. This would produce statistical data that could then be compared with a control group who use the standard methods of appointment scheduling available today.

Unfortunately, it was not possible to conduct clinical trials for this project due to insufficient resources and the time restrictions involved. Despite this, I was able to conduct the user evaluation successfully and collected good feedback to evaluate the user experience of the application.

6.3.3 User Observation

User observation was carried out on a ten willing participants who were given a sheet with a list of tasks that they had to attempt (see Appendix E). Tasks were deliberately designed to be vague, not offering any instructions on how to complete a task. By designing the tasks this way, it demonstrated the applications usability and aimed to identify areas that were not intuitive.

Applicants were also provided with a test account with three hypothetical appointments already set up. The first two appointments were unscheduled to demonstrate the scheduling functionality of the app, and the third demonstrated the synchronisation with the phones calendar feature.

From observing the entire applicant group, none had issues completing all of the tasks given. Feedback was generally very positive for most of the features except the scheduling process.

Some applicants showed signs of annoyance when trying to get the desired appointment which required a trial and error approach of inputting times and seeing if they were taken. Although the system offered a selection of choices close to the input time, this was not desirable for a user with a specific set of dates and times that they were available on.

Applicants also queried for a way to manage notifications, turning certain notifications on and off and highlighting a useful extension that could be added in the future.

6.3.4 User Questionnaire

The user questionnaire was designed (see Appendix F) to collect useful feedback and carried out after the user observation on the same participant group. The questionnaire consisted of numerical ratings of the mobile application and the following graph was composed from the average ratings provided from all ten applicants:

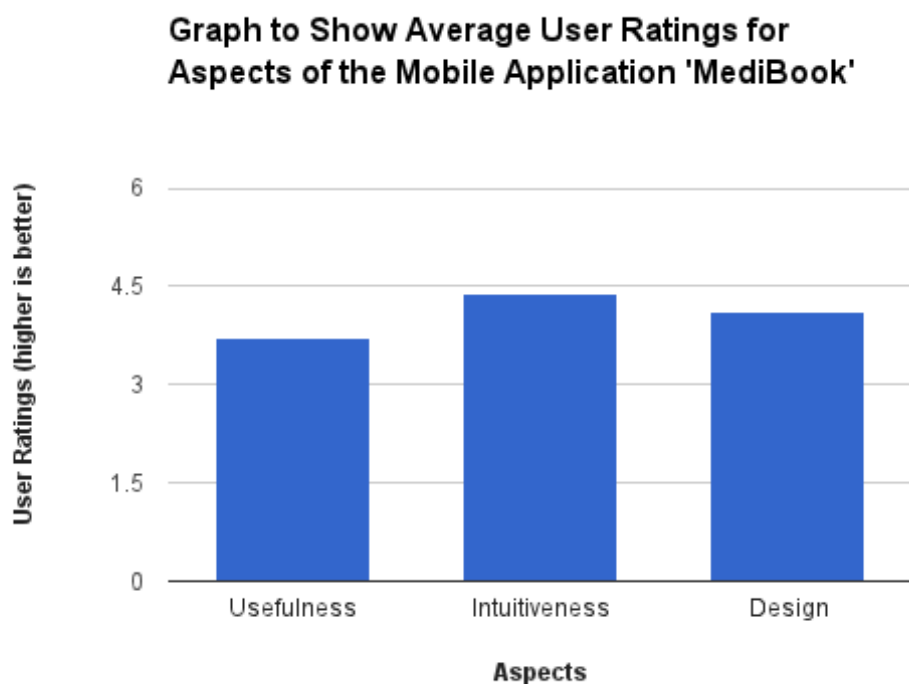


FIGURE 6.2: Graph to Show the Average User Ratings Collected from the User Questionnaire

The questionnaire also provided ways for the applicants to give feedback and suggestions. Some useful feedback included the suggestion of disabling unavailable times from the app's time picker fix the trial and error approach currently implemented. Some users also suggested possible future extensions such as using the notification feature for scheduling a patients medication, also providing them with prescription information and pharmacies located near the patient.

6.3.5 Further Evaluation

Besides the methods discussed above, many aspects are missing from the evaluation due to time restrictions and further evaluation could be performed in the future.

