The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

4287719

Supervisor: Natasha Alechina

Module Code: G53IDS

2019/04

Increasing customer satisfaction using chatbots

Submitted on April 2019, in partial fulfilment of

the conditions for the award of the degree **BSc Computer Science**

4287719

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: 4287719

Date 12/04/2019

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Abstract

Customer service is an important issue in today’s world. Customer service is something that customers have expectations of, because it greatly impacts their customer experience. As more and more companies are created every day, they are struggling with the problem of direct communication between the company and a customer.

In most of these cases, typical communication with emails should suffice. However, some customers don’t like to use emails, they would rather use something that they use every day, like a chat application. Nowadays, a lot of companies make use of platforms such as Facebook to communicate with their customers.

In this dissertation, a chatbot application will be presented, as well as a customer service application. The resulting product allows customers to submit issues with their orders, and customer service application is used for customer service agents to easily see all information about the user and their order, and to respond to the customer quickly.

Contents

1. Introduction

1.1. Overview

Customer service has always been one of the biggest concerns for a successful company. People need to be able to resolve their issues quickly and without any troubles. Recently, a lot of new companies started to adopt the chat-bot technology for communicating with customers. Advantages of that are: chat-bots can respond 24/7 at any time, chat-bots can reduce customer service workload, and they can be customised to do anything.

The focus of the project is to create an ultimate platform for customer service for a food delivery company that will replace existing outdated means of customer service communication (emails). In order to do that, I will produce a customer service chat-bot that will interact with customers, as well as creating an admin panel for customer service employees where they can respond to queries where human interaction is needed.

To produce this product, first I must identify existing ways of communicating through chat-bots, and then build a bespoke solution for a food delivery company that will be adapted for company needs.

This project is built in collaboration with Halalivery [1], a food delivery company that is based in Nottingham. Halalivery works in a similar way as Deliveroo and UberEats; customer orders food, restaurant/groceries store prepares the order, and drivers deliver from the place to a customer's doorstep. Halalivery consists of three apps: customer app, driver app, and vendor app. So, there could be a point of failure, like the driver not delivering food properly, or restaurant/grocery store forgetting to confirm the order.

1.2. Use cases and Business Problem Examples

Right now, the biggest and easiest platform to build chat-bots is Facebook Messenger. Facebook offers tools to build interactive chat-bots and it offers the easiest way for customers to reach companies. This is useful for a customer since they're able to use chat-bot in their habitual interface (Messenger), because they use it every day. Facebook offers widgets for chat-bots, which is a small piece of interface with buttons, input fields, date selection, etc.

Recently, big companies started to adapt and develop chat-bot services for their own needs. For example, Uber allows customers to order a ride directly through Facebook Messenger, without the user having to open the Uber app. Pizza Hut allows customers to order pizza to their place without the need to access the app or go to a website to order. eBay provides a customer chat-bot where customers can look up products to look for, and where they can get information like order status or ask for help.

1.3. Aims and objectives

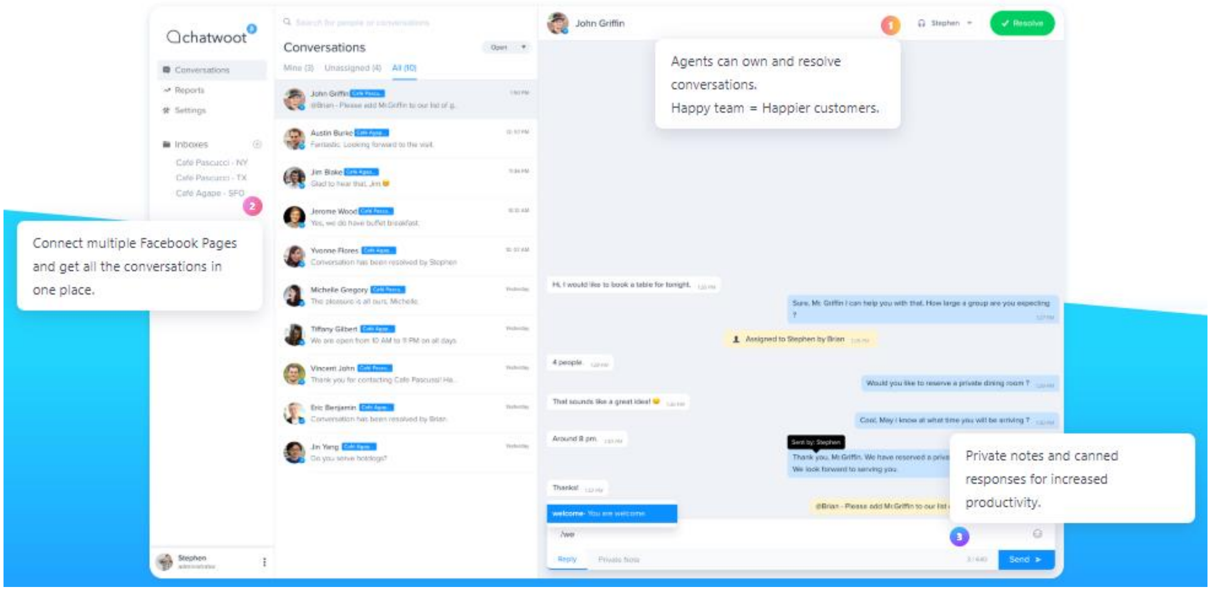
The aim of the project is to create a tool that allows customers to submit an issue with their food order in an easy and accessible way, and for simplifying customer support's job to identify and fix customer's issues.

Chatbot project would consist of: chatbot itself, API, and customer service portal. The **objectives** are as follows:

1. Develop a chatbot that customers can message to, and where customers can see their orders with relevant information, and where customers can submit issues about their order.
2. Implement an account linking process which will be used to link Facebook and Halalivery accounts.
3. Implement a backend server (API) which will be used to send and receive data to/from chatbot and customer service portal.
4. Implement a customer service portal for customer service agents which will show a list of submitted issues, and where agents can see all information about the order and message directly to the user through the chat window.
5. Implement security measures to protect the backend server from unauthorised requests.

1.4. Related work and Research

Analysis. During the research, I have found a few relevant applications. The first of them is Chatwoot [2], which is an app for managing Facebook pages. It allows to provide customer support to users from the Facebook page through the web dashboard. There are some useful features that help customer support to provide a quick response like pre-defined templates and private notes.



Connect multiple Facebook Pages and get all the conversations in one place.

Agents can own and resolve conversations.
Happy team = Happier customers.

Private notes and canned responses for increased productivity.

John Griffin — @Brian - Please add Mr.Griffin to our list of g...
Austin Burke — Fantastic. Looking forward to the visit.
Jim Blake — Glad to hear that, Jim
Jerome Wood — Yes, we do have buffet breakfast.
Yvonne Flores — Conversation has been resolved by Stephen
Michelle Gregory — The pleasure is all ours, Michelle.
Tiffany Gilbert — We are open from 10 AM to 11 PM on all days
Vincent John — Thank you for contacting Cafe Pascucci! He...
Eric Benjamin — Conversation has been resolved by Brian.
Jin Yang — Do you serve hotdogs?

Hi, I would like to book a table for tonight.
Sure, Mr. Griffin I can help you with that. How large a group are you expecting ?
Assigned to Stephen by Brian
4 people.
Would you like to reserve a private dining room ?
That sounds like a great idea!
Cool. May I know at what time you will be arriving ?
Around 8 pm.
Sent by: Stephen
Thank you, Mr.Griffin. We have reserved a priva
We look forward to serving you.
Thanks!
@Brian - Please add Mr.Griffin to our list
welcome- You are welcome
/we

Figure 1 Chatwoot dashboard interface

Another example of the customer service application would be ZenDesk [3]. For customer service, it uses a ticket approach. Through this application, customers can submit tickets through the system, and live agents are able to see tickets and respond to them. For a chatbot component, there is an application called Ada [4], which connects to ZenDesk, and it allows to create chat-bots that can be integrated with ZenDesk ticket system. That way, customers can communicate with the chat-bot, and if chat-bot can’t provide an adequate response to their inquiry, customers can directly submit a customer service ticket to ZenDesk.

The problem with those solutions is that it is not a bespoke software, and therefore can’t be adapted well to specific company needs.

There are plenty of chat-bots that exist as a standalone service. Osome [5] is a bookkeeping service that is made for Singaporean businesses that manage accounting and secretary, all through the chat-bot. For them, chat-bot helps customers to communicate directly with the bookkeeping company. Most important things they learned while making that product is to always show to a user who is talking to them: bot or human, and to reduce the response time so that chat-bot experience will feel like it saves time, compared to other means of communication. Speed is their main aspect; they try to respond to complex queries within the first 15 minutes, and chat-bot responds with simple queries if customers need an answer that can be answered by a bot. Another useful aspect they found out is customer development: by using chat-bot, the company always has a direct mean of communication with the customer. There’s no need to communicate by calling them or sending an email. If there is any feedback that is needed by a company, they can message and ask the customer directly. In preliminary research by Osome, they found out that communicating by chat is much easier than communicating by email in terms of recognising natural language. When the customer writes an email, they usually ask many questions on different topics. It’s harder to parse this kind of message and respond to each question correctly. When the user communicates by chat, the customer usually writes short sentences and asks questions one-by-one. It is much easier to parse and reply to messages from chat than parsing information from emails.

2. System Requirements

2.1. Overview

In this section, requirements of the whole system, including chatbot, backend API server, and customer service portal, will be discussed. The requirements are split into two categories: functional and non-functional. Functional requirements are the statements the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations [6]. Non-functional requirements are constrains on the functions offered by the system, which include constraints on the development process and constraints imposed by certain standards. Non-functional requirements often apply to a whole system.

