TO

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Course: COM547 – Computing Systems

Supervisor: Dr Colin Turner

Date: 22/04/2018

Project Final Report

Student: B00348751 – Paul Connolly

# Acknowledgements

Contents

[Acknowledgements ii](#_Toc512194086)

[1 Introduction 1](#_Toc512194087)

[1.1 Background 1](#_Toc512194088)

[1.2 Project Aims 5](#_Toc512194089)

[1.3 Project Objectives 5](#_Toc512194090)

[1.4 Project Activities 5](#_Toc512194091)

[1.5 Outline of Dissertation Structure 6](#_Toc512194092)

[2 Chatbots in Use Today 8](#_Toc512194093)

[2.1 What is a chatbot? 8](#_Toc512194094)

[2.2 Review of Chatbots 8](#_Toc512194095)

[2.2.1 Marvel 8](#_Toc512194096)

[2.2.2 RoofAi 8](#_Toc512194097)

[2.2.3 Lemonade 11](#_Toc512194098)

[3 Technical Background 13](#_Toc512194099)

[3.1 Frameworks 13](#_Toc512194100)

[3.1.1 Microsoft Bot Framework 13](#_Toc512194101)

[3.1.2 Wit.ai 14](#_Toc512194102)

[3.1.3 DialogFlow 15](#_Toc512194103)

[3.2 Development Language Options 16](#_Toc512194104)

[3.2.1 C# 16](#_Toc512194105)

[3.2.2 Ruby 16](#_Toc512194106)

[3.2.3 Python 16](#_Toc512194107)

[3.2.4 JavaScript 17](#_Toc512194108)

[3.3 Software Technologies 17](#_Toc512194109)

[3.3.1 REST 17](#_Toc512194110)

[3.3.2 JSON 17](#_Toc512194111)

[3.3.3 HTTPS 17](#_Toc512194112)

[3.4 Databases 18](#_Toc512194113)

[3.5 Source Control 18](#_Toc512194114)

[4 Development Life Cycle 19](#_Toc512194115)

[4.1 Software Development Process Models Considered 19](#_Toc512194116)

[4.1.1 Waterfall Model 19](#_Toc512194117)

[4.1.2 Spiral Model 20](#_Toc512194118)

[4.1.3 Agile Model 22](#_Toc512194119)

[5 Requirements Gathering & Risk Analysis 24](#_Toc512194120)

[5.1 Requirements Gathering Techniques 24](#_Toc512194121)

[5.2 Requirements Specification 24](#_Toc512194122)

[5.2.1 Functional Requirements 25](#_Toc512194123)

[5.2.2 Non-functional Requirements 25](#_Toc512194124)

[5.3 Requirements Validation 25](#_Toc512194125)

[5.4 Risk Analysis 26](#_Toc512194126)

[5.5 Data Protection 27](#_Toc512194127)

[6 Project Plan 28](#_Toc512194128)

[6.1 Project Scope 28](#_Toc512194129)

[6.2 Project Schedule 28](#_Toc512194130)

[6.2.1 Project Milestones 28](#_Toc512194131)

[6.2.2 Work Breakdown Structure 28](#_Toc512194132)

[6.3 Project Resources 29](#_Toc512194133)

[7 Design 30](#_Toc512194134)

[7.1 Conversation Flows 30](#_Toc512194135)

[7.2 Conversation Dialogs 31](#_Toc512194136)

[7.2.1 First Interaction 31](#_Toc512194137)

[7.2.2 Motor Dialog 31](#_Toc512194138)

[7.2.3 Home Dialog 31](#_Toc512194139)

[7.3 User Interface 31](#_Toc512194140)

[7.4 High-level System Architecture 32](#_Toc512194141)

[7.5 Class Diagram 32](#_Toc512194142)

[7.6 Database Design 32](#_Toc512194143)

[8 Implementation 32](#_Toc512194144)

[8.1 Development Environment 32](#_Toc512194145)

[8.2 Asynchronous Programing 33](#_Toc512194146)

[8.3 Solution Structure 33](#_Toc512194147)

[8.4 Technical Points of Interest 33](#_Toc512194148)

[8.4.1 Application Entry Point 33](#_Toc512194149)

[8.4.2 RootDialog.cs 35](#_Toc512194150)

[8.4.3 MotorDialog.cs 36](#_Toc512194151)

[8.4.4 HomeDialog.cs 40](#_Toc512194152)

[8.4.5 Models 41](#_Toc512194153)

[8.4.6 EmailHandler.cs 42](#_Toc512194154)

[8.4.7 Enums 44](#_Toc512194155)

[8.4.8 Helpers 47](#_Toc512194156)

[8.4.9 QuotingBot.DAL 48](#_Toc512194157)

[8.4.10 QuotingBot.DbUp 48](#_Toc512194158)

[9 Challenges and Solutions 49](#_Toc512194159)

[10 Testing and Results 49](#_Toc512194160)

[11 Evaluation 49](#_Toc512194161)

[12 Conclusions 49](#_Toc512194162)

[13 Suggested Future Improvements to Project 49](#_Toc512194163)

[13.1 Link to Applied Systems eQuote Leads System 50](#_Toc512194164)

[13.2 Add Natural Language Processing 50](#_Toc512194165)

[13.3 Convert to JavaScript Widget 51](#_Toc512194166)

[References 52](#_Toc512194167)

[Appendices 56](#_Toc512194168)

Table of Figures

[Figure 1 - Social Networking Tracker (Jan 2017) 3](#_Toc512194044)

[Figure 2 - Social Messaging Tracker (Jan 2017) 4](#_Toc512194045)

[Figure 3 - Roof.ai chatbot example (Moubarak, 2018) 9](#_Toc512194046)

[Figure 4 - Roof.ai widget on The Keyes Company website homepage (The Keyes Company, 2018) 9](#_Toc512194047)

[Figure 5 - The Keyes Company example conversation with Roof.ai chatbot 10](#_Toc512194048)

[Figure 6 - Lemonade chat example (Lemonade, 2018) 11](#_Toc512194049)

[Figure 7 - Example of Lemonade question with options (Lemonade, 2018) 12](#_Toc512194050)

[Figure 8 - HTTP vs HTTPS (Comodo, 2018) 18](#_Toc512194051)

[Figure 9 - Software Development Life Cycle (Gordiyenko, 2014) 20](#_Toc512194052)

[Figure 10 - Diagram of Spiral Model (Boehm, 1988) 21](#_Toc512194053)

[Figure 11 - Representation of the Agile Development Model (ISTQB Exam Certification, 2018) 22](#_Toc512194054)

[Figure 12 - Risk Register 27](#_Toc512194055)

[Figure 13 - High level system architecture 32](#_Toc512194056)

[Figure 14 - Post endpoint in MessagesController.cs 34](#_Toc512194057)

[Figure 15 - HandleSystemMessage method from MessagesController.cs 34](#_Toc512194058)

[Figure 16 - StartAsync method from RootDialog.cs 35](#_Toc512194059)

[Figure 17 - ShowQuoteOptions from RootDialog.cs 35](#_Toc512194060)

[Figure 18 - OnOptionSelected method from RootDialog.cs 36](#_Toc512194061)

[Figure 19 - Class variables in RootDialog.cs 36](#_Toc512194062)

[Figure 20 - StartAsync method from MotorDialog.cs 36](#_Toc512194063)

[Figure 21 - FormBuilder example from MotorDialog.cs 38](#_Toc512194064)

[Figure 22 - Field example from MotorDialog.cs 39](#_Toc512194065)

[Figure 23 - Code for requesting quotes from Motor web service 39](#_Toc512194066)

[Figure 24 - Completing the dialog 40](#_Toc512194067)

[Figure 25 - GetQuoteReceipts() with LINQ example 40](#_Toc512194068)

[Figure 26 - Section of HomeDialog.cs 40](#_Toc512194069)

[Figure 27 - Code for calling Home web service 40](#_Toc512194070)

[Figure 28 - Example of bot validation when wrong answer is given where Yes/no is expected 41](#_Toc512194071)

[Figure 29 - Risk defaulting snippet 42](#_Toc512194072)

[Figure 30 - EmailHandler usage in HomeDialog.cs 42](#_Toc512194073)

[Figure 31 - EmailHandler in MotorDialog.cs 43](#_Toc512194074)

[Figure 32 - SmtpClient() from EmailHandler.cs 43](#_Toc512194075)

[Figure 33 - SendEmailToUser() 44](#_Toc512194076)

[Figure 34 - SendEmailToBroker() 44](#_Toc512194077)

[Figure 35 - Example of question being asked based on enum data type 45](#_Toc512194078)

[Figure 36 - Emoji.cs 45](#_Toc512194079)

[Figure 37 - EnumExtension.cs 46](#_Toc512194080)

[Figure 38 - ConvertLicenceType() 46](#_Toc512194081)

[Figure 39 - Use of the ConvertLicenceType() 47](#_Toc512194082)

[Figure 40 - CapitilizeFirstLetter() in Formatter.cs 47](#_Toc512194083)

[Figure 41 - StoreQuote() in QuoteRepository.cs 48](#_Toc512194084)

[Figure 42 - Program.cs of QuotingBot.DbUp 49](#_Toc512194085)

# 1 Introduction

## 1.1 Background

Applied Systems have operated for over 30 years powering the insurance industry across the USA, Canada, Ireland and the United Kingdom, providing industry leading technology to the insurance industry (Applied Systems, 2018).

As a company; Applied Systems strive to provide insurance brokers with innovative software solutions to maximise the broker’s business profits and improve customer communications.

As an employee of Applied Systems, and being sponsored by them during my university studies, the challenge was set for me to use my Computing Systems Project as an opportunity to research, design and implement a proof-of-concept for the next innovative piece of software that they may put in to production for release to the market.

Considering my background as a Software Developer with experience working on web-based products, I decided to research emerging trends on the web in relation to business-to-customer interactions in the Insurtech (Insurance Technology) industry.

Research suggests that the number one predicted trend in the Insurtech market as predicted by “The Digital Insurer” is “automation will replace human effort across the entire insurance value chain” (Huckstep, 2017). Although this is not a trend isolated to the insurance industry, it is likely to have significant impact in the insurance world as a lot of the industry business methods continue to operate in a manner that is outdated and is more suited to consumers prior to the Internet. Given the exponential use of the Internet, consumers now want a full digital experience without need for human interaction (Huckstep, 2017).

The reality of a more digitalised consumer base is supported by Price Waterhouse Cooper’s (PwC) Irish Total Retail Survey. The survey showed found that 48% of Irish consumers had used their mobile phones to shop online at least a few times a year, and 30% felt their mobile would become their main method of shopping in the future (PwC, 2017).

Another prediction for 2018 comes from Jay Samit, a renowned digital media expert, who believes this year will be “the year of the bots” (Samit, 2017). Samit explains chatbots will become more intelligent in the use and understanding of natural language to become more capable of helping us with our daily routines.

Further research carried out into the use of social media platforms revealed that, of those surveyed, 64% of people living in Ireland are currently using Facebook (Figure 1), along with 58% using Facebook Messenger as their media messenger of choice (Figure 2).

Considering the aforementioned I decided to design and build an insurance chatbot for release to Facebook Messenger.

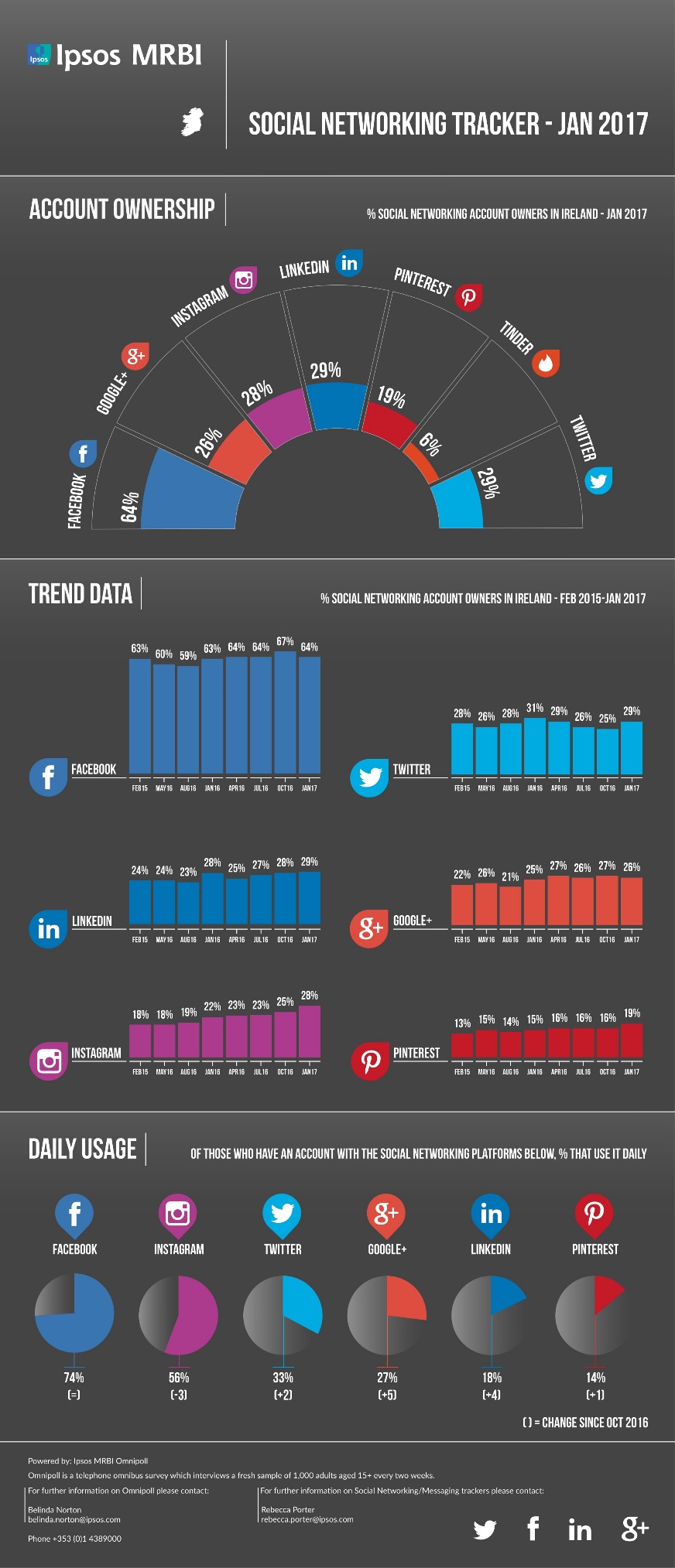


Figure 1 - Social Networking Tracker (Jan 2017)

