Investigating the implications of Conversational Commerce on OLTP Data Entry

Harry Messenger

School of Computing and Communications, Lancaster University

Computer Science

Declaration

 I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

Abstract

Traditional OLTP (On-Line Transaction Processing) orders are placed through order forms on standard websites. Recent innovations and opening of chatbot marketplaces such as Facebook Messenger, Slack and Telegram have brought the "chatbot" into the 21st century. This paper discusses the implications of utilising chatbots, or more specifically Conversational Commerce, the act of facilitating sales via such an agent, on OLTP transactions – those where there is no "sale" component, rather an order being placed.

Using the example of Playdale Playgrounds, a company who rely on a B2B distribution network to sell their products, this paper attempts to design, build and test the effectiveness and efficiency of using a chatbot to place orders in comparison to the traditional web "order form".

Results of this project show teething problems in the platforms involved as well as initial reluctance of users, however proof of concept and potential for future work and development is evident.

1. Introduction

1.1 Introduction to Playdale Playgrounds.

Playdale Playgrounds, the company which this case study is based on, are an international manufacturer and exporter of high-quality, British-made playground equipment. Based in Haverthwaite, UK, the company sell to two main markets. Firstly, they sell a large amount via B2C (Business to Consumer) methods in the UK, utilising both traditional e-commerce sales via their website as well as "over the phone" or catalogue sales. The main customers for this area of their business are schools and government authorities, as well as wealthy private customers. Additionally, they have a large export business selling to countries across the world. This is made up of an international distribution network of over 30 companies, operating across multiple continents. These export orders make up a hugely important area of their business not only currently, but with a huge scope to expand in the future. The business model of these export sales is a symbiotic one, as they can leave the actual selling to a third party, and charge a lesser fee (the third-party companies' profit margin) because of this.

1.2 Introducing Conversational Commerce

Conversational Commerce, defined by Chris Messina as "utilizing chat, messaging, or other natural language interfaces (i.e. voice) to interact with people, brands, or services and bots that heretofore have had no real place in the bidirectional, asynchronous messaging context"[1] This recent phenomenon, driven by the increase in use of messaging apps, which in 2016 eclipsed the use of even social media[2] is seen by many as the natural next step for e-commerce in a space where traditional web traffic is decreasing in favour of smartphone apps, messaging services and other "on-demand" methods of interaction. In a 2015 talk at Lancaster University, Microsoft's James Whittaker spoke of the move to a more transaction-based online economy, with small purchases and micro-transactions being more readily available at our fingertips[3]– and with WhatsApp's 2015 move to become a free app as well as introducing an API[4] to facilitate third-party conversational interactions and transactions the move to this transaction-based economy has begun, with the majority of major players including Microsoft, Facebook, Slack and Kik also beginning to support Conversational Commerce.

1.3 Application of Conversational Commerce to Data Entry

Ever since 1993, when HTML first defined the Web Form, using a form to enter data online has been pretty much the exclusive de-facto standard of data entry. With thousands of systems being "digitalised" every month across the globe, more and more Web Forms are being created with the aim of moving outdated, legacy systems online (As with Playdale, the case study for this project, are doing with their Export Orders!).

Despite the scope of this transition to online data entry, hardly any work or research has gone into alternative techniques for OLTP-style data entry, such that these legacy systems would require. A brief search of research journals finds hardly any reference to this topic, and no significant research into other digitalisable techniques for data entry has been

conducted. In this paper, we attempt to bridge this gap by utilising our discussion of Conversational Commerce chatbots to create something new.

1.4 Project Aims

Although Conversational Commerce is a great baseline term, further work is needed to investigate the application of this technology to the commercial world. In this case study, we look at the potential application of these techniques to the major industrial field of OLTP data entry and investigate via a case study the potential for placing business-to-business (B2B) orders via a chatbot.

The problem domain that this paper tackles can therefore be worded as a research question:

"Are Chat-bots a viable alternative to web forms for OLTP data-entry style systems and how can we measure their effectiveness and performance?".

In order to address this research question, a case study will be carried out based on Playdale Playgrounds B2B distribution network. This will take the form of building of a chatbot to conduct Playdale's OLTP transactions as an alternative to a web form.

Included in this report will be discussion of the background of chatbots and Conversational Commerce as well as give some insight into motivations, research and current methods of OLTP, as well as data entry. In addition, there will be a high-level overview of the work conducted for Playdale's new Export Order System (the groundwork for this case study as the baseline UI that we will be comparing our Chatbot against) as well as a review of existing Chatbots and some of their features and flaws. Details of technological choices, implementation, technical challenges, an overview of the finished Bot are contained within Chapters 4 and 5 with some limited user trials of the prototype and their thoughts on the system in comparison to existing methods found in Chapter 6. Finally, there will be a discussion of the future of this method of data entry as well as a review of the case study.

2. Background

2.1 Introduction - the groundwork placement project

During the project carried out between February - March 2017, a new "online" system for placing "distributor" orders for Playdale was created. The goal of this system was to optimise and improve their outdated existing system, which took the form of a purchase order, placed by filling out either Playdale's official paper form or using one of their own. With no standardised format or even standardised fields being required, it becomes a time-consuming process to "decode" the returned purchase orders as well as having to match all product codes to the system and attempt to manually calculate all the prices. This project therefore was to develop an "export order system" for the company - an online portal to replace existing paper/email/fax based export sales system that the company currently use. Required was for the system to be a central repository which houses and holds every order placed by the distribution network.

Benefits which Playdale hoped to gain from such a new system include having an accessible record of every order placed by each client as for both Playdale and their distributors, as well as orders having a standardised format and input method as well as automatically calculated prices and discounts. In addition to this, the paper-based system frequently causes issues with the orders that come in, such as incorrect product codes, incorrect prices, delivery dates being selected that aren't allowed as well as not including enough information in terms of order information and contact details. Standardising a format and providing validation for these would provide a better customer experience as well as streamline the process for accepting these orders. A further goal of the project was to automatically export the orders into Playdale's Sage CRM system – which was manually completed until now.

2.1.1 Interfacing with the system

The system which was built for Playdale between February and March 2017 was successful, allowing exporters to follow a standardized order form and submit their orders. Of note was the UI design of the form – important as this is the main point of interaction with the system the exporters would experience on a regular basis and therefore was one of the key elements of the system. There is a whole field of research about developing good user experience, with many differing opinions on how to achieve this, however in practice this boiled down to making the page visually appealing and using components which were obvious and easy to interact with. The best example of this was the twitter-style search functionality based on Typeahead.js [5] allowing users to search for product codes which exist and select the correct one. This core UI functionality was complex to build and included caching and image lookup to allow users to reference their catalogues and materials to see if the item that they were trying to order was the correct one.

2.1.2 Interfacing Issues

Although successful, the project interface of course had some issues. Firstly, browser compatibility for the Vue.Js components is not ubiquitous and therefore not ideal for all

users. Additionally, UX and UI components are of course subjective, and what works for some people might not work for others on an enjoyability and usability front. This lead to a short debate with Playdale staff about whether having a single data entry method was the best way to go about this – while ultimately we decided in the scope of the project a single method was indeed fine, this presented an opportunity for research into other effective ways of processing online transactions without using a traditional form for data entry – aimed at providing an alternative approach for those who don't find web forms intuitive to use as well as specifically to target the growing "smartphone generation".

2.2 The History of Conversational Commerce

The term "Conversational Commerce" was coined by Chris Messina as late as 2016 to describe the modern phenomenon of using a NLI (Natural Language Interface) to conduct e-commerce transactions. The idea of using a NLI however is much older – indeed, chatbots themselves have been around for over 50 years! In 1966, Joseph Weizenbaum published ELIZA, and introduced the idea of using a computer program to simulate a conversation had between two humans [6]. While simplistic, the "chatterbot" or chatbot was already able to parse and substitute answers and responses into a template – the basis of even modern day chatbots. In doing so, he discovered the "ELIZA effect[7]", where the participants gained a form of empathy and subconscious human attachment to the machine and its responses. This effect is one that future chatbots, specifically those targeting conversational commerce or support systems attempt to benefit from, as this empathy can be useful to help a customer or user feel at home or familiar with the system.

As time moved forward, Computer Scientists such as Kenneth Colby [8] and Rollo Carpenter [9] attempted to include more "human" behaviour – at first by mimicry and addition of spontaneous phrases designed to follow simple scripts but moving forward the inclusion of some AI elements and natural language processing allowed more complex conversations and pattern matching to begin to be developed. A.L.I.C.E [10] was one of the first bots to use a form of NLP. Pattern matching algorithms which attempted to mimic human responses by replying based on the input allowed A.L.I.C.E to have a "conversation" with the user which was much more technically complex and therefore more believable in comparison with the ones which had gone before.

Moving into the twenty-first century, chatbots began to fade in favour of more complicated AI systems and NLP interfaces. IBM's Watson won Jeopardy in 2011[11], only 5 years into development – and was able to function successfully in a way that mimicked human behaviour throughout the show, and "personal assistants" – voice activated NLP programs that had access to cloud processing and information retrieval such as Apple's SIRI, Google Now, Amazon Alexa and Microsoft's Cortana began to dominate the research and interest space, as well as moving NLP into the mainstream consumer space for the first time – indeed having a voice assistant is now an expected feature of all modern smartphone systems and indeed is considered a selling point with devices such as the Amazon Echo and Google Home commanding a share of an expanding market. These connected devices can do far more than the humble chatbot with voice-search systems allowing interaction via a Natural Language Interface (NLI) with other services via a system of API calls designed to

obscure the systems behind the scenes and make the interaction feel much more like talking to a single person than using a complex network of websites and interfaces.

These modern "assistants" again allowed humans to experience the ELIZA effect - with many people reporting the feeling of enjoying the feeling of the "human" interaction. This effect is also documented in popular culture – in an episode of the TV sitcom The Big Bang Theory, the effect was parodied where one of the characters interacted with the Siri assistant and developed an attachment. [12]

Moving into the latter half of the current decade, companies began to see the success of the commercialised vocal assistant NLIs and adapt their strategies around this. Messaging services such as Facebook Messenger, Slack, Kik and Weemo amongst many others already had both user interfaces and an established customer base so were amongst the first to create open source APIs to begin to commercialise the Chatbot space. Facebook launched Messenger Platform [13], to allow third party developers to create bots with permissions to access and interact with users and their data in April 2016, and by September of that year over 30,000 bots were already available [14] as well as permissions to conduct Conversational Commerce via an integrated payment gateway, legitimising the concept of conversational commerce as a viable market for the first time. These chatbots can respond with links, messages and provide "call-to-action" and even allow purchases to be made directly in the messenger interface [15]. Criticisms of this Facebook implementation was although these bots were styled as "chatbots" they are however realistically more like the early ELIZA style bots following scripted conversations with specified inputs, much like a command line program might be.

2.21 Research into Conversational Commerce
Conversational Commerce, while a new topic, can be considered related to M-commerce. Ngai and Gunaskaran [16] found that by 2007, many services already let their consumers access and conduct e-commerce activities on mobile devices, enabling firms to not only provide more ubiquitous access to their customers but to "establish and maintain more direct relationships with their customers" [17].

Van Eeuwen found that in a survey of 195 millennials, most respondents are already familiar with using their phone for shopping, and have done so at least once – however the majority do not have any experience with existing chatbots, but responded somewhat positively to their concepts and ideas presented as a demo video [18].

Van Manen [19] reflects that the strengths of Conversational Commerce are that the platform used already has a user base, and the users already possess the skills required to interact with the interface, as opposed to a third-party app or site which they would have to learn. He also suggests the convenience of this platform alongside the personalization available can be used as a marketing technique to "provide convenience, personalization and support decision making" [19]. The seemingly unbiased dialogue with the "agent" can persuade a user to make a specific decision.

2.22 WeChat – A Case Study

WeChat is a Chinese "messenger" application. With over a billion registered users and 785 million daily users as of November 2016 [20], WeChat is the largest social messaging app in China and one of the largest in the world. Notably, however, WeChat is exceptional on many demographics of financial flow and specifically has an estimated $7 ARPU (Average Revenue Per User), thought to be over 7x higher than a US competitor, WhatsApp in a similar timeframe [21]. WeChat started out as a messaging service much like WhatsApp and Facebook Messenger, however quickly became much more than that, offering increased functionality including options to request Taxi services, buy Movie tickets, play games, rent music and books and even donate to charity, all from the space of one app. This ubiquity of service allows users to stay within one integrated app for a lot of their mobile needs by using the medium of chat as the only user interface. This is managed via a system of "official accounts" (over 10,000!) which have access to APIs and payment portals allowing this functionality to be offered, much as Facebook Messenger is starting to allow with it's new Developer Libraries[13]. In order to facilitate payments on these micro-services, WeChat has a centralised payment system which allows the user to configure their payment in one place without having to pass out payment details to each individual provider [22].

All of these factors lead to what is without doubt the biggest and most successful model of Conversational Commerce available today, and while showing year-on-year growth of over 35% [20], WeChat is one of the cornerstones of proving the legitimacy of Conversational Commerce.

