Chatbot design supported by specialized platforms and frameworks

Arnaud Weyts

Aknowledgements

I want to thank these people...

Contents

Research question

The point of this bachelor thesis is to conduct research into chatbots and how they are built. What actually goes into building a chatbot that feels intuitive but is also functional?

1.1 Approach

This thesis will start with some use cases for chatbots. When it would be a good idea to build a chatbot as a solution for a concrete problem.

Already existing chatbots will be presented and compared to paint a picture of what a professional chatbot currently looks like. Some general research into chatbot best practices and worst practices will also be conducted based on these real life cases and studies.

Finally, different platforms and frameworks to create a fully-fledged chatbot will be compared by building similar small chatbots.

1.2 Metrics

The metrics for the comparison will be based on a couple of different factors. Their general business model will be researched, followed up by their learning curve and quality of documentation. Finally source code readability and scalability will be compared.

General chatbot design

2.1 Choosing a chatbot solution

2.1.1 Introduction

Why choose a chatbot as a solution to a problem? To find the answer to that question there should be a concrete description of the problem and research should be conducted into how that problem is currently handled.

2.1.2 Use case 1: Food delivery

Food delivery chatbots are some of the most common chatbots out there. That's because a basic chatbot for ordering food is easy to make and maintain. Most of the time they don't require complex language recognition and they will go through the same steps every time someone wants to order something.

A great example of this is Domino's. They own one of the most popular facebook messenger chatbots even though the bot itself is really simple.

Taking a look at the perspective of a new imaginary pizza place in town. This pizza place has a set menu with set formulas. They want to innovate and make it possible to receive and process online orders but don't want to lose the familiarity of their brand.

This is a perfect case for a chatbot. If they already have a facebook page, they can easily integrate a chatbot into it and promote it to their current customers. All they need on top of that is an admin panel for the company to maintain their menu and process orders.

Building a chatbot this way also opens up several possibilities for expansion in the future. They can easily start tracking customer's habits and improve their suggestions for specific customers.

There are some downsides to this solution as well however. Creating a chatbot from scratch for this purpose would cost a lot of money. That's why there are already solutions like Chatobook [1] popping up. This is an all-in-one solution that provides a restaurant with a messenger bot and an admin panel to manage promotion, reservations and the menu. You can also find templates for these kinds of bots that integrate with Google Sheets/Excel. [2]

The rise of general delivery services like Ubereats [3] and Deliveroo [4] also complicates this matter. If those companies manage to integrate full ordering services into their platforms using chatbots -which they will most likely succeed in at one point- that would simplify everything. But that also comes with a cost. The imaginary pizza restaurant would lose their unique identity as the chatbot's identity would be replaced with Uber's or Deliveroo's identity.

2.1.3 Use case 2: Banking industry

Next up is the banking industry. Lately there has been an influx of integrating AI-powered chatbots into this sector. This is because people like to engage with their bank and get answers that feel human, instantly. It establishes a form of trust. This form of communication is also very attractive to the banks themselves, because it allows them to quickly provide answers to repetitive questions.

There's lots of specific chatbot banking use cases out there. One of them is simply basic banking services. Checking account balance, transferring funds, analyzing expenditures. Chatbots can also hugely improve a bank's customer support, which should be available 24/7. They can help out people instantly, without long wait times which a lot of customer support services are currently struggling with. Further more, there's also the opportunity for intranet-based chatbots [5]. These chatbots communicate with the employees to improve their productivity and give them access to the right information in mere seconds. Another use case is the delivery of customer-focused marketing. Using a chatbot, it's easier to promote personalized banking services as if the customer is talking to a real life employee.

Overall, integrating a chatbot into a banking environment has lots of benefits. That's why more and more banks are starting to invest in them.

2.1.4 Use case 3: E-commerce

Lastly there's the quickly growing industry of E-commerce. This ties into both of the use cases above as the main focus is customer interaction and engagement, support and personalized marketing.

Online web-stores are one of the fastest growing industries out there, they are popping up everywhere. This is because more and more people are starting to buy online. It's convenient, fast and reliable (in most cases). However there is one key benefit missing compared to in-store shopping; interaction.

In comes the shopping assistant chatbot. It can do everything a normal store employee can and more. Like instantly retrieving your past orders to help you out quicker and more efficiently. It can have access to every single product the web-shop offers and instantly recommend one that fits your needs. It can seamlessly provide post-purchase customer support. All of this whilst not losing the brand identity of the web-store in question.

2.1.5 Conclusion

Chatbots are an obvious answer to lots of companies that are starting to innovate and are ready to experiment. It allows for them to establish their brand in a familiar and human way, whilst also increasing their in-company productivity and overall efficiency.