2.2. Functional requirements

Chatbot. Chatbot must have:

1. A way for the customer to see their past orders
2. A way for the customer to submit an issue about specified order
3. A way for the customer to see order's receipt: date and time, item list, item price, address, payment summary
4. A way for the customer to get available places to order by specified postcode
5. A way for the customer to get a menu for a specified marketplace

Backend API server. API server must have:

1. An authentication view for linking Facebook and Halalivery accounts
2. An endpoint for getting user's orders for chatbot
3. An endpoint for getting marketplaces by postcode
4. An endpoint for reporting an issue that comes from a chatbot
5. An endpoint for getting issues, which will be served to a customer service portal
6. An endpoint for sending a message to the user by specified order ID
7. An endpoint to serve a customer service portal (SPA)

Customer Service portal. Portal will be used for communicating between customer service agents and customers. A portal:

1. Must show a list of issues that were submitted by customers, with relevant information such as customer's name and vendor name
2. Must show a detailed view of a selected issue, with issue information such as a list of items that were ordered, and vendor's name, address and a phone number so that customer service agent can quickly solve a problem
3. Must have a chat window on a detailed issue view, where a customer service agent can communicate with the user by typing messages

2.3. Non-functional requirements

Compatibility.

The backend system must be able to run on any machine that is capable of running Python. The system must be installed on a public host so that other services are able to communicate with the backend. The machine must also be able to run the PostgreSQL database, so that database records can be saved and used. MySQL is also supported, although some changes have to be made in order to support it. [7]

Frontend system works on client machines, and support for modern browsers must be ensured. This includes any browser that is capable of running ECMAScript 5 [8] (Internet Explorer 8 and below are not supported).

Chatbot must be able to run on any machine that is capable of running Node.JS. Chatbot must also have a public callback URL, that is publicly available, and SSL-secured. This is because of Facebook requirements, which only allows sending webhook data to HTTPS URLs. [9]

Performance. Chatbot must be able to handle multiple users at once and perform under heavy load. API server must be able to handle multiple requests from the chatbot and customer service portal at once.

Security and validation. API server stores and serves a lot of user data, which can be sensitive. Therefore, some measures must be taken in order to secure it and to comply with the Data Protection Act [10], such as only allowing authorised users and services to get user data.

Communication between chatbot and backend API server must be secured by using an access token, which is included with each request. Access token then must be checked and confirmed as valid, and any requests without token or with invalid token must be rejected. Communication must also be HTTPS/SSL secured to prevent any malicious sources from tampering the data.

API server also has to validate any input that is submitted to it and reject any request with invalid data submitted.

3. Methodology

3.1. Microservices

Microservice Architecture has been increasingly more popular in the last few years. It is a way of developing applications through the association of independently deployed IT services. This way of methodology allows building around business processes and the requirements of the services. With the help of standard protocols of communication, such as HTTP requests and responses through API and message brokers, microservices can be written on different programming languages and can use different database technologies.

With the traditional way to the creation of an IT system on the “monolith” principle, management focuses more on technology. Developer teams are formed on technological principle – developers, database administrators, data analytics, etc. This approach causes serious difficulties in implementing business functionality in an IT system since they need to be produced through cross-team interaction.

Microservices allow forming independent (agile) cross-functional teams with a focus on a specific business task instead of technology. Such teams are self-sufficient, well-focused, use the approaches and technologies that are most effective in a particular case and allow them to switch from “project management” to “product”.

In microservices, each element performs only one function and can be "delivered to production" independently of other services, therefore, in order to make changes to the solution, it is not necessary to rebuild the entire algorithm. If there is a need for updating, debugging, restructuring, or making other adjustments, then it is enough to carry them out on a specific part of the IT system. For example, if there is a need to add a new method of payment or delivery on the website of an online store, it is enough to make changes in just one module.

In reality, adding new feature one way or another will affect the rest, but the principle of microservice architecture allows to release production-ready releases at different times and it gives each team the opportunity to work in their own rhythm. That is, two functionalities can be developed in parallel, for example, adding search and delivery options, while code changes in one part of the system can happen very frequently (every day), and changes in the other part of the system take more time to develop – for example, if the release must be synchronised with the launch of business processes.

3.2. Advantages of microservice architecture:

- **Simplicity**. Instead of a single and highly complex IT system, developers work with a more manageable architecture, where each component is responsible for its function.
- **Increased stability**. Since microservices are independent of each other, failures and defects in one of the microservice do not affect the work of the others. The system functions with minimal disruptions and downtime.
- **Quality scalability**. When a corresponding need arises, it is not necessary to scale the entire system, dismantling it to the ground, it is enough to make the necessary changes only in a certain area.
- **Multiplatform**. Microservices can work at any device, and also in cloud solutions.
- **Re-usability**. Microservices can be redeveloped for other tasks and purposes after the initial launch.
- **Ability to use different technologies**. Agile teams are not restricted to one platform or technology. With the microservices, teams can combine different technologies that work best in each area.

3.3. Difficulties of integrating microservice architecture

In a comparison of the "monoliths", microservices open the next level of flexibility of operations through the creation of new business processes around the existing legacy systems. This is why today many people are focused on microservice transformations – its advantages are too obvious.

However, there are a number of pitfalls in the implementation of such a transition from monolithic architecture to microservice architecture.

Large number of microservices. When the company needs around 5-15 microservices, it's adaptation and developing is not that hard. But when the company requires something like 50-100+ microservices, the issue of scalability arises. How to avoid duplicating functionality and keep control of the process, and also to prevent the transformation of the new microservice architecture into the same monolith, from which they initially tried to leave?

Balance of flexibility and chaos. Flexibility is the necessary condition for a microservice approach, but if it's not controlled, the risk of chaos in the system architecture increases dramatically. Understanding the principle of such "chaos" can often be a separate project. It is better to make restrictions from the beginning, even if it will postpone the release of the first functionality so that less time would be wasted in "dismantling fossils" of microservices.

Time management. Changing the architecture of the IT system can't occur instantly. It takes a lot of time, sometimes months/years. Not a lot of things are as complex as modern IT systems that are adapted to the needs of the business as well as specific ways of binding them together to each other. Nevertheless, despite the overall complexity of dismantling legacy systems, to rely on the "old school" IT approach in an era where the "New IT" paradigm is developing by leaps and bounds – means to condemn business to slow agony. Hence the need for microservice architecture, because it allows developers to write consistent code with the advantages of microservices.

Modern microservice architecture is adopted by big companies like Amazon, Google, and Netflix. For example, Amazon makes changes into its products every 11,6 seconds [11], and Netflix constantly conducts selective shutdowns of computing nodes to verify the impact of such accidents on the overall quality of the product received by Netflix subscribers. [12]

3.4. Applying microservice architecture

In the project, I will be applying microservice principles to my software product. There will be two main components – chatbot service, which is built on Node.JS, and web service, which is built on Python/Django web framework.

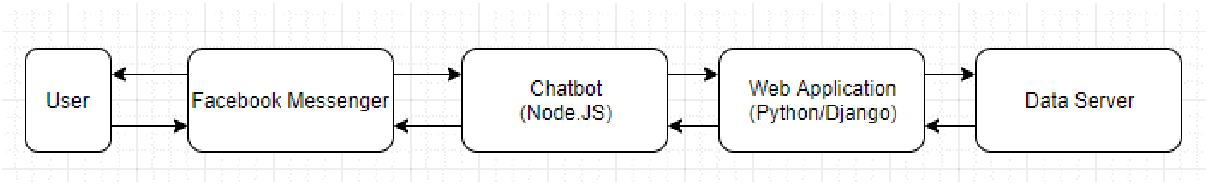


Figure 2 Microservice architecture (chatbot and web app)

Simply put, chatbot service will be responsible for sending messages to users by using Facebook API. This is done by sending requests to the Facebook endpoint with the required data (like the ID of recipient and message text). It will be also responsible for creating conversation components and filling them with relevant data – like rich-media, message templates, quick replies, buttons, menus and so on.

Chatbot service will be responsible for receiving messages through Facebook and sending data (messages, issues) to backend service.

The web application will be responsible for login procedure – linking the Facebook account with Halalivery account. It will provide the interface for users to type in their Halalivery account details, verifying if they are correct, and storing the Facebook User ID and Halalivery User ID together in a database record. If chatbot requires data such as order information, or user details from Halalivery, it asks the web application for that information.

Then, a web server can identify if the Facebook user has an active account with Halalivery, and if so, it asks the main Halalivery API service to provide that data.

4. Design

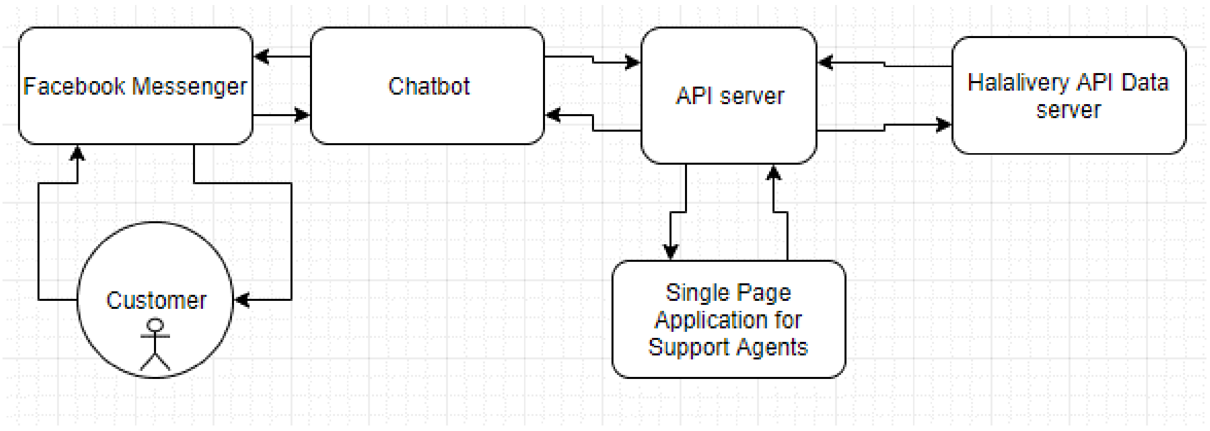
4.1. System Design

The system consists of three main components: Chat-bot, API server and customer service portal - SPA (single-page application). Chatbot and API server runs on a server machine, while SPA is executed on a customer support machine (in a web browser).