2.23 OLTP data input methods

The TPC (Transaction Processes Performance Council) define Online Transaction Processing (OLTP) as the "updating of a database system for things such goods, services or money".[23] This is a loose definition as frequently OLTP refers to large-scale or high-speed transactions, however in this context we refer more to the transaction itself than the speed. The overwhelming majority of current web transactions take the form of a database post request made via the interface of a simple form.

The HTML form was included in the very first HTML spec dated 1993 [24], and has made up a key part of the internet process for as long as the web has been around. Ever since then, as a data input method, the HTML form has been pretty much universally accepted as the only method for data entry online. Alternatives have always been tangible, as opposed to on the computer, and include phone calls, manual entry on paper forms which could then be emailed or faxed. Aside from this, very little research appears to have been done into an alternative method for data entry, potentially because of the lack of need – web forms do work well and there is no real drive to replace or supplement these. Much of the research in the area is instead about optimising the user experience of forms and checkouts as opposed to discovering alternative data entry techniques.

2.24 Conversation without the Commerce

Having said all of this, the question of if there is any scope for improved data entry, or any space for improvement of the traditional web form is still a question, and the research on Conversational Commerce as well as seeing some moderate success in china and recent US

innovation in the field lead to the creation of a research question: "Are Chat-bots a viable alternative to web forms for OLTP-style systems and how can we measure their effectiveness and performance?".

In order to answer this research question using the case study of Playdale, a demonstration prototype of a bot will be designed and built. This will then be subjected to a number of user trials, specifically with some expert users from the team on which the project was developed alongside last term, as well as a few tests on non-experts to determine if there is any difference in usability between experts and non-experts. Due to the timescales of this study, the number of tests will be limited, however hopefully enough information will be gathered to give an early indication of the viability of this technology for use as an alternative data input method.

2.3 Existing Bots

In order to gain inspiration for the user interface and the general "feel" and "style" of the current or existing chatbots on these marketplaces, two existing chat-bots were used and studied. The goal of this was to gain insight into the state of existing work in the field, what was possible in a short time-scale and the "feel" of using such an assistant to place an order or access a service. Unfortunately, no "data entry" chatbots are currently available as this is a new research area, so the general field instead was investigated.

2.31 DomBot

The first bot being analysed is operated by Dominos Pizza[25]. The bot is hosted on Facebook Messenger, using the Facebook Messenger Developer Platform (FMDP) and it integrates the company's existing EasyOrder application and website into the Conversational Commerce platform that Facebook offers. The bot is designed to replicate the functionality of the existing easy order technology and offers buttons and selection boxes as well as manual inputs in the style of a traditional chatbot or Messenger conversation. The pictures and selection boxes are implemented using FMDP's Send API, more specifically Generic Templates [26]. An example of this can be seen in Figure 1.

Payments on the app are not handled directly, but via a third party system integrated with their existing EasyOrder, however using the FMDP APIs it is now possible to take payment through the platform via the Beta release of the payments API [27].
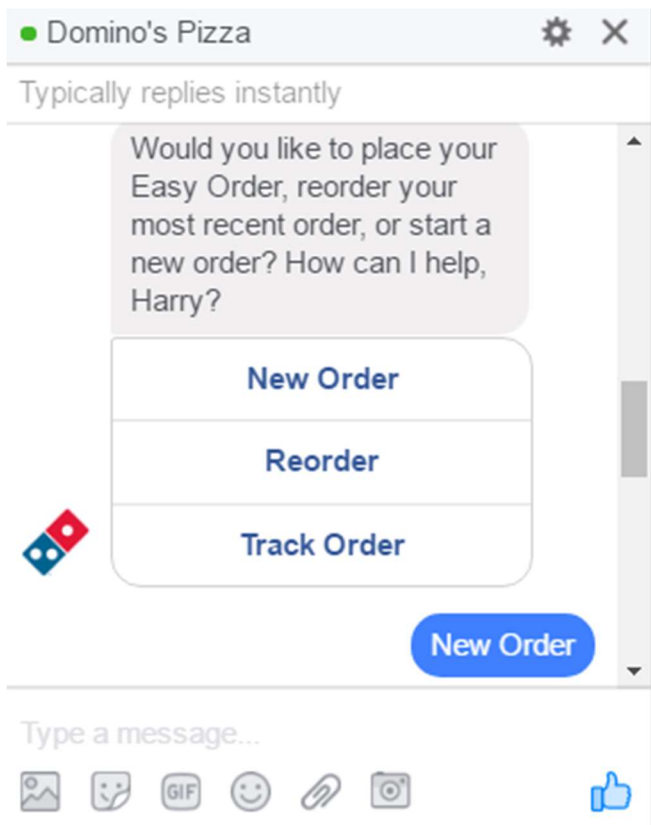
Figure 1: DomBot shows a Generic Template using the FMDP SEND api.

Other take-aways from using the Dom Bot was the ease of integration with their other system (EasyOrder). Linking your account with one click "login" opens a handy API window which allows you to link an existing EasyOrder account with your facebook account and access old orders and saved payment details.

One downside of the bot is when you make a mistake the error is handled and a message is returned, however the question is not repeated and instead you are expected to re-enter your answer to the question with no additional prompt. I feel like this is not a limitation of the platform, however, but rather a limitation of the implementation on the platform.

FMDP is a series of platform agnostic web APIs, meaning that any language or technique can be used to call the API and is controlled by a series of webhook-based callbacks, and the Dominos Bot is fired on the other end by using a Node.JS based API available on NPM. [28]

Overall, core takeaways of the experience of trialling DomBot are that the "conversational" feel is not restrictive even though the opportunity for free text (as opposed to button input) are limited, and that error handling is acceptable but not exceptional. The DomBot is effective at its job and handles data entry well, giving hope to this project.

2.32 SlackBot

Slack is a project management and team communication app. Its name is quite self-descriptive, the acronym SLACK standing for Searchable Log Of All Conversation and Knowledge [29]. The app catalogues and saves the chat, text, file uploads and conversation between users and allows these to be referenced later. Slack, as a messaging and conversation app, was naturally in a position to introduce a third-party API to create bots to interact and converse in it's chat channels, much like Facebook. However, unlike FMDP, the Slack equivalent is not intended for Conversational Commerce, but for help, support and personal assistance. The example of the bot we trialled was "SlackBot"[30]. Included with every new installation of a Slack channel, SlackBot is a "help and support" style bot. Long gone are the days of simple documentation, SlackBot can interpret your question and answer with the relevant section of the documentation or other information. Figure 2 is an example conversation with SlackBot.
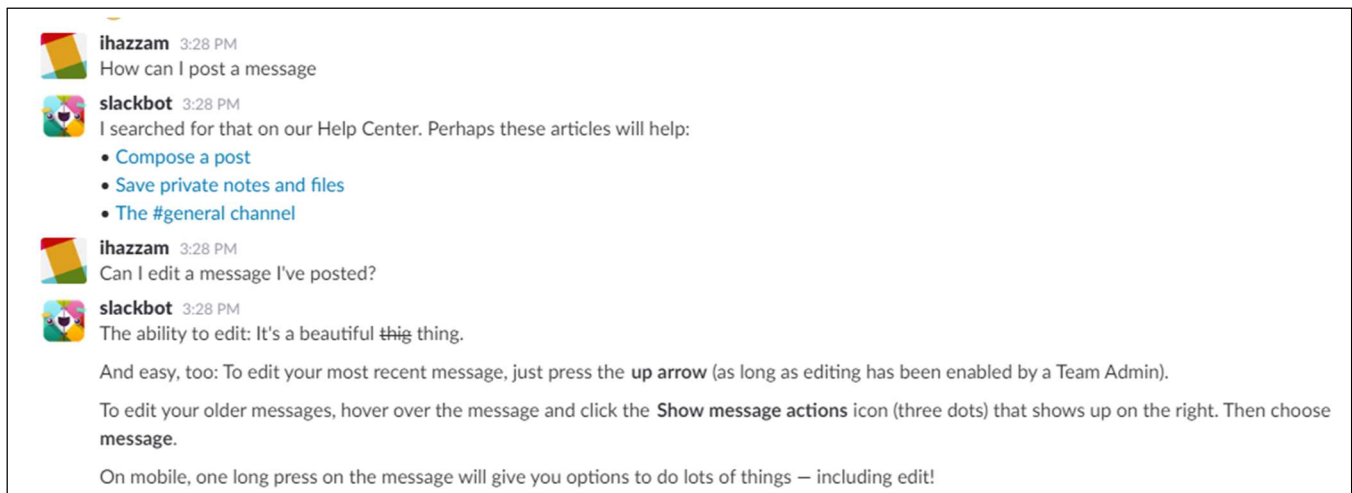
Figure 2: A conversation with SlackBot

As seen in figure 2, the bot can again monitor and parse messages via a webhook-based system, similar to FMDP, as well as post messages and reply to users. The implementation is slightly different relying on websockets as opposed to callbacks but both systems have buttons, text responses and API access.

2.33 Reflections

Core take-aways from using the SlackBot are that the system feels more refined to use than DomBot – the NLP interface being used to interpret the questions feels well refined however clearly much more complex to code. The idea of being able to receive comments in multiple conversations and threads was exceptionally useful – however for a single conversation style such as Playdale would require in our case study this style might not be as useful.

The initial review of the technologies provides plentiful evidence that the concept of using a chatbot for data entry tasks is technologically feasible. There is very little evidence that any research has been done into this concept in the past, however innovation partner network VASPP [31] prototyped an industrial concept, the VBot, in November 2016. VBot is a chatbot to streamline the data entry of booking support tickets to a task system, and the fact that the area of data-entry chatbots has already been commercialised indicates promise in the area. However, as a private company selling a service, VASPP are not required to publish details of their operation and therefore there is no evidence of popularity or effectiveness available from this operation, as well as no user feedback from the perspective of expert or non-expert users. The goal of this study, in respect to this, is therefore to try and further the understanding and investigate the premise further of chatbot data entry.

3 Design

3.1 Technology Selection

In order to build a prototype "PlayBot" to demonstrate the concept of a data-entry chatbot several aspects of technology need to be selected in order to facilitate development. The main points of contention include

1) Platform
2) Language and any Libraries or SDKs
3) Natural Language Processing API (optional)

Things that were considered in making these decisions included the previous project, built using Laravel PHP and Vue.JS, as well as the existing skills of the developer and the timescale in question – only two weeks for development means that using a long timescale to learn new technologies might not be possible. In addition, the experiences gained in trialling the platforms discussed in section 2.3 as well as the anecdotal experience of using both platforms over the last 3 years will assist in the making of these decisions.

3.2 Platform

Having trialled both platforms (Slack and Facebook) a comparison of the two can now be made. Although the Slack ecosystem seemed more polished and impressive, for our use case the Conversational Commerce aspects of the Facebook platform as well as Generic and Receipt template options fit the goal of single-conversation data entry well. Additionally, across the globe Facebook has more ubiquitous usage and market penetration, so potentially could be a better choice.

3.3 Language

Being that use of the Facebook Messenger Development Tools (FMDT) are a series of APIs that can be called from any platform by making a series of GET and POST requests, therefore the only limit to which language can be used is the skills of the developer and the ability to make API calls and accept webhook postbacks. Considering that, a web-based server should be used and below we consider a few of the viable options.

3.31 Node.js

One of the most popular web frameworks in recent years, Node.JS is a JavaScript based runtime environment built to allow JavaScript code to be run on the server. Event driven and Non-blocking [32], this would be a good option to run a web server for the goal of running a Facebook Messenger bot.

The language itself is familiar, as the syntax mirrors Javascript used on the frontend – and allowing the server to listen to requests is as easy as writing the command "listen" (see figure 3)

This allows us to forward our Bot requests to the server via Facebook's webhook system that is configured in the back-end of their application.

```
// Spin up the server
app.listen(app.get('port'), function() {
        console.log('running on port', app.get('port'))
})
```

Figure 3: Listen to incoming HTTP requests on a Node.JS server

The non-blocking architecture of Node makes it a great solution for a server like this where multiple simultaneous requests will be handled at the same time with multiple concurrent users, however concurrency within a single request is not handled. This would be less ideal for an architecture where multiple API calls to third-party providers are being made, however for simple applications Node.JS makes a lot of sense – to the point where it is the language used in the official example in the documentation for FMDT. The usual way to use Node.JS in a local environment is to deploy to a service like Heroku, however this does increase the complexity of the setup and requires deployment be considered as part of the development process.

3.311 Node JS library - Remixz/messenger-bot
In order to aide rapid development (the timescale for the design and build section of this case study is under two weeks!), it would be desirable for the project to be based around a third-party library. These libraries provide methods and pre-composed functions to facilitate the API calls required, and can speed up development time.

Remixz/messenger-bot [33] is a Node.JS framework built by Zach Bruggeman, intended to allow easier access to the core functionalities offered by the FMDT bot APIs. Offering a simple Bot instance, simple interfaces are offered over the "send" "senderActions" and "userProfile" APIs as well as a couple of buttons and menus, however essential tools such as templates do not have methods. This library appears mostly complete and has tests covering 89% of the code, however has many open issues on Github and no fixes for around 5 months at the time of writing.