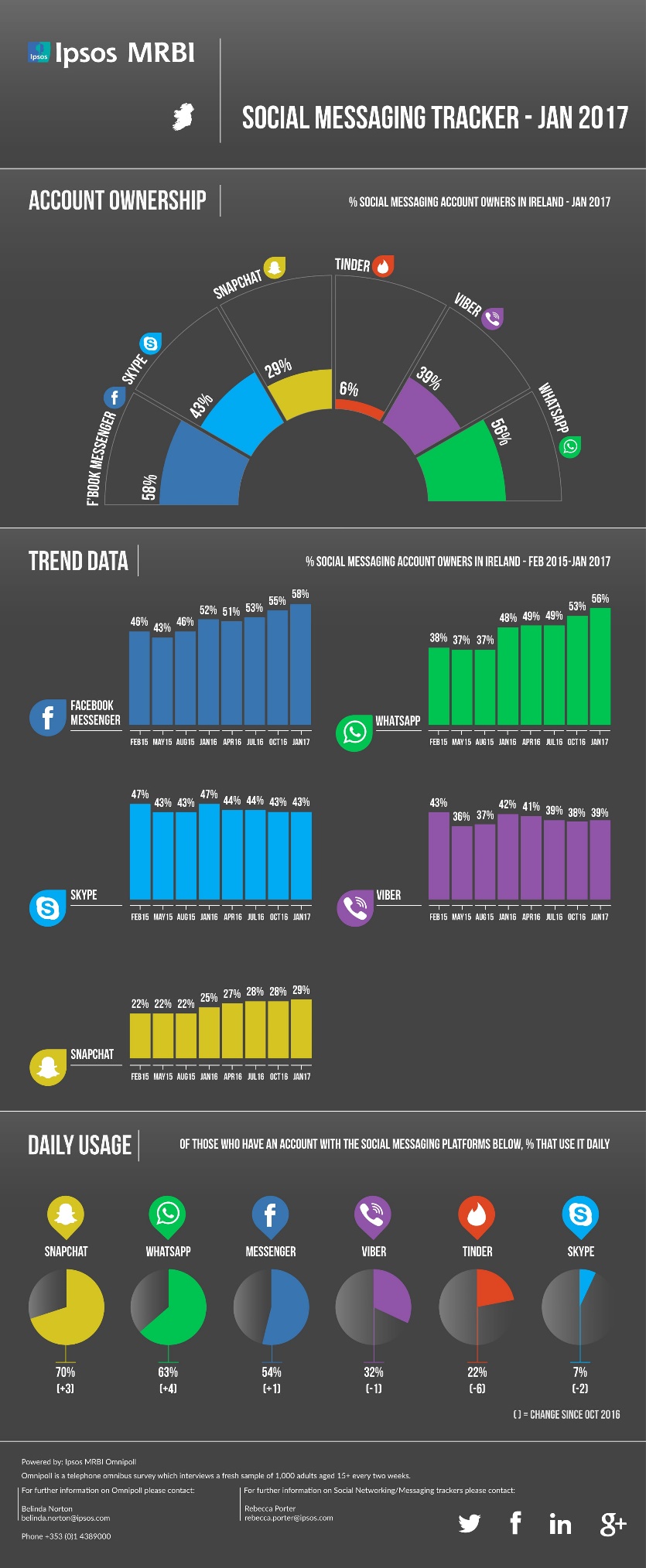


Figure 2 - Social Messaging Tracker (Jan 2017)

In terms of functionality, I envisioned the chatbot responding to and communicating with insurance broker customers who were requesting an insurance quote.

With the findings from the research and having taken into consideration my personal interest, I decided to develop a chatbot that could be deployed for use through Facebook Messenger. The chatbot would enable insurance broker customers to communicate with the bot and receive an instance insurance quote based on the date provided by the customer.

## 1.2 Project Aims

The aim of the project was to create a proof-of-concept chatbot for Applied Systems.

The chatbot would allow Facebook Messenger users to “chat” with the bot and receive an instant motor or home insurance quote based on the information they had entered.

## 1.3 Project Objectives

Project objectives were identified as follows:

* The user can receive an instant motor insurance quick quote[[1]](#footnote-1)
* The user can receive an instant home insurance quick quote1
* Retention of all conversations between the user and the bot
* Retention of all quotes provided to the user
* Track and retain any errors encountered during the workflow
* Allow user to choose a returned quote
* Allow user to choose to be contacted by the insurance broker that provided the quotes
* Follow up conversation with email to insurance broker and the user

## 1.4 Project Activities

To ensure project objectives were met, a list of project activities were identified.

* Determine questions and question order for motor insurance quick quote
* Determine questions and question order for home insurance quick quote
* Design database for storing conversations, quotes and errors
* Establish possible SQL queries for inserting, updating and reading from database
* Research possible frameworks for chatbot implementation
* Research most appropriate programming language to use for development
* Create chatbot solution
* Create logging service
* Create database
* Write and implement database
* Write and implement test plan
* Perform user testing
* Implement changes from user testing findings

## 1.5 Outline of Dissertation Structure

The dissertation has been structured as follows:

* Chapter 1 – Introduction

An overview of the project’s conception, aims, objectives and activities.

* Chapter 2 – Chatbots in Use Today

An evaluation of chatbots currently in operation. An array of chatbots were identified and critiqued; highlighting the advantages and acknowledging potential disadvantages of using a chatbot for a business.

* Chapter 3 – Technical Background

A review of possible technologies that could be used to build the chatbot, with justification for the chosen technology stack.

* Chapter 4 – Development Lifecycle

Elaboration on the Software Development Lifecycle to be used for this project.

* Chapter 5 – Requirements Gathering and Analysis

Explanation of how requirements for the project where determined. The project requirements are detailed alongside a risk analysis of the project.

* Chapter 6 – Design

High-level overview of the system design, the initial database design and the design of the chatbot conversations between the user and the bot.

* Chapter 7 – Implementation

A detailed assessment of the system implementation. Programming practices, libraries and frameworks used and code snippets are detailed and explained in this chapter.

* Chapter 8 – Challenges and Solutions

A review of some of the challenges faced during the project lifecycle with proposed solutions to these challenges noted.

* Chapter 9 – Testing and Results

Testing techniques used during the project along with testing results.

* Chapter 10 – Evaluation

An evaluation of the project management techniques used, system implementation, technology of choice, and personal experience gained throughout the project.

* Chapter 11 – Conclusions

Reflection on the project highlighting its successes and limitations.

* Chapter 12 – Suggested Future Improvements to Project

Suggestions for project improvements and/or enhancements focusing primarily on additional functionality, implementation and estimated implementation times.

# 2 Chatbots in Use Today

## 2.1 What is a chatbot?

Business Insider UK defines a chatbot as a robot that can maintain a conversation with a human. It is, fundamentally, a virtual conversation with a piece of software (Nguyen, 2017).

## 2.2 Review of Chatbots

To gain better understanding of how a chatbot should be designed and how users typically interact with them, a critical evaluation of a selection of chatbots from various business sectors has been performed with the findings detailed below.

The chatbots I selected for review were:

* Marvel
* RoofAi
* Lemonade

### 2.2.1 Marvel

The Marvel chatbot is a way for fans of Marvel Comics to chat directly to some of their favourite characters through Facebook Messenger or Twitter DM (Morse, 2017).

The chatbot was built with Conversable, a platform for building AI-enhanced messaging experiences (Conversable, 2018). Conversable is also known for making use of Natural Language Processing (NLP) and machine learning which bots are more frequently using.

The Marvel bot unfortunately is not a true conversationalist. The conversation has a couple of paths that the conversation can take but always ends in the same way – trying to sell the user merchandise. Which, admittedly, is a great business initiative but could be frustrating for the end user if they are inevitably going to end every conversation with an advertisement “enticing” them to spend money.

### 2.2.2 RoofAi

RoofAi promotes itself as a “smart chat” bot – “a combination of live chat and bots” (RoofAi, 2018). RoofAi view their website widget, a widget built for realtors in the United States, as a tool to manage real-time conversations as and when your website user has a query – stating that “you can now be instantly responsive 24/7, 365” (Moubarak, 2018).

The Roof.ai bot is capable of capturing user information, scheduling viewing appointments and directing leads to the best suited real estate agent; e.g. assigning a user looking to rent a property to an available agent responsible for rental properties (Moubarak, 2018).

The bot is implemented on a customer site in the form of a widget.

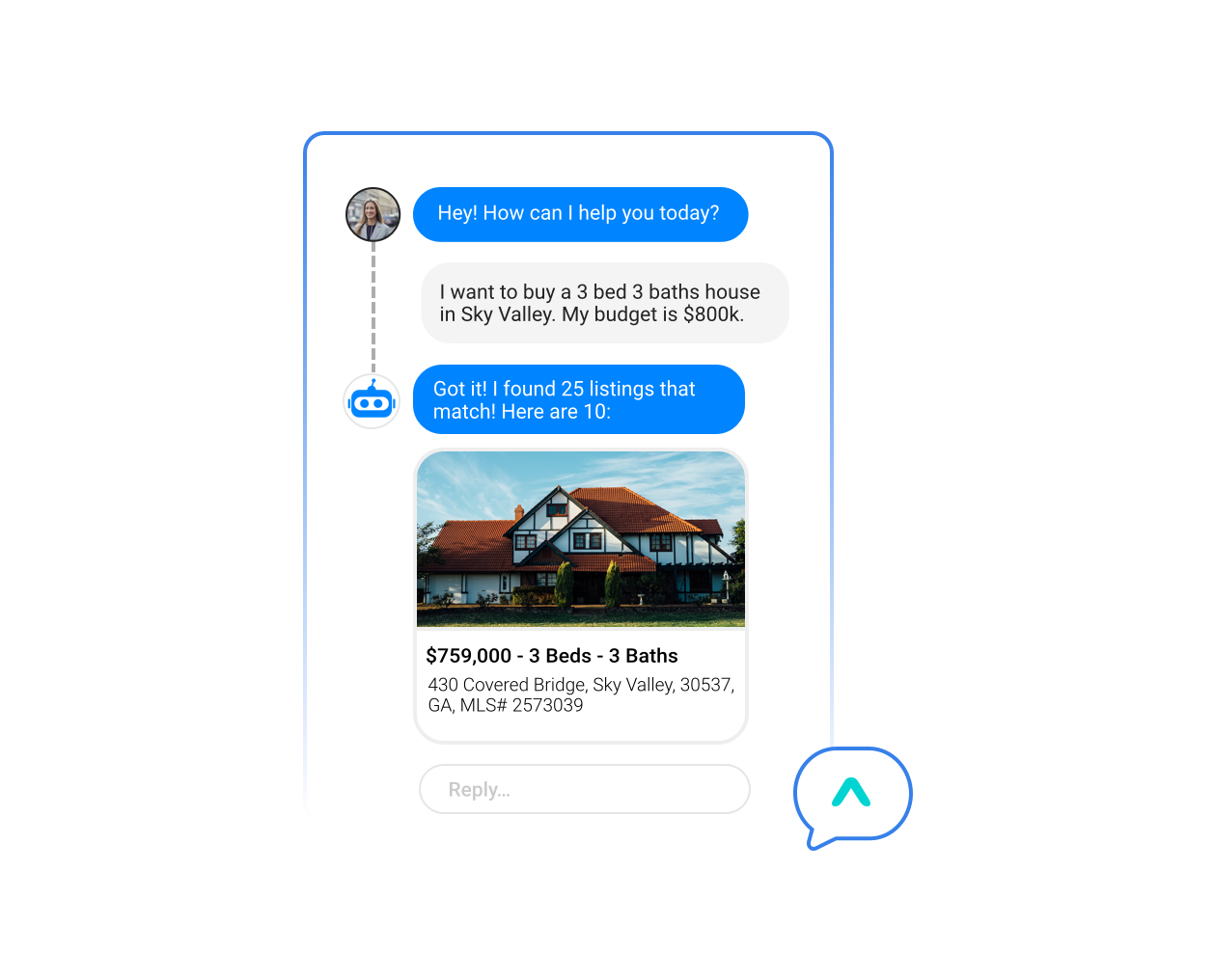


Figure 3 - Roof.ai chatbot example (Moubarak, 2018)

#### 2.2.2.1 Roof.ai Review

Using the website for The Keyes Company, a Florida based real-estate agency, they have added the Roof.ai chatbot widget to their homepage. The widget has been styled to fit with the website and makes use of The Keyes Company logo and branding colours.

It is immediately offering assistance as can be seen below. This is good as it draws attention to the bot in a subtle manner and highlights that the bot is actively available.

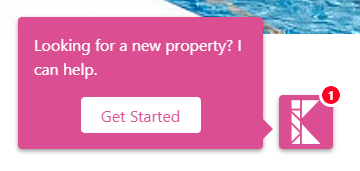


Figure 4 - Roof.ai widget on The Keyes Company website homepage (The Keyes Company, 2018)

Having used the bot to search for properties in the Florida area, the language the bot uses is a positive standout. The language is professional but friendly.

The conversation is guided by the bot asking questions as a human estate agent would. With some questions, the bot gives the user some choices which adds an extra level of guidance and validation to ensure that answers given are correct and match with expected answers.

Questions are asked immediately after an answer has been received so communication is instant and direct.

