Course Assignment

Master of

Applied Computer Science

Department of Technology

Assignment title	Chatbot for Kristiania University College
Course code	MS230
Course name	Interactive Technologies
Due date	24.04.2020

Declaration:

Through the submission of this assignment, I hereby declare that this report is the result of my own work, and that all sources have been properly cited to throughout the text.

Candidate number	105

Contents

Chatbot for Kristiania University College

1 Introduction

Chatbots have the potential to change how we interact with data and online services. Microsoft CEO Marco Della Cava said in 2016 that "Bots are the new apps" [1]. In the last 4-5 years, we have seen a lot of development in the chatbot market with services like Microsoft Bot Framework, Dialogflow (Google), Wit.ai (Facebook), and IBM Watson Assistant. We have also seen a boom for virtual assistants with Siri (Apple), Cortana (Microsoft), Alexa (Amazon), and Google's Assistant.

For this exam, I have developed a chatbot solution for Kristiania University College, which will help students easily find the information they need by just asking.

The paper is structured as follows. In Section 2, I give an overview of the concept and history of chatbots. Here I also talk about the most popular bot frameworks we have today and challenges/aspects of developing chatbots. In Section 3, I talk about my idea, how I did my usability test, and how I extracted requirements from it. In Section 4, I go through the different aspects of how I implemented the chatbot. In Section 5, I discuss how it was developing a chatbot and what I would like to do differently. In Section 6 I conclude how the project went.

2 Related Work

In this section, I go through the history of chatbots and what the concept of a chatbot is. I also give information on the most popular chatbot frameworks we have today. Lastly, I talk about the aspects and challenges of developing chatbots.

2.1 Chatbot concept and history

The first conceptualization of the chatbots we have today is credited to Alan Turing; he asked the question "Can machines think?" in his paper "Computing Machinery and Intelligence" from 1950 [2]. In the paper, he proposed the concept of the Turing Test, which he called "The Imitation Game", it would test to see if a computer is capable of thinking like a human being. In figure 1, you can see the traditional interpretation of the Turing Test. The test involves three participants: a machine, a human, and a judge. The judge would talk to either of them and decide if he is talking to the machine or the human at the moment. The test is only successfully passed if the machine is mistaken for a human more than 30% of the time during a 5-minute interrogation [4].

One of the first chatterbots who was able to attempt the Turing Test was ELIZA who was developed in 1996 by Joseph Weizenbaumat the MIT Artificial Intelligence Laboratory [5]. It was an early natural language processing program that simulated a conversation by using "patter matching." The program read the text and looked for keywords. If the program found a keyword, it would then transform the answer according to a predefined rule. In other words, it recognizes words in the input, and the output is a predefined sentence corresponding to the keyword.

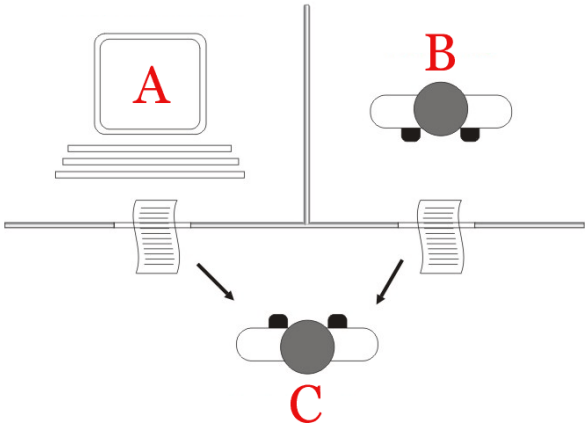

Figure 1. Interpretation of the Turing Test [3]

Since ELIZA there has been progress in the field of chatbots; they became increasingly more intelligent and sophisticated. Kenneth Colby at Stanford in 1972 created a bot named PERRY, who impersonated a paranoid schizophrenic [5]. PERRY and ELIZA met at a conference in 1973 and was the first conversation between two chatbots [5, 6].

In the 1980's, computer scientists started using new technology to power chatbots. It was a new type of AI technology, which we today call machine learning; this enabled the bots to learn from experience. So the chatbot became better the more conversations it had [7]. On of the first machine learning chatbots were Jabberwacky developed by Rollo Carpenter in 1998 [7]. It would later go on to win the Loebner Prize in 2005 and 2006. The Loebner Prize is a Turing Test competition, which was started in 1991 by Hugh Loebner [8].

The first time the term "ChatterBot" was mentioned in 1994 by Michael Mauldin, it was used to describe conversational programs. This was done at the twelfth national conference on artificial intelligence; since then, the term has been actively in use.

One year later, one of the first online chatbots was debuted by programmer Richard Wallace. This made it possible for people to interact with a chatbot over the internet. The bot was called ALICE (Artificial Linguistic Internet Computer Entity), but it did not use machine learning [7]. Like ELIZA and PERRY, ALICE only used pattern-matching against a static database, but it still won the Leobner Prize in 2000, 2001, 2004.

The rise of the internet and messaging allowed the chatbot actually to become useful for the general public. One of the first chatbots that were not only for entertainment but tried to provide the user with useful information was SmarterChild, which was developed by ActiveBuddy [9]. It was available for AOL and Windows Live Messenger, with 30 million users

[9]. Many call SmarterChild the precursor to Siri by Apple [7, 9]. Later other companies followed in their footsteps and developed their own AI assistants.