6.3.5.1 Training

Although the user evaluation showed that the app was intuitive, all participants admitted to owning their own smart device and had general knowledge on how to operate an Android application.

Further testing could be performed on a test group consisting of people who are not technically proficient. This would give a better analysis on training requirements and would help outline documentation and a manual for the applications operation.

6.3.5.2 Comparison with Phone Appointment Booking

As identified in the problem description, one of the main methods for booking a medical appointment currently is over the telephone, so further evaluation could be performed to identify the strengths and shortcomings of the prototype when compared with a telephone booking.

Participants would be asked to first book an appointment over the telephone, and then book the same appointment through the mobile application. They would then be interviewed on their findings. Some useful questions could be asked to compare the mobile app with telephone appointment bookings:

- Is it faster to book via mobile?
- How many telephone staff members were required to fulfil demand?
- How many staff members were required to maintain the mobile application?
- Was more information available by telephone?

6.3.6 conclusion

Based on the findings obtained by the user evaluation, the prototype application can be judged as a success in terms of providing a good user experience. It is easy to use, did not fail during the user evaluation and participants found that it was useful. Although it was missing some

features such as disabling unavailable times from the time picker, these could be implemented easily in the future.

However, although the user evaluation was successful, many other observations required for the evaluation of this system depend upon real world scenarios. More testing and evaluation of the system being used in real hospital and clinic environments would have to be executed before it could be deemed a success.

Chapter 7: Project Conclusion

7.1 Summary

This project aimed to design a prototype system that helped automate the booking and management of medical appointments. The project achieved this by providing direct communication links

7.2 Limitations

7.3 Future Extensions

By increasing communication with the patients, many useful extensions could be investigated in the future that were not feasible given the time restrictions of this project.

7.4 Final Conclusion

Appendix A: Personal Reflection

When first faced with my final year project, I was overwhelmed with choice. Throughout my university career, I have rarely had the freedom to choose precisely what I wanted to learn. It is therefore very tempting to pick an area that you already know well, with very little learning benefit.

To overcome this, I set out a list of personal goals that I wanted to achieve by doing a project, which helped me narrow my choices:

- I wanted to learn how to build mobile applications.
- I wanted to learn how to build an adaptable, scalable server.
- I wanted to provide a tool that would potentially help people.

It became clear after reading through project choices submitted by lecturers that none of them would help me obtain my goals. Thankfully, with the help of my supervisor, I was able to draft out a suitable project that was suitable enough to achieve both the universities and my own goals.

The second challenge I faced was project planning. I was not used to the lack of deadlines, and it took a long time for me to get started on my project. I overcame this problem through the use of personal deadlines. Although creating a Gantt chart helped keep track of the project's progress, I did not keep to it initially. I discovered the hard way by falling behind, and I spent many painful hours catching up to my initial schedule.

Another issue I faced with the project planning was encountering entire tasks. Although I had split my project into sub tasks such as implementation, it seemed very daunting when first facing them. To overcome this, I used project management tools like 'Trello' to split tasks down into subtasks, leave comments that I could remember and plan features out. This helped overcome the initial stress involved when planning specific parts of the project, making todo lists and remembering issues that occurred for when you write the report. I found this far more valuable than keeping a journal as the user interface is very flexible and appealing.

The aspect of the project I enjoyed the most was the implementation. Although I had outlined things that I wanted to learn by doing this project, I learned far more during implementation

due to issues that I had not even considered. An example of this was my problems with time synchronisation across multiple regions around the world, requiring me to look into solutions to this and learning about the various practices involved in geographical programming.

Besides implementation, this project has taught me invaluable skills that I will no doubt be utilising for years to come. I learnt how to formulate a report, conduct evaluation on research literature, software testing, communicating with various entities to get project resources, user based evaluation and many more skills involved.

Ultimately, my final year project was the highlight of my degree for me. It has been the greatest learning opportunity and challenge that I have faced in my university career, and I am grateful for having the opportunity to participate in it.

