Building Chatbots

with Microsoft Bot Framework and Node.js

Akshay Kulkarni

MEAP

MANNING

MEAP Edition
Manning Early Access Program

Building Chatbots

with Microsoft Bot Framework and Node.js
Version 4

welcome

Thanks for purchasing the MEAP of *Building Chatbots with Microsoft Bot Framework and Node.js*. My aim with writing this book is to help anyone with a basic knowledge in Node.js to get started with building intelligent chatbots. Conversational applications like chatbots are going to be the next big thing and I hope that this MEAP will help you start building chatbots immediately. And with your help, the final book will be great!

I started building chatbots as terminal applications many years before the current chatbot craze, because of my personal fascination with the idea of talking computers. There weren't many tools available to help in building these chatbots or easy ways to integrate them into messaging applications. So when Microsoft announced the Bot Framework to build chatbots for multiple platforms, I immediately started experimenting with it and was very impressed with it. Since then I have built multiple chatbots using Microsoft Bot Framework and launched them to the public.

Microsoft Bot Framework consists of two parts, the Bot Builder SDK and the Bot Framework Portal. The Bot Builder SDK is available in both .NET and Node.js flavours and it provides an easy way to build chatbots providing abstractions and classes to model the conversation flow inside the chatbot. The Bot Framework Portal helps you deploy the bot built with Bot Builder SDK to multiple messaging applications like Facebook Messenger, Telegram, Skype and Slack. In this book I have tried to explain in detail about the Bot Builder Library, the different options available while building chatbots and how to decide which option to use.

I have personally learnt more when I could follow along and see something in action while learning a new topic. As a result this book is designed to help you follow along and build a fully featured intelligent chatbot by the end of it. In the first chapter we will learn the basics of chatbots and build a simple echo chatbot. In each following chapter we will add more features to the chatbot while learning about the classes and methods in the library that enable us to do this.

Your feedback is essential in making sure this book lives up to the promise of helping any Node.js developer build intelligent chatbots. So, if you have any questions, comments, or suggestions, please share them in Manning's Author Online forum for my book.

—Akshay Kulkarni

brief contents

1 Introduction to Chatbots

This Chapter covers

- **Understanding chatbots**
- **Considering the differences between chatbots and traditional software**
- **Getting an overview of the core architecture used in building chatbots**
- **Exploring the different tools available to build chatbots**
- **Using Microsoft Bot Framework to build a simple echo chatbot**

In April, 2016, Facebook officials announced the new Messenger Platform at their developer conference F8, sparking an interest and excitement about chatbots that has been increasing ever since. The Messenger platform allows us as developers to write programs that can send automated messages through Messenger, opening up new business opportunities and also to provide a better and more personal user experience. Even though the concept of building chatbots and using them inside messaging applications had existed on platforms like Telegram, Slack and IRC channels, the Messenger announcement in my opinion was crucial to popularizing the field of chatbot development. All the major software companies like Google, Amazon, Microsoft and IBM have been betting on chatbots by releasing their own set of chatbot related products to capture the market early. Satya Nadella, the CEO of Microsoft, has claimed that, "Conversation as a platform will have as profound an impact as touchscreens and the web." Indeed, there is widespread belief that chatbots are the next biggest thing and will cause a shift similar in nature to the shift caused by the Graphical Interface or the Internet.

Chatbots, or more generally, *conversational applications* will play a huge role in the future. As we transition towards a wearable tech or a Augmented Reality/Virtual Reality based world, conversational interfaces will become the de-facto way to interact with technology. We can already see this happening with Apple putting Siri inside the Apple watch, and Amazon, Google and Apple releasing smart speakers you can talk to in the form of Echo, Google Home and HomePod, respectively. We also have smart home technologies that let you control the lights, switches, thermostats through voice. In the future, everything we use, from an air-conditioner to a car could get a voice-based assistant to manage it. So, there is no better time than now to get into building chatbots.

In this book, you will learn all about chatbots, how they work, and how to make your own chatbots. There are many software libraries available that can be used to build chatbots. Most of this book will focus on teaching you the principles of building a chatbot but I will be using Microsoft bot framework to show the examples. The ideas should be easily translatable to any other platform you decide to use. You will also learn how to connect them to multiple messaging platforms like Messenger, Slack and Telegram. We will also add analytics to track and measure the performance of the chatbot and learn how to use these analytics reports to constantly improve our chatbot.

My aim while writing this book is to make sure you get a complete understanding of the chatbot building process. By the end of this book you should be able to build chatbots that can be deployed to the real world. To this end, throughout the book I will introduce you to the practical aspects of building a chatbot along with the theory behind it. And you will learn about building a chatbot in the best way possible, by building a complete chatbot yourself, as you work your way through the book.

The chatbot that we will be building together will be called, “Stay Fit”, which will be a personal health assistant that will help you to stay fit by helping you work out and eat the right food. I know that the name might not be completely original or creative but we will make do with it. You can call it something else if you have a better name. The chatbot will be able to answer questions about nutritional information and tips on different exercises. It will also be able to send you daily notifications reminding you of your exercise and nutrition goals for that day. We will use both rule-based and AI based approaches and connect it to external API and databases. Finally, we will deploy our chatbot onto a public server and connect it to messaging channels like Messenger, Skype and Slack. There are many great chatbot ideas that we can implement but I am choosing this because I believe that its easily relatable for many people, all of us want to stay fit and would appreciate some help. More importantly this chatbot will help me explain all the different factors involved in building chatbots. It might be the case that you are not very excited by this idea and want to build something else and that is totally understandable. In such cases what I suggest and I encourage every reader to do this is to follow along with the chatbot we build in this book and at the end of every chapter, apply the principles learnt in that chapter and try to apply them to your own chatbot idea. This will help you better understand the principles and also build the chatbot of your choice.

In this chapter, I will start by introducing you to chatbots and the Microsoft Bot Framework, which is a very popular programming framework for building chatbots and the one we will use in this book to build our chatbot, “Stay Fit”. I will also talk about a few alternative software libraries that can be used to build chatbots. Finally, I will walk you through building a simple echo chatbot so you get the hang of the process. An echo chatbot is a bot that replies or echoes back the message sent to it. Then, in each successive chapter, we will add some functionality to it, and analyze any problems that exist or features that are missing. I'll use those opportunities to teach you about techniques to solve these problems, add in the missing features, and then we'll use these techniques to improve our chatbot.

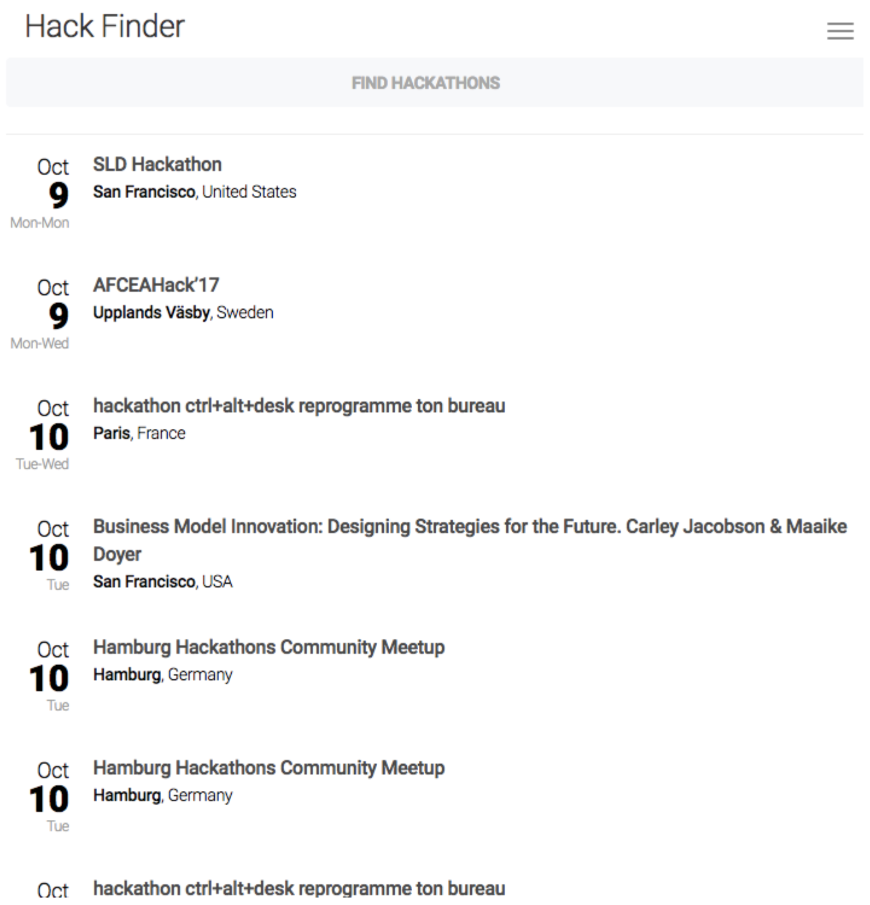
I would also advise you follow along with building the chatbot and understand all the pieces of code that we use. A suggested method is to first read the chapter and then try to write the code yourselves and only look back into the book if you are stuck, rather than copy pasting the code. Even if you copy paste the code make sure you understand what each line does. The final project will be available on GitHub at https://github.com/GanadiniAkshay/StayFit-Manning and if you face any issues or have questions you can ask them by opening issues on the GitHub repository. But before we start building the chatbot let's learn a bit more about what chatbots are, and what they can do.