3.32 Python/Flask
One of the more common languages for writing modern web applications is Python. Again, Python is not naturally necessarily a web server language, however there are plenty of ways to create such a server so this isn't a huge issue in this case, as we can use Flask or Django, frameworks that provide this functionality.

Flask[34] is a microframework to add the ability to do web development using Python, providing RESTful request dispatching and unit testing as well as much more. The ease of use as well as the lack of required dependencies, tools or libraries make Flask the most popular Python web development framework on GitHub.

Python's ease of learning and clear, easy-to-read syntax make using it for this task a possibility, as the documentation and large availability of sample code make the learning curve less steep than other languages. Python also has a huge availability of libraries that are easy to pull in and extend the core language.

3.321 Python library – Davidchua/pymessenger

Again, there is an unofficial library to offer enhanced out-of-the-box Facebook Messenger functionality for Python. Pymessenger [35] offers an abstraction over the whole "send" API offered by FMDT, including templates, however is much more limited in terms of core functionality and actual configuration of the bot, and is much less mature as well as not having tests making the library less reliable.

3.33 PHP/Laravel

Potentially the most obvious option is to use PHP – the original language used to create the Playdale project from last term. This has the added benefits of developer knowledge – already being versed in the language goes a long way to being able to develop the prototype bot within the two-week programming deadline.

PHP is a web development scripting language that has been around for a lot longer than both Python and Node.js. Written for the web, it has the advantage that everything is native to Web Development rather than having to be adapted as with Node and Flask.

Laravel[36] is a framework that brings a lot to enhance the core of PHP. Laravel includes features such as reverse routing, request handling and formatted http responses. Features such as these allow applications to be built without having to manually write as much code from scratch. In doing so, Laravel adds the Model, View, Controller architectural paradigm to PHP, benefitting the development experience as well as control flow immensely. One downside of this is the less-than-easy expansion of the service to huge web applications with thousands of concurrent users, however this is not going to be a problem building a Facebook bot for this purpose.

3.331 PHP library – mpociot/botman

Botman [37], by Marcel Pociot, is a PHP library that sits on-top of any chosen web framework (in this case, Laravel) which "simplifies the task of developing bots for multiple messaging platforms"[37] including Facebook Messenger. Of all the libraries reviewed, Botman is the largest and most developed, with around eight times the number of commits and stars on Github than the others, potentially due to the support of multiple platforms.

Botman offers again a full Bot instance, with interfaces over the Send API as well as other FBDT APIs. Botman also handles the received messages inherently using Laravel's Routing and controllers, giving it a natural advantage over the other libraries which don't offer this functionality. The ability to have multiple separate defined "conversations" which can offer the type of process flow needed for the application is again pleasing for the task at hand, as well as the requirement of offering interfaces over the "templates" offered by FMDT make this the library with the greatest functionality offered.

3.34 Language and Library Choice

After considering the pros and cons of the various languages, due to the short timescale of the project, sticking with a language already known by the developer (PHP) seemed a natural progression, even though the architecture of a Node.JS server would suit this specific use case slightly better and offer greater expandability. However, for a prototype this isn't a big issue. Added to this fact the obvious superiority in features and

documentation provided by the Botman library this lead to the decision to stick with a technology stack of PHP, Laravel and Botman.io.

3.35 Natural Language Processing

Natural Language Processing - or NLP – in this scope allows the ability to gain "intent" from the instruction or words given to the Bot. This is very powerful, as it allows the conversations to flow more naturally than a structured conversation, and therefore seem more realistic! However, this obviously comes at a cost, both in terms of development time, performance and complexity. Historically, this has been one of the major limiting factors of chatbot development as developing a system to parse language and complete NLP is very technical and outside the scope of all but research investigation. However, with this technology like any other moving into the 21st century and with the advent of SaaS (Software as a Service), companies have begun to offer Natural Language Processing as a service – which allows Bot platforms like Facebook Messenger to include aspects of NLP. There are a couple of options for the prototype in terms of this addition, and these are using Wit.AI, API.AI and not using any NLP.

3.351 WIT.AI

As part of developing the Facebook Messenger Platform and facilitating the use of conversational commerce, Facebook purchased a NLP as a solution service, Wit.AI [38]for an undisclosed amount. Wit.AI provides a HTTP-based API service to translate a question or statement into intent, an example can be seen in Figure 4.
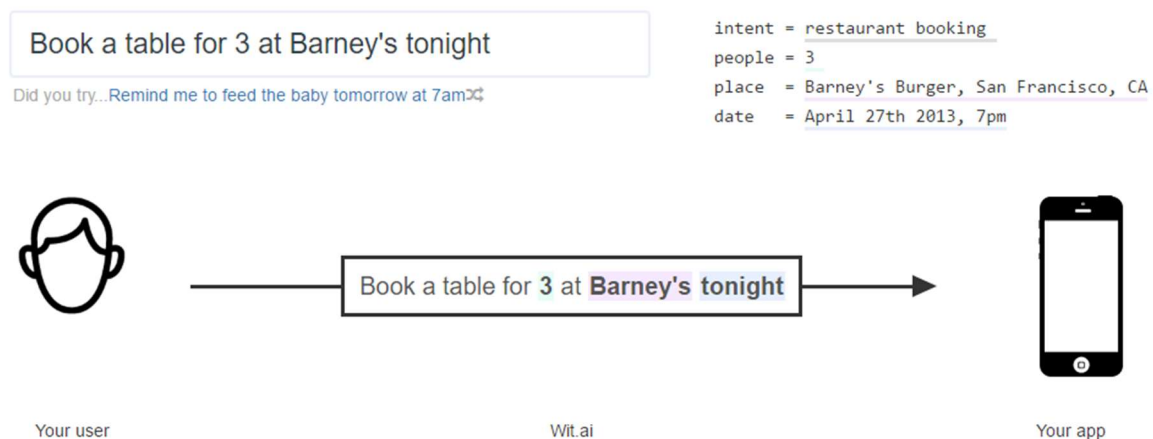

Figure 4: Example of Wit.AI input/output

This provides useful context information to the application making the request and helps translate between human "meaning" into something a machine can understand. The data is returned as JSON objects which are easily parsed using PHP, however there is no official PHP SDK to work with the API (but there is one for Node and Python). In addition, the API is free to use.

3.352 API.AI

Botman, the PHP library being used, has built in support for another NLP API, known as API.AI [39]. Similar to Wit.AI, it receives a query, or "event" and translates this into "intent" using the agent's learning model.
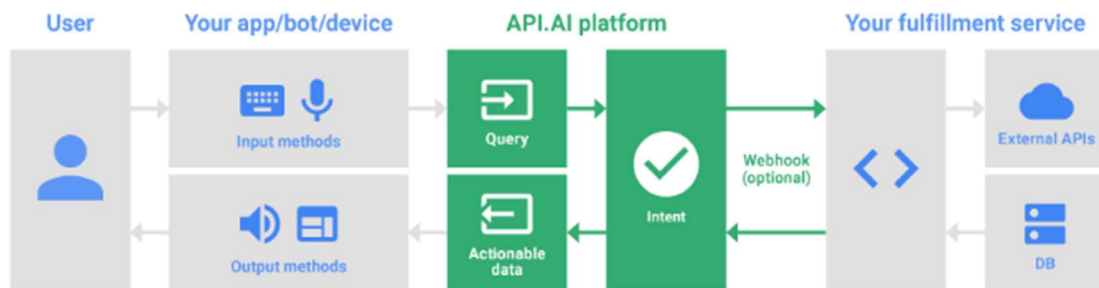

Figure 5: API.AI "translation" service in its place in an app. [40]

Figure 5 shows how Api.ai fits into the architecture of a client-server application such as a Facebook Messenger bot. As opposed to Wit.AI where you would receive a message from the client, process it via the API and return it to the app before processing it and sending it to the client – whereas in the case of API.ai the service hooks directly into the Facebook system and receives the data directly. This is more convenient in some ways as the data is passed less times in total, in theory speeding up the service, however also gives you less control of the moment when you process data and make requests to the server.

3.353 No NLP

The third option is to not use any form of NLP. This option initially sounds sub-optimal, due to all the benefits mentioned before – however as a data input device, there will not be many questions asked by the user in this specific Chatbot – and following a stricter conversation as seen in other Conversational Commerce as well as in the Dominos DomBot app will work very well. Other benefits of not using any NLP include faster development time as well as a lower overall complexity leading to faster response times by the chatbot.

3.354 NLP decision

Having considered all of the options, the decision which made sense was to go without NLP for this Bot. This was unfortunate as the technology is very exciting and makes the chatbot "feel" better to use, but for the use case and with the timescales the use of NLP doesn't make much sense. That having been said, a future production version of the bot could use this service, and from the evidence seen, both API.AI and Wit.AI appear to achieve very similar results – full performance testing for the use case would be required to pick between the two – however both seem to be easy to configure so this would not be arduous, but the natural integration of API.AI with Botman would make sense to use this library, if either were to be used.

3.4 Design - Architecture

Having planned the technologies to be used, the next step was to plan the architecture. The requirements of the project are a web server that can receive and process webhooks

returned to a single URL, as well as make post requests via Botman. In addition, information needs to be transferred between the Facebook Bot Server and the Playdale Server hosting the project built last term. The reasoning for this is that the orders will still need to be placed with this second data-entry system.

The solution to this is to have the two web servers as well as build a RESTful API on the Playdale server, and use this to communicate. Additionally, the server must communicate with the Facebook servers. The overall architecture will be Client->Server, however inside both the Playdale server and the Botman server, the two Laravel PHP systems will be Model-View-Controller. The architecture diagram can be seen in Figure 6.
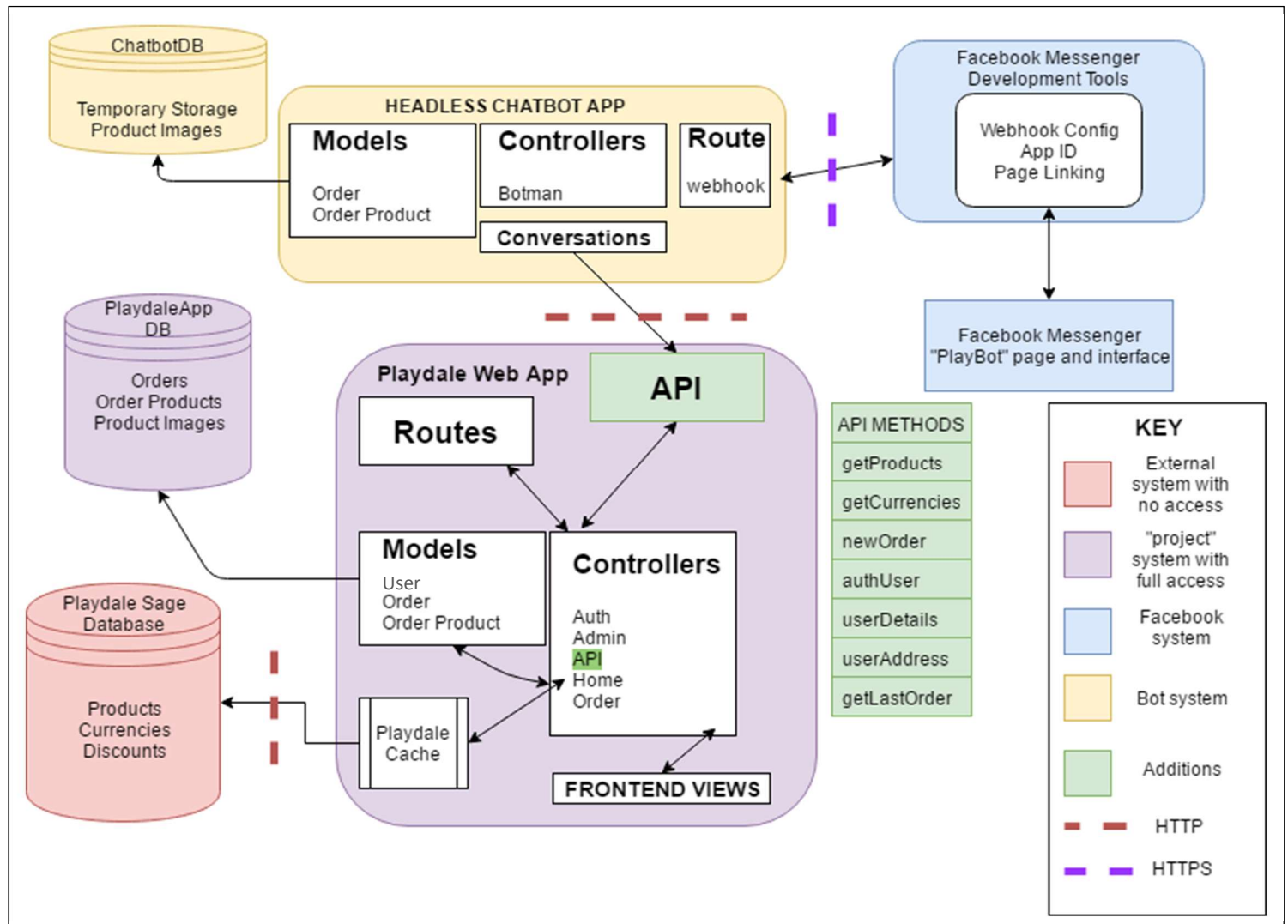
Figure 6: Architecture and API diagram

Additionally, there will be some caching and data duplication. Deciding what to cache, what to duplicate and what to simply make API calls to fetch is always a give and take of performance versus resource usage, so making the decision on which to cache has to be based on real-world testing and understanding.

