Ashwin Ahuja

Tech Support for the Elderly

AQA A Level coursework 2017

Table of Contents

[1 - Analysis 3](#_Toc477730001)

[1.1 - Background to and identification of problem 3](#_Toc477730002)

[1.1.1 - Elderly and Technology 3](#_Toc477730003)

[1.1.2 - The Millennials 5](#_Toc477730004)

[1.1.3 – Conclusion 5](#_Toc477730005)

[1.2 – Description of current solutions 5](#_Toc477730006)

[1.2.1 – Online forums 5](#_Toc477730007)

[1.2.2 - Professional Technical Support 6](#_Toc477730008)

[1.2.3 – Remote Technical Support 7](#_Toc477730009)

[1.2.4 – Conclusion 7](#_Toc477730010)

[1.3 - Identifying prospective end user(s) 7](#_Toc477730011)

[1.3.1 – Elderly 7](#_Toc477730012)

[1.3.2 – Teenagers 8](#_Toc477730013)

[1.4 - Identifying user priorities (interviews) 8](#_Toc477730014)

[1.4.1 - Gill Wilson 8](#_Toc477730015)

[1.4.2 – Philip Fernandes 10](#_Toc477730016)

[1.5 – Android vs iOS 10](#_Toc477730017)

[1.6 – Phones vs Tablets 11](#_Toc477730018)

[1.7 – Payment of helpers 11](#_Toc477730019)

[1.8 – Acceptable limitations 12](#_Toc477730020)

[1.9 – SMS System 12](#_Toc477730021)

[1.10 - Objectives for the proposed system 12](#_Toc477730022)

[Elderly System 12](#_Toc477730023)

[Teenagers’ System 13](#_Toc477730024)

[General 13](#_Toc477730025)

[1.11 – Explanations for objectives 14](#_Toc477730026)

[2 - Design 16](#_Toc477730027)

[2.1 - Introduction 16](#_Toc477730028)

[2.2 - Data 16](#_Toc477730029)

[2.2.1 - Storing Data 16](#_Toc477730030)

[2.2.2 - Communications with applications 23](#_Toc477730031)

[2.3 - WebAPI 23](#_Toc477730032)

[GENERAL STRUCTURE OF .NET CORE API 24](#_Toc477730033)

[OBJECT ORIENTATION 24](#_Toc477730034)

[2.3.1 - Communicating with SQL Server 24](#_Toc477730035)

[2.3.2 - Functions 24](#_Toc477730036)

[Securing data 29](#_Toc477730037)

[Verifying data entry 30](#_Toc477730038)

[2.4 - Android App 31](#_Toc477730039)

[2.4.1 - Important Features 31](#_Toc477730040)

[2.4.2 - Helpers 31](#_Toc477730041)

[2.4.3 - Elderly 36](#_Toc477730042)

[2.5 – SMS Server System 42](#_Toc477730043)

[Flowchart of entire system 42](#_Toc477730044)

# 1 - Analysis

## 1.1 - Background to and identification of problem

### 1.1.1 - Elderly and Technology

Today, the world is getting more and more connected by the internet, as people become reliant on the internet. For the first time, over 50% of people in the world (and over 80% of the western world) are connected by a single network. Ever since the CEO of IBM stated that the market for personal computers as around 5, people have attempted to prove him wrong with over 350,000 sold every year today. And alongside the rise of personal computers, mobiles and tablets are also continuing to expand, reaching an even greater group of people. And yet, the rise of the computer has left a generation behind, as elderly people today are termed the ‘generation that tech forgot’, living through a period of great technological advance, but ostracizes and isolated by it, as there was and continues to be very little technology specifically designed for them.



Figure 1: Internet use by age group, 2000 – 2012

As you can see from the graph, there is a stark difference between the number of people between the ages of 50 and 64 using the internet as opposed to the number above the age of 65. Yet, that number has increased vastly between 2000 and 2012, both as the demographic moves to include those who were once younger and as the technology became more wide-stream and the elderly attempted to use technology.

Yet, despite the changes in technology, only just over half the population over 65 use the internet. And this is despite all the major benefits that the internet can have for the elderly, from using Video Messaging services such as Skype to connect with relatives to using online diagnosis systems to help to diagnose any medical issues they may have. Yet, this is clearly not for lack of trying. Over 70% of the elderly, according to a YouGov poll, want to use technology and the internet. In fact, talking to a group of five of my elderly neighbours, one said ‘most of my friends and I do want to learn how to use technology but it’s very difficult’. Throughout my research, there were clearly two parts of the problem for allowing the elderly to use the internet, the first being teaching and the second being support. Much like at schools and universities we have lectures, in which material applicable to everyone is covered, and supervisions or tutorials, where the apparently personal and random issues that we each have are covered.

For the elderly and the internet, the first method appears to be well covered, with hundreds of organisations teaching the elderly how to use the internet, having classes in libraries and universities. In fact, according to a recent survey, over 30% of those over the age of 70 are in some form of technology class. And according to most, these are relatively successful as the majority of people appear to appreciate them – a recent AgeUK session I attended had members who mentioned that they had gone from knowing nothing to knowing how to communicate with their family in only a few months.

Yet despite these solutions for the first problem, the second appears harder to solve. From methods such as smaller classes to online forums which can answer questions and solve problems which the elderly are having, there appears to be no reasonable solution. How can the elderly receive personalised technical support, both for fixing the problems that they are having and teaching them how to use the technology they don’t understand?

Problems, as any computer scientist, or indeed anyone who has ever used technology has experienced, is simply a part of the problems associated with using technology in this day and age. Yet, young people today have the instinct to understand how the computers can break, from checking that the power wire is plugged in, to going onto the command line and debugging the wireless adapter of the computer. However, to the elderly, when learning has been shown to become more difficult, the process if far more complicated, as they are forced to consult a list of possible problems, looking at all various possibilities, inevitably excluding the actual problem that exists.

This is the purpose of tech support people, who should be able to solve all the various problems that the elderly have, easily. Yet, they are expensive, impersonal and often unreliable. Meanwhile, there is a vast array of people who have the requisite skills to solve problems without even realising it, and would be willing to help, yet don’t. In connecting these two groups, one can solve the second part of the problem for the elderly, therefore helping to allow the tremendous power of the internet to the elderly.

### 1.1.2 - The Millennials

Today, technology is engrained in the youth, as we rely upon Snapchat and Facebook Messenger to talk to friends, while using Wikipedia and the online Encyclopaedia Britannica to write essays for school. Through this, the group is highly technologically savvy. In fact, in a paper by Susan Herring of the Indiana University talks of the ‘technological divide’, where the majority of children, by the age of ten, are more technologically savvy than their parents. Meanwhile, by people of previous generations, the millennials are often thought of as the generation that don’t care. They claim that they are the generation that are so addicted to our phones and tablets that we don’t look up and see the problems that are being faced by large portions of society. Yet, the statistics fail to back it up, as more people are getting involved in social activism, more people are volunteering to help those in need, the elderly and infirm.

Yet, there is nothing that is currently harnessing the greatest skill that the youth do, that the elderly and in fact most adults struggle with, technological savvy. When taking an informal survey of a number of my friends, the vast majority indicated that they had been approached by elderly neighbours asking them to help out with fixing a broken computer or tablet. Moreover, they indicated that the neighbours complained that it was a regular occurrence which could easily be fixed if they had a manner of reaching willing technological savvy people who could easily solve their problem.

Even more importantly, they indicated that they would be willing to help their neighbours. While they indicated that they would probably want to be paid for their services, the conversation I had with friends also indicated a genuine interest in helping their elderly neighbours and society in generally, as this could help to create a closer community.

### 1.1.3 – Conclusion

Today, there is a genuine challenge for the elderly, who are already struggling to learn how to use technology, to find good technical support, as existing systems appear to be expensive or ineffective. Meanwhile, there is a genuine group of people, the teenagers, who have the requisite skills to help, and appear eager to do so. By attempting to connect the two groups of people, one can solve a couple of problems in one go, from the problem of how to get the youth more involved in society, exploring how best to use their skills, to how to ensure that the elderly get the technical support that they need.

## 1.2 – Description of current solutions

### 1.2.1 – Online forums

One of the primary methods for solving any technological problems, for the elderly and the rest of society alike, is through the use of online forums which claim to provide solutions in easy to understand English, as well as containing solutions to almost every problem that has existed. This immediately throws up a problem, as the elderly need to have access to an internet connection – which is often unlikely if the problem is related to accessing or using the internet. Additionally, they need a certain competency with the internet to be able to effectively search for and locate solutions to the problems.

Yet, even when they locate online forums, there are a number of problems with the systems. The forums can largely be split into two types – professional and amateur. In the professional forums, the authors of answers are largely professionals, often charge for visiting the site and are often run as a side service for companies which deal mostly with other issues – including professional technical support companies. One example of a ‘professional’ technical support forum is techsupportforum.com – a site where those giving answers are ‘IT professionals’. Yet, all the sites of this type share similar issues, that of ensuring that they cover sufficient material, addressing every problem that the elderly have. In fact, even looking up a basic router not working on techsupportforum yielded a single answer, suggesting that the setup was incorrect, without even suggesting simple, and often more likely, problems such as the router not being connected to power or to the telephone connection. Moreover, despite the best efforts of the authors the answers to problems were often complex and hard to follow without prior knowledge of the system – not providing simple enough instructions into how to solve the problem. This problem is even more prevalent in the second form of online forums, where they are largely community run, with answers being provided by people around the world with varying degrees of knowledge. Examples of these sites include r/techsupport (Reddit), StackExchange and even Yahoo Answers. Though they largely cover every possible problem that one might have, yet they generally fail to provide answers which are easy to understand for the elderly – with the answers largely being for the technologically savvy, with some prior knowledge of how to use technology.

### 1.2.2 - Professional Technical Support

Another possible current method of elderly gaining technical support is through professional services offered by many companies both local and global. They offer a much more personal experience than online forums, by the person visiting the elderly person able to offer a tailored service, ensuring that the problem gets fixed, as well as educating the elderly person how it has been done. Meanwhile, the personal contact ensures that the elderly person is able to ask all the questions they need to ensure that they understand how to fix similar issues in the future. However, there are also a number of drawbacks in this approach. Firstly, since the tech-support is run like a business, the people are incentivised to ensure that there are a number of problems – therefore not sufficiently showing the elderly how they fixed the problem, instead ensuring that they get repeat business if the problem reappears. In fact, talking to a neighbour who had used a Barnes based small tech support company complained that they charged ‘ridiculous’ amounts without actually fixing the problems that my neighbour had. Even for one problem that they did manage to fix, my neighbour claimed that they refused to explain the problem, instead claiming it was too complex and unworthy of explanation.

The issue of cost is an important issue with professional tech support, as they charge vast fees for services which are generally short and not too complex. I gathered information based on call-out fees and hourly rates for ten companies in the UK to show this was widespread.

|  |  |  |  |
| --- | --- | --- | --- |
| Company Name | Location | Call Out Fee | Hourly Rate |
| HTL London | Central London | £100 | £50 |
| IT Guy | West London | £75 | £50 |
| Andrew Tsai Technical Support | North London | £25 | £70 |
| Mitchell Wayne Technologies | Birmingham | £30 | £50 |
| D-Tech-IT | Liverpool | £0 | £50 |
| EverythingTech | Manchester | £60 | £60 |
| Grant McGregor | Edinburgh | £50 | £65 |
| AVERAGE |  | **£49** | **£56** |

Figure 2: Survey of cost of professional technical support around the country

Meanwhile, in order to show the fact that the use of the youth could help to solve this problem, I conducted an informal poll of ten teenagers who said that they would be interested in helping the elderly in my area, and they all suggested no call out fee, with an average hourly fee they would be willing to work for being in the vicinity of £10.

Finally, it is important to state that it is also often challenging to find people in your area as the vast majority of technically qualified people attempt to join jobs in industry and banks as opposed to offering technical support. Even finding eight companies across the country which offer technical support offered a certain challenge.

### 1.2.3 – Remote Technical Support

While remote technical support can be much cheaper than professional technical support, with no call-out fees and with much lower hourly fees as the people can be located in cheaper parts of the world such as India and China, there are a number of flaws with this system. Firstly, this often would be unable to solve some issues, especially those which are associated with hardware rather than with software. Furthermore, the people are often unable to show how they are solving the problem, as they have limited English skills and even if they are able to talk in English, they have little training or experience talking to the elderly, and so are ineffective at communicating the most salient points, instead puzzling their clients.

Another large issue with remote technical support is the problems associated with scammers, who instead of genuinely solving software problems on a computer can instead use the opportunity to install Trojans and other viruses. This would then allow them to have complete access whenever they want, gaining access to everything from people’s accounts as they log onto them on the computer to their webcam. This has led to a number of well-documented cases of identity fraud as the scammers gain enough information to imitate their victims, taking all their money.

### 1.2.4 – Conclusion

Though each of the three current methods of technical support for the elderly have flaws, it is clear that from an ideological perspective the second, professional in-person tech support is by far the best, as the human contact allows the elderly to both learn from the problems as well as have the problems fixed the most effectively. Hence, if it would likely be important to ensure that I mostly inherit from that method, though attempting to fix the problems associated with cost and lack of availability of tech support people.

## 1.3 - Identifying prospective end user(s)

From the initial problem, there are clearly two main groups of end-users who will both determine the objectives of the system which must be produced. Hence, it is important to consider them each in turn.

