Svetlana Sannikova

Chatbot implementation with Microsoft Bot Framework

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Thesis

19.03.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Svetlana Sannikova Chatbot-sovelluksen kehitys Microsoft Bot Frameworkilla 28 sivua 19.03.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Smart Systems and Software Engineering
Ohjaajat	Kimmo Saurén, Senior Lecturer

Insinöörityön tavoitteena oli kehittää chatbot-sovellusta projektitiimin Slack-kanavaa varten ja tutustua Microsoft Bot Framework ja Azure Cloud -teknologioihin. Chatbot-projekti suunniteltiin yhteistyössä projektitiimin jäsenten kanssa. Projektin tavoitteena oli saada chatbot-sovellus asennettuna tiimin yksityiseen Slack-kanavaan.

Sovellus kehitettiin Visual Studio IDE:llä käyttäen C# -ohjelmointikieltä ja muita Microsoftin teknologioita kuten Azure Cloud Services, Bot Framework and Unit Test Framework.

Sovellus suunniteltiin reagoimaan vain tiettyihin avainsanoihin, jotta se ei häiritsisi normaalia kommunikaatiota kanavalla. Jos viestissä on avainsana, sovellus tarkistaa, onko viestissä tämän lisäksi komentoa. Havaitessaan komennon, sovellus suorittaa siihen liittyvän toiminnon. Tässä tapaustutkimuksessa sovelluksen oli osattava lähettää päivän lounasruokalista sekä pyynnöstä että itsenäisesti kello 10.00 yhteiseen keskustelu kanavaan.

Sovellus on tallennettu Git -repositorioon ja sijoitettu Azure Cloud hosting-palveluun. Sovellus ei vaadi uudelleenkäyttöönottoa, vaan se päivittyy automaattisesti repositoriossa havaitun muutoksen yhteydessä.

Sovellusalustan käyttöönoton jälkeen, sovellusta on kehitetty monilla tässä työssä mainitsemattomilla toiminnoilla. Sovellus pysyy edelleen aktiivisena projektitiimin Slack-kanavassa.

Avainsanat	VS, C#, Slack, chatbot, Azure, Bot Framework, Git, .NET

Metropolia

Author Title Number of Pages Date	Svetlana Sannikova Chatbot implementation with Microsoft Bot Framework 28 pages 21 August 2017
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Smart systems
Instructors	Kimmo Saurén, Senior Lecturer

The purpose of this thesis is to create a chatbot application for a Slack channel and to get familiar with the Microsoft Bot Framework and Azure Cloud services. The chatbot project was designed in cooperation with .NET team members. The aim of the project is to deploy the chatbot to the team's private chat channel.

The application was developed on Visual Studio tools using C# as a primary programming language and others Microsoft technologies, such as Azure Cloud services, Bot Framework and Unit Test Framework.

The application developed in this thesis project is designed to react to a few keywords in messages, because without restriction it would interrupt normal communication in the channel reacting to every message. If the key word is detected, the application is checking if there is a command word, and if one is found it returns the provided by its program logic answer. In this case study the application should be able to send a lunch menu by request and also proactively at 10AM every day to the common chat channel.

The application is stored to a Git repository and deployed to Azure Cloud hosting with option of continuous delivery, which means that the application should not be redeployed every time a new feature was developed, but it is happening automatically, when new commit in the repository is detected.

After the working bot base was successfully deployed, many other features were added by team members. The application remains to be in active use in the team channel.

Keywords	VS, C#, Slack, chatbot, Azure, Bot Framework, Git, .NET

Metropolia

Contents

List of Abbreviations

API	Application programming interface
JSON	JavaScript Object Notation
IDE	Integrated development environment
REST	Representational state transfer
SDK	Software Development Kit
URL	Uniform Resource Locator
VCS	Version Control System

1 Introduction

Text communication nowadays takes huge part in peoples' lives. Almost all age groups for personal, family and social communication as well as for business purposes use text messaging. Many companies use text messaging for communication between employees.

Last several years in software development was growing a concept of "virtual teams", where team members work from multiple locations and communicate between each other mostly online. In such scenario, the communication channels are playing the key role in successful teamwork. As team members may work from different time zones, the easiest way to communicate is texting in team's private channels.

A number of services provide platforms for team online collaboration. One of such services is Slack, which offers persistent chat rooms for teamwork, option of creating private groups and direct messaging. Slack is mostly oriented to software development teams and it has integrations with large number of tools useful for developers.

The purpose of this thesis project was to create a chatbot for Slack channel of software developers' team, which would offer services useful for the team members. Another goal was to get familiar with functionality provided by Microsoft Bot Framework and Azure Bot Services.

The idea of the bot was developed by .NET team members in Digia Oy in spring 2017. The project did not have any commercial use purposes. It was designed and developed in free time as a hobby project only for internal use in private Slack workspace of the team.

2 Chatbot concept and history

2.1 Background

A chatbot is a computer program, which simulates human conversation, or chat, through artificial intelligence. Typically, a chatbot will communicate with a real person, but applications are being developed today can communicate also with each other.

The idea of chatbots is as old as a computer itself. For the first time it was introduced by a creator of theoretical computer science Alan Turing in his seminal paper "Computing Machinery and Intelligence" in 1950. In the paper, he introduced the concept of the Turing Test, which would test if a computer can act indistinguishably from the way a thinker acts.

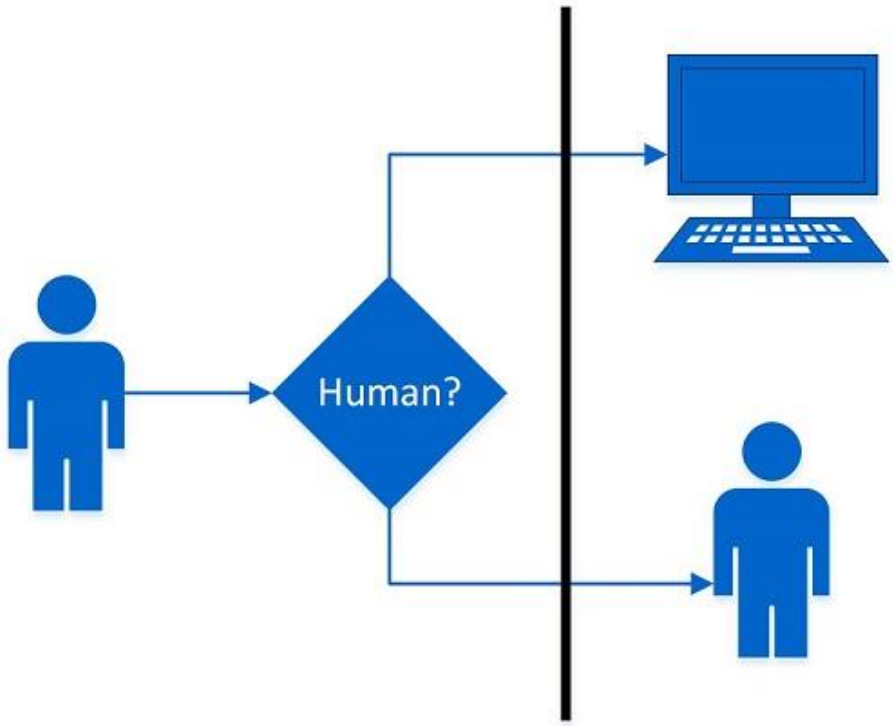

Figure 1. The traditional interpretation of the Turing Test. [1.]

The traditional interpretation of the Turing Test is presented in the figure 1. The test includes at least three participants: a human, a machine and a judge. The judge should decide whether it is a human or a machine he is talking to at the moment. The job of the judge would be to interrogate the human and machine with a series of questions and

based on their reactions, tone, and how they reply to the questions being asked, make a decision. [2.]