One area that will be duplicated is the database record associating product codes with image URL locations as this will cut the number of API calls required to fetch the image for

products in half. Aside from this, the headless bot-app will be very reliant on the API instead of the persistent database, with hardly anything being stored. Due to the way Laravel models work, the database is necessary – however something simple and low overhead such as a filesystem based SQLite database will work fine.

3.5 Design - API

In order to facilitate the required functionality in the chatbot, it was required to have an API to interface with the existing "order" systems designed and built last term. This system, built on Laravel PHP was designed to have API-like characteristics in the first place, so in theory, the addition of an API should not be technically complex. To design said API, it was necessary to first work through a "model" of how the chatbot conversation was going to go. Based on the form, the goal of the conversation is to be able to mirror the exact input of the form with the new interface – the reasoning for this is to have to change as little as possible on the (finished) system from last term, as the Playbot system will "post" the data over and then the data will be processed as if it had just arrived from the form.

This strategy decided, the next step was to figure out what data would be required in the two systems. The concept of a user-scoped "user ID", passed directly with every request from Facebook Messenger's webhook is essential to knowing that the correct user is talking to the server. This will have to be sourced and added to the Playdale User records.

On the other side of things, things that need to be requestable (GET) from the API include product details given product code, user details given FBMid, as well as the user's last order and products details. The only required POST method is going to be the ability to submit a new order.

3.6 Design – User Interface

Being a non-traditional interface, the design for the UI of a Facebook Messenger bot is very different to a traditional, front-end system. Instead of thinking about placement of components, design of front-end forms, javascript and CSS, we must think about User Journey through the conversation, as well as placement of pictures and use of templates.

3.61 Facebook Messenger Templates

One of the major draws of the Facebook system is the excellent UI components that make up the templates library. Buttons are core to any system, especially a non-NLP chatbot – as limiting the choices of the users is critical in making the system appear fluid and responsive with the minimum possible opportunity for the user to make errors. FMDT and the system notices this and includes buttons as standard into the options, both to cause callbacks (basically send a "pre-determined" message) and directly link to external resources. In addition to this, the system includes various other templates. The "Generic" template allows a picture and some information to be sent with some buttons, allowing information to be displayed in a convenient and useful form – especially conveying extra information via a picture. Figure 7 shows a Generic Template in use in another application. The UI design plan here is to use generic templates to show the items requested in product codes – a la the "twitter style dropdown" that makes up a core part of the UI on the web form in the original system (see Figure 8).
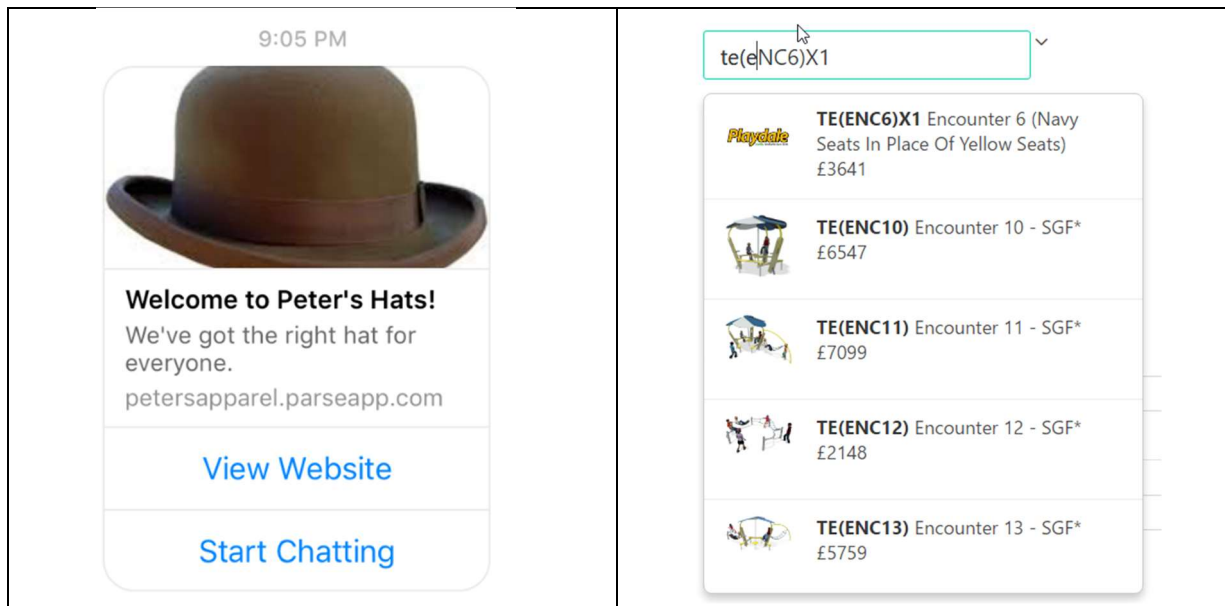
Figure 7) A generic template in use   Figure 8) Playdale Order Form Dropdown

In addition to the Generic Template, for displaying the finished order as well as for displaying the "last order", the API provides facility to generate a "Reciept" template. Although we are not conducting any Conversational Commerce using this app and the format is not that adaptable, this potentially will still work and allow a better looking and feeling UI overall.

3.62 User Journey

The design of the conversation will be one of the most critical parts of the success of the prototype app, and will be heavily reflected in the user feedback. The goals of this conversation are to effectively facilitate data entry, by providing helpful prompts to indicate the acceptable answers and format of answers to questions. The application will not need to handle much of the asking of questions by the user – both due to the inherent data entry format of the chatbot and the non-complex structure of the conversation without the use of any NLP. To facilitate this, a few simple "guidelines" were put into place to assist with the formatting of the conversation:

1) Ask questions, and where viable offer buttons or a summary of the expected format of the answer
2) Use simple, easy-to-understand language to avoid misunderstandings
3) Have helpful "error-message" style responses to assist with mistakes
4) Use a conversational, casual style message tone to reflect the platform and make the users feel comfortable using the system

While writing questions and building the conversation, these guidelines will be used to assure the system fits the purpose.

4. Implementation

4.1 Botman feasibility testing

The process of developing PlayBot began with a feasibility test of the BotMan library that was selected to use. This open-source library, built and maintained by German developer Marcel Pociot is still very new, and with no live demo available it was important to install and test the procedure for creating a demonstration bot before beginning work on the prototype. In doing so, the trickiest part was finding out how to configure the Facebook Bot to listen to messages. Although just following the procedures on the website was all that was required, there were just a lot of hoops to jump through.

4.11 Homestead and Ngrok

Configuring a local development environment to work with the external service that is Facebook Messenger Development Tools is more complicated than a normal local development environment, due to a couple of requirements of the Facebook system, namely the requirement that the bot server has a webhook that can be called directly by the system, and that the call can be received by a HTTPS request, not just HTTP. The solution to this problem was to supplement Laravel Homestead with Ngrok.

Laravel Homestead [41]is a virtual-machine based local development environment provided by Taylor Otwell, the developer of Laravel. Homestead allows the developer to deploy a Vagrant virtual machine that has all of the base requirements of Laravel, as well as providing an environment that mirrors a production level system – as opposed to local LAMP or WAMP stacks which might potentially mask problems that are revealed on deploying. The real reason for using Homestead was the easy integration with Ngrok, however. Ngrok is a tunnelling system which allows Localhost environments to be opened to HTTP or HTTPS. [42] Utilising this can sometimes be tricky, however Homestead's included "share" functionality opens up a Ngrok instance automatically. Figure 9 shows the simple command needed to launch a ngrok tunnel from your pretty-url compatible localhost server.

```
share homestead.app -region=eu -subdomain=laravel
```

Figure 9: Homestead has included Ngrok tunnelling

4.12 Botman feasibility testing cont.

Having deployed the botman app, the next thing to do was test the linking of this to Facebook. This process turned out to be somewhat smoother, with the simple app secret and webhook-verification process handled automatically by the library. The sample app however just replies to a simple greeting, so having deployed this and tested it out, it was time to develop the API to interface with the existing system as well as the prototype bot conversation.

4.2 API development

One of the most critical parts of the project was developing the API to access the data from last term's project. This data is crucial to the success of the project because being able to send and receive it on demand is one of the core requirements of a data-entry chatbot. Building the API consisted of adding to the existing Laravel project – luckily Laravel is well equipped for API development. The framework includes many features that allowed this to be completed quickly and easily, and they are detailed below.

4.3 Routing

Laravel's routing process abstracts the complex procedure of receiving and handling an incoming HTTP request, and we can leverage these features to provide a simple and easy REST interface for our API. To do this, an understanding of how the request lifecycle of Laravel works is critical. [43]

1) Request arrives in the framework, hitting the /public/index.php file as with any web server
2) The application is created and bootstrapped (environment, settings and service providers are loaded)
3) Application files are loaded, including the "routes" file
4) Request is sent to the application via the correct rule – the route
5) A response object is created and returned back to the client

Using this model, it is possible to define "middleware" for the application. This can be seen as an extra layer that fits between steps 4 and 5, and applies additional rules and configuration to the request. This can be used to check, approve, reject or add to the request.

As an API request is just another incoming HTTP request, this procedure is the exact same – therefore all we need to do to create successfully a delivery system for a full API is to add 4 things:

4.31 HTTP access control

In order to offer secure access between multiple domains, CORS or Cross-Origin Resource Sharing must be implemented to add http headers to requests that describe "the set of origins that are permitted to read that information using a web browser" [44]. In order to do this, we create a new middleware layer to apply this transformation to requests for the specific routes of our API only. The code for this can be seen in Figure 10.

4.32 Rate Limiting

Rate limiting on APIs prevents individual requesters from "spamming" – that is creating a huge number of requests to hit API endpoints to slow down or crash the server. In order to prevent this, Laravel has a built in "Throttle" middleware which will ratelimit requests using headers, throwing a "429 Too Many Requests" error should the rate-limit (default 60/min) be hit, and this will suppress requests for another minute, before allowing more to be made.

```
class Cors
{

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        return $next($request)->header('Access-Control-Allow-Origin', '*')
            ->header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
    }
}
```

Figure 10: CORS middleware

4.33 Authentication

Perhaps more important for public APIs is the concept of token authentication – to check the correct users only are requesting access for resources, for example. This is also included in Laravel via the Oauth2-based package Passport, however for the prototype system this was not implemented due to the inherent security of the "private" setting for applications on Facebook, assuring only pre-specified accounts could use the system anyway. As this chatbot will never be made "live", authenticating the API is not required in the scope of this project.

4.34 Return typing

Finally, the return typing of an API is important. Luckily, again, Laravel has a built-in solution, with automatic return conversion of objects to JSON. This easy to-implement format was perfect for being requested as on the other end of the system it is also easy to parse using built-in PHP methods. An example of this, and the result of calling it, can be seen in Figure 11.

```
public function getCurrencies()
{
    $currencies = Currency::all();
    $ret = array();
    foreach($currencies as $currency)
    {
        $ret[$currency->name] = $currency->modifier;
    }
    return $ret; //will be converted to JSON
}
```

GET ∨   http://playdale.me/api/currencies/get

{"EUR":1.2,"USD":1.3,"GBP":1}

Figure 11: An example API method and showing the return in Json

4.4 Building

Having discussed the routing and returns of the API, the next step was to build the methods required and decided during the planning stage. Using Laravel Eloquent models as an intermediary between the database and the return makes this easier as you can see in figure 11, where all "currencies" in the database are fetched within one line of code.

The full API documentation, methods and endpoints are available in the working documents.

4.5 Deploying

In order to test and use the API it was deployed to the live "staging" test server that has been hosted as part of the Playdale project. This is a Digital Ocean instance running Nginx, and replicating exactly the local Homestead development environment.

4.6 Chatbot Development

The next stage was to develop the chatbot itself, using the Botman library. The procedure for this was configuring a new Laravel instance on Homestead to contain the bot, and then replicating the installation steps including configuring a new Facebook page, account and bot called "PlayBot". Once this was configured, the next step was developing the actual bot itself.

Botman provides a "Conversation" class, which was the basis of the chatbot itself. Being that the chatbot has a single purpose, data entry following a specific format, the single conversation, triggered by saying "hi" to the bot, was the simplest way to code it – however Botman has facility to have multiple conversations as well as multiple responses to single messages.