In 2019, the chatbot market was valued at USD 17.17 billion and was projected to reach USD 102.29 billion by 2025 [10].

2.2 Bot Frameworks

A bot framework is a foundation of what many bots are built upon. It comes with preexisting functionality that helps bot developers write better code and produce working software faster. Many of the most well-known bot frameworks also help with integrating the bots into different platforms and channels, like Slack, Twitter, Messenger, Skype, etc.

Some of the most popular bot frameworks today are Dialogflow (Google) [11], Wit.ai [12], IBM Watson Assistant [13] and Microsoft Bot Framework [14].

Dialogflow

Dialogflow is an AI-powered chatbot development framework from Google; it helps you design and build both chatbots and voice apps for Amazon Alexa Skills and Google Actions. It is the most popular tool for building Actions for more than 1 billion devices [15]. It supports all the major messaging platforms and has Natural Language Processing(NPL) for over 20 languages.

Wit.ai

Wit.ai is an NPL platform that makes it possible for developers to configure intents and entities [16]; it is a Facebook owned company. It also has an HTTP API, which enables developers to connect it to a chatbot or other applications.

IBM Watson Assistant

IBM Watson Assistant is a framework that lets you create applications that understand natural-language and responds to the customer in human-like conversation [17]. It supports 13 languages.

Microsoft Bot Framework

Microsoft Bot Framework is an SDK like the other frameworks; it provides developers with the tools to build intelligent chatbots. It houses three products: Bot Builder SDK, Bot Framework Portal, and channels [18]. The SDK is what is actually used for building the chatbot. The portal is used to register the bot and manage it, hosting tools, and analytics. The framework also provides an easy and unified way of integrating the bot in different channels like Slack, Messenger, Skype, etc.

This project was implemented by using the Microsoft Bot Framework; this is because it was introduced to us in class and because it had some interesting tools, I wanted the explore like QnA Maker [19] and LUIS [20].

2.3 Development aspects and challenges

The number one challenge for chatbot developers today is how many messaging platforms there is. An important aspect that developers needs to keep in mind when developing chatbots is that the user expect the same consistent interaction with the bot on every platform. In figure 2, you can see that all

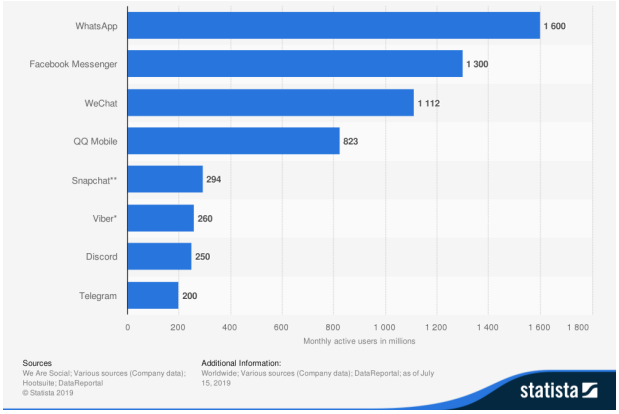

Figure 2. Most popular global mobile messenger apps as of October 2019 [21].

users are spread across a few messaging apps. All of these platforms are different both when it comes to technology, types of communication, and who the audience is. All of these aspects provide different challenges for developers. This is also why we have seen a lot of development been put into bot frameworks, which will make it easier for developers.

Another challenge for developers and businesses is to understand why people use chatbots; the reason this is important is that this can help developers and businesses develop better solutions that people want to use. Not a lot of research has been done on this topic. In [22], they did a study with 146 participants were they tried to get a better understanding. 68% of the participants reported productivity as the main reason for using a chatbot. Other reasons were entertainment, social/relational, and novelty/curiosity.

3 Scenario and idea description

When deciding what kind of chatbot I would like to create, I wanted something that I knew could be useful for myself and other students at Kristiania University College. I know from my own experiences that it can be a pain to dig around a website to find the information you need. This was when I came up with the idea for a chatbot that could help students find this information by just asking the bot. This bot could be deployed in different channels like Messenger, Slack, and even at Kristiania's own website.

When extracting the requirements for my chatbot, I did a short usability study of an already existing chatbot solution. The solution I decided to do the usability study of was DNB's chatbot. DNB's chatbot is there to help both private persons and companies. For doing the usability test, I used something called the System Usability Scale (SUS), which is a tool for measuring the usability of a system. The test consists of a 10 item questionnaire with 5 response options ranging from

strongly agree to strongly disagree [23]. With the current situation finding participants was not easy, but I used my family and me.

The scoring system for SUS can be a little complicated at first, but you get the hang of it after some studying. Here is how the scoring works: "To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7, and 9, the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU." [23]. In the end, the score will be between 0 to 100. In figure 3, you can see the categories the different scores fall under. The average SUS score is 68, so if you are under 68, it could point to issues with the design, but if you are over 68, it indicates that just minor improvements are needed. With so few participants I had in the study, it will not give the full representation of the general public, but we have different genders with different ages. The scores for DNB's

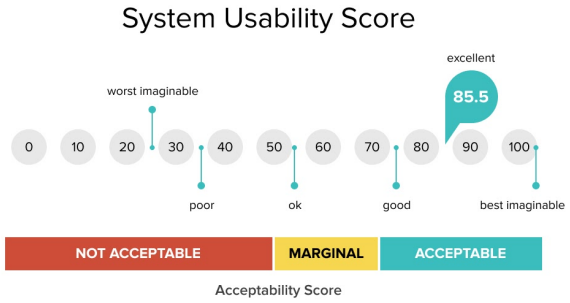

Figure 3. Acceptability Score [24].

chatbot was 60, 65, 80, and 85. Me and my sister, who are under age 25 scored the highest, and my parents who are over age 45 scored the lowest. So two scores are over the average, and two scores are under the average. The average of all four scores is 72.5, which indicates that DNB's chatbot has good usability and only need minor adjustments. All the participant surveys can be found in the appendix.