In Single Page Application, the application is fully loaded at once and the business logic is transferred from the client to the server. This results in client-heavy applications which communicate with the server through the API. There are three tiers that can be identified in Single Page Applications: presentation tier (which is a user interface built with HTML, CSS, and Javascript), business logic tier (getting data from API) and data tier (where data is stored). [13]

Chat-bot and SPA both rely on an API server to fetch data. Chat-bot uses API server to get information about the user, and SPA uses the API server to get information about all recorded issues from users.

This system structure is shown on a diagram below.


Facebook Messenger
Chatbot
API server
Halalivery API Data server
Customer
Single Page Application for Support Agents

4.2. Chatbot system

Simply put, chatbot service will be responsible for sending messages to users by using Facebook API. This is done by sending requests to the Facebook endpoint with the required data (like the ID of recipient and message text). It will be also responsible for creating conversation components and filling them with relevant data – like rich-media, message templates, quick replies, buttons, menus and so on.

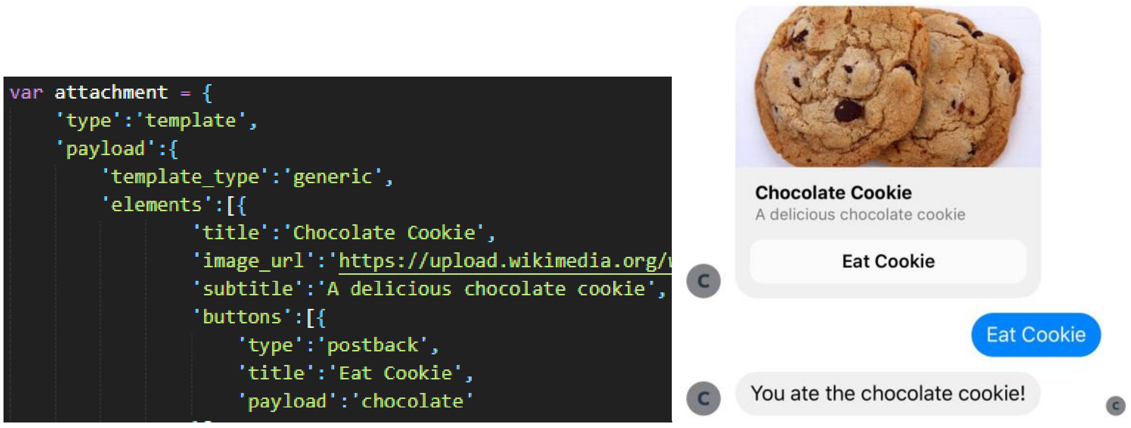
Chatbot service will also be responsible for receiving messages by using webhooks. Webhooks are “user-defined HTTP callbacks”. When a data source (Facebook) wants to emit an update to a service (such as a new message, postback event, etc.), it sends an HTTP request to a specified route at the chatbot’s domain. [14]

Webhook events can be specified in Facebook Messenger Developer settings. For now, ‘messages’, ‘messaging_postbacks’ and ‘messaging_account_linking’ webhooks are enabled. It means that if any of those events happen: new message, message postback (user pressed a button), account linking information – only those will be sent to chatbot callback URL.

A programming library called **Botkit** [15] was used to develop chatbot service. It is a library that provides developers a platform-independent, language-like interface for building a chatbot. It is designed to easily build conversational user interfaces.

The library is written in Node.JS, so it must be used to build a chatbot. The library contains lots of messaging platform integrations, such as Facebook Messenger, Slack, Web, Twilio, Google Hangouts and more. For this project, only Facebook Messenger is considered as it is one of the most popular platforms for chatbots, and it has lots of registered users already.

Botkit includes deep integration with Facebook Messenger platform that allows to build user interfaces inside a chatbot. This is done by sending a customized attachment with specified buttons and “postback” payload. When the user clicks the button, chatbot receives a message that the user clicked a “chocolate” payload.



```
var attachment = {
    'type':'template',
    'payload':{
        'template_type':'generic',
        'elements':[{
                'title':'Chocolate Cookie',
                'image_url':'https://upload.wikimedia.org/
                'subtitle':'A delicious chocolate cookie',
                'buttons':[{
                    'type':'postback',
                    'title':'Eat Cookie',
                    'payload':'chocolate'
```

Chocolate Cookie
A delicious chocolate cookie
Eat Cookie
Eat Cookie
You ate the chocolate cookie!

Figure 3 An example of a template with a button

Before a user is required to access the chatbot, the user is requested to perform an authorisation process so that Messenger account and Halalivery account are linked together. This is done by built-in Messenger authorisation flow, where the user clicks on a login button in Messenger and enters a Halalivery username and password. After a successful authorisation, a Halalivery token is stored in an API server database along with Facebook user ID. When an authorised user requests order information, chatbot sends a request to an API server with the user’s ID and checks if the user’s ID is in a database.

4.3. Backend API system

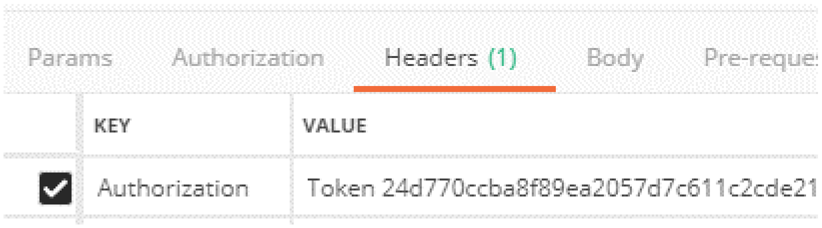
API server is responsible for serving API endpoints, receiving and sending data to the chatbot and customer service portal, along with storing information about the linked user and their issues in a database. Since a lot of information passes through the API server, it is important to design it in a robust way so that other services that rely on API server will receive the expected data they want.

Database model. The database model determines the logical structure of a database. For the API server, we need to store two things: information about linked users and information about recorded issues. This data comes from a chatbot service that sends that data to the endpoint.

First, we need to determine which data is needed to store for API server. There are two things which are required based on requirements: data about linked users (with information such as Facebook ID and Halalivery token for authenticated data fetching) and a list of issues which are recorded by customers in a chatbot flow. Based on that information, two data models are created: LinkedUser and Issues.

Security. API server has a built-in Django authorisation and user management mechanism that allows to protect endpoints from unauthorised requests. This is done by including an access token with the requests from a chatbot and SPA system. Since both systems require the same data, there is no need to build a permission-based restriction. For user management, it is only needed to set up “api” user account (which is used for chatbot), and customer service agent accounts for the web portal.

Access token is a common way to protect API endpoints. With each request from a chatbot, an access token is included in request’s headers.



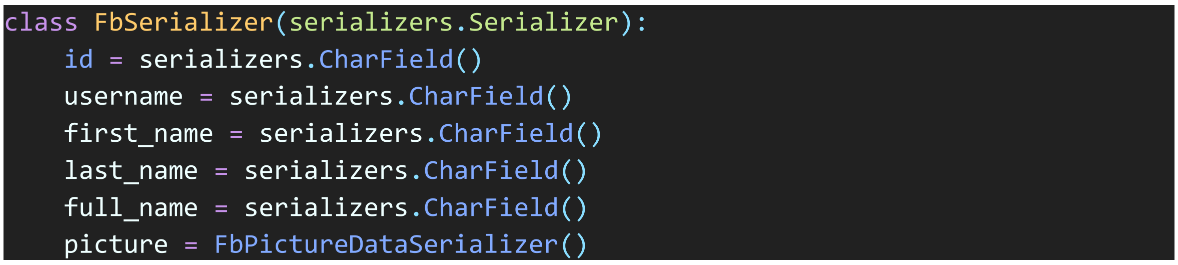
	KEY	VALUE
✓	Authorization	Token 24d770ccba8f89ea2057d7c611c2cde21

Figure 4 Authenticated request

Communication with data server is also done by using access tokens. When the user complete authorisation flow, a data server returns an access token, which is stored in the user’s record in a database. This mechanism allows using token to get specific user’s information without storing user’s login and password.

Error handling and validation. API server is designed to be robust, in case that endpoints receive incorrect data. A common scenario where validation is needed is the account linking process. When the user enters login and password, backend system checks that information with the Halalivery data server, and in such case where the server is unavailable or user enters incorrect login information, a correct error message must be shown to the user.

Another example of error handling includes receiving data from a chatbot. Though it is unlikely, a chatbot can receive incorrect data from Facebook API, and relay it to a backend server. To make sure that the data from Facebook is correct, serializers are used in a backend server. Serializers allows complex data like Facebook user information to be validated. This is done by creating a data schema that must be satisfied for the serializer to be valid.



```
class FbSerializer(serializers.Serializer):
    id = serializers.CharField()
    username = serializers.CharField()
    first_name = serializers.CharField()
    last_name = serializers.CharField()
    full_name = serializers.CharField()
    picture = FbPictureDataSerializer()
```

Figure 5 An example of the serializer for Facebook data

App structure. Backend server follows a strict app structure pattern. Each module is stored in its respective file:

- Endpoint views are stored in app/*views.py*
- Models are stored in app/*models.py*
- Data serializers are stored in app/*serializers.py*
- Forms are stored in *app/forms.py*
- Custom middleware (disabling CSRF check) is stored in *app/middleware.py*
- Admin panel model registering is stored in *app/admin.py*
- App settings are stored in *settings.py*

The structure follows the project skeleton standard from the official Django tutorial. [16]

Structuring the application by modules makes it easier to find and edit each module that needs to be modified, and it results in time saved during the development process. Various part of the framework shouldn't "know" about each other unless absolutely necessary [17].

