



Implementation of a ChatBot for helping users find the nearest and cheapest gas station

Aristotelis Gorgias

SID: 3306160001

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2019

THESSALONIKI – GREECE



INTERNATIONAL
HELLENIC
UNIVERSITY

Implementation of a ChatBot for helping users find the nearest and cheapest gas station

Aristotelis Gorgias

SID: 3306160001

Supervisor:

Dr. Ioannis Magnisalis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2019

THESSALONIKI – GREECE

Abstract

The present dissertation was written as a part of the MSc in Mobile and Web Computing at the International Hellenic University by the student Gorgias Aristotelis under the supervision of Dr. Ioannis Magnisalis in cooperation with Dr. Stergios Tegos.

The terrifying rapid growth of technology has led humanity towards new implementations, expertise and best practices in all aspects of production and creation. This growth comes from the humans' continuous effort to evolve, to create and to improve the way of living and working by researching and experimenting. A critical outcome coming from all this effort, is the continuous creation of data. Humongous amounts of data are being generated everyday which has also led to the naming of our current epoch to, "The era of Big Data". The constant need of automating things and trying to make life easier in general, has played a big role in the growth of artificial intelligence in combination with automations.

Speaking of automations, one can easily refer to robots, either hardware or software, that have a specific set of functionalities to provide assistance when needed. These robots are mainly created to make things and life easier. As already mentioned, these can either be consisted by both hardware and software, where software gives meaning to the hardware, like those you see in movies, or only by software itself.

A software bot is actually a clever algorithm which gets fed by data and gives as an output some best results. The outcome of the bot can be based on multiple factors. Some of these factors may be, the user's needs, the amount of data that will be fed to the bot, technical criteria, location criteria and more, but the most import factor is the purpose of the bot and the reason that it has been created (e.g. a medical bot will provide different output to doctors in contrast to humans with no medical knowledge).

The present dissertation will be consisted of two main chapters. The first will be a brief review on the current status of the existing chatbots, a report on technologies and tools that are being used to implement them and finally the scenarios and methods with which these can be implemented. The second chapter is going to be the implementation of a Chatbot prototype which will provide the best results to users who want to find the lowest price gas stations near them.

The implementation will be divided in four parts which are, first, the implementation of a web scraper which is a process to gather information from the Ministry's of Development and Competitiveness website for gas prices for all the gas stations in Greece, second, the design and creation of a Database to store the data extracted from the scraper, third, the implementation of a RESTful API which will be available at any time to serve and respond to requests, and finally the design and implementation of the Chatbot prototype.

At this point, I would like to deeply thank my supervisor Dr. Ioannis Magnisalis in cooperation with Dr. Stergios Tegos for giving me this opportunity to work alongside them on this interesting project.

I would also like to deeply thank my family for the consistent support they provided me throughout these two years for successfully graduating and also for all of the years that I have been a student, as they have always stood by my side and provided me the assistance that I needed.

Gorgias Aristotelis

02/12/2019

Contents

Abstract	3
Contents	5
Figures	6
Chapter 1: Introduction	7
1.1 Overview	7
1.2 Problem Statement and Objectives	8
1.3 Solving the problem	9
Chapter 2: Literature Review	10
2.1 What are the chatbots	10
2.2 Chatbot Frameworks	12
Chapter 3: Implementation	18
3.1 Tools, programming languages & technologies	18
3.1.1 Tools	18
3.1.2 Programming Languages	26
3.2.3 Technologies	29
3.2 Implementation Process	33
3.2.1 Scraper	33
3.2.2 Database Schema	41
3.2.3 PHP Rest API	45
3.2.4 The Chatbot prototype	47
Chapter 4: Discussion and results	54
4.1 Goals and implementation targets	54
4.2 Challenges and problems tackled	55
4.3 Limitations of current work	55
4.4 Future research	56
Chapter 5: Conclusions	57
Bibliography	58

Figures

Figure 1 – Integration between bots and channels	12
Figure 2 – Microsoft Bot Framework Flow Diagram.....	13
Figure 3 – Dialogflow Flow Diagram	14
Figure 4 – ChatterBot Flow Diagram	16
Figure 5 – Screenshot of Sublime	19
Figure 6 – Screenshot of XAMPP	20
Figure 7 – Screenshot of Chrome Developer Tools	21
Figure 8 – Screenshot of Postman	22
Figure 9 – Screenshot of Datagrip.....	23
Figure 10 – Screenshot of Botsociety UI.....	24
Figure 11 – Screenshot of Dialogflow UI.....	25
Figure 12 – PHP Sample Code.....	26
Figure 13 – SQL Sample Code	27
Figure 14 – Javascript written inside a JS Node running in Google’s servers	28
Figure 15 – cURL initialization, setting of options, execution and closing of session.....	30
Figure 16 – Datasets in XML form and browser’s output in raw text.....	30
Figure 17 – XML structure of response shown in developer tools	31
Figure 18 – XAMPP interface and services.....	32
Figure 19 – Datasets in Object form. Structured, readable and manageable	34
Figure 20 – Task scheduler interface.....	35
Figure 21 – Edit trigger interface	36
Figure 22 – Cron job scheduler	37
Figure 23 – Iteration through objects	38
Figure 24 – Total results of elements stored to the database	39
Figure 25 – Code sample of update file.....	39
Figure 26 – Update.php file output on browser	40
Figure 27 – Visual representation of the database schema	42
Figure 28 – Screenshot of Regions (dd) table.....	42
Figure 29 – Screenshot of Companies table.....	43
Figure 30 – Screenshot of Products table	43
Figure 31 – Screenshot of Stations table	44
Figure 32 – Screenshot of Mapping table.....	44
Figure 33 – Screenshot of Postman after call to the API endpoint.....	46
Figure 34 – Basic Authorization type to perform the call.....	47
Figure 35 – User - agent interaction flow diagram exported by Botsociety tool	48
Figure 36 – Agent’s main intent named GetFuel	49
Figure 37 – Intent’s parameters location and product	50
Figure 38 – Fulfillment configuration to perform the call to our API	51
Figure 39 – Final stage Usage of the whole implementation.....	52

Chapter 1: Introduction

In this chapter, a description will be given on what a Chatbot is and its definition. Firstly, there is going to be an overview of the current status on the evolvement of bots in general, what the purpose that these serve is, what makes them useful, what further possible purposes they could serve and where this sudden rapid growth of them is coming from.

Further on, there will be a description on what the problem is that the implementation and composing of this dissertation is trying to solve, giving a detailed explanation on the aforementioned problem and gaps in the users' needs. Therefore, objectives will be defined and described, which must be fulfilled successfully by completing this dissertation.

The final outcome of the current implementation will be divided in 4 parts. The first part will be the gathering of all of the information needed, which will feed the Chatbot with accurate data to respond to user calls. This information will be consisted by the gas prices for all of the gas stations in Greece, per county. The second part will be the design and the creation of a database, in which, all of the information will be stored and afterwards retrieved to serve queries made by users using the Chatbot. The third part is going to be an implementation of a RESTful API which will serve responses to queries coming from the Chatbot. Finally, the last part will be the designing of the Chatbot and its core functionalities and finally the implementation of an actual Chatbot prototype.

1.1 Overview

It could be easily agreed that speech is maybe the strongest and most efficient way of communication among humans. This big privilege and advantage that comes with speech interaction, has led industries and researchers towards the invention of technologies and methods to apply this kind of communication between humans and computers. The main goal of this purpose is to achieve a human-to-human like interaction between humans and computers.

While the speech interaction is gaining momentum as a technique between the communication of humans and computers, there has been an urgent need in speech oriented search engines and assistants. Almost all of the major tech organizations have implemented their own speech search engines, assistants and Question Answering mechanisms such as Siri, Google Now, Skyvi, Robin, Jarvis and more.

A Chatbot could be defined as an artificial intelligence (AI) piece of software which acts as a conversational agent that interacts with users in natural language. Also, it could be described as one of the most advanced, intelligent and promising interaction agent between

humans and machines. The purpose of these agents is to simulate conversations with users for whatever topic or domain they should serve. Its core functionality is to simply answer questions.

Normally, the main task of a chatbot is to provide help and assistance to users who are trying to complete a task. This task could be anything, for example, navigating through a website, finding something on the web, understanding the features and functionalities of a modern car and anything else that one could imagine. In other words, the way a chatbot works is usually by the user initiating the whole process by asking a question or making a comment on a topic and then the chatbot answering the question or continuing the discussion on the user's comment.

Finally, from a practical point of view, we could say that a chatbot only represents a question answering system based on the Natural Language Processing (NLP) technologies and techniques.

1.2 Problem Statement and Objectives

To easier define and state the problem that we are trying to describe and solve in this paper, we have to first think of the current era that we are living in, the era of "Big Data". There is a constant and violent bombardment of information, knowledge and data every day, coming from all means and kinds of media, either from the television, or blogs, or social media, or websites and many more.

It is terrifying the fact that there are 2.5 quintillion bytes of data created every single day whilst it is estimated that by 2020, 1.7MB of data will be generated every second for every person on earth. Additionally, it is amazing that 90% of the world's data has been created over the last two years.

Having realized the true meaning of the term "Big Data", we can now begin to understand some of the biggest problems that will occur by this humongous amounts of data. To state some, we can begin by thinking, how a user can find the best results to a query or the managing and maintaining of these data. Further on, software and applications built to successfully retrieve and serve information from this huge pool, must comply with modern technologies and techniques which will make the retrieval process fast and reliable. One of these technologies is the Natural Language Processing, on which, Chatbots rely and are built on.

Having described the general problem, we can now focus individually on the problem that has to be solved in this dissertation and that is nothing else than how to find the easiest way to retrieve the best information needed on specific queries.

Before continuing, it should be considered how the everyday human lifestyle is nowadays. People tend to live in big cities where businesses and work opportunities thrive. But, a disadvantage of living in such cities is the fact that everything takes more time to happen and to be done. Big cities are characterized by traffic jams, crowded places with noise and smaller amounts of free time since everything takes more time to be done.

In such a lifestyle, it is a natural behavior by humans to tend to make their life easier even by the smallest things. They will try to find the easiest ways to travel, to buy groceries, to pay bills, to book their tickets and most importantly, to search for things they need when they are in a rush or in a busy situation.

More specifically, this Thesis will concentrate on how one can find the best match for gas stations with the lowest prices and for the desired fuel products.

1.3 Solving the problem

The continuous growth of artificial intelligence, gives opportunities and advantages to software engineers and developers to create and build fast, intelligent, reliable and practical software. The biggest competitor against applications which must be downloaded first from an app store, installed, opened and finally use the user interface (UI) to find what is needed by doing taps on the screen, is the Chatbots, where it is only needed to speak to the device to launch the Assistant Agent and dictate queries, having as a result to get the desired responses. This is actually the true power of the chatbots, their instantaneous and extreme ease of use.

To better describe a busy situation, we can consider the moment of being in a car and in a rush and at the same time, running out of gas. At this point, there is an urgent need to immediately find a gas station before running out of fuel which would make the problem even bigger. Being in such a situation, it wouldn't be safe to simply get a device on hands and start searching for nearby gas stations on the web. This is where the Chatbots kick in.