Testing DNB's chatbot helped me extract requirements for my own projects. Examples of this are how to give information to the user, what kind of language to use, how to use buttons etc.

4 Implementation

In this section, I go in-depth about the development of the bot, what kind of tools I used, hosting, testing, and deployment.

4.1 Development

The development framework I decided to go with was the Microsoft Bot Framework(MBF). There are multiple languages you can develop in when using MBF; the three main languages are C#, Python, and Javascript/Typescript. I decided to go the Javascript/Typescript route since that is what we used in class, and I also have experience with the language from previous projects.

Setup

When setting up a project for MBF, it is several ways you can do this.

- You can use the package generator-botbuilder, which is a Yeoman generator for Bot Framework v4. It will let you quickly set up a chatbot with AI capabilities [25].
- You can set up a bot in the Azure portal and download the code that is generated for you. One of the benefits of this is that you get the deployment scripts that you can use for deploying to Azure later.
- You can also set up the project from scratch, which is what is decided to do; for this, you have to initialize npm and setup the packages you need.

I choose to set up the project from scratch to better understand the code I was working with. Setting up the project from scratch requires you to initialize npm yourself and install the packages you need. Some of the essential packages when using the MBF are:

- **botbuilder:** "A framework for constructing bots that can handle both freeform interactions and more guided ones where the possibilities are explicitly shown to the user." [26]
- **resitfy:** "A framework, utilizing connect style middleware for building REST APIs." [27]

When you have these packages installed, you can start building your bot. You start by creating an index.js file that will be the entry point for your application and import the packages.

```
const restify = require('restify');
const { BotFrameworkAdapter } = require('botbuilder');
```

You then create a bot adapter, which defines how the bot sends and receives messages.

```
const adapter = new BotFrameworkAdapter({
  appId: process.env.MicrosoftAppId,
  appPassword: process.env.MicrosoftAppPassword,
});
```

Then you create an HTTP server that you can connect to locally. This is were restify is used.

```
let server = restify.createServer();

server.listen(process.env.port || process.env.PORT ||
    3978, function () {
    console.log('\n${server.name} listening to ${server.
        url}');
});
```

The last step is to set up the code that listens for incoming messages at /api/messages.

```
server.post('/api/messages', (req, res) => {
    // Use the adapter to process the incoming web
        request into a TurnContext object.
```

```
    adapter.processActivity(req, res, async (turnContext)
          => {
        // Do something with this incoming activity!
        if (turnContext.activity.type === 'message') {
            // Get the user's text
            const utterance = turnContext.activity.text;

            // send a reply
            await turnContext.sendActivity(`I heard you
                say ${ utterance }`);
        }
    });
});
```

This way of doing it is the example used for the botbuilder npm package [26]. This is a bare minimum bot that just repeats what the user is typing in chat. In figure 4, you can see the result.

Figure 4. Bot conversation.

4.2 Architecture and code

It is not enough to have a bot that just echos your message back to you. You need something that can be useful for the user. Before i started coding i looked at what services could fit my needs, the two services that stood out was QnA Maker and LUIS.

LUIS is the name for the Language Understanding cloud-based API service that Azure offers. It applies machine-learning to a the conversation to predict the overall meaning and pull out the right information [28]. LUIS has something called utterances, intents and entities. Utterances are the input from the user, an intent is the desired outcome of the whole utterance and entities are pieces of data from the utterance. Intents are often connected to actions in the application, while entities are data needed to perform the action [29]. To give an example, the utterance "Buy me a train ticket from Oslo to Bergen" have a single intent which is to buy a train ticket with the two entities Oslo and Bergen.

QnA Maker is a cloud-based Natural Language Processing (NLP) service [30]. What it does is that it adds a layer over your data, so that the user can converse with it. The data is stored in something called a knowledge base (KB). The KB consists of question and answer sets. You can import data into the KB from your website, files or just create the sets directly in the QnA Maker portal. Importing from a website can be especially useful when you have a FAQ section with questions and answers you would like the bot to know. QnA Maker also has a feature named multi-turn prompts which enable developers to connect question and answer pairs [31]. The most common use cases for this is for when a question can't be answered in a single turn. You can see an example of this in figure 5. In the figure a student asks for help with booking a room, the bot answers with the instructions but there is also a button at the bottom of the answer. The button is a follow up question which will tell the student the rules of room booking. When the button gets clicked the bot will automatically type the question for the user and answer it.