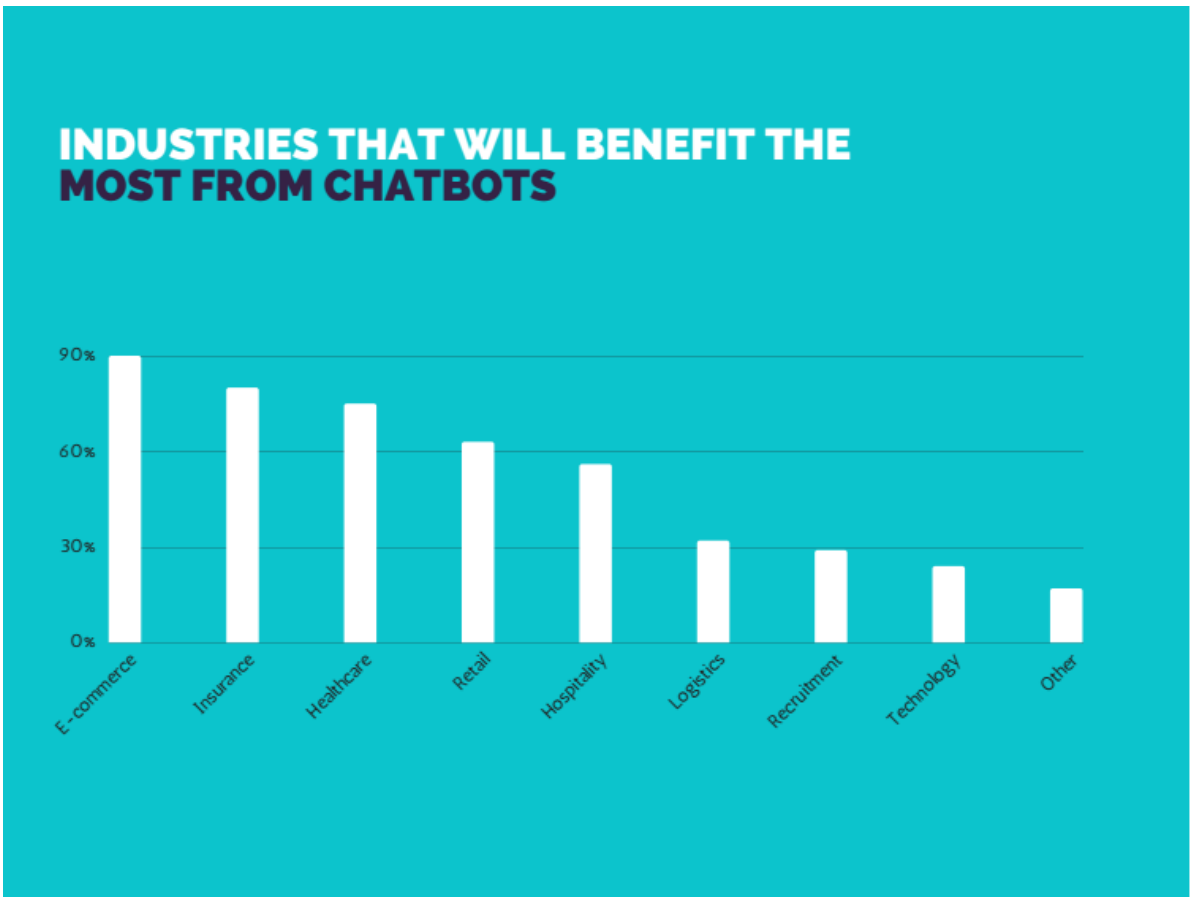

Figure 2.1: Industries projected to benefit from chatbots [6]

