Sajjad Hosseini

Chat Bots

Implementation and User Engagement

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

2 April 2017

To my mother,

whose presence is the source of warmth in my heart;

from whom I have what I have,

I am who I am.

echo -e '\xE2\x9D\xA4'

Author(s) Title Number of Pages Date	Sajjad Hosseini Chat Bots Implementation and User Engagement 31 pages 2 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Olli Hämäläinen, Senior Lecturer

The purpose of this project was to study the concept of Chat Bots, and to analyse its importance and place in today's communication platforms. The goal of the project was to build a Chat Bot prototypically for Metropolia University of Applied Sciences helpdesk on a messaging platform, which would replace the traditional way of reserving old devices for purchase sold at Metropolia's helpdesk in different times throughout the year.

The chat bot was developed using a high-level implementation library of Telegram's Bot Application Programming Interface, and it was hosted on Heroku Cloud Application Platform. The requirements of the Minimum Viable Product Chat Bot application were already defined, and therefore the project was carried out with Waterfall software development model.

As a result, Metropolia's helpdesk Chat Bot was successfully developed according to the requirements, and it was found that due to the nature of the Chat Bots, the development can become complex quickly.

All in all, major software companies in the world are investing heavily on the Chat Bot concept and its usage is increasing steadily throughout the world.

Keywords	Chat Bot, Telegram, software development

Contents

1 Introduction

We live in the era of constant high-volume high-intensity communication. Different applications, different approaches, and different platforms. However, it was not until recently that a new phenomenon came to life, named Chat Bots. Chat Bots being new leaves us with questions such as whether they are financially supported to ensure their continuous growth, how is their software development practices, and more importantly, whether they are gaining popularity worldwide.

The answers to the overall growth of Chat Bots are important in a sense that they would give us information whether to invest our funds, time, and energy as business managers, developers and end-users in developing and using Chat Bots. I chose the topic of Chat Bots because we are at the early stages of this new phenomenon and therefore I noticed a gap of information related to the overall structure and concept of Chat Bots.

This thesis will serve the purpose of introducing the fundamentals of the Chat Bot concept, and analyse its importance by researching its current state. Furthermore, a narrowed research is done to provide enough information to evaluate a short-term growth potential of Chat Bots. The outcome is then analysed for its reliability and it will be looked upon from an objective perspective for the shortcomings of the results.

To be able to gain a practical understanding of the concept of Chat Bots, I developed a Chat Bot on the Telegram messaging application platform. The Chat Bot enables students from Metropolia University of Applied Sciences to reserve a monitor which is going to be sold by Metropolia's helpdesk from the older monitors which they can no longer use within the Leppävaara campus. Traditionally, the process of reserving peripherals which were about to be sold never existed, and students had to wait in long queues for hours before the sale was to start, and then proceed to purchasing the device they wanted to purchase. There was no need to develop a client application for the students to use the logic of reserving monitors, there was no need to authenticate the students as the authentication was handled already by the Telegram platform, and there was no need to worry about the maintenance of the client application as this was handled by the platform itself. I just had to write the logic necessary and deploy the application to Heroku, a cloud application platform, and students could immediately start using the Chat Bot and reserve monitors.

2 Background to Chat Bots

To understand any complex system, at first, an abstracted high-level introduction of the subject and explanation of the architecture which holistically looks at the complex system and allows for a better grasp of details later is not only necessary but also important. Therefore, in this chapter, there will be an introduction section to the fundamentals of Chat Bots and how they fit relative to other messaging concepts into the bigger picture of our modern communication. In addition, there will be a section that goes through the history of Chat Bots and from when they started to appear and gain growth. Finally, there will be a section about Chat Bots and their current state, the unknown and the known about them, which illustrates the importance of this thesis to tackle the unknown aspect.

2.1 Fundamentals of Chat Bots

In our modern world, virtual communication comprises of a big portion of our daily lives. We communicate virtually in our personal lives with friends, family and the loved ones. In our professional lives, we rely on virtual communication heavily for transferring information from a place to another, and to do tasks which had to be carried out traditionally manually; for instance, approving someone's documents and notifying them of the decision electronically. Electronic management of tasks have revolutionized the way we work and live nowadays, but with the introduction of Chat Bots, there has been even more disruption brought into our personal and professional lives.

A Chat Bot, could be thought of as someone who is awake 24 hours 7 days a week and they will answer the user to the best of their knowledge, except that they are not humans. From a technical perspective, a Chat Bot refers to a computer program written to engage in a text-based conversation with a human, with the intention of simulating human-like behaviour. There are simple Chat Bots all over the Internet whose technical implementation only allows them to be used for very narrowed use cases like searching for the nearest restaurant after retrieving one's location. However, there are complex Chat Bots on the Internet which benefit from Artificial Intelligence using Natural Language Processing and therefore can handle unpredictable inputs from users and learn from their data over a course of time.

To understand the bigger picture of our virtual communication from a messaging perspective, Figure 1 illustrates the hierarchy.

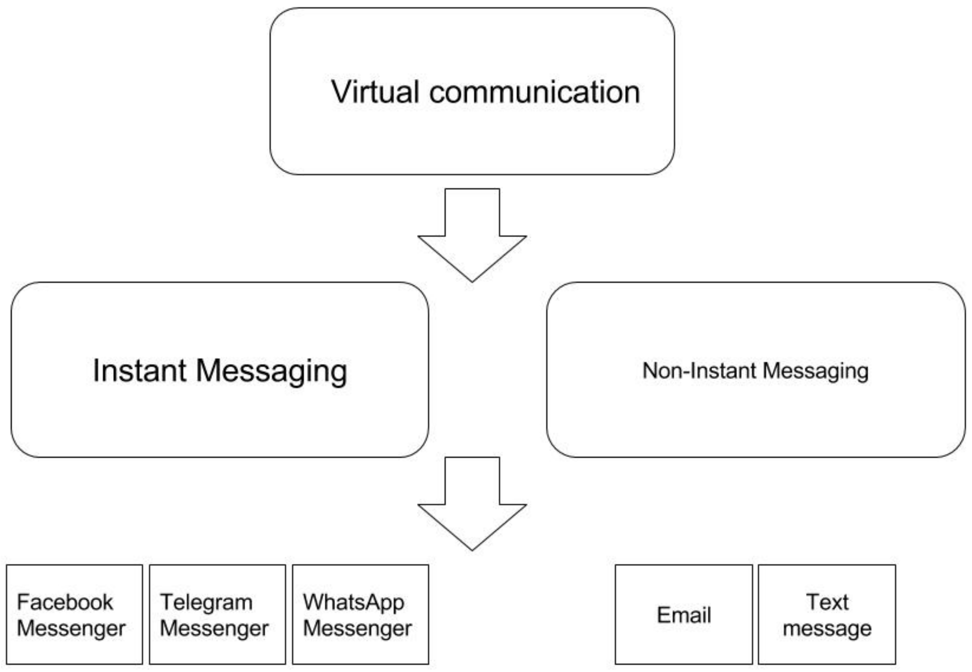


Figure 1. High-level architecture of our virtual communication

As can be seen from Figure 1, virtual communication could be divided into two different types of messaging concepts. Instant Messaging could be an application where one feels more connected with individuals while communicating textually. From a technical perspective, Instant Messaging applications are computer programs which use WebSocket or other real-time communication protocols which provide the reliability and speed required for real-time text transmission. On the other hand, Non-Instant messaging is a virtual communication type of messaging where one would feel the transmission is more sequential and the communication could be thought of as throwing a ball to another person, once received by the other party, it would be observed and acted upon accordingly. From a technical perspective, Non-Instant messaging applications use communication protocols which do not provide constant real-time transmission of information needed for the concept of Chat as we think about it nowadays but rather serially executed requests, like Simple Mail Transfer Protocol or Hypertext Transfer Protocol.