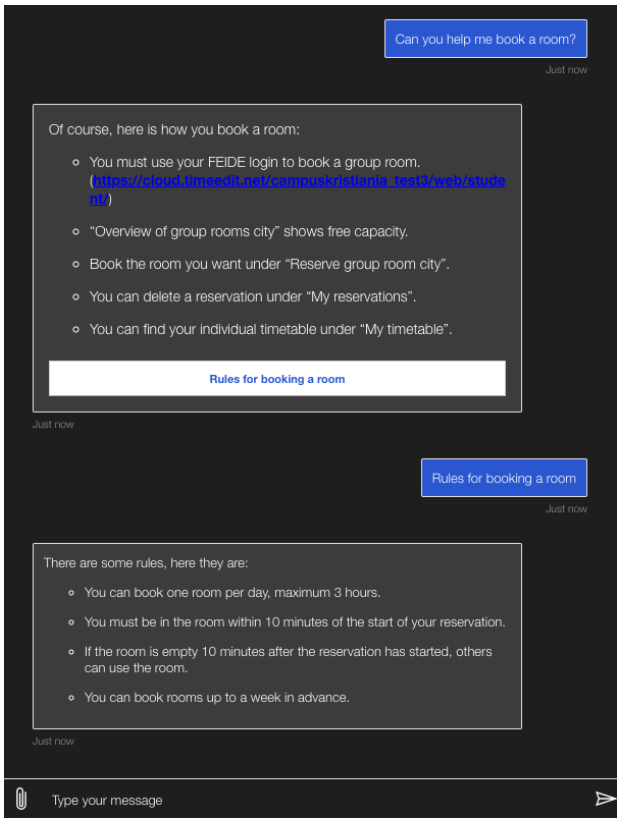

Figure 5. A multi-turn prompt

Both of the alternatives would in theory work, but QnA-Maker seemed like a better fit for me. Microsoft has some guidelines for when you should use QnAMaker [30], here are the bullet points:

- When you have static information in your KB.
- When you want to provide the same answer to a request, question, or command.
- When you want to filter static information based on meta-information.
- When you want to manage a bot conversation that includes static information.

The information I wanted to provide was similar to the type of information they recommended for QnAMaker in the guidelines. So this is what I ended up using for my project.

In figure 6, you can see the architecture of a QnA Maker bot. Here is how it works:

1. QnA Maker extracts questions and answers from files or URL. You can also create your own questions and answers in the QnA Maker portal.

2. They get indexed and ranked.
3. The KB gets published, and the API endpoint for the KB is created. The endpoint can then be used to extract information from the KB.
4. You connect your KB to the Microsoft Bot Framework, this was the connection between the bot and the KB get created.
5. The bot is then deployed to the channels you want your bot to be available from, like Messenger, Skype, Slack, etc.

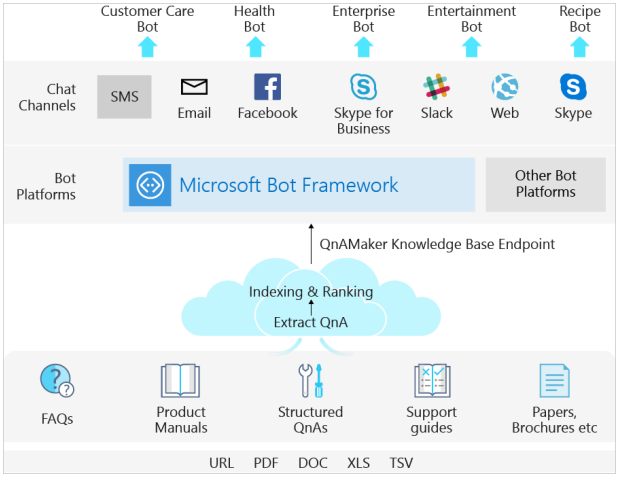

Figure 6. QnA Maker architecture [32].

In figure 7, you can see the architecture for how my QnABot code works. You have a .js file called QnABot. In these files you have the onMessage method:

```
this.onMessage(async (context, next) => {

}
```

The method is called when user input is received. It then accesses your connector using the values from the .env file. The method getAnswers is then called:

```
const qnaResults = await this.qnaMaker.getAnswers(context
    );
```

The method connects to the KB and if an answer is returned it will be displayed, if not you will get a message saying that no answers were found.

For the different cards the chatbot displays, I have used a tool from Microsoft called Adaptive Cards. You use JSON to design the cards, an example of this is my Welcome Card you can see in figure 8.

4.3 Bot Framework Emulator

The Bot Framework Emulator is what I used to test and debug the chatbot. With the emulator, you can test the bot either locally on your machine or connect to a bot remotely [34]. How it works is that the emulator connects to the endpoint URL you

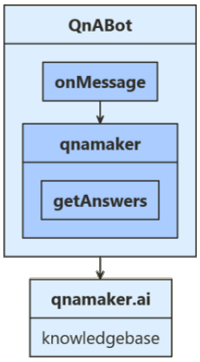

Figure 7. QnABot Architecture [33].