Developing the conversation was somewhat technical and difficult to debug because there is no included debugging with the library, as well as no "view" to see debug messages – if something goes wrong and you are using the Facebook Messenger chat to interface with the bot you receive no error message in the chat! Therefore, including the Laravel Monolog implementation was critical to add debug messages as well as see exceptions in the log. There were some major problems with the documentation, being such a new platform, as well as a couple of issues with the library itself – however this will be discussed in the evaluation section of the report.

The development of a "conversation" follows a few steps. Firstly, you must create the interface which triggers the conversation to start. This takes the form of a command which links a specific word or phrase to start the conversation. In this case, the command was made to be multiple forms of the word "hello", using a simple '(?i) |' regex. This then acts as a route and directs the requests forward to a specified place, in our case a specific controller function.

Linked to this controller is the "conversation" – an abstract class defined by Botman, which is then extended by the developer. Within this conversation is a serialised sequence of function calls which must be followed in order. This function call sequence can be seen visualised in Figure 12.
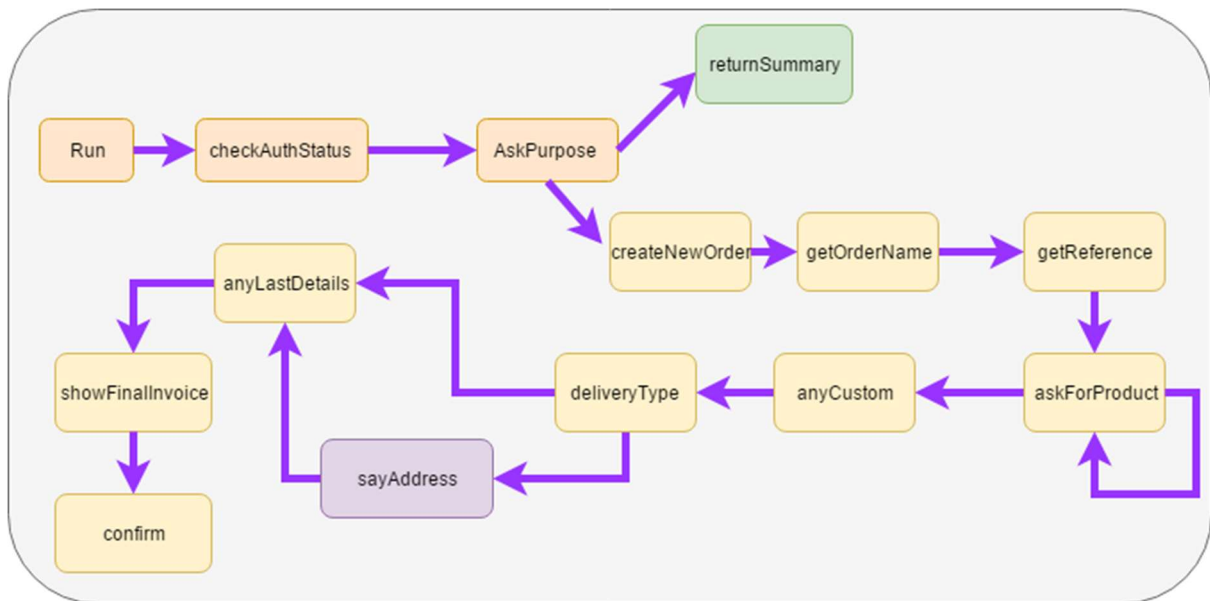
Figure 12: Flow of functions in the conversation

```php
public function getOrderName()
{
    $this->ask('What would you like the order to be called?', function (Answer $response) {
        $this->order->project_name = $response->getText();
        $this->say('Gotcha! What\'s next...');
        $this->getReference();
    });
}
```

Figure 13: Example of a simple "ask" function

4.7 Resolution of SSL issue

One issue which was encountered during development was the interfacing between the chatbot and the Playdale.me API/image repository created last term. The problem was an error thrown by Facebook, preventing messages being replied to. Checking in the developer console showed an error which stated the problem was an insecure item being loaded. After some research, the solution was found in the Send API Reference [26], where it is discovered that all external resources, as well as webhooks must be loaded over SSL. The solution to making this work with my API was to move my playdale.me demo site to HTTPS. Doing so was much easier and cheaper than in the past, as LetsEncrypt is now available, providing "free, automated and open" SSL certificates.[45] As a Certificate Authority, Lets Encrypt make the process of proving ownership and security of a domain much simpler, and this allowed the project to proceed (whereas prior to 2016, this would have cost over £70 to install and might have moved the project out-of-scope).

5. The system in operation

The finished system is, as stated, a Facebook messenger "bot", accessible by starting a conversation with the page on Facebook Messenger. Available on any platform, the system looks and feels like a normal conversation with the any Facebook user, with the added features such as buttons and templates. Below are some example screenshots of the system – and in addition there is a video available in the working documents of this paper.
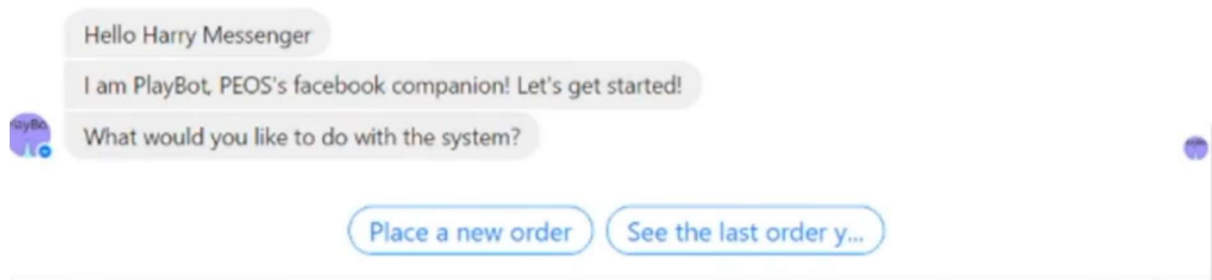

Figure 14: Call to action and buttons

Figure 14 shows the first question of a conversation, including two buttons that can be selected to proceed with the conversation. The messages sending on different lines is aimed to feel more like a conversation with a human, as well as the casual style and addressing by name. Pressing the buttons triggers the response that follows the conversation flow shown in Figure 12 in the previous section.
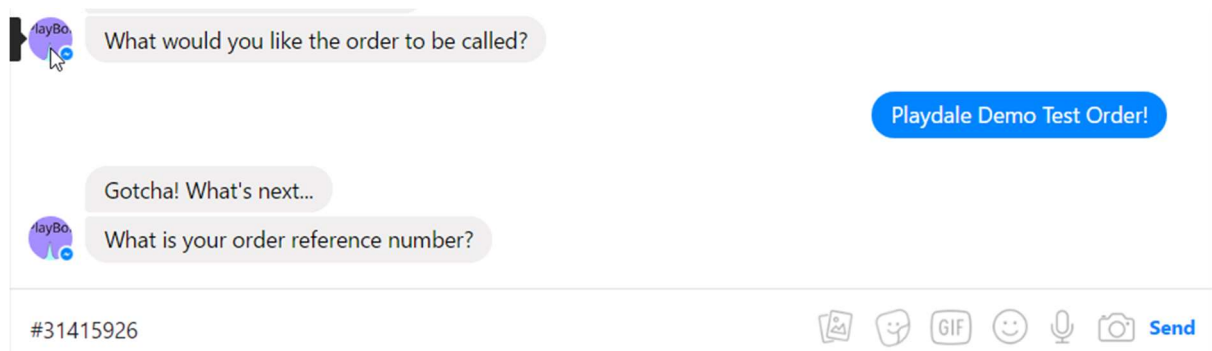

Figure 15: An example of a question-answer

Figure 15 shows an example of two simple text-entry question and answer. As you can see, for these questions where the answer will be simply converted to a string and stored, there is no need for added complexity. There are error messages automatically included and if any validation fails, or no results are found by resulting API calls a message will be generated to inform the user of what went wrong and how to fix it. An example is shown in figure 16.
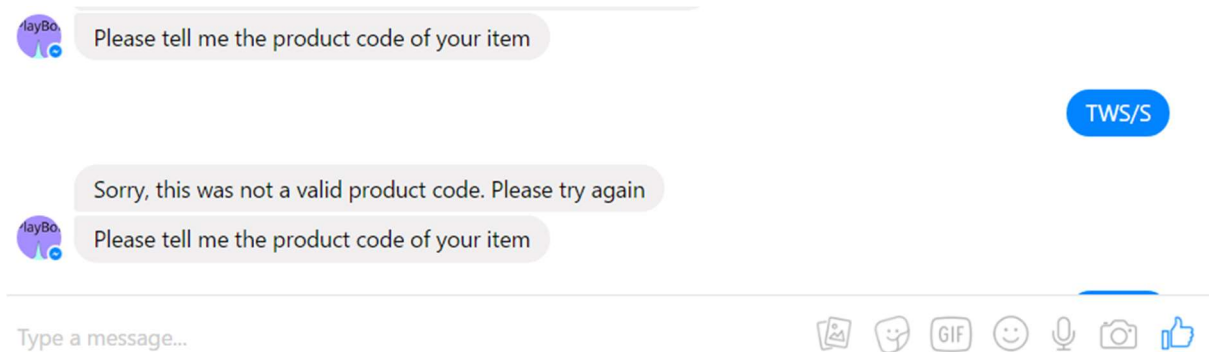
Figure 16: An error message

An example of the implemented use of a Generic Template can be seen in figure 17. One API call is made to get the product details for the product code entered (see figure 16) including the picture – one thing that changed since the planning stage. This is superior as data is duplicated less while still only making a single API call. The buttons are then created and linked to the product, while the image is loaded from the remote server like any URL image would be and then displayed. Creating this in code required some clever string manipulation including altering 'http' to 'https', due to Facebook's requirement of everything within the app operating over the SSL. This will be discussed more in section 7.

The questions continue in the same vein, with some optional questions and more buttons the same as initially, before reaching the end, and the request to order. The final "new" thing to show in this section is the "Receipt" template, adapted and extended for use in this setting to be used as "Show last order" functionality. This can be seen in figures 18 and 19. Because of required fields, there are some elements that don't make 100% sense, however due to the strict nature of the FMDT "send" API there is nothing that can be done about this.
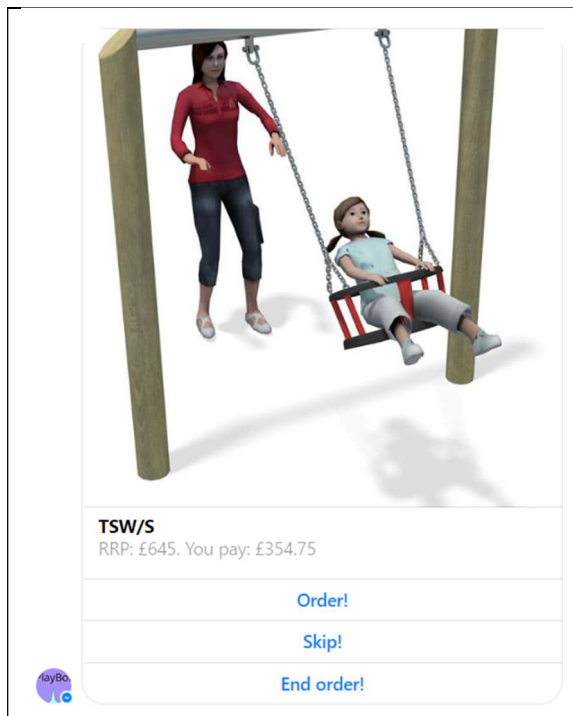
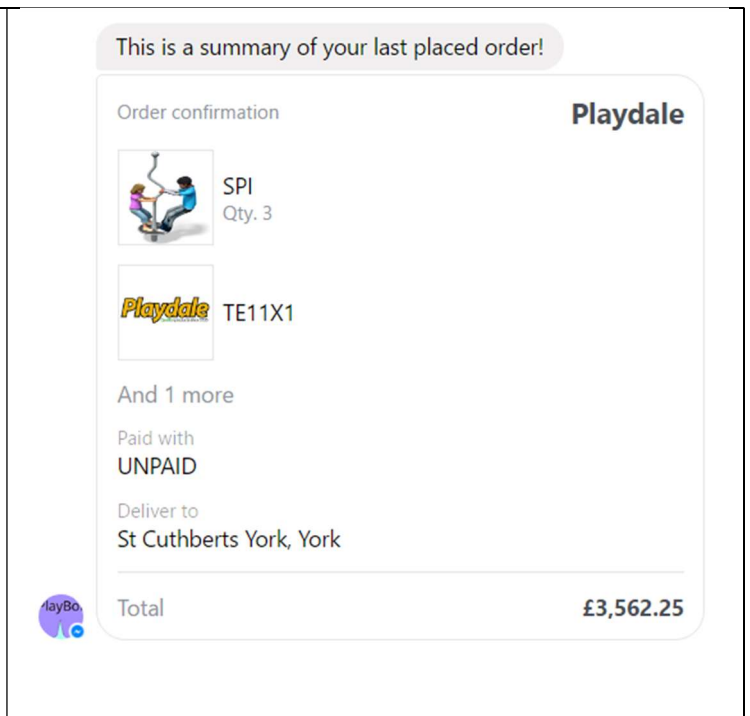Figure 17: A facebook Generic Template implemented

Figure 18: "in message" view of last order summary

Figure 19: "Full" view of last order summary, viewable by clicking on the message which causes a modal to open.