1.1 Understanding Chatbots

A chatbot is a computer program or an application that you interact with in a conversational way via a chat or a voice interface. A chatbot can communicate using text, images, visual card like interfaces and speech. It can either be powered by a fixed set of rules where it relies on pattern matching also known as rule based approach or use artificial intelligence to mimic a conversation with a real person. Using AI means instead of using menus and buttons like traditional websites or mobile applications or using a limited set of rules for interacting, you use a natural language like English in an intuitive way to ask the application to do a task. For example, you can ask a chatbot application to book you flights from one city to another on a given day by saying or texting something like "I want to fly from NYC to Orlando next Monday". The application would then book the flight tickets or tell you price options. This replaces the traditional way of interaction where you have to select options from menus and click buttons. This is a more natural way for most people to use the service as in this case the conversation is similar to what you would tell your travel agent in a regular conversation.

1.2 Chatbots vs Websites & Mobile Applications

Let's look at an example to understand how chatbots work and how they can help us. Figure 1.1 shows a website which lists all hackathon events. You can browse through the list of hackathons or even apply filters for city, date and hackathon type from the menu.



Hack Finder

FIND HACKATHONS

Oct 9 Mon-Mon — **SLD Hackathon**
San Francisco, United States

Oct 9 Mon-Wed — **AFCEAHack'17**
Upplands Väsby, Sweden

Oct 10 Tue-Wed — **hackathon ctrl+alt+desk reprogramme ton bureau**
Paris, France

Oct 10 Tue — **Business Model Innovation: Designing Strategies for the Future. Carley Jacobson & Maaike Doyer**
San Francisco, USA

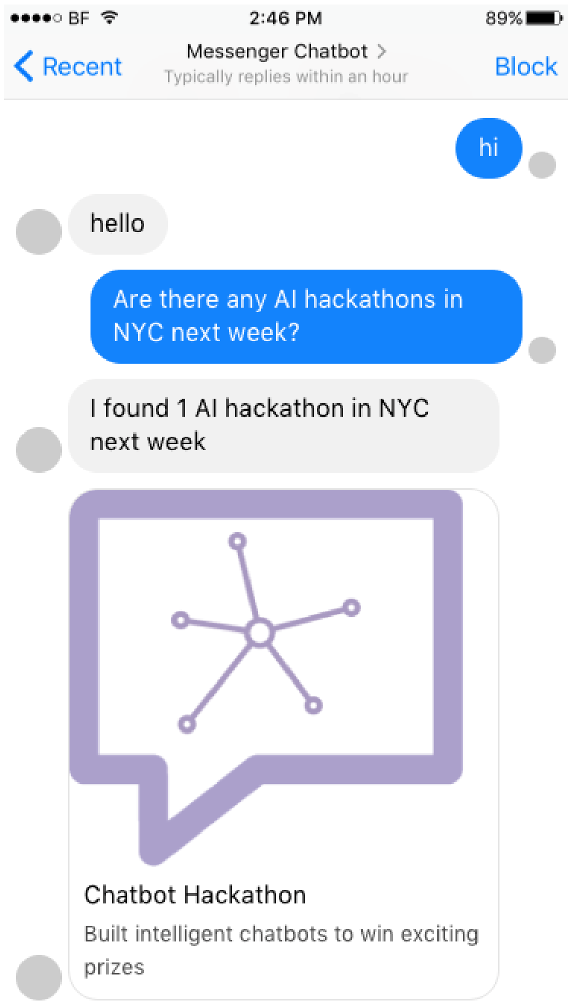
Oct 10 Tue — **Hamburg Hackathons Community Meetup**
Hamburg, Germany

Oct 10 Tue — **Hamburg Hackathons Community Meetup**
Hamburg, Germany

Oct — **hackathon ctrl+alt+desk reprogramme ton bureau**

Figure 1.1 A website showing a list of hackathon events requires users to scroll through a long list or use multiple menu items to filter the results.

However, a chatbot could accomplish the same goal in a more efficient way. You could just ask it to find a hackathon by theme, place and date naturally like you would ask a friend. The conversation between a user and chatbot to find a hackathon is shown in Figure 1.2.



hi

hello

Are there any AI hackathons in NYC next week?

I found 1 AI hackathon in NYC next week

Chatbot Hackathon

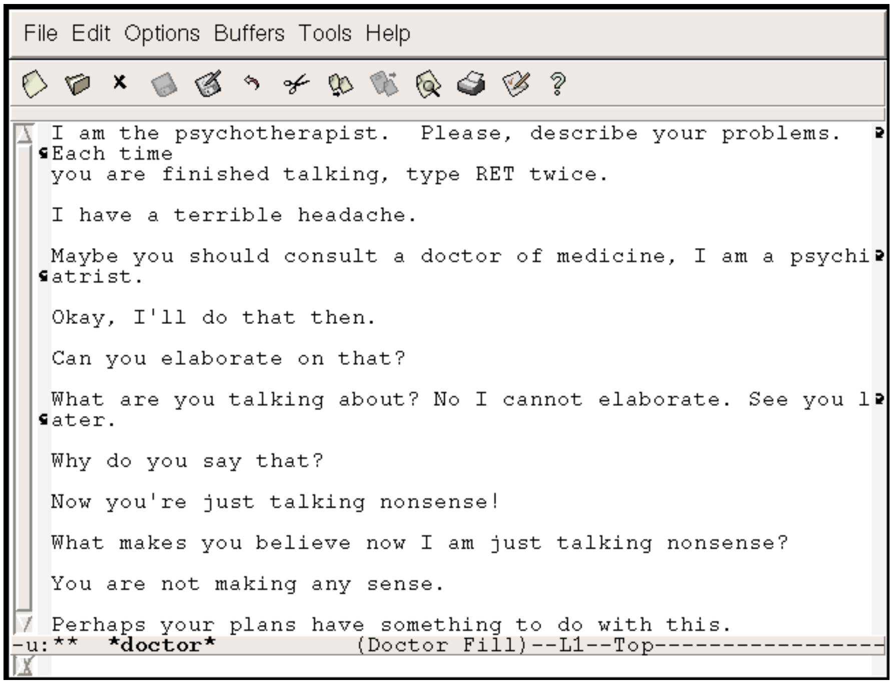
Built intelligent chatbots to win exciting prizes

Figure 1.2 A sample conversation between a user and a chatbot to find a hackathon is more natural than searching on a traditional web site.

As you can see, a chatbot gives you a more natural and efficient way of using a service, avoids a tedious interface with menus, lists and dropdowns, and lets you use natural language to perform any operation you want.

1.2.1 Evolution of chatbots

Although the huge growth in interest in the field of chatbots is pretty recent, people have been building chatbots since early 1960's. One of the most famous early chatbots was Eliza, which was created in 1966 by Joseph Weizenbaum. Eliza simulated conversation with a real human by using a pattern matching and substitution method. One of the most famous scripts that used the pattern matching method for Eliza was the DOCTOR script which simulated a psychotherapist and used rules to respond with non-directional questions to user queries. The script was good enough to give users an illusion of understanding on part of the program. Figure 1.3 shows a screenshot from the original Eliza program running the DOCTOR script.



```
I am the psychotherapist.  Please, describe your problems.
Each time
you are finished talking, type RET twice.

I have a terrible headache.

Maybe you should consult a doctor of medicine, I am a psychi
atrist.

Okay, I'll do that then.

Can you elaborate on that?

What are you talking about? No I cannot elaborate. See you l
ater.

Why do you say that?

Now you're just talking nonsense!

What makes you believe now I am just talking nonsense?

You are not making any sense.

Perhaps your plans have something to do with this.
-u:**  *doctor*          (Doctor Fill)--L1--Top-----------------
```

Figure 1.3 A screenshot from the original Eliza program running the DOCTOR script where the program simulates a psychotherapist

However, the program was still quite basic, and couldn’t reach farther than a short conversation. Since then multiple attempts have been made to build chatbots to imitate a human conversation, but we simply lacked the technological knowledge for it to become a reality. Also, most of the work that had happened in this space had taken place in academic research labs as a curiosity, and not a lot in the commercial sphere.