4.4. Customer service portal

As mentioned before, customer service portal is responsible for showing recorded issues, which come from customer's submitted data in a chatbot. The portal provides customer service agent with a list of issues, along with a chat interface and relevant information such as vendor's name, address, phone number and a list of items which have been ordered. The agent can send a message to a user directly through an interface, and it will relay it back to the user on Facebook Messenger.

The aim of the portal is to simplify the communication and make it faster to respond to user's issues.

4.4.1. User Interface Design

Customer service portal’s design basis is taken from Material Design [18], which is developed by Google in 2014. Material Design is defined as “a visual language for users that synthesizes the classic principles of good design with the innovation of technology and science”. The customer service portal uses Material Design elements such as grids and lists.

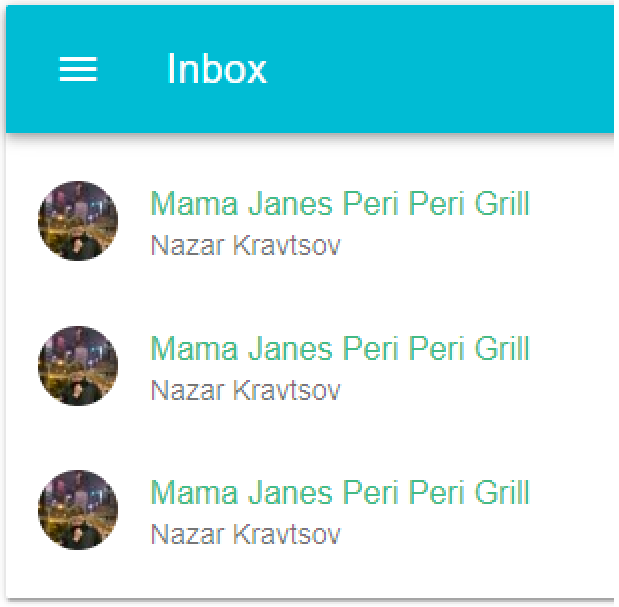

Inbox
Mama Janes Peri Peri Grill
Nazar Kravtsov
Mama Janes Peri Peri Grill
Nazar Kravtsov
Mama Janes Peri Peri Grill
Nazar Kravtsov

Figure 6 Material Design - list of issues

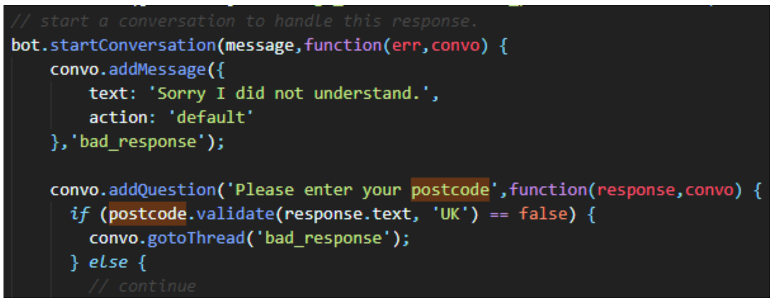
For applying Material Design, Vuetify [19] is used, which is a semantic development framework for Vue.js that provides all the tools necessary to create application interfaces which solely rely on Material Design pattern.

5. Implementation

5.1. Chatbot system

Overview. As mentioned before, chatbot is built on a Botkit framework, which includes many features to make it easier to develop chatbots. Other options were also considered, like Botpress [20], but as at the time of development, there was no Facebook integration available, which would have made development much harder.

Conversation objects. For complex interactions between a chatbot and a user, conversation objects [21] are used. It is used to string together multiple messages, including questions for the user, into one cohesive unit. Conversations objects allow creating branching, data validation, repeating or looping through sections of dialogue. This is done by using *threads* feature that are pre-built chains of dialogue between the chatbot and a user. For example, threads can be used to make common scenarios like *yes/no/quit* prompts from the user:



```
// start a conversation to handle this response.
bot.startConversation(message,function(err,convo) {
    convo.addMessage({
        text: 'Sorry I did not understand.',
        action: 'default'
    },'bad_response');

    convo.addQuestion('Please enter your postcode',function(response,convo) {
      if (postcode.validate(response.text, 'UK') == false) {
        convo.gotoThread('bad_response');
      } else {
        // continue
```

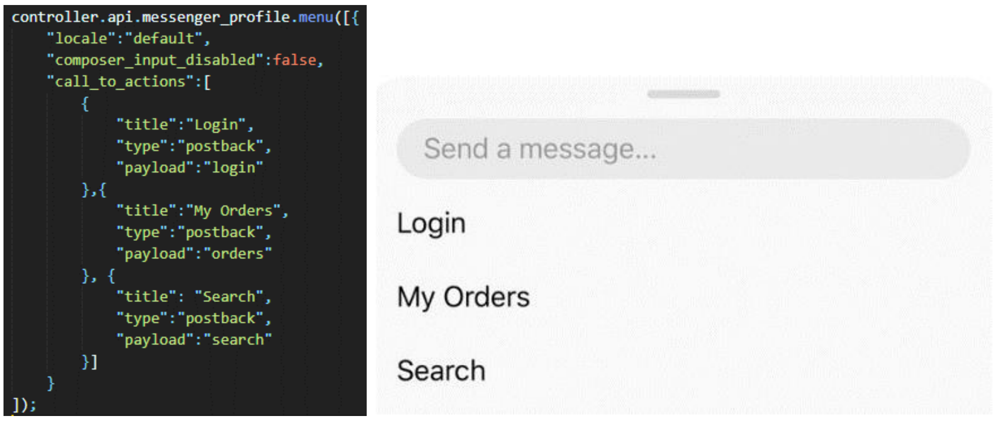
Figure 7 Example of "threads". First, bad response thread is defined which sends a bad response message to the user. gotoThread function is used to go to "bad_response" thread, and then after the message is sent, a thread is set back to "default"

Hears objects. In order to respond to the messages received by the user, chatbot needs to define "trigger" words. For example, if a user clicks on a button or types in a command, chatbot needs to differentiate between different entry points.

```
controller.hears(['Orders'], 'message_received,facebook_postback', …
```

In the figure above, we have a defined "controller" object and *.hears* method that is used to hear commands. Then, we have a 'message_received' parameter that is used for hearing a raw message – i.e. if a user sends "Orders" message, and also we have a "facebook_postback" that is used for Facebook Messenger button action – i.e. if the user clicks on "Get orders" with "Orders" postback button.

Persistent Menu. Facebook Messenger allows creating a 'persistent' menu – which can be used anywhere by either swiping up (on a phone) or clicking on a menu button (web). The menu can be set with different elements that do different actions.



```
controller.api.messenger_profile.menu([{
    "locale":"default",
    "composer_input_disabled":false,
    "call_to_actions":[
        {
            "title":"Login",
            "type":"postback",
            "payload":"login"
        },{
            "title":"My Orders",
            "type":"postback",
            "payload":"orders"
        }, {
            "title": "Search",
            "type":"postback",
            "payload":"search"
        }]
    }
]);
```

Figure 8 Example of the persistent menu with Login, My orders and Search actions

Chatbot web server. In order for a chatbot to receive data from the backend server, integrated web server (Express) is used. For now, only one action is needed, which is sending a message from a customer service portal back to the user.

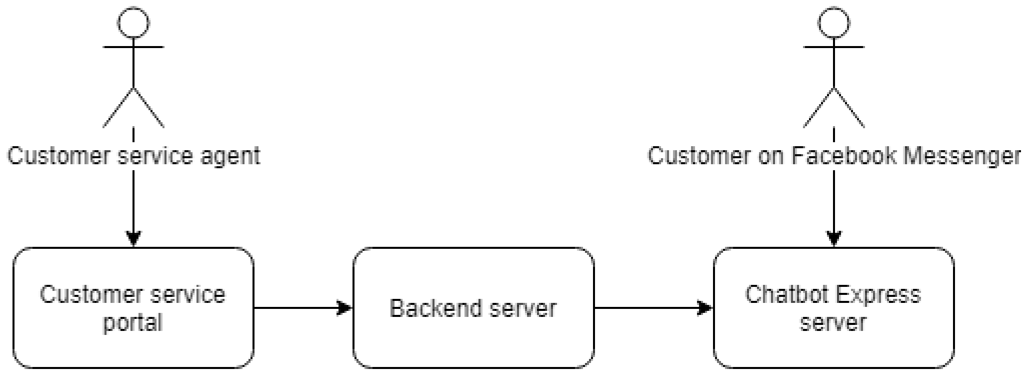


Figure 9 Sending message from customer service portal back to the user on Facebook Messenger

5.2. Backend API system

Overview. The backend API system is developed using Django, which is a web framework on Python [22]. Django was chosen because for a number of reasons: Django has a lot of built-in stuff that makes developing back-end much easier; it has a built-in administration panel for easier editing and looking up data; and it is based on Model-View-Controller (MVC) design pattern, which means that backend code, database and front-end code (templates) are separated. It also has a lot of useful packages that help with development. All these factors are important for maintaining the application.

Another dependency for backend application includes Django Rest Framework [23], a toolkit to build web APIs. It has some features like serializers and API authentication, which are used for parsing Facebook data and securing the endpoints.

Views. A view function is a Python function that takes a web request and returns a web response [24]. The view itself contains a logic to process the request and returning an adequate response.