### 1.3.1 – Elderly

For the elderly part of the system, it was important that the end-user would use technology as well as either be elderly or communicate with the elderly on a daily basis. In the end, I considered the two main possibilities for the main elderly end-user of my product, either a group of my neighbours who I had previously interacted with about the problem, and who were currently attempting to learn how to use technology. Though by including a group of my neighbours, I could achieve a greater range of opinions than simply one person, the perspective was still largely the same, as people from the same street, attending the same lessons, of a similar age. It was therefore important to seek out a person who had more general experience of the problem who could represent the entire elderly community who were trying to use technology. I therefore contacted a number of charities which give technology lessons to the elderly attempting to reach a person who had experience in teaching them, and therefore was aware of the various problems that affected them. By reaching a person such as this, they would have experience of a wide range of my target market, while still having a technical perspective on the project in general. In the end, I contacted Mrs Gill Wilson, the IT Project Manager of AgeUK Richmond, who agreed to be the main end-user from the elderly perspective, ensuring that the user experience would be satisfactorily simple for a technologically challenged elderly person to user, while ensuring that it properly solved the problem of tech support for the elderly.

### 1.3.2 – Teenagers

The other aspect of the system is the teenagers part, where they can use the service to get work, solving technological problems of nearby elderly people. Thereby, it would be important that the system worked well for them, as otherwise few teenagers would signup rendering the system useless. In order to gain a teenager’s perspective, I interviewed Philip Fernandes, a technologically savvy 18 year old at St Paul’s School who would be interested in using my system to help those around him.

## 1.4 - Identifying user priorities (interviews)

### 1.4.1 - Gill Wilson

**What are some common problems that the elderly face when using technology?**

A number of the most common problems are problems which would otherwise be obvious to those who had grown up with technology, for example failing to plug in one wire or not double clicking on a shortcut to open the application.

**How do most of the elderly currently get tech support?**

While we encourage the people we teach to attempt to find solutions to their problems online, we find that a number of them are forced to seek professional tech support as the online forums are difficult or impossible to understand or they require a certain amount of knowledge to even access.

**How effective is the professional tech support?**

From talking to some of the people I have taught about their experiences in using tech support that they have got from flyers and posters they see around, they have had very mixed experiences, with some finding that they are very good. Some talk about how they are very attentive, help them solve their problem while teaching them how the problem arose and how to detect and fix it in the future. However, others complain of particularly poor tech support, where they charge exorbitant amounts to spend a long time and not even fix the problem that they were asked to fix.

However, a general problem that I find throughout talking to people is the relative difficulty of getting tech support. Unless they receive a flyer, they are forced to ask around and attempt to find people who can help. In fact, they often are forced to rely upon asking family or neighbours who are often busy or unable to help.

**Would the elderly be comfortable getting tech support from teenagers?**

I think in many ways the elderly would be more comfortable with teenagers than others, as they often can be the nicest people. I’ve found in talking to teenage volunteers, that they are among the most enthusiastic people to help out. Especially when the amount of money that stand to gain is minimal (or nothing), only those who want to help will actually do so. Even more so, I think they would be rather effective, they’d probably be able to fix the problems as well as a professional, since the problems are generally rather simple.

I think though, that there is also a certain comfort in professionals, where there are specific references and reviews that they have if the person has been recommended by a friend. If you can emulate that with your system, possibly by getting customers to review the teenager who helps them, I think you could reduce this problem.

**What is the best way for the elderly to interact with a technological system? (Mobile app, website, phone call, SMS)**

There’s a growing trend that people aren’t really bothering to learn how to use desktop or laptop computers, instead they are trying to learn how to use a smartphone or tablet. I think that you therefore need to have some mobile interface. Maybe it should just be a website which could be accessed from all devices, but I’d actually recommend mobile apps. They appear to be the easiest for the elderly to use – they have been finding them easier than anything else I’ve seen.

But, I’d also be aware that if people are having technological issues they may not be being able to use their phones or tablets. It might be worthwhile having an SMS system. Maybe they could send an SMS to a phone number and then automatically a person would be set up to be sent to their house. I think SMS is probably the most reliable thing that most elderly people are relatively happy being able to use, so if you can tap into this then that would be very useful.

**For the mobile apps, iOS or Android?**

Ideally, both. I think from talking to people around the area, while lots of people have iPhones, more have android phones. This is becoming especially more significant as there is a boom in cheap Android smarthphones, and elderly people are purchasing these devices. I would suggest starting off, therefore with Android, so more people are using it, but with a view to expanding quickly to iOS.

I think there’s, counter to what I would expect, mostly android tablets around the elderly. Things like the Hudl and other cheap Android tablets seem to have been especially popular with the elderly and they seem to be really taking to them. That being said, there are a large number of people with iOS devices, so if you could, I would suggest attempting to make a system to work with both.

**What are the major points that you would suggest in creating a user interface for the system?**

I think the most important thing by a long way is to keep it simple. Think, what do they really need. I mean all they actually need is a list of people, how far away they are and their reviews. And maybe a payment system to allow them to pay the teenagers through the app. Minimise how many screens they need to deal with and simplify each screen is as simple as possible.

Take it for granted that they don’t necessarily have the instinct of just trying things so try and make each screen very clear. What should they do to get from one screen to the other.

**With regards to payment, how do you think that the elderly would be most comfortable?**

I know that the teenagers would probably be most comfortable with payment over the internet and going directly to a bank account or something like that, I’d suggest that this would probably not work for the elderly who would anyway be unhappy buying things online, let alone paying some random website.

I’d therefore stick with cash payment, with them paying the teenagers directly. With regards to how much they should get paid, I think the main thing is being is as clear and transparent as possible. I don’t really think it particularly matters the amount, as long as it is cheaper than the competition, though this is not really challenging. I might suggest something like 15-20 pounds per hour, or something like that.

### 1.4.2 – Philip Fernandes

**Have you got any experience helping the elderly in your area with technological issues?**

I have a little experience of helping the elderly, mostly just helping my neighbours when their internet wasn’t working. However, I love the fact that your system can help me do it more often – I find that idea very valuable and powerful. I also think that you could potentially have a huge number of people in the teenage group who would be willing to help out, especially if they get paid for their work.

**What are the most important aspects of the system for you?**

I think it’s basically a system to manage all the people that are requesting their help and being able to message them and arrange a time when to work. It might actually be helpful to have a calendar, booking like interface for each person so and elderly person could see when I might be free and request to get support at that point – even if it is that evening. Therefore, being able to have the ability to input when I would be free would be important – as well as other stuff like how far I would be willing to travel.

**Would you prefer a mobile app or a website?**

I think I would prefer the freedom of having both options available – a website would be useful so I could check the status of things even if I don’t have my phone. However, I think a mobile app would be very important. Certain things like messaging people are just more natural on a phone. Also, I carry my phone everywhere, so being able to check on jobs from there would streamline my life.

**iOS or Android?**

Though I think having both an android and iOS app would be useful, I’d say the android app is the most important. Just from my observations, I’ve found that not too many techie people have iPhones in my age group, probably because of the cost.

## 1.5 – Android vs iOS

A primary decision in creating the system is considering whether to develop the application specifically for iOS or for Android. Though it could be possible in the future – as the product is rolled out and advertised more specifically, that an iOS application can be created, for now, it has been decided, in conjunction with the discussion with Mrs Wilson to concentrate on an Android application for the elderly.

For the teenagers, the decision was similarly made to focus on Android. For this, it is clear that the decision would have less of an impact on the system. Anecdotally from my friends who might be interested in being helpers for the system, most of the people own Android devices which could be considered.

It was also considered whether a web interface could be used to ensure that everyone, whether they own iOS, Android or own no mobile device, could be developed. However, it was decided that in fact, this is largely unnecessary as the elderly people would find the internet harder to use than an app for their phone. In fact, Mrs Wilson mentioned that today most people are choosing for their first forays into connected devices being phones or tablets, rather than desktops and laptops, as previously.

With the decision made to target Android, it is also important to consider the compatavility of the application produced. As Mrs Wilson mentioned that many of the devices that the elderly are using tend to be cheap, they are likely to use outdated versions of Android software. Therefore, it is important to ensure that the application can work with old versions of software.



However, it is also important to consider that newer APIs also introduce many important features which would make producing the app simpler. For example, only from Android Ice Cream Sandwich onwards (4.0+) is there inbuilt software to help with REST API requests, which is likely to be an important feature of the application. Since over 97% of Android devices would work on an application designed for Android 4.0+, this would be a good version to set as a minimum compatibility version.

## 1.6 – Phones vs Tablets

Another important decision to be made is whether to design the applications specifically for mobiles or tablets. Though Android would automatically make the application compatible with all screen sizes, if the application were to be specifically designed for tablets, the UI should be designed for this, taking advantage of the larger screen size.

Since, in fact, a large part of the considerations in designing the UI of the application for the elderly is that it is as simple as possible, containing as little information per screen as entirely necessary, this decision is less important. Therefore, since a greater volume of Android phones exist than Android tablets, it would be better to produce an application specifically for Android phones, with a possible route to expansion being to produce a tablet specific application. Specifically, the UI will be designed to fit best for a 4.7” device, the most common screen size of phones sold today (and incidentally the screen size of my phone, therefore ensuring it is easy to test).

## 1.7 – Payment of helpers

Another important aspect to be considered is how the helpers will be paid for their services. Though Phil mentioned how he would be much happier if the payment were to be done through the apps, Mrs Wilson was very clear that this would be less than acceptable for the elderly. Though it might be possible for a system in the future where a local conduit is paid in cash by the elderly and then the helper would get paid over the app, this adds unnecessary complexity and therefore is unnecessary. Throughout the process, the most important point is to ensure that the system is as simple for the elderly person as possible. Therefore, the payment should happen over cash, directly from the elderly person to the helper.

As per Mrs Wilson’s advice, the system should also have a £20 per hour standard rate. Since the people will largely be coming from nearby, they do not need a callout fee and can simply be paid from this hourly rate.

## 1.8 – Acceptable limitations

One important aspect which was ignored for the purposes of the project was the fact that each teenager might have special areas of knowledge, while may not be very competent in others. This was largely because it was largely felt that the project was mostly targeting the trivial problems often faced by the elderly which could be fixed by almost any technology literate teenager, therefore would be unlikely to stretch the skills of the individual. Additionally, reviews could easily cover the specific task that the person did, therefore allowing future elderly people to read that they are competent at that task.

Additionally, the fact that the helper probably has a maximum distance that they would be happy to travel has been omitted, for simplicity, instead allowing a system where the helper could cancel the appointment for any reason, including if they are unwilling to make the long journey. However, the elderly person should be able to see the location of the helper, which would both add a sense of trust, as they would know where they were (and therefore happy that they are not getting scammed) and so they can see how far away the helpers are.

## 1.9 – SMS System

The SMS system, a possible idea mentioned by Mrs Wilson, would allow an entirely separate path by which the helpers could create an appointment with a helper, simply sending a text with information which would allow the system to create an appointment. This would allow elderly people who may not otherwise be able to use the application, in case they are unable to get their devices working for example, they could still get assistance. This is because SMS is much simpler to use and therefore is likely to be understood by the majority of the elderly people, as they may also be using it to keep in contact with others.

To define this, the amount of data required to be sent by the elderly person needs to be considered. There were a few options, from defining the date, time and helper, to simply sending a text with any content and then the next available appointment would be created. While the former requires too much information, as the elderly person must already know who they want, the latter may be too simple, as the elderly person may not be free at that exact time.

Therefore, a compromise was created, where the SMS system could work off a date and time. If no date is defined, it should default to today’s date, assuming that the elderly person wants an appointment as soon as possible.

## 1.10 - Objectives for the proposed system

### 1.10.1 - Elderly System

1. **Input**
   1. There must be an Android app, which works with all Android phones with software later than Android Ice Cream Sandwich (4.0).
   2. The elderly person should also be able to create an appointment with an SMS messages to a specified phone number.
   3. The SMS should only require the user to send a Date and Time and the information about the appointment should be communicated directly back to the user over SMS.
   4. To log onto the Android application, the system should remember the password of the user once they have logged in once, thereby allowing them to not need to remember the password.
2. **User Interface (on all systems)**
   1. The screens should be as simple as possible, removing unnecessary information to other screens if necessary.
   2. As a maximum, there should be two possible (very different) paths to follow from every screen, with the difference in what each button does being marked clearly.
   3. All text should be large, above 12pt, to ensure that the elderly person can see it easily, with worse eye-sight.
   4. There should be a screen where the user will be able to scroll through the nearby teenagers who can provide tech support.
   5. This screen should be organised by how near the helper is, therefore meaning that the helper is unlikely to have to travel from very far.
   6. For each teenager, the elderly person should be able to see their average rating (if they have been rated), location and all of their reviews.
   7. The user should be able to look at the various appointments a teenager has free, allowing them to choose a time which would suit them.
   8. They should be able to book an appointment for any available time at least within the next two weeks.
   9. The user should be able to see the list of upcoming and past appointments, seeing the details about the appointment.
   10. For appointments in the past, the user should be able to rate and review the helper, therefore helping future elderly people to decide whether the helper would be able to help them.
   11. For appointments in the future, the user should be able to cancel the appointment, if they are no longer free.

### 1.10.2 - Teenagers’ System

1. **Input**
   1. There must be an Android app through which the person should be able to interact with the system.
   2. There should be a system to ensure to remember the password for the android application, so that they would not have to remember the password every time they log onto the application.
2. **User interface**
   1. The user must be able to see everyone who has requested their help.
   2. They should be able to cancel any appointments which they are no longer able to be available for.
   3. The user should also be able to define when they are free for the next six weeks, allowing them to set the times that they would be free for appointments.

### 1.10.3 - General

1. The appointments should be two hours in length for all appointments.
2. The payment should be defined as £20 per hour for the appointments, therefore being £20 if the appointment takes up less than half the appointment slot, or £40 otherwise.
3. The system should be secure, with passwords hashed to ensure that they are not easy to utilise in case the server is hacked.