But things have changed in the last few years, owing to two main factors. One is the rapid development in the field of deep learning which can help us build the artificial intelligence required to mimic human conversations. And the second, and more important, factor is the explosion of messaging applications like Messenger, Slack, WhatsApp etc. According to reports by BI Intelligence (https://www.businessinsider.in/Messaging-apps-have-finally-caught-up-to-social-networks-in-user-numbers-and-now-dominate-mobile/articleshow/46879368.cms), a research organization that publishes research reports and newsletters, the number of monthly active users using messaging applications has now surpassed the monthly active users on social networks. This means that people spend more time inside email and messaging platforms than they do inside other applications on their phone. This shift has created a huge market opportunity and has transformed the work in this field from being purely academic into a more commercial nature.

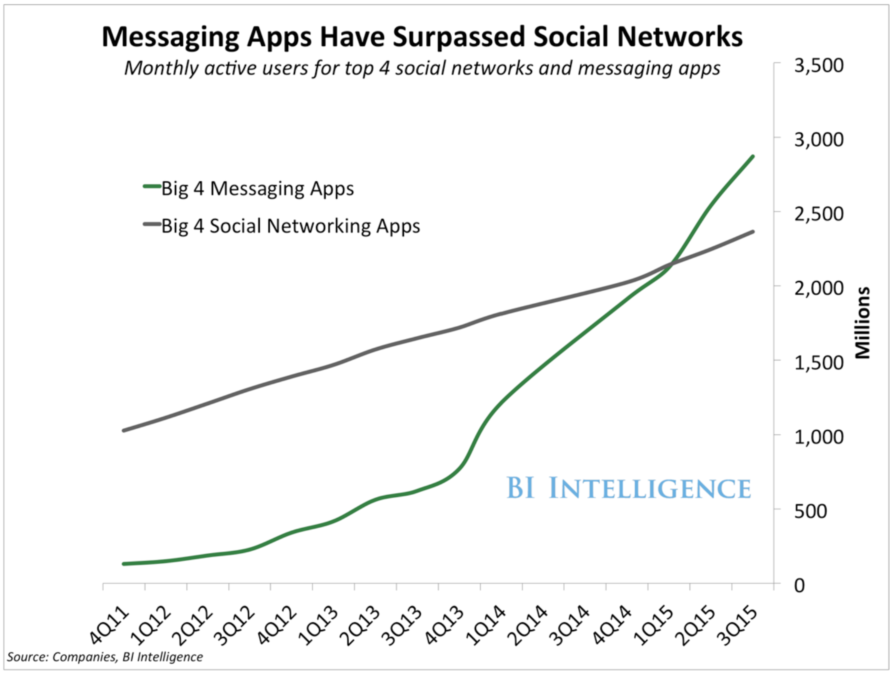


Figure 1.4 BI Intelligence report showing messaging apps surpassing social networks in monthly active users

1.2.2 Why Chatbots?

A valid question to ask at this point is, why do we need chatbots? How are they better than traditional applications? After all, a lot of research and work has gone into the field of building beautiful and efficient User Interfaces (UI) accompanied by great User Experience (UX) for traditional applications. So, are chatbots a significant improvement over them and do they provide real benefit to the user over traditional applications or are they just a science project fad? These are all good questions to ask, not just for chatbots but for every emerging technology. It’s important to focus on technologies that can give real benefits rather than spend energy and resources on science projects.

I personally believe that chatbots are going to have a huge impact on how our future shapes up. Since the industry interest in chatbots is fairly recent, there are no commonly agreed upon conclusions, so we have to take any criticism or praise for the technology with a little consideration. But I believe they are going to have a big impact due to two main reasons. One is a purely business reason and the other is a more philosophical one.

The business reason is that people are tired of installing and using multiple apps. This is also more popularly known as App Fatigue. There are apps for every brand, every task and every event. An Apple or Android user has a choice of over 2 billion apps. Apps are complicated: they need to be downloaded, they take up memory on the device, it takes time and effort to understand the interface (it took me quite a while to understand the interface for Snapchat), they need updates, and so on and so forth. If you are a business with a mobile application one of the biggest challenges is to get people to install your app. Assuming you cross this hurdle, the next challenge is to get them to actually open the app and spend time in it.

However, as you can see in Figure 1.3, users already spend most of their smartphone time inside a messaging app like Messenger, Telegram, Kik etc. And Chatbots take advantage of this use by *giving businesses an opportunity to provide their services to the users of these messaging applications without having them install a separate application*. If you build a chatbot on Facebook Messenger, for example, you immediately have access to over a billion people who have Facebook Messenger installed on their phones and can use your services or application from right inside of Facebook Messenger.

A related business benefit for chatbots comes with the growth of Augmented Reality/Virtual Reality and Wearable technologies like HoloLens, Oculus and other smart glasses and smart watches. All of these technologies are a move away from traditional screens so they also require a new kind of interface. In fact, we are already beginning to see this shift happening. One example is the recently launched apple airpods, the wireless earphones that let you use the voice chatbot Siri (to control music and other settings. In the future, this could be opened up to developers and other use cases might come out of it.

A more philosophical reason for the potential usefulness of chatbots is the accessibility it would provide to users. As of 2016 only 40% of the world population has access to an internet connection. It can also be assumed that most of the remaining population doesn't have any experience using traditional applications with Graphic User Interfaces (GUIs). In the next decade, as more and more of these people start coming onto the internet, they will face a huge challenge of learning the interface i.e. knowing how to use menus, swipes, which buttons to click etc. This would mean that businesses would have to spend extra resources and time on educating this new set of users who are having their first experience with the GUI. Chatbots however avoid this problem by relying on something that most of these new users already know, a natural language.

Imagine, say, a farmer in India who, instead of having to learn the UI of a farming mobile app, can just ask google now or any other voice chatbot on his phone, in his own regional language like Telugu or Kannada, to help him find the right pesticide. The chatbot could then respond to his query in his own language. There are other times where the user interface might be so complicated that it requires training to use it. For example, something like Adobe Photoshop. Even after multiple years of trying I haven't been able to master Photoshop and I am pretty adept at using technology. I have always had to resort to finding someone who has experience using photoshop and tell them how I wanted the picture to be edited and hope they would do it. But imagine if you could just tell Photoshop how you wanted your picture to look and it did it. This will enable everyone to create beautiful pictures and amazing digital art without the burden of having to learn the tool.

1.2.3 Sample use cases for chatbots

Although in theory chatbots can be used to replace any traditional software, there are some use cases where chatbots really shine. A good way to decide if you need a chatbot for any service is by asking the following questions:

1. Does it replace a real human conversation?
2. Is the GUI too complicated and does it force users to go through multiple selections of categories and sub-categories?

The first question is pretty straightforward. If users are accustomed to having a conversation with someone to solve their problem, for example calling customer support, you can make the process more efficient by using a chatbot to have the conversation instead of a real person. Sometimes there might not be a direct translation to users actually having real conversations but instead an indirect relation. The Indian farmer from the previous section is a good example of such a scenario. Traditionally farmers might use resources like books or guides to find the right pesticide. So, in that sense we are not really replacing a conversation exactly. But the indirect connection here is that farmers probably talk among themselves or with friends to discuss and share or learn about these pesticides. Our chatbot can then act like a friend to the farmer and help solve his queries.

The second question is a little trickier. You have to ask yourself if the user has to go through a lot of menus before they get to where they want and if replacing that process with a conversational interface make it easier. A good way to assess this is to observe how new users react to your application. Some things to look for are: Is there a learning curve to using the interface? Does the user need training (as in the case of Photoshop)? Does it take multiple interactions by the users to achieve a simple task? If any of these are true then it's probably a good idea to consider creating a chatbot to make the full or at least part of the application easier to use.

Some great examples of chatbot use cases are:

- A customer support chatbot – Instead of talking to or texting with a real person, users can be directed to a chatbot that can handle the common questions and as a fallback be routed to a real person if the chatbot is unable to handle a query. This can let businesses cut down on resources for customer support and also make it more efficient. Currently users are frequently frustrated due to the long wait time before you have a person who can answer the query. With automation, most of the queries can be instantly handled and human intervention only needed for new or complicated queries and because most of the human support personnel will be free a lot more, the wait times for these complex query answering will also go down.