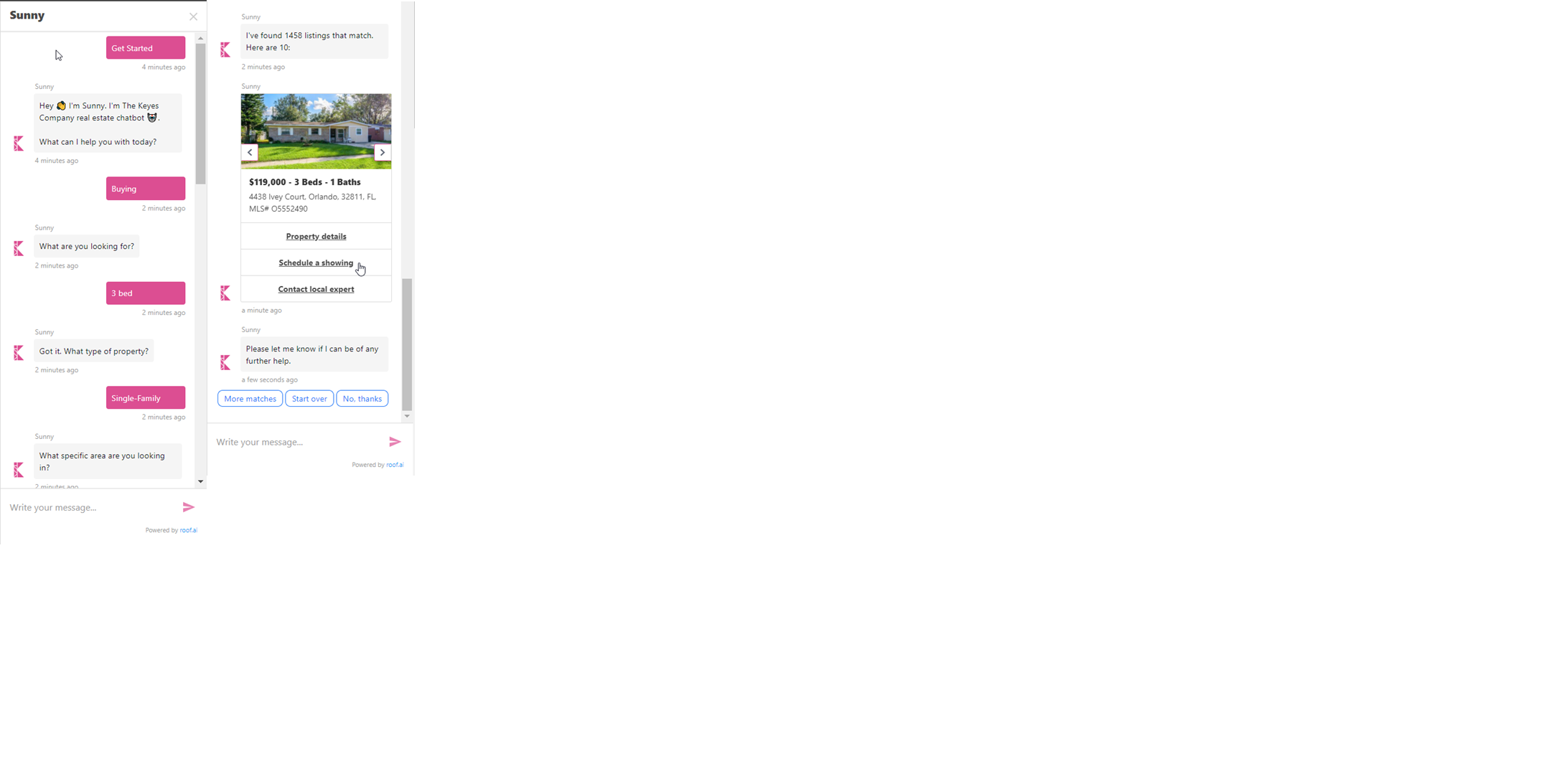


Figure 5 - The Keyes Company example conversation with Roof.ai chatbot

As can be seen above, the chatbot was able to return a selection of properties based on the criteria entered. The properties are nicely displayed in a scrollable carousel making it easy for the user to browse the properties.

A similarity between the Roof.ai chatbot and the chatbot being developed in this project is the use of guided conversation. The benefit of this being that the user should not go too far off the main purpose of the chatbot. The Roof.ai chatbot also does a good job of giving the user options for answering questions whilst mixing in open ended questions so the chat feels more natural; this is also something to be implemented in this project’s chatbot.

### 2.2.3 Lemonade

The chatbot that also currently operates in the insurance industry is used, and was developed by, a company called Lemonade Insurance Agency based in New York.

Lemonade tell their users to “Forget everything you know about insurance” (Lemonade, 2018), as they sell insurance based on a new business model and a central component to this is their artificial intelligence bot (Wissner-Levy, 2016). They aim to make the process of getting insurance faster, more honest and more transparent (Wissner-Levy, 2016).

Lemonade also offer a widget and Application Programming Interface (API) for integration on other websites or applications (Lemonade, 2018).

#### 2.2.3.1 Lemonade Review

Through the Lemonade website, a review of the process to get an insurance price was completed.

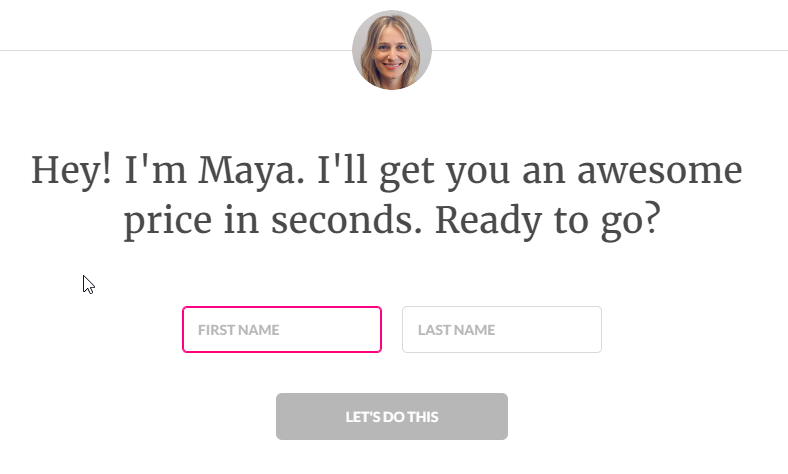


Figure 6 - Lemonade chat example (Lemonade, 2018)

Lemonade have given their bot a name and a profile picture, Maya (Lemonade, 2018). This adds a personal touch to the bot and makes it feel less like chatting to a piece of software and more like chatting to a human being which will have a positive impact on the end user.

Similarly to Roof.ai, the Lemonade chatbot asks questions in a manner that guides the user through the workflow whilst providing options to answer the questions, as below.

The language used by the bot again is natural and polite.

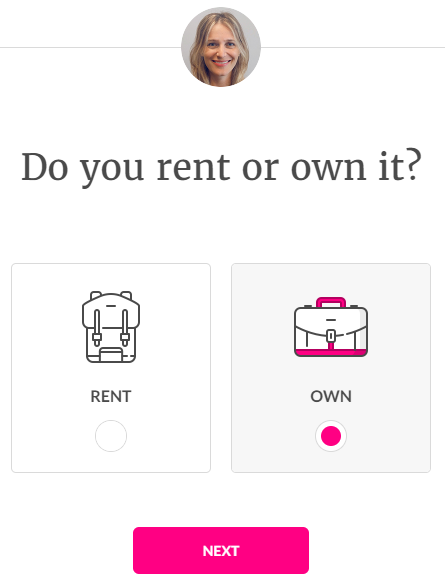


Figure 7 - Example of Lemonade question with options (Lemonade, 2018)

# 3 Technical Background

Before development; research was carried out on how to best build a chatbot and complete the project to meet all the project objectives. The review included; development languages, frameworks or libraries that could be utilised, database options and development environments.

Options for source control have also been included in the research.

## 3.1 Frameworks

As with development of any project, it is possible to use frameworks to aid development. Using frameworks can quicken development. Wikipedia defines a framework as being “a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions” (Wikipedia, 2018).

A framework, in the context of a chatbot, i.e. a bot framework, is a great tool to have as it abstracts a lot of the manual work involved in creating a bot (Maruti Techlabs, 2018).

To develop a chatbot, the following bot frameworks have been considered:

* Microsoft Bot Framework
* Wit.ai
* DialogFlow

### 3.1.1 Microsoft Bot Framework

The Microsoft Bot Framework is comprised of various tools in its software development kit (SDK). The main tools are:

* Bot Connector
* LUIS

#### 3.1.1.1 Bot Connector

Research showed that this service, in the Microsoft Bot Framework, is what enables the bot to communicate messages on channels (the platform the bot is configured to run on, e.g. Facebook Messenger, Slack, Skype, etc.) (Microsoft, 2017).

Communication on these channels is achieved using industry-standard REST and JSON over HTTPS.

#### 3.1.1.2 LUIS.ai

Meaning Language Understanding Intelligent Service – LUIS enables a bot to understand natural language when input by the user (Berry, 2017). It uses machine learning to accept input and extract an intended meaning from the input so it can return a relevant response (Berry, 2017).

The key concepts to LUIS are:

* Intents – these can be considered the “action” a user wishes to perform within an application. Within a LUIS language model, the developer defines intents and maps these to actions (Berry, 2017)
* Utterances – these are the text the user may input that the bot needs to be able to receive and understand (Berry, 2017). There can be many variations to an utterance but the utterance will be linked to a specific intent.
* Entities – these are pieces of information that may appear in an utterance. Identifying entities in an utterance, LUIS is able to choose the best suited action to response to the user (Berry, 2017).

In using LUIS, a developer defines a domain specific language model and fills it with intents, utterances and entities (Berry, 2017). The model then must be trained and published. The LUIS app will then receive an utterance as a HTTP request. From this request it determines the user interaction and responds (Berry, 2017). The user utterance sent from the client application is evaluated to a JSON object by LUIS which is then sent back to the client app (Berry, 2017).

### 3.1.2 Wit.ai

Wit.ai is an open-source API (Application Programming Interface) that “makes it easy for developers to build applications and devices that you can talk or text to” (wit.ai, 2018). It is a natural language platform that uses each interaction to learn so it can provide more accurate responses (wit.ai, 2018).

One appealing aspect of wit.ai is that because it is open-source, it is able to share what it has learned across all developers using wit.ai.

Wit also uses entities and intents to understand the action the user is trying to perform.

### 3.1.3 DialogFlow

On the same premise as Microsoft’s LUIS and Wit.ai; DialogFlow uses machine learning to understand meaning from what a user has input or said.

DialogFlow use an “agent” to manage the conversation between the user (human) and bot (DialogFlow, 2018). DialogFlow describe these agents as Natural Language Understanding (NLU) modules (DialogFlow, 2018). The NLU module converts the user input into data that can determine an action.

Other important aspects of DialogFlow are:

* Entities – domain-specific phrases that can be mapped to NLP (Natural Language Processing) phrases (Maruti Techlabs, 2018)
* Intents – the action to be taken based on what a user has input (Maruti Techlabs, 2018)
* Actions – what will happen based on the identified intent (Maruti Techlabs, 2018)
* Contexts – a string representation to evaluate the user expression. Useful for determining meaning (Maruti Techlabs, 2018)

## 3.2 Development Language Options

Across the frameworks mentioned in the previous section, there are various options for developers to pick from when developing their bot.

The following languages can be used to develop with the aforementioned frameworks:

* C#
* Ruby
* Python
* JavaScript

A brief assessment was made of each language with the findings given below.

### 3.2.1 C#

Closely connected to the .NET Framework, C# is an object-orientated development language that is type-safe (Wagner, Wenzel, & Levin, 2015). It supports object-orientated concepts of encapsulation, inheritance and polymorphism (Wagner, Wenzel, & Levin, 2015).

The .NET Framework is comprised of the virtual execution system, CLR (Common Language Runtime) and a series of class libraries to help aid development.

### 3.2.2 Ruby

An open-source programming language, Ruby gives itself the title of “A Programmer’s Best Friend” (Ruby, 2018). Ruby states that it is a language focused on being simplistic and productive (Ruby, 2018).

Mostly used be developers on Linux, Ruby is by design a simple, complete, extensible and portable programming language (Rouse, 2010).

### 3.2.3 Python

According to the Python homepage, “Python is a programming language that lets you work more quickly and integrate your systems more effectively” (Python, 2018).

Python is another object-orientated programming language. It is a high-level language and is suited to Rapid Application Development (RAD) due to its high-level data structures, dynamic typing and dynamic binding. Also, it promotes modular programs and code reuse.

### 3.2.4 JavaScript

A programming language most associated to building things for the web. It is usually used to build dynamic component for webpages (Mozilla, 2018).

JavaScript is boosted in it abilities by integrating with Application Programing Interfaces (APIs) to help a developer build complex programs.

## 3.3 Software Technologies

### 3.3.1 REST

Acronym for Representational State Transfer, REST is a way for systems to communicate with one another (Codecademy, 2018). It is a web standard that separates the concerns of the client and the server (Codecademy, 2018). This makes it easier to scale and has no concern on what the user interface is doing, providing both the client and server are aware of the format messages must be sent in.

### 3.3.2 JSON

JSON, JavaScript Object Notation, “is a way to store information in an organised, easy-to-access manner” (Lengstorf, 2018). JSON makes it easy to get data quickly and cross platform (Lengstorf, 2018).

### 3.3.3 HTTPS

This is a secure, encrypted version of the HTTP protocol. The HTTP (Hyper Text Transfer Protocol) is the protocol determining how data should be communicated between the web browser and a website (Comodo, 2018). HTTPS adds “Secure” to this protocol, meaning all communication is encrypted.

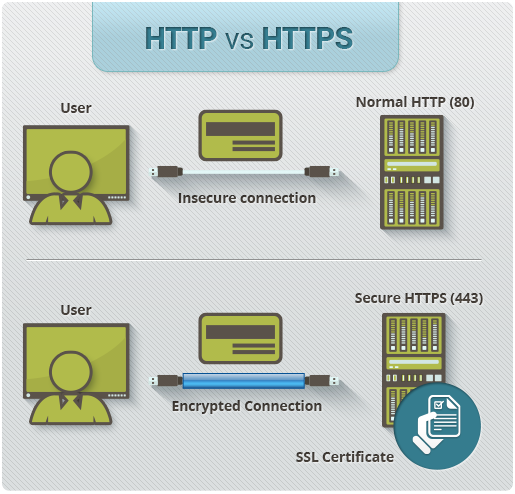


Figure 8 - HTTP vs HTTPS (Comodo, 2018)

## 3.4 Databases

The database used for this project is Microsoft SQL Server. It is a RDBMS (Relational Database Management System) (Rouse, Microsoft SQL Server, 2017). A relational database is data organised into tables with well-defined relationships between these tables (Techopedia, 2018).