## 1.11 – Explanations for objectives

1. **Input**
   1. In talking to Mrs Wilson, she deemed it important that the system be at least available on both phones and tablets, especially on Android since this is what the majority of the people she worked with used. She importantly stated that this was by far more important than having a web version of the application.
   2. She also said that having a system which would receive SMS messages would be highly useful as most elderly people would be able to send SMS messages. Though it would be possible to insist upon a certain format for the messages to be sent in, it would make for a highly clunky interface. Therefore, if the system could simply interpret whatever the user is attempting to say, this would be optimal. It would also be very important that the system be as compressed as possible, for example removing the ability for the elderly person to choose a helper, instead selecting the person closest to them.
   3. As discussed above, this system offers the optimal compromise between allowing the elderly person flexibility, while still keeping it as simple as possible.
   4. As one large issue faced by elderly people is dealing with passwords. Therefore, by ensuring that the system does not require them to remember their passwords to the system, this reduces the problems. This also means that there is no requirement for a ‘Forgot my password’ system. Though this means that anyone who is able to unlock the elderly person’s phone would be able to book an appointment, it was decided that this would be a worthwhile compromise.
2. **User Interface (on all systems)**
   1. Mrs Wilson was clear that the simplicity per screen was far more important than minimising the number of screens.
   2. She advised two paths from each screen as a maximum, as a guideline to help to reduce the complexity as much as possible.
   3. According to the United Kingdom government, all printed (and digital) materials should use size 12 as a minimum font size. Therefore, this would a worthwhile limit for this application as well.
   4. The ability to view a list of people who would be willing to help is important, so an elderly person can see the helper in more detail including where they are and seeing their reviews and ratings.
   5. Since the maximum distance that a helper is willing to travel has been omitted, this is important to ensure that the likelihood is that even if the elderly person has not specifically attempted to find someone nearby that the person they have chosen (who would be near the top of the list) would be reasonably near.
   6. By seeing the ratings and reviews of a teenager, the elderly person can make sure that the person is reliable and would be able to fix the problem they are having. Additionally, the review may include the nature of the problems they have fixed in the past, indicating whether the teenager has existing knowledge in the area. By being able to see their location, the elderly person can ensure the helper does not have to travel from very far.
   7. Especially in the case where the elderly person is free all day, the ability to simply request the teenager’s services is a useful one. However, it would also be important for an elderly person to get an appointment to work with their schedule.
   8. Two weeks ensures that the elderly person can book appointments well enough, while still offering the opportunity for helpers to ensure their timetable is accurate for that period of time, since they have the ability to change their timetable many weeks ahead of when elderly people are able to book appointments.
   9. This ensures that the elderly person can check all the appointments they made in the past, allowing them to ensure they are around for the next appointment or to cancel it.
   10. Mrs Wilson described how this was important as the people who are helping out are teenagers and not professional, so a number of them may not be very good and a few bad people, if not obvious to the elderly when booking an appointment may ruin the system.
   11. If the problem has been solved, it would likely be no longer necessary for the appointment to occur, therefore the ability to cancel the appointment is an important one.
3. **Input**
   1. Philip and my own research indicated that the majority of technological people use Android phones so iOS can be omitted.
   2. Though less important for the helpers, who would be technologically savvy, this would be a welcome addition.
4. **User interface**
   1. This is relatively self-explanatory as the user must be able to see who has requested their services.
   2. Especially important for the helpers, this ensures that they can cancel an appointment if they are no longer available or if they have been requested by an elderly person who is outside of the radius which the teenager would be willing to travel within.
   3. The teenager may have an often-changing timetable of when they are free therefore it would be important that they can easily adjust this timetable.
5. This length was decided in conjunction with Mrs Wilson, who said this was the maximum time which an appointment would take. This fixed length of appointment also simplifies the system, ensuring the person can only book appointments at set (even hour) times.
6. This was decided in section 1.7.
7. Though it would be highly advisable (and this could be mentioned on the application) that passwords should not be reused, and therefore it shouldn’t matter if passwords for this system were stolen, it is likely that not everyone would follow this advice. Therefore, it would be important that passwords are stored securely, hashed in the database and never transmitted unencrypted over a network, in order to make it harder for them to be stolen.

# 2 – Design

## 2.0 - Overview of entire system



## 2.1 - Introduction

In designing the solution to the problem, much attention was paid to centralizing as much as possible of the system, both for reliability and security and to ensure that the minimal amounts of coding as possible could be done on the applications side. This ensures that the applications will be fast to run, as well as being easier to produce. Hence, the solution relies upon a server which will respond to requests to return information which can then be interpreted by the application which is being used to respond as expected.

While creating the user interface, I spent more time with my end user for the elderly specifying how I could specifically design the user interface to be as easy to use as possible, discussing everything from the sizes of button to the colours which would be the most obvious.

## 2.2 - Data

### 2.2.1 - Storing Data

Since I wanted the data to be accessible from all the various applications, such that it could be interacted on by multiple users, this created the problem of ensuring all the data that was used could be stored centrally. To do this, I decided to create a centralised server which could act as the hub for the data and which could be accessed when it was required.

While I considered some databases such as MariaDB, I decided that the MySQL packages by Oracle provided the most secure features, while having acceptable documentation for the syntax of SQL commands. Additionally, as compared to Microsoft SQL Server, it could easily run on an Ubuntu 14.04 server that I already had access to, as it used it as an Apache Web Server.

In considering what data would be required to be stored, I went through the specification that I created and considered where the data would be needed to be stored. Starting from a general number of tables, I began to draw out the connections between the tables of data, ensuring that I got the database into third normal form, to ensure that the data would be stored most efficiently.

#### ER Diagram



#### User Data

Though the accounts for helpers and accounts for the elderly could easily be combined together, this would create a relationship between appointments and the accounts table which would not be desired. Therefore, separating these two ensures that the relationship is One-One and is as efficient as possible. Additionally, this ensures that if in the future, more or less information is desired from one of the groups, then this could be easily adjusted without affecting the other group.

However, at the most, the same information is being stored about both parties with the exception of the inclusion of a timetable id for the helpers.

Hence, we can consider them together for now.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| id | Int | This is the key for the table, acting as the counter. This auto-increments every time a new user is added to the table – using this function when creating the MySQL table. |
| Username | Varchar(50) | The username is the username which the user creates when registering the account. This means that they don’t necessarily need to remember their email address which could be bulky.  By creating a reasonably long length string (varchar), we ensure that the user can use most reasonable usernames. |
| emailAddress | Varchar(50) | The email address acts as part of the contact information provided to both parties when an appointment is set up. |
| password | Varchar(50) | The password contains information (though hashed) of the password which the user needs to use to access the application and connect them to their account information. |
| firstname | Varchar(25) | Contains the user’s firstname. |
| surname | Varchar(25) | Contains the user’s surname. |
| firstLineOfAddress | Varchar(100) | Contains the user’s first line of the address. This field is optional in the helpers database, while necessary in the elderly database. |
| secondLineOfAddress | Varchar(100) | Contains the user’s second line of their address. Again optional in the helpers database while necessary in the elderly database. |
| telephoneNumber | Varchar(100) | Contains the user’s telephone number. |
| postalCode | Varchar(25) | Contains the user’s postcode. Optional in the helpers database while necessary in the elderly database. |
| City | Varchar(25) | Contains the user’s city. Optional in the helpers database while necessary in the elderly database. |
| Country | Varchar(35) | Contains the user’s country. Optional in the helpers database while necessary in the elderly database. |
| *timetableId* | *Int* | *In the case of the helpers database, this contains the id of the weeklyTimetables database which contains information of the timetable.* |

#### Timetables

While the timetables table could theoretically be combined with the main user’s database for the helpers, by separating these two tables, the system is marginally more clear and easier to work with.

The timetable system itself is separated into a few different tables. The top level table defines all the weeks for which the data is stored, containing ids for all the weeks. When a new timetable is created, if required, a new column is created in the SQL database, containing the information of that week.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| Id | Int | This is the key of the table, and auto increments whenever a new entry is added. |
| defaultWeekId | Int | This contains the id of the default week which is created when the user first signs up to an account. |
| DD\_MM\_YYYY  (i.e. Week Beginning DD/MM/YYYY) | int | There may be n columns of this, where each column contains either a null, if this entry simply uses their default timetable for this week, or contains an id for another week entry linking to the week table.  Whenever a new column is added, a check is run whether the DD/MM/YYYY is in the past, and whether it can therefore be deleted. This ensures that the data remains reasonably small and cannot threaten to break the server. |

As mentioned, defaultWeekId and WeekBeginning… link to the week table, which provides links to an entry in the day table for each day of the week.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| Id | int | This is the key of the table, and is referred to by the WeeklyTimetables table. |
| MondayId | Int | This contains a link to an entry of the day table, containing the specific time when a helper is free. |
| TuesdayId | Int | … |
| … | Int |  |

This then has links to the ids of entries in the day table, where the actual information as to whether a user is free is stored.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| Id | Int | This is the key of the table, and is linked to by the entries of the week table. |
| a02 | Bool | This contains the information as to whether the user is free at this time. It was decided that 2 hour intervals were the correct length – offering simplicity for both booking appointments and for creating a timetable as a helper. |
| a24 | Bool | As above |
| … | Bool | As above. |
| a2224 | bool | As above. Though it was considered unlikely that appointments would be booked in some of these timeslots it was decided that it would be better if they existed anyway, never to be used.  However, it is important that in the User Interface for creating this timetable, that the work is minimised by defaulting that the timeslots which are the least likely are defaulted as being false. |

#### Appointments

The appointments information contains the information of the appointment for both the helper and the elderly person, with an entry being created when an elderly person creates an appointment, either via the app or via the SMS system.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| Id | Int | This is the key for the table. |
| helperId | Int | This contains the information of who is the helper who has been assigned to solve the problem. |
| elderlyId | Int | This contains the information of the elderly person to be helped. |
| dateAndTime | DateTime | This contains the date and time of the appointment |
| dateAndTimeCreated | DateTime | This contains the date and time that the appointment has been created. |
| ratingId | Int | This contains the id (reference to the ratings table) of the rating given to the helper after this job has been completed. |
| reviewId | Int | This contains the id (reference to the reviews table) of the helper once the job has been completed. |

#### Ratings

The ratings table is where the ratings of the job are stored, with each appointment having one or no entry in the ratings table.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| Id | int | Key of the table |
| 1Star | Bool | This contains the information of how many stars the helper has been given for this particular job. |
| 2Star | Bool | As above |
| … | Bool |  |
| 5Star | Bool | As above |

It’s important to note that it was decided that the options for the ratings of the helpers would only be integers between 1 and 5. This was decided in order to simplify the system as much as possible – ensuring that the system would be easy to use for the elderly person who is trying to rate their helper.

#### Reviews

The reviews table is where the reviews of the job are stored, with each appointment having one or no entry in the table.

|  |  |  |
| --- | --- | --- |
| Name | Variable Type | Description |
| id | int | Key of the table. |
| review | Varchar(10000) | This contains the review – which has been decided to have a maximum length of 10,000 character – around 1,500 words. This has largely been put as an exceedingly large maximum. We expect the length of each review to be significantly shorter. Therefore, the MySQL database has been programmed to reduce the amount of space used by this field to be reduced to the length of the text, once it has been added. |

### 2.2.2 - Communications with applications

There were a couple of immediate ways to communicate with the database to get all the required data from it. The first was through immediately communicating with it, while the second involved producing a local API which could be used to communicate to the SQL server and return the required information. The first, while inherently simpler, offers the challenges that code would have to be repeated on all applications when tasks like verification of data entry would be required. Even more importantly, it means that the security is much reduced as the data could be intercepted en-route as encrypting this transfer of data would be much more challenging. In fact, though Oracle do provide a system by which this connection can be established, they expressly suggest that you avoid it. Therefore, I decided to produce an API which could allow the code to be securely centralised.

## 2.3 - WebAPI

I decided to use the REST standard for creating an API, which offers a number of important steps for communicating with HTTP requests, ensuring they are only of type: “GET”, “POST”, “PATCH”, “DELETE”, “PUT”. It also established the use of Json to both send and receive data between the client and the server.

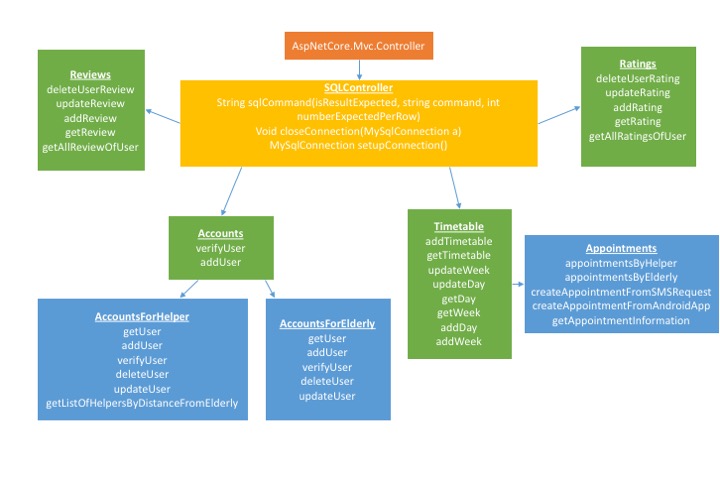
I also decided to use a .NET core API, as I had prior knowledge of .NET programming, therefore would not have to learn how to use PHP, the other language which appears to be largely used for this. By using Microsoft’s IIS and .NET core libraries for Ubuntu 14.04 meant that I could largely test using my computer as a ‘connection to the server’ using a local SQL server as the database to communicate with. However, because the .NET libraries also exist with Ubuntu, changing to the server would be largely seamless, with no problems associated with differing libraries.