The easiest and fastest solution to find the information needed is to simply speak to the device and activate the Chatbot agent. Then, a speech dialog is initiated by the user, between the user and the software, having as a result some desired answers which the Chatbot will respond to the user's questions. Therefore, in this dissertation, a Chatbot for finding the lowest price gas stations near its users, is going to be implemented. To be more specific, the implementation will be structured by a process for extracting the data of gas stations and gas prices in whole Greece, the creation of a database to efficiently store the data organized and categorized, an API which will be responsible for the communication between the database and external authorized applications such as Chatbots and finally the design and implementation of a Chatbot prototype which will be getting fed by data from the database, through the API, aiming to return accurate information back to its users.

It should be mentioned here that these data will be extracted from the <https://www.fuelprices.gr> website, which is an asset of the Greece's Ministry of Development and Competitiveness.

Chapter 2: Literature Review

In this chapter a report will be given on the existing chatbot assistant agents that are currently being used. Also, the theoretical background on which these are based on will be described, and finally there will be a comparison of the available tools and frameworks for implementing assistant agents. It should be noted here that Chatbots and Assistant Agents are referred to as the same thing.

2.1 What are the chatbots

As previously mentioned on the first chapter, a chatbot is an artificially intelligent answering agent, which serves the purpose of assisting users on different tasks in natural language. Their biggest advantage which distinguishes them among other services for information retrieval procedures, is their ease of use in terms of, how fast they can be asked for information having the least amount of clicks or taps made and as well as how fast and efficiently they respond back.

A Chatbot's aim is to make the communication between humans and computers happen in natural language just like humans communicate with each other. The first implementation on Chatbots roots back in the 1960's. The reason of their creation back then was to see if they will be able to fool users that they were real humans, nevertheless, the main reason that the implementation of chatbots began was not for entertainment.

Chatbots are usually built on specific domains, to provide assistance on specific topics but this is not always the case since the knowledge base that is running on the background may be rich with lots of data, rather than poor. It is also dependent on the way that the engineer will design, develop and train the Chatbot and its engine.

There are currently many chatbots created and deployed on the Internet for the purpose of FAQ answering, information seeking, website guidance and more. Most of these are structured by dialogue management modules via which they control the conversation process and by chatbot knowledge bases to respond to user queries. These bases contain a set of templates that match user inputs and further on, generate responses. The implementation of chatbot knowledge bases is a time demanding and time consuming procedure because the templates that are currently used in chatbots are hand coded.

Depending on the way that the implementation has been done, chatbots could respond to users' queries in a conversation-like style or in a direct-answers style. For example, there is a chatbot on the web named Cleverbot which has been implemented for entertainment purposes and its responses are directed towards a conversational behavior. On the other hand, a chatbot could be looking like more a Question Answering (QA) system, where the user asks for information and the chatbot returns direct answers. For this Thesis, a chatbot that will mainly behave like a QA system will be developed instead of a conversational agent.

In other words, the goal is the creation of a chatbot where users will ask for things like "Find me a gas station" or "I am out of fuel" and the chatbot will return a list of gas stations, returning the most relevant results first. The relevancy of the results will be depending on the gas prices and the distance from users' current location.

Before describing and analyzing the chatbot frameworks, first it should be mentioned in detail, why chatbots are useful and what are the benefits for businesses of having their own.

To mention the most important benefits, these are the availability and reusability. Availability refers to the fact that a chatbot is always available, at all hours of the day and all days of the week. The frustration of waiting and queuing until queries get answered or service is provided, is well known. Not only that, but this also results to work being delayed which further on, leads to loss of interest in a customer.

Chatbots are an ultimate solution for this issue as they are always connected to the back-end services that makes them able to accurately answer to questions in a human-like way, through live chats (i.e. Facebook Messenger, Whatsapp etc.) which are available at any time.

Continuing, reusability states the fact that chatbots can handle and serve multiple users simultaneously. Also, they can personalize the user experience, having as a result a better service in general for the business.

Further on, it's a cost effective solution. Having personnel to serve real time customer service via chat, is a demanding and expensive task. In addition, not all personnel will provide the same quality of information. On the other hand, a chatbot is an one time investment. Contrasting Chatbots to personnel, the expenses of maintaining a chatbot are less, comparing to the recurring expenses for personnel and also the service provided is more effective.

Finally, via the use of chatbots, analytics can be gathered to generate have final reports on the usefulness and usage of them, helping to the business' growth.

2.2 Chatbot Frameworks

As we previously mentioned, a chatbot can save time and money for businesses and as well help employees work efficiently by dedicating time on other tasks rather than customer service.

Chatbots are supported by a range of different platforms and so, they can be built on pre-existing models like Skype, Slack, Facebook Messenger, Whatsapp, Cortana, Kik and more, to interact with the users.

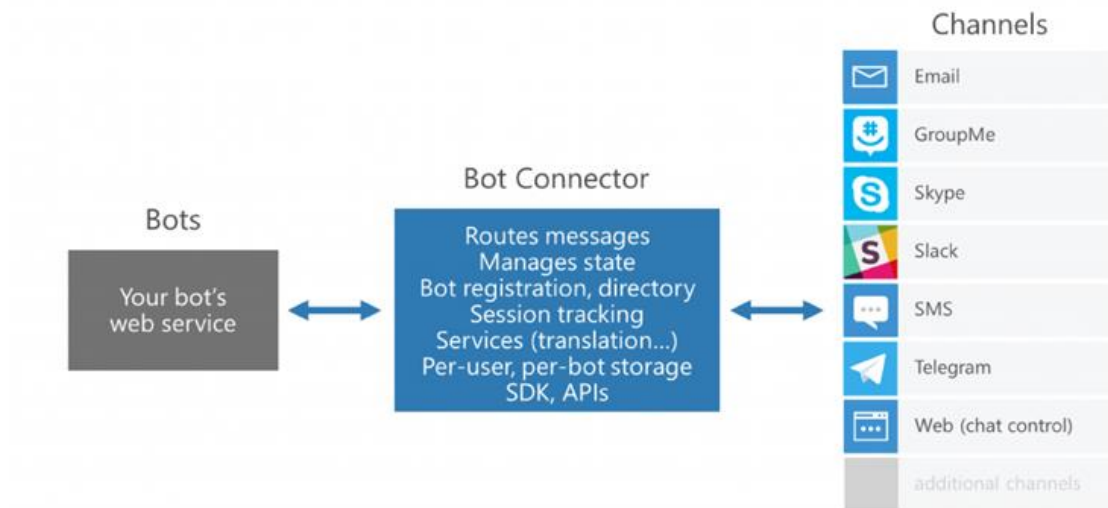


Figure 1 – Integration between bots and channels

For example, Facebook Messenger chatbots are appearing everywhere as companies try to improve the quality of customer service for low level functions mostly. A bot built on Facebook Messenger, is an interactive intelligent agent which gets initiated when users click on the Messenger icon on a Facebook Page or Website and then a Messenger session is launched automatically, allowing the users to type a question and begin chatting with the bot.

There are several ways to create a chatbot. In this part of the dissertation, the most known available frameworks for implementing a chatbot are going to be described, compared and contrasted.

Some of the most popular frameworks are the:

- Microsoft Bot Framework
- Dialogflow

- IBM Watson
- Botkit
- Pandorabots
- ChatterBot
- Wit.ai
- Botpress
- RASA Stack

Further on, the benefits, pros and cons of each framework are going to be analyzed and described.

Microsoft Bot Framework

Starting with the Microsoft Bot Framework, it is a platform which allows to build, connect, publish and manage smart and interactive chatbots, providing the best user experience.

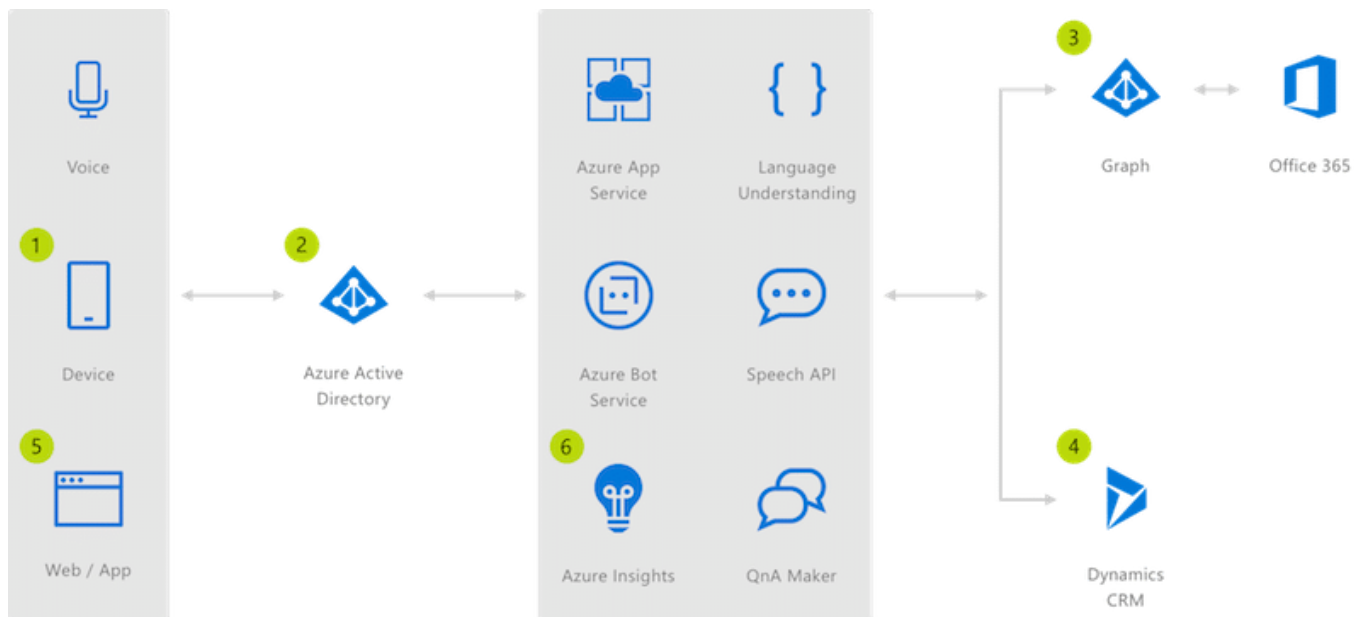


Figure 2 – Microsoft Bot Framework Flow Diagram

An advantage of building a bot on Microsoft’s framework is that it can be integrated with a Microsoft’s API called Microsoft Cognitive Services. These services are a set of machine learning algorithms that Microsoft has developed to solve problems in the field of Artificial Intelligence (AI). Developers and programmers can use these algorithms through REST calls

over the Internet to the Cognitive Services. As Microsoft states, “*Microsoft Bot Framework will let you turn your ideas into a reality*”.