A simple distinction which could be easily observed while working with Instant and Non-Instant messaging applications is the ability to observe the other person’s actions while communicating. For example, in modern Instant messaging applications one could observe if the other person is typing at the same time, whereas this is not possible while using e-mail clients or while sending text messages via phone operator’s network.

Chat Bots being new in the scene of modern communication applications needs to be categorized. Since Chat Bots inherently use the same communication protocols as Instant messaging applications, they are to be categorized under Instant-Messaging. The high-level illustration of virtual communication architecture could be more complex and take human-based communication into account, like communicating textually with a friend, and computer-based communication, like communicating textually with a Chat Bot. However, this level of abstraction and architecture detail suffices for understanding of the fundamentals of Chat Bots and how they fit into the bigger picture.

2.2 History of Chat Bots

Regarding the term “Chat Bots”, Michael Loren Mauldin, born in 1959, is an American inventor and scientist who created one of the early Chat Bots named Verbot [1]. In addition, Mauldin also patented the term ChatterBot in 1994 to explain conversational software programs [2].

Regarding the behaviour and nature of Chat Bots, one needs to know the concept Turing test. Alan Turing, an Englishman known as the father of computer science, who was a computer scientist, cryptanalyst, logician and mathematician, in 1950 published "Computing Machinery and Intelligence", an article which is used nowadays as the basis of intelligence assessment of computer programs. The Turing test criteria depends on a computer program’s ability to simulate human-like behavior in a written real-time conversation, approved by a human judge, well enough so that the human judge is unable to distinguish reliably enough whether the person replying is a human or a computer program. With the Turing test, there was clear criteria for people to depend on when thinking of implementing a computer program which would simulate human-like behavior, and this set a starting point for people in the 1950’s and the 1960’s to investigate the concept further. [3.]

Joseph Weizenbaum, a German-American computer scientist and professor at Massachusetts Institute of Technology, influenced by the Turing test criteria, published a program in 1966 called ELIZA. The ELIZA program could give users communicating with it an illusion of communicating with a real human being. Joseph Weizenbaum did not think of ELIZA as an “intelligent” program, but rather as a program which only gives the illusion of intelligence.

ELIZA’s primary logic to be able to give responses and move the conversation forward in a human-like fashion was to analyse the input given by the users by searching for phrases and keywords, and give pre-planned pre-programmed responses. For example, given a sentence like “I used to hang out with my father a lot in my childhood.”, ELIZA would do the following in order:

- Receive the input and store them in memory for further analysis
- Searching the sentence, or technically put the whole string, for keywords
    - The keywords are already defined
    - For the sake of this example, we could think of the keywords being
        - “mother”, with a response like “tell me more about your mom”
        - “father”, with a response like “tell me more about your father”
- If any of the words within the sentence would match, giving back the pre-programmed response to the user
- If there’s no clear match, giving back a default answer like “Sorry, don’t know about that”.

Given the steps above, an illusion of understanding is created, even though the analysis method is trivial from a development perspective. [3.] From 1966 onwards, business owners and computer programmers realized the usefulness Chat Bots can provide to end users, especially when the information can be categorized into concrete and predictable subjects. For example, a repair store which has technicians who can repair refrigerators, microwaves, and washing machines could provide its end users a conversational Chat Bot which can engage with the customers in a human-like fashion. Then, since it is highly probably that customers would refer to the item they have at home which is broken, the Chat Bot could refer them to the appropriate technician with their respective phone number, their availability, and other relevant pre-saved information accessible to the Chat Bot.

2.3 Current State of Chat Bots

From a community perspective, at the moment there is a website called ChatBotMagazine which focuses on providing the latest content with regard to Chat Bots. ChatBotMagazine was founded in April 2016, one year prior to the publication of this thesis, which indicates that the whole concept of Chat Bots is new. With above 250,000 views on a monthly basis, they claim to be the fastest growing Bot magazine with an audience from a wide spectrum of software developers, business owners, bot makers and journalists. ChatBotMagazine is managed by over 400 Chat Bot enthusiasts all around the world. [4.]

From a development perspective, Facebook, the only social media platform with as high a number of users, launched their Messenger Platform for public in April 2016, again, one year prior to the publication of this thesis. Messenger is the name of a computer program owned by Facebook which aimed to replace the traditional built-in simple chat program within the Facebook web application itself. Facebook Messenger Platform is the foundation upon which people can use the Messenger to chat with their friends and bots, and at the same time, for software developers to develop bots for the Messenger application. At the moment, to be able to develop and deploy a Chat Bot on Facebook, there are terms of usage and policies one has to follow, and afterwards, one has to submit their Chat Bot for review for the Facebook development team, a process which Facebook claims to ensure the best interactions between developers and Facebook and to put people first. [5.]

There are other messaging applications which provide a platform upon which software developers could create and maintain their Chat Bots. Slack, Telegram, Skype, Viber and Line are a few examples. The platforms will differ when it comes to smaller details of implementation and deployment of the Chat Bot to their respective platforms. However, from a broader perspective, the steps are as follows:

- Selecting the platform upon which one wants to develop a Chat Bot
- Analysing the platform's Application Programming Interface
- Researching whether the platform provides a Software Development Kit for the language of one's choice
- Researching platforms which provide support for deploying one's application logic

- Implementing the Chat Bot either manually with the Application Programming Interface or using the platform’s Software Development Kit
- Submitting the Chat Bot for review, if the platform requires it
- Ready for testing and being used.

To handle the details of implementation regarding each platform’s specific implementation and to eliminate the need for creating the same application logic for each platform if one wishes to provide users with multiple platforms to engage with, there are software libraries. Claudia Bot Builder is an example which allows writing the application logic once in the JavaScript programming language and deploy it to Amazon Web Services Lambda Service, and therefore abstracting each of the supported platforms specialities and allowing the developer to focus on improving the core logic of the application. [6.]

Modern Chat Bots are new, only 1-year old at the time of this thesis publication, and therefore it is important to understand the opportunities and challenges both at the same time. On the side of opportunities, we can see already how different platforms are pushing their users into a better understanding of Chat Bots and therefore increasing the user’s total engagement time with the Chat Bots. High-engagement with Chat Bots and messenger applications in general provides businesses a platform from which they can get access to over 1 billion people on the Facebook platform alone, an opportunity which no other platform or advertising method could provide. In addition, for the people unfamiliar with usually complex web applications to reach for relevant information, a Chat Bot they could chat with and immediately get relevant responses increases user satisfaction.

On the side of challenges, from a financial perspective it is unclear how much companies are investing in long-term support and advancement of Chat Bots. In the absence of financial support for further development of modern Chat Bot platforms, new businesses will be hesitant to provide their users with Chat Bot applications since the risk of investment and low Return on Investment remains high without clear financial goals set by each platform for advancement of their respective Chat Bot platforms.

From a user engagement and statistical perspective, with only a year worth of modern messaging application usage by the users, it is difficult to assess whether the data will repeat itself, or just like any other newly born technology, the number of users will peak in the beginning and drop in the coming years. Statistics are the essential information businesses rely on to assess whether they can reach their targeted users and if yes, to what extent and how efficiently.