### General Structure of .NET Core API

The .NET core API effectively creates a tunnel through which a number of controllers can act as listeners, awaiting a command which meets the HTTP request type that they are defined to listen to – responding to the command when the command type and URL are in one of the controllers.

In order to produce this, there must be a number of files which define how the API works, including defining the namespace of the controllers and defining the dependencies and the runtimes that the API should work on.

All the controllers must inherit from the AspNetCore.Mvc.Controller class. In fact, in order to simplify in dealing with the SQL server, there is another intermediary class, SQLController, which inherits from Controller. This is inherited by all of the controllers. Hence, there is the following hierarchy chart.



In order to communicate with the SQL server, the .NET Oracle SQL Server package is used, being acquired with a NuGet request to ensure that it can be retrieved every time the server script is run, thereby ensuring it exists everywhere.

### Communicating with JSON

Communicating with JSON was the obvious choice, with the only other option being XML. However, the greater availability of libraries to parse the JSON data on both the android application side and the Web API side meant that JSON was a better choice.

For the Web API, in order to deserialise data which is inputted to the WebApi, the ‘[From Body]’ tag is used, which calls the System.Net Json deserializer to process the data to the object which is defined in the definition of the function. When data is to be sent as Json, it is either passed directly as the object, as this would automatically be converted to Json or it is explicitly converted to Json using the NewtonSoft.Json library if this is necessary.

### Communicating with the SMS Server

In order to communicate with the SMS server and the Google Maps API, which require GET and POST requests, the HttpWebClient library is used (in a composition fashion, in order to allow the rest of the system to function without adaptation).

### 2.3.2 - Functions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Function Name | HTTP type | URL | Input | Output |
| Accounts |  |  | /ac |  |  |
|  | HelloWorld | GET | /ac |  | “Hello World” |
|  | AddNewUser | POST | /ac/users/ad | User | Boolean (success or failure) |
|  | VerifyUser | POST | /ac/users/vf | UserToBeVerified | User (empty if failed, populated if successful) |
|  |  |  |  |  |  |
| AccountsForHelpers |  |  |  |  |  |
|  | HelloWorld | GET | /ach |  | “Hello World” |
|  | GetInfo | GET | /ach/users/gi{id} | Int id | Returns a user (see the definition of the user class below). |
|  | GetNearestHelpers | GET | /ach/users/gl{id} | Int Id of the current user, | List of the nearest helpers, to allow the elderly to pick which of the people to pick. |
|  | DeleteUser | DELETE | /ach/users/rm | Int id of the user | Returns a bool of success or failure. |
|  | UpdateUser | PATCH | /ach/users/up | Int id of user, user (see definition of class below) | Returns bool of success or failure |
|  | AddNewUser | POST | /ach/users/ad | User (see definition of class below) | Returns bool of success or failure |
|  | VerifyUser | POST | /ach/vf | UserToBeVerified (see definition of class below) | Returns id of userLoggedInto or null if failedToLogin |
|  | mergeSort | NA | NA | List of users and address of elderly to sort by nearest to | Sorted list of users |
|  | merge | NA | NA | Two lists of user to merge | One list of merged users |
|  | getDistance | GET | /ach/users/gd | Two string addresses | Double distance between the two in km |
|  | fileGetContents | NA | NA | String filename | String contents |
|  |  |  |  |  |  |
| AccountsForElderly |  |  |  |  |  |
|  | HelloWorld | GET | /ace |  | “Hello World” |
|  | GetInfo | GET | /ace/users/gi{id} | Int id | Returns a user (see the definition of the user class below). |
|  | GetInfoByTelephoneNumber | NA | NA | String telephoneNumber | Returns a user (see the definition of the user class below). |
|  | DeleteUser | DELETE | /ace/users/rm | Int id of the user | Returns a bool of success or failure. |
|  | UpdateUser | PATCH | /ace/users/up | Int id of user, user (see definition of class below) | Returns bool of success or failure |
|  | AddNewUser | POST | /ace/users/ad | User (see definition of class below) | Returns bool of success or failure |
|  | VerifyUser | POST | /ace/vf | UserToBeVerified (see definition of class below) | Returns id of userLoggedInto or null if failedToLogin |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Timetables |  |  | /tt |  |  |
|  | getTimetablesOfAUser | GET | /tt/gt | Int helperID | Returns timetable (see below for definition) |
|  | addNewTimetable | POST | /tt/adt | timetable | Returns bool for success or failure |
|  | UpdateWeek | NA | NA | week |  |
|  | AddDay | NA | NA | day | Int id (in database) |
|  | UpdateDay | POST | /tt/ud | day | Returns bool for success for success or failure. |
|  | addWeek | NA | NA | week | Int id of week |
|  | getWeek | NA | NA | Int id | week |
|  | getDay | GET | /tt/gd | Int id | day |
|  |  |  |  |  |  |
| Appointments |  |  |  |  |  |
|  | Create Appointment From SMS Request | POST |  | DateTime time, string telephoneNumber | Returns appointment (see definition below) |
|  | Create New Appointment From Android App | POST |  | Appointment (see definition below) | Returns bool of success or failure |
|  | Get Appointment information | GET |  | Int AppointmentID | Returns appointment |
|  | Remove Appointment | DELETE |  | Int appointmentID | Returns bool of success or failure |
|  | Get Appointments by Elderly | GET |  | Int elderlyID | Return a list of appointments |
|  | Get Appointments by helper | GET |  | Int helperID | Returns a list of appointments |
|  |  |  |  |  |  |
| Ratings |  |  |  |  |  |
|  | Get All Ratings Of a User | GET |  | Int helperId | Returns list of ratings (for definition see below) |
|  | Get One specific rating | GET |  | Int appointmentID | Returns rating |
|  | Add a rating to an appointment | POST |  | Int appointmentID, rating | Returns bool of success or failure |
|  | Update rating | PATCH |  | Int appointmentID, rating | Returns bool of success or failure |
|  | Delete rating | DELETE |  | Int appointmentID | Returns bool of success or failure |
|  |  |  |  |  |  |
| Reviews |  |  |  |  |  |
|  | Get all reviews of a helper | GET |  | Int helperID | Returns a list of reviews |
|  | Get one specific review | GET |  | Int appointmentID | Returns review |
|  | Add a review to appointment | POST |  | Int appointmentID, review | Returns bool of success or failure |
|  | Update a review | PATCH |  | Int appointmentID, review | Returns bool of success or failure |
|  | Delete a review | DELETE |  | Int appointmentID | Returns bool of success or failure |
|  |  |  |  |  |  |

The classes which are used on both the clients and the server which define the types and objects for the objects user in communicating with the WebAPI are as follows:

|  |  |  |
| --- | --- | --- |
| Class | Item Name | Type |
| User |  |  |
|  | Id | int |
|  | Username | String |
|  | emailAddress | String |
|  | Password | String |
|  | Firstname | String |
|  | Surname | String |
|  | firstlineOfAddress | String |
|  | secondLineOfAddress | String |
|  | telephoneNumber | String |
|  | postalCode | string |
|  | City | String |
|  | Country | String |
|  | timetable | timetable |
|  |  |  |
| Timetable |  |  |
|  | id | Int |
|  | defaultWeek | week |
|  | listOfOtherWeeks | List<week> |
|  |  |  |
| Week |  |  |
|  | id | int |
|  | weekBeginning | DateTime |
|  | monday | Day |
|  | tuesday | Day |
|  | Wednesday | Day |
|  | Thursday | Day |
|  | Friday | Day |
|  | Saturday | Day |
|  | Sunday | Day |
|  |  |  |
| Day |  |  |
|  | id | int |
|  | timesFree | Bool[12] |
|  |  |  |
| Appointment |  |  |
|  | Id |  |
|  | helperid | Int |
|  | elderlyid | int |
|  | dateAndTime | DateTime |
|  | dateAndTimeCreated | DateTime |
|  | Rating | Rating |
|  | review | Review |
|  |  |  |
| Rating |  |  |
|  | id | Int |
|  | rating | Int |
| Review |  |  |
|  | Id | Int |
|  | review | String |
|  |  |  |
| UserToBeVerified |  |  |
|  | emailAddress | String |
|  | password | String |
|  | helper | Boolean |
|  |  |  |
| DateTimeExtensions |  |  |
|  | Func getWeekBeginning (DateTime dt) | DateTime |
|  |  |  |

While the majority of these simply require a number of SQL commands to return specific data from the MySQL database, a couple of the functions require a little more explanation of how they will work.

#### **Returning List of Nearest Helpers**

In order to order the ordered list of helpers, there are broadly three stages which must be followed:

1. Get all helpers
2. Find the distance from the elderly person for each helper.
3. Sort the helpers by this distance.

**Get all helpers**

In order to get all the helpers, it is simply a question of calling a get command of all helpers (this command is only available internally and is not available outside of the API and can’t therefore be exploited).

**Find the distance from the elderly person for each helper**

There were two methods considered to find the distance between two addresses. The first, utilises, just the Google Maps API and treats the direction distance (distance on the road) as the distance. The second uses the Google Maps API to get the longitude and latitude and uses the Haversine Formula to extract the distance between two points using this formula.

It was decided that the former is better as it is more representative of the time taken by the helper to get to the elderly person’s house. Therefore, the Haversine formula is not needed.

**Sort the helpers by this distance**

In order to sort the helpers by this distance, a recursive MergeSort is used, recursively slitting the list into halves, sorting each half before combining them together using a merge. The process is outlined in the below pseudocode:

MergeSort (List a):

If (a.Count() == 1)

Return a;

Else

Return merge(MergeSort(firstHalfOfA), MergeSort(secondHalfOfA))

Merge (List a, List b):

List c

While(a and b have elements):

If (a[0].distance < b[0].distance)

c.Add(a[0]);

a.RemoveAt(0);

Else

c.Add(b[0]);

b.RemoveAt(0);

While(a has elements):

c.Add(a[0]);

a.RemoveAt(0);

While(b has elements):

c.Add(b[0]);

b.RemoveAt(0);

return c

#### Create New Appointment from SMS request

In order to do this, the function completes the following tasks:

1. Gets the user information of the elderly person from the telephone number
2. Gets the list of nearby helpers to the elderly person
3. Parses the sms into a date and time
   1. ^
4. Find the first helper who is free at that date and time, book appointment (using create new appointment from Android App)
   1. \*
5. Send an SMS to the original sender with the information about the appointment.

**Considering ^**

In order to parse the SMS and return the date and time that the user wants to have the appointment, there were a few possible ways of doing this. The first involves creating my own system to search for all possibilities including things like ‘tomorrow’ and ‘today’ which would each need to separately catered for.

Therefore, it would be simpler to make use of existing APIs. While it would be significantly more effective to use something like the IBM Watson Natural Language processing API, it proved much simpler to use a DateTimeNaturalLanguageParser for C# that I found published under the MIT license on GitHub. While not perfectly effective – for example the year must always be stated otherwise, it does not return any of the correct date – it generally works.

**Pseudo Code of \***

Foreach helper in list of helper:

Timetable t = Get Timetable (helper.timetableId)

DateTime dt = findWeekBeginning(dateTimeSMS)

If dt.ToString() exists in t.weeks[x].weekBeginning:

Day correctDay = t.weeks[x].week.(dateTimeSMS.DayOfWeek.ToString() + “id”)

If correctDay.timesFree[dateTimeSMS.Hour / 2]:

Create appointment with this helper at this time

Send SMS to user with this information

Else

Next helper

Else

Day correctDay = t.defaultWeek.(dateTimeSMS.DayOfWeek.ToString() + “id”)

If correctDay.timesFree[dateTimeSMS.Hour / 2]:

Create appointment with this helper at this time

Send SMS to user with this information

Else

Next helper

If stil haven’t found, then send SMS saying “Noone could be found”

#### Create New Appointment from AndroidApp

The function must complete the following tasks:

1. Convert the dateAndTime into the week beginning data and the day.
2. Check whether that week beginning is a special week for the helper, or whether it falls inside the default week.
   1. It does this by going into the INFORMATION\_SCHEMA for the table and looking at the names of the columns
      1. If it isn’t one of the weeks already there, then it creates a new column and copies the value of defaultWeek from all the items into the new column.
      2. Then, it recurs on the same function, this time confident that it will find the new column
3. Check whether the helper is free at the time on that day, looking inside the correct entry of the ‘day’ table.
4. If it is free, the appointment can be booked and the time should be changed to being used and the appointment added to the tableOfAppointments.

### 2.3.3 - Securing data

An important part of the decision to choose to use a WebAPI was in order to maximise the security of the system, by ensuring that people couldn’t retrieve the placement of the server and simply issue SQL commands to retrieve all the data on all the users. There are a couple of other simple measures which can be taken to further maximise the security of the data, even if the connection is insecure.

#### Hashing Passwords

By hashing password, we can maximise the security of the passwords in a couple of different ways. Firstly, from a user’s perspective, it assures them that we don’t have access to the password and thereby ensures that they are comfortable with using the site (though it would still be advisable to use a different password than others for this account). It also means that should hackers manage to penetrate the security of the server, which will be minimised as far as possible, through root security in Ubuntu 14.04, they will struggle to retrieve passwords from the hashed values of the passwords.

Therefore, the passwords will be stored as the hashed value – with the passwords being hashed using the MD5 cryptographic hash, which produces a 128-bit hash. This, while not perfectly secure, draws a good compromise between the time taken to hash on the client side (on the app) and security.