Dialogflow

Continuing with Dialogflow, this framework better supports built text or voice-based conversational interfaces for bots and applications. Dialogflow is powered by Google’s machine learning, making it able to connect to users on Google Assistant, Amazon Alexa, Mobile apps, Messenger, websites, Slack, Twitter and many more.

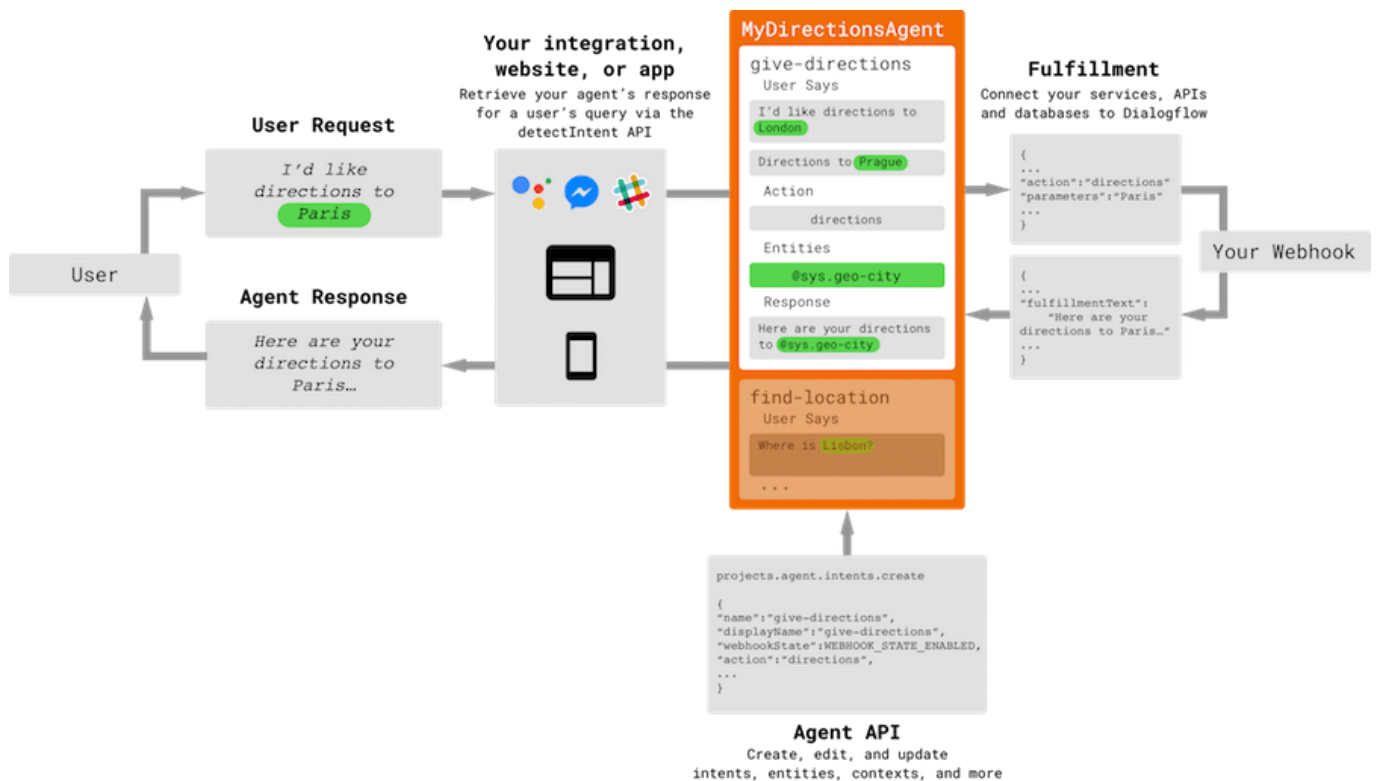


Figure 3 – Dialogflow Flow Diagram

Dialogflow is able to serve millions of users as it runs on the Google Cloud Platform, making it scalable. Also, it is user and developer friendly, it supports many languages giving it the ability to become personalized and it is the best solution for NLP based applications development. Node.js, PHP, Java (Maven), Ruby (Gem), Python, C# and Go are some of the programming languages that can be used to develop bots in Dialogflow.

IBM Watson

Next comes IBM Watson, which is built on a neural network of one billion Wikipedia words. Users can use bots built on this framework through platforms like Mobile Devices, Websites, Robots and Messaging Applications. It uses machine learning to respond to natural language input from the aforementioned platforms.

Watson assistant is a helpful tool which guides engineers through the development of a chatbot for businesses, easily and quickly. It comes with a free 10.000 calls per month and has available some sample bot codes for retail and banking.

Botkit

Botkit is one of the leading developer tools for building bots as there are more than 10.000 bots developed on Botkit and are already in use. It makes the development process easier as it assists the developers to build their bots with the help of a visual conversational builder and not only that, it also allows them to add plugins regarding their needs. It works on the natural language processing engine LUIS.ai and also includes open source libraries like Node.js. Finally, it integrates with platforms like Slack, Microsoft, IBM Watson, Facebook, Twilio, Api.ai, Glitch, Heroku and Cisco Spark.

Pandorabots

Pandorabots is widely recognized as an Artificial Intelligence as a Service (AIaaS) platform as it uses the Artificial Intelligence Markup Language (AIML) and as well includes the Artificial Linguistic Internet Computer Entity (A.L.I.C.E.) which is a natural language processing bot. In addition, the framework has recently been updated to include a new feature wherein, AIML can be visualized.

The integration of Pandorabots is possible on Websites and various applications such as Cortana, Messaging platforms and more. Finally, it comes with SDKs like Java, Node.js, Ruby, Python, PHP and Go.

ChatterBot

This framework is driven by Node.js and the bot automates the whole flow through machine learning. It is also language independent and functions by creating a Python Library, having as a result for the bot to be trained in any desired language.

The working mechanism of the bot is direct and clear. The more it gets fed by data and input, the more it will be able to process the output efficiently by providing accurate results.

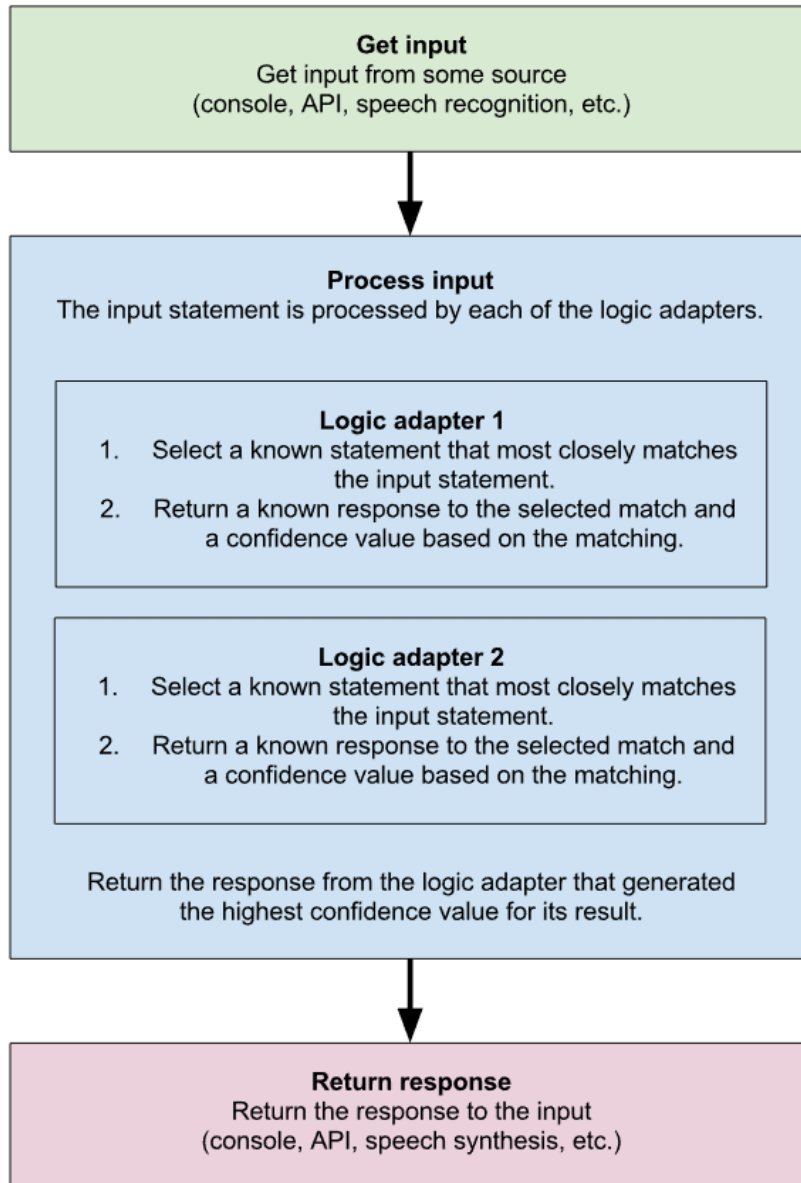


Figure 4 – ChatterBot Flow Diagram

Wit.ai

Apart from only building bots, this framework also supports the development of automation for wearable devices, voice interface for a mobile and home electronics hardware. It is free and available with the Node.js, Python and Ruby SDKs. It can also be used as HTTP API.

Botpress

Botpress is an open-source platform which is based on a modular architecture having as main features the following: Editor (flexible flow management system), Natural Language understanding, Actionable Analytics and finally it can be used on all significant platforms like Skype, SMS, Wechat and more. By using this platform, bots can be implemented locally and then deployed on a cloud.

RASA Stack

RASA is as well an open-source platform for building bots which is based on machine learning and it functions on two main integrands, the Rasa NLU which is a natural language processor of the bot and Rasa Core which works on the inputs based on intent and entities. Some of its main features are: Manage Contextual Dialogues, Recognize Intents, Exact Entities, Full Data Control, Connect your APIs and Custom Models.

Chapter 3: Implementation

In this part of the dissertation, the whole implementation process is going to be described thoroughly. First of all, all of the tools that have been used throughout the implementation will be mentioned and analyzed. Afterwards, an emphasize will be given on the programming languages that were used to develop the backend, which will feed the chatbot with accurate data. Continuing, there will be a detailed description on the tools and frameworks that have been used to build the Chatbot and finally, the technologies and best practices behind the whole aforementioned infrastructure are going to be explained.