Finally, from a development perspective, as will be discussed in the following chapter, developing Chat Bots could become complex quickly. There are not enough libraries to do the heavy lifting of Chat Bot development for the developer and help with the modularisation and separation of concerns, and therefore it is challenging from a software engineering perspective to develop complex Chat Bots with the currently available tools.

Given the opportunities Chat Bots could potentially provide us, it is worthwhile to address the challenges. In Chapter 4 of this thesis, there will be information about all three challenges of financial, statistical and development, mentioned above, in an effort to minimize the ambiguity and provide better clarity over the matters.

3 Chat Bot Concept Growth

Potential growth of Chat Bots is essential information to businesses, software developers, and users. In the absence of information which would provide insight as to whether Chat Bots continue to exist and evolve at least in the short term, business owners cannot risk investing in Chat Bots, software developers cannot risk investing time in learning all there is to Chat Bots in terms of development practices, and users would not feel comfortable getting familiar with a new phenomenon which will cease to exist. Therefore, in this chapter, I will first investigate the financial situation and short-term potential growth of Chat Bots. Then, I will investigate the statistics regarding the number of users and their usage models when it comes to Chat Bots. Finally, I will investigate the available tools for creating Chat Bots and whether there are co-ordinated plans to further develop those tools or to create better new ones.

3.1 Financial Aspect

Since 2008 until 2016, more than 140 million dollars were invested in Chat Bots, with around 60 million dollars of which only in 2016.

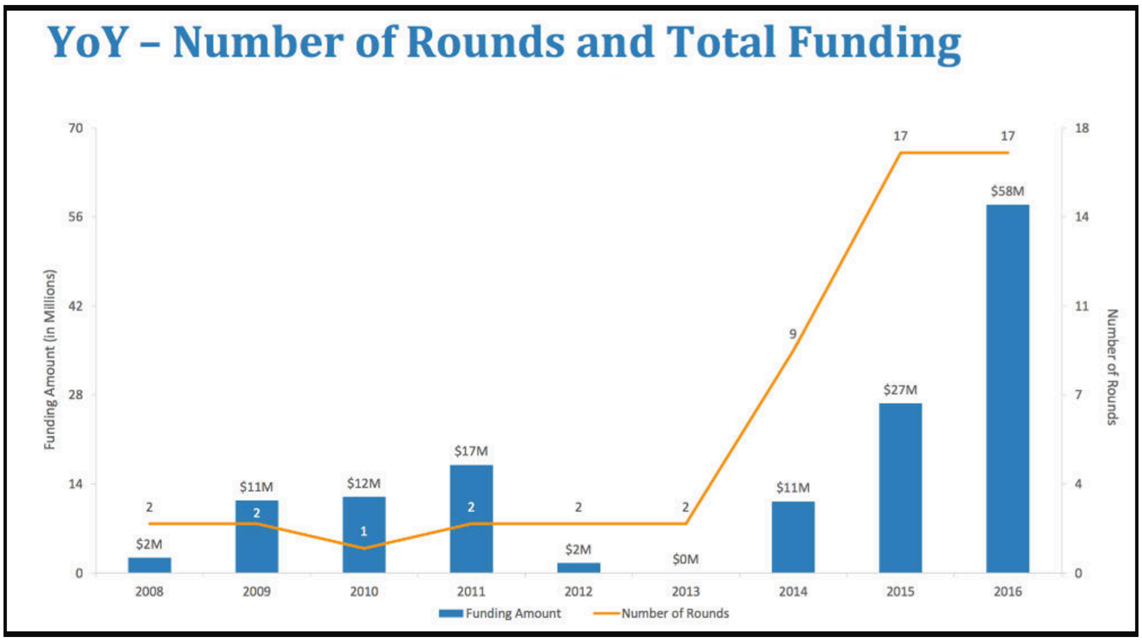


Figure 7. Illustration of funding amounts on Chat Bots [12].

As can be seen from Figure 7, the amount of funding has been steadily increasing if we factor out 2012 and 2013. The dramatic increase in funding which happened in 2016 could be explained by the fact that in 2016, widely known technology companies such as Google, Facebook, Microsoft and Apple announced opening of their Chat Bot platforms for developers to build Chat Bots upon [13].

One of the industries that has focused on Chat Bots is Financial Institutions. According to research, 80 percent of Financial Institutions globally perceive Chat Bots as an opportunity and more than 70 percent of Financial Firms either already have active projects on Chat Bots or they already have a plan for the next 12 months. [13.]

3.2 Statistical Aspect

Currently there are no statistics which provide insight into how many users interact with Chat Bots on popular messaging platforms and how much time each user spends in a day interacting with Chat Bots on each platform. However, Facebook Messenger which has more than 1 billion active users, has publicly mentioned statistical information regarding users on their Chat Bot platform which would be of help in the scope of this thesis.

In September 2016, Head of Facebook Messenger David Marcus announced that there are 34,000 software developers who have joined the platform since its launch in April 2016, and have created 30,000 Chat Bots since then. The number of software developers who had joined the platform since its launch until May was only 10,000 and the number of Chat Bots created on Facebook Messenger platform from its launch until July was only 11,000. That means a 24,000 increase in the number of software developers within a time span of 4 months, and an increase of around 19,000 in the number of Chat Bots within a time span of 2 months. [14.]

From users' perspective, according to research, more than 50 percent of people think that a business availability every hour every day is important. Furthermore, 45 percent of people prefer contacting a business through messaging than the traditional e-mail. Finally, almost 50 percent of people rather contact a business through messaging than a phone call. [15.]

3.3 Development Aspect

One of the immediate problems that comes across while developing Chat Bots is a situation in which one would like to have their Chat Bot on multiple platforms. One solution would be to develop a Chat Bot for each messaging platform one would like to have their Chat Bot on. However, developing an identical application logic several times only to be able to have them on multiple platforms is not a sustainable solution. Then each minor and major fix should be applied in all instances of Chat Bots across multiple platforms and it is not cost-efficient.

After researching a possible solution, I found a project called Claudia Bot Builder which claims to abstract away the underlying details of Facebook Messenger, Telegram, Skype, Slack, Twilio, Kik and GroupMe platforms and allows the developer to write the application logic once while at the same time giving the ability to further configure each platform it supports for more complicated workflows.

There is another Chat Bot framework called Botmaster which claims to abstract away the underlying details of writing a Chat Bot for Facebook Messenger, Slack, Twitter DM, Telegram and socket.io platforms. However, with only 300 commits, 2 contributors, and 90 stars as of April 2017, Botmaster is in its early stages of its development and therefore not a library to be used in production scale without taking extra steps for handling the predictable immaturity of the project.

4 Chat Bot Service Design

To explore the functionalities offered by current time's messaging application platforms and to apply theory to practice, I will explain the implementation of the Chat Bot I created for Metropolia University of Applied Sciences. In this chapter, there will be a section to analyse application logic minimum requirements and features, making it a Minimum Viable Product. In the following section, I will explain the high-level architecture of the application and planning required before Chat Bot implementation. The most important elements in Metropolia's helpdesk Chat Bot service design are Telegram messaging platform and Heroku hosting service. There are other messaging platforms offering the possibility of creating Chat Bots, but Telegram's security for its end users is the highest priority for the platform and therefore I decided to choose Telegram for creating Metropolia's helpdesk Chat Bot. In a similar fashion, I decided to choose Heroku hosting service over its competitors for its high focus on the security aspect even though the price of the platform is higher due to providing abstraction and simplicity and high security all at once to end users.