The first said to be a program able to pass the Turing Test was ELIZA, created in 1966 by Joseph Weizenbaum at the MIT Artificial Intelligence Laboratory. In the program, text was read and inspected for the presence of a keyword. If a keyword was found, the answer was transformed according to the rule, associating with the word. [3.] In other words, it was recognizing words or phrases in the input, and producing the output with corresponding pre-programmed sentences. The same method is used by chatbot programmers ever since.

Michael Mauldin mentioned the term ”ChatterBot” first time in 1994 for describing conversational programs at the twelfth national conference on artificial intelligence and after this the term was actively in use.

In 1997 was lunched first chatbot with real-time learning algorithms, Jabberwacky. While all older programs got responses from a static database, Jabberwacky collected phrases used by human participants and added them to its database, dynamically growing its own content.

The popularity of artificial intelligence concept significantly increased, when the smartphone came to mass market in the 2000s. It happened that messaging applications became the most popular, as they are especially well suited for mobile devices. [4.] Affordable mobile internet networks only increased their popularity.

Nowadays the global chatbot market is valued at about $200 million and it is expected to grow in the coming years. According to a study by Aspect Software Research, 44% of consumers said they would prefer to interact with a chatbot over a human customer service representative, because a bot is able to provide instant responses to questions. [5.] A big advantage of the bot applications in customer service is that people feel free to put the questions they would not put to a human representative.

2.2 Use of chatbots

Chatbots are usually integrated into the dialog systems of, for example, virtual assistants, giving them the ability of natural communication or engaging in casual conversations unrelated to the scopes of their primary expert systems.

They can be useful in many aspects of the customer experience, including providing customer service, presenting product recommendations and engaging customers through targeted marketing campaigns.

In most cases, chatbots use messenger apps to communicate with customers. A person can type or ask a question and the chatbot responds with the right information. Depending on the situation, many chatbots can learn from what a customer says to personalize the interaction and build off previous interaction. For example, if a customer talks with a chatbot and asks for movie recommendations, the chatbot can remember which movie the customer saw and follow up with it later when providing a recommendation for a restaurant or another movie. [6.]

One of the best examples of using chatbots is Facebook Messenger app. Facebook is the most popular messaging app with over 1.2 billion active users. In 2016, Facebook Messenger allowed developers to place chatbots on their platform and in one year by developers was created over 100,000 bots and amount of messages between business applications and customer has reached to 2 billion per month.

For example on image 2 is shown a Spring Bot, it was one of the first bots launched on Facebook Messenger in 2016. It is a real time messaging and personal shopping assistant with a direct-to-consumer sales model created in 2014. Fashion brands can use it to connect with customers bypassing traditional stores, which cost a lot, especially for smaller brands. At the application launch in 2014, Spring had 150 brands on board, but by 2017 it had 3,000 brands selling in application and sells were growing by 20% month over month for almost 2 years. [7.]

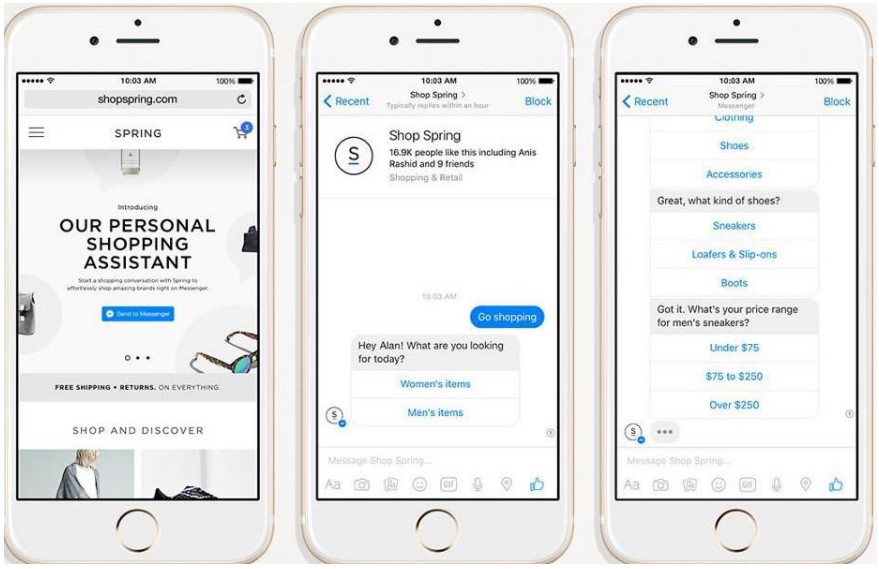

Figure 2. Spring shopping assistant for Facebook Messenger. [8.]

2.3 Development aspects

The biggest challenge chatbot developers face today is a big amount of messaging platforms. To create a successful bot, developer should make sure, that users of every of these platform experience a similar, consistent interaction with the bot, and that they are able to share these services with friends, who are using other platforms.

Top 3 Social Messaging Apps Among Smartphone Users in Select Countries, Jan 2015

% of respondents

China	
1. WeChat	84%
2. QQ	83%
3. Sina Weibo	50%
Germany	
1. WhatsApp	77%
2. Facebook Messenger	49%
3. Skype	30%
Japan	
1. LINE	75%
2. Twitter	42%
3. Facebook Messenger	25%
UK	
1. Facebook Messenger	63%
2. WhatsApp	46%
3. Skype	36%
US	
1. Facebook Messenger	60%
2. Skype	29%
3. Twitter	25%

Note: n=2,500 Android and iPhone users
Source: On Device Research as cited in company blog, March 18, 2015

187763 www.eMarketer.com

Figure 3. Top social messaging apps among smartphone users. [9.]

Currently all users of messaging applications are divided between few primary platforms shown on figure 3. Every platform differentiate itself from other with technical possibilities, audience and communication types.

The limitations of every platforms make software developers to look for new approaches and bot frameworks provide them with an opportunity to solve the problem in the short term. Modern bot frameworks let developers create cross-platform applications, which can be integrated into multiple channels, without making changes to the source code.

3 Implementation alternatives

3.1 Bot frameworks

Bot framework is a set of predefined and preinstalled methods and classes created for bot developers. It gives to developer a set of tools which help write the code better and faster. In simple terms bot developers and programmers use development frameworks to create chat bots from scratch using programming language.

The most famous and modern bot platforms, which allow developers to create own bots from scratch are Facebook Bot Engine, Dialogflow developed by Google and Microsoft Bot Framework.

3.1.1 Facebook Bot Engine

Facebook released Facebook Bot Engine in April 2016. It is based on Wit.ai technology, bought by Facebook in 2015. Wit.ai runs from its own server in the cloud. The Bot Engine allows developing bot applications for Facebook Messenger platform. Being a huge social media network with more than a billion users Facebook decided to stay focused on Facebook Messenger only.

Facebook adopted a new strategy with the Facebook Bot Engine. If developers grasp the framework, Facebook Messenger users are going to get a variety of specialized chatbots.

The Facebook Bot Engine actively relies on Machine Learning. Users feed the Bot Framework sample conversations and it can handle many different variations of the same questions. The potential is quite big as developers could improve their chatbots over the period.

Being one of the most modern framework, Facebook Bot Engine does not support any other platforms, except of Facebook Messenger and does not fit the purpose of the project. [10.]

3.1.2 Dialogflow

Dialogflow is a Google-owned technology of developing human-computer interaction based on natural language conversations. Dialogflow provides a platform that allows developers to design and implement conversation interfaces which can be embedded in external applications like bots. [11.] The basic data flow in Dialogflow system in shown if figure 4.