6. Testing and Evaluation

6.1 Testing Strategy

One of the most critical parts of any implementation is the testing and evaluation of the product. In order to ensure the functionality, effectiveness and usability of the system it is important to conduct both simulated system tests and some user trials with both qualitative and quantitative questions and data.

6.11 System Integration Testing Plan

In order to create a testing strategy, it was crucial to outline a simple framework that would allow testing of both the Chatbot and API, as well as user trials and to gather user feedback. This framework followed a traditional "system testing" format. In an ideal world, Test Driven development would have also been used.

Test driven development is the practice of automating unit tests to drive the design of software. Considered by many as best practice in web and API development these days, this can be achieved using packages such as PHPunit [46]. There are extensive tutorials and complex code to write in order to use these testing facilities, however, and in such a short timescale project there simply wasn't time to learn to use the testing suite as well as implement the tests, so unfortunately the unit tests were never completed, despite best intentions. This will be reflected on further in the conclusion of this paper, however despite this, the testing strategy is relatively thorough and should be sufficient.

The testing strategy defined for use in this project is therefore more of an "integration testing" strategy, adapted from a framework published on the website StrongQA [47].

The questions written to assess the application, based on this framework, are listed below:

1) **Launch and Link Verification**

   A) The application is successfully linked through Facebook Messenger
   B) The application launches successfully and displays welcome UI

2) **End-to-end testing**

   C) Messages are sent and retrieved to the API without error messages
   D) It is possible to parse a whole conversation from beginning to end
   E) All paths defined in the design document can be traversed

3) **Boundary Value Testing and Cause-Effect testing**

   F) Manual entry of edge cases does not break the application
   G) Incorrect answers to questions produce an error message
   H) Failed API requests produce an error message

4) **Security testing**

   I) Application is not accessible outside of developer's primary Facebook account
   J) User details are not available unless correct User ID is supplied in request
   K) Playdale API enforces rate limiting above 60 req/s

5) Performance testing

    L)   Total conversation length with instant replies is no more than 2 minutes in length
    M) Bot Response average time is under 10 seconds
    N) API requests made manually average under 200ms response

6) User interface testing

    O) All questions are viewable and understandable by English speaking user
    P) All buttons are functional and produce "message" feedback
    Q) All templates render correctly and are visually appealing

From these 17 questions, we should be able to "pass" the application for use in user trials, and be comfortable that the prototype is effective and functional.

6.12 User Testing Plan

Following completion of these tests, the next step is to conduct trials with both expert and non-expert users. In order to establish some useful "feasibility" understanding and help to answer our research question for the study and prototype, these user trials will compare this system with both the paper-based system currently in use at Playdale as well as the "Web Form" system created last term, and time the users in doing these 3 tasks with the same "order". In doing this, we will not only get qualitative timings and data to see if in terms of time spent, this method is equivalent to the web form and paper methods or not, but also ask some qualitative questions about the user's background, technical proficiency as well as their thoughts on the system and the future for Conversational Commerce and OLTP using such a system as well.

Using a scenario, users will be asked to complete the same data entry task on all three systems (in a randomised order to negate a form of bias) and will be timed to determine baseline efficiency – which is the most important factor in data entry. Then, a series of questions designed to gain background information on the users, feedback about the system as well as more general feedback about the user's opinion on the future of chatbots in both Conversational Commerce and Data Entry/OLTP will be asked.

6.121 User Testing Questionnaire

The first step in deriving a questionnaire was to decide on the format of questions to be used. Deciding to use Google Forms for the entry of questionnaire data (the irony not being lost on the researcher), the types of questions range from simple one-line answers to multiple choices as well as Likert Scale questions. R. Likert, in 1932 developed "A technique for the measurement of attitudes"[30] and this framework is commonly used in online surveys to assign qualitative values to quantitative answers. An example from this questionnaire is shown in figure 20. This example question will allow more precise judgement on the level of thought of my users, as opposed to a less precise "yes/no" answer. This should in turn increase the level of usefulness of my results and therefore conclusions.

I would like to use FBM bots for OLTP orders in the future

	1	2	3	4	5	
never	○	○	○	○	○	frequently

Figure 20: Example Likert Scale question from the questionnaire

The full questionnaire given to participants can be seen in the working documents of this project, however a short summary of the desired information gain is given here. Aside from the 3 questions asking timings of the three trials in the study, there are 15 short questions, with the study and survey intended to take a combined 20 minutes to complete.

The goals of the questions are to establish the user's demographics as well as thoughts on:

- Age, Education Level and Language
- Baseline existing use of Facebook Messenger
- Previous use of "Facebook Bots"
- Preference between 3 systems
- Ease of learning, pros and cons of FBM system
- Further required functionality in FBM system
- Future of FMB bots in OLTP and E-Commerce

6.122 User Testing Scenario

To test the three systems fairly, the same "scenario" was used for all trials. This was a simple task involving checking a past order and then placing a new order using each system. The full scenario is shown in Figure 21

Scenario:

Using All 3 systems...

View your previous order, and note down the product code of the item you ordered last
Place a new order for:
The item you previously ordered
3 x spinner (Product code SPI)
1 x Seat (Long) Dark Blue (C.G.L) (Product code TE11X1)
Project name "Lancaster Primary School Playground"
Internal order code "#LPSP17"
Delivery to:
St Cuthberts, Main Street, Helperby, York, YO612NS, ENGLAND
Date: August 16th
Extra info: Please include a catalogue in the order

Figure 21: Order Scenario for the trials

The systems were all configured to have the "last order" be easy to find and view, as well as having a researcher on-hand to answer any questions – however the intention of the study

was to have the user explore the systems if they were non-experts and if they were experts to have the user explore the new system.

6.2 System Integration Testing

Before showing the system to users, the system and integration testing needs to be completed. Below are the results of all the testing, with in-depth discussions on problematic tests as well as API testing.

Test	Pass?	Notes/Methodology
1) Launch and Link Verification		
A) The application is successfully linked through Facebook Messenger		Tested on Mobile & Desktop
B) The application launches successfully and displays welcome UI		Looks good on mobile/desktop
2) End-to-end testing		
C) Messages are sent and retrieved to the API without error messages		Tested via Postman & App 18/05
D) It is possible to parse a whole conversation from beginning to end		Tested via app 16/05
E) All paths defined in the design document can be traversed		
3) Boundary Value Testing and Cause-Effect testing		
F) Manual entry of edge cases does not break the application		Sometimes have to send messages twice?
G) Incorrect answers to questions produce an error message		Occasional "zero response" scenario on yes/no answers
H) Failed API requests produce an error message		
4) Security testing		
I) Application is not accessible outside of developer's primary Facebook account		Confirmed by trying another account
J) User details are not available unless correct User ID is supplied in request		ID not secured in any way but yes
K) Playdale API enforces rate limiting above 60 req/s		Yes
5) Performance testing		
L) Total conversation length with instant replies is no more than 2 minutes in length		Facebook Webhook Issue
M) Bot Response average time is under 10 seconds		Facebook Webhook Issue
N) API requests made manually average under 200ms response		Average 112.8ms (164ms inc. non-used methods)
6) User interface testing		
O) All questions are viewable and understandable by English speaking user		Y
P) All buttons are functional and produce "message" feedback		Y
Q) All templates render correctly and are visually appealing		Size of image on generic templates fills screen on mobile

Figure 22: System Integration Testing Results

As can be seen from the testing results, there were a majority of passes, as well as some major and minor issues. Section 6.21 will now discuss a positive (API testing) a minor issue (zero-response) and a major issue (Webhook issue) discovered during the testing procedure.

6.21 API Testing

The API testing methodology for this project was first to test every endpoint with a variety of edge cases, with incorrect authentication users and systematically trial every endpoint to check the return was as expected. This was done using the Postman chrome extension by making manual requests.

 In addition, each test was run 5 times (with caching disabled), with the time in milliseconds (ms) recorded and averaged to check the access times of the resources were within acceptable levels. The results are as follows in Figure 23 and 24.

Route	Avg
products/get	473.25
currencies/get	114.6
auth/{uid}	118.4
auth/{uid}/details	118.6
auth/{uid}/address	109
last/order/{uid}	112.8
last/order/{uid}/{oid}	106.8

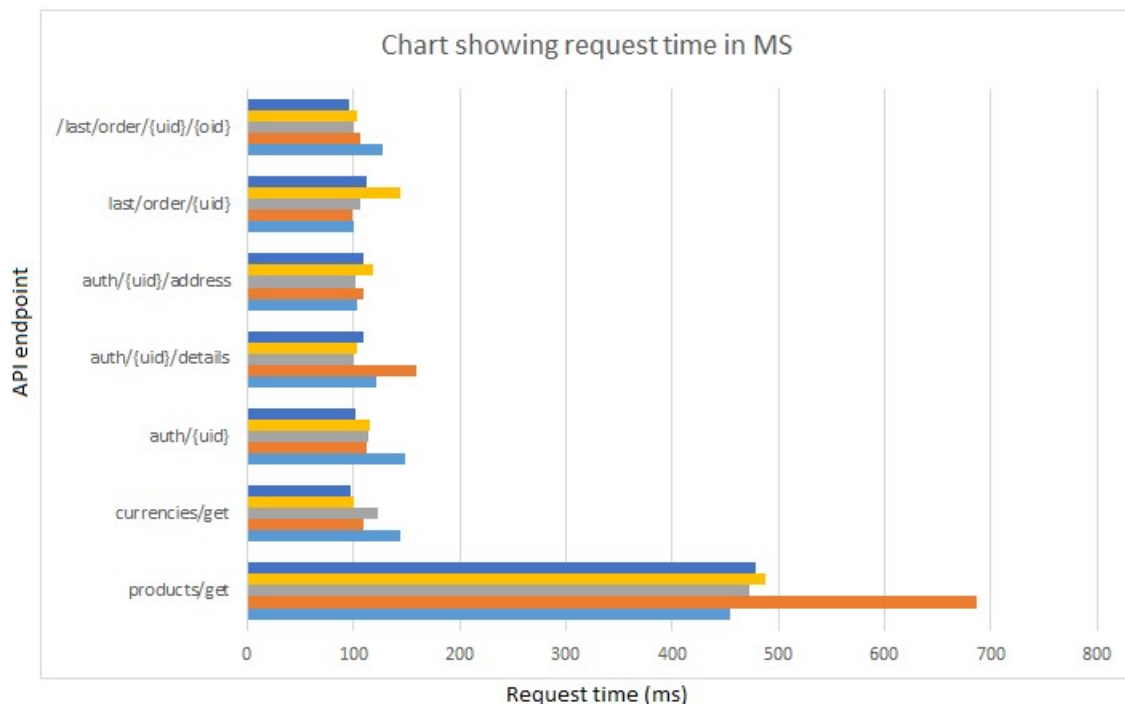Figure 23: Average times for API speed testing

Figure 24: Request times in ms

As can be seen from the figures 23&24, the API performs very well, with the sole exception of the /products/get method which returns the full product catalogue, a JSON request with over 20,000 records. This is not used in the chatbot, but on the web application – but is included for fullness. Even with this exception, the average request was well below the 200ms threshold set in the testing methodology, and the tests could be passed.

6.22 Minor Issue – Zero Response

One small problem discovered during testing was that in circumstances when asked to enter a specific result (e.g a question where the user is asked to respond "YES or NO") and the user makes a typo or an incorrect response is entered (e.g maybe) the system will hang, while waiting for a response that fits it's Regex of acceptable answers. This was easy to resolve, by simply following the message with a correct answer – and was seen as quite intuitive rather than waiting for a response – however it was less optimal than receiving a less direct error message. A proposed solution to this problem is to have better overall error handling – however this is complex in the current implementation of Botman with things being as linear as they are, having to have multiple return points with multiple if() else()

statements for every error. This will be improved upon on release of Botman Version 2.0 so a potential future version of the app could be migrated to use that after release expected to be in August. Due to this, for the user tests a temporary solution was implemented which added more words to the vocabulary listened for including synonyms like "yeah" and "alright".