To interact with these databases, the best practice is to use Structured Query Language (SQL). This allows CRUD (Create, Read, Update and Delete) interaction with the database.

## 3.5 Source Control

Amazon Web Services defines source control as “the practice of tracking and managing change to code” (Amazon Web Services, 2018). Using a source control system gives developers a method of tracking the development history of a project. It is also a useful tool when conflicts in code arise when two versions of a code base are merged together.

Source control allows a developer to create branches that isolate their development code before it is combined with the main code base.

This project will use Git for source control. Git is a distributed version control system (Ngan, 2018). Main functionality of Git includes branches and commits. Branches allow development code to be kept separate from the main code base and commits are how developers save their code changes (Amazon Web Services, 2018).

# 4 Development Life Cycle

The Software Development Life Cycle is a process that aims to produce software with the highest quality and with the lowest cost possible in the shortest amount of time (Stackify, 2017).

According to (Stackify, 2017), the advantages of following the SDLC are:

* It allows a high level of management control
* Gives developers a good understanding of what they are trying to build
* An agreement is made upfront on what the project outcome should be
* It sets out an agreed plan on how to reach the proposed goal

The Software Development Life Cycle defines six stages that can be implemented in various ways by different SDLC models.

These 6 stages are:

* Planning
* Defining
* Designing
* Building
* Testing
* Deployment

The models, also called Software Development Process Models, will follow steps unique to each model but will still all either strictly or loosely follow the Software Development Life Cycle stages (Tutorials Point, 2017).

## 4.1 Software Development Process Models Considered

The following models were considered as a development model to use on this project.

### 4.1.1 Waterfall Model

The first process model to be introduced to the software development industry, it was designed to be used in a way that processes do not overlap; one process must finish before the other starts (Tutorials Point, 2017).



Figure 9 - Software Development Life Cycle (Gordiyenko, 2014)

#### 4.1.1.1 Advantages of Waterfall Model

* Simple to use and understand
* Each process has specific goals and outcomes
* Stages of the project are well defined

#### 4.1.1.2 Disadvantages of Waterfall Model

* A working product is not delivered until near the end of the life cycle
* Not suitable for projects with requirements that are likely to change
* Stages must wait on their predecessors to finish before they can start

#### 4.1.1.3 Why the Waterfall model was not chosen

Due to its strict phase completion rules, the Waterfall model was not suited for this project as it is open to changing requirements throughout the duration of the project.

The plan is also to have regular feedback from stakeholders and users which again, is not conducive with the Waterfall model process.

### 4.1.2 Spiral Model

Consisting of four phases; Planning, Risk, Engineering and Evaluation, a software project using this model will pass through each phase iteratively until the project is delivered (International Software Testing Qualifications Board, 2017).

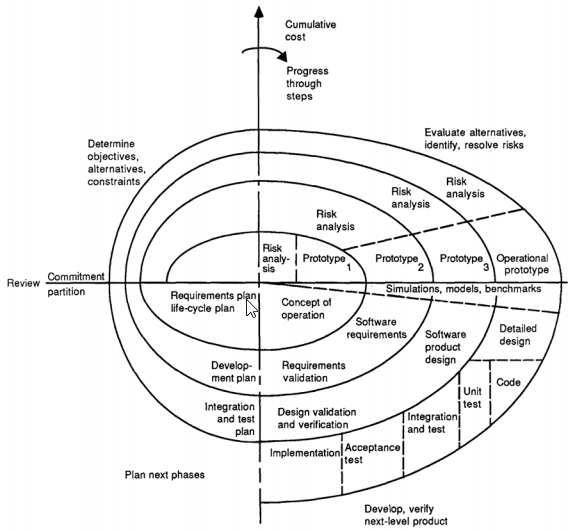


Figure 10 - Diagram of Spiral Model (Boehm, 1988)

#### 4.1.2.1 Advantages of Spiral Model

* Emphasis on risk analysis means risks are identified early and can be managed or avoided
* Software is produced early and frequently
* Software functionality can change or be added late in to the project

#### 4.1.2.2 Disadvantages of Spiral Model

* Can be an expensive model
* Not suited to small projects
* Risk analysis requires experienced analysts

#### 4.1.2.3 Why the Spiral model was not chosen

The Spiral model was not selected for use on this project due to the short life span of the project. Spiral is more suited to larger projects with a long-term commitment.

The project was also considered low risk and the requirements of the project are clear and concise.

### 4.1.3 Agile Model

The Agile SDLC model put focus on delivering products to the customer on a regular basis. It combines iterative and incremental models and focuses on customer satisfaction and welcomes change (Tutorials Point, 2017).

A project using the Agile model will break a project down in to smaller pieces of work and deliver these in an iterative manner.

Each iteration is able to have multiple processes active at any one time. The processes include:

* Planning
* Requirements Analysis
* Design
* Coding
* Unit testing
* Acceptance testing

The iterations that produce working software are usually time boxed to an amount of time decided by the team.

Iterations can happen in a similar way to the image below.



Figure 11 - Representation of the Agile Development Model (ISTQB Exam Certification, 2018)

The Agile model comes with an Agile Manifesto (Agile Manifesto, 2017) that states the following principles:

* Individuals and interaction over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiations
* Responding to change over following a plan

#### 4.1.3.1 Advantages of Agile

* A realistic take on how software is best developed
* Functionality is developed quickly
* Processes work with pre-defined or changing requirements
* Little or no planning required

#### 4.1.3.2 Disadvantages of Agile

* Depends heavily on stakeholder and customer communication and feedback
* Less focus on documentation can lead to problems when on boarding new team members or handing a project on to another team
* Changing requirements and functionality can have an adverse effect on the project delivery deadline and can lead to scope creep on the project

#### 4.1.3.3 Why Agile was chosen for this project

Agile was chosen as the software development model for the project as the Agile model is one widely practised within Applied Systems.

The opportunity to develop software quickly and get regular feedback on it means the end product is more likely to meet the Project Sponsor’s expectations.

Not having to focus on detailed documentation also means the emphasis can be placed on building a working system. Due to the tight schedule of this project, this is a benefit that cannot be overlooked.

# 5 Requirements Gathering & Risk Analysis

Before development on a project begins, requirements gathering is an essential process to giving the project the best chance of success.

Requirements gathering, also referred to as requirements elicitation, is the process of producing a list of requirements based on what the project stakeholders want and need from the system (Inflectra, 2018).

Requirements come in the form of functional and non-functional.

Functional requirements can be described as requirements defining what the system to be built should do. Non-functional requirements are a set of requirements that determine how the system works.

## 5.1 Requirements Gathering Techniques

During the requirements gathering process there were numerous techniques that could have been used. For this project; the project stakeholders where consulted during a brainstorming session on how the chatbot should operate.

Details from the brainstorming session are attached in Appendix A.

This project has taken a slightly different and less formal approach to requirements gathering. The reason for this is that the chatbot being developed for Applied Systems is based on the current quick quote solution already live on Applied System’s customer websites. It means that a lot of the requirements for the quick quote solution can be transferred to be applied to the chatbot.

## 5.2 Requirements Specification

After the requirements brainstorming session with the project stakeholders; a formal requirements document was produced.

This document was presented to all stakeholders for review. Any changes to the requirements were submitted by the stakeholders, the requirements document updated and sent back to the stakeholders for sign-off.

Once signed-off, should any changes need to be made to the requirements, a formal change request had to be submitted and evaluated on how it would impact the project and the project delivery date. A decision would then be made on whether the changed requirement could be accepted or not.

**TODO – get example of change request document**

### 5.2.1 Functional Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Requirement Description** |
| FR01 | User should be able to get a motor quote |
| FR02 | User should be able to get a home quote |
| FR03 | The bot should be able to look-up and display a user’s vehicle details based on the user registration |
| FR04 | The user should be able to request contact from the broker after receiving a quote |
| FR05 | User should receive an email with information about their quote after the quote is received |
| FR06 | The broker should receive an email with information about the users quote after a quote has been given to the user |
| FR07 | User should not be able to get a motor quote if under the age of 17 |
| FR08 | User should be able to change their answers during the workflow |

### 5.2.2 Non-functional Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Requirement Description** |
| NFR01 | The language of the bot should be professional and friendly |
| NFR02 | The language of the bot should be easily understood |
| NFR03 | The responses of the bot should be visually appealing to the end user |

## 5.3 Requirements Validation

Requirements validation is the process of ensuring that any requirements specified as necessary to complete the project objectives and achieve the overall aim are questioned and clarified. Not validating requirements early in a project could lead to scope creep, unexpected costs or cause a project to miss its scheduled delivery date. Ultimately, the customer may not end up with the solution they wanted.

A valid requirement should have the following four characteristics:

* Correct – a requirement should accurately describe the required functionality
* Complete – a requirement should give enough information to a developer in order for the developer to implement the required functionality
* Technically achievable – a requirement should not be too ambitious so that developers cannot implement it
* Clear – a requirement should only have one possible outcome

During the requirement gathering phase of this project, all requirements were carefully considered and agreed upon in relation to the four characteristics noted above.

## 5.4 Risk Analysis

The purpose of a good risk analysis before a project begins is to help expose potential risks in a project at an early stage. It is important to identify the likelihood of the risk occurring and to hopefully find an early solution to remove or limit the risk. It is a key tool in project planning.

A qualitative risk analysis was carried out to identify any potential negative implications and collated to the risk register below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk Id** | **Description** | **Probability of Occurrence** | **Preventive Measures** | **Severity** |
| 1 | Unrealistic time schedule | Medium | Incremental development, modify milestones, requirement prioritisation | High |
| 2 | Data loss | Low | Backup database regularly, secure repository for source code | High |
| 3 | User interface does not meet requirements | Low | Regular customer interaction, careful design, attention to requirements | Medium |
| 4 | Poor product performance | Medium | Ensure data is structured correctly, performant UI components and database | Medium |
| 5 | Scope creep | Medium | Set functionality boundaries, closely manage changing requirements | Medium |
| 6 | Lack of skills | Medium | Build software with skills I have, iterative builds to get more complex if time allows | Medium |

Figure 12 - Risk Register

## 5.5 Data Protection

**TODO – research data protection stuff**

# 6 Project Plan

To give the project the best chance of success, a Project Plan was developed to set out defined expectations and project deliverables.

The Project Plan also consisted of a list of project milestones and a Work Breakdown Structure.

## 6.1 Project Scope

The delivered system would consist of a chatbot that can run on Facebook Messenger.

It should be able to interact with a SQL database and a service that will log errors.

A user of the chatbot should be able to answer questions and receive a motor or home insurance quote based on the insurance product they have selected.

The chatbot will have to retrieve the quotes from rating services owned by Applied Systems. On retrieving these quotes the chatbot will display the quotes to the user. The user and the insurance broker should both receive an email detailing the information of the quote.

The project needs a service for recording errors to the database. These errors should be detailed enough to help the developer understand why the error happened so they can take action to prevent it from happening again.

## 6.2 Project Schedule

The project was scheduled to run for around 7 months, which breaks down in to estimated periods of 1 month of planning and design, 5 months of development and another month for testing and deployment.

### 6.2.1 Project Milestones

The project milestones were added to the GitHub quoting-bot, available Appendices A-D. The milestones selected outlined important stages throughout the project and determine points at which significant amounts of work had been completed.

### 6.2.2 Work Breakdown Structure

As part of project planning, a Work Breakdown Structure, Appendix E, had been developed to give an oversight to which project milestones fell into each stage of the Software Development Life Cycle (SDLC).

## 6.3 Project Resources

To ensure delivery of this project, the following required resources were identified:

* Development computer (PC and/or laptop)
* Microsoft Visual Studio
* Microsoft Bot Emulator
* SQL server

# 7 Design

When designing a chatbot the most important aspect to consider is the user experience of the bot. As bots can be built for specific platforms, e.g. Slack, Skype, Facebook Messenger, etc., the user interface of the bot will be determined by the platform. Instead the focus of design should be on the user experience.

Contributors at Microsoft (Velloso, Iqbal, & Standefer, 2017), suggest the following as key considerations to include in the bot design.

* Is a user problem solved by the bot in as few steps as possible?
* Is the user experience to solve the user problem better/faster/easier than a web or app alternative?
* Is the bot available on the platforms demanded by the user?
* Is using the bot intuitive?

A bot should be designed minimally but to have maximum impact (Sens, 2017). A bot should interact with a user through engaging conversation, all the while trying to accomplish the following two goals quickly and efficiently (Sens, 2017).

1. Engage and Collect – communicate with the user in a direct manner to collect the information that will identify their needs and wants.
2. Parse and Deliver – the information collected should then be immediately parsed so it can be used to deliver content back to the user to fulfil their needs.

A bot should be designed with a leading question to determine the intent of the user. This should then be followed by as many follow-on questions as is required to gather enough information to solve the user’s issue as quickly as possible.

## 7.1 Conversation Flows

Designing how the conversation between bot and user will flow is critically important to the user experience of the chatbot. Knowing why a chatbot is being built should be the starting point when designing the conversational flow (Maruti Techlabs, 2018).

The conversation will flow between structured and non-structured interactions (Maruti Techlabs, 2018). Structured interactions will give the user options and choices when replying to a question, whereas non-structured interactions are more open to free text answers or queries and the bot will be required to handle these responses or requests (Maruti Techlabs, 2018).

**TODO – design conversation flow**

## 7.2 Conversation Dialogs

The first interaction between the bot and the user is very important to the user experience (Velloso & Standefer, Design a bot's first user interaction, 2018). It is not recommended to open the conversation with an open question. The question should be asked with giving the user a choice of options as their answer.