#### Encrypting Communications with WebApi

Additionally, in case the connection is insecure, the communications, after establishing the login can be encrypted. This will be encrypted using SSL, which though rather substantial as an algorithm, is very fast. The key that will be used for this will be the integer value of the user id which is returned to the client when the userFirstLogsIn. Though if a hacker is able to intercept this, they would have access to the contents of the request, this simply adds another step which the hacker would have to pass. Even more importantly, even if the hacker managed to find the contents of the message, they would be unable to know the destination of the message.

### 2.3.4 - Verifying data entry

It has been decided that for speed of the system, the main verification will be done on the server side, before being completed if necessary on the client side to find more information about the error so that the end user can have more coherent error messages. This is with the one exception of the password checks which must be completed on the side of the apps as the server would never receive the message before it is hashed.

Therefore, the following checks must be carried out during verification:

|  |  |  |
| --- | --- | --- |
| Task | Input | Check |
| Adding or updating a user |  |  |
|  | emailAddress | Already used? |
|  | telephoneNumber | Already used? |
|  |  |  |
|  |  |  |
| Adding an appointment |  |  |
|  | helperId | Valid user? (checked by doing a lookup) |
|  | elderlyPersonId | Valid user? (checked by doing a lookup) |
|  | dateAndTime | Is it in the past?  Is helper free at this time? |
|  | dateAndTimeCreated | It it in the future? |

## 2.4 - Android App

While I had otherwise intended to create two android apps, one for the helpers and one for the elderly since I assumed that they would largely be entirely separated, I have found they are in fact rather similar, and by creating ‘one’ app, they could inherit large portions of the backend code from one another. Therefore, I am only creating one android app, though with two separate timelines for the elderly and for the helpers. Therefore, for the purposes of this report, I am going to deal with the design together, though they share a couple of screens.

For the Android apps, I chose to work in Xamarin.Android, which allows me to use existing skills in C# and XML and combine them to produce a functioning Android Application. This also means that I can reuse much of the code for the classes in the WebAPI in order to produce the Android Apps, for example the classes for the users and timetables.

### 2.4.1 - Important Features

#### Communicating with Web API

In order to communicate with the WebApi, the Android App makes use of the RESTSharp library, which deals with the communication entirely, allowing the programmer to just say the method, the data, and the URL of the communication.

Unfortunately, this function must be often repeated on a number of screens as there must be subtle changes in the URL and / or the data expected.

In order to serialise the data to be sent to the API and to deserialise the JSON input from the API, the NewtonSoft JSON library is used, with it defining the object and object type to convert to JSON.

#### Communicate with SMS Server

To communicate the SMS server, the Android App again makes use of the RESTSharp and NewtonSoft.Json Libraries which can be acquired using a NuGet packages and therefore are included as part of the package of the Android application when it is installed.

#### Remembering Logins

In order to remember the login and password, the Android application stores the information in a SQL database, using the SQLite database to store the following information:

|  |  |
| --- | --- |
| Name | Type |
| emailAddress | VARCHAR(50) |
| password | VARCHAR(50) |

When the user logs in, the username and password is added to the database. The password is added in plaintext as it would be impossible for anyone to gain access to the database. When the login screen is reached again at any point, the application first checks through the database and attempts to login with information from the database.

### 2.4.2 - Helpers

#### Main Flowchart

The flowchart shows the system without the addition of in case of problems. In fact, these are dealt with very simply, using Android alert windows, which are easy to code and offer a method by which the user is unable to miss the alert. Therefore, these ought to be added wherever problems could occur, for example at the login screen, where a failed login will in fact be reported with an alert preventing the user from getting into the rest of the app.



#### User Interface

##### Start

|  |  |
| --- | --- |
| **UI** | **Activity** |
|  | **OnCreate**  **LoginButtonClick:** When the button is clicked, go to the login screen.  **RegisterButtonClick:** When the button is clicked, go to the register screen. |

The start allows a user to create a new account or login by clicking one of the buttons.

##### Log In

|  |  |
| --- | --- |
|  | **OnCreate:** Go through the database and attempt to login with any of the accounts stored in the database or create a table if it doesn’t already exist.  **LoginAsHelperClick:** Attempt to login as a helper.  **LoginAsElderlyClick:** Attempt to login as an elderly.  **checkIfEmpty:** Shows a dialog box if all information is not present  **login:** Returns a true or false dependent on whether the login is successful. |

The login window provides two text boxes with default text explaining what is to go in the boxes. This default text will appear only when there is not text in the boxes. The password boxes will, once filled in, appear as a series of dots, by using the Android password box, in order that those in the vicinity of the user not be able to see the password.

##### Register

|  |  |
| --- | --- |
|  | **OnCreate:**  **RegisterRegisterAsHelperClick:**   * checkValid * registerTheUser * verifyNumber * checkVerify to the get the userId * if it doesn’t get verified, deleteUser   **RegisterRegisterAsElderlyClick:**  **checkValid:**   * Checks if any fields are empty or if the telephone number is invalid. * If invalid then show dialog box and returns false   **deleteUser:**   * Sends the REST request to the API to delete the user account.   **testVerify:**   * Completes the same function as in login and returns the user.   **register:**   * POSTs the data to the API to add the account.   **verifyNumber:**   * POSTs the number to the SMS server and shows the return, the verification code, to verify the telephoneNumber. |

The registration screen, common for both the elderly and helpers offers the opportunity for the users to enter all the required information to register. Though I considered creating a toggle switch for elderly or helper, this separate buttons method appears more clear and easy to use – this is very important for the elderly. Once the registration has gone through, there is an alert saying “registration succeeded” or “registration failed”. If the latter occurs, the user will return to this page. If it succeeds, they will be instead directed to the login page.

##### Logged in

|  |  |
| --- | --- |
|  | **OnCreate:**   * Populate the name textview using information from GetUserDetails.   **SetTimetableClick:**   * Open the setATimetable activity.   **ListOfAppointmentClick:**   * Open the listAnAppointment activity.   **GetUserDetails:**   * Send the REST request to the Web API and returns the user details. |

Upon login, the user gets to access either part of the app, the appointment side, or the timetable side through two buttons.

##### View Appointments

|  |  |
| --- | --- |
|  | **OnCreate:**   * Get list of appointments. * Foreach of the appointments, get the name of the elderly person (using getUserDetails). * Populate the list view in the order of upcoming first (closest first), then going back in history   **OnListItemClick:**   * Call the activity to show more details about that appointment.   **returnListOfAppointments:**   * From the helperId, this makes a call to the REST api and returns the list of appointments.   **getUserDetails:**   * Makes a call to the Web API and returns information about the user. |

From any deeper than this in the app, there are always back buttons (on the android device) which allow the user to move back in the pages they have visited, therefore allowing them to visit every site. This screen easily shows all the appointments, displaying their times.

##### See details of Appointment

|  |  |
| --- | --- |
|  | **OnCreate:**   * GetAppointmentDetails of appointment to show in more details. * Populate the text views using details from GetUserDetails * If in the past, ensure that cancel appointment is not enabled. If in the future, it should be enabled.   **CancelAppointmentClick:**   * Send the request to the Web API.   **GetUserDetails:**   * Send request to Web API and return the user information.   **GetAppointmentDetails**   * Send request to Web API and return the appointment. |

By clicking on one of the appointments, the helper can see more details about the appointment, including the contact details of the elderly person who has booked the appointment.

##### Set Timetable

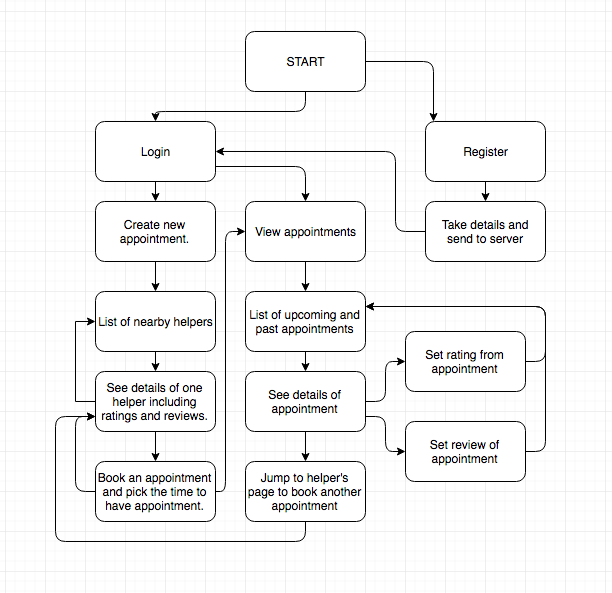
|  |
| --- |
|  |

In the set timetable section, the helper can set whether they are free for a specific time. The first screen allows them to choose the specific week for which they want to change their timetable for. The second week allows them to choose the specific day to change their timetable for. Finally, the last screen allows the helper to select what time they would free on that day, with the submit timetable making the changes to their timetable.

### 2.4.3 - Elderly

#### Main Flowchart

The flowchart below is the main flowchart of the app, showing how a user can carry out all of the task- with each of the boxes corresponding to a single screen on the design. It must however be noted that this assumes successes in all activities, such as assuming that the password is valid and the user successfully logs in. All errors are being dealt with by using Android Pop-Up screens since these are easy to use, requiring very little coding, while also being very hard to miss and obvious for the user to interpret.



#### User Interface

The first few are the same as for the helpers, so no new commentary is provided.

##### Start

|  |
| --- |
|  |

##### Login

|  |
| --- |
|  |

##### Logged in Interface

|  |
| --- |
|  |

Here, for the elderly, instead of offering the timetable, the user can book an appointment.

##### Create new appointment

|  |
| --- |
|  |

When the person goes into the booking section, they are presented with a list of the nearest helpers as well as their rating (to the nearest integer).

##### Book Appointment

|  |
| --- |
|  |

By clicking on any one of the individuals, they get more details about them, as well as a link to see all their reviews.

##### Set time to book appointment

|  |
| --- |
|  |

Clicking the book button moves the user to a place where they can book an appointment with that helpers, choosing a date, before selecting from the times at which the helper is free.

##### View appointments

|  |
| --- |
|  |

This screen, identical to the similar one on the elderly side, offers the opportunity for the user to see their current and past appointments.

##### See details of appointments

|  |
| --- |
|  |

In past appointments only, when the specific appointment, the elderly person will be able to set a review and a rating, using the intuitive next screens.

##### Set rating

|  |
| --- |
|  |

##### Set review

|  |
| --- |
|  |

## 2.5 – SMS Server System

The SMS system relies upon the Twilio system which provides a phone number and an API through which text messages can be received and sent. In order to communicate with the Twilio REST API, the helper library is used with Python.

This has only one class, and consists of four functions:

1. “@app.route('/sms', methods = ['GET','POST'])”
   1. This deals with incoming SMS messages and simply passes them onto the .NET core Web Api as well as sending a generic response: 'Thanks for your request. Your appointment details will be sent to you shortly'
2. “@app.route('/sendSms', methods = ['POST'])”
   1. This deals with any messages to be sent to other number and passes them on, using the Twilio helper library, to be sent.
   2. (This receives the required data in JSON, which it uses the Requests library to deal with)
3. “@app.route('/verifyNumber', methods = ['POST'])”
   1. This deals with verifying a new phone number (so it can text the Twilio number and receive messages from it),
   2. It POSTs lots of information to the Twilio REST API and receives a validation code which is then returned to the Android application.
4. “if \_\_name\_\_ == '\_\_main\_\_':”
   1. This defines important things like the port number on which the API should be run.

**Example of SMS Appointment Server in action (appointment allocated by .NET Core API)**

****

## 2.6 – Testing Strategy

The general aim of the testing is to test each part in turn, before testing the entirety of the system. Hence, the following test series were developed:

**Test Series 1: Testing the Web API (Black-Box Testing)**

In order to test the WebAPI, the requests from the Android App will be spoofed, using Postman, which allows me to send HTTP requests to each of the HTTP functions. In addition to HTTP functions, there will be a number of Unit test systems to test the functions that would have results that would have static results and thereby check whether changes in the program have any unintended side-effects. The code for these unit tests are in the appendices.

The testing of the WebAPI gets further split into testing the following:

|  |  |
| --- | --- |
| Test Series Number | Purpose of Test Series |
| 1.1 | Test SQL command class |
| 1.2 | Test DateTime Parsing |
| 1.3 | Test HTTP GET (get it to send commands to a computer which will process and log the requests) |
| 1.4 | Test Accounts controller |
| 1.5 | Test AccountsForHelper controller |
| 1.6 | Test AccountsForElderly controller |
| 1.7 | Test Appointments controller |
| 1.8 | Test Ratings controller |
| 1.9 | Test Reviews controller |
| 1.10 | Test Timetable controller |

**Test Series 2: Testing the Python SMS Server API (Black-Box Testing)**

In order to test the Python SMS Server, the tests can be triggered by a combination of SMS messages from a phone and using Postman to trigger the API functions.

|  |  |
| --- | --- |
| Test Series Number | Purpose of Test Series |
| 2.1 | Test GET on ‘/’ |
| 2.2 | Test SMS receipt on /SMS |
| 2.3 | Test /sendSMS |
| 2.4 | Test /verifyNumber |

**Test Series 3: Testing the Android Application**

In order to test the Android Application, the flow of the android app is tested, ensuring that buttons send the application to correct places.

|  |  |
| --- | --- |
| Test Series Number | Purpose of Test Series |
| 3.1 | Test ‘Login or Register’ screen |
| 3.2 | Test ‘Login’ screen |
| 3.3 | Test ‘Register’ screen |
| 3.4 | Test ‘List of Appointments’ screen |
| 3.5 | Test ‘Appointment Details’ screen |
| 3.6 | Test ‘Ratings submit’ screen |
| 3.7 | Test ‘Review submit’ screen |
| 3.8 | Test ‘Helper details’ screen |
| 3.9 | Test ‘See reviews’ screen |
| 3.10 | Test ‘Book appointment weeks’ screen |
| 3.11 | Test ‘Book appointment days’ screen |
| 3.12 | Test ‘Book appointment times’ screen |
| 3.13 | Test ‘Book details confirm’ screen |
| 3.14 | Test ‘Appointment details (helper)’ screen |
| 3.15 | Test ‘Set Timetable’ screen |
| 3.16 | Test ‘Set Timetable days’ screen |
| 3.17 | Test ‘Set Timetable times’ screen |
|  |  |
|  |  |

**Test Series 4: Full system testing**

Since much of this data relies upon data which is not static, it is hard to have tests which have set results, since the results will vary based on when the test is completed. Therefore, it is important that the results are generally reasonably easy to verify by using an application with a GUI to look at the raw data in the database and ensure that things like the timetable point to correct days. The application which is being used for this is HeidiSQL.

|  |  |
| --- | --- |
| Test Series Number | Purpose of Test Series |
| 4.1 | **Test registering** |
| 4.2 | Test logging in |
|  | **ElderlyTests** |
| 4.4 | Test list of appointments |
| 4.5 | Test one appointment |
| 4.6 | Test rating |
| 4.7 | Test review |
|  | **ElderlyTests.BookAppointment** |
| 4.9 | Test list of helpers |
| 4.10 | Test helper details |
| 4.11 | Test book appointment |
| 4.12 | **General UI Testing** |
| 4.13 | Test SMS appointment creation system |
| 4.14 | **HelperTests** |
| 4.15 | Test list of appointments |
| 4.16 | Test set timetable |
| 4.17 | **General UI Testing** |

# 3 – Implementation

See the appendices for the code.