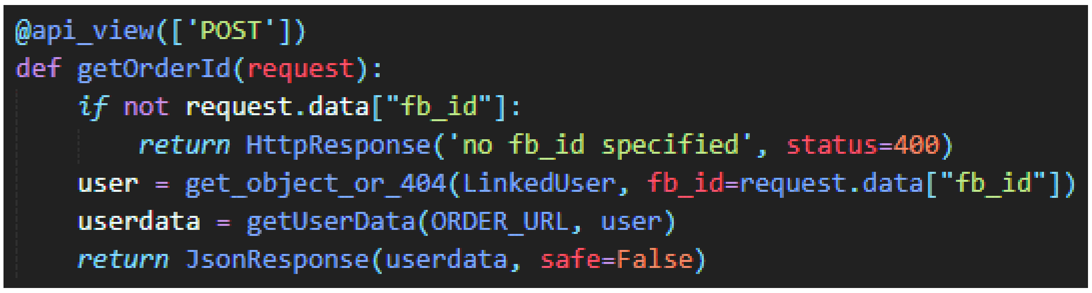
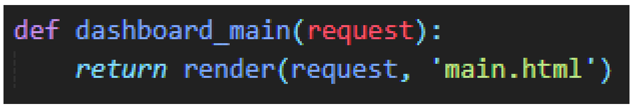

```
@api_view(['POST'])
def getOrderId(request):
    if not request.data["fb_id"]:
        return HttpResponse('no fb_id specified', status=400)
    user = get_object_or_404(LinkedUser, fb_id=request.data["fb_id"])
    userdata = getUserData(ORDER_URL, user)
    return JsonResponse(userdata, safe=False)
```

Figure 10 An example of the view function that takes Facebook ID as a parameter and returns order information for the specified user.

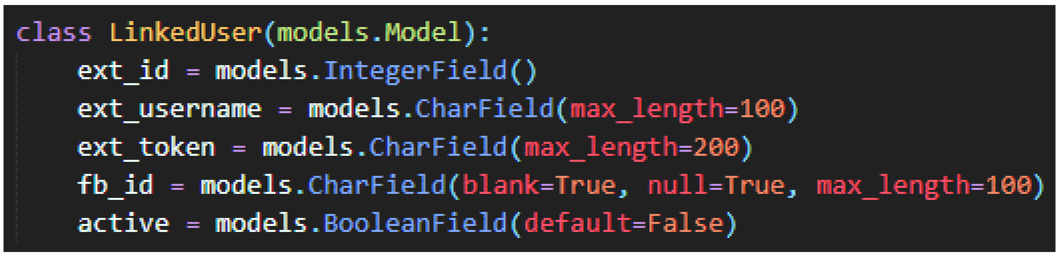
First, *@api_view([‘POST’])* specifies that this view function only allows POST request to it. Any other types of requests such as GET, PUT, PATCH, etc – will be rejected. For this example, there are two validation checks for the request: if there is no “fb_id” in the request, an error is returned with a status code 400 (Bad Request). After that, there is a “if that record exists” check – if the record with specified “fb_id” is found – it saves the object to “user” variable, otherwise, it returns an “NotFound” exception with status code 404 (Not Found).

Views are used for serving customer service portal; is it a simple *render* function that takes HTML file as a parameter and returns it to the user.



```
def dashboard_main(request):
    return render(request, 'main.html')
```

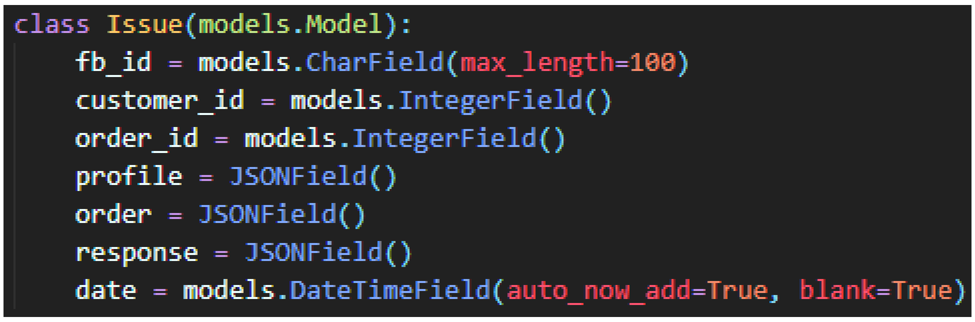
Database model. A model is a source of information about the data, that contains the database fields [25]. Models define the structure of data in the database. In the case of backend application, it is used for two models: “LinkedUser” and “Issue”. Each model has a number of fields with a specified type of data that can be inserted into a field.



```
class LinkedUser(models.Model):
    ext_id = models.IntegerField()
    ext_username = models.CharField(max_length=100)
    ext_token = models.CharField(max_length=200)
    fb_id = models.CharField(blank=True, null=True, max_length=100)
    active = models.BooleanField(default=False)
```

Figure 11 LinkedUser model

Fields “ext_id”, “ext_username” and “ext_token” are used to store Halalivery account ID, username and unique token. The token is used to fetch order data and other data which requires authorization.



```
class Issue(models.Model):
    fb_id = models.CharField(max_length=100)
    customer_id = models.IntegerField()
    order_id = models.IntegerField()
    profile = JSONField()
    order = JSONField()
    response = JSONField()
    date = models.DateTimeField(auto_now_add=True, blank=True)
```

Figure 12 Issue model

The figure above is the Issue model. It contains “fb_id”, which is a Facebook ID of the user, “customer_id”, which is the customer ID of Halalivery account, “order_id” is the order ID which the issue relates to, and three JSON fields: profile, order and response, which are used to store extended data. “profile” field contains an extended profile of Facebook user, such as first/last name and profile picture. “order” field contains a full order information, like total amount paid, ordered items and information about the vendor. “response” field contains data about the user’s message, and it also contains NLP data. “date” field is auto-generated when the record is first inserted into database.

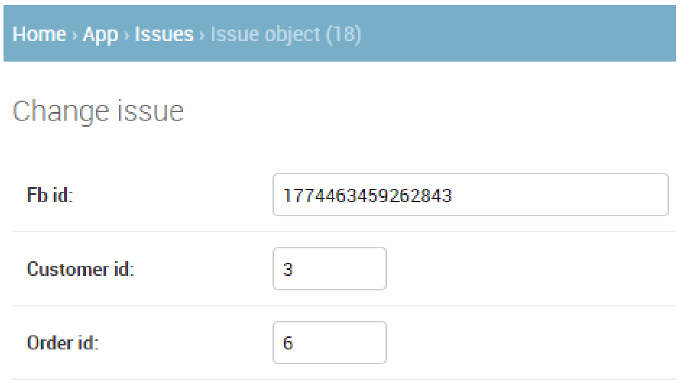
Logging. For debugging purposes, logging module was used. Logger is the entry point into the logging system [26], and there are various logging modes that can be used, like: DEBUG, INFO, WARNING, ERROR and CRITICAL. For example, if an error has occurred, this code is used to log an error:

```
import logging
logger = logging.getLogger(__name__)

...
logger.error('Something went wrong!')
...
```

Logging was used extensively throughout the backend server code, to ensure that it received correct data from chatbot and customer service portal.

Admin panel. Django has a built-in administration panel that can be used to add/modify new users, as well as modifying custom model data. It reads metadata from existing models and shows all records that are stored for that model.


Home › App › Issues › Issue object (18)

Change issue

Fb id:	1774463459262843
Customer id:	3
Order id:	6

Figure 13 Changing issue model data

Linking process. Facebook Messenger Developer platform has an in-depth guide on how to build an account linking service [27]. First, a callback URL is registered by using the Log In button. This URL must be public, and SSL secured. Then, when the user clicks on a log in button, the user gets redirected to a callback URL, along with *redirect_uri* and *account_linking_token*. The only parameter that is needed is *redirect_uri*, which is used to redirect the user after a successful login. Account linking token is used for getting PSID (user id in scope of the page), but it is not necessary, as the backend receives ID from a chatbot service.

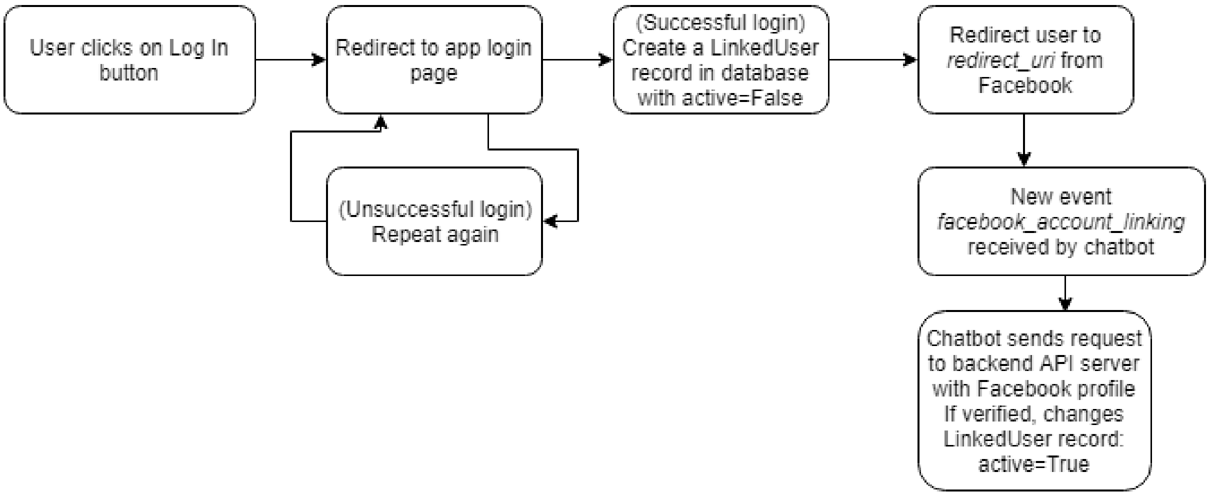
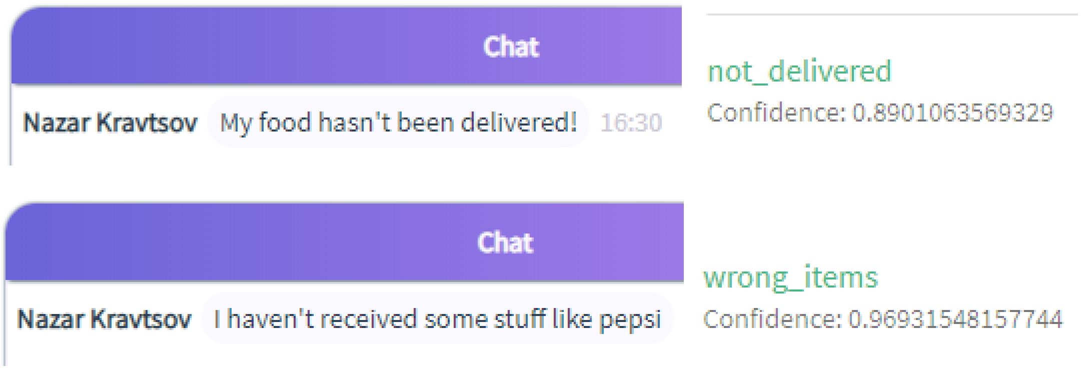