As a conventional website or app uses screens for users to interact with, a bot uses dialogs. Dialogs give the developer the ability to separate out areas of functionality and give the conversation a flow. Dialogs will contain actions to redirect to other dialogs or for processing user input.

### 7.2.1 First Interaction

For this project, the first interaction from the bot is a welcoming message followed by a list of options showing the insurance products available to get a quote for.

### 7.2.2 Motor Dialog

Aim of the dialog is to obtain enough information from the user that the bot can then make a request for motor quotes and provide them to the user.

**TODO – add dialog flow**

### 7.2.3 Home Dialog

**TODO – add dialog flow**

## 7.3 User Interface

As the chatbot will be built for publishing on Facebook Messenger, the user interface will be determined by the styling of the Facebook Messenger application.

Leveraging on the capabilities of Facebook Messenger, the chatbot will use emoji to convey a friendlier persona to the end user. Emoji’s are displayed on the user interface by using Unicode values.

## 7.4 High-level System Architecture

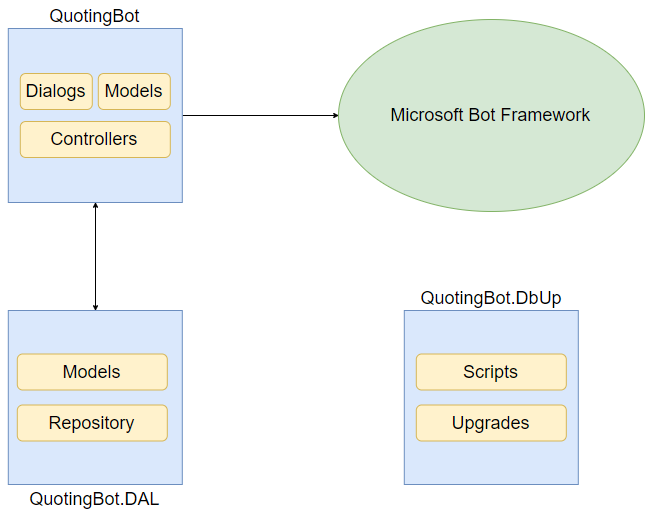


Figure 13 - High level system architecture

## 7.5 Class Diagram

**TODO – generate when program completed**

## 7.6 Database Design

# 8 Implementation

A crucial stage of any software project, the implementation phase is were all previous stages of the Software Development Life Cycle combine and the developers start to build the software solution.

## 8.1 Development Environment

This solution was developed in the Microsoft Visual Studio 2017 Integrated Development Environment (IDE) with a Microsoft SQL Server database used for data storage.

The software was developed in C# and used the Microsoft Bot Framework to aid development of the chatbot. The Bot Framework Emulator was used to simulate how the bot would work on Facebook Messenger.

## 8.2 Asynchronous Programing

Developing a chatbot using the Bot Framework means there was a heavy involvement of asynchronous programming. Asynchronous programming is parallel programming where a task will run on a separate thread to the main application thread. The task with notify the calling thread when it completes, fails or progresses (Vogel, 2011).

The main benefits of asynchronous programming are that an application will typically be have improved performance and a responsive User Interface (UI) (Vogel, 2011).

## 8.3 Solution Structure

The structure of this solution consisted of 4 projects further split in to folders and classes.

Four projects were added so there was a clear separation of concerns. Focused logic could be retained in one solution while more generic code could be separated to another solution. This was done to follow one of the principles of clean code.

## 8.4 Technical Points of Interest

### 8.4.1 Application Entry Point

The entry point for the bot is called when the user first sends a message to the bot. The bot receives the message to its ‘Post’ endpoint in the MessagesController.cs (Figure 14) and determines if the incoming ‘Activity’ – a class with the Bot Framework – is of type ‘Message’. If the incoming Activity is if type Message, the bot then starts up a Conversation, directing the user to a specified dialog. In this solution, the code is directed to the RootDialog.cs where it is picked up in the RootDialog StartAsync method.

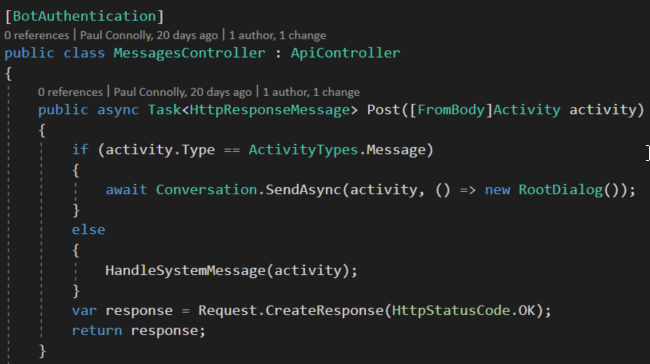


Figure 14 - Post endpoint in MessagesController.cs

As can be seen in Figure 14, if the incoming Activity is not of type of Message it will be handled appropriately in the HandleSystemMessage method (Figure 15). Nothing was implemented for the other message types as it was not required for this project.

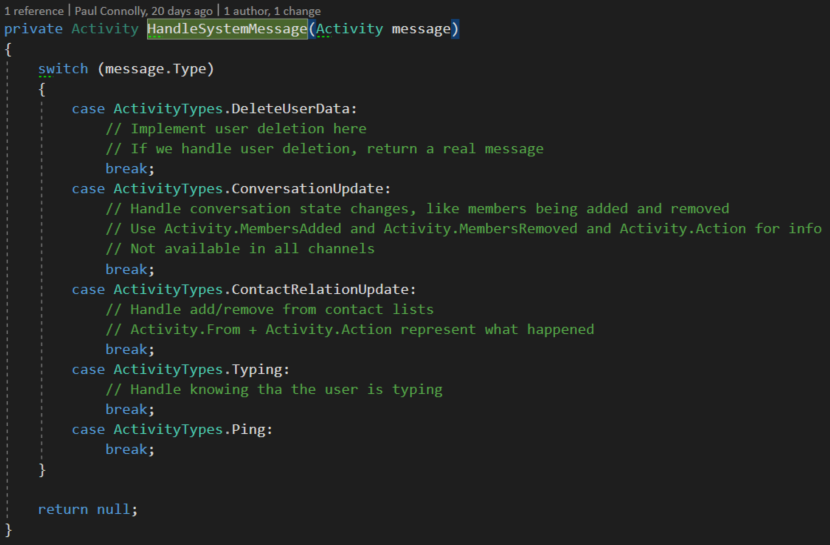


Figure 15 - HandleSystemMessage method from MessagesController.cs

### 8.4.2 RootDialog.cs

As stated before; when the RootDialog class is instantiated by the MessagesController class, it goes to the StartAsync method (Figure 16). This method takes the current context of the conversation and posts a welcoming message to the conversation for the user to see using ‘context.PostAsync()’.

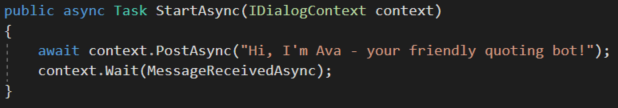


Figure 16 - StartAsync method from RootDialog.cs

The next line; ‘context.Wait(MessageReceivedAsync)’ call the MessageReceivedAsync method which calls ShowQuoteOptions (Figure 17) to create a new message for the conversation to give the user a choice on which product to start getting a quote for.

This is done by creating a new PromptDialog of type Choice from the Bot Framework. A PromptDialog needs to be given the context of the conversation, a process to carry on with when an option is selected, a list of options, a message to be displayed on screen asking the user to make a selection and text to tell the user to retry if they give an invalid answer.

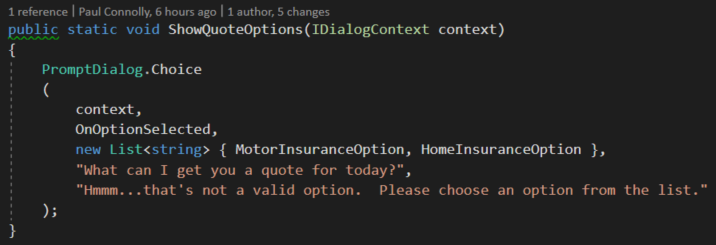


Figure 17 - ShowQuoteOptions from RootDialog.cs

The method given to the PromptDialog to continue to after an option was selected is OnOptionSelected (Figure 18). This method determines which option was selected from the options given using an “if” statement. Depending on the option selected the conversation is instructed to call the MotorDialog, HomeDialog or display an error message.

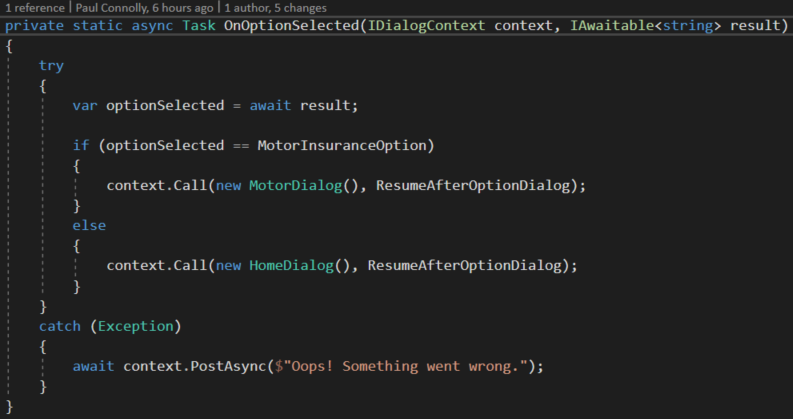


Figure 18 - OnOptionSelected method from RootDialog.cs

The possible options passed to the PromptDialog.Choice (Figure 17), have been declared as class variables in the RootDialog.cs (Figure 19).



Figure 19 - Class variables in RootDialog.cs

These variables were declared and assigned at this level so they could be reused throughout the class.

### 8.4.3 MotorDialog.cs

Should the user select ‘Motor Insurance’ from the PromptDialog.Choice the conversation context will be passed to the MotorDialog class where it enters the StartAsync method. A couple of messages are posted to the conversation by the bot before it creates and calls a new FormDialog (Figure 20).

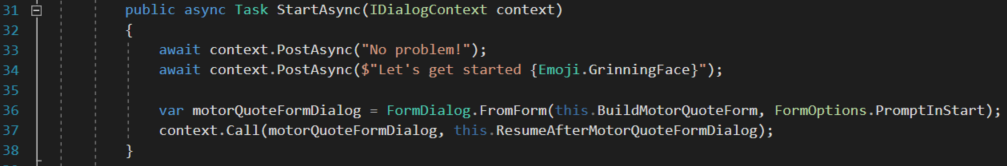


Figure 20 - StartAsync method from MotorDialog.cs

The form dialog is created on Line 36 in Figure 20 by calling the BuildMotorQuoteForm method and also instructing the bot on when the dialog should be presented to the user.

On Line 37 of Figure 20 the ‘context.Call’ first parameter tells the bot where the dialog the conversation should be switched to and in its second parameter it is passed the dialog the bot should continue to on completing the first dialog – motorQuoteFormDialog in this instance.

#### 8.4.3.1 BuildMotorQuoteForm()

Returning a built form to use in a dialog, the BuildMotorQuoteForm method is where the bot is instructed on the order in which questions should be asked.

The method instantiates the FormBuilder from the Bot Framework and is passed a class of MotorQuote which contains a number of properties that will be determined by the Bot Framework as questions to be asked.

The FormBuilder interface comes with various methods which can be used to build the form, as seen in Figure 21.

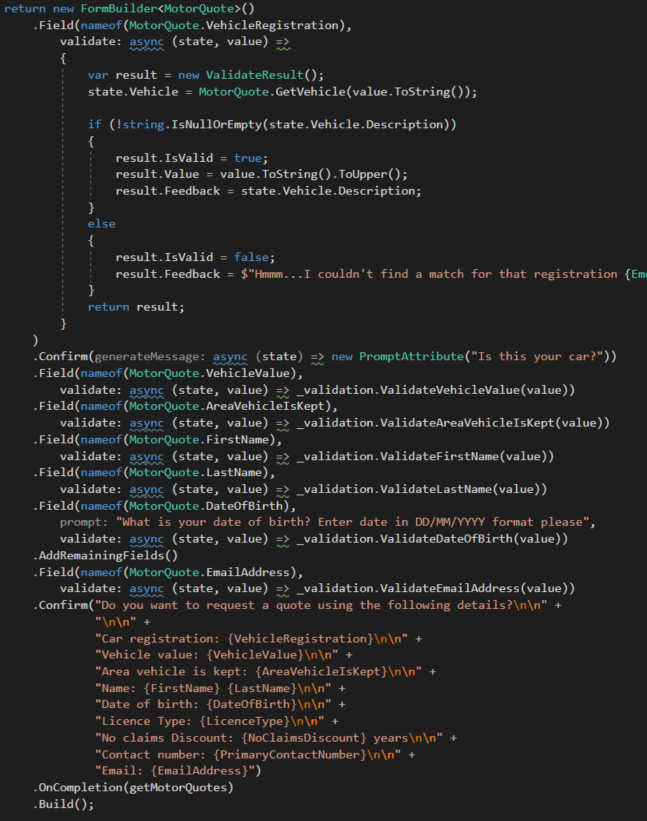


Figure 21 - FormBuilder example from MotorDialog.cs

Calling the ‘Field’ method, the developer assigned it a name which came from the model MotorQuote.cs. This field can then be given custom validation to ensure the entered value is correct.

In Figure 22, the developer performs logic to determine if the entered VehicleRegistration is valid and capable of matching a vehicle to it using the GetVehicle() from the MotorQuote class. If the lookup returns a vehicle it means the registration was valid and the bot is told the result is valid and it will move on to the next question. If no vehicle is returned, it means the car registration was invalid and an error message is returned to the user asking them to try another registration.