# 4 – Testing

## 4.1 – Test Series 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Purpose / Description | Test Data | Reason | Expected Result | Passed |
| 1.1.1 | Test general select | T1.1.1 | Normal Data | E1.1.1 | Yes |
| 1.1.2 | Test incorrect syntax select | T1.1.2 | Erroneous Data | E1.1.2 | Yes |
| 1.1.3 | Test ungettable data in database | T1.1.3 | Erroneous Data | E1.1.2 | Yes |
| 1.1.4 | Test update data | T1.1.4 | Normal Data | E1.1.3 | Yes |
|  |  |  |  |  |  |
| 1.2.1 | Test with correct date and time | T1.2.1 | Normal Data | E1.2.1 | Yes |
| 1.2.2 | Test with keyword 'tomorrow' | T1.2.2 | Normal Data | E1.2.2 | Yes |
| 1.2.3 | Test without time | T1.2.3 | Boundary Data | E1.2.3 | Yes |
| 1.2.4 | Test without date | T1.2.4 | Boundary Data | E1.2.4 | Yes |
| 1.2.5 | Test with correct date and time with american format | T1.2.5 | Normal Data | E1.2.5 | Yes |
|  |  |  |  |  |  |
| 1.3.1 | Test with correct HTTP GET | T1.3.1 | Normal Data | E1.3.1 | Yes |
| 1.3.2 | Test with incorrect URL HTTP GET | T1.3.2 | Erroneous Data | E1.3.2 | Yes |
|  |  |  |  |  |  |
| 1.4.1 | Test "Hello World" | T1.4.1 | Normal Data | E1.4.1 | Yes |
| 1.4.2 | Test add new user | T1.4.2 | Normal Data | E1.4.2 | Yes |
| 1.4.3 | Test verify new user | T1.4.3 | Normal Data | E1.4.3 | Yes |
|  |  |  |  |  |  |
| 1.5.1 | Test "Hello World" | T1.5.1 | Normal Data | E1.5.1 | Yes |
| 1.5.2 | Test GetInfo Successful | T1.5.2 | Normal Data | E1.5.2 | Yes |
| 1.5.3 | Test GetInfo Fail | T1.5.3 | Erroneous Data | E1.5.3 | Yes |
| 1.5.4 | Test AddNewUser Success | T1.5.4 | Normal Data | E1.5.4 | Yes |
| 1.5.5 | Test AddNewUser AlreadyExists | T1.5.5 | Erroneous Data | E1.5.5 | Yes |
| 1.5.6 | Test Delete Success | T1.5.6 | Normal Data | E1.5.6 | Yes |
| 1.5.7 | Test Delete fail | T1.5.7 | Erroneous Data | E1.5.7 | Yes |
| 1.5.8 | Test update success | T1.5.8 | Normal Data | E1.5.8 | Yes |
| 1.5.9 | Test update failure | T1.5.9 | Erroneous Data | E1.5.9 | Yes |
| 1.5.10 | Test verifyUser fail | T1.5.10 | Erroneous Data | E1.5.10 | Yes |
| 1.5.11 | Test verifyUser success | T1.5.11 | Normal Data | E1.5.11 | Yes |
| 1.5.12 | Test GetListOfUsersByDistance success | T1.5.12 | Normal Data | E1.5.12 | Yes |
| 1.5.13 | Test GetListOfUsersByDistance incorrectId | T1.5.13 | Erroneous Data | E1.5.13 | Yes |
|  |  |  |  |  |  |
| 1.6.1 | Test "Hello World" | T1.6.1 | Normal Data | E1.6.1 | Yes |
| 1.6.2 | Test GetInfo Successful | T1.6.2 | Normal Data | E1.6.2 | Yes |
| 1.6.3 | Test GetInfo Fail | T1.6.3 | Erroneous Data | E1.6.3 | Yes |
| 1.6.4 | Test AddNewUser Success | T1.6.4 | Normal Data | E1.6.4 | Yes |
| 1.6.5 | Test AddNewUser AlreadyExists | T1.6.5 | Erroneous Data | E1.6.5 | Yes |
| 1.6.6 | Test Delete Success | T1.6.6 | Normal Data | E1.6.6 | Yes |
| 1.6.7 | Test Delete fail | T1.6.7 | Erroneous Data | E1.6.7 | Yes |
| 1.6.8 | Test update success | T1.6.8 | Normal Data | E1.6.8 | Yes |
| 1.6.9 | Test update failure | T1.6.9 | Erroneous Data | E1.6.9 | Yes |
| 1.6.10 | Test verifyUser fail | T1.6.10 | Erroneous Data | E1.6.10 | Yes |
| 1.6.11 | Test verifyUser success | T1.6.11 | Normal Data | E1.6.11 | Yes |
|  |  |  |  |  |  |
| 1.7.1 | Test create new SMS correct | T1.7.1 | Normal Data | E1.7.1 | Yes |
| 1.7.2 | Test create new SMS impossible | T1.7.2 | Erroneous Data | E1.7.2 | Yes |
| 1.7.3 | Test create new SMS unrecognised number | T1.7.3 | Erroneous Data | E1.7.3 | Yes |
| 1.7.4 | Test sendSMS | T1.7.4 | Normal Data | E1.7.4 | Yes |
| 1.7.5 | Test createNewAppointmentsForAndroidApp correct | T1.7.5 | Normal Data | E1.7.5 | Yes |
| 1.7.6 | Test GetAppointmentInformation fail | T1.7.6 | Erroneous Data | E1.7.6 | Yes |
| 1.7.7 | Test GetAppointmentInformation success | T1.7.7 | Normal Data | E1.7.7 | Yes |
| 1.7.8 | Test RemoveAppointment incorrectId | T1.7.8 | Erroneous Data | E1.7.8 | Yes |
| 1.7.9 | Test RemoveAppointment success | T1.7.9 | Normal Data | E1.7.9 | Yes |
| 1.7.10 | Test AppointmentsByElderly incorrectId | T1.7.10 | Erroneous Data | E1.7.10 | Yes |
| 1.7.11 | Test AppointmentsByElderly correctId | T1.7.11 | Normal Data | E1.7.11 | Yes |
| 1.7.12 | Test AppointmentsByHelper incorrectId | T1.7.12 | Erroneous Data | E1.7.12 | Yes |
| 1.7.13 | Test AppointmentsByHelper correctId | T1.7.13 | Normal Data | E1.7.13 | Yes |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 1.8.1 | Test GetAllRatingsForHelper correct | T1.8.1 | Normal Data | E1.8.1 | Yes |
| 1.8.2 | Test GetAllRatingsForHelper incorrect | T1.8.2 | Erroneous Data | E1.8.2 | Yes |
| 1.8.3 | Test GetOneSpecificRating correct | T1.8.3 | Normal Data | E1.8.3 | Yes |
| 1.8.4 | Test GetOneSpecificRating incorrect | T1.8.4 | Erroneous Data | E1.8.4 | Yes |
| 1.8.5 | Test AddARating | T1.8.5 | Normal Data | E1.8.5 | Yes |
| 1.8.6 | Test UpdateRating | T1.8.6 | Normal Data | E1.8.6 | Yes |
| 1.8.7 | Test DeleteRating | T1.8.7 | Normal Data | E1.8.7 | Yes |
|  |  |  |  |  |  |
| 1.9.1 | Test GetAllReviewsForHelper correct | T1.9.1 | Normal Data | E1.9.1 | Yes |
| 1.9.2 | Test GetAllReviewsForHelper incorrect | T1.9.2 | Erroneous Data | E1.9.2 | Yes |
| 1.9.3 | Test GetOneSpecificreview correct | T1.9.3 | Normal Data | E1.9.3 | Yes |
| 1.9.4 | Test GetOneSpecificReview incorrect | T1.9.4 | Erroneous Data | E1.9.4 | Yes |
| 1.9.5 | Test AddAReview | T1.9.5 | Normal Data | E1.9.5 | Yes |
| 1.9.6 | Test UpdateReview | T1.9.6 | Normal Data | E1.9.6 | Yes |
| 1.9.7 | Test DeleteReview | T1.9.7 | Normal Data | E1.9.7 | Yes |
|  |  |  |  |  |  |
| 1.10.1 | Test AddTimetable | T1.10.1 | Normal Data | E1.10.1 | Yes |
| 1.10.2 | Test GetTimetable | T1.10.2 | Normal Data | E1.10.2 | Yes |
| 1.10.3 | Test GetTimetable fail | T1.10.3 | Erroneous Data | E1.10.3 | Yes |
| 1.10.4 | Test updateWeek success | T1.10.4 | Normal Data | E1.10.4 | Yes |
| 1.10.5 | Test UpdateWeek fail | T1.10.5 | Erroneous Data | E1.10.5 | Yes |
| 1.10.6 | Test addWeek | T1.10.6 | Normal Data | E1.10.6 | Yes |
| 1.10.7 | test getWeek fail | T1.10.7 | Erroneous Data | E1.10.7 | Yes |
| 1.10.8 | Test getWeek success | T1.10.8 | Normal Data | E1.10.8 | Yes |
| 1.10.9 | Test GetDay success | T1.10.9 | Normal Data | E1.10.9 | Yes |
| 1.10.10 | Test GetDay fail | T1.10.10 | Erroneous Data | E1.10.10 | Yes |

## 4.2 – Test Series 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Purpose / Description | Test Data | Reason | Expected Result | Passed |
| 2.1.1 | Test GET on '/' | T2.1.1 | Normal Data | E2.1.1 | Yes |
| 2.2.1 | Test SMS receipt from validated number | T2.2.1 | Normal Data | E2.2.1 | Yes |
| 2.2.2 | Test SMS receipt from invalidated number | T2.2.2 | Boundary Data | E2.2.2 | Yes |
| 2.3.1 | Test /sendSMS correct | T2.3.1 | Normal Data | E2.3.1 | Yes |
| 2.3.2 | Test /sendSMS to invalidated number | T2.3.2 | Erroneous Data | E2.3.2 | Yes |
| 2.3.3 | Test /sendSMS with no body | T2.3.3 | Erroneous Data | E2.3.3 | Yes |
| 2.3.4 | Test /sendSMS with different phone format | T2.3.4 | Boundary Data | E2.3.4 | Yes |
| 2.3.5 | Test /sendSMS with incorrect phone number | T2.3.5 | Erroneous Data | E2.3.5 | Yes |
| 2.3.6 | Test /sendSMS without correct information | T2.3.6 | Erroneous Data | E2.3.6 | Yes |
| 2.4.1 | Test /verifyNumber correct | T2.4.1 | Normal Data | E2.4.1 | Yes |
| 2.4.2 | Test /verifyNumber with different phone format | T2.4.2 | Boundary Data | E2.4.2 | Yes |
| 2.4.3 | Test /verifyNumber without correct information | T2.4.3 | Erroneous Data | E2.4.3 | Yes |