Finally, I will summarise my findings to compose this list of recommendations for future students:

- Don't pick a project that sounds easy, choose what **you want to learn first, then find a project that fits your learning goals.**
- **Make personal deadlines to achieve project tasks on time. Plan accordingly in advance!**
- Use tools such as Trello to plan your implementation, write down any issues you encounter so that you can discuss them in your report.
- Plan your evaluation in advance, make sure you've finished your implementation with plenty time to spare!
- Backup your project in multiple places, Github and dropbox are great tools!
- Enjoy your project and make it what you want it to be.
- Give yourself extra time to deal with unforeseen errors, they are a certainty.

Appendix B: External Resources and Materials

- Use of the Xamarin Framework to develop the Android application obtainable at:
<http://xamarin.com/>
- Use of the ASP.Net Framework to develop the Server application obtainable at:
<http://asp.net>
- Use of the Entity Framework library to create and manage SQL databases in C#, available at: <http://msdn.microsoft.com/en-gb/data/ef.aspx>
- Use of the Rest Sharp library for the creation of RESTful web requests and the deserialisation of response data, available at: <https://github.com/restsharp/RestSharp>
- Use of the GCM Client library for integrating GCM with Xamarin, obtainable at:
<https://github.com/Redth/GCM.Client>
- Windows Azure was used to host the server application and the database, available at:
<http://medibook.azurewebsites.net/>
- Use of the Masters/Doctoral Thesis Latex Template for the creation of this report, obtainable at: <http://www.latextemplates.com/template/masters-doctoral-thesis>

Appendix C: How ethical issues are addressed

C.1 Introduction

This section describes the ethical issues involved throughout the project. I also propose techniques for solving and minimising these issues.

C.2 Project background

A few ethical issues arise from the projects aims and objectives. Firstly, personal data will be used and stored in order to optimise the projects aims. Secondly, the project aims to optimise the scheduling process, which could make some employees jobs redundant.

C.2.1 Personal Data

Personal data will be stored and used both locally (on the mobile device) and on an external server to try and optimise scheduling software. It may also be required to transmit this data regularly to keep the system running effectively.

This brings into security issues, as the data could be very valuable to certain individuals for marketing or other purposes. In order to solve this, several areas should considered:

- Patients must be informed about what data is being stored and why
- The system should be as secure as possible with the current technology
- The data should be as anonymous as possible, with no individual besides its owner having access to it.

C.2.2 Employee Downsizing

The system aims to optimise the scheduling process and would replace a lot of the manual labour involved. It is therefore possible that the system would make some employees redundant.

Although this system aims to optimise the current system, it would not be able to replace it entirely, and so not all jobs would be lost. Also, this is only a short term problem, and with more training, employees could be allocated elsewhere.

C.3 Testing and Evaluation

In order to evaluate this project, I plan to perform user based assessments.

C.3.1 User Based Assessments

When performing user based assessments, care must be taken to ensure that participants are well informed of their rights, what the project is about and how their feedback would be stored and used. The participants data would also need to be stored anonymously to preserve confidentiality.

To ensure this is done correctly, the following steps will be taken:

1. The nature of the project will be explained to the user.
2. The user will be informed that their opinions will be used and stored both anonymously and securely.
3. The user will be informed that they can stop the assessment at any time
4. The user will be asked to sign a consent form to confirm that they have agreed to take part and that their data can be used to evaluate the project
5. The data will be collected and stored anonymously