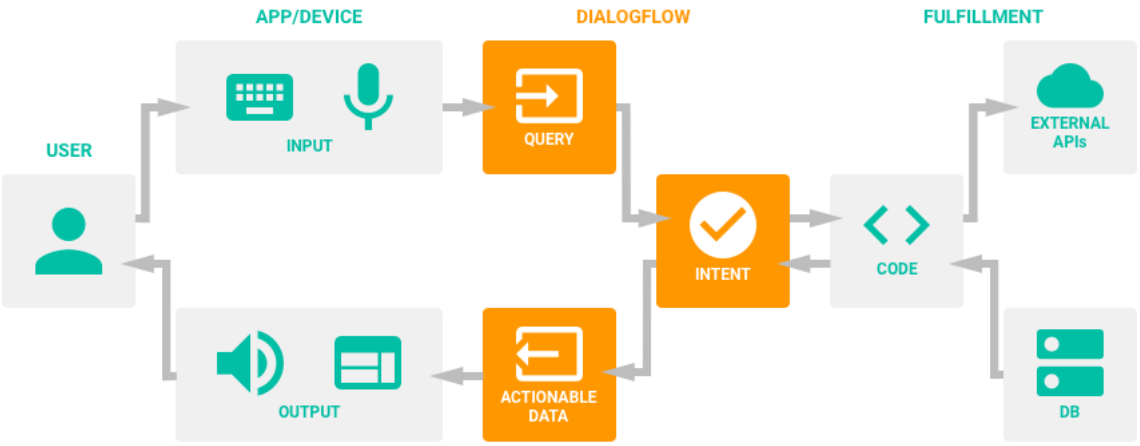

Figure 4. Data flow in Dialogflow. [12.]

Dialogflow supports 14 different platforms including Skype, Telegram, Slack, Cortana, Alexa and Facebook Messenger. However, even though bot application created with Dialogflow is technically possible integrate with Azure Bot service; it would demand additional preparation, because Dialogflow bots are designed to work with Google's own Cloud Platform.

3.1.3 Microsoft Bot Framework

Microsoft introduced own bot framework in early 2016. Microsoft bot framework SDK, like all the other frameworks, provides the resources a developer needs to build intelligent conversational chatbot that interact naturally.

My final work project was implemented using Microsoft Bot Framework, because I have a lot of experience with Microsoft technologies and C# - programming. Microsoft Bot Framework's full SDK is available in C#.

The Bot Framework provides components for developers to help solve such problems as automatic translation to different languages, user and dialog state management and debugging. Main components of the framework are:

1. Bot Connector is a service, which allows a bot to exchange messages with channels that are available in Microsoft Bot Framework, by using REST API and JSON over secure protocol HTTPS. When a user sends a message, the Bot Connector sends a POST request to the endpoint that is specified during bot registration. An example of body of the request is shown in Listing 1.

```
{
  "type": "message",
  "text": "message text",
  "from": {
    "id": "default-user"
  },
  "locale": "en-GB",
  "textFormat": "plain",
  "timestamp": "2018-03-09T15:22:55.174Z",
  "channelData": {
    "clientActivityId": "1520608946838.7637367733445164.0"
  },
  "entities": [
    {
      "type": "ClientCapabilities",
      "requiresBotState": true,
      "supportsTts": true,
      "supportsListening": true
    }
  ],
  "id": "hc11k7eafh8f",
  "channelId": "emulator",
  "localTimestamp": "2018-03-09T17:22:55+02:00",
  "recipient": {
    "id": "96elffmf1d0d",
    "name": "Bot"
  },
  "conversation": {
    "id": "3gh9i70d5eb5"
  },
  "serviceUrl": "http://localhost:53369"
}
```

Listing 1. Example of request body

When bot gets a JSON object, the information from it can be used to create a response to user.

2. Bot Builder is a SDK for .NET Framework developers for developing bots using Visual Studio and Windows. The SDK supports C# and Node.js programming languages. The kit consists of Bot Application, Bot Controller and Bot Dialog templates. Bot Application template already contains a simple project with all of the components for a simple bot. It includes a POST method to accept messages and a dialog builder to generate a response as shown on image 5.

```
[Serializable]
public class RootDialog : IDialog<object>
{
    public Task StartAsync(IDialogContext context)
    {
        context.Wait(MessageReceivedAsync);

        return Task.CompletedTask;
    }

    private async Task MessageReceivedAsync(IDialogContext context, IAwaitable<object> result)
    {
        var activity = await result as Activity;

        // calculate something for us to return
        int length = (activity.Text ?? string.Empty).Length;

        // return our reply to the user
        await context.PostAsync($"You sent {activity.Text} which was {length} characters");

        context.Wait(MessageReceivedAsync);
    }
}
```

Figure 5. Autogenerated dialog builder in Bot Application project.

4 Project Implementation

4.1 Version control

Version control is a system used for recording changes to a file or set of files over time so that user can return specific version later, if it is necessary. The need for a logical way to organize and control revisions has existed for almost as long as writing has existed, but revision control became much more important and complicated, when the era of computing began. Today the most complex version control systems are the ones using in software development, where many developers work on the same project at the same time. In software development, version control usually is used for tracking source code changes, but it also can be used for documentation maintaining.

The most popular available version control systems today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. [13.]

The major difference between Git and any other VCS is the way Git groups its data. With Git, every time a new commit is pushed it takes a picture of what files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git does not store the file again, just a link to the previous identical file it has already stored. [14.] The process is shown in figure 5.

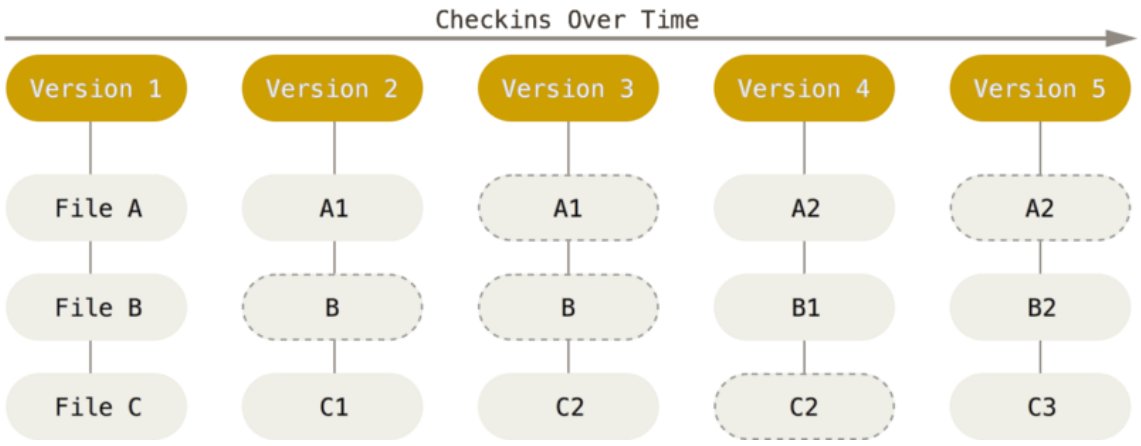

Figure 6. Storing data as snapshots of the project over time. [14.]

As the chatbot project was in common access with all the team, it was important to use version control. Many web-based hosting services provide support for Git repositories. Some of them are oriented mainly to host open-source software project like GitHub. The chatbot's source code was stored in Bitbucket, which allows creating private repositories also for free accounts.

Usually in software projects work is happening in many branches. Branching is a function, which is available not only in Git, but also in all modern VCSs. A separate branch represents a line of development. In most cases for every new feature should be created a new branch as shown in figure 6. After the feature is ready and tested, the branch can be merged to master, main branch containing production version.