3.1 Tools, programming languages & technologies

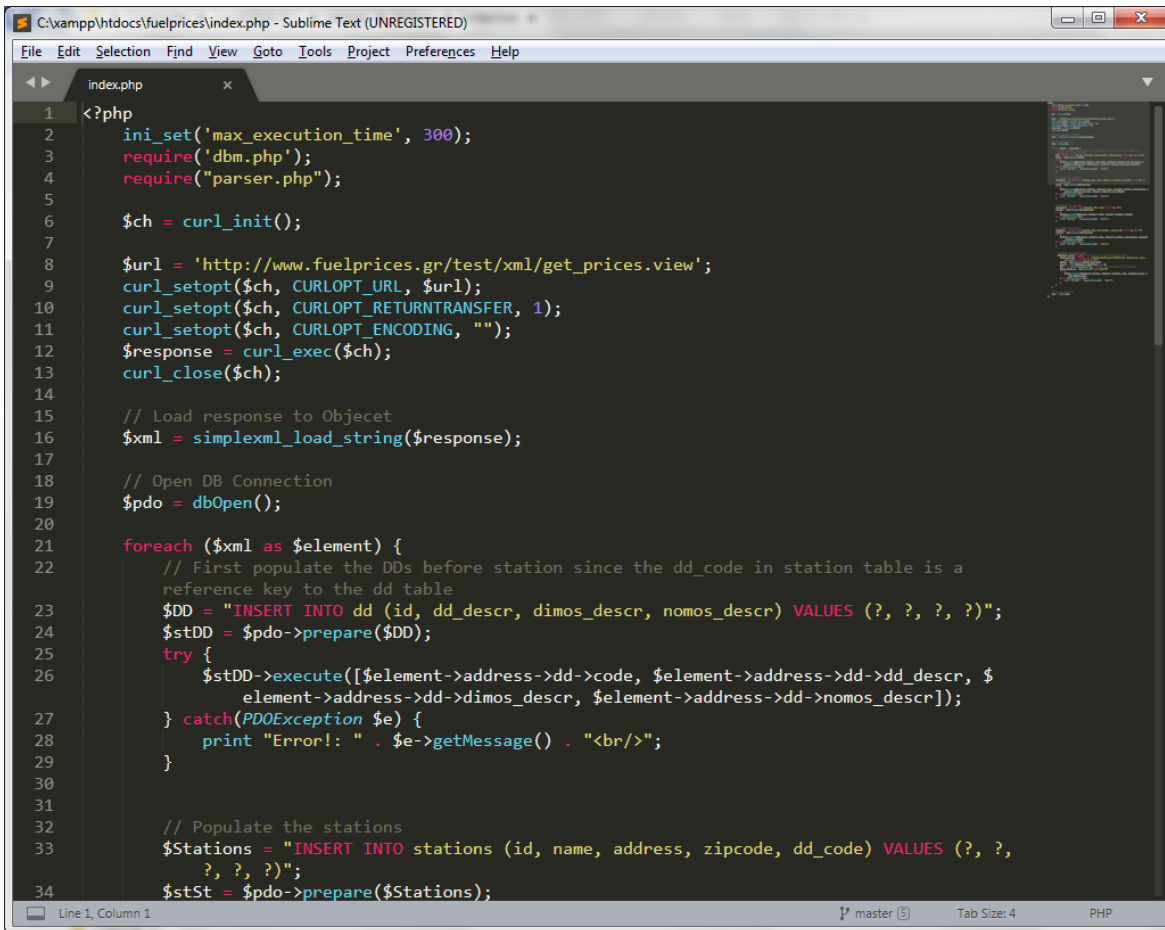
3.1.1 Tools

The tools that were used throughout the implementation of the Chatbot will be listed below and then will be analyzed one by one.

- 1. Sublime**
Text editor for the code development.
- 2. XAMPP**
Cross platform web server application.
- 3. Google Chrome**
Web browser for loading, rendering and browsing websites.
- 4. Postman**
Platform for API development.
- 5. Datagrip**
Strong and advanced database management software tool.
- 6. Botsociety**
Design tool for mockups creation for chatbots or voice assistants.
- 7. Firebase Dialogflow - Google Assistant**
Google's advanced platform to design, develop, test and deploy Chatbots or Assistant Agents.

Sublime

All of the back-end has been coded from scratch. All of the code has been written by hand using the Sublime text editor. Sublime is a free powerful text editor widely used by software engineers and developers all over the world. It supports many programming and markup languages. Its strongest attribute which make it famous and worth using it, is that it supports the installation of community-built add-ons or plugins, which can extend its functionalities and features. One great add-on is the “Code Prettify” which can extend and prettify minified CSS and JS files and of course any other kind of code. It also has a good default dark skin with a selection of good contrasting colors, making coding an easy task for developers without getting their eyes tired.



```
1 <?php
2     ini_set('max_execution_time', 300);
3     require('dbm.php');
4     require("parser.php");
5
6     $ch = curl_init();
7
8     $url = 'http://www.fuelprices.gr/test/xml/get_prices.view';
9     curl_setopt($ch, CURLOPT_URL, $url);
10    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
11    curl_setopt($ch, CURLOPT_ENCODING, "");
12    $response = curl_exec($ch);
13    curl_close($ch);
14
15    // Load response to Object
16    $xml = simplexml_load_string($response);
17
18    // Open DB Connection
19    $pdo = dbOpen();
20
21    foreach ($xml as $element) {
22        // First populate the DDs before station since the dd_code in station table is a
23        // reference key to the dd table
24        $DD = "INSERT INTO dd (id, dd_descr, dimos_descr, nomos_descr) VALUES (?, ?, ?, ?)";
25        $stDD = $pdo->prepare($DD);
26        try {
27            $stDD->execute([$element->address->dd->code, $element->address->dd->dd_descr, $
28            element->address->dd->dimos_descr, $element->address->dd->nomos_descr]);
29        } catch(PDOException $e) {
30            print "Error!: " . $e->getMessage() . "<br/>";
31        }
32
33        // Populate the stations
34        $Stations = "INSERT INTO stations (id, name, address, zipcode, dd_code) VALUES (?, ?,
35        ?, ?, ?)";
36        $stSt = $pdo->prepare($Stations);
```

Figure 5 – Screenshot of Sublime

XAMPP

XAMPP is a free and open source cross platform web server tool, developed by Apache Friends. It is consisted by the Apache HTTP Server and the MariaDB database. It can make any computer function as a server, making it able to serve websites. It is widely used by millions of developers and is easy to set it up fast.

First of all, XAMPP has been used to create a local server. This server was then used for the testing and hosting of the software that was written for the implementation of the back-end.

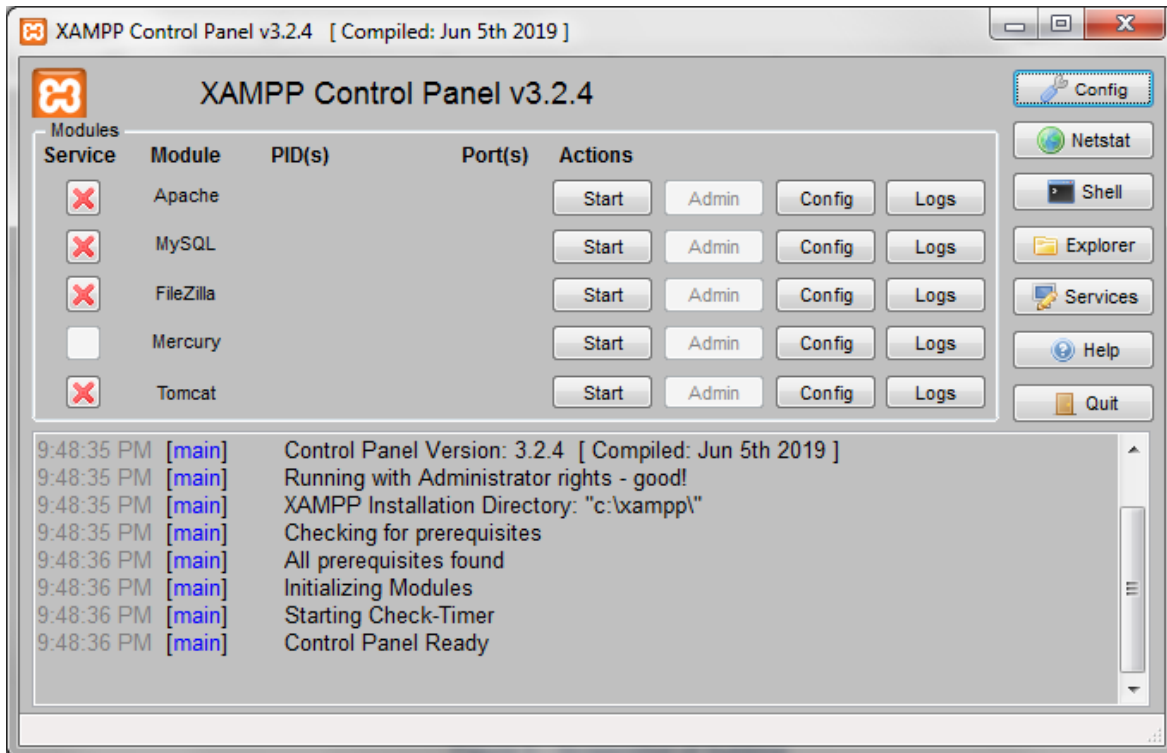


Figure 6 – Screenshot of XAMPP

Google Chrome

Google Chrome is a cross-platform web browser created by Google and is probably the most used browser worldwide. This has been selected for the implementation due to its great developer tools.

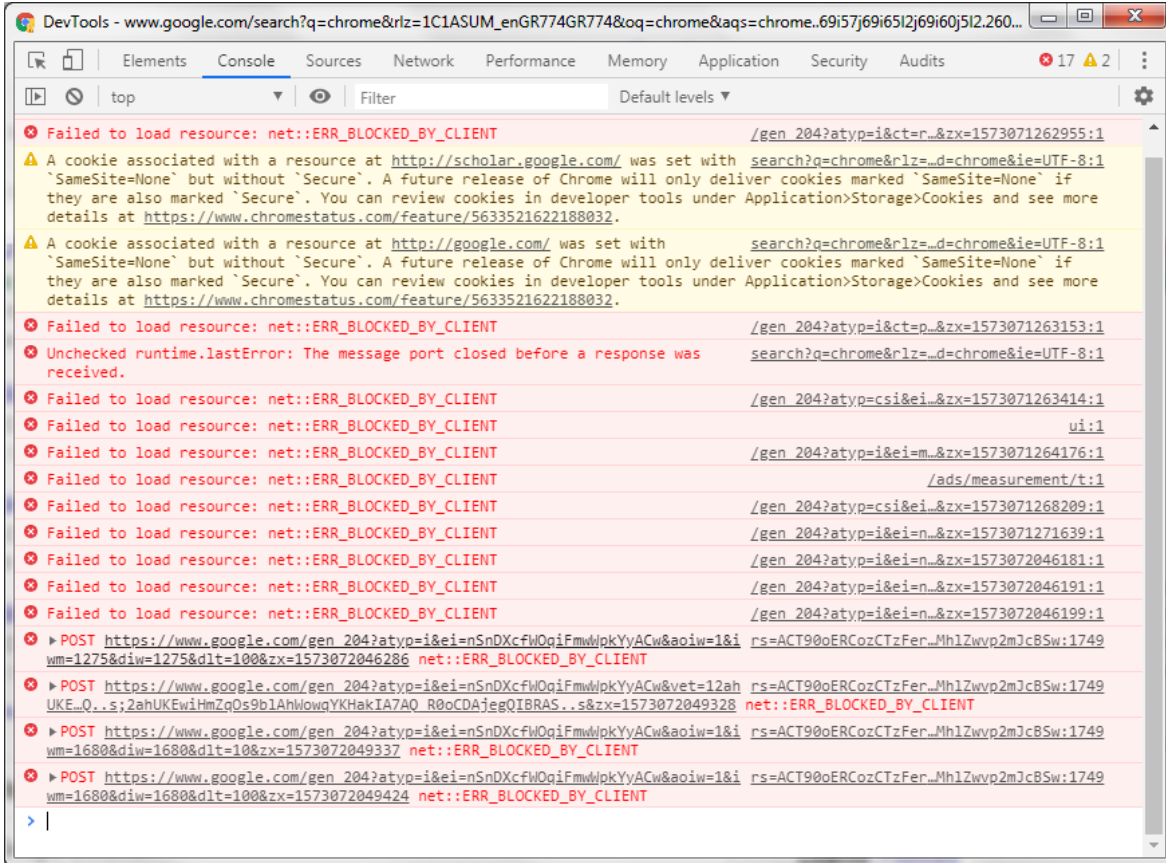


Figure 7 – Screenshot of Chrome Developer Tools

Postman

Postman is a collaboration platform for API development. It contains strong features that simplify each step of building an API and streamline collaboration to create strong reliable APIs.