Appendix D: Regression Test

Test #	Feature Category	Feature Type	Action	Pass / Fail	Comments / Fault Description
1	Login	Login Button	Logs the user in and progresses to the appointment list screen	Pass	
2	Login	Register Button	Registers the user using the entered username and password, logs the user in and progresses to the appointment screen	Pass	
3	Login	Username/Password Text-Field	Tap on either field brings up the android keyboard. Input is correctly passed to the field and used as login credentials.	Pass	Keyboard covers the buttons partially, however pressing back hides the keyboard
4	Appointment List	List	Downloads correct appointments for the user and displays type, status and scheduled time.	Pass	
5	Appointment List	List Item	Tapping on a list item should open the appointment information screen.	Pass	Colour should match buttons so patients know to click on items.
6	Appointment List	Refresh Button	Tapping re-downloads appointments. Shows animation while downloading and correctly resets when done.	Fail	Shows on the notifications tab.
7	Appointment List	Logout Button	Logs the user out returns to the login screen	Pass	
8	Notifications	Notification	Device Receives notification from server	Fail	Sometimes notification is not received.Cause: The device Id is sent to the server after some notifications are sent resulting in them not to be delivered.
9	Notifications List	List	Notifications	Pass	
10	Notification / Appointment List	Tab	Tapping on the tabs switch between notification and appointment lists.	Pass	
11	Information screen	Doctors Information	Loads a portrait image, name and type of doctor.	Pass	Doctors image sometimes slow to load.
12	Information Screen	Doctor/ Clinic Number	Shows correct number for the doctor or clinic and tapping on it opens phone dialer	Fail	Tapping doesn't open dialer.Cause: Missing button Id.
13	Information Screen	Map Button	Shows the map screen	Pass	
14	Calendar	Add to Calendar Button	Only visible if appointment is not added to calendar. Disabled if appointment is not scheduled. Tapping adds to calendar	Pass	
15	Calendar	Remove from Calendar Button	Only visible if appointment is added to calendar. Tapping removes from calendar	Fail	Does not remove from the phone's calendar correctlyCause: Incorrect SQL selection query
16	Information Screen	Schedule Appointment Button	Shows button text as re-schedule if appointment already scheduled. Tapping button opens scheduling screen.	Pass	
17	Information Screen	Cancel Appointment Button	Cancels the appointment	Fail	Cancels the appointment correctly however does not update the appointment list which still shows it as scheduled.
18	Information Screen	Map Button	Opens the map screen	Pass	
19	Information Screen	Scheduled Time	Shows the appointment time	Fail	Shows the incorrect time (1 hour before).Cause: Time is sent in UTC format and is wrong time zone.
20	Map	Map	Shows appointment Location	Pass	Only works in release.
21	Schedule Appointment	Schedule Appointment	Schedules appointment correctly, giving three options if time is taken and allowing user to specify a time.	Pass	

Appendix E: User Evaluation

E.1 Introduction

The purpose of this evaluation is to determine the usability and success of the mobile application 'MediBook'. You are asked to carry out all of the tasks listed below and then answer a short survey to describe your experience. Your responses and feedback gained from this survey will aid in the evaluation of the project. You will also be observed during the evaluation to collect further data.

All data collected from this activity is confidential and your identity will be kept anonymous. Thank you for agreeing to participate in this evaluation.

You will be given a test account with appointments already setup for you. Once you have received this information, please carry out the following tasks.

E.2 Task List

1. Find the MediBook icon in the app library and open the app.
2. Login with the test account and find the appointment list.
3. Try to find the name of the location where the first appointment occurs.
4. If you have found this location, try and open it on the map screen.
5. Return to the appointment list.
6. Select the second appointment and try to schedule it for 15:00pm today.
7. If you successfully scheduled the appointment, try and find the message displayed in the notification you just received.
8. Return to the app and select the third appointment. Try and add it to the phone's calendar.
9. Check that the appointment is successfully inserted into the phones calendar.
10. Return to the app appointment list and select the logout button.
11. Once returned to the login screen, close the app.

Appendix F: User Evaluation Questionnaire

1) Are you satisfied with the current methods of managing medical appointments?

- 6 - Extremely satisfied
- 5 - Moderately satisfied
- 4 - Slightly satisfied
- 3 - Slightly dissatisfied
- 2 - Moderately dissatisfied
- 1 - Extremely dissatisfied

2) How do you currently manage your medical appointments?

3) Do you have access to a smart device of your own with internet access? (Please circle those that apply)

- I have access to an Android device.
- I have access to an iOS device.
- I have access to a Windows Phone device.
- I do not have access to a smart device.