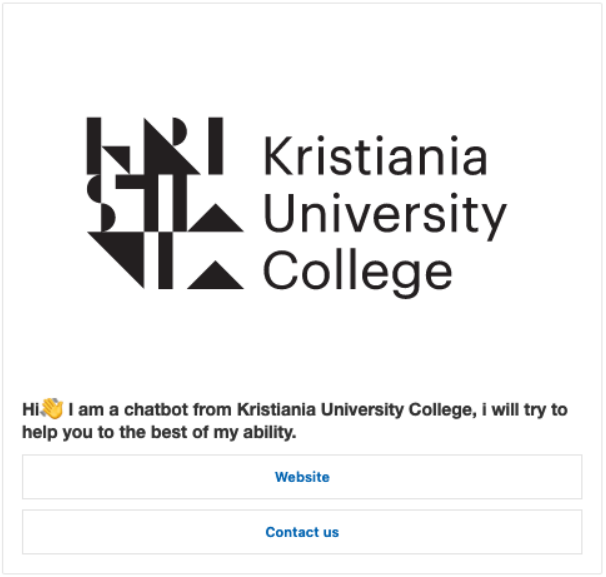

Figure 8. Adaptive Welcome Card

specified in your code, so for the example we used previously, the URL would look like http://localhost:3978/api/messages. In figure 9, you can see how the emulator view looks when you use it. The view is separated into three parts:

1. You have the dialog window on the left side, this is where you converse with the bot. It will show the inputs and the outputs of the conversation.
2. You have a log view on the lower right side. This view contains information about API requests and responses.
3. You have a JSON object view on the upper right side. This makes it possible for the developer to see the JSON data for each message.

4.4 Testing

When testing my KB, I used the testing tool in the QnA Maker portal, here you can interact with the KB and ask questions. In figure 10 you can see how the testing window looks. The most useful tool here is the inspect button. When you click

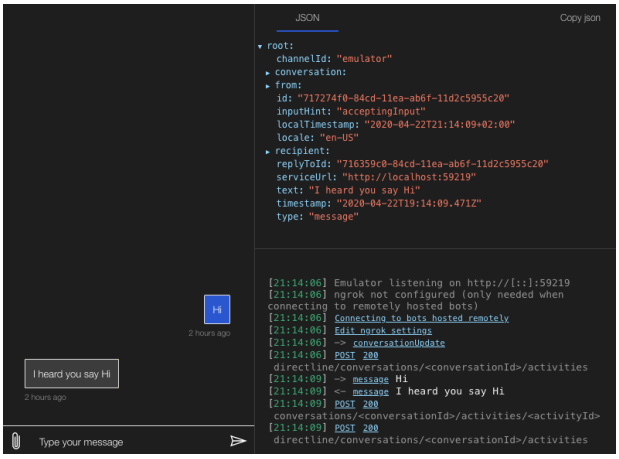

Figure 9. Bot Framework Emulator.

the inspect button, you can inspect the score for the answer given; this score is called the confidence score. This score reflects how confident the KB is about the answer it has given. If the top answer is wrong, you can correct it and use the save and train button to teach the KB. In the inspect window, you can also give alternative phrasings for the question, which helps the KB to be more accurate.

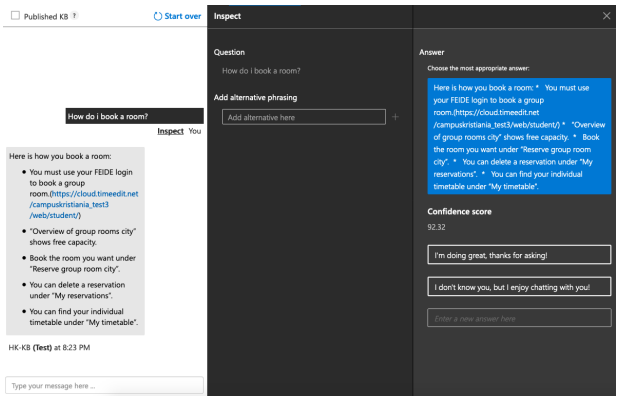

Figure 10. Testing KB in QnA portal

Figure 11. Save and train button

4.5 Hosting service

When you want to make your bot accessible for different channels, you need to host the bot on a web service. There are several of these services which provide hosting. The most popular services are Amazon Web Services (AWS), Google Cloud Platform, and Microsoft Azure.

For my project, I choose to go with Azure. Since both Azure and the Bot Framework is developed by Microsoft, it made for a seamless integration. Azure was announced in 2008 at the Microsoft Professional Developers Conference and offered services in several categories such as computing, storage, networking, and several others [35]. In figure 12, you can see an overview of some of the services Azure offers today. One of the services Azure offers is Azure Bot Service

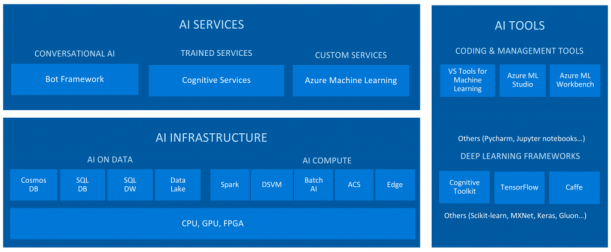

Figure 12. Azure Services [36]

which was made generally available in 2017 together with their Cognitive Language Understanding service known as LUIS [37]. At the time over 240,000 developers had signed up for the service. In figure 13 you can see how the Azure Bot Service uses the Cognitive Services to create a conversational AI. The channels in the figure is all the different messaging platforms you can integrate the bot with like Messenger, Slack, Skype, etc.

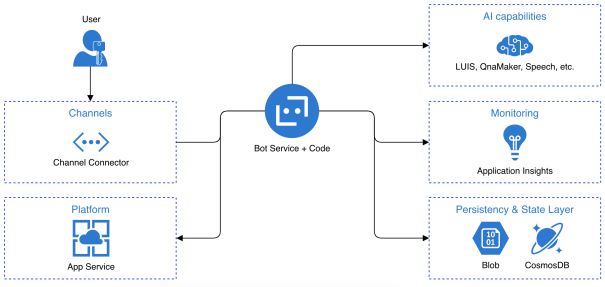

Figure 13. Azure Bot Service + Cognitive Services [38].