- A chatbot for e-commerce – Chatbots can also be used for selling things. In this case they can act like the sales person in the shop who helps customer decide inside an online store. Thus, bringing a more personalized in person store experience to online stores.
- A chatbot for personalized financial/medical/legal advice – Chatbots can be used to provide advice or help with issues where you would generally talk to a consultant. Some examples of this are money saving advice, legal paperwork filing advice or advice with dosage and side effects and nutrition.
- A chatbot for personalized movie/restaurant/food/event recommendation – Chatbots can also be used to get recommendations for food and events. These recommendations could be scheduled like daily or weekly or they can be based on some external triggers like current location.

1.2.4 Different types of Chatbots

As I said earlier, chatbots can be of two types, either rule based or AI based. Apart from this there are also some other variations in chatbots. Let's take a look at these variations, because understanding their distinctions can us decide what kind of chatbot would be the best for our use case.

RULE BASED VS AI BASED BOTS

Rule based bots use a set of rules to process incoming messages. They do not understand the meaning or context of a message. They are simple question answer bots which use simple pattern matching which is based on a fixed set of rules to answer questions. It is relatively cheap and fast to implement these chatbots. Because they can't understand the meaning or context they must be explicitly programmed to handle the different ways a user might communicate with it. These bots generally rely on buttons and menus to get user input to limit the input variations from the user. Figure 1.5 shows an example of a rule based bot.

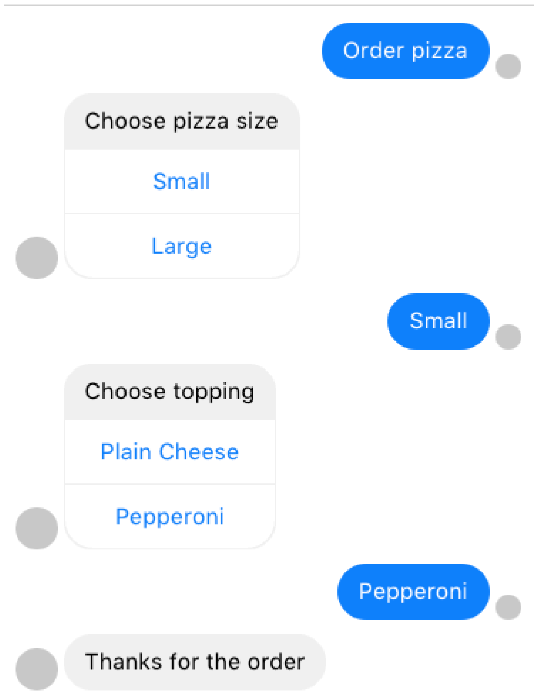
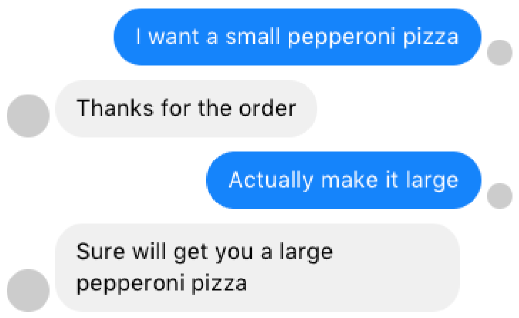


Figure 1.5 A sample rule based bot where you can order a pizza by clicking buttons instead of natural language text input

AI based chatbots, in contrast, use Natural Language Processing (NLP) to understand the meaning of the input message, also known as intent. They can also extract other relevant information like city name, time or any other contextual information that might be needed for handling the request. The process of extracting this contextual information is also called entity recognition. These bots generally use freeform text input, as they are capable of handling the various kinds of user requests. Figure 1.6 shows an example of an AI based bot handling the same pizza order.



I want a small pepperoni pizza

Thanks for the order

Actually make it large

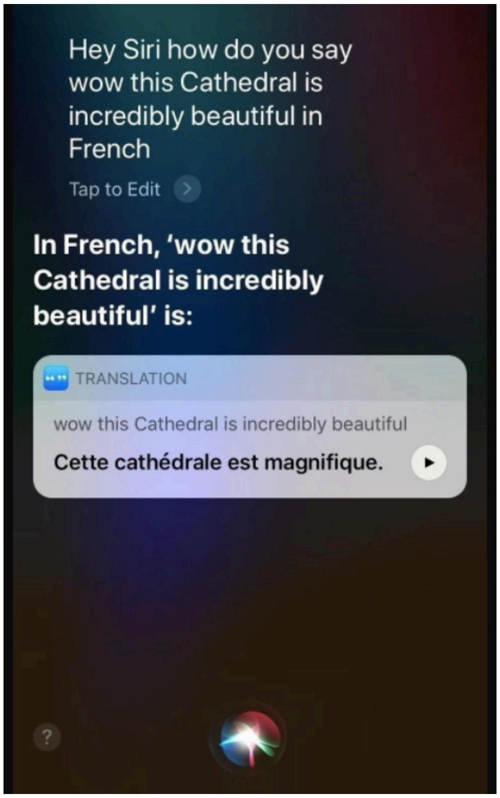
Sure will get you a large pepperoni pizza

Figure 1.6 A sample AI based bot where you can order pizza with natural language text input

VOICE BASED CHATBOTS VS TEXT BASED CHATBOTS

Chatbots can either use voice or text as a way to interact with the users. In case of voice chatbots, the voice part can be thought of as an add on to a regular chatbots. A voice bot would use speech to text to convert user's voice requests to text and then use speech synthesis to convert the reply text into voice and respond with that voice. The chatbot part of it still essentially remains the same with just voice added as a new component. Some popular examples of voice based chatbots are Siri, Cortana and Alexa. Figure 1.6 shows an example of Siri, a voice based chatbot. In contrast, a Text based bot uses text to interact with users as shown in Figure 1.5.

In this book we will be focusing on Text based chatbots primarily. If you are interested in learning about building voice based chatbots, Manning has an excellent book on building voice applications for Alexa and Google Assistant called "Voice Applications for Alexa and Google Assistant" written by Dustin A Coates.



Hey Siri how do you say wow this Cathedral is incredibly beautiful in French

Tap to Edit

In French, 'wow this Cathedral is incredibly beautiful' is:

TRANSLATION

wow this Cathedral is incredibly beautiful

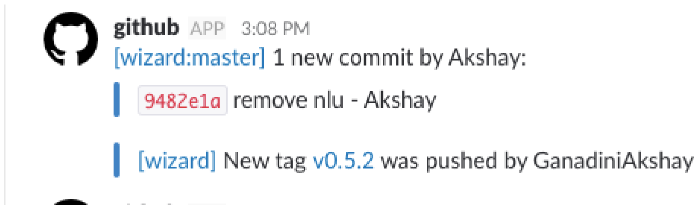
Cette cathédrale est magnifique.

Figure 1.7 A sample conversation on Siri asking for a French translation – an example of an open ended chatbot

PRIVATE VS GROUP BOTS

Private or personal bots communicate directly with an individual user on a one-to-one basis. These bots can provide personalized responses based on the user they are communicating with. Both the pizza ordering bots showed in previous section are examples of private bots.