4) How easy to use was the application? (Please circle one)

- 6 - Extremely easy
- 5 - Moderately easy
- 4 - Slightly easy
- 3 - Slightly difficult
- 2 - Moderately difficult
- 1 - Extremely difficult

5) How useful would this application be to you in the future? (Please circle one)

- 6 - Extremely useful
- 5 - Moderately useful
- 4 - Slightly Useful
- 3 - Not very useful
- 2 - Rarely useful
- 1 - I would never use it

6) How pleasing is the design of the application (Please circle one)

- 6 - Extremely pleasing
- 5 - Moderately pleasing
- 4 - Slightly pleasing
- 3 - Slightly displeasing
- 2 - Moderately displeasing
- 1 - Extremely displeasing

7) Did you have any problems carrying out any of the tasks?

8) Do you think the application's reminder notifications would help you fulfil appointment requirements?

9) Which features did you find useful?

10) Was the application missing any features you would find useful?

11) Do you have any suggestions as to how the application could be improved.

Appendix G: Project Resources

G.1 Code and Mobile Application Download

The project code can be found at the following address:

<https://github.com/andrewmunro/Final-Year-Project/tree/master/Prototype>

The mobile application apk can be downloaded at the following address:

<https://www.dropbox.com/s/4lzrs7gnjauw7sj/MediBook.Client.Android.Release.apk>

The hosted server application can be found at the following address:

<http://medibook.azurewebsites.net/>

G.2 Mobile Application Screens

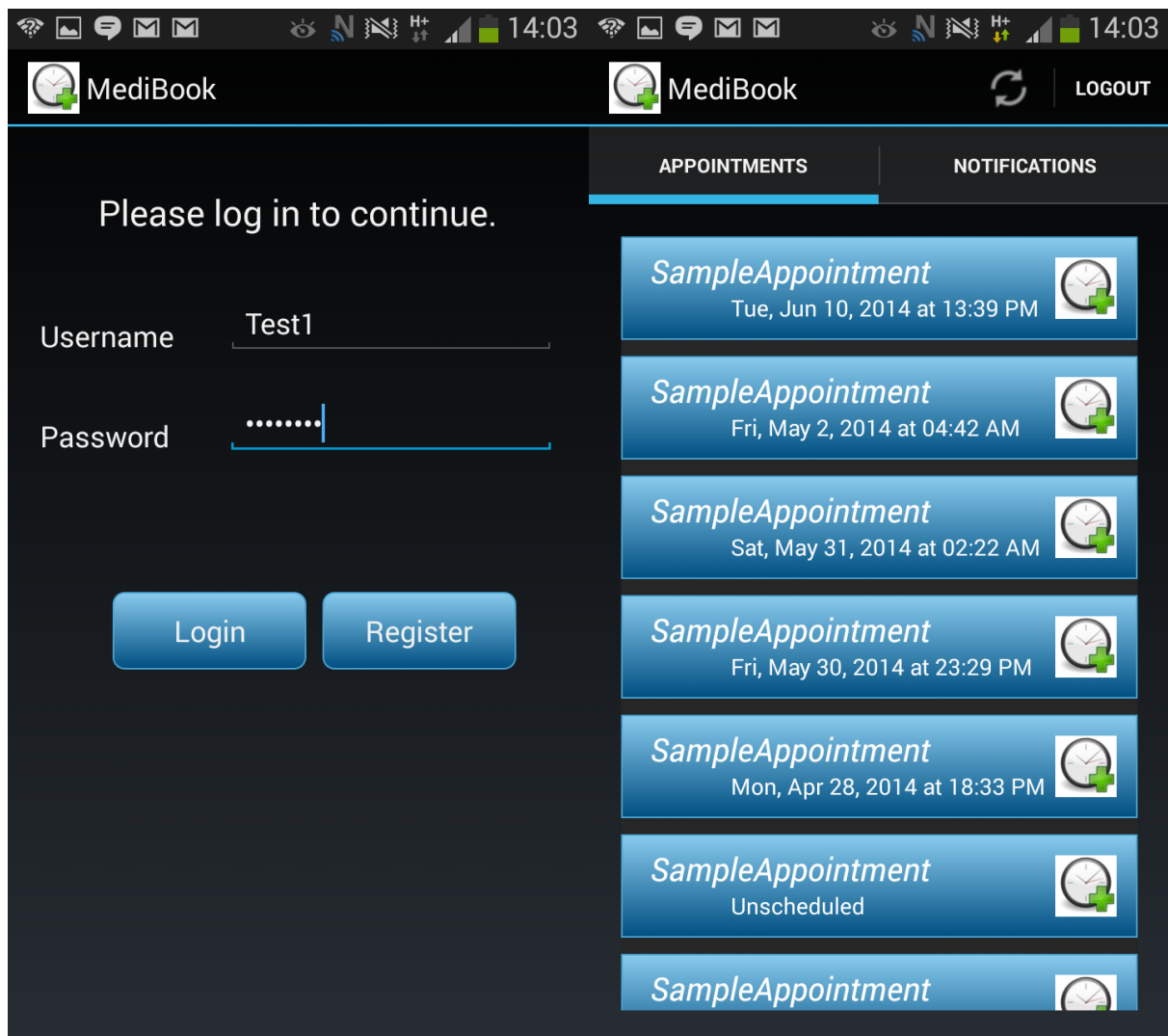


FIGURE G.1: Login Screen (Left), Home Screen (Right)