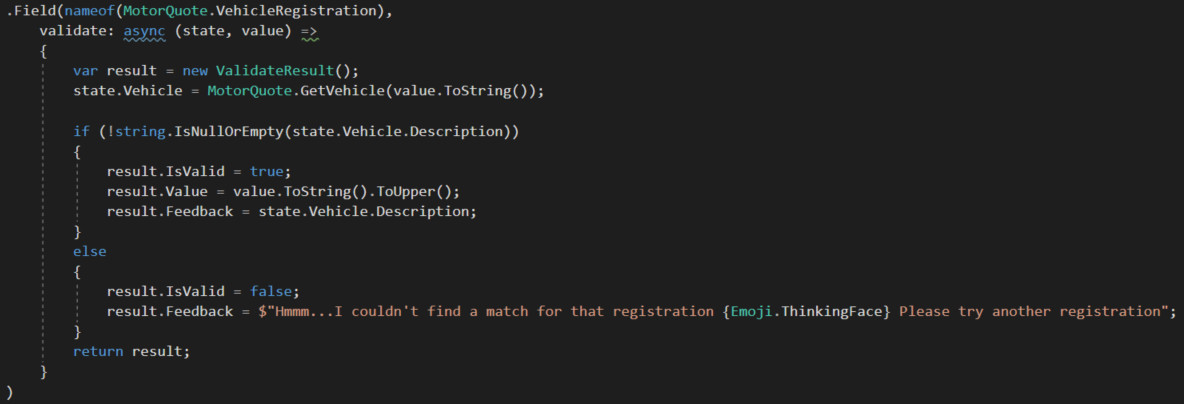


Figure 22 - Field example from MotorDialog.cs

Validation is carried out on other fields but this will be looked at in more detail later in the chapter.

#### 8.4.3.2 ResumeAfterMotorQuoteFormDialog()

Once all questions have been answered and validated, the bot progresses to the ResumeAfterMotorQuoteFormDialog() to fetch the insurance quotes for the user. The code to make the calls for returning the insurance quotes are shown below, Figure 23.

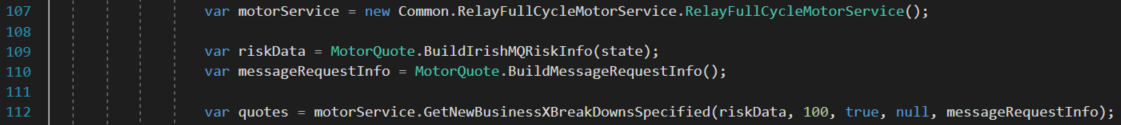


Figure 23 - Code for requesting quotes from Motor web service

Line 109 uses answers to the questions asked in the form to build up the risk information to be sent to the web service request on line 112. This web service was added as a reference to the project so quotes could be retrieved.

Checks are then performed to make sure quotes were returned and the quotes are then stored to the database, returned by the bot to the user, emails are sent to the user and the broker and the conversation is also stored in the database.

When that completes, the dialog passes conversation control back to the RootDialog.cs using ‘context.Done(state)’ (Figure 24).

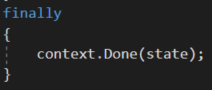


Figure 24 - Completing the dialog

#### 8.4.3.3 GetQuoteReceipts()

This method is interesting because it uses LINQ to perform a function on each element inside of array (Figure 25).

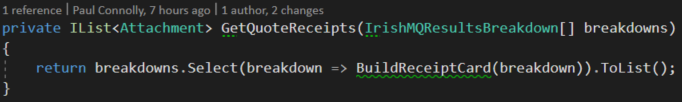


Figure 25 - GetQuoteReceipts() with LINQ example

.NET Language-Integrated Query (LINQ), gives developers the ability to query any IEnumerable information source (Box & Hejlsberg, 2007).

### 8.4.4 HomeDialog.cs

The HomeDialog.cs is implemented in a very similar way to the MotorDialog.cs. It has the required StartAsync method, a BuildHomeQuoteForm() and a ResumeAfterHomeQuoteFormDialog() as can be seen in Figure 26.

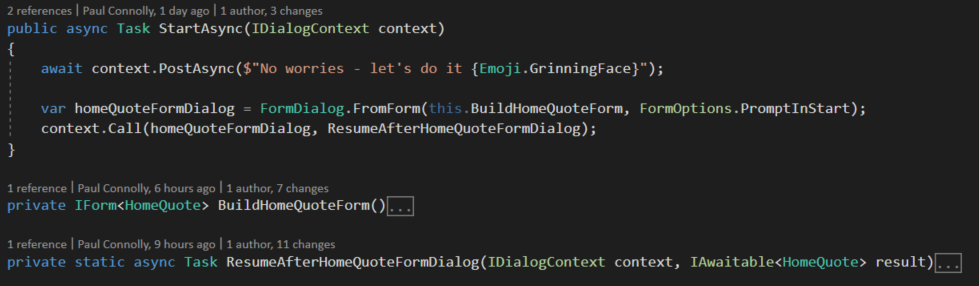


Figure 26 - Section of HomeDialog.cs

The differences are in the model used and the web service called for quotes (Figure 27).

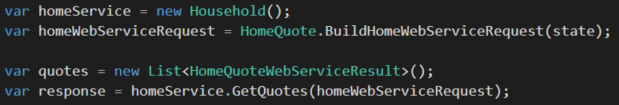


Figure 27 - Code for calling Home web service

### 8.4.5 Models

The models in the solution contain properties and methods to be used on the respective dialogs.

The properties in a model are defined with the data types the bot expects the user to provide answers in. This acts as the first form of validation because if the user enters a string value to answer a question with a property of type integer, the bot will reject the answer and tell the user that the answer was invalid (Figure 28).

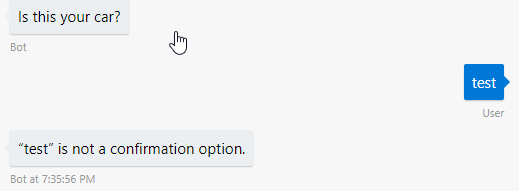


Figure 28 - Example of bot validation when wrong answer is given where Yes/no is expected

Within the models, there are also methods for defaulting items of proposer risk that are not asked as questions in the dialog. These questions were not needed to be asked because this bot is only for returning quick quotes to the user.

An example of this defaulting in the HomeQuote.cs model can be seen in Figure 29.

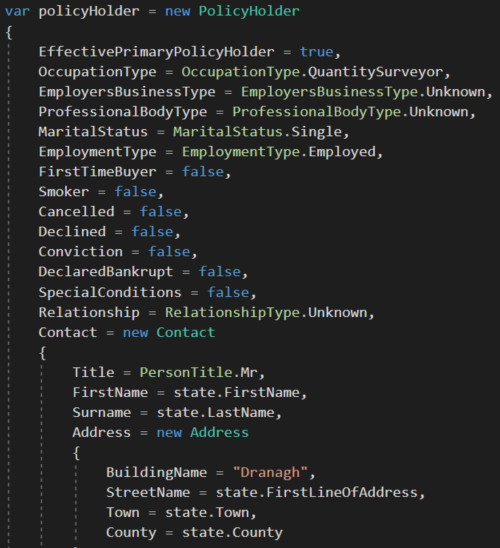


Figure 29 - Risk defaulting snippet

### 8.4.6 EmailHandler.cs

The EmailHandler.cs is responsible for building and sending emails to the user and the broker.

This class and its methods are used in both the MotorDialog.cs (Figure 31) and the HomeDialog.cs (Figure 30).

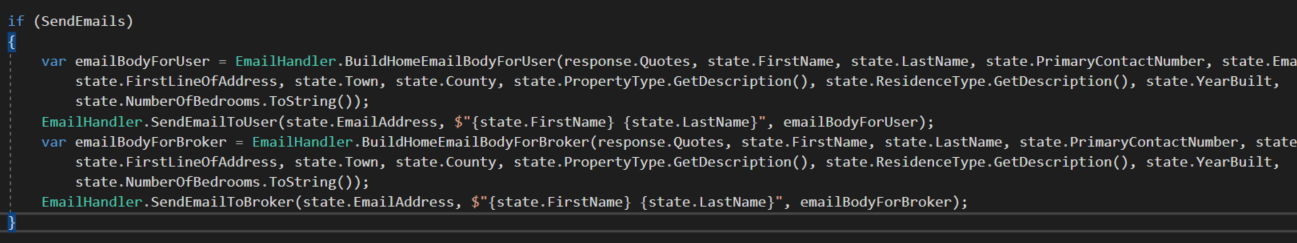


Figure 30 - EmailHandler usage in HomeDialog.cs

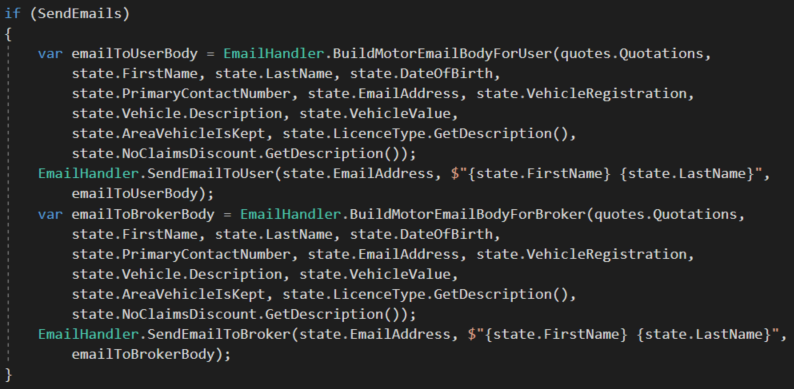


Figure 31 - EmailHandler in MotorDialog.cs

An important method in this class is the SetupStmpClient() (Figure 32). This method creates a new instance of a SmtpClient and assigns the settings and user credentials for the client. These setting come from settings in the web.config file.

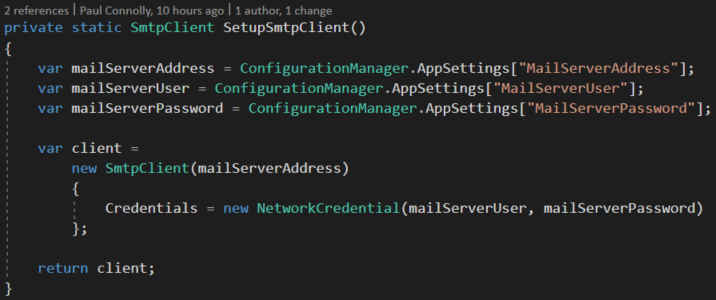


Figure 32 - SmtpClient() from EmailHandler.cs

In the SendEmailToUser() (Figure 33) and SendEmailToBroker (Figure 34), more setup to the mail message is done from app settings and parameter values it has been passed.

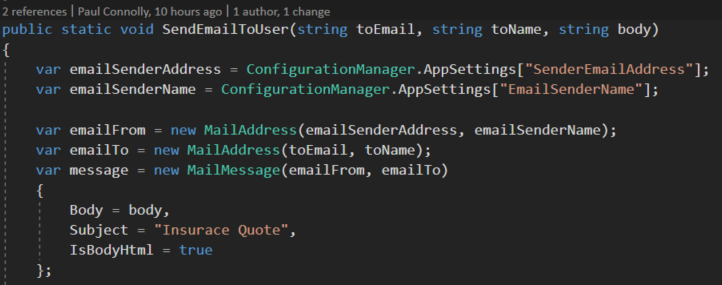


Figure 33 - SendEmailToUser()

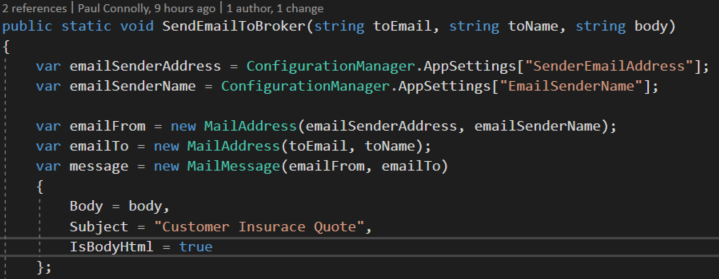


Figure 34 - SendEmailToBroker()

An important setting to note in Figures 33 and 34 is IsBodyHtml = true. This allows the developer to include Hyper Text Markup Language (HTML) in the email body and have it render in the email.

The body of the email is built in other methods in this class that are called in the Motor and Home dialogs. There are four methods for building the email body as the information in each email is different depending on the product and whether the email is going to the user or the broker.

### 8.4.7 Enums

They have been used in this solution for different reasons. The first reason is so they can be used in the models to tell the Bot Framework FormBuilder that this question must be displayed to the user with the question and a list of possible answers (Figure 35).

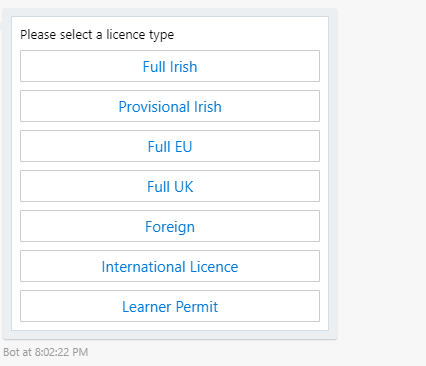


Figure 35 - Example of question being asked based on enum data type

The second way that enums have been used is to enable the reuse of certain strings across the system. This has been done to the Emoji.cs by giving each enum a “Description” tag (Figure 36) which is assigned the string value the developer wants to use for displaying the emoji.

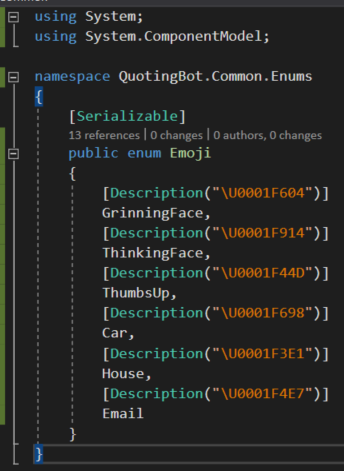


Figure 36 - Emoji.cs

#### 8.4.7.1 EnumExtension.cs

The EnumExtension.cs (Figure 37) is what makes it possible to add the Description to enums. This extension method can then be called anywhere the enum is used and the developer wants to get the description tag value rather than the enum value.