Group bots, in contrast, communicate with multiple users at the same time. Some platforms like Slack, Kik and more recently even Messenger, let you build bots that can communicate with multiple users inside a channel or a group chat window. Figure 1.7 shows the slack GitHub bot which sends message on a channel to everyone on the channel.



github APP 3:08 PM
[wizard:master] 1 new commit by Akshay:
9482e1a remove nlu - Akshay

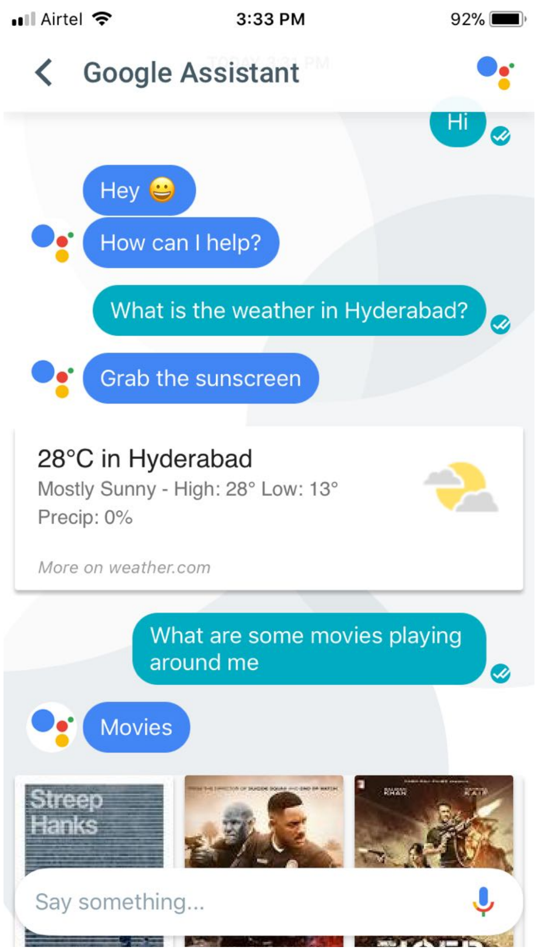
[wizard] New tag v0.5.2 was pushed by GanadiniAkshay

Figure 1.8 The GitHub bot on Slack which updates the team of new changes to code repositories – an example of a group bot

CLOSE ENDED VS OPEN ENDED BOTS

Close ended chatbots solve only one problem or only a set of related problems associated with a specific domain. The pizza bot examples we saw previously are good examples of a close ended chatbot. They are great for ordering a pizza but if you ask it to book you tickets to a movie, the chatbot will not be able to handle it. They also can hold only limited generic conversations, meaning they are not able to do much small talk or other aspects of normal human conversation.

Open ended chatbots on the other hand can solve problems in multiple domains and are usually better at having generic conversations as well. Some examples of open ended chatbots are Siri, Cortana and Google Assistant which have both a voice and text component. However, I have to emphasize that it’s not necessary that open ended chatbots be voice based. Figure 1.8 shows a sample conversation with Google Assistant.



Airtel 3:33 PM 92%

Google Assistant

Hi

Hey 😀

How can I help?

What is the weather in Hyderabad?

Grab the sunscreen

28°C in Hyderabad
Mostly Sunny - High: 28° Low: 13°
Precip: 0%

More on weather.com

What are some movies playing around me

Movies

Figure 1.9 A sample conversation on Google assistant an example of an open ended chatbot asking about weather and movies

It is important to understand that is very hard to build a perfect open ended chatbot because by definition of being open ended there are a lot of different categories of queries that the bot needs to handle. Platforms like Alexa and Google Assistant are examples of open ended chatbots which aim to answer questions from multiple domains. These platforms let developers add additional functionality to the platform to allow the chatbot to answer new and different type of questions. For example a developer can add the functionality where these platforms can answer questions about food recipes. The idea is that if a lot of developers add functionalities for different domains then over time the platform itself might become more and more open ended. On Alexa, these extensions are known as skills and on Google Assistant they are called actions. For these reasons, we as developers are mostly only concerned with building close ended chatbots for solving a particular problem or adding our chatbot as an extension to the existing open ended chatbot platforms.

1.3 How do chatbots work?

The two basic types of chatbots, rule-based and AI-based, work in different ways. Rule- based chatbots generally use a decision-tree based structure. This type of bot is very limited as it only understands a few specific commands, and the bot is only as smart as it is programmed to be. These bots generally use buttons to guide the user through the conversation. Chatbots that use Artificial Intelligence can understand language--not just specific, defined commands.

Even though there are two ways in which chatbots process and respond to inputs, the workflow is similar for both of them. The differences show up in how they do the processing step.

Basically, any chatbot is essentially just a web service which can respond to HTTP requests. A high-level overview of how they work is show in Figure 1.8

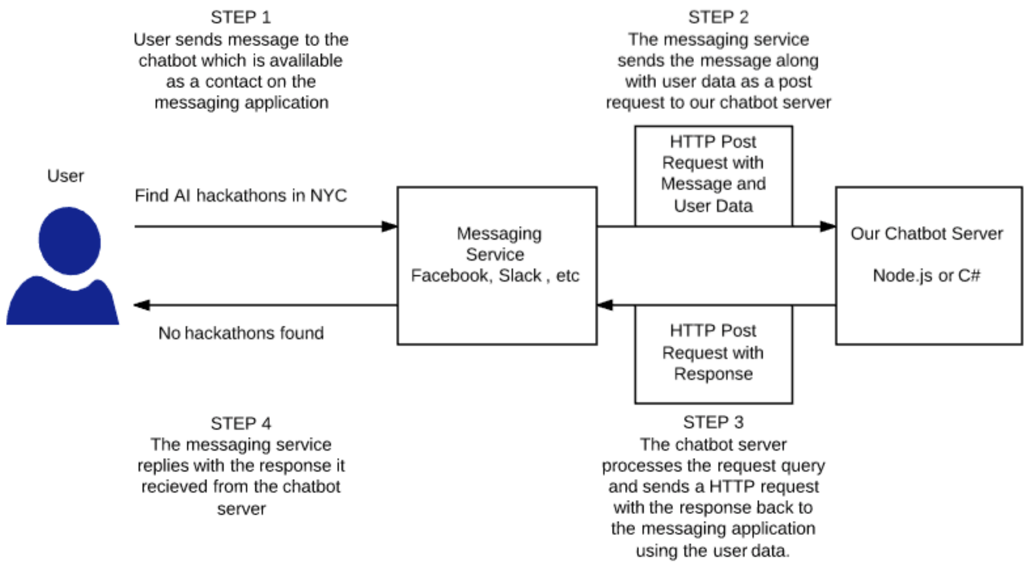

STEP 1
User sends message to the chatbot which is avalilable as a contact on the messaging application
STEP 2
The messaging service sends the message along with user data as a post request to our chatbot server
HTTP Post Request with Message and User Data
User
Find AI hackathons in NYC
Messaging Service Facebook, Slack , etc
Our Chatbot Server
Node.js or C#
No hackathons found
HTTP Post Request with Response
STEP 4
The messaging service replies with the response it recieved from the chatbot server
STEP 3
The chatbot server processes the request query and sends a HTTP request with the response back to the messaging application using the user data.

Figure 1.10 A Flow chart explaining how chatbots work from a high level for a simple query where a user asks to find AI hackathons in NYC

STEP 1

In the first step in Figure 1.9 a user sends a message, “Find AI hackathon in NYC”, to the chatbot on the messaging service. The messaging service could be Messenger, Slack, Skype or even a website. From the user's perspective, this is similar to sending a message to a contact or friend on the messaging service. The messaging service sends the message and the user data in the form of a HTTP Post request to the chatbot server. As chatbot developers, there is not much we have to do in this step apart from configuring a chatbot client on the messaging service and linking it with our server. Also, the configuring is a one-time operation we have to do while setting up the bot. Even though most of the processing in this step is taken care by the messaging service it helps to understand what is happening in the background.

STEP 2

In the second step our chatbot server receives the HTTP Post request. In the example shown in Figure 1.9 this request "Find AI hackathon in NYC". The post request also contains information about the user like the username, time zone of the user etc. The chatbot server is then responsible for taking this request, processing it, generating a response and then sending the response back to the messaging service. This is the main part of the chatbot as this is where all the processing happens.

STEP 3

In most cases the response is sent as a separate HTTP request rather than as a response to the original request. This allows you to batch incoming requests and send a single response based on them or in the opposite way send multiple responses to a single request. In our example, if a user makes a request to find hackathons in a given city, rather than making the user wait until your service finds the hackathon list from a database, you can send an immediate response saying you got his request and are working on it and then a few seconds later send the list of events once you have retrieved them. You can think of this as the conversational equivalent of a loading screen in traditional websites or apps. To keep things simple for this example we will assume that our chatbot server replies with, "No hackathons found" after processing the request.

STEP 4

Once the response is sent to the messaging service, it forwards the response as a reply to the appropriate user. From a user's perspective, this feels like a normal conversation on a messaging channel where they send a message and get a reply.

In the next section, we'll take a deeper look at the heart of the chatbot application--the chatbot server—and how it processes the request and generates the response.