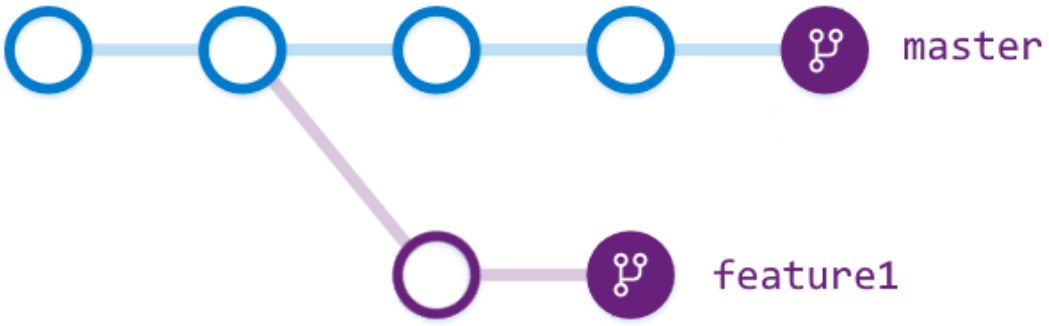

Figure 7. New branch for feature1. [15.]

However, while the project was relatively small and amount of developers actively taking part in it was limited by one or two persons, the development was easy to coordinate and it happened only in one master branch.

4.2 Hosting service

To make bot accessible for a Slack channel, it should be hosted on a web hosting service. There are a number of cloud platforms, which provide hosting services. The most common cloud platforms are Amazon Web Services, Microsoft Azure and Google Cloud Platform. All of them have similar services for web application hosting and possibility to use limited functionality free of charge.

As for implementation was chosen Microsoft Bot Framework, the best option for hosting became Azure, also developed by Microsoft.

Microsoft Azure is a cloud computing service announced by Microsoft in 2008. Azure allows to host web application, servers, databases, file storages, virtual machines or user directories. Many companies use it instead of buying their own hardware, as it is cheaper option and in case when company needs more resources, it is easier to add them.

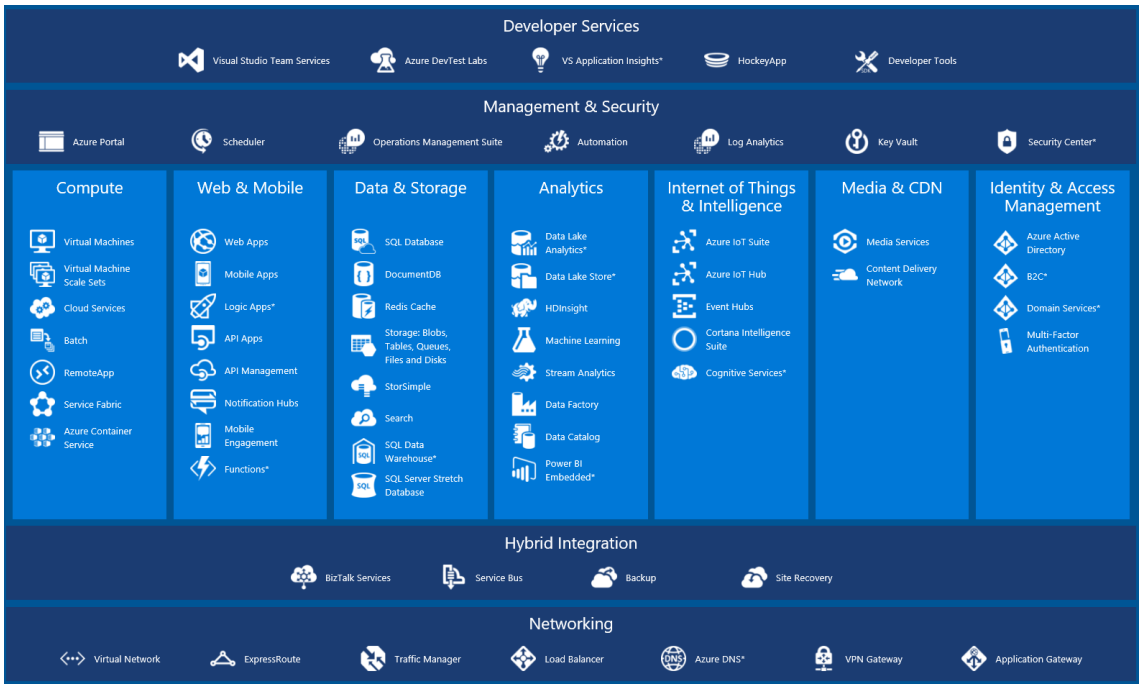

Figure 8. List of Microsoft Azure services. [16.]

The figure 8 gives an overview of all the services Azure platform provides to developers.

Azure Bot Service became generally available in late 2017 on Azure platform. It provides a scalable, integrated bot development and hosting environment for conversational bots that can reach customers across multiple channels on any device. At the same time, Azure presented Microsoft Cognitive Services Language Understanding service (LUIS) which helps to create customized natural interactions. LUIS designed to identify important information in conversations. [17.]

The basic data flow is illustrated in figure 9. Chatbot provides the interface for user input; it can be in traditional text format or, for example, speech or image. The Azure Bot Service supports fourteen channels for communication with users including Slack, Facebook Messenger, Skype, etc. Intelligence is enabled in the Azure Bot Service through the cloud AI services forming the bot brain that understands and reasons about the user input. [17.]

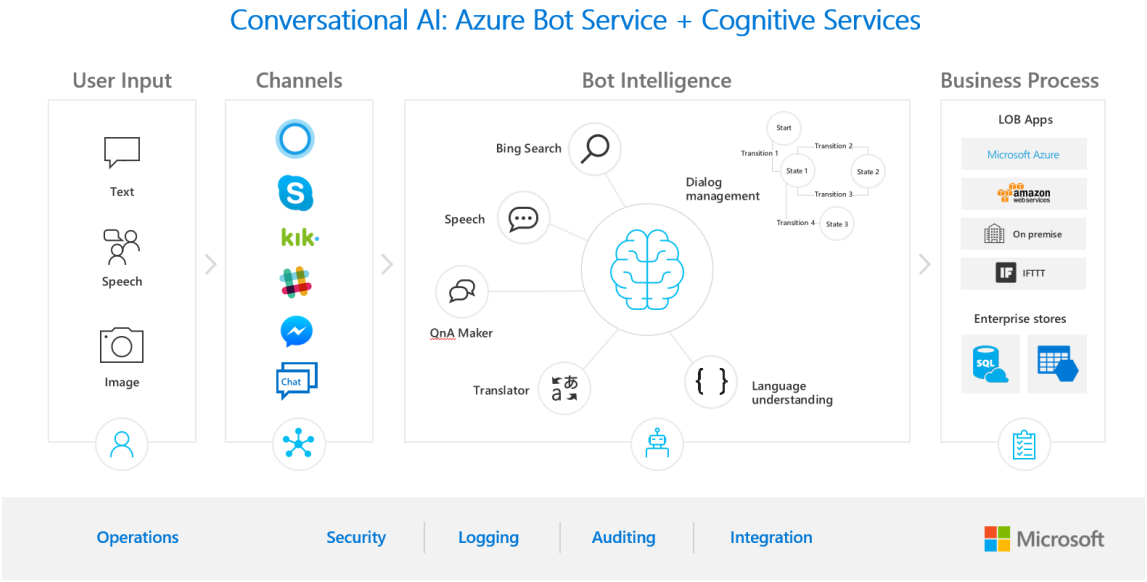

Figure 9. Bot Application data flow. [17.]

4.3 Development

First step in software application development is creation of an empty application project. As for implementation was chosen Microsoft Bot Framework and Azure hosting services, the most appropriate IDE for my goals is Microsoft Visual Studio and C# as a programming language.