4.6 Deployment

When deploying the bot, I did that trough the Azure Portal. They have different subscription plans, but they have a free option that I used. I start by creating a Web App Bot; I then headed the deployment center where you have several options for deploying your bot. I decided to go with the GitHub option, which you can see in figure 14. When using GitHub, you use something called GitHub Actions. This allows for continuous deployment; how this works in practice is that every change pushed to the GitHub repository will automatically be deployed if the build succeeds. When the bot is deployed, you can connect it to different channels. In figure 15, you can see an overview of the different channels.

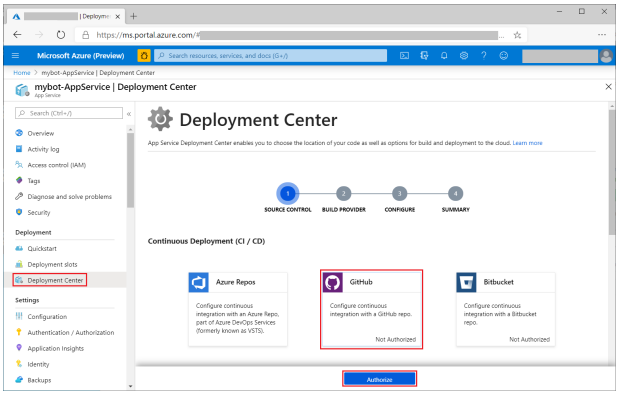

Figure 14. Deploying to Azure with GitHub [39].

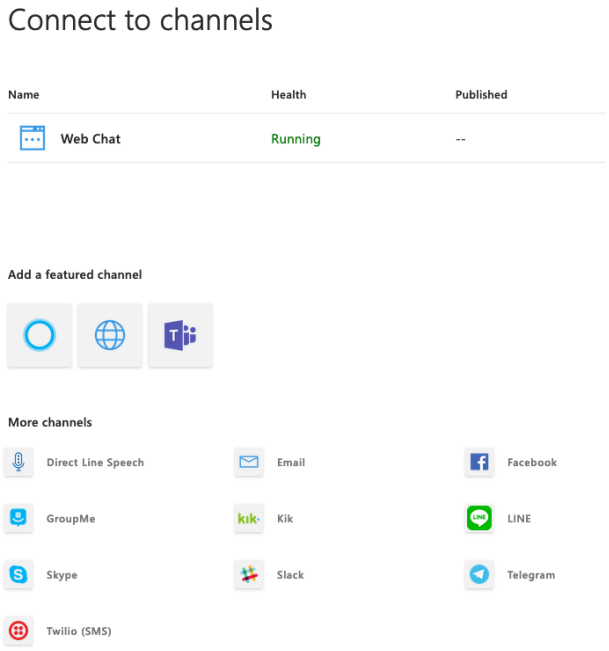

Figure 15. Connecting to channels

5 Discussion

What I wanted to do for the project was to create a chatbot for students at Kristiania University College. The results were a fully functioning prototype that uses QnA Maker to provide the answers. I am satisfied with the result, but for future work, I would like to expand the functionality of the chatbot significantly. I would like for the chatbot to be able to automate several processes like room booking, renting books, ordering a new student card, etc. This would require more access to the school systems, maybe even building some of the systems from the ground up if they are not possible to integrate with.

Because of the current situation, I did not have the possibility to do a usability study with more than 4 participants; for future work, I would like to do a bigger study to get a better understanding of chatbot usability and design. The authors Brandtzaeg and Følstad have done a great study on exactly this, and it helped me tremendously when designing my bot [22]. In the paper, they conclude that "chatbot designers should focus on designing and developing chatbots that are perceived as useful because they provide necessary help or information effectively and efficiently." [22]. This is something I agree with, and I have tried to do so with the bot I have developed. They also find out that even a productivity chatbot like mine can benefit from a friendly and empathic appearance. They use the example of Google's Api.ai, which includes things like chitchat. It is important to find a balance.

6 Conclusion

We see that the market for chatbots is growing rapidly and will have an essential role in how businesses interact with their customers in the future. For this project, I have created a chatbot for Krisitania University College that will help students find the information they need. For the project, we explored the Microsoft Bot Framework and several of the tools the Azure platform had to offer.

References

1. "Microsoft CEO Nadella: 'Bots are the new apps'," https://www.usatoday.com/story/tech/news/2016/03/30/microsof-ceo-nadella-bots-new-apps/82431672/.
2. A. M. Turing, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.
3. "Turing test," *Wikipedia*, Apr. 2020, page Version ID: 952031302.
4. O. Aishah, "What's the Turing Test and Which AI Passes It?" Jan. 2017.
5. J. Weizenbaum, "ELIZA—a computer program for the study of natural language communication between man and machine," p. 10.
6. V. Cerf, "PARRY encounters the DOCTOR," RFC Editor, Tech. Rep. RFC0439, Jan. 1973.
7. D. CANCEL, *CONVERSATIONAL MARKETING & SALES: How to Grow Leads, Shorten Sales Cycles, and Improve Your ... Customers' Experience with Real-Time Conversations.* Place of publication not identified: JOHN WILEY & Sons, 2019, oCLC: 1043199720.
8. M. L. Mauldin, "1994-ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition," p. 6.
9. R. Khan and A. Das, *Build Better Chatbots*. Berkeley, CA: Apress, 2018.
10. "Chatbot Market | Growth, Trends, and Forecast (2020 - 2025)," https://www.mordorintelligence.com/industry-reports/chatbot-market.
11. "Dialogflow," https://dialogflow.com/.