4.1 Requirements of Minimum Viable Product

In Metropolia University of Applied Sciences helpdesk, there are old devices sales for students between two to four times a year. Metropolia's helpdesk maintains a list of old devices no longer in use and makes them accessible to students. Students are informed via e-mail or Metropolia's internal portal about the upcoming Metropolia helpdesk sales and they check the availability and specifications of the devices, and if they find a device they would like to purchase, they will attend the sales on the specified date.

On the sales day, two to four hours before the sales even starts, students wait in lengthy lines. Students would like to be among the first ones in the queue so they would be able to purchase the device they have in mind, without the risk of it being sold to the person in front of them in the line, and them waiting in vain. Lengthy lines cause frustration and confusion for students, and it puts a demanding and challenging task of logistics on the shoulders of Metropolia's helpdesk staff to give away all the devices purchased by the students all at the same time. In a course offered by Metropolia University of Applied Sciences where students get a chance to find a solution to a problem in an innovative fashion, I decided to solve Metropolia's helpdesk sales lengthy lines and the confusion and frustration that it causes to students.

After discussing with the teacher of the course in which we wanted to solve the old devices sales issue, we concluded the following characteristic for a Minimum Viable Product solution:

- The application could be used by people within and outside Metropolia University of Applied Sciences
- The application should not be hard to use to justify the usage instead of purchasing devices in the traditional way
- The application should have an admin role for Metropolia's helpdesk staff to be able to add devices to the database of available devices for sale
- The application should have a student role which can reserve the devices added by the Metropolia's helpdesk staff
- For now, students being able to reserve monitors suffices.

The following features were dropped to be able to concentrate on the most viable features providing the most value, but scheduled for future development plans if agreed by Metropolia's helpdesk staff:

- Production ready application
    - Robust with close to none downtime
    - Fault tolerance towards human input mistakes
    - Automated tests to ensure quality and prevent regression
- Reserving devices should include all devices sold at Metropolia's helpdesk sales and not only monitors
- The application should provide students a Portable Document Format document after reserving a device to be used as a receipt of reservation
- The application should provide a payment method for students to be able to pay for the device while reserving, or even afterwards.

After freezing the requirements, application architecture planning could start.

4.2 High-level Architecture

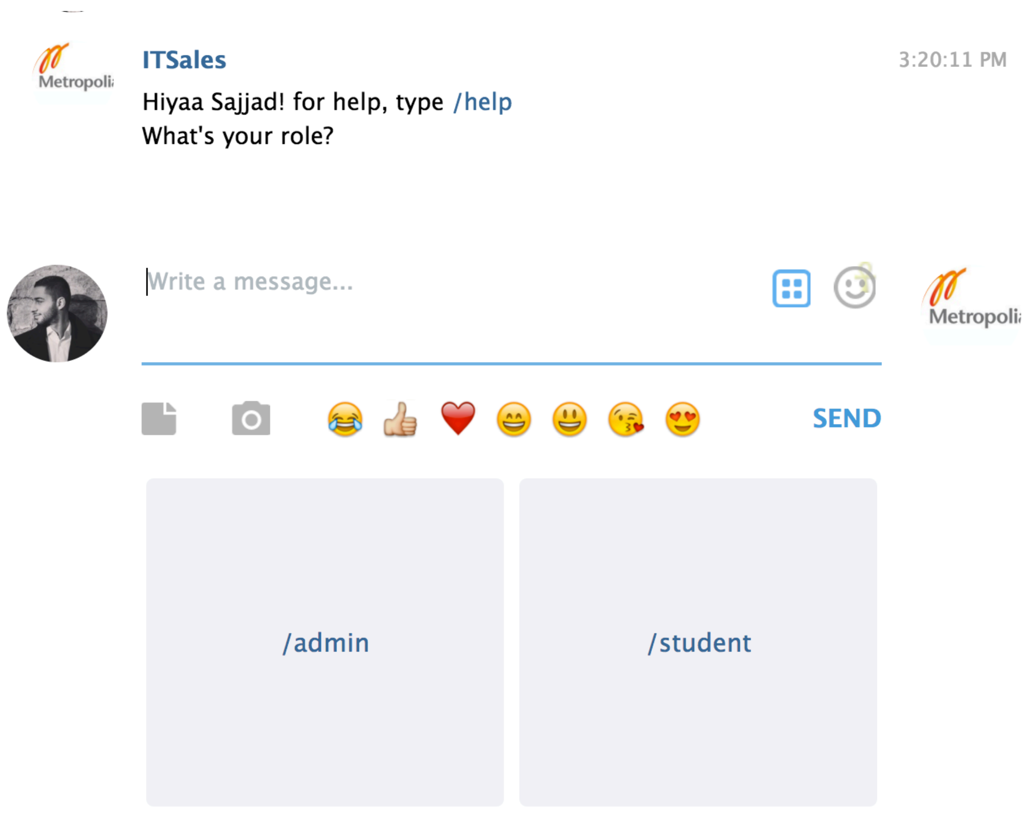
The application whose requirements were explained in the requirements section of Chapter 3 has many aspects which I had to take into consideration while architecting how to approach its implementation. In the end, I decided to build a Chat Bot to solve Metropolia's helpdesk old devices sales issue.

The application could be used by people inside Metropolia University of Applied Sciences as well as people outside. Before I am able to explain how to tackle the requirements, I will need to clarify two terms closely related but entirely distinct from one another, authentication and authorization. Authentication is the process of verifying an identity. For instance, when Sajjad and Sauvage would like to login to a university system, they will use their passwords respectively and the system will check whether the map between their username and password also exists in the application entity responsible for user management. Given the correct username and password, the system will authenticate Sajjad and Sauvage successfully. Authorization is the process of verifying whether one has enough rights and permission to access certain information within the system. In the example mentioned above, given Sajjad and Sauvage already authenticated and given that students are not allowed to access each other's' documents, Sajjad is not authorized to access documents owned by Sauvage, and vice versa.

For people within and outside Metropolia University of Applied Sciences to access its helpdesk sales, there were three options. The first option was to use Metropolia University of Applied Sciences own Single-Sign-On system, and ask Metropolia University of Applied Sciences' permission for adding a group-level permission to the Single-Sign-On system so people outside Metropolia can sign in with the Single-Sign-On system. The first option was not possible to be implemented due to limited time and the bureaucracy that had to be tackled before I would get my users to sign in. The second option was to build and maintain a user management system so that students in Metropolia University of Applied Sciences and the people outside would be able to use it. The second option was reinventing the wheel and due to limited time and the complexity involved in building a secure maintainable user management system, it was dropped. Finally, the last option was to delegate the user management and authentication to a third-party service.

Telegram web and mobile applications have the ability to authenticate users using the latest methods in the industry, like sending a confirmation message to the user's mobile phone or on top of that, using two-step verification to further secure their users and their platform. By having a secure sign-in procedures, it is not possible to easily sign up fake users programmatically, if at all possible. Therefore, by using the Telegram Chat Bot platform, I delegated users' authentication to the platform itself, which allowed me to focus on the core logic of the Chat Bot.

To tackle the requirements of the application being user-friendly and easy to use, I used ReplyKeyboardMarkup feature of Telegram Chat Bot platform.