Visual Studio provides a suite of tools for developers for creating software projects, from the planning phase through user interface design, coding, testing, debugging, analysing code quality and performance, deploying to production, and gathering telemetry on usage. [18.]

Visual Studio has a set of editions with different service plans depending on developer's needs. For student and individual developers there is a free Community edition of Visual Studio. It provides almost full functionality free of charge. All the necessary tools for a chatbot application development are functioning correctly in the free edition of Visual Studio.

The first step of creation of a new bot project is to download application templates from Microsoft Bot Framework web page. The templates should be saved to Visual Studio project templates directory. After it is done, Visual Studio suggests a Bot Application template, when creating a new project as shown on figure 10.

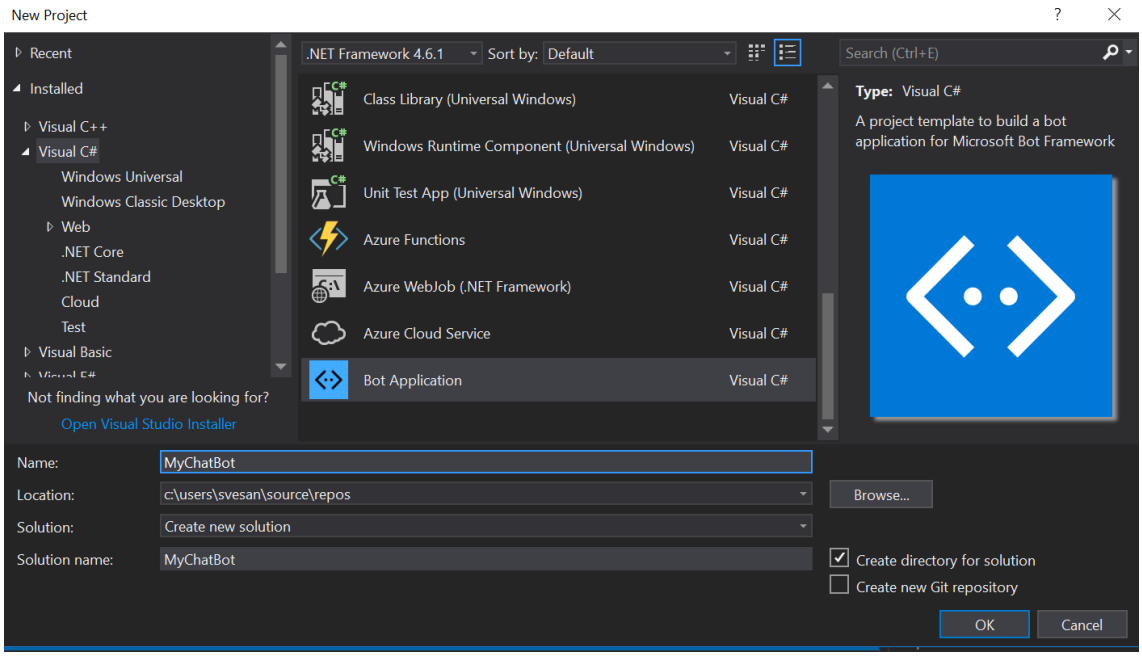

Figure 10. New Bot Application creation

When the project is created, it already contains all of the required references to the Bot Builder SDK for .NET and components for building a simple bot.

The sample application includes a POST method to accept messages and send a response telling how many characters were in the sent message.

As the bot is designed to work in a team's Slack channel, the first improvement needed to do is restricting messages it is reacting to, not to interrupt normal communication between team members, when messages are not addressed to the bot. With the code example shown in Listing 2 the chatbot will react only to messages, which contain word "bot".

```
if (activity.Type == ActivityTypes.Message)
{
      try
```

```
      {
            var text = activity.Text.ToLower();
            if (text.Contains("bot")
                  await Conversation.SendAsync(activity, MakeRoot);
      }
      catch (Exception ex)
      {
            throw new Exception(ex.Message);
      }
}
```

Listing 2. Limitation for messages bot is sending response to.

When bot is configured to react only to certain messages, some other functionality can be added. One of the most important function for the bot by opinion of the team is posting lunch menu in Amica restaurant downstairs. Amica group provide menus on restaurant's web page also in JSON format to help developers in embedding menus into their web applications.

As the bot project did not have strictly defined architecture or functionality and it would evolve further development, it is better way to create separate classes for every new service, not to run into code readability issues in the future. The service for sending menu lists can have name FoodService, and inside of it, a method GetAmicaMenu. Such naming allows to add other restaurants in neighbourhood later. The example code for parsing the menu is shown in listing 3. The method returns a text variable with list of food items separated with line feeds.

```
public String GetAmicaMenu(string url)
{
      try
      {
            dynamic menuList = JsonConvert.DeserializeObject(
                new WebClient().DownloadString(url));
            string response = "";
            foreach(dynamic item in menuList.MenusForDays[0].SetMenus){
                  response += item.Name + ": ";
                  foreach(dynamic component in item.Components){
                      response += component + "\n\n";
                  }
            }
            byte[] bytes = Encoding.Default.GetBytes(response);
            response = Encoding.UTF8.GetString(bytes);
            return response;
        }
        catch (Exception e){
              throw new Exception(e.Message);
        }
  }
```

Listing 3. Method for menu parsing

For separating the menu request from others, there is a NuGet package called BestMatchDialog for Bot Framework, which allows to match the incoming messages against a list of strings it takes as a parameter. It can be separate words or sentences. If a request, in an addition to the key word “bot”, consists any of the words from the list, dialog handler calls the following function. If there is no match in the list, then the default NoMatchFound handler will be called.

After the NuGet package is installed, the RootDialog can be changed to be the BestMatchDialog type. After that, the default response method can be replaced with two separate methods for each necessary case as shown in listing 4. There is a method to handle a case, when request contains some word of BestMatch list, and a method for a case, when no match was found. In first case bot responses with a lunch list and in second just says, “I don't know what to say”.

```
[BestMatch("food, lunch, amica, menu")]
public async Task HandleFood(IDialogContext context, string messageText)
{
     FoodService foodService = new FoodService();
     var response = foodService.GetAmicaMenu("http://www.amica.fi/mod-
ules/json/json/Index?costNumber=3121&language=fi");

     await context.PostAsync(response);
     context.Wait(MessageReceived);
 }

 public override async Task NoMatchHandler(IDialogContext context, string mes-
sageText)
 {
     var response = "I don't know what to say";
     await context.PostAsync(response);
     context.Wait(MessageReceived);
 }
```

Listing 4. BestMatchDialog implementation in root dialog.

Now the bot is functional and can be deployed, but for comfort of the users, bot could be proactive and post the menus every day before lunchtime without a request. For this bot needs, in addition to the automatically generated POST method, a GET method which would respond with a message to the specified Slack channel. The MenuResponse method is shown on listing 5, it will be called if API gets a request to address *hostname*/api/requests/menu. If the request succeeded, API responds with success code 200. In case if some error happened during the request, API returns the error information.

```
[Route("api/requests/menu")]
[HttpGet]
public async Task<HttpResponseMessage> MenuResponse()
```

```
{
      FoodService foodService = new FoodService();
      var msg = foodService.GetAmicaMenu(
            "http://www.amica.fi/modules/json/json/Index?costNumber=3121&lan-
guage=fi");
      return await SendMessage(msg);
}
```

Listing 5. MenuResponse function

SendMessage method creates a message and send it to the specified conversation channel. Channel parameters will be known after bot's deployment.

4.4 Testing

Testing is the process of evaluating a system or its components with the intent to find whether it satisfies the specified requirements or not and discover software bug before deploying it to production. The universe of testing automation can be neatly split into two predominant testing techniques known as white-box and black-box testing. [19.]