It can be easily used by simply setting the URL of the API that needs to be called and by setting the headers and body of the call that will be performed via HTTP. It also supports all kind of requests like Post, Get, Put, Delete, Patch and more.

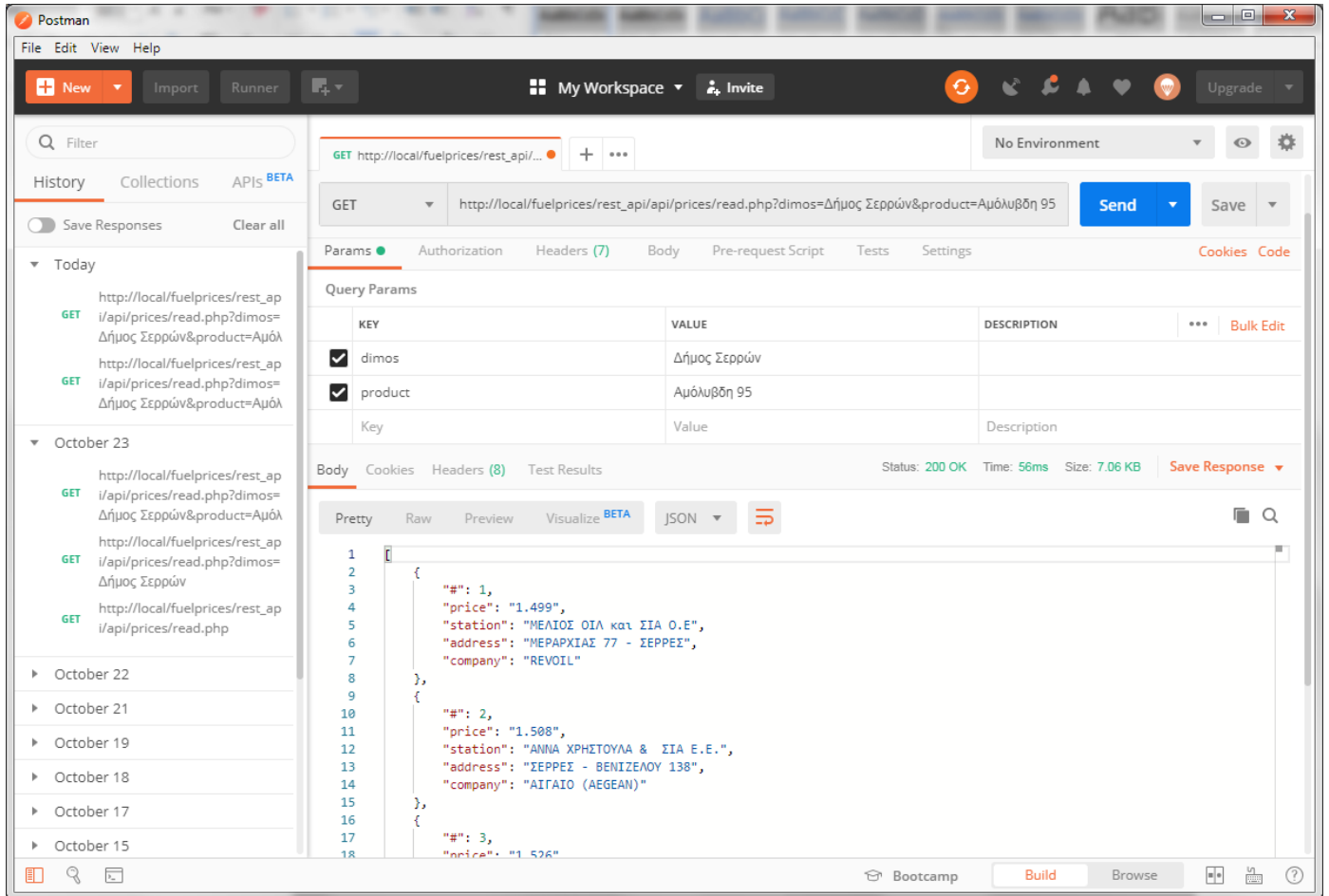


Figure 8 – Screenshot of Postman

Datagrip

Datagrip is a strong database IDE that is tailored to suit the specific needs of professional SQL developers. It gives an extended insight into how the queries work and into the database engine behavior, making the queries more efficient. Jumping and switching fast to any table, view, or procedure gets totally easy and efficient due to the great user interface it has. Also, it is possible to execute queries in different modes and provides a local history that keeps track of all the activity and protects developers from losing their work.

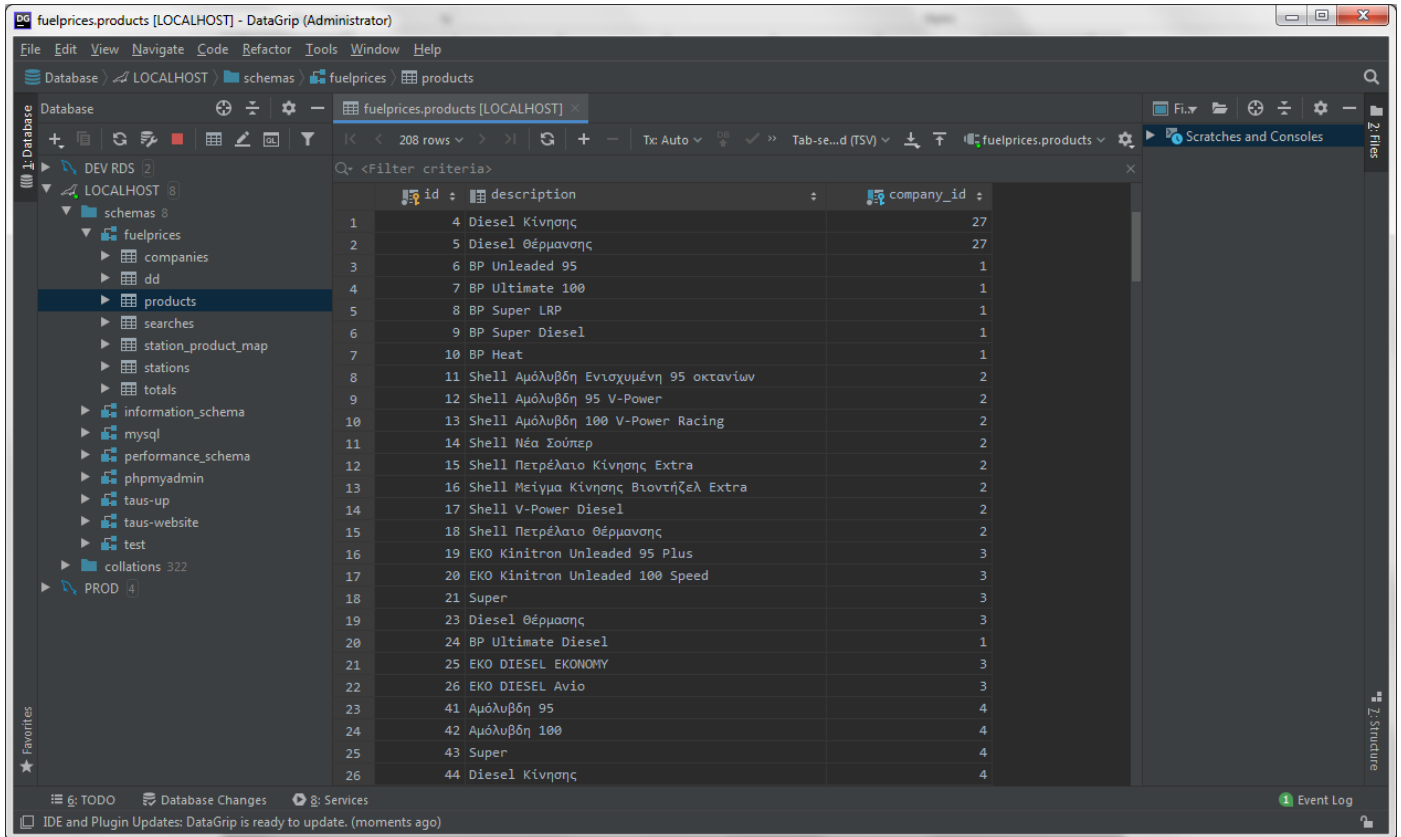


Figure 9 – Screenshot of Datagrip

Botsociety

Botsociety is a design tool for designing, creating, previewing and prototyping Chatbots and voice assistant agents. It has a friendly user interface where developers can design the flows and paths that the users are going to follow during the conversation with the agent. A great advantage of using this tool is that it can export the prototype design to Google's Dialogflow or Rasa directly, where developers can then further develop the actual agent and all of the functionalities that it will serve.