User clicks on Log In button
Redirect to app login page
(Successful login) Create a LinkedUser record in database with active=False
Redirect user to *redirect_uri* from Facebook
(Unsuccessful login) Repeat again
New event *facebook_account_linking* received by chatbot
Chatbot sends request to backend API server with Facebook profile If verified, changes LinkedUser record: active=True

Figure 14 Linking flow

5.2.1. Natural Language Processing

NLP is used when the new issue is added by the user. It works by using Wit.AI Natural Language Processing service [28] which is integrated into Facebook Messenger. When the user submits an issue, NLP engine scans submitted message for possible matches from pre-defined intents, such as food not delivered, wrong items, etc. If the match is found, NLP engine includes matched intent and confidence score with the message data.

This feature allows to reduce each message into categories, which can be used to further analyse data. The list of submitted issues can be used to analyse on how many of those issues belong to each 'fault' category and filter them – for example, filter issues that are related to "not delivered" category or any other category.



Nazar Kravtsov My food hasn't been delivered! 16:30

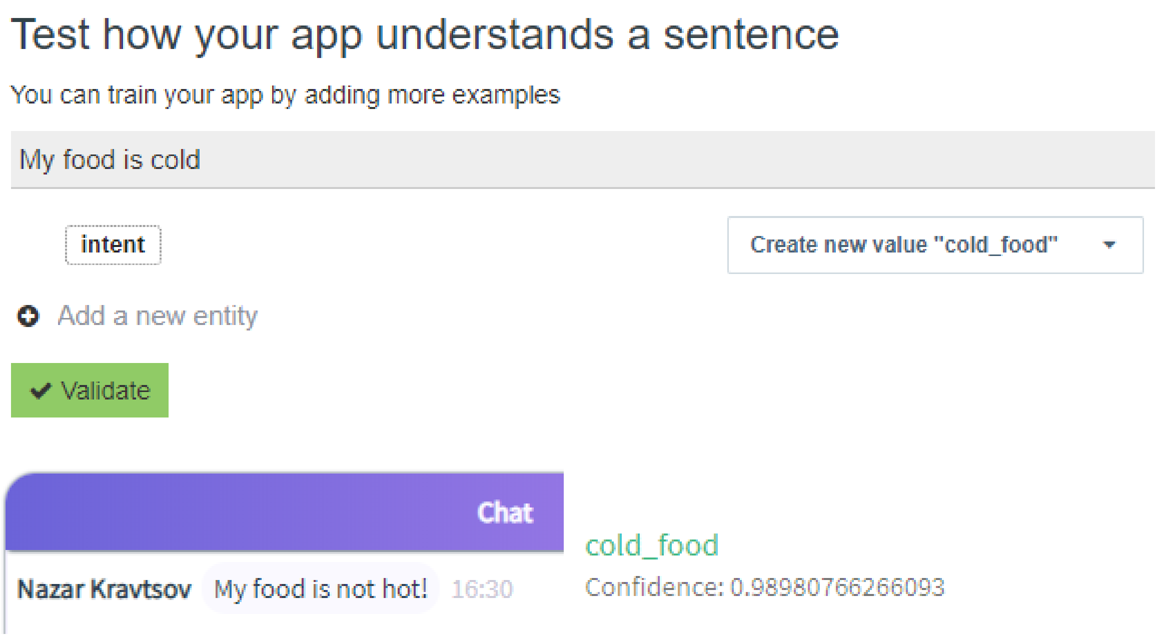
not_delivered
Confidence: 0.8901063569329

Nazar Kravtsov I haven't received some stuff like pepsi

wrong_items
Confidence: 0.96931548157744

Figure 15 User's message and NLP result

In order for NLP engine to work correctly, some training is required. In Wit.AI dashboard, it's very easy to do so, all that is needed is a single sample message. Then, this message is assigned to new intent. After validation and training process, new intent can be seen in NLP data.


Test how your app understands a sentence

You can train your app by adding more examples

My food is cold

Nazar Kravtsov My food is not hot! 16:30

cold_food
Confidence: 0.98980766266093

Figure 16 New intent "cold_food" has been added and tested

Database queries can be used to filter out specific issue categories. For example, if we need to get all issues that are related to “cold_food” category, this query can be used:

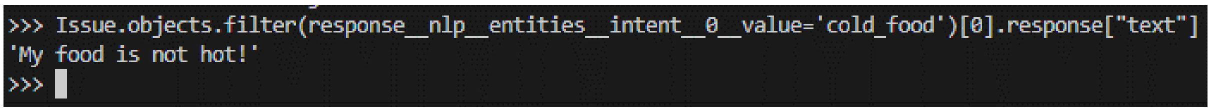

```
>>> Issue.objects.filter(response__nlp__entities__intent__0__value='cold_food')[0].response["text"]
'My food is not hot!'
>>> 
```

Figure 17 Filtering NLP categories

5.3. Customer service portal

5.3.1. Overview

A web portal is primarily built on Vue.js [29], which is a popular progressive JavaScript framework for building single-page applications. It is tiny in size (~24KB) and allows to easily create user interfaces. Vue.js is also very flexible, and because of MVVM architecture [30] (Model-View-ViewModel) allows to facilitate two-way communication between View and Model, which allows to create data bindings. That means that if data in a model changes, views are updated automatically based on new data.

5.3.2. Routing

In Vue.js, routing is done by a package called *vue-router*. It supports nested routes to nested components and offers simple API for navigation hooks and advanced control of routes.

Routing is used for navigating through the list of issues. Whether a user clicks on a specified issue, the route gets changed in an address bar that shows that it is in the “Chat” view and also it shows ID of the issue. That is useful if someone from customer service wants to send the link to another agent and show the specific issue.

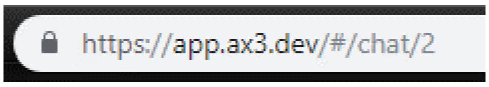
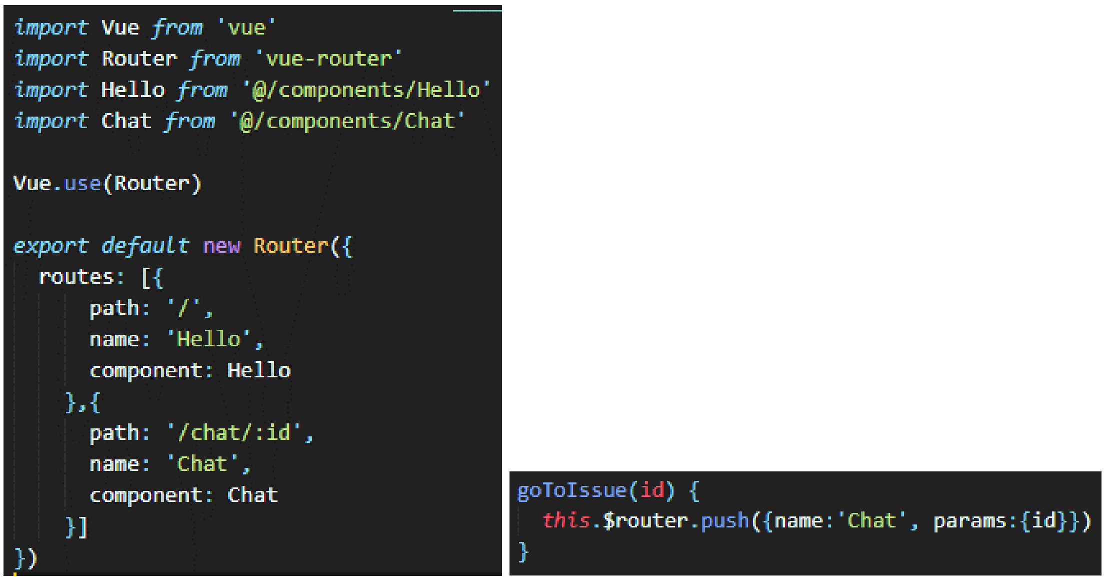

```
https://app.ax3.dev/#/chat/2
```

Figure 18 An example of Chat route with ID 2


```
import Vue from 'vue'
import Router from 'vue-router'
import Hello from '@/components/Hello'
import Chat from '@/components/Chat'

Vue.use(Router)

export default new Router({
  routes: [{
      path: '/',
      name: 'Hello',
      component: Hello
    },{
      path: '/chat/:id',
      name: 'Chat',
      component: Chat
    }]
})
```

```
goToIssue(id) {
  this.$router.push({name:'Chat', params:{id}})
}
```

Figure 19 router/index.js file and routing push example

5.3.3. Fetching data from API

For populating a list of issues, fetching a data from API is required. This is done by sending a GET request to API server, receiving JSON data and populating HTML file with received data. Axios was used for sending requests [31], which is a promise-based HTTP client that can be used in a web browser.