ITSales 3:20:11 PM

Hiyaa Sajjad! for help, type /help

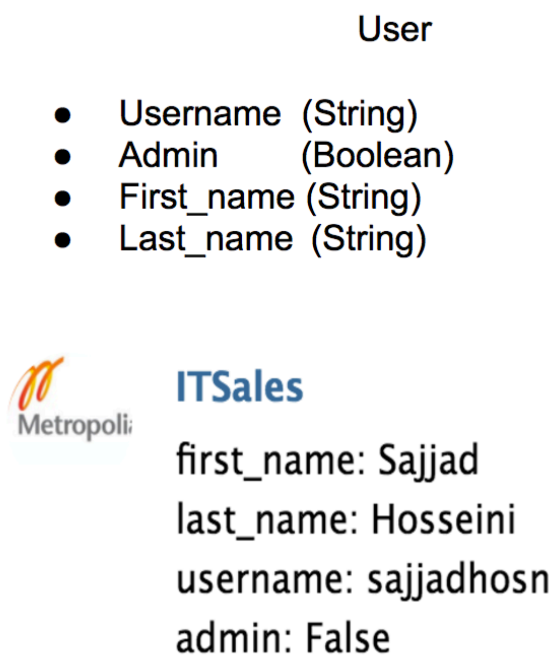
What's your role?

Figure 2. Illustration of ReplyKeyboardMarkup feature of Telegram Chat Bot platform

As can be seen from Figure 2, ReplyKeyboardMarkup allows customizing user's reply keyboard. In Figure 2, ReplyKeyboardMarkup is constructed with '/admin' and '/student' keywords, effectively allowing to guide the users to choose the appropriate answer in the flow of the application logic.

With proper delegation of ReplyKeyboardMarkup feature, the Telegram Chat Bot for Metropolia’s helpdesk does not fall into a state where the user does not have a customized keyboard to reply appropriately to the questions raised by the Chat Bot. Users can always issue the ‘/help’ command which allows them to see the basic commands of the Chat Bot. Therefore, with the users always having the appropriate responses at hand, the application can be justified to be easy to use.

To tackle the requirement of having an admin role on top of a student role, I added an admin flag to the “user” table of the database.



User

- Username (String)
- Admin (Boolean)
- First_name (String)
- Last_name (String)

ITSales

first_name: Sajjad
last_name: Hosseini
username: sajjadhosn
admin: False

Figure 3. Illustration of User table model in the database together with an example user

In this approach, each user communicating with Metropolia’s helpdesk Chat Bot will be added to the User table in the database whose data structure can be seen in Figure 3. An entry in the User table could be thought of as a student or someone outside Metropolia University of Applied Sciences. Abilities of a normal entry in the User table without the admin flag set to true will be limited to viewing as well as reserving monitors.

Through a process which will be explained in Chapter 5, a User could be promoted to an admin. After promotion, the admin flag on the promoted user will be set to true, and this will expand the user's permitted operations to include adding and removing monitors, and will exclude reserving monitors. Therefore, having an admin flag in the User table allows us to have a student role as well as an admin role, which fully tackles the requirements.

Finally, for fulfilling the last requirement, students having the ability to reserve monitors, I added a table named "Monitor" to the database, with the following attributes:

- id
- brand
- manufacture_year
- price
- reserved
- reserved_for

The entries added to the Monitor table would start with the reserved flag set to false. Once someone reserves a monitor, the corresponding entry's reserved attribute will be set to true, and reserved_for attribute will be set to the first_name of the user who reserved the monitor.

With all the requirements tackled, the next step in the software development cycle of Metropolia's helpdesk Chat Bot was the actual implementation. The model used to develop Metropolia's helpdesk Chat Bot is called Waterfall. Waterfall is a sequential model where each major step in the holistic process of creating software happens one after another, as opposed to iterative agile software development models where the final software is a result of numerous small iterations. The development of Metropolia's helpdesk Chat Bot served an academic showcasing purpose and its scope was limited and therefore the use of Waterfall was an advantage.

5 Chat Bot Implementation

5.1 Version Control System

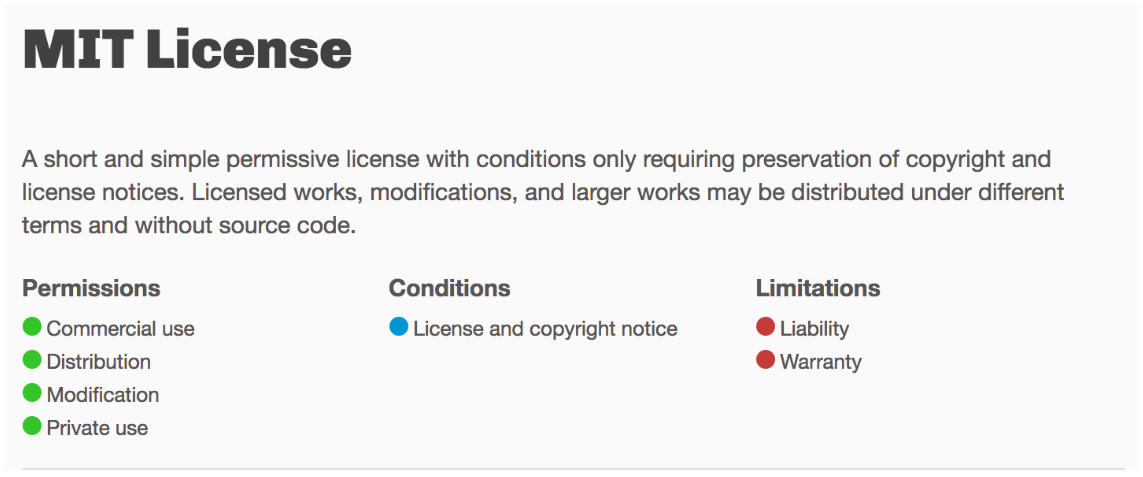
The first element in writing Metropolia's helpdesk Chat Bot was to choose a version control system. A version control system is a computer program designed to track changes that occur to computer files and to co-ordinate workflows for people working on those computer files. Git, a version control system created in 2005 by Linux Torvalds to support Linux kernel development, is nowadays the choice of an overwhelming majority of software developers [7]. Since I had around 2 years of experience with Git prior to the development of Metropolia's helpdesk Chat Bot, I decided to choose Git for my version control system.

5.2 Version Control System Hosting Service

GitHub is a popular Git version control system hosting service. GitHub, as of April 2017, with around 20 million registered users and around 60 million repositories, is the largest source code hosting service in the world [8]. Since GitHub as the largest source code hosting service offers issue tracking, feature-rich collaboration tools and other features not available in its competitors like BitBucket, I chose GitHub for my Git version control system hosting service.

5.3 Software License

There are plenty of licenses one can choose for their software projects, ranging from overly restricted ones like the 'All rights reserved' license to overly lenient licenses which allow for obtaining a copy and doing what one wishes with only mentioning the original author like Massachusetts Institute of Technology License and Apache License 2.0. Given the complexity of understanding technical terms and statements in texts related to law, I had to obtain help from a website called ChooseALicense.


MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions	Conditions	Limitations
● Commercial use	● License and copyright notice	● Liability
● Distribution		● Warranty
● Modification		
● Private use		

Figure 4. Illustration of Massachusetts Institute of Technology License [9].

As can be seen from Figure 4 permissions, conditions and limitations sections, the Massachusetts Institute of Technology license is described as a simple and permissive license which allows for distribution, modification and private use without holding the original author liable for possible damages. Since I wanted the software I was going to write to be accessible and distributable, based on the information I obtained from ChooseALicense website, I decided to choose the Massachusetts Institute of Technology license in the end.