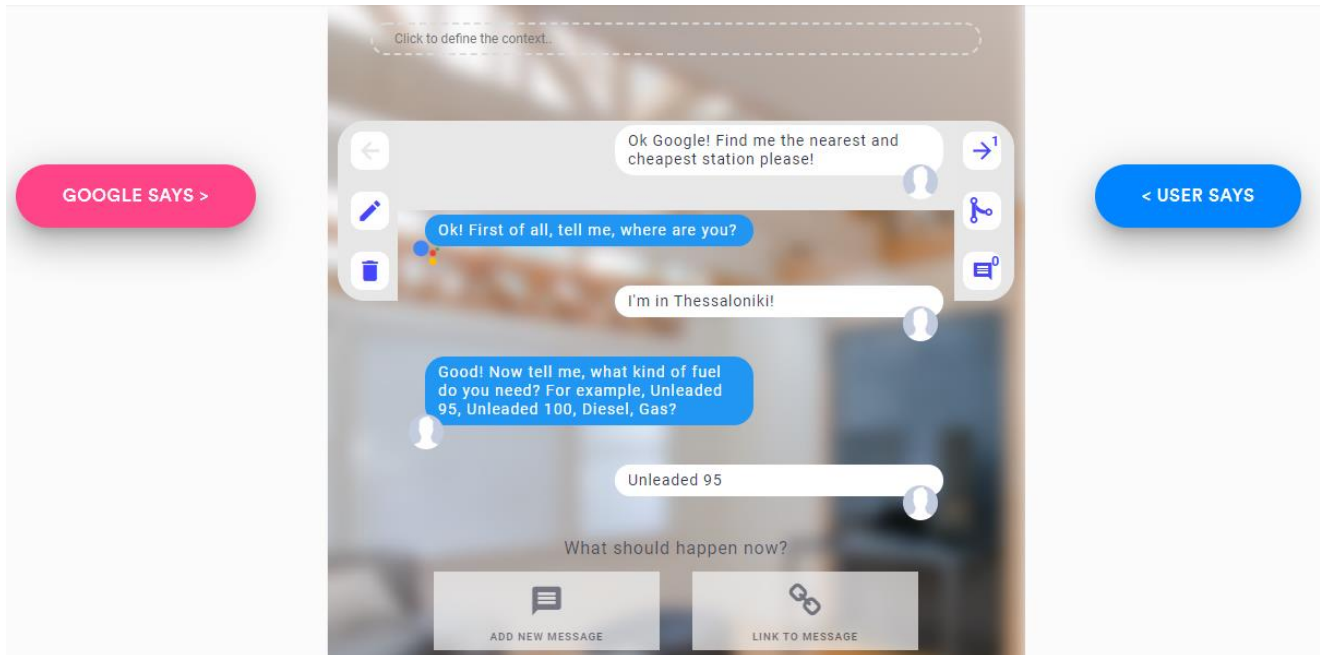


Figure 10 – Screenshot of Botsociety UI

Dialogflow - Google Assistant

Google's Dialogflow is a strong platform for developing, testing and deploying chatbot agents. There is a lot of documentation supporting the whole platform and with its advanced and friendly user interface, developers can almost do anything regarding the requirements of the chatbot's implementation.

The screenshot displays the Dialogflow console for an agent named 'GetFuel'. The left sidebar shows the navigation menu with 'Intents' selected. The main workspace is divided into several sections:

- Contexts**: A section for defining contexts.
- Events**: A section for defining events.
- Training phrases**: A section for adding user expressions. It includes a search bar and three example phrases: "I'm in Serres and I need Unleaded 95", "I'm in Thessaloniki and I need Diesel", and "I'm in Thessaloniki and I need Diesel".
- Action and parameters**: A section for defining actions and their parameters. It includes a text input for the action name and a table of parameters.

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	locat	@sys.any	Slocat	<input type="checkbox"/>
<input type="checkbox"/>	prod	@sys.any	Sprod	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

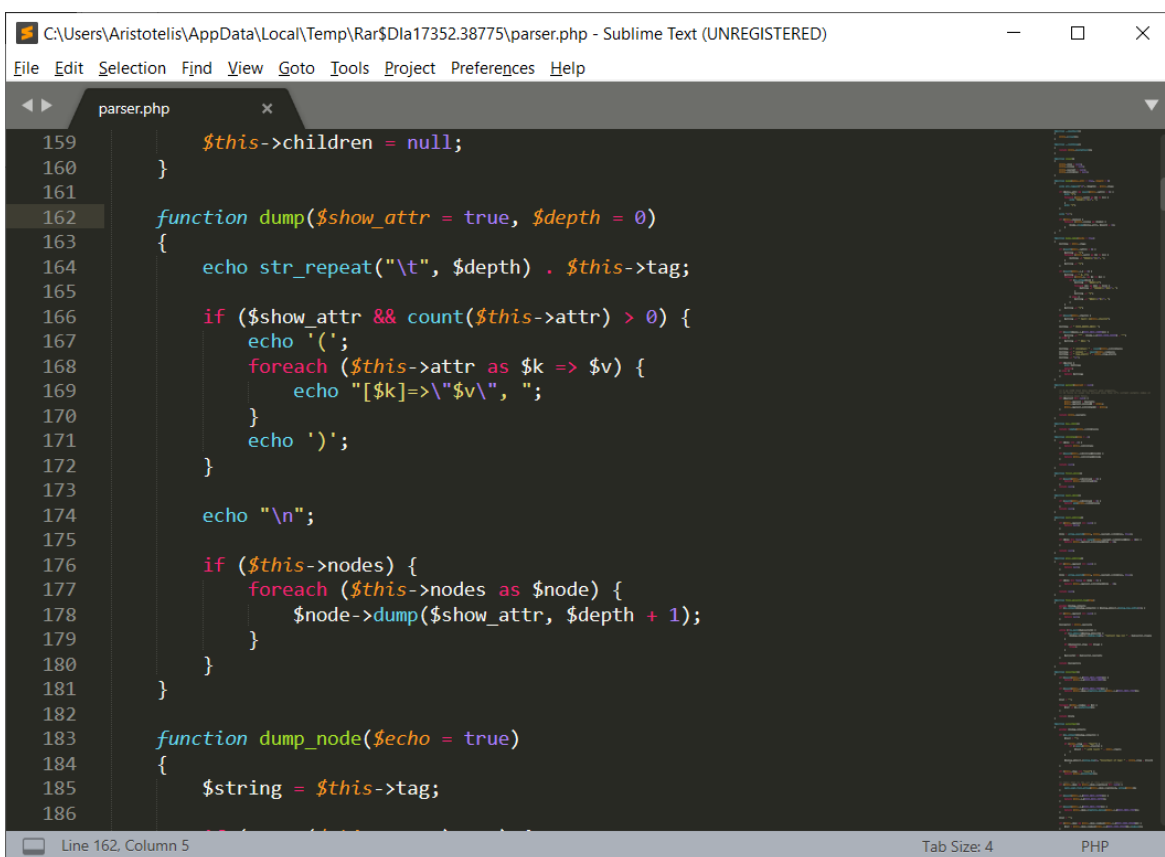
Figure 11 – Screenshot of Dialogflow UI

3.1.2 Programming Languages

PHP

The main programming language that has been used for the implementation of the back-end infrastructure is the PHP. The PHP acronym originally derived from “Personal Home Page” Tools but now stands for Hypertext Preprocessor. It is a powerful backend language used in more than 80% of the global web applications. Also, it is very simple and easy to learn, making it one of the top programming languages. Also, there is a lot of documentation behind PHP on the web and it comes with lots of developer friendly ready-made functions and libraries for many purposes and functionalities such as CURL and PDO.

On the other hand, coding in PHP can be a tedious and time consuming job, as in many cases, writing in PHP requires to be done from scratch even for single functions. Therefore, built in PHP libraries got introduced in the market to make coding more efficient and less time consuming at the same time.



```
C:\Users\Aristotelis\AppData\Local\Temp\Rar$Dla17352.38775\parser.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

parser.php
159     $this->children = null;
160     }
161
162     function dump($show_attr = true, $depth = 0)
163     {
164         echo str_repeat("\t", $depth) . $this->tag;
165
166         if ($show_attr && count($this->attr) > 0) {
167             echo '(';
168             foreach ($this->attr as $k => $v) {
169                 echo "[$k]=>\"$v\", ";
170             }
171             echo ')';
172         }
173
174         echo "\n";
175
176         if ($this->nodes) {
177             foreach ($this->nodes as $node) {
178                 $node->dump($show_attr, $depth + 1);
179             }
180         }
181     }
182
183     function dump_node($echo = true)
184     {
185         $string = $this->tag;
186
```

Figure 12 – PHP Sample Code

SQL

The Structured Query Language (SQL) is a standard language for storing, manipulating and retrieving data in databases. Also described as a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS) or for stream processing in a relational data stream management system (RDSMS), it is particularly useful in handling structured data.

Its bigger advantages over the older read–write methods to store and access data such as ISAM or VSAM are, firstly, it introduced the concept of accessing many records with one single command and secondly, it eliminates the need to specify how to reach a record, with or without an index.

SQL is based upon relational algebra and tuple relational calculus and provides many types of statements, which may be informally classed as sublanguages. These are: a data query language (DQL), a data definition language (DDL), a data control language (DCL), and a data manipulation language (DML). The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification) and data access control. Although SQL is essentially a declarative language, it includes also procedural elements.

```
39 create table companies
40 (
41     id int not null,
42     name varchar(255) null,
43     constraint company_pk
44     primary key (id)
45 );
46
47 select * from fuelprices.stations where address like '%address%';
48
49 alter table stations
50     add constraint stations_dd_fk
51     foreign key (dd_code) references dd (id);
52
53 alter table station_product_map
54     add constraint station_product_map_station_fk
55     foreign key (station_id) references stations (id);
```

Figure 13 – SQL Sample Code

Javascript via Node.js

JavaScript, often abbreviated as JS, is a high-level, instantaneously compiled, object-oriented programming language. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Also, it is the programming language of HTML and the Web and is easy to learn.

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.



```
index.js package.json
26 function getFuelHandler(agent) {
27   const locat = agent.parameters.locat;
28   const prod = agent.parameters.prod;
29   // agent.add(locat);
30
31   return axios.post('https://benzini.gr/rest_api/api/prices/read.php', {
32     dimos: locat,
33     product: prod
34   }).then((response) => {
35     response.data.map(dataObject => {
36       agent.add('Address: '+dataObject.address+ ' | Price: € '+dataObject.price);
37     });
38   }, (error) => {
39     console.log(error);
40   });
```

View execution logs in the [Firebase console](#) Last deployed on 11/25/2019 18:57 [DEPLOY](#)

Figure 14 – Javascript written inside a JS Node running in Google’s servers

3.2.3 Technologies

For the implementation of this dissertation, apart from the use of tools and the programming languages, there has also been a selection of specific technologies which serve certain purposes. The technologies used are:

- cURL Requests
- PHP's PDO
- Apache Server
- MariaDB Storage Engine
- Axios JS

cURL is a PHP library that executes HTTP requests and also transfer files or download data over HTTP and FTP protocols. It supports proxies, transfer of data over SSL connections, can set cookies and even get files that are behind a login.

In order to use PHP's cURL functions, libcurl package must be installed first. The XAMPP web server that has been used, had already the cURL package installed and many other PHP libraries as well.

In order to perform a cURL HTTP request, a cURL session must be initialized first by calling the function `curl_init()`. This function instantiates a new session and returns a cURL handle to be used with the functions `curl_setopt()`, `curl_exec()` and `curl_close()`.

The `curl_setopt()` function has to be used next to set the options for the cURL transfer. There is a very big number of available options to be used and the selection of these is being made by the needs and the circumstances associated with the call that has to be performed.

Continuing, the `curl_exec()` function, executes the given cURL session. This function is normally being called after initializing a cURL session and setting all the options for the session.