## 4.3 – Test Series 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Purpose / Description | Test Data | Reason | Expected Result | Passed |
| 3.1.1 | Test login button | T3.1.1 | Normal Data | E3.1.1 | Yes |
| 3.1.2 | Test register button | T3.1.2 | Normal Data | E3.1.2 | Yes |
| 3.2.1 | Test back button | T3.2.1 | Normal Data | E3.2.1 | Yes |
| 3.2.2 | Test incomplete fields click | T3.2.2 | Erroneous Data | E3.2.2 | Yes |
| 3.3.1 | Test incomplete fields click | T3.3.1 | Erroneous Data | E3.3.1 | Yes |
| 3.3.2 | Test incomplete fields click | T3.3.2 | Erroneous Data | E3.3.2 | Yes |
| 3.3.3 | Test incomplete fields click | T3.3.3 | Erroneous Data | E3.3.3 | Yes |
| 3.3.4 | Test incomplete fields click | T3.3.4 | Erroneous Data | E3.3.4 | Yes |
| 3.3.5 | Test incomplete fields click | T3.3.5 | Erroneous Data | E3.3.5 | Yes |
| 3.3.6 | Test incomplete fields click | T3.3.6 | Erroneous Data | E3.3.6 | Yes |
| 3.3.7 | Test incomplete fields click | T3.3.7 | Erroneous Data | E3.3.7 | Yes |
| 3.3.8 | Test incomplete fields click | T3.3.8 | Erroneous Data | E3.3.8 | Yes |
| 3.3.9 | Test incomplete fields click | T3.3.9 | Erroneous Data | E3.3.9 | Yes |
| 3.3.10 | Test incomplete fields click | T3.3.10 | Erroneous Data | E3.3.10 | Yes |
| 3.3.11 | Test password not matching click | T3.3.11 | Erroneous Data | E3.3.11 | Yes |
| 3.3.12 | Test back button | T3.3.12 | Normal Data | E3.3.12 | Yes |
| 3.4.1 | Test clicking on appointment | T3.4.1 | Normal Data | E3.4.1 | Yes |
| 3.4.2 | Test back button | T3.4.2 | Normal Data | E3.4.2 | Yes |
| 3.5.1 | Test back button | T3.5.1 | Normal Data | E3.5.1 | Yes |
| 3.5.2 | Test set rating | T3.5.2 | Normal Data | E3.5.2 | Yes |
| 3.5.3 | Test set review | T3.5.3 | Normal Data | E3.5.3 | Yes |
| 3.6.1 | Test back button | T3.6.1 | Normal Data | E3.6.1 | Yes |
| 3.7.1 | Test back button | T3.7.1 | Normal Data | E3.7.1 | Yes |
| 3.8.1 | Test see reviews | T3.8.1 | Normal Data | E3.8.1 | Yes |
| 3.8.2 | Test book appointment | T3.8.2 | Normal Data | E3.8.2 | Yes |
| 3.9.1 | Test back button | T3.9.1 | Normal Data | E3.9.1 | Yes |
| 3.10.1 | Test back button | T3.10.1 | Normal Data | E3.10.1 | Yes |
| 3.10.2 | Test clicking on week | T3.10.2 | Normal Data | E3.10.2 | Yes |
| 3.11.1 | Test back button | T3.11.1 | Normal Data | E3.11.1 | Yes |
| 3.11.2 | Test clicking on day | T3.11.2 | Normal Data | E3.11.2 | Yes |
| 3.12.1 | Test back button | T3.12.1 | Normal Data | E3.12.1 | Yes |
| 3.12.2 | Test clicking on time | T3.12.2 | Normal Data | E3.12.2 | Yes |
| 3.13.1 | Test back button | T3.13.1 | Normal Data | E3.13.1 | Yes |
| 3.14.1 | Test submit button | T3.14.1 | Normal Data | E3.14.1 | Yes |
| 3.15.1 | Test back button | T3.15.1 | Normal Data | E3.15.1 | Yes |
| 3.15.2 | Test clicking on week | T3.15.2 | Normal Data | E3.15.2 | Yes |
| 3.16.1 | Test back button | T3.16.1 | Normal Data | E3.16.1 | Yes |
| 3.16.2 | Test clicking on day | T3.16.2 | Normal Data | E3.16.2 | Yes |
| 3.17.1 | Test back button | T3.17.1 | Normal Data | E3.17.1 | Yes |

## 4.4 – Test Series 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Purpose / Description | Test Data | Reason | Expected Result | Passed |
| 4.1.1 | Test twilio verification fail | T4.1.1 | Erroneous Data | E4.1.1 | Yes |
| 4.1.2 | Test twilio verification success | T4.1.2 | Normal Data | E4.1.2 | Yes |
| 4.1.3 | Test no internet | T4.1.3 | Boundary Data | E4.1.3 | Yes |
| 4.2.1 | Test no internet | T4.2.1 | Boundary Data | E4.2.1 | Yes |
| 4.2.2 | Test incorrect password | T4.2.2 | Boundary Data | E4.2.2 | Yes |
| 4.2.3 | Test correct password | T4.2.3 | Normal Data | E4.2.3 | Yes |
| 4.2.4 | Test autologin | T4.2.4 | Normal Data | E4.2.4 | Yes |
| 4.4.1 | Test and see all appointments | T4.4.1 | Normal Data | E4.4.1 | Yes |
| 4.5.1 | Test and see one appointment accurate | T4.5.1 | Normal Data | E4.5.1 | Yes |
| 4.5.2 | Test past appointment | T4.5.2 | Normal Data | E4.5.2 | Yes |
| 4.5.3 | Test current appointment | T4.5.3 | Normal Data | E4.5.3 | Yes |
| 4.5.4 | Test cancel appointment | T4.5.4 | Normal Data | E4.5.4 | Yes |
| 4.6.1 | Test rating | T4.6.1 | Normal Data | E4.6.1 | Yes |
| 4.7.1 | Test review | T4.7.1 | Normal Data | E4.7.1 | Yes |
| 4.9.1 | Test list of helpers on account with failed postcode | T4.9.1 | Erroneous Data | E4.9.1 | Yes |
| 4.9.2 | Test list of helpers on account with successful postcode | T4.9.2 | Normal Data | E4.9.2 | Yes |
| 4.10.1 | Test helper details | T4.10.1 | Normal Data | E4.10.1 | Yes |
| 4.10.2 | Test see reviews | T4.10.2 | Normal Data | E4.10.2 | Yes |
| 4.11.1 | Test book appointment in past | T4.11.1 | Erroneous Data | E4.11.1 | Yes |
| 4.11.2 | Test book appointment on used date | T4.11.2 | Boundary Data | E4.11.2 | Yes |
| 4.11.3 | Test book appointment successful | T4.11.3 | Normal Data | E4.11.3 | Yes |
| 4.11.4 | Test book appointment and then cancel | T4.11.4 | Normal Data | E4.11.4 | Yes |
| 4.11.5 | Test cancel appointment | T4.11.5 | Normal Data | E4.11.5 | Yes |
| 4.13.1 | Test SMS send success | T4.13.1 | Normal Data | E4.13.1 | Yes |
| 4.13.2 | Test SMS send incorrect data | T4.13.2 | Erroneous Data | E4.13.2 | Yes |
| 4.13.3 | Test SMS when no space in anyone | T4.13.3 | Boundary Data | E4.13.3 | Yes |
| 4.14.1 | Test set timetable and see if changes | T4.14.1 | Normal Data | E4.14.1 | Yes |
| 4.14.2 | Test set timetable and see if you can book appointment | T4.14.2 | Normal Data | E4.14.2 | Yes |

### 4.12 and 4.17

In order to attempt to complete test 4.12 and 4.17, the applications were placed into the hands of two random users (different to the end users), with a bare minimum explanation of the purpose of the application to see if they could complete the following tasks:

#### Elderly App

1. Create an account
2. Book an appointment with a nearby helper tomorrow after looking at the helper’s rating and review.
3. Cancel the appointment.
4. Book another appointment tomorrow.
5. (At this point, the raw data was altered and the appointment changed to be in the past)
6. Rate the helper.
7. Review the helper.
8. Book an appointment with an SMS message for two days time at 12pm.

#### Helper App

1. Create an account
2. Set the timetable to reasonable hours for the next two weeks.
3. (At this point an appointment was created with the user using another device)
4. Cancel the newly created appointment.

Both test users managed to complete all tasks with no additional assistance, so the tests can be deemed to be successful.

# 5 – Evaluation

|  |  |  |
| --- | --- | --- |
| Objective | Objective met? | Details |
| 1 - Input |  |  |
| There must be an Android app, which works with all Android phones with software later than Android Ice Cream Sandwich (4.0). | Yes | Though the application has only been tested to be working above Android 5.0 (due to a lack of test devices), it has been simulated at lower device versions (including 4.0) and has been working. |
| The elderly person should also be able to create an appointment with an SMS messages to a specified phone number. | Yes | Tests 4.13 show that this system is working. |
| The SMS should only require the user to send a Date and Time and the information about the appointment should be communicated directly back to the user over SMS. | Yes | Tests 4.13 show that the system works when a date and time is sent over SMS and successfully returns errors when not all data is present.  4.13.1 shows the receipt of the information of an appointment. |
| To log onto the Android application, the system should remember the password of the user once they have logged in once, thereby allowing them to not need to remember the password. | Yes | Test 4.2.4 shows that the autologin system ensures that users do not need to log onto the system multiple times on the same device. |
| 2 - User Interface (on all systems) |  |  |
| The screens should be as simple as possible, removing unnecessary information to other screens if necessary. | Yes | Test 4.12 shows that the interface is very simple as a person with little instruction is able to use it.  Additionally, having given it to an elderly neighbour at a few points throughout the development, he now believes it is at a point at which anybody can use the system with little to no training. |
| As a maximum, there should be two possible (very different) paths to follow from every screen, with the difference in what each button does being marked clearly. | Yes | Looking at the design shows that there are never more than two different paths through the application at one time – with the exception of ListViews, where each lead to the same path, just with different data. |
| All text should be large, above 12pt, to ensure that the elderly person can see it easily, with worse eye-sight. | Yes | Looking at the code, it is clear to see that all text is at least of size 12pt. |
| There should be a screen where the user will be able to scroll through the nearby teenagers who can provide tech support. | Yes | This is the ‘Book Appointment’ screen, shown in section 2.4.3 and tested in 4.9.2. |
| This screen should be organised by how near the helper is, therefore meaning that the helper is unlikely to have to travel from very far. | Yes | This is tested and shown to be working in test 4.9.2. |
| For each teenager, the elderly person should be able to see their average rating (if they have been rated), location and all of their reviews. | Yes | The average rating is visible in the ‘Book Appointment’ screen (2.4.3), while the ‘See Reviews’ screen is described in 2.4.3 and tested in 4.10.2 |
| The user should be able to look at the various appointments a teenager has free, allowing them to choose a time which would suit them. | Yes | The user can view the times that the helpers are free over the next four weeks, therefore being able to choose whichever one of them suits the elderly person best. |
| They should be able to book an appointment for any available time at least within the next two weeks. | Yes | They can in fact pick any appointment within the next four weeks. |
| The user should be able to see the list of upcoming and past appointments, seeing the details about the appointment. | Yes | They can see all this information in the ‘List of Appointments’ screen, described in 2.4.2 and tested in 4.4.1. |
| For appointments in the past, the user should be able to rate and review the helper, therefore helping future elderly people to decide whether the helper would be able to help them. | Yes | Designed in 2.4.3 and tested in 4.5.2 vs 4.5.3. |
| For appointments in the future, the user should be able to cancel the appointment, if they are no longer free. | Yes | This was designed in 2.4 and tested in 4.5.4 |
| 3 – Teenager’s Input |  |  |
| There must be an Android app through which the person should be able to interact with the system. | Yes | There is an android app which has been designed in section 2.4 and tested in section 4. |
| There should be a system to ensure to remember the password for the android application, so that they would not have to remember the password every time they log onto the application. | Yes | Test 4.2.4 shows that the autologin system ensures that users do not need to log onto the system multiple times on the same device. |
| 4 – Teenager’s User Interface |  |  |
| The user must be able to see everyone who has requested their help. | Yes | They can see all this information in the ‘List of Appointments’ screen, described in 2.4.2 and tested in 4.4.1. |
| They should be able to cancel any appointments which they are no longer able to be available for. | Yes | This was designed in 2.4 and tested in 4.5.4 |
| The user should also be able to define when they are free for the next six weeks, allowing them to set the times that they would be free for appointments. | Yes | This ability was designed in 2.4.2 and tested in 4.14. The application allows the user to set their times that they are free for the next eight weeks, firstly choosing the week to edit, then the day and finally choosing the times that they are free for that day. |
| 5 - The appointments should be two hours in length for all appointments. | Yes | As throughout the design and in the implementation, the appointments are designed to be two hours long, running between even hours. |
| 6 - The payment should be defined as £20 per hour for the appointments, therefore being £20 if the appointment takes up less than half the appointment slot, or £40 otherwise. | Yes | The payment, while not being discussed throughout the app has been defined to be £20 per hour and this should be discussed in any user manual. |
| 7 - The system should be secure, with passwords hashed to ensure that they are not easy to utilise in case the server is hacked. | Yes | The passwords are hashed with MD5, see section 2.3.3 |

## 5.1 – User Feedback

### 5.1.1 – Mrs Wilson (AgeUK)

### 5.1.2 – Phil Fernandes

Phil largely found that the system was exceedingly effective and easily met his requirements. He commented on the ease of use of the system, saying that anyone could definitely use the helper side of the software, which, though not a particularly important a thing (this was not a specification point), is always a positive.

He says the interface is intuitive and easy to use, as long as there is a little bit of understanding with regards to what the application does. Therefore, if the application were to be more broadly launched, it would be useful to produce a ‘User Manual’ or something to this effect which could briefly show how to use the application.

He mentioned that the one area where he anticipated that problems may have arisen from, the setting of the timetable and booking of timetables has an effective interface, by allowing the users to first choose the week, then the day and then edit the hour or book an appointment at a particular hour.

He said that there were a few possible issues which could be addressed is simplifying this process even more might be useful, attempting to bring in a calendar-esque interface, rather than this approach. Though perhaps harder to produce, the calendar approach would allow for more effective usage of the application.

He also mentioned that the aspect of no maximum distance for each helper user being unhelpful and something which should be added if there were to be time.

Finally, he mentioned that there were a couple of issues related to the speed of the application, with the login and ‘Book Appointment’ screens taking longer than could be hoped, as they were processing data.

## 5.2 – Analysis

### 5.2.1 – Learnability

Throughout the process, the easy of use of the application by the elderly has been foremost in my mind, with a number of tests throughout the programming, to ensure that the application could be used by a random person without any instruction. This culminated in test 4.12, which shows that someone, given little to no instruction, can perform a set of tasks with the application. This, alongside the comments by Mrs Wilson and Phil seem to vindicate this effort as they all mention that they find it very easy to use and therefore usable by all elderly people without any issues.