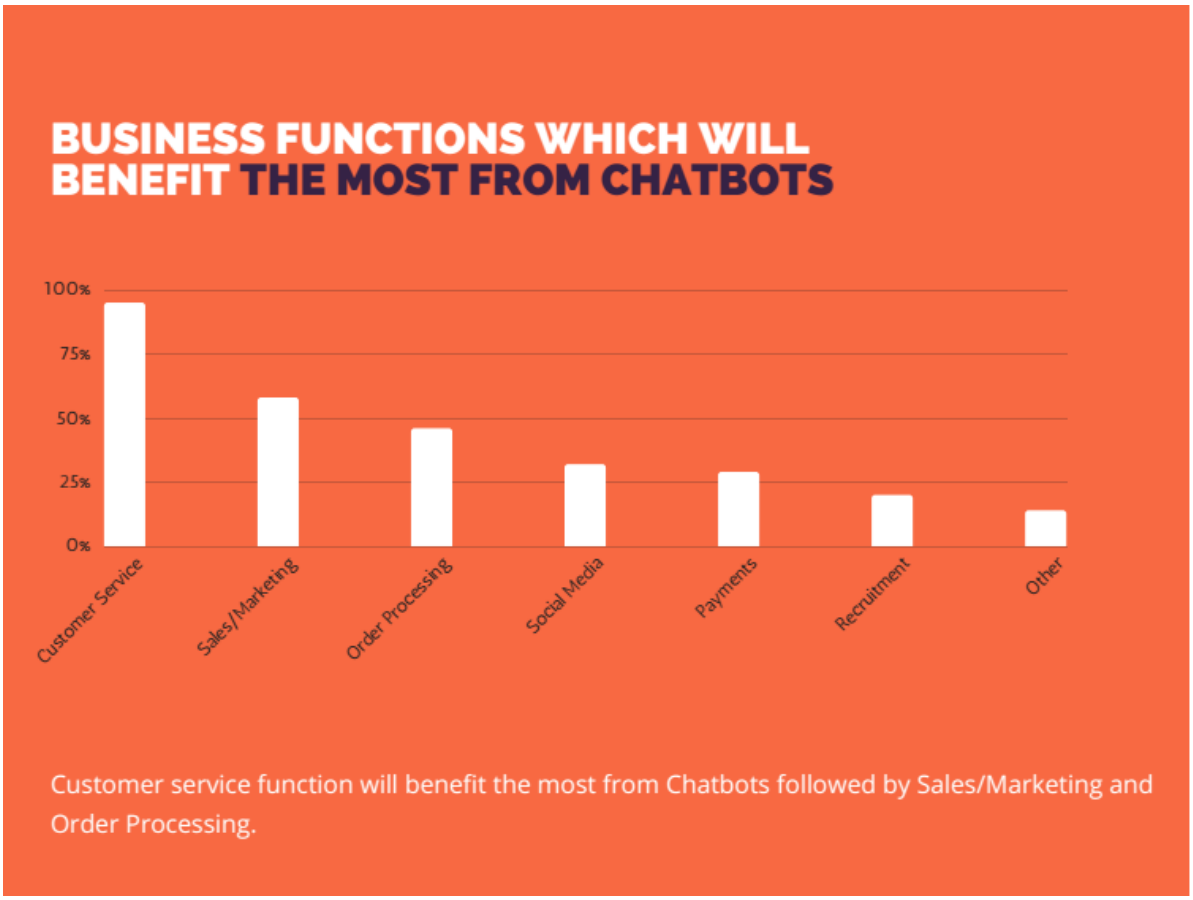

Figure 2.2: Business functions inside a company projected to benefit from chatbots [6]

2.2 Comparison of existing chatbots

Today there are plenty of well-implemented solutions already. There's food-ordering services like Domino's that offer an ordering service. But also stores, ranging from specific ones (H&M) to big, fully fledged web-stores like Bol.com, a Dutch-Belgian web-store.

2.2.1 Dom, Domino's ordering assistant bot

This is a facebook messenger bot for ordering food from Domino's, a fast food chain. Something that can immediately be noticed is the use of tappable quick reply answers. The bot starts of with some simple questions to establish the user's intent and some information about the user like the address and what store he or she wants to order from.

Once this has been established, the user can choose what he wants to order and checkout.

All in all, the bot is pretty simplistic. It takes orders and passes them through to the store. There's a lot of room for improvement here. The bot could in theory build a profile of you using your past orders and recommend you an order off the bat. It could also keep you updated on new items being released. On top of that it seems the bot isn't very smart yet. The user needs to follow the steps clearly and in the right order. In general, it doesn't mimic a real life conversation.

In conclusion Dom is a very utility-focused bot. It doesn't aim to connect with the user and spread Domino's brand identity, it's solely focused on the ordering aspect.

2.2.2 Bol.com, Customer support bot

This bot's main focus lies in customer service. It replaces the long list of frequently asked questions that most customers really don't want to go through to find their answer. The bot also keeps your past or current orders in mind and integrates them into the conversation.

The bot keeps a very friendly and familiar tone towards the user throughout the entire conversation.

The language recognition seems to be focused on single word recognition to link the question to an answer. It also keeps spelling errors into account.

2.3 Best and worst practices

2.3.1 Creating a personality

One of the most important factors in creating a chatbot is to develop a personality for it. This makes the bot a lot more engaging to the users and makes it harder for the user to get frustrated.

If there is already a company personality in place, the chatbot should get its traits there. If that's not the case, adding a name or a face, which doesn't necessarily have to be human, to it can already be a huge help.

2.3.2 Making the conversation flow

Conversations with a chatbot aren't as straight forward as they seem to be. Users can easily get distracted, overwhelmed or annoyed by your chatbot.

That's why research and a lot of testing should be conducted into the conversational flow. An bad example of this is not having any wait time between messages. This overwhelms the user as he doesn't have time to read the previous message. Adding typing bubbles between messages not only gives the user time to read the previous message, but it also humanizes the bot.

2.3.3 Predicting failure and user behavior

Predicting failure and user behavior is hard in any user interface. But there are some common mistakes in chatbots that can be avoided before testing.

There are several ways to handle unexpected user input. One of them is revisiting the previous state of the conversation, another one is restarting the conversation, or asking the user politely what he's trying to accomplish.

Conversations that end up in this path should also be logged for research afterwards to avoid further dead ends.

When using buttons in a bot, to for example visit a website, or start the conversation, they should be functional at all times.

Facebook's messenger platform supports 2 main buttons for their chatbot. General buttons, and quick replies. Regular buttons will always be clickable multiple times. Quick replies are bound to the message they are meant to follow up on, and disappear as soon as they are clicked.

Of course this best practice goes hand in hand with actual user testing and applies to every user interface, not just chatbots.