White-box approach evaluates work of internal structures or services, instead of functionality provided to users. The developer itself creates test cases for the services and chooses input parameters for every unit to determine the appropriate outputs.

Black-box testing focuses mainly on the functionality of the software product, without information about its internal structure and implementation methods. It usually represents the end-user point of view.

For the white-box approach, developers use unit and integration tests and for black-box testing of the application interface by real users or testers.

For testing the services in the bot application, I used Microsoft Unit Test Framework to create and run unit tests. Interface testing happened with Bot Framework Emulator, also developed by Microsoft.

4.4.1 Unit testing

Unit testing is a level of software testing where individual units/components of a software are tested. The purpose is to validate that each unit of the software performs as designed. [20.]

For the testing to the solution should be added a new Unit Test template project and should have a reference pointing to the project it is testing.

As an example of unit tests, the code in listing 6 is testing that Food Service returns not empty text variable.

```
[TestMethod]
public void TestGetAmicaMenuSuccess()
{
     string menu = "";
     menu = foodService.GetAmicaMenu("http://www.amica.fi/modu-
les/json/json/Index?costNumber=3121&language=fi");
     Assert.AreNotEqual(menu, "");
}
```

Listing 6. Test method

Similarly can be tested, that Food Service throws an error, if it cannot load a JSON from the given URL, using attribute ExpectedException, which indicates that during the test method execution application should throw an exception of specific type.

4.4.2 Bot Framework Emulator

The Bot Framework Emulator is a desktop application that allows bot developers to test and debug their bots, either locally or remotely. Using the emulator, developer or tester can chat with the bot and inspect the messages that it sends and receives. The emulator displays messages as they would appear in a web chat UI and logs JSON requests and responses. [21.]

The Bot Emulator connects to the given endpoint URL, when chatbot application is running, and emulates a communication between bot and user.

The endpoint URL is configured in Bot Builder sdk and default endpoint for local debugging is usually http://localhost:3978/api/messages.

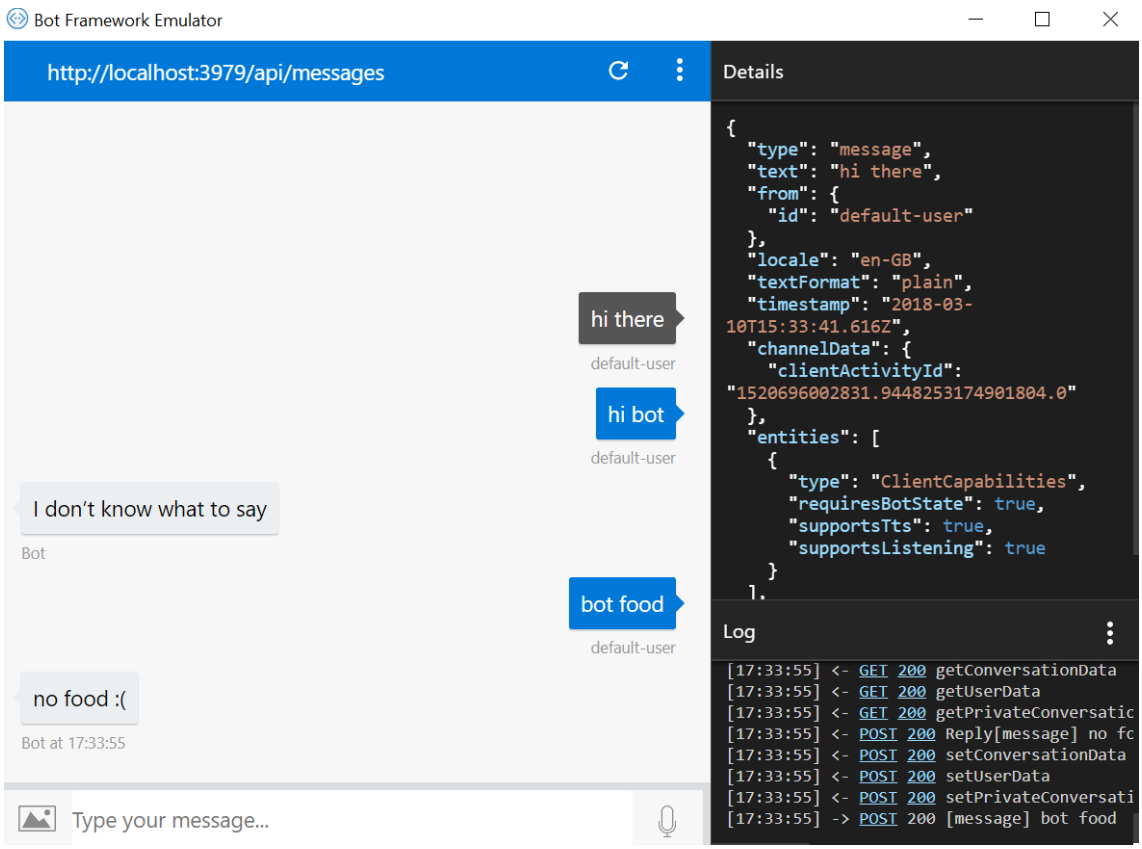

Figure 11. Microsoft Bot Emulator view

As shown on figure 11, the emulator window consists of 3 separate view:

1. The dialog window on left side where user can see input sent to the bot and output produced by the bot.

2. JSON object of chosen message on right upper side. The view helps developers see the sequence sent to the application and response from it.

3. Logs of requests and responses in right lower corner with information about API activities and codes of statuses.

For testing the posting menu GET method the specified URL can simply opened by web browser, as browsers by default send a GET request to the given address.

Another option is to use special tools for API developers, for example provided by Google Postman. The main profit of such tools is possibility to choose type of request, see and modify request body, see full response and possible errors.

In both cases, response sent to Bot Emulator is similar and shown in listing 7. Answer “no food” only means, that request was sent on weekend and JSON did not contain any menu items.


```
{
  "type": "message",
  "channelId": "emulator",
  "from": {
    "id": "96elffmf1d0d",
    "name": "Bot"
  },
  "conversation": {
    "isGroup": true,
    "id": "j9f71ai0592k"
  },
  "recipient": {
    "id": "n66bc2bmjdie"
  },
  "membersAdded": [],
  "membersRemoved": [],
  "text": "no food :(",
  "attachments": [],
  "entities": [],
  "id": null
}
```


Listing 7. API response to lunch menu request

After chatbot’s functionality was tested and all bugs fixed, it can be deployed to the Azure Cloud hosting service.

4.5 Deployment

The deployment of the bot starts from registration on Azure Portal. After registration user can choose a subscription and support plans. There are free options with Pay-As-You-Go subscription and I used it for the project.

After registration user can add a new source and choose a Web App Bot application template. When application created, there is a bot template, which just echoes back users input. This template should be replaced with the Slack bot code, stored in Bitbucket repository. For this in Bot Management section user can configure continuous deployment.

Continuous deployment on practice means that every change pushed to the version control is deployed to production automatically, if the build succeeded. For Azure web application continuous deployment can be configured to use straight Bitbucket version control repository as shown in figure 12.