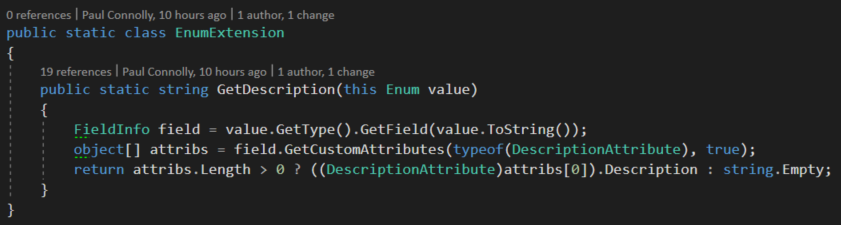


Figure 37 - EnumExtension.cs

#### 8.4.7.2 EnumConverter.cs

The class for converting enums is important because it enables the correct information to be passed to the web service requests. Taking the ConvertLicenceType() (Figure 38) as an example, it takes the selected enum value, passes it through a switch statement and returns the string value expected by the service. These methods are used in the models as seen in Figure 39.

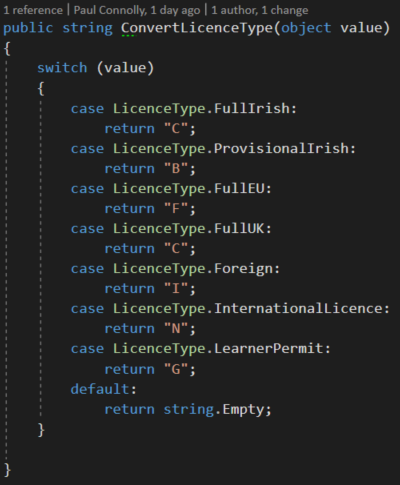


Figure 38 - ConvertLicenceType()

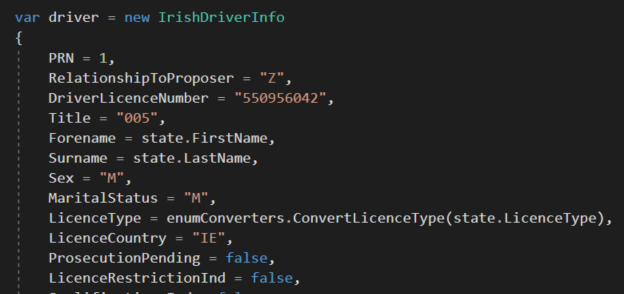


Figure 39 - Use of the ConvertLicenceType()

### 8.4.8 Helpers

The helper classes are designed to be generic so they can be used anywhere across the system.

#### 8.4.8.1 Formatter.cs

The formatter class has one method that takes a string and ensures that the first character of the string is capitalised before the string value is returned (Figure 40).

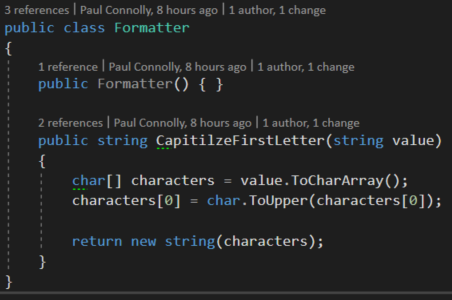


Figure 40 - CapitilizeFirstLetter() in Formatter.cs

#### 8.4.8.2 Validation

The validation methods are used when the user has provided an answer. The validation makes sure the answer is in a suitable format or that the answer is an acceptable answer.

Two methods in the validation class use Regex to check if the provided answer is valid. The Regex used in IsYearBuiltValid() is “^[0-9]{4}$”. This ensures that the provided answer is a four digit value only containing numbers between 0-9. The Regex in IsEmailAddressValid() is “\b[A-Z0-9.\_%+-][+@[A-Z0-9.-]+\.[A-Z]{2,}\b](mailto:+@[A-Z0-9.-]+\.%5bA-Z%5d%7b2,%7d\b)”. This ensures that the string value answer has an “@” followed by any length of alphanumeric characters and a “.” followed by any length of alphanumeric characters.

Other validation methods check if an answer exists in a predefined list, if an answer is an integer and falls between a numeric range and if the user is of a valid age to drive.

### 8.4.9 QuotingBot.DAL

This project within the solution is responsible for all database interaction within the solution. The main point of interest here are the classes inside of the Repository folder.

These repository classes have methods that create a database connection, build an object and make a call to a stored procedure held in the database (Figure 41).

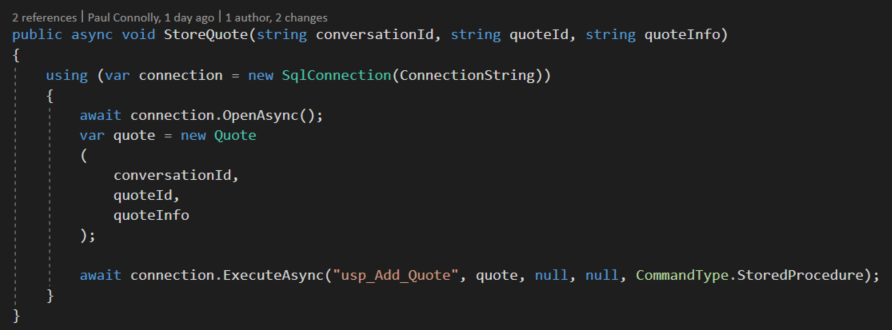


Figure 41 - StoreQuote() in QuoteRepository.cs

These repositories make use of Dapper. Dapper is an object auto-mapper framework that maps query output to a C# class (Code Digest, n.d.).

### 8.4.10 QuotingBot.DbUp

DbUp is a library for .NET that helps with the deployment of changes to SQL Server databases (DbUp, 2018).

It has been implemented in this solution to make the creation of the database fast and simple. Database changes and the schema of the database are better managed using DbUp.

#### 8.4.10.1 Program.cs

Creating a console application and using DbUp the code in Figure 42 was added to automate the running of the baseline schema script and upgrade scripts.

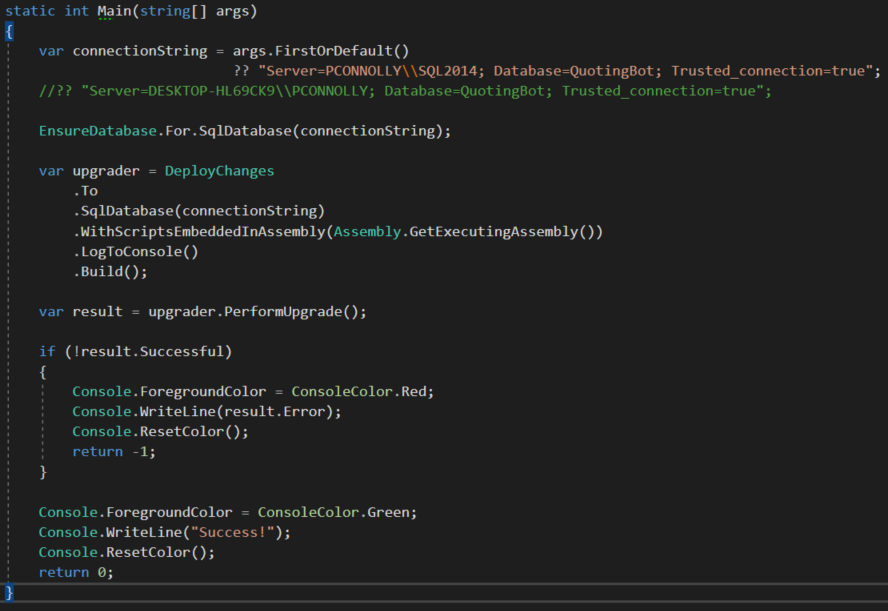


Figure 42 - Program.cs of QuotingBot.DbUp

# 9 Challenges and Solutions

# 10 Testing and Results

*Validation and verification*

*Evaluation of process and methods used to reach outcome*

# 11 Evaluation

Fulfilment of project objectives

# 12 Conclusions

# 13 Suggested Future Improvements to Project

Although the aim of the project was achieved, areas for future development and improvement of the bot have been identified.

All identified areas have been documented with recommendations made for how the work could be complicated and an estimation given on how long the work would be expected to take.

## 13.1 Link to Applied Systems eQuote Leads System

The Applied Systems eQuote Leads System is a system that collates all online business for an insurance broker. Currently this system is only populated with users, referred to as “leads”, and their data when they complete a quote insurance form via a broker’s website.

The chatbot functionality could be extended to insert a lead to eQuote Leads when a user has received a quote from the chatbot.

This would be achieved by making a call to an exposed endpoint in the eQuote Leads API for inserting a lead. Work would also need to be undertaken on the eQuote Leads system to extend the list of enumerations that determine the source of the lead. These enumerations could be extended to include ‘chatbot’ or ‘Facebook Messenger’ as the source.

An estimation on this work would be for it to take between 4 and 5 days to complete the implementation and testing.

## 13.2 Add Natural Language Processing

As this chatbot was built to solve a very definite problem, it was not so important to include a natural language processing service in this solution.

Should the chatbot be required to answer more “open” questions, introducing LUIS to the current solution would be a suitable method to handle these requests. Creating and training a LUIS model to handle user intents would give the chatbot the ability to deal with whatever the model is designed to handle.

The current solution would need to be extended to include a new dialog with methods that are “tagged” with an intent setup with the LUIS model.

It is difficult to estimate a period for implementing a natural language processing service as it would be an on-going process to make sure the natural language processing model is trained to handle all intents required. However, the initial implementation of creating a model in LUIS with one or two intents and implementing the model to the chatbot solution would take no longer than 1 day (7.5 working hours).

## 13.3 Convert to JavaScript Widget

Another method of creating chatbots is using JavaScript. Doing this means the chatbot can be placed directly on a brokers site rather than going through a social media platform such as Facebook Messenger, as this chatbot does.

To do this would be a major rework of the project and would be estimated to take months rather than days or weeks.

Converting to a widget for a website would require more planning and research. It may transpire from this research that another framework and development language would be better suited instead of using the Microsoft Bot Framework and C#.

# References

Agile Manifesto. (2017, 10 29). *Manifesto for Agile Software Development*. Retrieved from Agile Manifesto website: www.agilemanifesto.org

Amazon Web Services. (2018). *What is Source Control?* Retrieved April 16, 2018, from AWS: https://aws.amazon.com/devops/source-control/

Applied Systems. (2018, 04 11). *About Us - About Applied*. Retrieved from Applied Systems website: https://www1.appliedsystems.com/en-ie/about-us/about-applied/

Berry, D. (2017, March 03). *About Language Understanding (LUIS)*. Retrieved April 15, 2018, from Microsoft Azure: https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home

Boehm, B. W. (1988). *A Spiral Model of Software Development and.* TRW Defense Systems Group. Retrieved April 14, 2018, from http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

Box, D., & Hejlsberg, A. (2007, February). *LINQ: .NET Language-Integrated Query*. Retrieved April 22, 2018, from Devloper Network: https://msdn.microsoft.com/en-us/library/bb308959.aspx

Code Digest. (n.d.). *What is Dapper? How to Use Dapper in Asp.Net MVC?* Retrieved April 22, 2018, from Code Digest: http://www.codedigest.com/quick-start/17/what-is-dapper-how-to-use-dapper-in-aspnet-mvc

Codecademy. (2018). *What is REST?* Retrieved April 16, 2018, from Codecademy: https://www.codecademy.com/articles/what-is-rest

Comodo. (2018). *What is HTTPS?* Retrieved April 16, 2018, from Instant SSL: https://www.instantssl.com/ssl-certificate-products/https.html

Conversable. (2018). *About Conversable*. Retrieved April 14, 2018, from Conversable: http://conversable.com/about/

DbUp. (2018). *DbUp: Easy SQL Server change script runner*. Retrieved April 22, 2018, from DbUp: https://dbup.github.io/

DialogFlow. (2018, February 21). *Agents*. Retrieved April 15, 2018, from DialogFlow: https://dialogflow.com/docs/agents

Gordiyenko, S. (2014, November 3). *Software Development Life Cycle (SDLC)*. Retrieved April 14, 2018, from XB Software: https://xbsoftware.com/blog/software-development-life-cycle-waterfall-model/

Huckstep, R. (2017, December 13). *10 Insurtech Trends at the Insurance Crossroads 2018*. Retrieved April 13, 2018, from The Digital Insurer: https://www.the-digital-insurer.com/blog/10-insurtech-trends-insurance-2018/

Inflectra. (2018, March 1). *Requirements Gathering*. Retrieved April 16, 2018, from Inflectra: https://www.inflectra.com/ideas/topic/requirements-gathering.aspx

International Software Testing Qualifications Board. (2017, 10 29). *What is Spiral model- advantages, disadvantages and when to use it?* Retrieved from ISTQB website: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/

ISTQB Exam Certification. (2018). *What is Agile model – advantages, disadvantages and when to use it?* Retrieved April 14, 2018, from ISTQB Exam Certification: http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/

Lemonade. (2018). *Lemonade Renters and Home Insurance*. Retrieved April 15, 2018, from Lemonade: https://www.lemonade.com/

Lemonade. (2018). *Sign up for Lemonade Renters & Home Insurance*. Retrieved April 15, 2018, from Lemonade: https://www.lemonade.com/start/1

Lemonade. (2018). *The Lemonade Insurance API*. Retrieved April 15, 2018, from Lemonade: https://www.lemonade.com/api

Lengstorf, J. (2018). *JSON: What It Is, How It Works, & How to Use It*. Retrieved April 16, 2018, from Copter Labs: https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/

Maruti Techlabs. (2018). *Complete Guide on Bot Frameworks*. Retrieved April 15, 2018, from Maruti Techlabs: https://www.marutitech.com/complete-guide-bot-frameworks/

Maruti Techlabs. (2018). *How to Design a Conversation for Chatbot?* Retrieved April 20, 2018, from Maruti Techlabs: https://www.marutitech.com/design-chatbot-conversation/

MedWhat. (2017). *About Us*. Retrieved April 14, 2018, from MedWhat: https://medwhat.com/about-us/index.html

Microsoft. (2017). *Mircosoft Bot Framework*. Retrieved April 15, 2018, from Bot Framework: https://dev.botframework.com/