2.4 Case study

To compare the frameworks, a simple chatbot will be designed and created employing the techniques above.

The chatbot will be a food ordering chatbot that offers the user a menu and takes in orders, similar to Domino's bot.

2.5 Conversational state diagram

To shape the bot, a conversational diagram needs to be created first. This diagram explains the main flow of the conversation, and the steps to reach the eventual purpose of the bot. In this case ordering food.

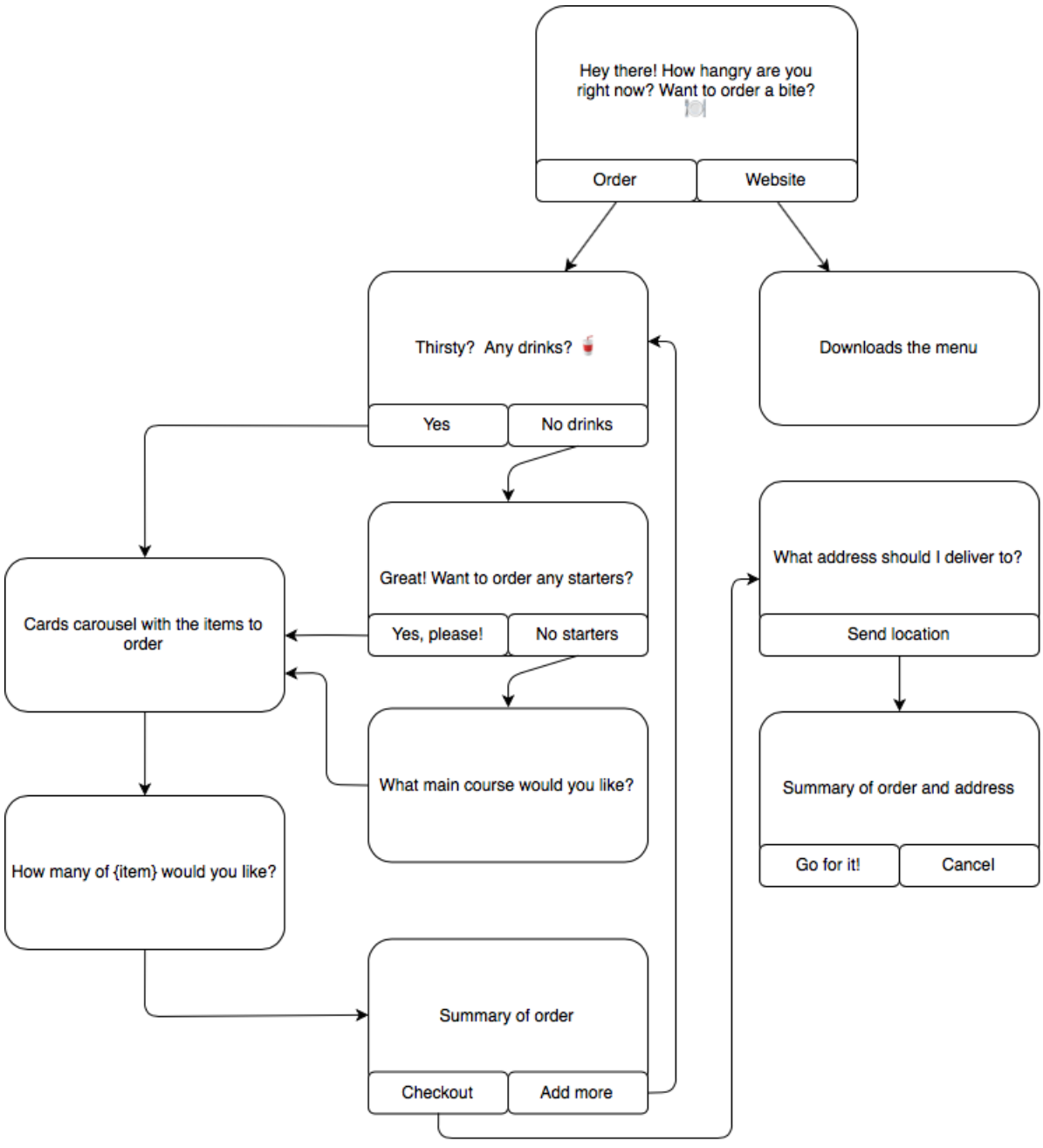

Figure 2.3: Conversational flow diagram for the food ordering bot

Microsoft Bot Framework

The BotBuilder SDK is an open-source solution developed by Microsoft to write a chatbot in once and make it possible to connect that same chatbot to multiple channels. Examples are Skype, Slack, Facebook Messenger, etc.

3.1 Business Model

The Microsoft Bot Framework can be used in several ways. The base platform can be self-hosted on a server of choice. But to access the extra services, like spell checking, or recognizing user intents using Microsoft's LUIS (Language Understanding), an Azure subscription plan is required.

Azure is Microsoft's Cloud service that can be tailored to the customer's need. The customer can decide what modules or extra features he wants to add and calculate the price.