However, even given this, it would almost certainly be useful, before the product were to be released, to produce a User Manual to outline the various things the application can do and outline how they can be done.

The User Manual would also offer an opportunity to shine a light on two areas where the least information is provided to users, payment and the SMS messaging system. At no point in the application is payment discussed, despite a fixed fee system having been decided in the Analysis. In the application however, there is no correct page where this information would fit. Additionally, the exact format for SMS messages to be sent by the elderly person to the server and the phone number to send them to is not properly defined and could be easily made mistakes on. Though not a particularly complex issue, not a challenging one to talk about, it would be an important thing to mention in any user manual.

### 5.2.2 – Latency

Phil complained that there was too much of a time lag in the transition between two screens, the login screen and the welcome screen (for the elderly) and between the welcome screen and ‘Book Appointment’ (i.e. the list of nearby helpers) screen.

For the login screen, the time lag is almost certainly down to the time taken by the MD5 hashing algorithm. The MD5 algorithms gets is impressive security by being vastly complex and algorithm and therefore this conversion inevitably will take some time. Additionally, since this is only likely to be completed once every time the app is used, this seems acceptable.

For the book appointment screen, it seems far more a consequence of the speed of the Google Maps API, since it returns lots of information, including the route between the two points, traffic, etc. Since all I need in fact, is a longitude and latitude for each point, and then the distance can be calculated using Haversine Formula, this should be saved in the database and calculated upon registration. Alternatively, another API, such as the OpenMaps API could be used instead for simpler results. Another important matter to consider when considering the solution is that if this were to be scaled up, there might be problems due to the number of calls made to the Google Maps API – which limits this to ten per second, unless you pay for the paid account.

### 5.2.3 – Improvements

In order to improve the system, it would be better to store the location of each helper as a longitude and latitude in the database as this would allow for a much faster sorting of the system.

Additionally, it would be better to make use of a more significant Natural Language processing system, since the one I am currently using is very specific as to the information it requires to work. For example, if a year is not specified when the entirety of a date is specified then it assumes the date is today’s date, rather than parsing the rest of the date and assuming the year is this year (the far more logical assumption). Additionally, it relies upon the user abiding by a specific timing format (HH:MM). If I had used a library like IBM Watson’s inbuilt NLP API, it would be able to deal with all formats, in addition to dealing with things like ‘quarter past’ etc which are substituted instead of the time. It would also continue to work even if there are a few spelling mistakes, something which currently prevents the system from parsing the date.

### 5.2.4 – Extensions

**iOS Applications**

The first extension would be to ensure that the application could be produced for iOS to ensure that elderly people and teenagers with iOS phones and tablets could also use the system. This by far extends the market for the solution.

In fact, since the Android application was produced using Xamarin, this shouldn’t pose too much of a challenge as the vast majority of the code could be reused if the application for iOS were to be coded in Xamarin iOS.

**Tablet Specific Application**

Another extension would be to produce a tablet specific application, especially catering to cheap Android tablets, which have proved very successful with the elderly people, according to Mrs Wilson. Though the application as it currently stands can be used on a tablet, the system should be redesigned to cater for the larger screen and the added functionality which can be gotten from this.

**Variable time-slot appointments**

As mentioned by Phil, another change that could be made is introducing variable length appointments or offering a number of possible lengths. This would increase the efficiency of the process as the teenagers could help more elderly people in the same amount of time. This would be especially important when the elderly person knows the problem would be quick to solve but can’t solve it themselves.

**Maximum travelable distance for helpers**

The current solution has no method to stop elderly people booking appointments with teenagers who live hours, or even further, away. Therefore, if a maximum distance decision could be added when the helper signs up for an account, this can be taken into account into whether to display their account in the ‘Book an Appointment’ view for a specific elderly person, by simply comparing this decided maximum distance with the distance between the helper and the elderly person.

# Appendix A – Web API Code

## Backend

### HttpCommandOut.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Net.Http;

using System.Net;

using System.IO;

namespace TestApi.Backend

{

public class HTTPCommandOut

{

/// <summary>

/// Completes a HTTP Get

///

/// </summary>

public HTTPCommandOut() { }

/// <summary>

/// Since it's asynchronous, it returns a task. However, in reality, the only

/// thing which is used from the task return is the result.

/// </summary>

/// <param name="url">This is the URL to complete the GET command from</param>

/// <returns>Returns a Task (of a stream). From here, the stream can be created and the contents read using a StreamReader</returns>

public async Task<Stream> get(string url)

{

using (var client = new HttpClient()) // The using ensures the HTTPClient cannot be used outside of scope.

{

client.BaseAddress = new Uri(url); // Important to note that since it is used directly as a URI, the url must be fully defined

// i.e. also including the http://

var response = await client.GetAsync(""); // Asynchronously calls the task

Stream stringResponse = await response.Content.ReadAsStreamAsync(); //Gets the response

StreamReader x = new StreamReader(stringResponse); //

return stringResponse; // Returns the task of the stream

}

}

}

}

### SMSStuff.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Net.Http;

using System.IO;

using TestApi.Backend;

using TestApi.Controllers;

using Newtonsoft.Json;

using TestApi.Types;

using System.Text;

namespace TestApi.Backend

{

public class sms

{

/// <summary>

/// Send SMS to the number listed.

/// Method is asynchronous as it calls assynchronous methods in HTTPClient methods.

/// </summary>

/// <param name="from">The phone number to send the SMS to</param>

/// <param name="body">The contents of the message to send</param>

async void sendSMS(string from, string body)

{

Controllers.sms s = new Controllers.sms(); // Creates an SMS object

s.Body = body;

s.From = from;

string json = JsonConvert.SerializeObject(s); // Converts the object into JSON

string url = "http://178.62.87.28:700/sendSms"; // Defines the URL for the POST call to the SMS server.

var httpContent = new StringContent(json, Encoding.UTF8, "application/json"); // Adds the correct header to the JSON

using (var httpClient = new HttpClient())

{

var httpResponse = await httpClient.PostAsync(url, httpContent);

if (httpResponse.Content != null) // if there is a response - deal with it.

{

var responseContent = await httpResponse.Content.ReadAsStringAsync(); // gets the value of the response

}

}

}

}

}

### SQLStuff.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MySql.Data.MySqlClient;

using MySql.Data.Common;

using MySql.Data.Types;

namespace TestApi.Sql

{

public class SQLStuffOnTopOfController : Controller

{

public const bool server = false; // on the test machine, this is set to false to set the url to point to the server. Otherwise, for speed, it should point to localhost

/// <summary>

/// Returns a command - follows the following process:

/// (1) Creates a new connection with the db.

/// (2) Completes the SQL query

/// (3) Returns the value got from the MySQL server

/// </summary>

/// <param name="isResultExpected">true if a result is expected, false if not</param>

/// <param name="command">String value of the query</param>

/// <param name="numberExpectedPerRow">Defaults at one, but otherwise knows how to split the response</param>

/// <returns></returns>

public static string sqlCommand(bool isResultExpected, string command, int numberExpectedPerRow = 1)

{

MySqlConnection dbConnection = setupConnection(); // Setup Connection and return MySqlConnection object

MySqlCommand commandForSql = new MySqlCommand(command, dbConnection);

if(isResultExpected)

{

MySqlDataReader a = commandForSql.ExecuteReader();

string toReturn = String.Empty;

while(a.Read())

{

for (int i = 0; i < numberExpectedPerRow; i++)

{

string ab = a.GetDataTypeName(i); //In case we are expecting a DateTime - deal with this

if (ab == "DATETIME")

{

toReturn += a.GetDateTime(i).ToString() + "#"; // Gets the DateTime as a DateTimeObject which can definitely be parsed by a DateTime.Parse()

}

else

toReturn += a.GetValue(i) + "#";

}

//# is used to seperate items in the same line

toReturn += "\n";

//NewLine ('\n') is used to seperate different lines

}

a.Close(); //For stability, the DataReader is closed

closeConnection(dbConnection); // Connection is closed

return toReturn;

}

else

{

commandForSql.ExecuteScalar(); // just execute the command

}

closeConnection(dbConnection); // Close the connection

return String.Empty; // if you haven't already returned the string, then return an empty string

}

public static void closeConnection(MySqlConnection a)

{

a.Close();

}

/// <summary>

/// Setup connection

/// (1) Defines the correct string for the connection

/// (2) Sets up the connection and returns the connection object.

/// </summary>

public static MySqlConnection setupConnection()

{

string connectionString = String.Empty;

if(server)

connectionString = "server=localhost;user id=ashwin;password=Poonam123;port=3306;database=ComputingProject;SslMode=None;Convert Zero DateTime=True;Allow Zero DateTime=True";

else

connectionString = "server=178.62.87.28;user id=ashwin;password=Poonam123;port=3306;database=ComputingProject;SslMode=None;Convert Zero DateTime=True;Allow Zero DateTime=True";

MySqlConnection dbConnection = new MySqlConnection(connectionString);

dbConnection.Open();

return dbConnection;

}

}

}

## Types

### Appointment.cs

using System;

namespace TestApi.Types

{

public class appointment : Object

{

public int id;

public int helperId;

public int elderlyId;

public DateTime dateAndTime;

public DateTime dateCreated;

public int ratingId;

public int reviewId;

public static bool Equals(appointment a, appointment b)

{

if (a.id != b.id)

return false;

if (a.helperId != b.helperId)

return false;

if (a.elderlyId != b.elderlyId)

return false;

if (DateTime.Compare(a.dateAndTime, b.dateAndTime) != 0)

return false;

if (DateTime.Compare(a.dateCreated, b.dateCreated) != 0)

return false;

if (a.ratingId != b.ratingId)

return false;

if (a.reviewId != b.reviewId)

return false;

return true;

}

}

}

### Day.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestApi.Types

{

public class day

{

public int id { get; set; }

public bool[] timesFree = { false, false, false, false, false, false, false, false, false, false, false, false };

}

}

### Google.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TestApi.Types

{

public class Northeast

{

public double lat { get; set; }

public double lng { get; set; }

}

public class Southwest

{

public double lat { get; set; }

public double lng { get; set; }

}

public class Bounds

{

public Northeast northeast { get; set; }

public Southwest southwest { get; set; }

}

public class Distance

{

public string text { get; set; }

public int value { get; set; }

}

public class Duration

{

public string text { get; set; }

public int value { get; set; }

}

public class EndLocation

{

public double lat { get; set; }

public double lng { get; set; }

}

public class StartLocation

{

public double lat { get; set; }

public double lng { get; set; }

}

public class Distance2

{

public string text { get; set; }

public int value { get; set; }

}

public class Duration2

{

public string text { get; set; }

public int value { get; set; }

}

public class EndLocation2

{

public double lat { get; set; }

public double lng { get; set; }

}

public class Polyline

{

public string points { get; set; }

}

public class StartLocation2

{

public double lat { get; set; }

public double lng { get; set; }

}

public class Step

{

public Distance2 distance { get; set; }

public Duration2 duration { get; set; }

public EndLocation2 end\_location { get; set; }

public string html\_instructions { get; set; }

public Polyline polyline { get; set; }

public StartLocation2 start\_location { get; set; }

public string travel\_mode { get; set; }

public string maneuver { get; set; }

}

public class Leg

{

public Distance distance { get; set; }

public Duration duration { get; set; }

public string end\_address { get; set; }

public EndLocation end\_location { get; set; }

public string start\_address { get; set; }

public StartLocation start\_location { get; set; }

public List<Step> steps { get; set; }

public List<object> via\_waypoint { get; set; }

}

public class OverviewPolyline

{

public string points { get; set; }

}

public class Route

{

public Bounds bounds { get; set; }

public string copyrights { get; set; }

public List<Leg> legs { get; set; }

public OverviewPolyline overview\_polyline { get; set; }

public string summary { get; set; }

public List<object> warnings { get; set; }

public List<object> waypoint\_order { get; set; }

}

public class RootObject

{

public List<Route> routes { get; set; }

public string status { get; set; }

}

}

### Rating.cs

using System;

namespace TestApi.Controllers

{

public class rating

{

public int id;

public int starRating;

public int helperId;

}

}

### Review.cs

using System;

namespace TestApi.Controllers

{

public class review

{

public int id;

public int helperId;

public string comment;

}

}

### SMS.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestApi.Controllers

{

public class sms

{

public string From { get; set; }

public string Body { get; set; }

}

}

### Timetable.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestApi.Types

{

public class names

{

public DateTime weekBeginning { get; set; }

public week week { get; set; }

}

public class timetable

{

public int id { get; set; }

public week defaultWeek { get; set; }

public List<names> weeks = new List<names>();

}

}

### User.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestApi.Types

{

public class user

{

public int id {get;set;}

public string username {get;set;}

public string firstName { get; set; }

public string surname { get; set; }

public string password { get; set; }

public string emailAddress { get; set; }

public string firstLineOfAddress { get; set; }

public string secondLineOfAddress{get;set;}

public string telephoneNumber {get;set;}

public string postalCode{get;set;}

public string city {get; set;}

public string country {get;set;}

private bool \_helper = false;

public bool helper { get { return \_helper; } set { \_helper = value; } }

public int timetableId { get; set; }

private double \_distance = double.MaxValue;

public double distance { get { return \_distance; } set { \_distance = value; } }

}

}

### UserToBeVerified.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestApi.Controllers

{

public class userToBeVerified

{

public string emailAddress {get; set;}

public string password {get;set;}

public bool helper { get; set; }

}

}

## Controllers

# Appendix B – Android App Code

# Appendix C – SMS Server Code

# Appendix D – Test Data and Expected Results