12. "Wit.ai," https://wit.ai/.

13. IBM, "Watson Assistant | IBM Cloud," https://www.ibm.com/cloud/watson-assistant/, Oct. 2017.

14. "Microsoft Bot Framework," https://dev.botframework.com/.

15. "Google says Assistant will be on a billion devices by the end of the month."

16. "17 Chatbot Development Frameworks and Chatbot Platforms [2020]," https://www.cedextech.com/blog/chatbot-development-frameworks.

17. "Watson Assistant - Overview - Norway," https://www.ibm.com/no-en/marketplace/watson-assistant, Apr. 2020.

18. S. Janarthanam, *Hands-on Chatbots and Conversational UI Development: Build Chatbots and Voice User Interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills*. Birmingham Mumbai: Packt, 2017, oCLC: 1037013936.

19. "QnA Maker," https://www.qnamaker.ai/.

20. "LUIS (Language Understanding) – Cognitive Services – Microsoft Azure," https://www.luis.ai/home.

21. "Most popular messaging apps 2019," https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.

22. P. B. Brandtzaeg and A. Følstad, "Why People Use Chatbots," in *Internet Science*, I. Kompatsiaris, J. Cave, A. Satsiou, G. Carle, A. Passani, E. Kontopoulos, S. Diplaris, and D. McMillan, Eds. Cham: Springer International Publishing, 2017, vol. 10673, pp. 377–392.

23. J. Brooke, "SUS - A quick and dirty usability scale," p. 7.

24. "The System Usability Scale & How it's Used in UX | Adobe XD Ideas."

25. "Generator-botbuilder," https://www.npmjs.com/package/generator-botbuilder.

26. "Botbuilder," https://www.npmjs.com/package/botbuilder.

27. "Restify," https://www.npmjs.com/package/restify.

28. diberry, "What is Language Understanding (LUIS)? - Azure Cognitive Services," https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis.

29. "Design with models - LUIS - Azure Cognitive Services | Microsoft Docs," https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-model.

30. diberry, "What is QnA Maker service? - Azure Cognitive Services," https://docs.microsoft.com/en-us/azure/cognitive-services/qnamaker/overview/overview.

31. ——, "Multi-turn conversations - QnA Maker - Azure Cognitive Services," https://docs.microsoft.com/en-us/azure/cognitive-services/qnamaker/how-to/multiturn-conversation.

32. "Build a FAQ chatbot with QnA Maker and Microsoft Bot Framework," Nov. 2019.

33. ivorb, "Use QnA Maker to answer questions - Bot Service - Bot Service," https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-howto-qna.

34. "Microsoft/BotFramework-Emulator," Microsoft, Apr. 2020.

35. "What is Microsoft Azure?" https://www.datamation.com/cloud-computing/microsoft-azure.html.

36. "Microsoft Artificial Intelligence: A platform for all information worker skill set levels - US Partner Community Blog - Microsoft," https://www.microsoft.com/en-us/us-partner-blog/2018/05/01/microsoft-artificial-intelligence-a-platform-for-all-information-worker-skill-set-levels/, May 2018.

37. "Announcing the General Availability of Azure Bot Service and Language Understanding, enabling developers to build better conversational bots," https://azure.microsoft.com/en-us/blog/announcing-the-general-availability-of-azure-bot-service-and-language-understanding-enabling-developers-to-build-better-conversational-bots/.

38. C. Siebler, "Microsoft Bot Framework v4 explained (JavaScript)," Jan. 2019.

39. ivorb, "Configure continuous deployment for Bot Service - Bot Service - Bot Service," https://docs.microsoft.com/en-us/azure/bot-service/bot-service-build-continuous-deployment.