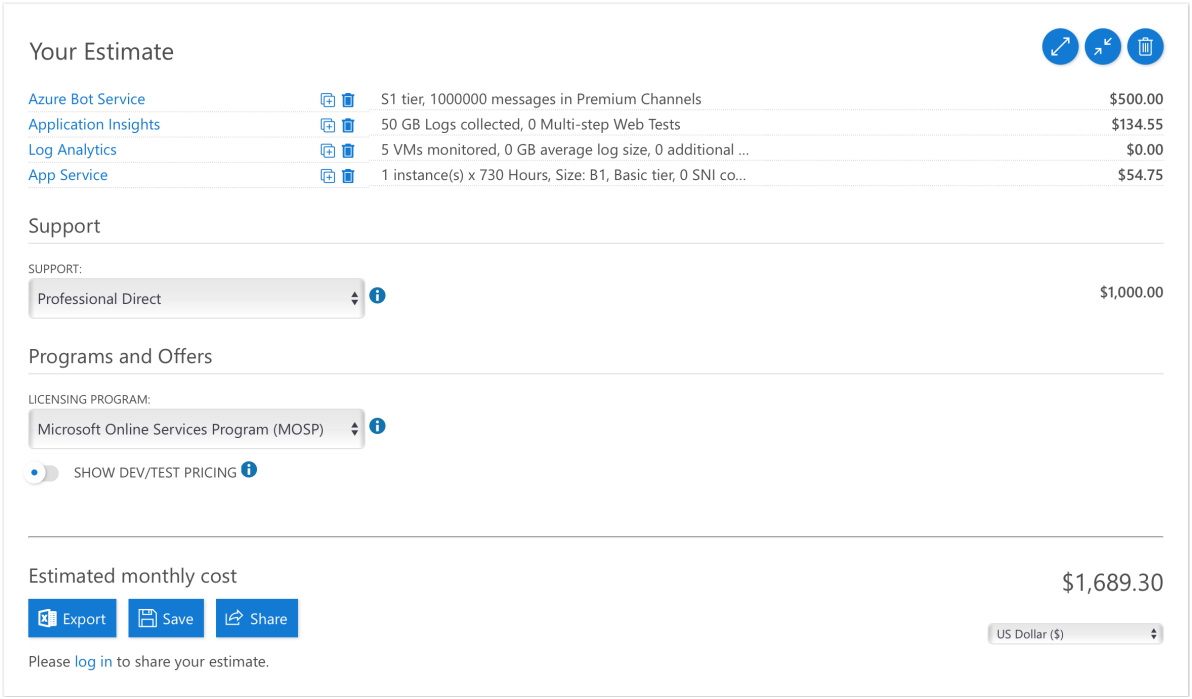

Figure 3.1: Microsoft's Azure Pricing Calculator [7]

3.2 Technical implementation

Microsoft provides the bot framework for 2 languages: Node.JS and C#. If C# is used however, development can only be done inside of a Windows environment, or the limited online code editor.

One solution would be to use Node.JS in a code editor of choice. Preferably Visual Studio Code, Microsoft's cross-platform code editor. This way everything can be tailored and configured to the developer's needs.

3.2.1 Developer environment

Setting up a developer environment can be configured from scratch to be as complicated and complex as needed by the company or developer. Node.JS is an open-source, cross-platform JavaScript environment that can be run on a server. And in the end that's what a chatbot is, a sever API that responds to requests (messages or events from the user).

It's important to note that JavaScript or also called ECMAScript is a language that is advancing very quickly and Node.JS cannot keep up with all the newest yearly features. And all of the documentation involving the Bot Framework is written in ES5. However it's still possible to use all of the newest features of ECMAScript if you compile your code down the ES5.

The documentation on setting up an environment like this is very limited. Microsoft does not provide any instructions or boilerplates on how to set up an efficient environment. This allows for more freedom but increases the learning curve for a developer who wants to start coding a bot using newer ECMAScript features or the option to hot reload his code.

Project structure

Structuring the project is straight forward. Just like most production JavaScript projects there is a source folder containing all of the actual source code. Developers can decide to divide the test files into a separate folder but generally it's recommended to keep the test files close to the source file they reference.

The node_modules folder is a dependency folder, this contains any dependencies needed to build the project. There are two types of dependencies: Firstly there are developer-dependencies, these are dependencies needed to build the project and participate in development. An example is eslint, these are a set of specific language rules configured for the project to follow. There are also regular dependencies, these are external libraries the project might use to do calculations, or wrappers for certain technologies. The Microsoft Bot Framework works using their botbuilder SDK as a dependency.

Next up are the config files, these are the workhorses of the developer's environment. There are 2 separate configuration files for the developers to build and test the bot as for the production server to build a functional output file. Webpack is a very powerful, well documented build tool and allows for easy customization. It is commonly used for front-end development but after some tweaking it can be used for Node.JS development as well. The development config starts a server that hot-reloads any changes made to the source files. This way it's easy for the developer to seamlessly check his changes instead of manually recompiling and restarting the server. The production config is very straightforward and simply compiles a minified and optimized output file into the build folder.