1.3.1 Inside the Chatbot Server

The chatbot server, in step 2 in figure 1.10, is the piece responsible for responding to the user requests. It needs to perform three steps, which you can see in Figure 1.9

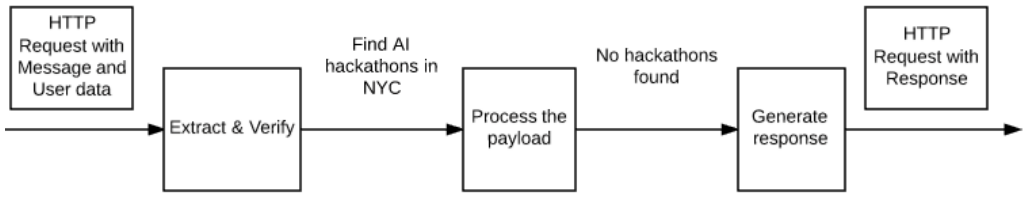

HTTP Request with Message and User data
Extract & Verify
Find AI hackathons in NYC
Process the payload
No hackathons found
Generate response
HTTP Request with Response

Figure 1.11 The three steps involved in the request processing on the chatbot server. Extracting and verifying the request, processing the request and generating a response.

STEP 1 – EXTRACT & VERIFY

The first step in the process is to receive the HTTP request, verify it and then extract the payload from it. Before extracting the payload it's important to verify if the request is actually coming from the messaging service and if it's valid. This is to make sure the request is coming from a real user request rather than some malicious person trying to perform a man in the middle attack to your chatbot or trying to use your chatbot's backend server with their chatbot. Most messaging services give you a way to authorize the request being sent often by using App Secret Keys. We will learn about using the App Secret Keys when we try to connect our chatbots with different messaging channels. Since the keys are secret and not public they can be used for validating a request. The validation process may differ from one messaging service to another, and you don't need to worry about the details of the exact validating process for now. We will discuss more about them when we look at connecting our chatbot to the messaging channels.

The HTTP payload contains information about the message, the user and other metadata that might be required to handle the request. In our example in Figure 1.10 the payload will contain the message, "Find AI hackathon in NYC" and also information regarding the user who made the request. Extraction is however not as straightforward as it seems initially. Every messaging service has its own payload format, i.e. the JSON structure or schema is different for every messaging service. The JSON payload also contains information about the type of request. Messaging services often send additional requests apart from the user message to inform you about things like the user reading a message you sent, a ping request to verify the server etc. Your extraction code should be able to deal with all these variations in the request and handle them appropriately. Additionally, sometimes users might send non-text data like images, videos, files etc. All of these are sent as file attachments and your extraction code should be able to handle them. Listing 1.1, shows an example of the payload sent by Messenger for the message in our example.

Listing 1.1 JSON payload sent by messenger when user sends a text message saying, "Find AI hackathon in NYC"

```
{
      "object": "page",
      "entry": [{
        "id": "<page id>",
        "time": 1458692752478,
        "messaging": [{
                "sender": {
                        "id": "<user id>"                       (1)
                },
                "recipient": {
                        "id": "<page id>"
                },
                "timestamp": 1458692752478,
                "message": {
                        "mid": "<message id>",
                        "text": "Find AI hackathons in NYC"  (2)
                }
        }]
      }]
}
```

(1) **Information about the user who made the request**
(2) **The message sent by the user**

As you can see in 1.1 the payload contains information, in this case the user id which can be used to get more information, about the user in the sender section inside the entry object. It also contains the actual message sent in the message section inside the entry object. Similarly, an image, video, file sent also have their own payload structures. Also, other messaging services have different payload structures. An authorization token is also sent as part of the request which uses the secret keys and this token can be used to validate the request. The process of extraction is to basically consume this JSON structure and pull the required information out into variables to be used later.

STEP 2 – PROCESS THE PAYLOAD

The second step is to use the data extracted from the payload and use that data to generate a response. In our case we extract the query as "Find AI hackathons in NYC". Now we need to process the message and generate a response for that query. In this step, we handle the request by first understanding the request and then performing some computation to get the results for the query. We will take a deeper look into understand the query with intent and entity recognition in chapters 3 and 4.

STEP 3 GENERATE RESPONSE

The third step in the request processing by the chatbot server is the response generation. Similar to the first step in the chatbot server's processing this step is not as straightforward as it initially seems. That's because every messaging service requires the post request to be sent in a different JSON schema. So, your code should be able to generate the response JSON in the appropriate format as required by the messaging service. Also, these formats could change from time to time and you need to update your code to reflect these changes. Listing

1.2 shows an example of the JSON payload for sending a text response back to messenger. Luckily Bot Framework automatically handles this for us.

JSON payload for sending a text response back to messenger

```
{
     "recipient": {
         "id": "<Page Specific User ID>"
       },
       "message": {
         "text": "No hackathons found"
       }
}
```

Finally, after the response is generated we need to send it back to the messaging service from where we received the initial request.

1.4 Introduction to Microsoft Bot Framework

Now that we have a good idea of what chatbots are and also a high-level overview of how they work let’s learn how to actually build them. In this book, we will learn about building chatbots and implement the examples using Microsoft Bot Framework.

Microsoft Bot Framework is a software framework that helps you easily build, test and then deploy a chatbot to multiple messaging channels. The framework is available in C# and Node.js. It provides powerful tools to solve a lot of the common challenges involved in chatbot development. Here is a list of some of the tools the library provides.

- Chatbot I/O that deals with how the messages are received and how we can respond with replies to the appropriate messaging channel.
- Easy NLP integration for AI based chatbots to make our bots more human like and intuitive to use.
- Dialog classes to manage conversational flow. There might be a certain flow of conversation that our bot needs to handle in a given order. The dialog classes help us manage this flow of dialog.
- Prompts to accept and process user input to questions to get required data to solve a query for the user. These can be thought of as the conversational equivalents of traditional form fields where users are asked to fill in data.
- Databags to store and reuse state from the conversations to maintain context and remember things previously mentioned or inferred from the conversation. This can be thought of as an in-memory database that our chatbot has access to.
- Response classes to send images, cards, carousels and attachments. This helps us reply with more than just text as might be needed in some cases.
- Easy way to connect to multiple messaging applications

Do not worry if none of these terms make sense to you at this point of time. We will learn about each of these in more detail later in the book. I just wanted to get you excited about some of the features of Microsoft Bot Framework.

You can think of Microsoft Bot Framework as a software library i.e. a NuGet package for the C# version and a npm module for the Node.js version. I have purposefully kept the introduction short so as to not overwhelm you with details. I believe that the best way to learn about something is through practice. We will learn about specific features about bot framework while trying to add new features and capabilities to the chatbot but before that lets try to understand why we need a new software library or a framework to build the bots in the first place.

1.4.1 Why Bot Framework?

At this point you might be wondering why we need to use a separate framework for building chatbots. After all, from what we learnt in the previous sections, a chatbot is basically a web service. So why can't we just build a web service using any of the popular tools like Node.js or Flask which is a python framework. The answer to that question is that yes, we can build a chatbot using just Node.js or Flask without any external library. But using a library like the Microsoft Bot Framework makes it much easier for us to build a bot than it would be without a library. So, in theory even though we can build a bot by making a simple node app that listens on a HTTP port it is not particularly an efficient way of doing it. This is because there are many things that a bot needs to do apart from just listen for messages and return a reply.

For example, we would have to write code to do all the following:

1. Verify the incoming POST requests. This is to make sure the request is from the messaging service by using the secret keys and not by a malicious program or person.
2. Handle different message formats like text, image, emojis, files, locations etc. Users might send a message in multiple formats and our chatbot needs to be equipped to handle them.
3. Send replies in multiple formats. There might be cases where our bot might have to reply with an image or a video based on the query and we need to make sure our bot can send replies in this format based on the guidelines given by each messaging channel.
4. Maintain state to remember things that were previously mentioned or inferred from the conversations and use them to make the chatbot more natural. Asking for the same information that was previously mentioned or could be easily inferred from previous conversations could also frustrate users in some cases.

Adding code for all the above features will make the code messy and hard to manage. Also, we need to keep in mind that the code for the above features will vary from one messaging channel to another as each have their own set of APIs and guidelines to integrate the bot into their messaging channel.

A library like Microsoft Bot Framework does a lot of heavy lifting for us. It handles the security aspects, it extracts the message information from the payload, it helps us maintain state and it can also generate the appropriate response based on the messaging channels being used. With Bot Framework, the same code can work for multiple channels and you do not have to worry about the changes made to the platforms or channels as the library updates would take care of that. As an added benefit, parts of the Microsoft Bot Framework are open source which means you can look at the code and even create a fork of the project and make any changes in the library you need which might be specific to your use case.