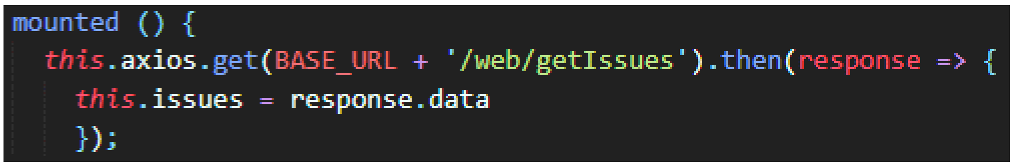

```
mounted () {
  this.axios.get(BASE_URL + '/web/getIssues').then(response => {
    this.issues = response.data
    });
```

Figure 20 Getting data from API in a mounted lifecycle

This process is done in the *mounted* lifecycle when the component first gets mounted (i.e. loaded). Then, after getting data, it iterates through the array of issues and displays a single tile for every issue in an array.

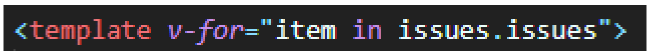

```
<template v-for="item in issues.issues">
```

Figure 21 Example of iteration through the array of issues

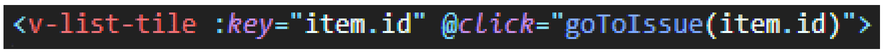

```
<v-list-tile :key="item.id" @click="goToIssue(item.id)">
```

Figure 22 Example of usage of iterated data

5.3.4. Deploying

For deploying a Vue application, building (compiling) is necessary to convert Vue code into compiled Javascript code. This is done by doing *npm run build* command, that runs the Webpack building process. Webpack [32] is a JavaScript module bundler, that merges code from multiple files into one. While Webpack bundles the code itself, it also transforms code with plugins and necessary loaders. For the app, Webpack transforms Vue code into the HTML, CSS and JS that browsers can consume.

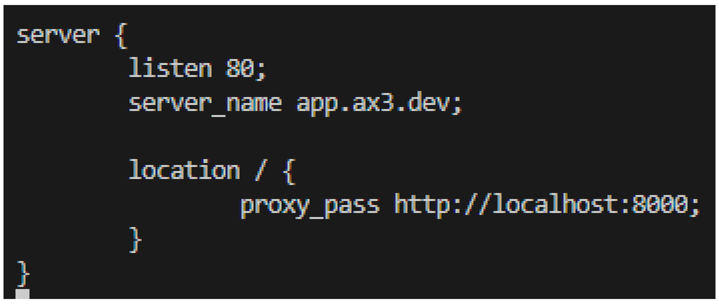
To make it easier to deploy an application, I've created *deploy.sh* script that automatically builds Vue application and runs backend server.

```
echo 'Run npm build'
npm run build
echo 'Done...'
export PORT=8000
echo 'Server runnning on port ' $PORT
python3 manage.py runserver
```

Figure 23 deploy.sh script

5.3.5. Server configuration

In order for the server to be publicly available with proper SSL configuration, I used Nginx [33], which is a high-performance web server that can also be used as a reverse proxy. Reverse proxy works as follows: backend application runs on port 8000, Nginx works on port 80, and Nginx is configured to relay all traffic from domain to backend application. Benefits of doing this way and using Nginx as a web server are increased security (ensures that the identity of the backend system remains unknown) and better performance (better static file serving).



```
server {
        listen 80;
        server_name app.ax3.dev;

        location / {
                proxy_pass http://localhost:8000;
        }
}
```

Figure 24 Reverse proxy configuration

6. Evaluation

6.1. Overview - Chatbot

Tests are an important part of the project of any scale, and it is critical to test every function thoroughly to ensure that it works correctly. Because of the nature of the project (chatbot), to test each function within the scope of the chatbot flow, we need to send messages and interact with the interface normally.

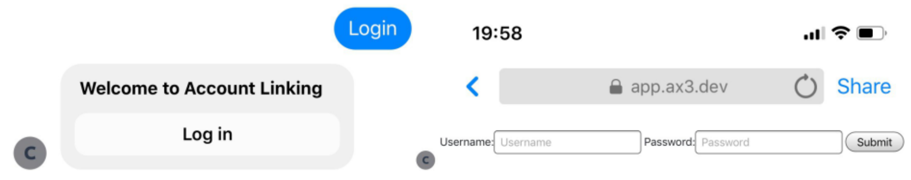
In the testing, a **black-box** approach is used for testing chatbots [34], using observation and evaluation of the response.

These are the available flows in the chatbot:

- Login flow
- My orders flow
- Search by postcode flow

6.1.1. Login flow

The login process is where customers inputs login and password from Halalivery account. When the user clicks on "Login" through the menu or sends a "Login" message, chatbot replies back with a log in button. User clicks on a log in button, and a web view shows username and password fields where the user should put their account details into. The log in process should be done within 5 minutes of pressing the button, otherwise Facebook throws an "Account Linking Failed" error.

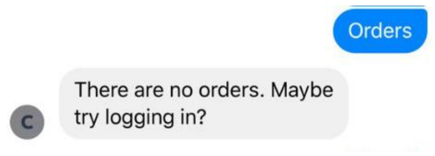

Login

Welcome to Account Linking

Figure 25 Linking process

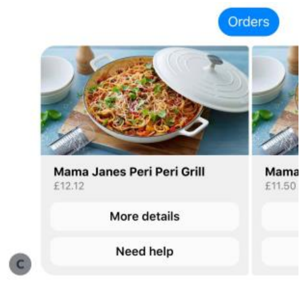
6.1.2. My orders flow

This is where users can see their past orders, including information about the order (receipt) and it's where users can submit a problem with their order. When a user clicks on a "My orders" button, chatbot requests data from backend server about available past orders. If there are no orders or the user is not logged in, the following message is shown:


Orders

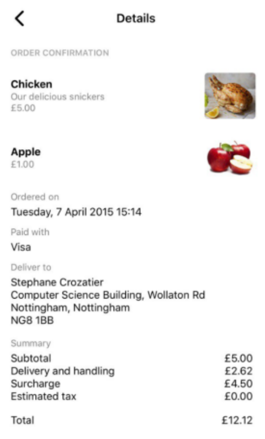
There are no orders. Maybe try logging in?

If the user has past orders, chatbot sends a "carousel" with all the past orders, with "More details" and "Need help" buttons:


Orders

Mama Janes Peri Peri Grill
£12.12

When a user clicks on a “More details” button, chatbot replies with a receipt about the order. The receipt includes ordered items with names, description and price, order time, payment method, delivery address and payment summary.



Details

ORDER CONFIRMATION

Chicken
Our delicious snickers
£5.00

Apple
£1.00

Ordered on
Tuesday, 7 April 2015 15:14

Paid with
Visa

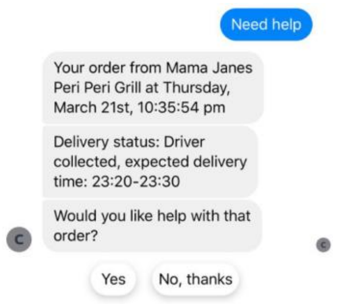
Deliver to
Stephane Crozatier
Computer Science Building, Wollaton Rd
Nottingham, Nottingham
NG8 1BB

Summary

Subtotal	£5.00
Delivery and handling	£2.62
Surcharge	£4.50
Estimated tax	£0.00
Total	£12.12

Figure 26 Receipt of the order

When a user clicks on a “Need help” button, the user is presented with a quick summary of the order, including time of expected delivery and order time. Chatbot also suggests quick replies with “Yes” and “No” options.



Need help

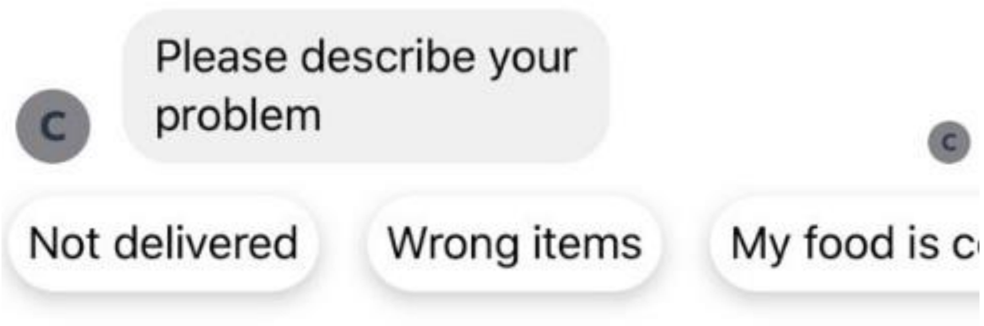
Your order from Mama Janes Peri Peri Grill at Thursday, March 21st, 10:35:54 pm

Delivery status: Driver collected, expected delivery time: 23:20-23:30

Would you like help with that order?

Figure 27 "Need help" flow

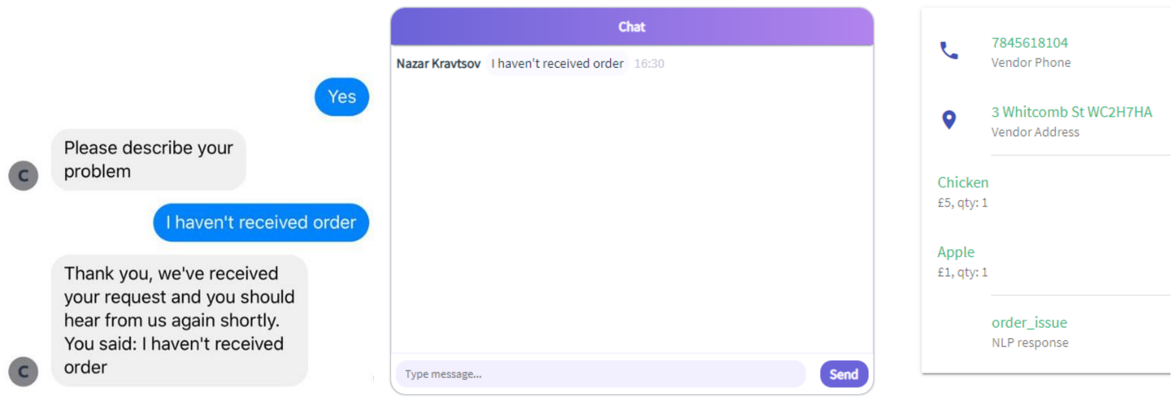
When the user clicks on “Yes”, chatbot asks the user to describe the problem. Chatbot also suggests some common problems like an order being not delivered, wrong items, food is cold, etc.