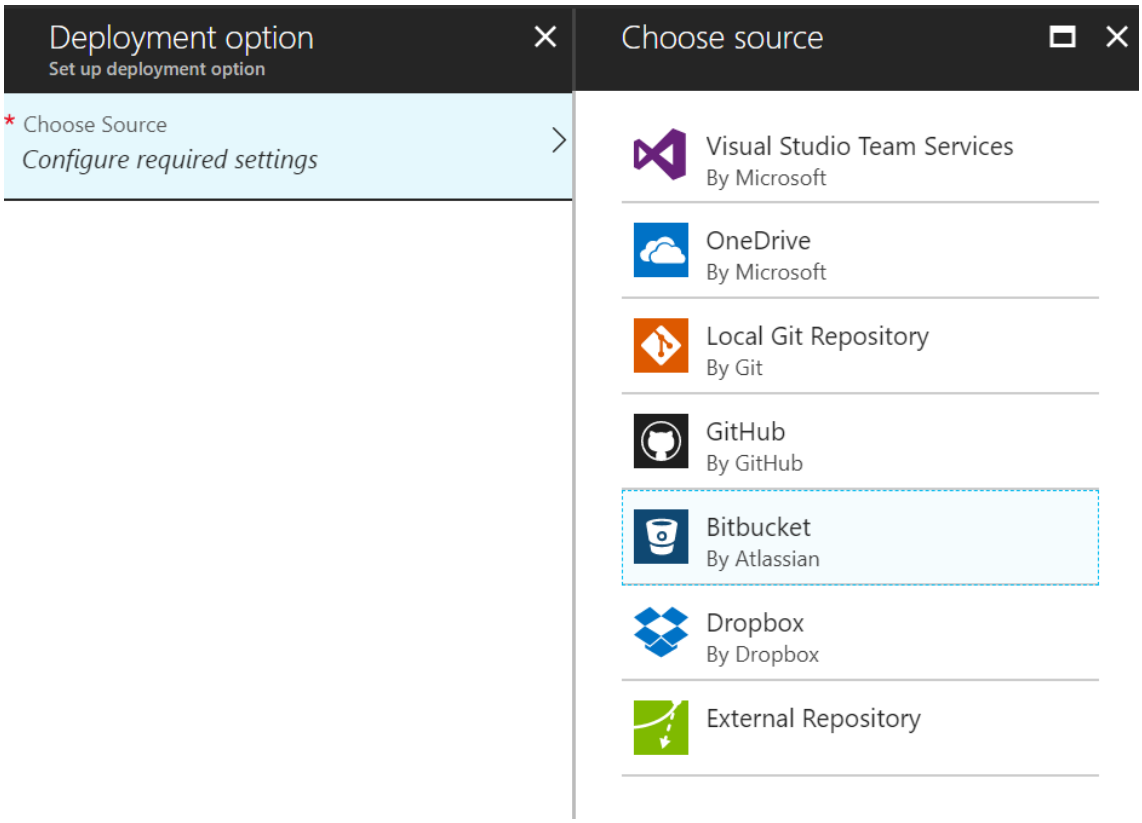

Figure 12. Azure continuous deployment options.

After authorization, system allows to choose the branch it would use for deployment, by default it is master branch. If the given repository contains any project, Azure fetch the files from there and replace the template project, created by Bot Service. When build is complete, the bot application is hosted in Azure Cloud on URL *https://botname.azurewebsites.net/api/messages* and ready to use in communication channels. There is also an option to test the application in Azure web chat (figure 13), to make sure that application works correctly after deployment.

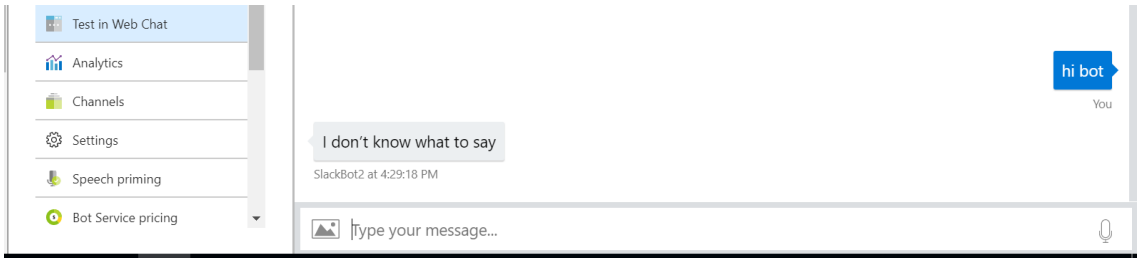

Figure 13. Azure Web Chat for testing Web Bot applications.

For using the bot in Slack, it should be registered as a new application in *api.slack.com*, a hub for developers, which allows integrating complex services and third-side applications with Slack. When choosing a Create app option, first should be chosen a Slack channel, the application will be installed to and an application name. During the registration, new bot user account should be set pointing to the Azure bot and the application credentials, generated by Slack should be stored to the Azure Cloud to channel information for Slack. Information about active channels is available in *Channels* section of the bot application. If authorization was successful, the channel status changes to running as shown on figure 14.

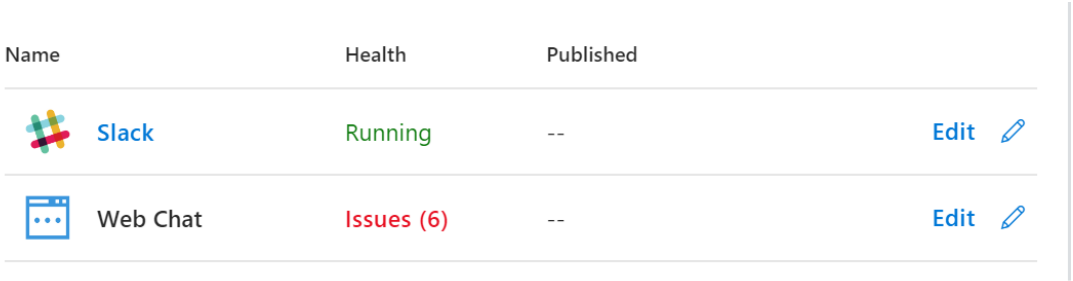

Figure 14. Running bot application in Slack channel

After registration on both sides is completed, the running bot application appears as an App in the specified Slack channel (figure 15).

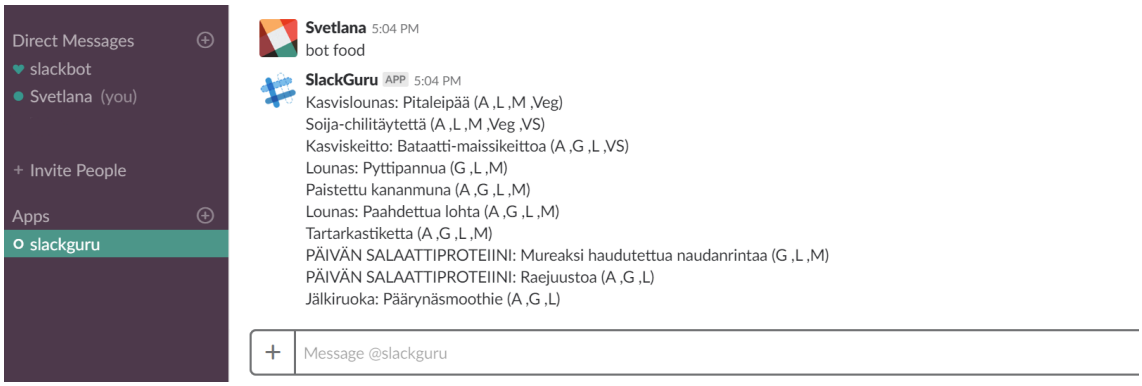

Figure 15. Bot application in Slack channel

Once the application added to a workspace, an administrator can add the bot to any channel inside the workspace he or she wants.

For the proactive messaging can be used any webhook service. One of the easiest is IFTTT (If this, then that), which takes a trigger “this” and if it is hit, then it makes action specified in “that”. It allows to create webhooks with requests to given URL and set time as a trigger of the request. Good time to post lunch menu is about 10 am. The ready webhook is shown in figure 16.