And finally, the `curl_close()` closes the cURL session and frees all the resources that had been reserved for the session.

cURL has been used for this Thesis, in order to get the contents of the "http://www.fuelprices.gr/test/xml/get_prices.view" URL end point, which contain the full dataset of all the gas prices information for all the gas stations in whole Greece. The data returned by the cURL is a raw string and therefore, it is not easily manageable. For this reason, PHP's `simplexml_load_string()` function has been used to transform the string into data objects, which can be easily read, processed and manipulated.

```

1 <?php
2     ini_set('max_execution_time', 300);
3     require('dbm.php');
4
5     $ch = curl_init();
6
7     $url = 'http://www.fuelprices.gr/test/xml/get_prices.view';
8     curl_setopt($ch, CURLOPT_URL, $url);
9     curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
10    curl_setopt($ch, CURLOPT_ENCODING, "");
11    $response = curl_exec($ch);
12    curl_close($ch);
13
14    // Load response to Object
15    $xml = simplexml_load_string($response);
16
17    // Open DB Connection
18    $pdo = dbOpen();

```

Figure 15 – cURL initialization, setting of options, execution and closing of session

As already said, the response of the cURL call, is a huge raw text string. As shown in the figure below, a browser renders the data in an unstructured and compressed huge string. The part of the text that has been marked, is the information for a gas station for one single product which in this case is the fuel Unleaded 95.

```

2 ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ 10442 Α1010400 Δ' Δ.Δ. Δήμου
Αθηναίων<BR />(Ακ. Πλάτωνος - Κολωνός - Σεπόλια) ΔΗΜΟΣ ΑΘΗΝΑΙΩΝ ΝΟΜΑΡΧΙΑ ΑΘΗΝΩΝ 11
Shell Αμόλυβδη Ενισχυμένη 95 οκτανίων 1.538 1573101131823 2 SHELL 2 ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ
ΛΕΝΟΡΜΑΝ ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ 10442 Α1010400 Δ' Δ.Δ. Δήμου Αθηναίων<BR />(Ακ.
Πλάτωνος - Κολωνός - Σεπόλια) ΔΗΜΟΣ ΑΘΗΝΑΙΩΝ ΝΟΜΑΡΧΙΑ ΑΘΗΝΩΝ 12 Shell Αμόλυβδη 95 V-
Power 1.648 1573101159740 2 SHELL 2 ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ ΛΕΝΟΡΜΑΝ 187-191
ΑΘΗΝΑ 10442 Α1010400 Δ' Δ.Δ. Δήμου Αθηναίων<BR />(Ακ. Πλάτωνος - Κολωνός - Σεπόλια) ΔΗΜΟΣ
ΑΘΗΝΑΙΩΝ ΝΟΜΑΡΧΙΑ ΑΘΗΝΩΝ 13 Shell Αμόλυβδη 100 V-Power Racing 1.948 1573101190663 2
SHELL 2 ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ 10442 Α1010400 Δ' Δ.Δ.
Δήμου Αθηναίων<BR />(Ακ. Πλάτωνος - Κολωνός - Σεπόλια) ΔΗΜΟΣ ΑΘΗΝΑΙΩΝ ΝΟΜΑΡΧΙΑ
ΑΘΗΝΩΝ 16 Shell Μείγμα Κίνησης Βιοντήζελ Extra 1.298 1573101214990 2 SHELL 2 ΕΡΜΗΣ ΑΕΜΕΕ
ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ 10442 Α1010400 Δ' Δ.Δ. Δήμου Αθηναίων<BR />

```

Figure 16 – Datasets in XML form and browser’s output in raw text

It is worth mentioning here that inspecting the results shown in the figure above using the Google Chrome's developer tools, revealed the actual structured XML format of the response, something that is being shown in the figure below.

```
<pricelist>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
  <priceentry>
    <station>3</station>
    <name>ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΠΕΤΖΙΚΙΟΥ</name>
    <address>
      <fulladdress>4,5 ΧΛΜ. ΘΕΣ/ΝΙΚΗΣ-ΑΣΒΕΣΤΟΧΩΡΙΟΥ, ΚΟΙΝΟΤΗΤΑ ΠΕΥΚΩΝ - ΘΕΣ/ΝΙΚΗ</fulladdress> == $0
      <zipcode>57010</zipcode>
      <dd>
        <code>54420400</code>
        <dd_descr>Δ.Δ.Χορτιάτη</dd_descr>
        <dimos_descr>ΔΗΜΟΣ ΧΟΡΤΙΑΤΗ</dimos_descr>
        <nomos_descr>ΝΟΜΟΣ ΘΕΣΣΑΛΟΝΙΚΗΣ</nomos_descr>
      </dd>
    </address>
    <product>
      <code>16</code>
      <description>Shell Μείγμα Κίνησης Βιοντήζελ Extra</description>
    </product>
    <price>1.338</price>
    <timestamp>1573190284930</timestamp>
    <company>
      <code>2</code>
      <name>SHELL</name>
    </company>
  </priceentry>
  <priceentry>...</priceentry>
  <priceentry>...</priceentry>
```

Figure 17 – XML structure of response shown in developer tools

PDO stands for PHP data objects and is currently the most secure way to read and write data to a database with PHP and SQL. PDO is a PHP library included in PHP 5.1 and later, and is a lightweight and consistent interface for accessing databases in PHP.

Apache is a free and open-source cross-platform web server software. The vast majority of Apache HTTP Server instances run on Linux but newer versions can also run on Microsoft Windows and other Unix-like systems. Apache played a key role in the initial growth of the World Wide Web. Apache web server was used for the development and thorough testing of the back end written in PHP.

MariaDB is a community developed and commercially supported fork of the MySQL relational database management system, intended to remain a free and open-source software under the GNU General Public License. Both Apache and MariaDB come ready in the

XAMPP package and therefore this is the reason of choosing to use MariaDB for the SQL engine to serve queries to the database.

To use XAMPP, the only thing that has to be done is to download the package from the official website and install it. After that, by launching the program, a list of services can be started to be used. As shown in the figure below, for the implementation of the chatbot, Apache and MySQL have been used to fetch and store the data in the database.

XAMPP can be thoroughly configured and create Apache servers to cover the needs of developers. With some deep knowledge on networks, XAMPP can also be configured in such a way to serve websites not only locally but on the World Wide Web as well.

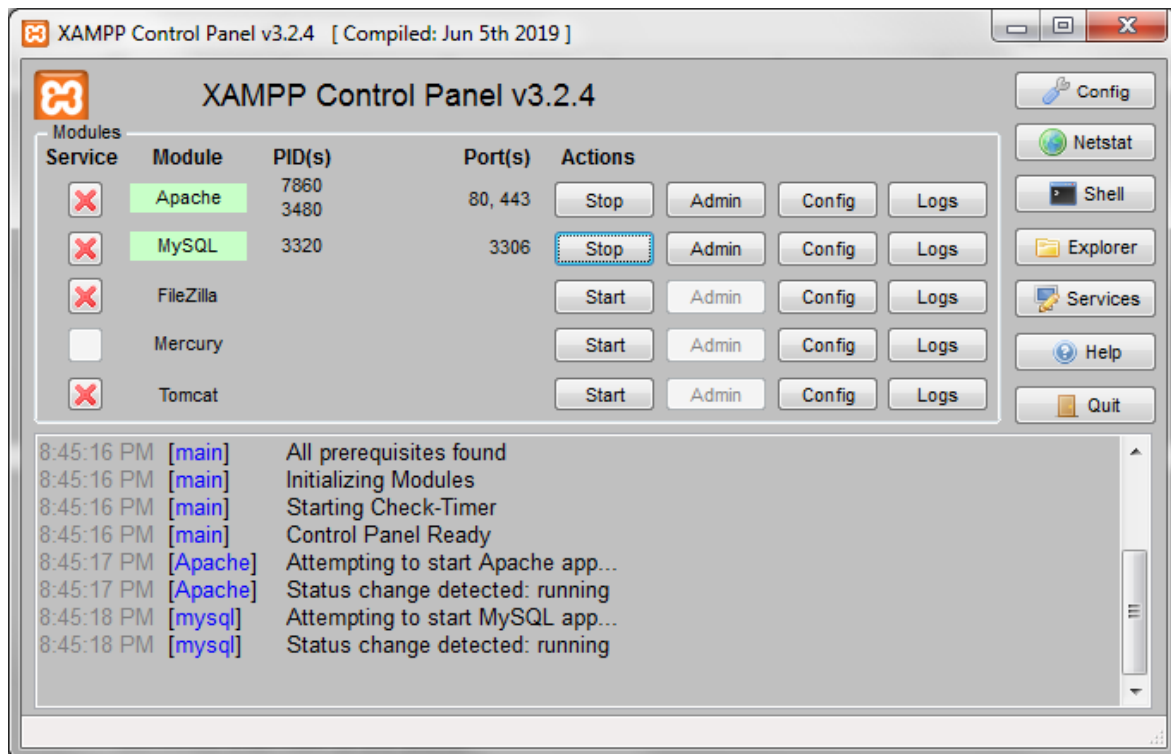


Figure 18 – XAMPP interface and services

Axios is a promise-based HTTP client that works both in the browser and in a Node.js environment. It basically provides a single API for dealing with XMLHttpRequests and node's http interface. Besides that, it wraps the requests using a polyfill for ES6 new's promise syntax.

3.2 Implementation Process

In this part of the dissertation, the whole implementation is going to be described in detail. As said before, the development of the back-end was divided in three parts. The first one was the writing of the Scraper. The second was the design and creation of the database and finally, the third part was the development of the RESTful API.

3.2.1 Scraper

The first part of the implementation was the designing and coding of the web scraper. A scraper's job is to get the contents from a website, then structure those, analyze them and finally take actions on them like to store them in a database or manipulate them in any way. The scraper that has been implemented for this dissertation does exactly this.

First of all, PHP's cURL library is used to perform a call programmatically to the URL "http://www.fuelprices.gr/test/xml/get_prices.view" which is an end point from the Greece's Ministry of Development and Competitiveness. This URL contains full information for each gas station all over Greece and it is served in an XML structure. More specifically, a call to the above URL will return the following information:

1. Station ID
2. Station Name
3. Station Address
4. Station Zipcode
5. Municipal District ID
6. Municipal District Name
7. Municipal District Description
8. County Name
9. Product ID
10. Product Name
11. Product Price (for that station)
12. Timestamp (the date and time that the price was updated)
13. Company ID (the company that owns the product)
14. Company Name

Therefore, the scraper that has been implemented, dynamically gets all this information for all the gas stations in whole Greece, which are approximately a bit less than 5000 in total. The response of the call is an unstructured huge raw string which afterwards gets converted to data objects which can be analyzed, stored and manipulated.

The options that were chosen to perform the cURL session are the CURLOPT_URL, CURLOPT_RETURNTRANSFER and CURLOPT_ENCODING, as shown in figure 15. The first one is to set the URL to fetch the data from. In other words, the URL from which the gas stations and fuel prices data are going to be pulled. The second option is being used when the response of the call needs to be returned as a string instead of outputting it directly. In this case, this option has been used to fetch all of the data and then another PHP's built in function has been selected to transform the raw string to data objects. Finally, the third option has been used to enable the decoding of the response. The data that the cURL call fetches, contain Greek letters and characters which need to be decoded. Otherwise, the system would translate them to weird unreadable characters.