Further more there is support for environment variables using the dotenv dependency. This file is not versioned to git and will contain any API keys used in the project.

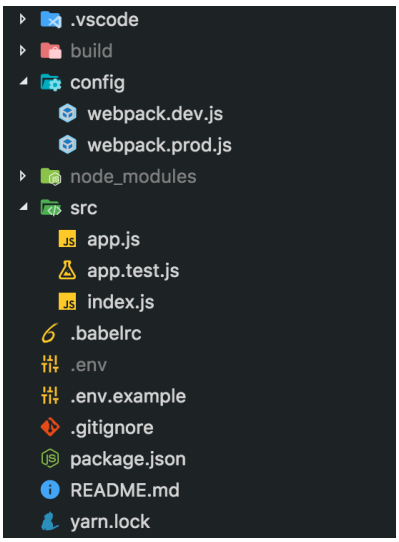

Figure 3.2: Project structure

3.2.2 Testing

The bot can be tested locally or remotely on any platform using Microsoft's own Bot Emulator [8], which is also open-source. This provides the developer with a live testing interface and information about everything the bot receives, including requests and the data that comes with them.

Speech Recognition is also supported for speech enabled bots. Sending system activities is another useful feature because it allows the developer to emulate user events like for example a user joining the conversation. Lastly, payment processing is supported as well to emulate a transaction.

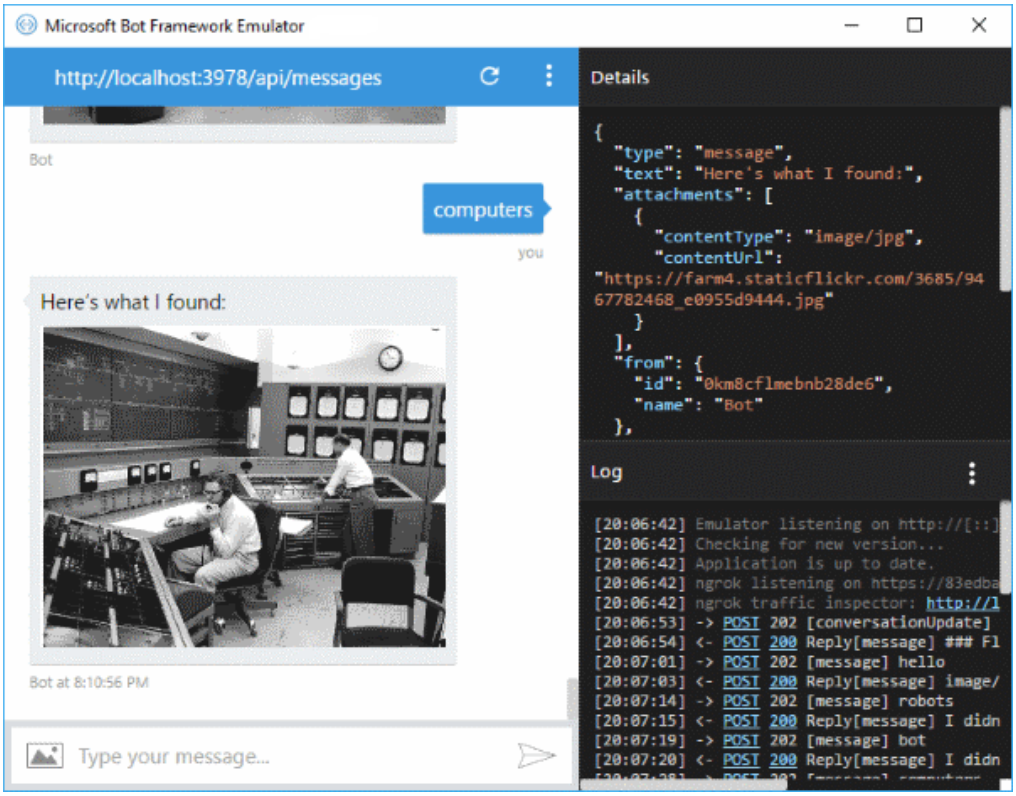

Figure 3.3: Microsoft's Bot Emulator [8]

3.3 Building the bot

3.3.1 Initialization

Initializing the bot is straightforward, the official documentation recommends restify as the server. During initialization it's also possible to define a default message the bot will always use as a reply if no other reply options match.

```
1  import restify from 'restify';
2  import * as builder from 'botbuilder';
3
4  // Setup Restify Server
5  const server = restify.createServer();
6  server.listen(process.env.port || process.env.PORT || 3978, () => {
7    console.log('%s listening to %s', server.name, server.url);
8  });
9
10 // Create chat connector for communicating with the Bot Framework Service
11 const connector = new builder.ChatConnector({
12     appId: process.env.MicrosoftAppId,
13     appPassword: process.env.MicrosoftAppPassword
14 });
15
16 // Listen for messages from users
17 server.post('/api/messages', connector.listen());
18
19 const bot = new builder.UniversalBot(connector, (session) => {
20   session.send(
21     "Sorry, I didn't get that. Type 'help' if you need assistance or try a
          different sentence.",
22     session.message.text,
23   );
24 }).set('storage', inMemoryStorage);
```