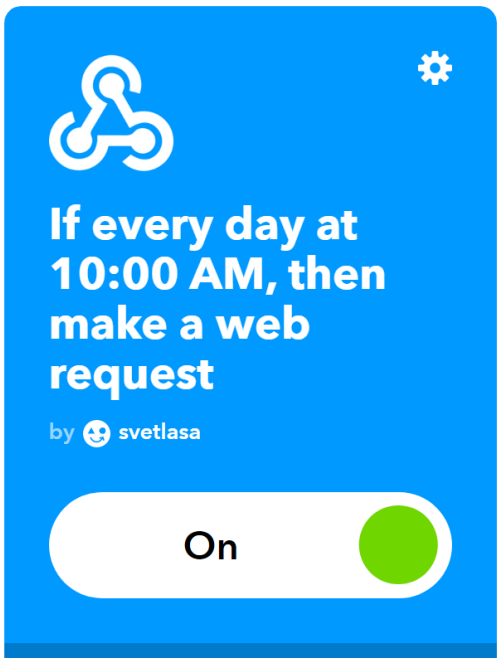

Figure 16. IFTTT request set at 10 AM.

5 Results

As a result was created a fully functioning chatbot, which could be integrated into a number of channels including Slack.

After the application deployment, the functionality of the bot was significantly extended by the team members.

In addition to the lunch menus, the bot is able to provide weather forecast, news, cat videos, Instagram pictures, daily Fingerpori and Dilbert comics, train timetable for Pitäjänmäki train station, Digia's share price and other services. It also was taught to joke, motivate and support team members, remind about coffee breaks and end of the working day, recognize languages and, if no matching phrases were found, answer to messages with twitter posts.

At the moment, when this report is being written, the bot is still in active use and development of current and former .NET team members.

6 Conclusions

The purpose of this project was to create a chatbot for a Slack channel of software developers' team, which would offer services useful for the team members.

The technology stack for the project was selected considering advantages provided by the tools as well as suitability of every technology for the stack. Using beforehand defined stack of technologies was implemented functionality and REST API of a bot application using quality assurance practices and stored into version control repository. After the functionality was approved, it was deployed to Azure Cloud hosting service and added as an application to team's Slack channel.

The project was developed using agile software development approach. First version of the bot produced only basic functionality, but later it was extended based on users' needs and feedbacks. The project also involve adding more features in the future and existing application base is a big advantage in further development.

The objective of developing bot application was successfully achieved.

Another goal was to get familiar with functionality provided by Microsoft Bot Framework and Azure Bot Services. For these purposes was applied “learning by doing” approach. The team members, who participated in the project development, could learn the new technology by implementing an application using it and discussing with the rest of the team results and best practices.

The quality of self-learning may vary, however the goal of getting familiar with a Microsoft Bot Framework and Azure services can also be recognized as achieved, at least not on the professional level.

References

1. Clark, Nate. 2015. What is Turing test? [online] http://www.clarkoncode.com/What-is-the-Turing-Test/. 7.10.2015. [Accessed 15.03.2018].

2. Irwin, Paula. 2017. What's the Turing Test and Which AI Passes It? [online] https://www.knowmail.me/blog/whats-turing-test-ai-pass/. 11.01.2017. [Accessed 15.03.2018].

3. Weizenbaum, Joseph. 1966. ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine. [online] http://web.stanford.edu/class/cs124/p36-weizenabaum.pdf. 01/1966. [Accessed 15.03.2018].

4. Kholod, Anastasia. 2017. The Brief History of Artificial Intelligence and Chatbots. [online] https://api2cart.com/business/brief-history-artificial-intelligence-chatbots/. 16.08.2017. [Accessed 15.03.2018].

5. Sweezey, Mathew. 2018. The Value Of Chatbots For Today's Consumers. [online] https://www.forbes.com/sites/forbescommunicationscouncil/2018/02/13/the-value-of-chatbots-for-todays-consumers/#36b530762918. 13.02.2018. [Accessed 15.03.2018].

6. Morgan, Blacke. 2017. What Is A Chatbot, And Why Is It Important For Customer Experience? [online] https://www.forbes.com/sites/blakemorgan/2017/03/09/what-is-a-chatbot-and-why-is-it-important-for-customer-experience/#6ab644c37188. [Accessed 15.03.2018].

7. Milnes, Hilary. 2017. The spiral of Spring's great mobile experiment. [online] https://digiday.com/marketing/spiral-springs-great-mobile-experiment/. 31.03.2017. [Accessed 15.03.2018].

8. Arthur, Rachel. 2016. Shopping Startup Spring Launches One Of First Bots On Facebook Messenger. [online] https://www.forbes.com/sites/rachelarthur/2016/04/12/shopping-start-up-spring-launches-one-of-first-bots-on-facebook-messenger/#68d8f5a4e3ce. 12.04.2016. [Accessed 15.03.2018].

9. Garg, Arun. 2015. Most Popular Messaging Apps – Country Wise. [online] http://www.quytech.com/blog/most-popular-messaging-apps-country-wise/. 18.09.2015. [Accessed 15.03.2018].

10. Maruti, Techlabs. 2017. Which are the best chatbot frameworks? [online] https://chatbotslife.com/which-are-the-best-on-site-chatbot-frameworks-3dbf5157fb57. 12.04.2017. [Accessed 15.03.2018].

11. Zyane, Rania. 2017. AI Platforms—You'll need them to make your chatbot smarter. [online] https://chatbotsmagazine.com/ai-platforms-youll-need-them-to-make-your-chatbots-smarter-rex-a884e1fea1ea. 12.10.2017. [Accessed 15.03.2018].

12. Huhtanen, Hanu. 2017. Chatbots made easy with Dialogflow. [online] https://blog.huhtanen.eu/2017/10/15/chatbots-made-easy-dialog-flow.html. 15.10.2017. [Accessed 15.03.2018].

13. Bitbucket tutorials. What is Git. [online] https://www.atlassian.com/git/tutorials/what-is-git. [Accessed 15.03.2018].

14. Git. Getting Started - Git Basics. [online] https://git-scm.com/book/en/v2/Getting-Started-Git-Basics. [Accessed 15.03.2018].

15. Microsoft. 2017. Git branch statistics. [online] https://www.visualstudio.com/en-us/docs/integrate/api/git/stats. [Accessed 15.03.2018].

16. Swinkels, Mark. 2016. Microsoft Azure Cloud Services Overview. [online] http://markswinkels.nl/2016/06/microsoft-azure-cloud-services-overview/. 30.06.2016. [Accessed 15.03.2018].

17. Microsoft Azure. 2017. Conversational Bots Deep Dive – What's new with the General Availability of Azure Bot Service and Language Understanding. [online] https://azure.microsoft.com/en-us/blog/conversational-bots-deep-dive-what-s-new-with-the-general-availability-of-azure-bot-service-and-language-understanding/. 13.12.2017. [Accessed 15.03.2018].

18. Microsoft. Developer Network. Visual Studio IDE. [online] https://msdn.microsoft.com/en-us/library/dn762121.aspx. [Accessed 15.03.2018].

19. Ericsson, Ulf. 2015. TEST DESIGN TECHNIQUES EXPLAINED #1: BLACK-BOX VS WHITE-BOX TESTING. [online] https://reqtest.com/testing-blog/test-design-techniques-explained-1-black-box-vs-white-box-testing/. 08.06.2015. [Accessed 15.03.2018].

20. Software testing. Fundamentals. Unit Testing. [online] http://softwaretestingfundamentals.com/unit-testing/. [Accessed 15.03.2018].

21. Microsoft. Bot Framework. 2017. Debug bots with the Bot Framework Emulator. [online] https://docs.microsoft.com/en-us/bot-framework/bot-service-debug-emulator. 13.12.2017. [Accessed 15.03.2018].