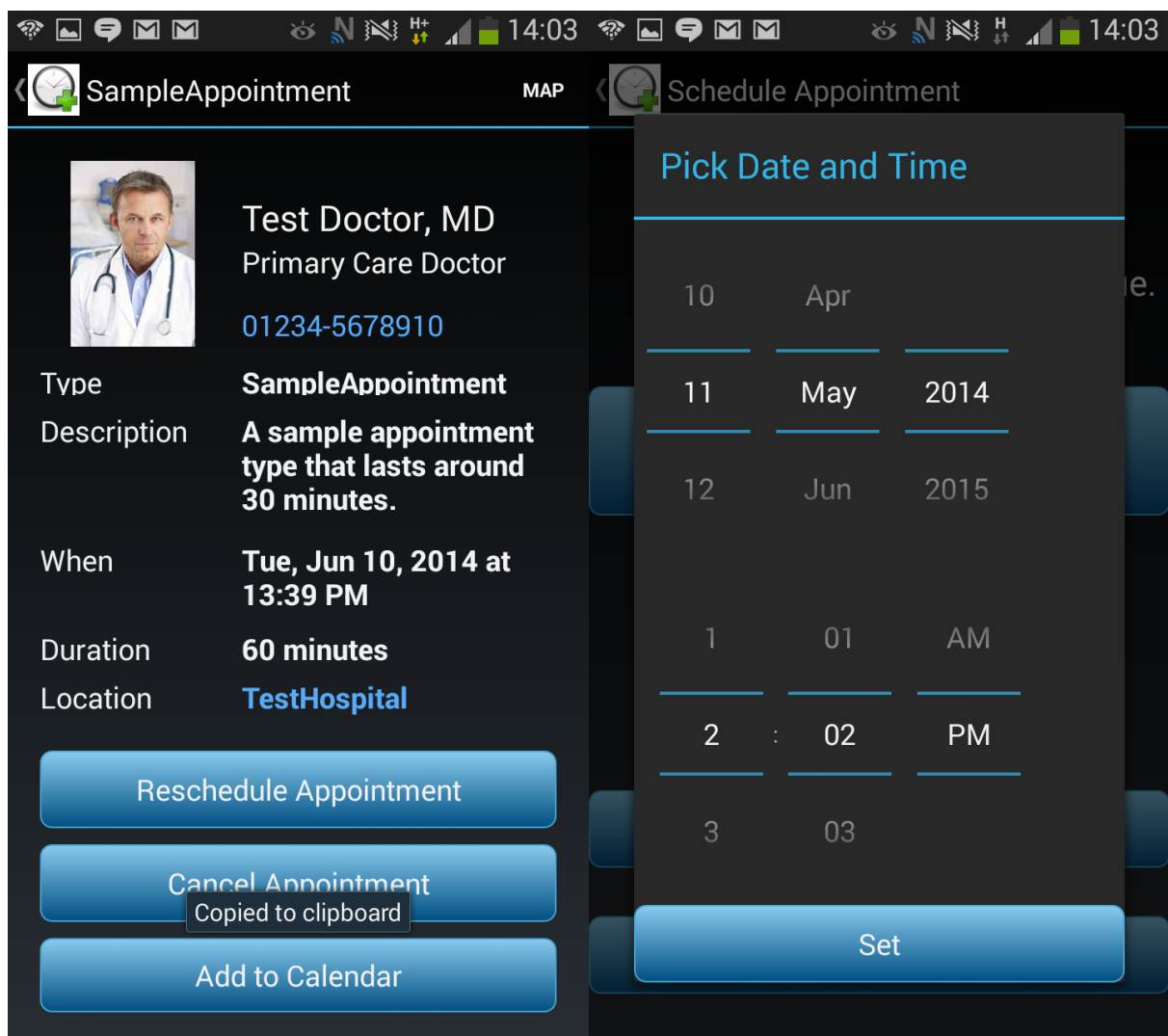


FIGURE G.2: Appointment Information Screen (Left), Appointment Time Picker Screen (Right)