Listing 3.1: Initialization of the chatbot

To make the bot start the conversation certain events can be used. In this case a 'conversationUpdate' event will be used to trigger the root dialog.

```
1  bot.on('conversationUpdate', (message) => {
2    if (message.membersAdded) {
3      message.membersAdded.forEach((identity) => {
4        if (identity.id === message.address.bot.id) {
5          bot.beginDialog(message.address, '/welcome');
6        }
7      });
8    }
9  });
```

Listing 3.2: Initial message

3.3.2 Event types

- message
- conversationUpdate
- contactRelationUpdate
- typing
- ping
- deleteUserData
- endOfConversation
- event
- invoke
- messageReaction

The previous listing lists all of the available events during a conversation. A message event simply represents any communication between bot and user. The conversationUpdate event is triggered when any members are added/removed from the conversation, including the bot. Or when conversation metadata has changed. The contactRelationUpdate event indicates that the bot was added or removed to a user's contact list.

Most of these events can be triggered using the bot emulator or in code and sending it to the bot. This last one is useful for unit testing.

```
1  const event = {
2    address: { bot: { id: '0' }, user: { id: 0 } },
3    agent: 'botbuilder',
4    source: 'facebook',
5    sourceEvent: '',
6    type: 'conversationUpdate',
7    membersAdded: [{ id: '0', isGroup: false, name: 'test' }],
8    user: { id: '0', isGroup: false, name: 'test' },
9  };
10
11 connector.processEvent(event);
```

Listing 3.3: Sending a mock event to the bot

3.4 Dialogs and waterfalls

One of the key concepts in the Bot Builder SKD are dialogs. Dialogs help the developer manage the conversational logic and is a fundamental part of developing a chatbot in the bot framework.

A dialog can be compared to a function. It can perform a specific task and be called at any point in time. Dialogs can also contain other dialogs to advance and branch off a conversation.

A waterfall is an example of a dialog that guides the user through a set amount of steps or tasks. A great example of this is ordering food. First the user will be asked for drinks, followed up by the amount, again followed up by what starters the user wants and so on. There is a clear structure the bot goes through, hence the name Waterfall.

A waterfall is defined by creating an array of consecutive functions or waterfall steps inside of a dialog.

3.4.1 Prompts

To help the bot wait for a reply Microsoft provides developers with so called 'Prompts'. These already contain some extra functionality to validate the user's reply. A prompt should be used whenever the bot expects a reply from the user.

Prompt types

- text
- confirm
- number
- time
- choice
- attachment

When used inside of a waterfall dialog, the reply of the user will be passed to to next step using the 'results' object. Another method called 'next' can be used from the arguments to skip to the next step of the waterfall.

```
1  bot.dialog('orderButtonClick', [
2    (session) => {
3      builder.Prompts.choice(session, 'Thirsty? Want to order any drinks?', 'Yes
           |No drinks', {
4        listStyle: builder.ListStyle.button,
5      });
6    },
7    (session, results, next) => {
8      if (results.response.entity === 'Yes') {
9        session.beginDialog('orderDrink');
10     } else {
11       next();
12     }
13   },
14 ]);
```

Listing 3.4: 2-step waterfall using a prompt

3.4.2 User actions

There's multiple ways to handle user input other than prompts. Another way of doing this is using actions.

Triggeraction

The most common action is a 'triggerAction'. This can be connected to a dialog to invoke it whenever the user inputs a matched term. Whenever this is triggered, the dialog stack is cleared and the invoked dialog will be the new first dialog on the stack.

This behavior isn't always desired when the conversation needs to be temporarily redirected and resumed later on. That's where the 'onSelectAction' option comes in, to add the matched dialog to the existing stack.

BeginActionDialog

A specific dialog can also be attached to a dialog using the 'beginActionDialog' function. This can be useful to send the user to a contextual action. An example of this is when the user asks for specific help in a dialog context.

CustomAction

Unlike all the other actions, a customAction does not have any default action defined. It's up to the developer to define what it should do. The main benefit of using these is providing quick answers to a user without manipulating the dialog stack at all.

Further more a customAction does not bind to a dialog, instead it binds to the bot itself.

ReloadAction

The reloadAction is pretty self-explanatory. Binding this to a dialog means it will restart the dialog whenever the action is invoked. It's also possible to pass arguments to this action for the dialog to receive.

CancelAction

This action cancels the dialog it is bound to when triggered. The parent dialog will also receive an indication the child dialog was canceled.

EndConversationAction

Similar to the cancelAction action, the endConversation action will end the entire conversation when triggered. This clears the entire dialog stack and persisted state data.

Important to note

Important to note is the fact that some of these actions are quite interruptive to the conversation's flow. To make sure the user really wants to clear the entire dialog stack and start over, a confirm prompt can also be added to these actions.

Google Dialogflow

4.1 Business Model

4.2 Technical implementation

4.2.1 Developer environment

4.2.2 Testing

List of Figures

References

[1] (2017), [Online]. Available: https://chatobook.com/ (visited on 2018-03-20).