6.23 Major Issue – Facebook Webhooks

Unfortunately, during testing, a major issue was encountered. The bot's response times are extremely erratic, ranging from the expected 200ms range to over 10 minutes. This provides an extremely inconsistent experience when using the chatbot which is frustrating and annoying to use. Graphs showing the time waited per reply in a test run (see video in working documents) are shown in figure 25 and 26.
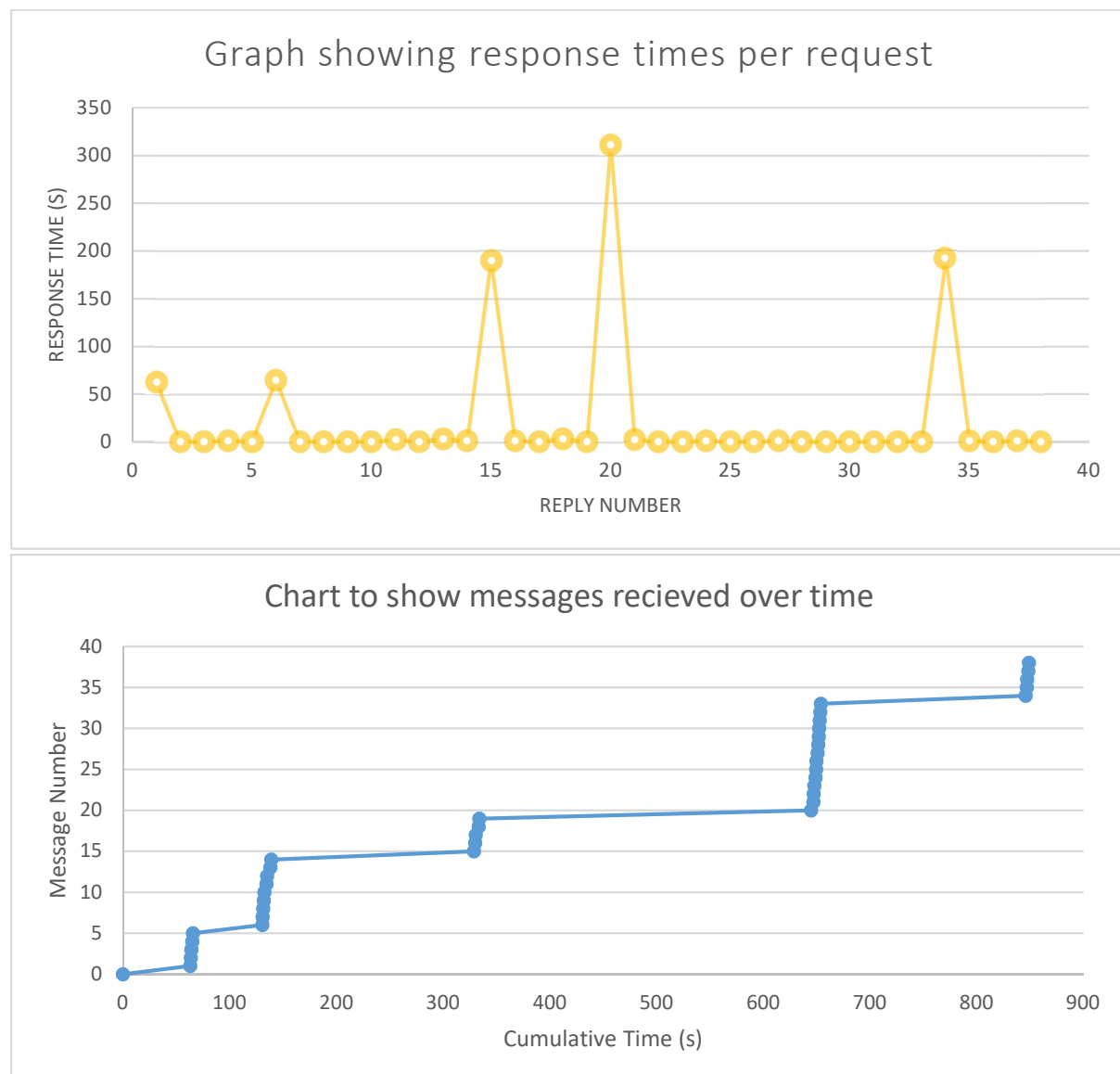
Figures 25 and 26: Graphs showing messages received over the course of a complete conversation

As you can see from these graphs, the performance on the majority of the requests is well within normal limits and is functioning correctly. However at seemingly random intervals the performance tanks for long periods, making the conversation take exponentially longer. Investigation into the cause of this was fruitless – the server doesn't receive the webhook from Facebook for this seemingly random time. Watching the Ngrok interface was futile – no requests arrived and then many would arrive at once when Facebook got "around" to sending them. Multiple users reported similar issues on Facebook's developer platform [48][49] around the time of development and testing of the application which leads to the conclusion that the issues were on Facebook's end as opposed to being Botman or Playbot specific, or caused by developer error. Another possibility is that the issue was caused by the use of Ngrok tunnelling as opposed to a live web server, however this shouldn't have been the case. Also, rate limiting could have been in effect due to server performance or degradation, however the issue was still occurring as of last testing, so this also shouldn't have been the case. Playbot, being a free app with a non-live page and "private" mode enabled would be considered low priority in terms of resources, potentially leading to these issues as well.

Resolving this issue would potentially require opening a support case similar to the links above with Facebook and waiting for an investigation, or trialling deploying the bot on a live server as opposed to Ngrok – options unfortunately were outside the scope and timescale of this project. Future work could consider a more developed or robust platform than Facebook Messenger, however!

6.3 User Trials

Despite the issues, we went ahead with running user trials. Four users, two considered experts for the reason of being on Playdale's staff – and two considered non-experts were asked to take part in the study as described in section 6.12. The user demographics were 3 females and 1 male, with ages ranging between 21 and 35. All users had previous experience of Facebook Messenger, with 3 selecting "Frequently" and 1 selecting "rarely" having used the platform, however none of the users had previous experience of using chatbots or Facebook Messenger Bots.

The users were asked to complete the same task given in the scenario (6.122) in a randomised order, and the time taken to do so was recorded. These results are shown on the graph below (See figure 27).

Unfortunately, the time waiting for Playbot exceeded an exceptional waiting time for the users of trial 3 and 4, and they abandoned the system mid trial. This, again, was due to the Facebook Webhook problem.

From the results, it's clear that the prototype Data Entry Chatbot with this issue cannot compete with the others, however if you extrapolate the results and remove "outlier" waiting times from example conversation shown in 6.23 – demonstrated by the length of the edited video included in the working documents - and compare this to the average times of the trials for the other systems, the chatbot, although still slower than the web form, compares much more favourably (See figure 28).
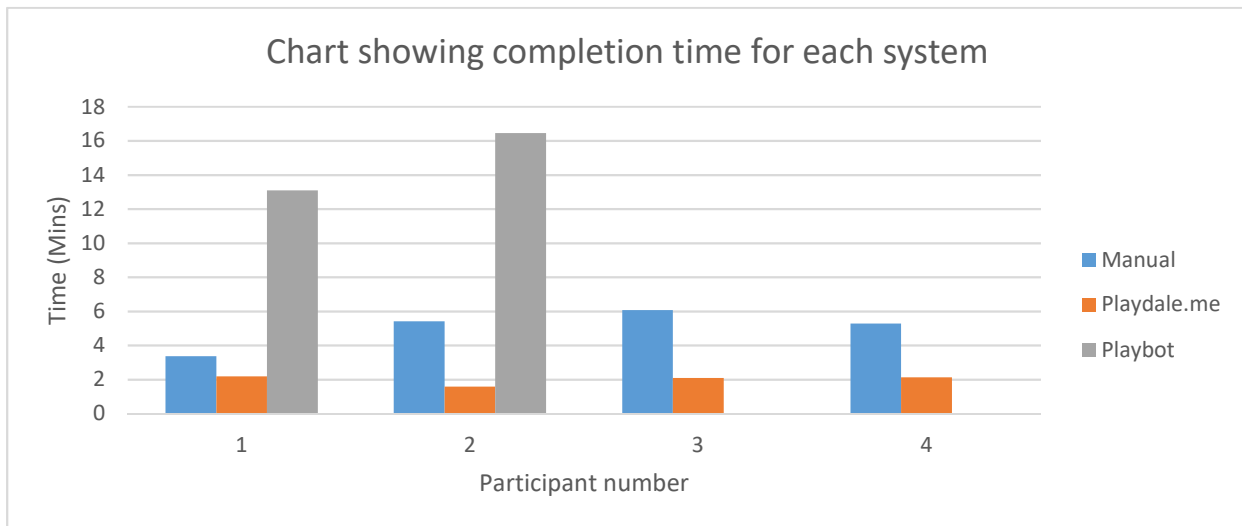
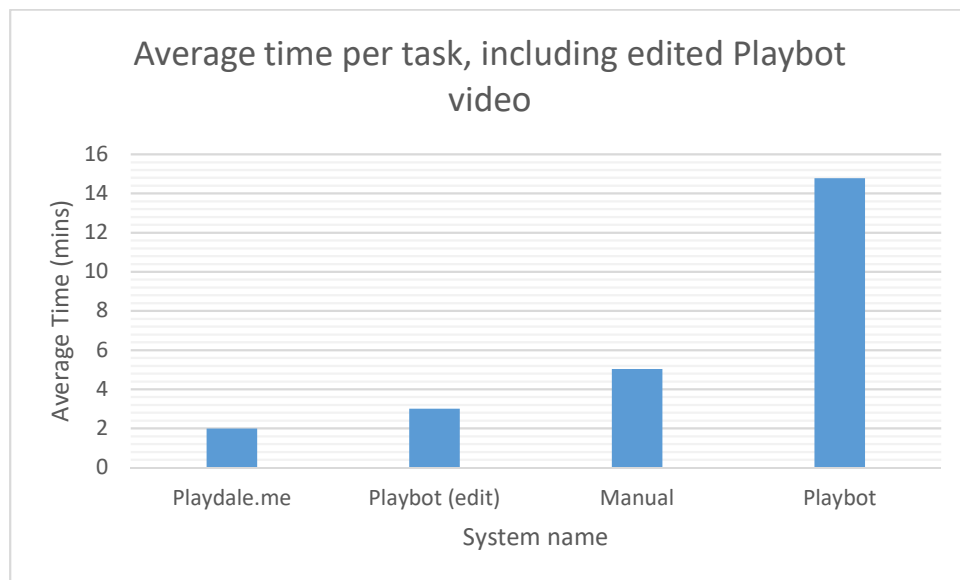Figure 27: Graph showing times taken on each system in trials

Figure 28: A graph showing the average system completion time for each, as well as the time Playbot could have taken without the webhook delays

Although this is non-scientific, it is a better representation of how a non-buggy Playbot would have performed in the trial – this however is not assumed in the rest of the report.

The qualitative feedback on Playbot itself was also punished by the exceptionally long waiting times for the bot to respond. When asked "Which system felt the most intuitive" and "Which system would you prefer to use", 100% of the participants felt the Web Form (playdale.me) was the best technique. When asked "What would you say the worst features of the Playbot system are?", again, 100% of the responses mentioned the system being too slow and "not working". This was unfortunate, as the users actually liked a lot of the features of the bot. Good features mentioned included the automatic information filling from the API, the conversational tone and the convenience and familiarity of the known platform, however all of these benefits were overshadowed by the major issue mentioned above, limiting the potential useful results from the study.

Feedback from the more general Likert scale questions about the user's opinions on the future of e-commerce and data entry chatbots was more positive when told to ignore the issues they had experienced with the system they had trialled. Of the three users with "frequent" experience of the Facebook ecosystem, the question "I would like to use Facebook Messenger chatbots for OLTP data entry in the future" scored an average of 2.66/5, showing some cautious acceptance of the idea being feasible. Including the less invested member dropped this result to 2.25 however, equating to "Disagrees" on the likert scale. However, with such a small sample size this is not a conclusive result by any means. When asked if the user agreed that "Facebook Messenger bots are the future of data entry" the result was an average of 1.8/5, offering a resounding "no" to that question, unfortunately.

The concept of Conversational Commerce was slightly better received however, with users averaging a 3.25/5 and 3 respectively with the same questions as before. This shows that the small sample of users have not dismissed entirely the potential future use of chatbots despite all experiencing a bad first impression of one.

In summary, a disappointing setback gave an overall negative summary of chatbots for data entry – however potential for e-commerce and future more refined data-entry chatbots should not be dismissed as users invested in the platform seemed somewhat optimistic about the potential future implications of such a technology.

7. Conclusions

Overall, the project has achieved the aims it set out to do. Despite some disappointing and demotivating user trials, this paper advances the topic of Conversational Commerce as well applying a new concept – data entry – to the field.

7.1 Objective analysis

At the start of the report, a set of objectives were defined to study and reflect on throughout the project, and these have been achieved throughout the study. These were:

7.1.1 Background of Chatbots and Conversational Commerce

The background was reviewed in depth, discussing at first the history and starting points of Chatbots, from ELIZA through to the modern day. Telling a historical story via the medium of progressing technology, the report shows the way chatbot technology developed, boomed, fell off again and then was revived by commercial applications in the 21st century.

The new area, Conversational Commerce, was defined and discussed and research was summarised in this burgeoning field, and a discourse about the potential future of the area was also opened – as well as giving a case study of WeChat, the most successful CC platform up until this point.

The current state of OLTP and data entry was also discussed, with justifications for the overall project and lamentations for the lack of innovation in the area of data entry were discussed.