Although there are other frameworks available to build chatbots, I personally believe that Microsoft Bot Framework is one of the best because it is the most complete of all of them. It takes care of authentication, it can connect to multiple channels, it can help you maintain context easily, it has a simple dialog system. We will look at all of these features individually and see why they are very important and how Microsoft Bot Framework shines in these aspects in later chapters. However, if you are not comfortable with coding you might prefer to use some of the drag and drop based chatbot builders in the market. Also if you are more comfortable with another programming language say for example Python you might prefer to use a chatbot framework like Flask-Wizard.

1.4.2 The core parts of Microsoft bot framework

Microsoft bot framework has 2 core parts that make it a great tool to build chatbots. They are

- The Bot Builder
- The Bot Framework Portal

The bot builder is the set of language libraries or SDKs that will help you build the chatbot. The SDKs are available in Node.js and .NET platforms. Bot builder also provides an emulator which we can use to test and debug the bot locally before deploying to the messaging applications. The SDK is open source and is hosted on GitHub at https://github.com/Microsoft/BotBuilder.

The bot framework portal at http://dev.botframework.com is the admin panel to register and manage your chatbots. The portal lets you configure your chatbot to make it work with multiple channels or messaging applications. The bot framework portal currently supports the following channels.

- Facebook Messenger
- Slack
- Skype
- Skype Teams
- Telegram
- Twilio
- Email
- Bing

- Cortana
- Kik
- Web

The portal also gives you diagnostic tools to test your chatbot.

Apart from these Microsoft also provides a set of services called Cognitive Services which you can use to add intelligence to your chatbots. Cognitive services are a collection of AI services developed by the AI experts at Microsoft. In the case of building a chatbot we are mostly concerned with language understanding and cognitive services provides 5 APIs for language understanding. They are

1. LUIS – It helps us get the intent and entity information from the input message text.
2. Text Analytics – It gives us information about sentiments, topics and language.
3. Bing Spell Check – It provides powerful spell check capabilities.
4. Linguistic Analysis – It can be used for parts of speech tagging and understanding the structure of a given text.
5. Web Language Model – It uses advanced language models for tasks like word frequency and next word prediction.

Cognitive services also provide APIs for Knowledge extraction, Speech recognition, image and video processing which can be used to add extra capabilities to our bot if our use case needs them.

1.4.3 Node.js vs .NET

As discussed earlier, Microsoft bot framework provides SDKs for both .NET and node.js platforms, so you might be wondering how to pick the right one. The generally acceptable answer to such questions which ask for a choice of language or framework to use is to go with the one you are most familiar with. The same answer applies here. If you are only familiar with one of them you should go with it rather than trying to learn a new language. However, if you are familiar with both .NET and node.js, I would like to argue that node.js is a better choice.

1.4.4 Why Node?

Chatbots are a natural use case for asynchronous programming and Node is built to be async from the ground up. Also Node.js is powered by a single threaded event-based architecture which is perfect to build something like a chatbot where the whole application is event driven by design. Another not so technical reason to prefer node.js over .NET is the vast community support for building chatbots using node.js. This can be clearly seen in the fact that almost all the major frameworks available to build chatbots are either exclusively for Node.js or have a Node.js flavor. In this book, we will be learning about using the Node.js SDK to build chatbots.

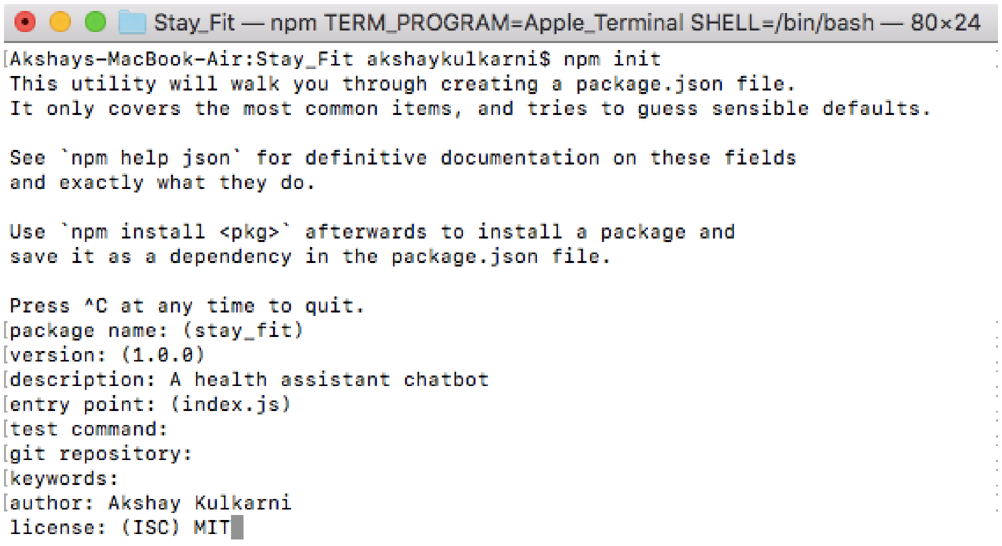
1.5 Building a simple echo chatbot

In this section, we will get our hands dirty by building our first chatbot using Microsoft bot framework. We will build an echo chatbot which will basically echo or reply with the same message that the user sends it. As we go further in the book we will add more features to it and turn it into the fully functional health assistant bot "Stay Fit" that we had planned to build at the start of this chapter. Before you can run this example, you will need to make sure you have node.js installed on your system. The instructions for installing node.js are in the Appendix.

First, we will create a folder called "Stay_Fit" where we will put the code for our chatbot. Next, we will change our directory to "Stay_Fit" and initialize npm by running

```
npm init
```

Fill up the options as shown in the figure 1.12 and your node project should be initialized. You can verify if your node project is initialized by checking to see if a package.json file is created in your project folder.



Stay_Fit — npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 80×24

```
Akshays-MacBook-Air:Stay_Fit akshaykulkarni$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (stay_fit)
version: (1.0.0)
description: A health assistant chatbot
entry point: (index.js)
test command:
git repository:
keywords:
author: Akshay Kulkarni
license: (ISC) MIT
```

Figure 1.13 Initializing npm inside the project

The next step is to install all the required libraries. For the simple echo bot, we will be using two libraries.

1. botbuilder – botbuilder is the bot framework library that provides you tools to build and manage conversational flows.
2. restify – restify is a node.js module that lets you easily run a http web server.

3. Run the following command to install the libraries

   ```
   npm install —-save botbuilder restify
   ```

The save flag is important as it adds these library names into the package.json file. This will help you easily deploy your node.js code on another system or server without having to transfer all the library files as well. These libraries can be installed on the new system by running `npm install`

Once the packages get installed we are ready to build our echo chatbot. We will be using the ES2015 coding standards to build our chatbot. ES2015, formally known as ECMAScript6 is the sixth major release of ECMAScript which is the "proper" name for JavaScript. The reason I am using ES2015 in this book is that I believe it is the future of JavaScript and also it is important to stay updated with the current trend.

Using ES2015, also known as ES6, rather than using ES5 which was the previous version of JavaScript adds a few new quirks with it. If you are confused with the different versions of JavaScript and are not sure which version you are using don't worry about it. For the sake of this book and building chatbots all you need to know is that ES2015 is the latest version of coding standards for the JavaScript language. Also, if you are not sure which version of JavaScript you are currently using, it is highly probable that you are using ES5. As mentioned earlier there are a few quirks and setups needed before we can run ES2015 code. We will learn about these in the next section.

1.5.1 Setting up our ES2015 project

Even though support for ES2015 is always increasing, it is not universally supported across browsers and servers. Some older versions of Node do not support ES2015 and won't be able to run code written in ES2015 format and if the server we publish our code to doesn't have the right configuration it might not be able to run our ES2015 code. There are two methods to solve this and make our code work across different machines without having to worry about configurations or the version of Node etc.

The first method is to pass a harmony flag while running our Node.js application. We would do this by using the following command to start our Node.js application.

```
node —-harmony index.js
```

The –harmony flag allows you to run code with ES2015 features that are not yet distributed as part of the standard Node.js installation.

Another method to run ES2015 code is by converting it to ES5. The process of converting ES2015 code to ES5 is called transpiling. Transpiling is important and necessary when we want to create browser applications using JavaScript. It is not strictly required for a server-side application like our case where we can get away by just using the –harmony flag. However, in the spirit of trying to learn new and interesting things we will be using transpiling as a method to run our ES2015 code.

USING BABEL

Luckily for us there are excellent tools available to convert our ES2015 code into ES5 or in other words transpile our code from ES2015 to ES5. One such tool is babel.