5.4 GitHub Repository

For initializing any repository on GitHub, one needs:

- A repository name
- An optional description for the repository
- A decision whether to make the repository private or public
- A decision whether to add Readme to the repository
- A decision for the programming language of the gitignore file
- A decision for the license.

I chose telegram_bot_for_IT_sales for the repository name. For description of the repository, I added “A bot that lets you reserve and manage monitors for sale”. Since I wanted the repository to be accessible to everyone, I decided to make it public. To be able to demonstrate how Metropolia’s helpdesk Chat Bot works and introduce its basic usage, I decided to add a Readme file to the repository. gitignore is a file used by Git version control system which is in charge of configuring what Git tracks and what it ignores. For the repository, I decided to initialize the repository with a gitignore file aimed at Python projects. The reasoning behind choosing Python as the programming language will be discussed later in section 5.5. As described earlier, I chose Massachusetts Institute of Technology for the license.

After successfully initializing the repository, the next step was to clone the repository from GitHub onto my personal computer. Once the repository is cloned, I could start adding files to the repository and make changes, and once finished, push them back to the GitHub. With this approach, my changes are immune to be lost and I can revert any changes in the history of editing my files if need be, using Git features.

5.5 Choice of Programming Language

In software development, to prevent reinventing the wheel and to delegate and build upon what has already been built, software developers use libraries and Software Development Kits. Software Development Kits are practically a collection of tools and functions written to abstract away the underlying details of communicating with a platform’s Application Programming Interface. Telegram Chat Bot platform, as of April 2017, does not offer any official Software Development Kit and only exposes its Application Programming Interface for software developers to develop Chat Bots with.

In the absence of official Software Development Kit for the Telegram Chat Bot platform, and to save development time by using what has already been built, I had to search for unofficial libraries. After researching several libraries which offered unofficial support for the Telegram Chat Bot platform, I decided to use pyTelegramBotAPI. pyTelegramBotAPI is a Python third-party library which offers high-level support for the Telegram Chat Bot platform. Since pyTelegramBotAPI offered the necessary features I was looking for and provided just enough abstraction for both flexibility and ease of use, I decided to use Python to write the application logic with, which I had around 3 years of experience prior to development of Metropolia’s helpdesk Chat Bot.

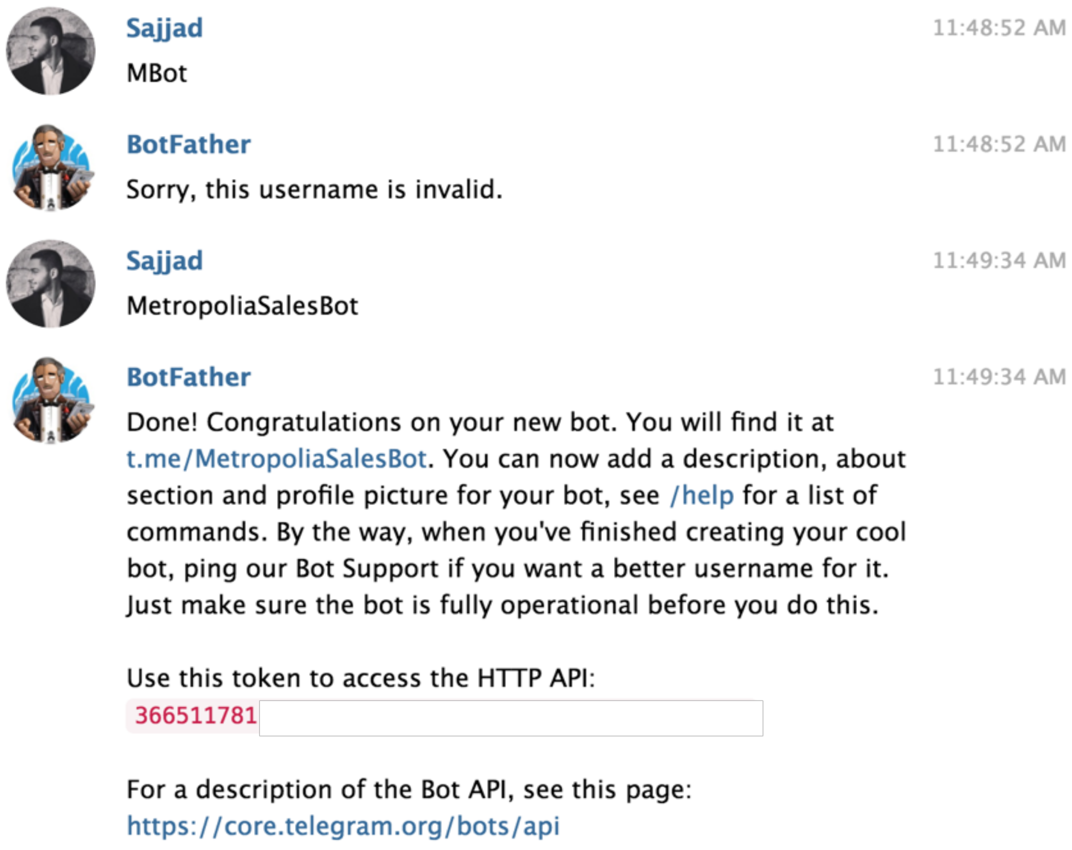
5.6 Development Environment

To initialize an environment in which I could start developing the application logic in, I had to create a development environment. In the Python ecosystem, it is common practice to use a computer program named virtualenv to create isolated development environments. After creating an isolated environment using virtualenv, one activates the environment and then installs the libraries within the isolated environment. The benefit of isolated environments is the fact that installed libraries within the global environment do not affect the state of the isolated environment and vice versa.

Therefore, to create an environment in which I could develop Metropolia’s helpdesk Chat Bot, I created an isolated environment using virtualenv and after activating it, I installed pyTelegramBotAPI library. Afterwards, I froze the libraries installed in the isolated environment onto a file named requirements.txt so that an exact environment could be produced elsewhere. Freeze is a technical term in the Python ecosystem coming from pip package manager and it practically means writing exact versions of libraries installed in the isolated environment to a text file so that the pip package manager could install the exact versions in a newly created environment later.

5.7 Chat Bot Creation

To create a Telegram Chat Bot, I had to use Telegram’s own Chat Bot named BotFather. BotFather Chat Bot is a complex Chat Bot designed to automatically create, edit, and remove Chat Bots for users.



Sajjad 11:48:52 AM
MBot

BotFather 11:48:52 AM
Sorry, this username is invalid.

Sajjad 11:49:34 AM
MetropoliaSalesBot

BotFather 11:49:34 AM
Done! Congratulations on your new bot. You will find it at t.me/MetropoliaSalesBot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
366511781[redacted]

For a description of the Bot API, see this page:
https://core.telegram.org/bots/api

Figure 5. Illustration of creating a new bot with Telegram’s BotFather Chat Bot

As can be seen from Figure 5, after communicating with BotFather and following its instructions, I created a Chat Bot named ITSales with username MetropoliaSalesBot. Each Chat Bot created by BotFather receives a token to be able to access Telegram’s Application Programming Interface exposed via Hypertext Transfer Protocol. A token could be thought of as both identity and password of one’s Chat Bot, and it should be treated accordingly.