In reflection, this research provides the most complete summary and literature review of this new research area available anywhere up until this point, and in the opinion of the author this is exciting, as it gives a new baseline for future researchers to build upon when conducting further research into what is sure to be a growing area and industry.

7.1.2 Existing bot overview

Existing chatbots from two platforms were reviewed in depth from a user's perspective, however if given more time it would have been good to discuss existing chatbots from a technical perspective as well as considering more than two platforms. Available on Github is source code for some completed projects for example 'lins05/slackbot' [50] and these could have been reviewed for a deeper reflection on existing technology, however the bot reviews provided good insights into the available area and technologies of Facebook and Slack and allowed an educated choice be made.

Perhaps a further study into the potential drawbacks of both platforms and a thorough review of the "issues" sections of both platform's code might have led to identification of the issues in the Facebook webhook system that caused all the problems which were experienced, however this seems unlikely.

7.1.3 Case study

The main section of the study, the case study was completed and overall successful – with the objective of building and assessing the effectiveness met completely. However the success of the actual prototype was less than optimal, and perhaps with better time-

management or a longer development period than 2 weeks, the issues could have been resolved, leading to better results. Nonetheless, the objectives were completed!

In terms of the analysis, more users would certainly have been useful however due to the issues the bot was experiencing, and the 20-plus minute attempts that some users experienced, the decision was taken to cap the number of users at 4. This limited the potential of the results, however as a proof-of-concept prototype, the limited user base was not a huge deal. If the project were to be continued, or more research would be done, this could be expanded to a cohort of 40 users, as well as trialled with actual customers of Playdale, to get better and more focussed feedback.

7.2 The project as a whole/Revisions

Taking the project as a whole, the amount achieved in 6 weeks is exceptionally pleasing. With most design-and-build projects lasting between two and three times this length, the pleasing completion of the paper and prototype are an achievement in themselves.

However, there are plenty of technical improvements that could be made if the project was to be repeated. Specifically, the platform used could have been adapted – BotMan provides multi-platform drivers and an adaptation to use Slack, WeMo, Wechat or Telegram could have avoided the issues we ran into with the Facebook Webhooks. Additionally, the lack of unit testing is again an issue which could have been resolved in a longer timescale and would have greatly assisted in the peace of mind.

These two factors would also have lead to a more robust system, allowing more user testing to be completed and a greater results set gathered. This In turn would have allowed further statistical analysis and data processing, something that this paper is missing and that would have complimented the rest of the project well.

7.3 Future Work

Being a brand-new research area, Conversational Commerce has a slew of further work to be investigated. The actual commerce area would be interesting to look into as the users in the study indicated this would be their preferred application of the technology, and this would also be an area where profit could be made by a corporate sponsor of the research.

Another "case study" style investigation into the design, build and launch of a more traditional Conversational Commerce chatbot would be very interesting, especially to see if a profit could be made and further legitimise the research area. A second area for future work would be alternative methods for data entry, outside of web forms and chatbots. Further innovations in NLP for voice could make for interesting Phone-style chatbot conversations for data entry, potentially allowing hands-free data entry for those with disabilities. Both of these future continuations could expand on the existing work done during this project.

7.4 Lessons Learned

As part of the project, the researcher has learned much about both building a chatbot and the area of research of Conversational Commerce. Particular skills gained were in writing a literature review, time management to fit such a large project into a 5-week span, as well as

improving research skills travelling to Playdale to conduct the user trials. In addition, the skills gained in development, especially in the process of configuring APIs, HTTPS and webhooks were extremely valuable and will help out in future goals, research and employment.

7.5 Recap

Overall, a successful project was attained. A well thought out summary of the previous work and the area lead to a successful definition of a project and a prototype was designed, built, tested and analysed within a 4 week window. Although the results of the chatbot were limited due to some third-party faults outside the control of the researcher, many lessons were learned and much progress was made.

References

[1]   Chris Messina, "2016 will be the year of conversational commerce – Chris Messina – Medium," *MEDIUM*, 2016. [Online]. Available: https://medium.com/chris-messina/2016-will-be-the-year-of-conversational-commerce-1586e85e3991. [Accessed: 31-May-2017].

[2]   BI Intelligence, "The Messaging App Report - Business Insider," *Business Insider*, 2016.

[3]   J. Whittacre, "Internet of Things Seminar." Lancaster University, 2015.

[4]   WhatsApp, "WhatsApp FAQ - I'm an iPhone developer, how can I integrate WhatsApp into my app?" [Online]. Available: https://faq.whatsapp.com/en/iphone/23559013. [Accessed: 07-Jun-2017].

[5]   Twitter, "typeahead.js." [Online]. Available: https://twitter.github.io/typeahead.js/. [Accessed: 07-Jun-2017].

[6]   J. Weizenbaum, "ELIZA---a computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966.

[7]   J. Weizenbaum and Joseph, *Computer power and human reason : from judgment to calculation*. W.H. Freeman, 1976.

[8]   K. M. Colby, S. Weber, and F. D. Hilf, "Artificial Paranoia," *Artif. Intell.*, vol. 2, no. 1, pp. 1–25, 1971.

[9]   R. Carpenter, "Jabberwacky - About, Thoughts - An Artificial Intelligence," 1997. [Online]. Available: www.jabberwacky.com/j2about. [Accessed: 30-May-2017].

[10]   Dr. Richard S. Wallace, "How it all started," *Science And Justice*, 2000. [Online]. Available: http://www.marestaurantassoc.org/abouthistory.htm. [Accessed: 07-Jun-2017].

[11]   A. Gabbatt, "IBM computer Watson wins Jeopardy clash," *Guardian*, 2011. [Online]. Available: https://www.theguardian.com/technology/2011/feb/17/ibm-computer-watson-wins-jeopardy. [Accessed: 07-Jun-2017].

[12]   C. Lorrie, "The Big Bang Theory: The Beta-Test Initiation," CBS, 2012.

[13]   FacebookDevelopers, "Facebook Messenger Platform," pp. 1–16, 2015.

[14]   D. Marcus, "Life Inside The Tech Bubble," in *Disrupt SF 2016*, 2016.

[15]   J. Constine, "Facebook launches Messenger platform with chatbots | TechCrunch," *Techcrunch*, 2016.

[16]   E. W. T. Ngai and A. Gunasekaran, "A review for mobile commerce research and applications," *Decis. Support Syst.*, vol. 43, no. 1, pp. 3–15, 2007.

[17]   G. H. Van Bruggen, K. D. Antia, S. D. Jap, W. J. Reinartz, and F. Pallas, "Managing Marketing Channel Multiplicity," *J. Serv. Res.*, vol. 13, no. 3, pp. 331–340, Aug. 2010.

[18] M. van Eeuwen, "Mobile conversational commerce: messenger chatbots as the next interface between businesses and consumers," 2017.

[19] T. van Manen, "Bot or not: dit is waarom Facebook inzet op chatbots | Marketingfacts," *Marketingfacts.nl*, 2016. [Online]. Available: http://www.marketingfacts.nl/berichten/chatbots-facebook-inzet-chatbots-messenger. [Accessed: 07-Jun-2017].

[20] WeChat, "The 2016 WeChat Data Report | WeChat Blog: Chatterbox," *WeChat Blog*, 2017. [Online]. Available: http://blog.wechat.com/2016/12/29/the-2016-wechat-data-report/. [Accessed: 07-Jun-2017].

[21] L. Kuo, "WeChat is nothing like WhatsApp—and that makes it even more valuable," *Quartz*, 2014. [Online]. Available: https://qz.com/179007/wechat-is-nothing-like-whatsapp-and-that-makes-it-even-more-valuable/. [Accessed: 07-Jun-2017].

[22] C. Chan, "When One App Rules Them All: The Case of WeChat and Mobile in China | Andreessen Horowitz," pp. 1–14, 2016.

[23] Tpc.org, "About the TPC." [Online]. Available: http://www.tpc.org/information/about/abouttpc.asp. [Accessed: 07-Jun-2017].

[24] W3, "Fill-out Forms and Input fields," *w3.org*, 1993. [Online]. Available: https://www.w3.org/MarkUp/HTMLPlus/htmlplus_41.html. [Accessed: 07-Jun-2017].

[25] D. P. Group, "Domino's delivers 'Easy Order' bot to Facebook Messenger." [Online]. Available: http://corporate.dominos.co.uk/news/"hello-dom"…-it's-me-from-the-other-side. [Accessed: 07-Jun-2017].

[26] FacebookDevelopers, "Generic Template - Messenger Platform." [Online]. Available: https://developers.facebook.com/docs/messenger-platform/send-api-reference/generic-template. [Accessed: 07-Jun-2017].

[27] FacebookDevelopers, "Payments (beta) - Messenger Platform." [Online]. Available: https://developers.facebook.com/docs/messenger-platform/complete-guide/payments. [Accessed: 07-Jun-2017].

[28] RIAEvangelist, "Node-dominos-pizza-api by RIAEvangelist." [Online]. Available: https://riaevangelist.github.io/node-dominos-pizza-api/. [Accessed: 07-Jun-2017].

[29] Slack, "Slack: Where did it get its name? - Business Insider." [Online]. Available: http://uk.businessinsider.com/where-did-slack-get-its-name-2016-9?_ga=1.138509299.1175501442.1475074973?r=US&IR=T. [Accessed: 01-Jun-2017].

[30] Slack, "Slackbot: personal assistant and helpful bot – Slack Help Center." [Online]. Available: https://get.slack.help/hc/en-us/articles/202026038-Slackbot-personal-assistant-and-helpful-bot-. [Accessed: 07-Jun-2017].

[31] VASPP, "VASPP Technologies » Innovate | Create | Deliver." [Online]. Available: http://vaspp.com/vbots/. [Accessed: 03-Jun-2017].

[32] Node.js, "Node.js." [Online]. Available: https://nodejs.org/en/. [Accessed: 03-Jun-2017].

[33]   Z. Bruggeman, "remixz/messenger-bot," *Github*. [Online]. Available:
        https://github.com/remixz/messenger-bot. [Accessed: 07-Jun-2017].

[34]   Flask, "Welcome | Flask (A Python Microframework)," 2014. [Online]. Available:
        http://flask.pocoo.org/. [Accessed: 07-Jun-2017].

[35]   D. Chua, "davidchua/pymessenger," *Github*. [Online]. Available:
        https://github.com/davidchua/pymessenger.

[36]   Taylor Otwell, "Laravel - The PHP Framework For Web Artisans." [Online]. Available:
        https://laravel.com/. [Accessed: 07-Jun-2017].

[37]   M. Pociot, "Installation BotMan - Create messaging bots in PHP with ease." [Online].
        Available: https://botman.io/1.5/installation. [Accessed: 03-Jun-2017].

[38]   Facebook, "Wit.ai," 2017. [Online]. Available: https://wit.ai/. [Accessed: 03-Jun-2017].

[39]   API.AI, "Conversational UX Platform for Bots, Apps, Devices, Services - Api.ai."
        [Online]. Available: https://api.ai/. [Accessed: 07-Jun-2017].

[40]   API.AI, "Introduction to API.AI." [Online]. Available: https://docs.api.ai/docs/key-
        concepts. [Accessed: 07-Jun-2017].

[41]   Taylor Otwell, *Laravel 5, The PHP Framework For Web Artisans*. 2017.

[42]   A. Shreve, "ngrok - secure introspectable tunnels to localhost," 2017. [Online].
        Available: https://ngrok.com/. [Accessed: 07-Jun-2017].

[43]   Taylor Otwell, "Laravel - Request Lifecycle," 2014. [Online]. Available:
        https://www.laravel.com/. [Accessed: 07-Jun-2017].

[44]   Mozilla, "HTTP access control (CORS) - HTTP | MDN," *MDN*, 2015. [Online]. Available:
        https://developer.mozilla.org/en/docs/Web/HTTP/Access_control_CORS. [Accessed:
        05-Jun-2017].

[45]   Linux Foundation, "Let's Encrypt - Free SSL/TLS Certificates," *letsencrypt.org*, 2017.
        [Online]. Available: https://letsencrypt.org/. [Accessed: 07-Jun-2017].

[46]   S. Bergmann, "PHPUnit," *Online*, 2017. [Online]. Available: https://phpunit.de/.
        [Accessed: 06-Jun-2017].

[47]   StrongQA, "SOFTWARE TESTING - TEST STRATEGY What is Test Strategy ?"

[48]   FacebookDevelopers, "Webhooks received with delay up to several hours - Facebook
        for Developers." [Online]. Available:
        https://developers.facebook.com/bugs/329858984097928/. [Accessed: 06-Jun-2017].

[49]   FacebookDevelopers, "Conversation webhook not received by my Messenger bot -
        Facebook for Developers." [Online]. Available:
        https://developers.facebook.com/bugs/772347522945688/. [Accessed: 06-Jun-2017].

[50]   S. Lin, "lins05/slackbot," *Github*. [Online]. Available:
        https://github.com/lins05/slackbot.

Appendix items available in working documents (https://ihazzam.github.io/PlaydalePages/)