[2] (2018), [Online]. Available: https://botmakers.net/chatbot-templates/pizza-delivery-chatbot-for-facebook-messenger (visited on 2018-03-20).

[3] (2018), [Online]. Available: https://www.ubereats.com (visited on 2018-03-20).

[4] (2018), [Online]. Available: https://deliveroo.co.uk (visited on 2018-03-20).

[5] (2018), [Online]. Available: https://acuvate.com/solutions/meshbot/ (visited on 2018-03-26).

[6] (2017), [Online]. Available: https://reviewkiss.com/2017-year-chatbot-booming/ (visited on 2018-03-26).

[8] (2018). Microsoft bot emulator, [Online]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-debug-emulator (visited on 2018-03-28).

[19] (2018), [Online]. Available: https://www.just-eat.com (visited on 2018-03-20).

Bibliography

[7] (2018). Azure pricing, [Online]. Available: https://azure.microsoft.com/en-us/pricing/#explore-cost (visited on 2018-03-28).

[9] J. Verplanken. (2018). Learning ux design? build a chatbot., [Online]. Available: https://www.invisionapp.com/blog/learn-ux-design-build-chatbot/ (visited on 2018-03-13).

[10] F. Teixeira. (2017). Why chatbots fail, [Online]. Available: https://chatbot.fail (visited on 2018-03-13).

[11] N.N. (2018). Chatbots & conversational ui, [Online]. Available: https://uxdesign.cc/chatbots-conversational-ui/home (visited on 2018-03-13).

[12] W. Fanguy. (2017). What every designer can learn from alexa, [Online]. Available: https://www.invisionapp.com/blog/every-designer-learn-alexa/ (visited on 2018-03-13).

[13] J. Toscano. (2016). The ultimate guide to chatbots, [Online]. Available: https://www.invisionapp.com/blog/guide-to-chatbots/ (visited on 2018-03-13).

[14] I. Jindal. (2017). Building chatbots: Why message length matters, [Online]. Available: https://www.invisionapp.com/blog/chatbots-message-length/ (visited on 2018-03-13).

[15] D. B. Sera Koo. (2016). Chatbots: Your ultimate prototyping tool, [Online]. Available: https://www.invisionapp.com/blog/chatbots-prototyping-tool/ (visited on 2018-03-13).

[16] D. Wright. (2018). Us messenger usage for top brands 2018 march, [Online]. Available: https://medium.com/@davidwright_68835/us-messenger-usage-for-top-brands-2018-march-d4f1ef786622 (visited on 2018-03-13).

[17] D. Faggella. (2017). 7 chatbot use cases that actually work, [Online]. Available: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (visited on 2018-03-19).

[18] V. Chettupalli. (2018). Important use cases of ai-powered chatbots in the banking industry, [Online]. Available: https://medium.com/swlh/important-use-cases-of-ai-powered-chatbots-in-the-banking-industry-19115411af54 (visited on 2018-03-19).

[20] E. ". Seo. (2017). 11 more best ux practices for building chatbots, [Online]. Available: https://chatbotsmagazine.com/11-more-best-ux-practices-for-building-chatbots-67362d1104d9 (visited on 2018-03-28).

[21] (2017). What are the best practices for chatbot development? [Online]. Available: https://chatbotsmagazine.com/which-are-the-best-practices-for-chatbot-development-91c1b05fdb07 (visited on 2018-03-28).

Appendices

Botframework developer build script

```
const webpack = require('webpack')
const path = require('path')
const nodeExternals = require('webpack-node-externals')
const StartServerPlugin = require('start-server-webpack-plugin')
const Dotenv = require('dotenv-webpack')

module.exports = {
  mode: 'development',
  entry: [
    'webpack/hot/poll?1000',
    './src/index'
  ],
  watch: true,
  target: 'node',
  externals: [nodeExternals({
    whitelist: ['webpack/hot/poll?1000']
  })],
  module: {
    rules: [{
      test: /\textbackslash.js?\$/,
      use: 'babel-loader',
      exclude: /node_modules/
    }]
  },
  plugins: [
    new StartServerPlugin('server.js'),
    new Dotenv(),
    new webpack.NamedModulesPlugin(),
    new webpack.HotModuleReplacementPlugin(),
    new webpack.NoEmitOnErrorsPlugin(),
  ],
  output: {
    path: path.resolve('./build'),
    filename: 'server.js'
  }
}
```

Botframework production build script

```
const webpack = require('webpack')
const path = require('path')
const nodeExternals = require('webpack-node-externals')

module.exports = {
  mode: 'production',
  entry: [
    './src/index'
  ],
  target: 'node',
  externals: [nodeExternals()],
  module: {
    rules: [{
      test: /\.js?$/,
      use: 'babel-loader',
      exclude: /node_modules/
    }]
  },
  plugins: [
    new webpack.NamedModulesPlugin(),
    new webpack.NoEmitOnErrorsPlugin(),
  ],
  output: {
    path: path.resolve('./build'),
    filename: 'server.js'
  }
}
```