Please describe your problem

Figure 28 "Need help" flow cont.

User types in a description of the problem or click on those quick replies. After the user's problem explanation, an issue is recorded and sent to the backend server.


Yes

Please describe your problem

I haven't received order

Thank you, we've received your request and you should hear from us again shortly. You said: I haven't received order

Chat

Nazar Kravtsov I haven't received order 16:30

Type message... Send

7845618104
Vendor Phone

3 Whitcomb St WC2H7HA
Vendor Address

Chicken
£5, qty: 1

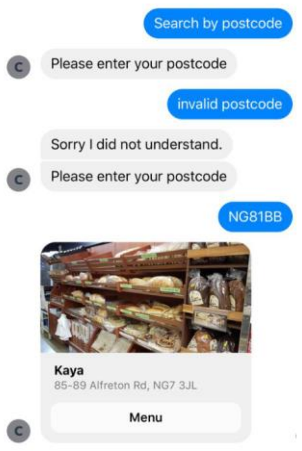
Apple
£1, qty: 1

order_issue
NLP response

Figure 29 Issue is recorded into the database

6.1.3. Search by postcode flow

This is where postcode validation is required. When the user clicks on the "Search by postcode" button, chatbot asks for the postcode input. If the postcode is invalid, chatbot replies with "Sorry I did not understand" message.


Search by postcode

Please enter your postcode

invalid postcode

Sorry I did not understand.

Please enter your postcode

NG81BB

Kaya
85-89 Alfreton Rd, NG7 3JL

Menu

Figure 30 Search by postcode flow

If the postcode is valid, chatbot then sends a request to the backend server, and then the backend server sends a request to Postcodes.io API [35], which returns a location data for a specified postcode (latitude and longitude). Then, the backend server sends a request with latitude and longitude data to Halalivery API data server. If there are any matching places that deliver to that latitude and longitude, chatbot sends back a list of available places to order. If there are no matching places or those places are closed, chatbot sends a message saying that there are no available places to order from.

7. Summary and Reflections

7.1. Project Management

The whole work, including planning, designing, developing, and documentation work was split into multiple sprints.

1. Sprint 1 – 15/10/18 – 28/10/18
   Goal: The goal of the first sprint was setting out requirements for the software and discussing the expected work with an external company (Halalivery) and project supervisor.
   Result: The sprint went as planned
2. Sprint 2 – 29/10/18 – 16/11/18
   Goal: The goal of the second sprint was to identify existing software and the needs of the product. The goal was also to develop a design of the software and identifying the tools and libraries to use.
   Result: The second sprint didn’t go as planned, because of choosing the software which was not capable of delivering based on requirements. Initially, I planned to use Botpress, but their version had no integration with Facebook Messenger at the time of development.
3. Sprint 3 – 17/11/18 – 10/12/18
   Goal: The goal of the sprint was to build a prototype of the chatbot.
   Result: The third sprint went as planned, although the prototype wasn’t able to respond properly because I required sample order data.
4. Sprint 4 – 28/01/19 – 10/02/19
   Goal: The goal of the sprint was to build a prototype of the backend server which communicates with the main Halalivery API
   Result: Some features have been implemented, but not all of them. Basic authentication was working correctly.
5. Sprint 5 – 11/02/19 – 25/02/19
   Goal: The goal of the sprint was to continue working on developing backend server.
   Result: Sprint went as planned, although authentication flow had to be redone for native Facebook log in flow. Order data fetching was working correctly.
6. Sprint 6 – 26/02/19 – 10/03/19
   Goal: The goal of the sprint was to build a prototype of a customer service dashboard
   Result: Sprint didn’t go as planned as I was still learning about Vue.js framework.
7. Sprint 7 – 11/03/19 – 24/03/19
   Goal: The goal of the sprint was to continue building a prototype of the dashboard and hooking up chatbot, backend server and dashboard together.
   Result: Chatbot and backend server communication work correctly, but I still had to spend some time in order to get dashboard working.
8. Sprint 8 – 25/03/19 – 7/04/19
   Goal: Final chatbot, backend server and dashboard deployment and adding ‘Search by Postcode’ feature
   Result: The sprint went as planned.

7.2. Future Development

The best possible way for future development is to deploy a production-ready application so that real customers can use the chatbot. It includes deploying an application to Halalivery server and creating a sub-domain specifically for a chatbot. However, there would be a need for further improvements like making linking accounts more user-friendly for mobile devices and making sure that the application fully complies with the Data Protection Act [10].

Another possible way for future development is to integrate a chatbot into Halalivery customer iOS/Android app, but that will require a complete re-development as current version of chatbot is deeply integrated with Facebook Messenger features.

7.3. Personal Development

This project has helped me learn new technologies, such as frontend development with Vue.js and chatbot development, which will be very helpful in a future career. It has also helped with experience with project planning, project estimation and time management.

7.4. Summary

Overall, the project went quite well. There were a few planning issues at the start of the development process, but overall the application works well and performs all the functions needed.

8. Bibliography

[1] Halalivery, “Halalivery - Halal food at your doorstep,” [Online]. Available: https://halalivery.co.uk/.

[2] “Chatwoot,” [Online]. Available: https://www.chatwoot.com/.

[3] ZenDesk, “ZenDesk | Customer Service Software & Support Ticket System,” [Online]. Available: https://www.zendesk.co.uk/.

[4] ZenDesk, “Ada App Integration with Zendesk Support,” [Online]. Available: https://www.zendesk.com/apps/support/ada/.

[5] Osome, “Osome - AI-based business assistant in Singapore online,” [Online]. Available: https://osome.com/.

[6] I. Sommerville, “Software Engineering, 9th Edition,” Pearson, 2011.

[7] Django-MySQL, “JSONField - Django-MySQL,” [Online]. Available: https://django-mysql.readthedocs.io/en/latest/model_fields/json_field.html.

[8] Kangax, “ECMAScript 5 compatibility table,” [Online]. Available: http://kangax.github.io/compat-table/es5/.

[9] Botkit, “Configure Botkit and Facebook Messenger,” [Online]. Available: https://www.botkit.ai/docs/provisioning/facebook_messenger.html.

[10] gov.uk, “Data Protection,” [Online]. Available: https://www.gov.uk/data-protection.

[11] Jon Jenkins, Director of Platform Analysis at Amazon.com, “Velocity Culture,” in *Velocity Conference*, 2011.

[12] Netflix, “The Netflix Simian Army,” 19 July 2011. [Online]. Available: https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116.

[13] K. Nygard, “Single page architecture as basis for web applications,” Espoo, 2015.

[14] J. Morgan, “EchoChamber: Rule-Based Semantic Webhooks,” in *International Semantic Web Conference 2018*, Atlanta, 2018.

[15] Botkit, “Botkit: Building Blocks for Building Bots,” [Online]. Available: https://botkit.ai/.

[16] Django Foundation, “Writing your first Django app, part 1,” [Online]. Available: https://docs.djangoproject.com/en/1.11/intro/tutorial01/#creating-a-project.

[17] Django Foundation, “https://docs.djangoproject.com/en/2.2/misc/design-philosophies/,” [Online].

[18] Google, “Introduction - Material Design,” [Online]. Available: https://material.io/design/introduction/#principles.

[19] Vuetify, LLC, “Vue.js Material Component Framework - Vuetify,” [Online]. Available: https://vuetifyjs.com/en/.

[20] Botpress, “Botpress - A Chatbot Maker & Bot Development Framework,” [Online]. Available: https://botpress.io/.

[21] Botkit, “Botkit Core | Botkit Documentation - Multi-message conversations,” [Online]. Available: https://botkit.ai/docs/core.html#multi-message-conversations.

[22] Django, “Django Project,” [Online]. Available: https://www.djangoproject.com/.

[23] “Django Rest Framework,” [Online]. Available: https://www.django-rest-framework.org/.

[24] Django, “Writing Views,” [Online]. Available: https://docs.djangoproject.com/en/2.1/topics/http/views/.

[25] Django Foundation, “Models,” [Online]. Available: https://docs.djangoproject.com/en/2.1/topics/db/models/.

[26] Django, “Logging,” [Online]. Available: https://docs.djangoproject.com/en/2.2/topics/logging/.

[27] Facebook, “Account Linking - Facebook for Developers,” [Online]. Available: https://developers.facebook.com/docs/messenger-platform/identity/account-linking/.

[28] Wit.Ai, Inc, “Wit.AI,” [Online]. Available: https://wit.ai/.

[29] Vue.js, “Vue.js,” [Online]. Available: https://vuejs.org/.

[30] Li, XiaoLong, “Application of MVVM Design Pattern in MES,” in *5th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, Shenyang, 2015.

[31] Axios, “Axios - Promise based HTTP client for the browser and node.js,” [Online]. Available: https://github.com/axios/axios.

[32] Webpack JS, “Webpack,” [Online]. Available: https://webpack.js.org/.

[33] Nginx Inc., “NGINX | High Performance Load Balancer, Web Server & Reverse Proxy,” [Online]. Available: https://www.nginx.com/.

[34] Ong Sing Goh, Cemal Ardil, Wilson Wong, Chun Che Fung, “A Black-box Approach for Response Quality Evaluation of Conversational Agent Systems,” 2010. [Online]. Available: https://www.semanticscholar.org/paper/A-Black-box-Approach-for-Response-Quality-of-Agent-Goh-Ardil/39eaeb84cb76ea8e580989cda1459ed7f5eea1df.

[35] IDDQD, “Postcodes.io,” [Online]. Available: https://postcodes.io/.