A System Usability Scale Results

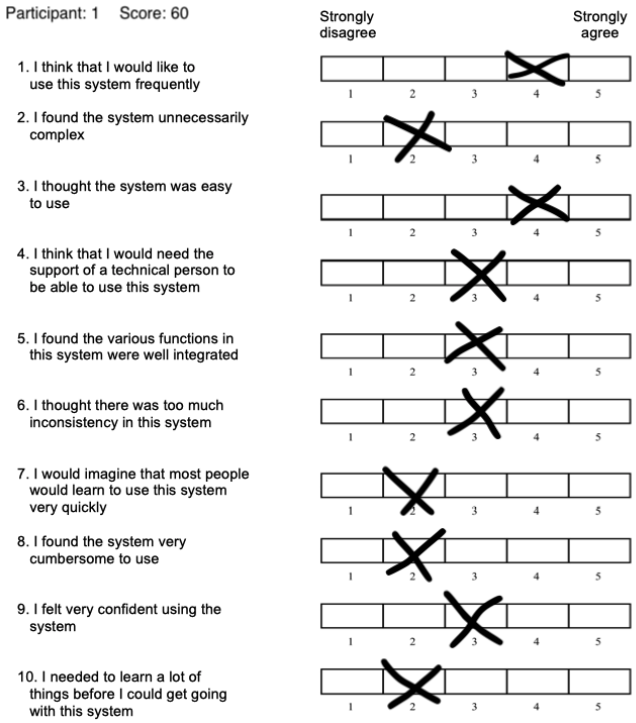

Figure 16. Participant 1

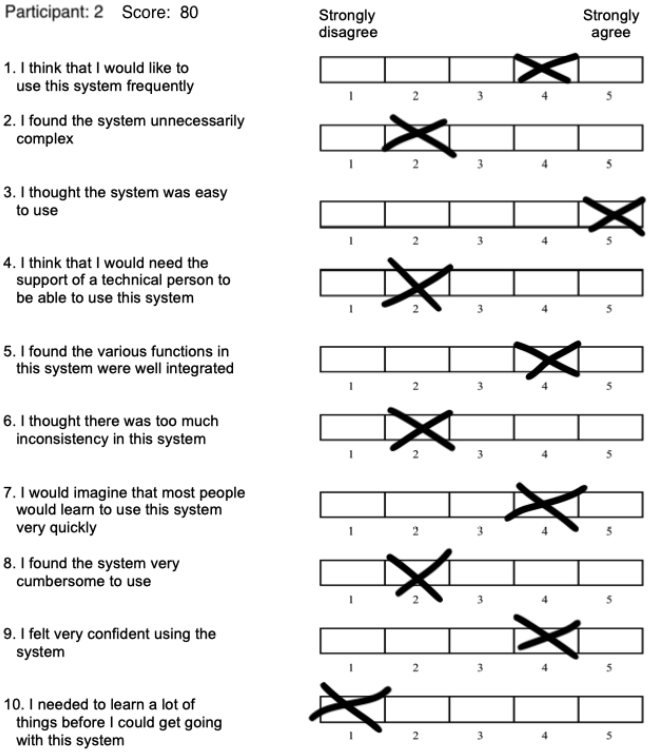

Figure 17. Participant 2

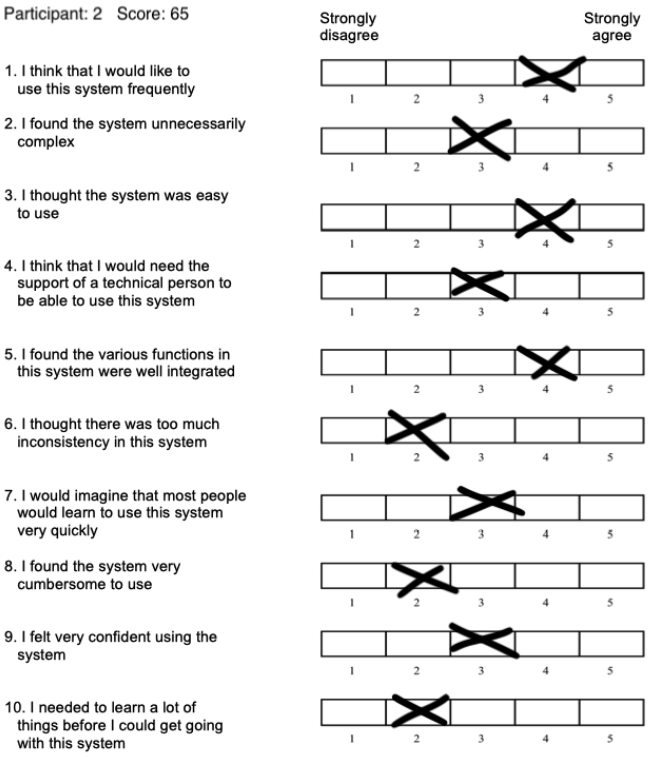

Figure 18. Participant 3

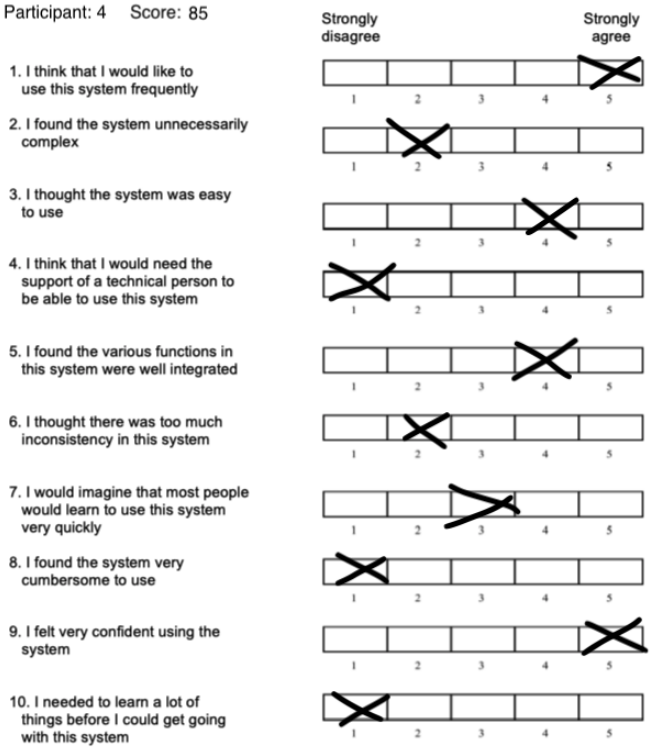

Figure 19. Participant 4