5.8 Chat Bot Application Logic

With Metropolia's helpdesk Chat Bot token in possession, the next step was to write the application logic. In a Python module named bot.py, I imported telebot which is the installed name of pyTelegramBotAPI third-party library [10]. The imported telebot library has a TeleBot class which is used to instantiate an object representing one's Telegram Chat Bot. To instantiate Metropolia's helpdesk Chat Bot, I gave its token to the TeleBot class and I named the object 'bot'.

As can be seen from bot.py module [10], to prevent people who have access to Metropolia's helpdesk Chat Bot source code to obtain the Chat Bot token, the token is not hard coded into the bot.py module. It is common practice in web application development to store application-related critical information, like connection strings to the databases, secret codes, tokens, and so on in the server's environment variables. Therefore, I store Metropolia's helpdesk Chat Bot token in a variable named TELEGRAM_TOKEN in Heroku's environment variables, and in bot.py module, I wrote the code so that the bot object would be instantiated from the token that was previously stored in TELEGRAM_TOKEN environment variable. Heroku's role will be explained in section 5.9.

After instantiating the bot object which represents Metropolia's helpdesk Chat Bot, I started writing message handlers. A message handler is a function responsible for handling a certain kind of message sent by the user. For Metropolia's helpdesk Chat Bot, I declared the following message handlers:

- start_handler to handle /start command
- about_me handler to handle obtaining information about current user
- help_me handler to handle returning user to appropriate flow in any state
- Role handlers
    - student_handler to handle /student command
    - admin_handler to handle /admin command
- update_monitors_handler to handle appropriate actions based on the role of user
    - add_monitor_handler to handle adding a monitor to database
    - view_monitors_handler to handle viewing all monitors in the database
    - remove_monitor_handler to handle removing a monitor from database
    - reserve_monitor_handler to handle reserving a monitor for a user

As can be seen, holistically there are three types of message handlers in Metropolia's helpdesk Chat Bot. The first type is the start, help, and me commands which act as introductory and helper message handlers. The second type is role handlers which manage the user's current role. Finally, the third type is add-, view-, remove-, and reserve-monitor commands which are in charge of basic operations on monitors for both users and admins.

To separate the application core logic written in bot.py from other application-related logics, I wrote two other Python modules named db_clients and db_utils. db_clients module is in charge of abstracting away the details of creating database client objects which other modules can use after importing the clients. db_clients.py module achieves the abstraction by importing Redis library and Redis connection Uniform Resource Locators, previously stored in environment variables. Afterwards, two objects are instantiated with Redis connection Uniform Resource Locators named Users and Monitors respectively.

db_utils.py module is in charge of basic operations on users and monitors in the database, with the following functions:

- User related operations
    - save_user
    - get_user
- Monitor related operations
    - save_monitor
    - get_monitors
    - remove_monitor
    - reserve_monitor

Since db_utils.py is built upon db_clients.py, there is no need to instantiate database clients in db_utils.py anymore and all the functions defined in db_utils.py could be imported in bot.py to keep the application core logic at a minimum for better readability and modularity of the software.

Finally, the last Python module whose functions have been imported in bot.py is markup_utils. markup_utils module is in charge of abstracting away the underlying details of creating a two-, three-, and four-option customized reply keyboard which was discussed in section 3.2. In the bot.py module, it is enough to import markup_utils.py declared functions and pass the appropriate number of options to the corresponding function and the function will return a customized markup keyboard.

5.9 Deployment to a Hosting Service

The last step with the application logic ready was to deploy it to a hosting service. Heroku is a Platform-as-a-Service cloud hosting service which provides a high-level of abstraction of underlying details of setting up infrastructure to host one's service and the inevitable maintenance work that comes with it afterwards. Since Heroku focuses strongly on the security of its platform compared to its competitors like Amazon Web Services in which freedom of choice with services could lead to less security, I decided to choose the Heroku hosting service to deploy Metropolia's helpdesk Chat Bot source code to.

Heroku hosting service serves applications in Dynos. A Dyno is a smart and lightweight Linux container which is able to run a single command available in its Linux stack or in the source code available within the Dyno. In a typical web application, one has a web server which accepts incoming requests from the router and a background job executor. In Heroku hosting service environment it gets translated into a web process type for incoming requests and a worker process type for executing the background job. A Dyno could be of type Web, Worker, and several others which are out of the scope of this thesis. For Heroku hosting service to understand what type of Dyno to create for one's application and what command to run within those Dynos, a file named Procfile is needed. [11.]

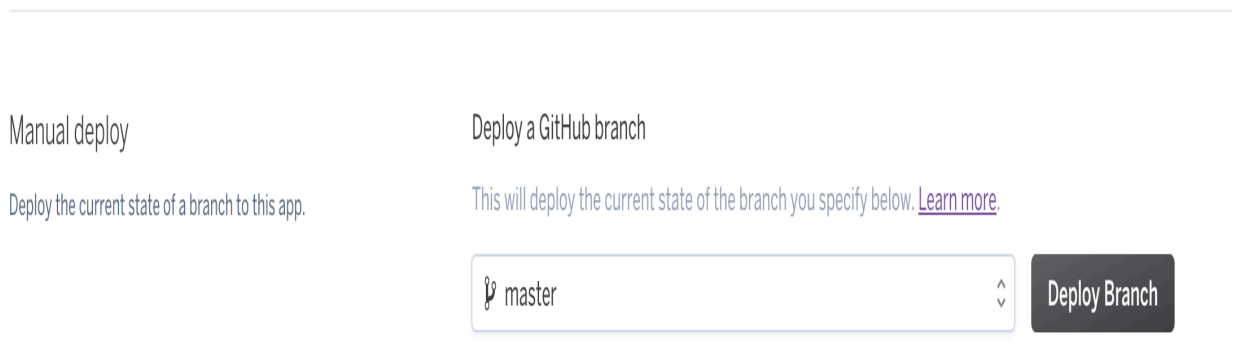
Procfile is a simple flat text file with a simple syntax to write. On each line, you specify the Dyno type followed by a colon and a space, and then the command to be executed. Since in Metropolia's helpdesk Chat Bot, I did not need to have a Dyno of the web process type to accept the incoming requests from Heroku routers, I declared only a Dyno of worker process type with a single-line command to run the bot.py module, as follows:

```
worker: python bot.py
```

The last line of code in bot.py module is a statement in which the polling function is called upon the bot object which is representing Metropolia’s helpdesk Chat Bot. The polling function called upon the bot object is a helper method that spawns a new thread which calls another internal function to retrieve incoming updates from Metropolia’s helpdesk Chat Bot and notify the appropriate message handler functions declared prior to call to polling function. Therefore, the underlying thread spawned with polling helper functions acts in a similar fashion to a Dyno of the web process type.

Finally, for Heroku’s hosting service to know what Python version to install in one’s Python application Dynos, one needs to create a file named runtime.txt. For Metropolia’s helpdesk Chat Bot, since I had written the application using Python version 3.5.2, I only needed to write ‘python-3.5.2’ in the runtime.txt file.

With the application logic ready and requirements for deployment to Heroku hosting service set, the next step was the actual deployment. By logging into Heroku’s website, creating an application which can host application Dynos, and connecting Heroku’s application to the GitHub repository, deployment is straightforward.


Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Figure 6. Illustration of deployment on Heroku hosting service

As can be seen from Figure 6, after selecting the Git branch from the GitHub repository of Metropolia’s helpdesk Chat Bot, deployment can simply be done by clicking on the ‘Deploy Branch’ button. The Heroku hosting service will automatically install the Python version defined in runtime.txt as well as all third-party libraries defined in requirements.txt and setup Dynos according to Procfile. After successful deployment, the implementation was complete.