Morse, B. (2017, April 20). *Become Part of the Story with the Marvel Comics Chatbot*. Retrieved April 14, 2018, from Marvel: https://news.marvel.com/comics/63529/become-part-story-marvel-comics-chatbot/

Moubarak, M. (2018, February 16). *Smart Chat For Your Real Estate Website: Qualify Buyers And Sellers in Real-Time With Roof.ai*. Retrieved April 15, 2018, from roof.ai: https://blog.roof.ai/roofai-real-estate-chatbot/

Mozilla. (2018, April 3). *What is JavaScript?* Retrieved April 16, 2018, from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\_steps/What\_is\_JavaScript

Ngan, K. (2018). *What is Git?* Retrieved April 16, 2018, from Microsoft Visual Studio: https://www.visualstudio.com/learn/what-is-git/

Nguyen, M.-H. (2017, October 6). *How artificial intelligence & machine learning produced robots we can talk to*. Retrieved April 14, 2018, from Business Insider UK: http://uk.businessinsider.com/what-is-chatbot-talking-ai-robot-chat-simulators-2017-10

PwC. (2017). *PwC Irish Total Retail 2017 Survey*. Retrieved April 13, 2018, from PwC Irish Total Retail 2017 Survey: https://www.pwc.ie/publications/2017/ten-retailer-investments-for-an-uncertain-future-total-retail-survey.pdf

Python. (2018). *What is Python? Executive Summary*. Retrieved April 16, 2018, from Python: https://www.python.org/doc/essays/blurb/

RoofAi. (2018). *Smart chat for your real estate website*. Retrieved April 14, 2018, from RoofAi: https://www.roof.ai/

Rouse, M. (2010, April). *Ruby*. Retrieved April 16, 2018, from Whatis.com: https://whatis.techtarget.com/definition/Ruby

Rouse, M. (2017, August). *Microsoft SQL Server*. Retrieved April 16, 2018, from TechTarget: https://searchsqlserver.techtarget.com/definition/SQL-Server

Ruby. (2018). *Ruby is...* Retrieved April 16, 2018, from Ruby: https://www.ruby-lang.org/en/documentation/

Samit, J. (2017, December 26). *4 Technology Trends That Will Transform Our World in 2018*. Retrieved April 13, 2018, from Fortune: http://fortune.com/2017/12/26/4-technology-trends-2018/

Sens, R. (2017, June 13). *Want to design better chatbot dialog?* Retrieved April 17, 2018, from Medium - Prototypr: https://blog.prototypr.io/design-better-chatbot-dialog-using-the-five-ws-framework-628bda212f39

Stackify. (2017, 10 04). *What is SDLC? Understand the Software Development Life Cycle*. Retrieved from Stackify website: https://stackify.com/what-is-sdlc/

Techopedia. (2018). *What is a Relational Database (RDB)?* Retrieved April 16, 2018, from Techopedia: https://www.techopedia.com/definition/1234/relational-database-rdb

The Keyes Company. (2018). *Florida Real Estate & Homes For Sale*. Retrieved April 15, 2018, from The Keyes Company: https://www.keyes.com/

Tutorials Point. (2017, 10 29). *SDLC Agile Model*. Retrieved from Tutorials Point website: https://www.tutorialspoint.com/sdlc/sdlc\_agile\_model.htm

Tutorials Point. (2017, 10 29). *SDLC Overview*. Retrieved from TutorialsPoint: https://www.tutorialspoint.com/sdlc/sdlc\_overview.htm

Velloso, M., & Standefer, R. (2018, December 13). *Design a bot's first user interaction*. Retrieved April 17, 2018, from Microsoft Azure: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-design-first-interaction

Velloso, M., Iqbal, K., & Standefer, R. (2017, Decmber 13). *Principles of bot design*. Retrieved April 17, 2018, from Microsoft Azure: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-design-principles

Vogel, E. (2011, March 24). *Asynchronous Programming in .NET: I'll Call You Back*. Retrieved April 22, 2018, from Visual Studio Magazine: https://visualstudiomagazine.com/articles/2011/03/24/wccsp\_asynchronous-programming.aspx

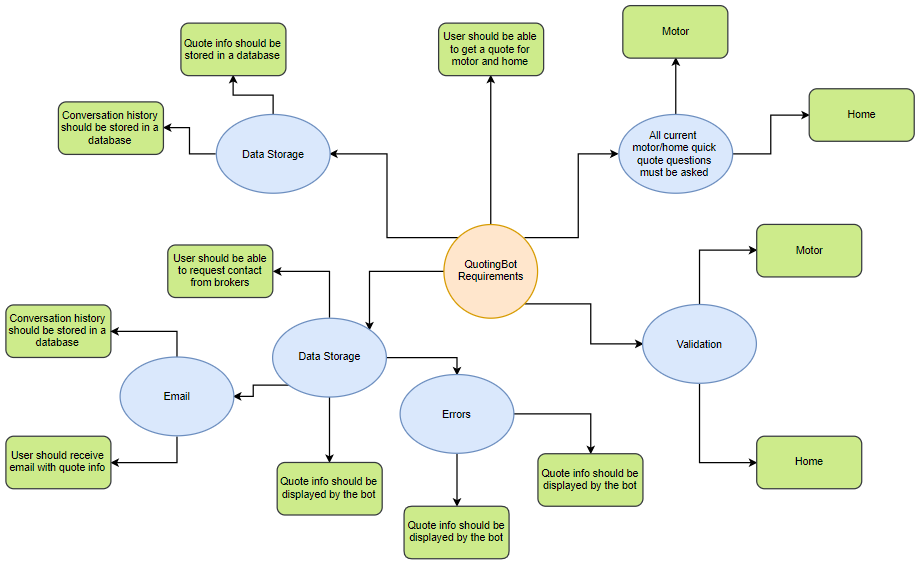
Wagner, B., Wenzel, M., & Levin, I. (2015, July 20). *Introduction to the C# Language and the .NET Framework*. Retrieved April 16, 2018, from Microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Wikipedia. (2018, April 3). *Software framework*. Retrieved April 15, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Software\_framework

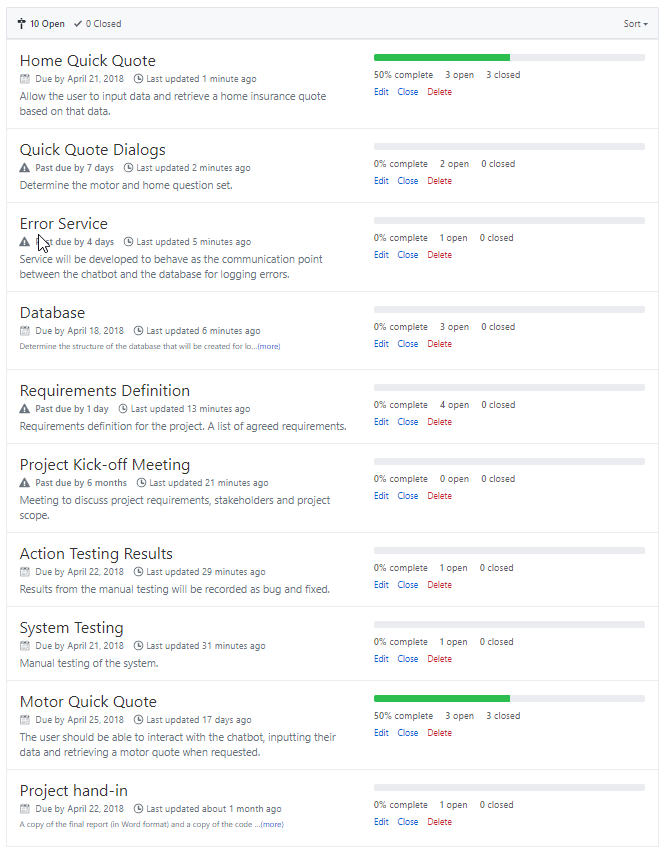
Wissner-Levy, Y. (2016, September). *How Lemonade is Fixing a Broken System*. Retrieved April 15, 2018, from Lemonade: https://www.lemonade.com/blog/hello-world/

wit.ai. (2018). *Wit.ai*. Retrieved April 15, 2018, from wit.ai: https://wit.ai/

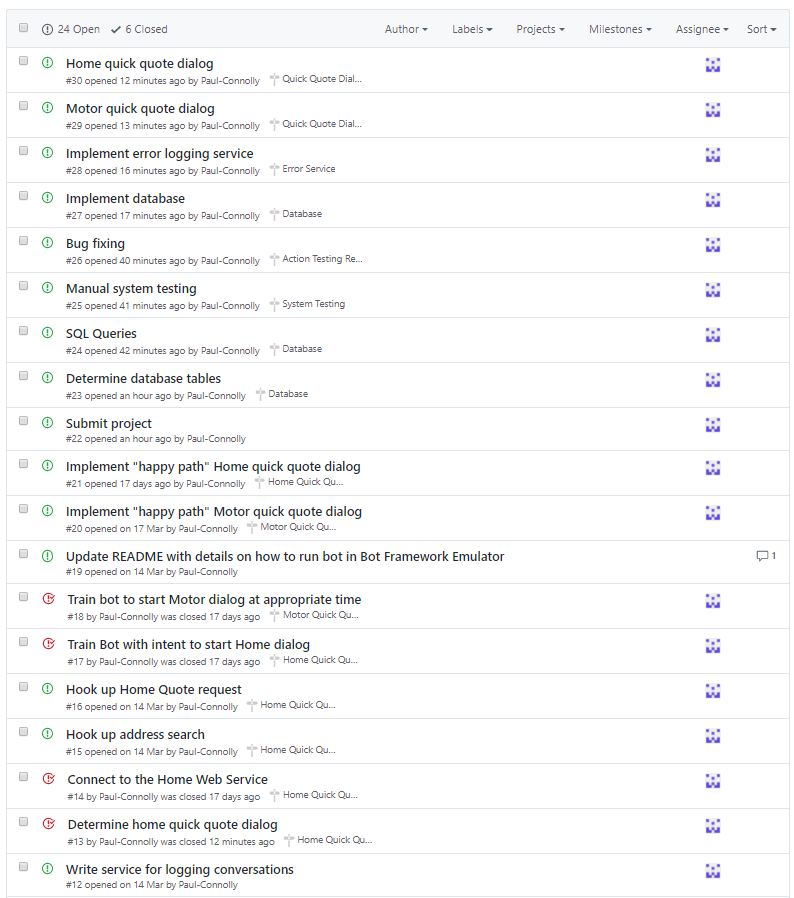
# Appendices



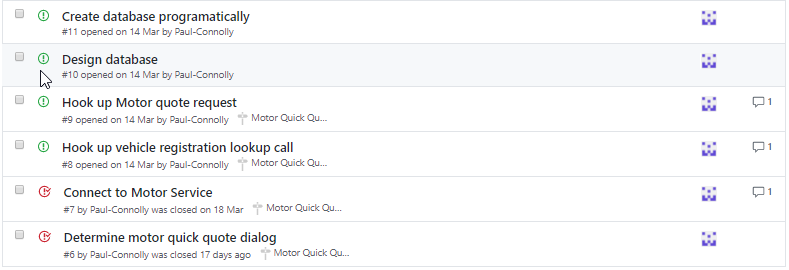
Appendix A - Brainstorming from requirements gathering



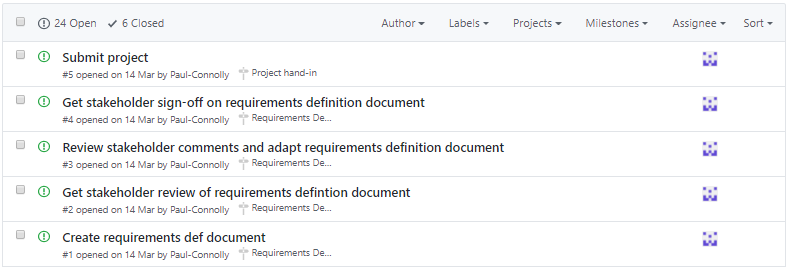
Appendix B – Project milestones taken from GitHub



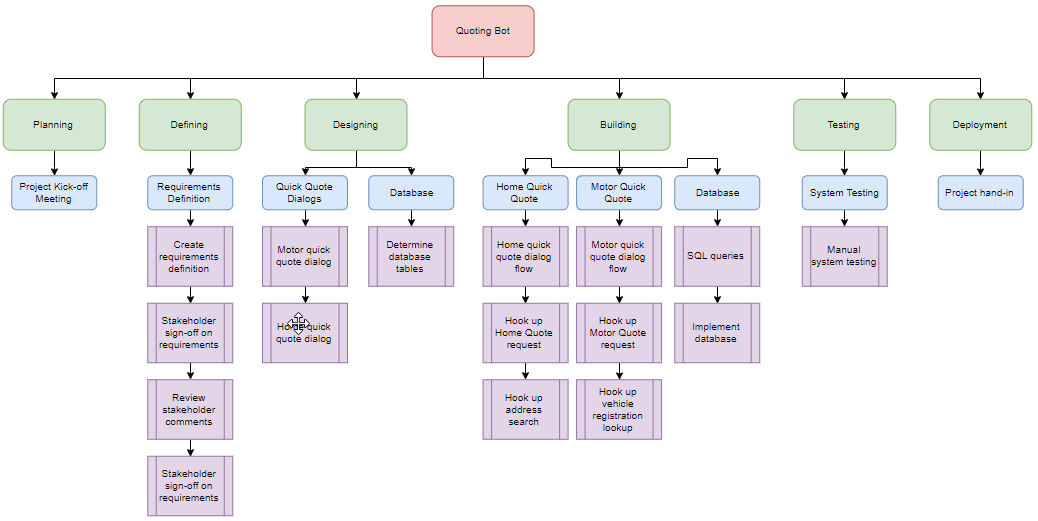
Appendix C - Issues from GitHub



Appendix D - Issues from GitHub cont.



Appendix E - Issues from GitHub



Appendix F - Work Breakdown Structure

1. A ‘quick quote’ is a reduced set of questions that still allows insurance quotes to be returned when requested. For questions not asked, default answers are set. [↑](#footnote-ref-1)