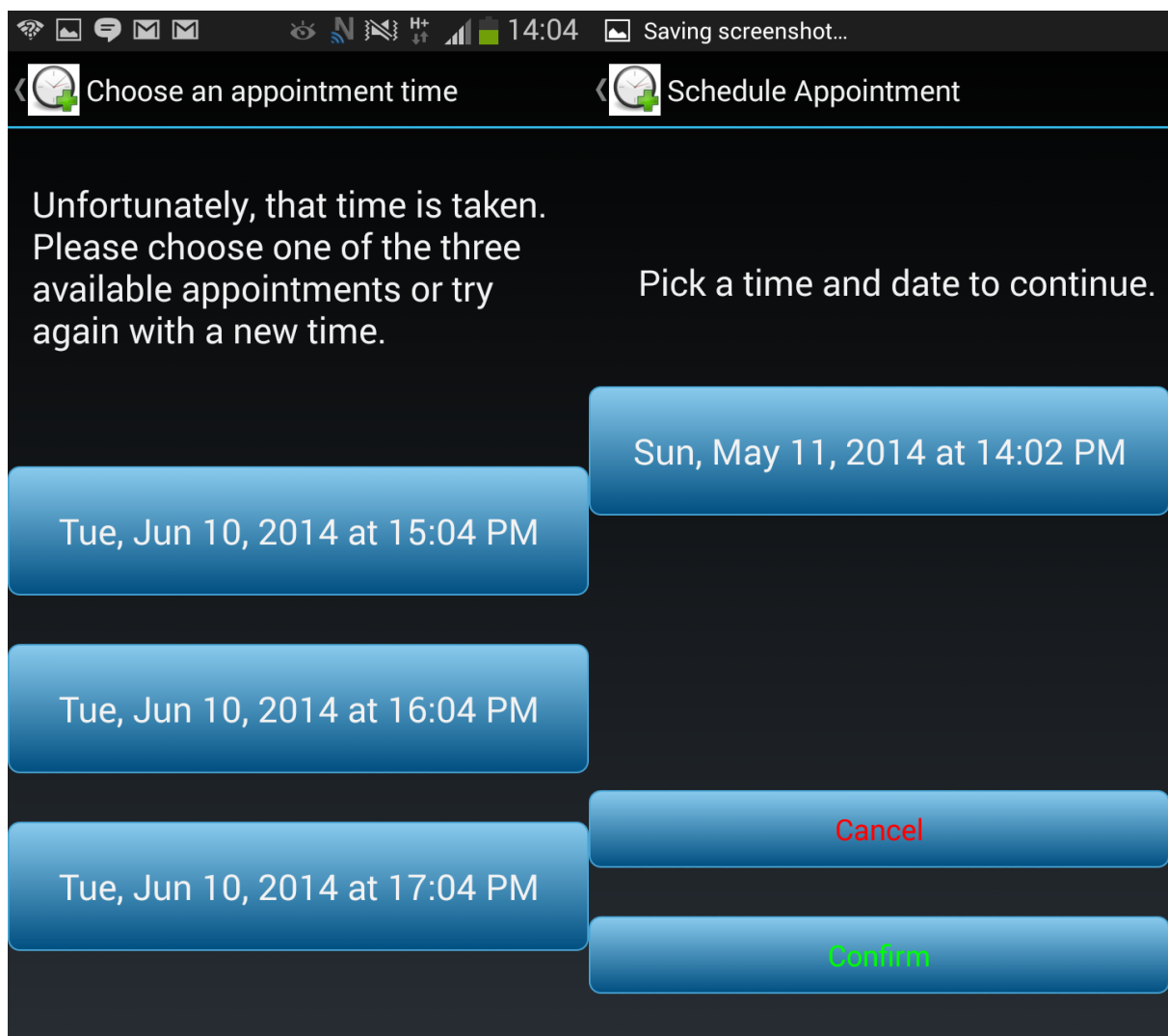


FIGURE G.3: Conflicting Appointment Screen (Left), Confirm Appointment Time Screen (Right)

Bibliography

- [1] Gerard Gallucci, Wayne Swartz, and Florence Hackerman. Brief reports: Impact of the wait for an initial appointment on the rate of kept appointments at a mental health center. *Psychiatric Services*, 56(3):344–346, 2005.
- [2] Peter J Saxby Christopher A Stone, John H Palmer and Vikram S Devaraj. Reducing non-attendance at outpatient clinics. *Journal of the Royal Society of Medicine*, 92:114–118, March 1999. URL <http://jrs.sagepub.com/content/92/3/114.full.pdf+html>.
- [3] McWhinnie DL Mann DV. A simple extended-cavity diode laser. *Appointment Reminders*, 303:1526, December 1991. URL <http://dx.doi.org/10.1136/bmj.303.6816.1526>.
- [4] Stephen Frankel, Alexandra Farrow, and Robert West. Non-attendance or non-invitation? a case-control study of failed outpatient appointments. *BMJ: British Medical Journal*, 298(6684):1343, 1989.
- [5] Naomi L Lacy, Audrey Paulman, Matthew D Reuter, and Bruce Lovejoy. Why we don’t come: patient perceptions on no-shows. *The Annals of Family Medicine*, 2(6):541–545, 2004.
- [6] NHS. Productive general practice: Plannning and scheduling, Febuary 2014. URL http://www.institute.nhs.uk/productive_general_practice/general/planning_and_scheduling.html.
- [7] NHS. Choose and book, Febuary 2014. URL <http://www.chooseandbook.nhs.uk/patients/whatiscab>.
- [8] NHS. Choose and book use drops further, Febuary 2014. URL <http://www.ehi.co.uk/news/ehi/7835/choose-and-book-use-drops-further>.
- [9] NHS. Your hospital outpatient appointment, Febuary 2014. URL <http://www.nhs.uk/NHSEngland/AboutNHSservices/NHShospitals/Pages/hospital-outpatient-appointment.aspx>.
- [10] Alison J Macarthur, Colin Macarthur, and Joan C Bevan. Determinants of pediatric day surgery cancellation. *Journal of clinical epidemiology*, 48(4):485–489, 1995.

- [11] GOV. Health and social care act, January 2012. URL <http://www.legislation.gov.uk/ukpga/2012/7/enacted>.
- [12] Tim Kelsey. Personalised care plans will give patients control of their own health, September 2013. URL <http://www.england.nhs.uk/2013/09/25/tim-kelsey-2/>.
- [13] Simon Walford. Choose and book. *Clinical medicine*, 6(5):473–476, 2006.
- [14] Judith Green, Zoe McDowall, and Henry WW Potts. Does choose & book fail to deliver the expected choice to patients? a survey of patients’ experience of outpatient appointment booking. *BMC medical informatics and decision making*, 8(1):36, 2008.
- [15] Gcm diagram, October 2012. URL <http://1.bp.blogspot.com/-5uC3H0ZQnvs/UIPzoWZduiI/AAAAAAAAACNM/qFuiBA7fpi0/s1600/gcm-image.png>.
- [16] Pietrino Atzeni. Rest diagram, July 2011. URL <http://di-side.com/di-side/services/web-solutions/rest-webservice-symfony/>.
- [17] Regis Frey. Mvc diagram, May 2010. URL <http://en.wikipedia.org/wiki/File:MVC-Process.svg>.