6 Results and Discussion

The goal of this thesis was to build a Chat Bot prototypically to understand what it takes to develop Chat Bots, as well as to do a narrowed research into the potential growth of Chat Bots in the short-term future. In terms of development of the prototypical Chat Bot, the goal of the thesis is fully met. During development of Metropolia's helpdesk Chat Bot, however, it was found that due to lack of mature frameworks or libraries to do the boiler-plate coding required to set up a Chat Bot both alone and on multiple platforms, the development of complex Chat Bots could become complex quickly even with separation of concerns and modularisation that was done in section 3.3. It is, however, a step forward to see that the open-source community is aware of the complexity and difficulties of developing complex Chat Bots and software developers are taking measures to address the issue, like the Claudia Bot Builder example discussed in section 4.3.

In terms of potential growth of the Chat Bot concept, due to lack of complete analytics and unwillingness of companies to share about their users' usage model when it comes to Chat Bots, the goal of the thesis is not fully met. However, lack of statistics is understandable since it has only been a year from the time when major technological companies like Facebook, Google, Microsoft and Apple opened their respective Chat Bot platforms. There has been one common factor in financial, statistical and development aspects related to the potential growth of Chat Bots over time, and that is the constant upward growth tendency. As discussed in chapter 4, the amount of investment in Chat Bots is steadily increasing, the number of developers joining Chat Bot platforms are steadily increasing as well as the number of Chat Bots they create, and the development tools are moving towards maturity and reliability.

To be able to provide critical evaluation of the results and this project as a whole, I need to explain the foundations upon which this thesis is written on. Firstly, as of April 2017, I have worked professionally as a software developer for two years, and therefore the technical complexity prediction for developing complex Chat Bots is a reflection of my current expertise and the judgement that is based on that expertise. Secondly, as with any other new phenomenon happening in the Information Technology world, it is challenging to predict how users will respond to the phenomenon and how it will shape our future, if at all. Chat Bots being only a year old at the time of writing this thesis are not an exception to this rule and any statement regarding the potential growth of Chat Bots is an effort to understand current information and have a better reasoning behind the approach one would like to take towards Chat Bots.

There are two parts about statements provided in this chapter. The first part is about the development perspective of Chat Bots. Given the immaturity of the libraries and frameworks available to create Chat Bots at the moment, attempting to develop a complex Chat Bot with the current available tools is a decision which needs to be taken carefully. However, tools for development of Chat Bots seem to be improving and the immaturity will be decreased over time when more software developers contribute to the development of tools and the community around Chat Bots.

The second part is about potential growth. With increasing funding amounts, more software developers joining Chat Bot creation platforms and thus creating more Chat Bots, and with the tools which seem to be improving over time, it can be concluded that Chat Bots are here to stay, at least for the foreseeable future. The extent to which they will change how businesses interact with customers is yet unknown, but their existence in the near future is an insight which can be concluded from the information collected in Chapter 4.

7 Conclusions

The goal of this project was to develop a prototypical Chat Bot for Metropolia University of Applied Sciences helpdesk to better understand Chat Bot creation and to provide insight into whether Chat Bots are a lasting phenomenon. Over the course of this thesis, it was concluded that even though the available tools for Chat Bot creation are limited and immature at the moment, it is indeed possible to develop complex Chat Bots and the available tools seem to be improving over time.

Furthermore, based on the amount of investments in the concept of Chat Bots, and the number of software developers who join Chat Bot creation platforms and create Chat Bots consequently, and the community behind the available tools for Chat Bot creation, it is concluded that Chat Bots are a lasting phenomenon, at least for the foreseeable future. The insight which argues that Chat Bots are a lasting phenomenon is only a conclusion based on the current information at hand and a best-effort prediction of the short-term future, and therefore it should be treated accordingly.

Acknowledging that Chat Bots are here to stay is an insight of great importance. Given the number of users already active in messaging platforms, which are estimated to be more than one billion only on the Facebook Messenger platform alone, only proves the significance of opportunities for business owners, software developers and users in general.

The strength of this project was the opportunity to be among the first analysists of the Chat Bot concept. However, Chat Bots being new proved to provide certain obstacles especially in terms of analytics and user engagement, which has to be thought of as the limitation of this project.

For further study of the concept of Chat Bots, I recommend following closely the latest happenings in the Chat Bot community, using Chat Bot magazines and content distributors such as ChatBotMagazine.

References

1 Shachtman Noah (19 September 2002). A Stab at Celebrity Glamour Fails to Save TV's BattleBots [online]. New York Times. Accessed 10 March 2011.

2 Mauldin, Michael (1994), ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition [online]. Proceedings of the Eleventh National Conference on Artificial Intelligence, AAAI Press, Accessed 5 March 2008

3 Weizenbaum, Joseph (January 1966), ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine, Communications of the ACM, 9 (1): 36–45.

4 Be Featured In Front Of Thousands Of People Interested In Bots [online]. Chatbotsmagazine; URL: https://chatbotsmagazine.com/be-featured-in-front-of-thousands-of-people-interested-in-bots-e7040c4080df Accessed 8 April 2017.

5 Seth Rosenberg. How To Build Bots for Messenger (12 April 2016) [online]. Facebook; URL: https://developers.facebook.com/blog/post/2016/04/12/bots-for-messenger/ Accessed 8 April 2017.

6 Claudia-bot-builder [online]. Claudiajs; URL: https://github.com/claudiajs/claudia-bot-builder Accessed 8 April 2017.

7 Developer Survey Results 2017 [online]. StackOverflow; URL: http://stackoverflow.com/insights/survey/2017 Accessed 14 April 2017.

8 Georgios Gousios; Bogdan Vasilescu; Alexander Serebrenik; Andy Zaidman. Lean GHTorrent: GitHub Data on Demand. The Netherlands: Delft University of Technology & †Eindhoven University of Technology: 1. Retrieved 9 July 2014.

9 MIT License in simple terms [online]. ChooseALicense; URL: https://choosealicense.com/licenses/mit/ Accessed 14 April 2017.

10 Sajjad Hosseini. Metropolia's helpdesk Chat Bot [online]. URL: https://github.com/Sajjadhosn/telegram_bot_for_IT_sales Accessed 14 April 2017.

11 Dynos and the Dyno Manager [online]. URL: https://devcenter.heroku.com/articles/dynos Accessed 16 April 2017.

12 Trac Xn. Report: Top Funded Startups, Business Models, And The Most Active Investors In Chatbots. Tracxn Startup Research; URL: https://blog.tracxn.com/2016/06/16/top-funded-startups-business-models-and-the-most-active-investors-in-chatbots/ Accessed 16 April 2017.

13 Jim Marous. Financial Institutions Bullish on Chatbots. thefinancialbrand; URL: https://thefinancialbrand.com/63596/financial-banking-bots-chatbot-voice-ai/ Accessed 16 April 2017.

14 Josh Constine. Facebook Messenger now allows payments in its 30,000 chat bots. techcrunch; URL: https://techcrunch.com/2016/09/12/messenger-bot-payments/?ncid=rss Accessed 16 April 2017.

15 Alex Debecker, Ubisend. 3 stats that show chatbots are here to stay. venturebeat; URL: https://venturebeat.com/2016/08/26/3-stats-that-show-chatbots-are-here-to-stay/ Accessed 16 April 2017.