To start using babel we first need to install it as a dev dependency in our project. The difference between a dev dependency and a normal dependency is that dev dependencies are not installed on the servers or new machines where the code gets deployed. This is useful for libraries which are only required during the development phase and hence the name dev dependencies. Install babel as a dev dependency in our project by running the following command.

```
npm install —-save-dev babel-cli babel-preset-env
```

In the above line of code, we are installing two npm packages as dev dependencies. The first one called babel-cli lets us run babel from command line to convert ES2015 to ES5 and the second package called babel-preset-env is the module that is responsible for the transpiling from ES2015 to ES5. Once these packages are involved we need to create a configuration file in our project to let babel know which preset to use for transpiling. The preset basically gives babel information about the set of plugins needed to support particular language features. In other words, it tells babel about the source format and the destination format for the transpiling and the different plugins needed for this transpiling. I am sure this sounds confusing but for the purpose of this book and bots it's not required that you understand the specific details about how the transpiling process works. As long as you are able to set up the project and get it running it should be fine. However, if you are interested in learning more about this I encourage you to take a look at babel's documentation. Let's add the required configuration by creating a file called ".babelrc" in the root folder of our project and adding the contents shown in code listing 1.2.

Listing 1.2 Adding the preset information to ".babelrc" file instructing babel to use babel-preset-env to transpile ES2015 code to ES5

```
{
      “presets”:[“env”]
}
```

The final step to do after creating the configuration file is to set up npm scripts to run babel to transpile and execute our ES2015 code. We will create 2 scripts. The first script will be to build our code which will basically transpile our ES2015 code to ES5 and store it in a different directory. Add the build script by adding the following line into the scripts part of the "package.json" file.

```
"build": “babel src -d lib"
```

This command is telling npm to run babel to convert all ES2015 code in the "src" directory to ES5 code and store it in the "lib" directory. The -d flag stands for destination. This command can be used by running `npm run build` from the command line terminal in the current project.

The next script we will add is to first run babel to transpile the ES2015 code in the "src" directory and then run the transpiled ES5 code stored in the "lib" directory. To add this script add the following line inside the scripts part of the "package.json" file after the "build" script.

```
    "start": "npm run build && node lib/index.js"
```

In the above command we are combining the previous "build" script which stores the transpiled code in the lib directory and then running this code using the node lib/index.js command. After adding both the scripts to "package.json" we are done with the setup. Also delete the "test" script that gets added by default to the file. We will add a "test" script in a later chapter when we discuss testing our chatbot. For a sanity check make sure the contents of your "package.json" file look similar to code listing 1.3. Note that the versions of the modules might differ for you but that should not be an issue.

Listing 1.3 Package.json file after adding the required dependencies, dev dependencies and the npm scripts required to run the project.

```
{
  "name": "stay_fit",
  "version": "1.0.0",
  "description": "A health assistant chatbot",
  "main": "/lib/index.js",
  "scripts": {
    "build": "./node_modules/.bin/babel src -d lib",
    "start": "npm run build && node lib/app.js"
  },
  "author": "Akshay Kulkarni",
  "license": "MIT",
  "dependencies": {
    "botbuilder": "^3.14.1",
    "restify": "^7.1.1"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.6.1"
  }
}
```

This completes the initial setup of the ES2015 project but before we can start building our chatbot in the next section another useful but optional thing to set up is 'nodemon'. While coding our bot or any node.js application for that purpose, we will have to keep making changes to the code and retesting the app. In such cases it can get very annoying to manually keep restarting the server. This is where 'nodemon' comes in, it automatically restarts when we make changes to our source code. You can install nodemon by running 'npm install nodemon' and use it to run the server by calling `nodemon <filename>.js'.

However, in our case before we call nodemon we will have to call babel to convert our ES2015 (ES6) code to ES5 and we will also have to tell babel to do the conversion (transpiling) every time we make a file change. So let's add a new script in the package.json to do this by adding the code below in the scripts section of the package.json file.

```
"scripts": {
    "build": "./node_modules/.bin/babel src -d lib",
    "start": "npm run build && node lib/app.js",
    "dev": “nodemon --exec npm run babel-node -- ./index.js”
  },
```

1.5.2 Coding our bot

The first step is to create a “src” directory where our source code will live and add a file called index.js in the directory. This will be the main entry point to our chatbot application. Inside the file add the following code.

Listing 1.4 Echo chatbot built with Microsoft Bot Framework

```
import restify from “restify”                              ❶
import * as builder from 'botbuilder'                      ❶

// Setting up the server
const server = restify.createServer()                      ❷
server.listen(3978, () => {
  console.log(“${server.name} listening to ${server.url}”)
})

// Create chat bot
const connector = new builder.ChatConnector({              ❸
  appId: “”,                                               ❸
  appPassword: “”,                                         ❸
})                                                         ❸

const bot = new builder.UniversalBot(connector)

server.post(“/api/messages”, connector.listen())           ❹

bot.dialog(“/”, (session) => {                             ❺
  // Edit more dialog here                                 ❺
  session.send(“You said: “ + session.message.text);       ❺
})                                                         ❺
```

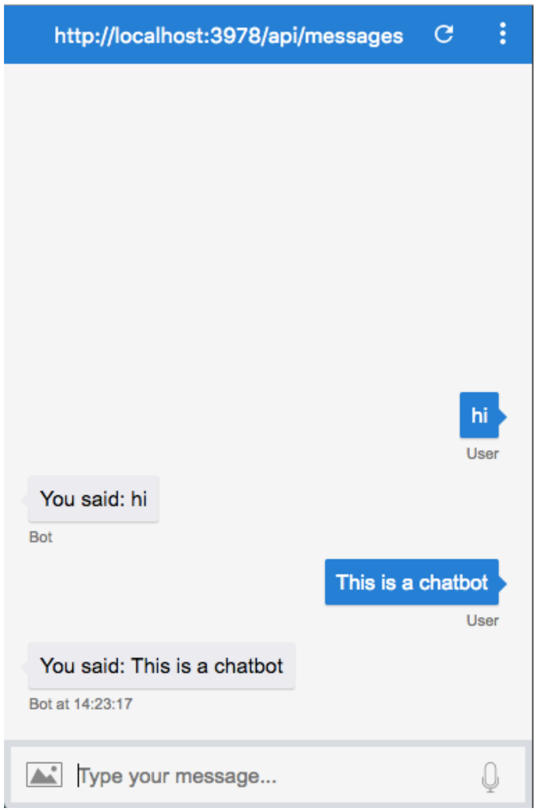
❶ **Importing the required libraries**
❷ **Creating a restify based HTTP server to listen for HTTP requests**
❸ **Connecting to the bot builder platform to deploy our bot to multiple channels. We are leaving it empty for now and will at a later stage.**
❹ **Listening to POST requests on this endpoint where messages will be sent.**
❺ **Echoing the message sent by the user back to them,**

Now run the code by running the following command in the terminal `npm run start`.

This will transpile the code and store the ES5 transpilation in the “lib” directory and run it.

To test the functionality of the bot we will need to install the Bot Framework Emulator from https://github.com/Microsoft/BotFramework-Emulator/releases and install the latest release. At the time of writing this book I used the version 3.5.34 of the emulator so if you use a different version your emulator might look a little different compared to the screenshots in the book. Detailed instructions on the installation process can be found in the Appendix. Once you have the emulator installed, open it and put in the URL, http://localhost:3978/api/messages and leave the AppId and AppPassword fields blank for now.

Now you can start testing the bot by typing in a message in the chat window as shown in figure 1.13. Your bot should now be echoing every message sent to it. You might see a warning pop up saying "The Bot State API is deprecated", ignore the warning for now. We will look at why we get that warning and how to fix it in the coming chapters.



hi

User

You said: hi

Bot

This is a chatbot

User

You said: This is a chatbot

Bot at 14:23:17

Figure 1.14 Echo chatbot in action

1.6 Looking Ahead

In this chapter we learnt about the basics of the chatbot, Microsoft Bot Framework and we also build a simple chatbot using ES2015 that can echo the message we send to it. This is clearly not very useful. Can you think of ways of how we can make our bot give proper responses to our questions and not just repeat what we ask it? We will talk about adding this capability to our chatbot in the next chapter.

1.7 Summary

- Chatbots are applications that you can interact with via a chat or speech interface.
- Microsoft Bot Framework is an SDK to build chatbots and it supports node.js and .NET platforms
- Microsoft Bot Framework has two core parts the bot builder and the bot framework portal.
- The bot builder is a set of language libraries with powerful tools to manage conversational flow and respond to users.
- The bot framework portal is an admin panel to register and manage your chatbot.

The bot framework portal lets you connect your bot to multiple messaging channels easily