The image below represents the XML structure for one product only of one gas station. In detail, what it represents is an object with seven keys from which, the keys address, product and company contain nested objects as well.

```

index.php x price-entry-format.txt
SimpleXMLElement Object (
  [station] => 2
  [name] => ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ
  [address] => SimpleXMLElement Object (
    [fulladdress] => ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ
    [zipcode] => 10442
    [dd] => SimpleXMLElement Object (
      [code] => Α1010400
      [dd_descr] => Δ' Δ.Δ. Δήμου Αθηναίων (Ακ. Πλάτωνος - Κολωνός - Σεπόλτια)
      [dimos_descr] => ΔΗΜΟΣ ΑΘΗΝΑΙΩΝ
      [nomos_descr] => ΝΟΜΑΡΧΙΑ ΑΘΗΝΩΝ
    )
  )
  [product] => SimpleXMLElement Object (
    [code] => 11
    [description] => Shell Αμόλυβδη Ενισχυμένη 95 οκτανίων
  )
  [price] => 1.518
  [timestamp] => 1570538573583
  [company] => SimpleXMLElement Object (
    [code] => 2
    [name] => SHELL
  )
)

```

Figure 19 – Datasets in Object form. Structured, readable and manageable

To further explain how the scraper has been written, let's start by saying that it is consisted of two files and these are the index.php and the update.php. These two files serve more or less the same purpose but the main difference is that the handling of data differs between them. In more detail, the index file is used only for the initialization of the database. What this means is, this file is being ran only once, in cases where the whole infrastructure gets set up in a new server for the first time and therefore the database is empty and needs to be filled with all the current information that the Ministry's end point serve or when for example something went wrong and the data in the database got corrupted and needs to be refreshed by dropping all the tables, recreating them and finally refilling them with the aforementioned data.

The update file now is being used to maintain and update the prices for all of the existing information. This file also handles the cases where new stations or new products or new regions appear for the first time and stores them appropriately. The first file is being ran manually for the initialization whilst the second file gets run automatically by the server's cron job scheduler. The cron job tool is available only in linux environments. Throughout the implementation though, the environment used was Windows 7 and therefore, the Window's task scheduler was used to implement the cron jobber's functionality.

The time interval chosen for the updating of the database was 30 minutes in the development phse and will be 5 minutes for the production application. By updating the database every 5 minutes, accuracy of the data is ensured with the current up to date prices.

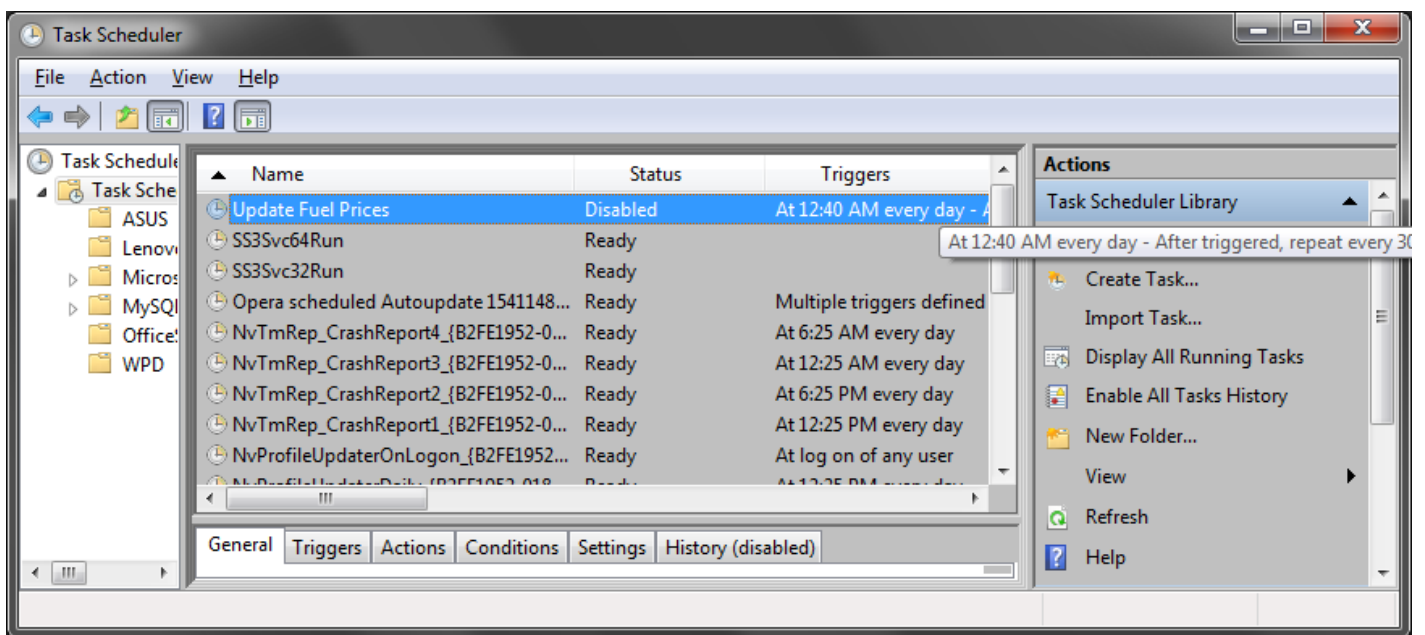


Figure 20 – Task scheduler interface

The figure above shows the task scheduler's interface. To schedule the execution of a PHP file every 30 minutes, a task must be created. The next step is to create a trigger which will perform the execution depending on the time interval.

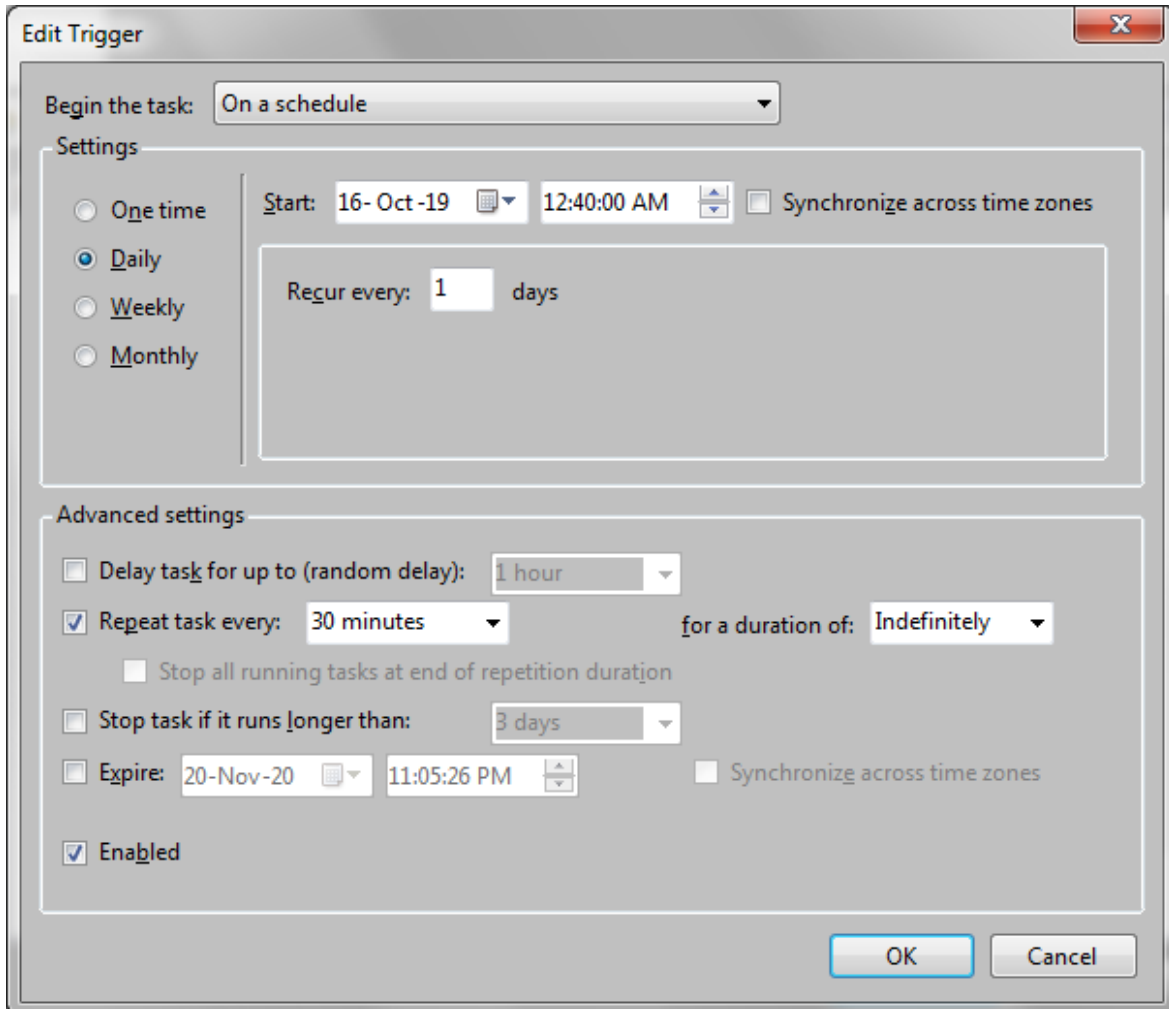


Figure 21 – Edit trigger interface

The equivalent schedule in a cron job scheduler is shown at the figure below.

Common Settings

Twice Per Hour(0,30 * * * *)

Minute: 0,30 Once Per Thirty Minutes(0,30)

Hour: * Every Hour (*)

Day: * Every Day (*)

Month: * Every Month (*)

Weekday: * Every Day (*)

Command:

Add New Cron Job

Current Cron Jobs

Minute	Hour	Day	Month	Weekday	Command
0,30	*	*	*	*	php -q /home/public_html/update.php

Figure 22 – Cron job scheduler

Now let's analyze the code that composes those two files. To begin with the index file, as mentioned before, the first thing that happens is the cURL call to get the data with the corresponding options and then the response's transformation from raw text to data objects which is shown in figure 15.

The next step is to iterate through every single object and store its data to the database.

```

foreach ($xml as $element) {
    // Populate the companies
    $Companies = "INSERT INTO companies (id, name) VALUES (?, ?)";
    $stComp = $pdo->prepare($Companies);
    try {
        if($stComp->execute([$element->company->code, $element->company->name])) {
            $totalCompanies += 1;
        }
    } catch(PDOException $e) {
        print "Error!: " . $e->getMessage() . "<br/>";
    }
    // Populate the mapping table
    if($element->price != 0) { // If price has 0 value then don't insert
        $stat_prod_map = "INSERT INTO station_product_map (station_id, product_id, price, last_edit) VALUES (?, ?, ?, ?)";
        $stMap = $pdo->prepare($stat_prod_map);
        $epoch = substr($element->timestamp, 0, 10);
        $dt = new DateTime("@$epoch"); // convert UNIX timestamp to PHP DateTime
        $last_edited_on = $dt->format('Y-m-d H:i:s');
        try {
            if($stMap->execute([$element->station, $element->product->code, $element->price, $last_edited_on])) {
                $totalPrices += 1;
            }
        } catch(PDOException $e) {
            print "Error!: " . $e->getMessage() . "<br/>";
        }
    }
}
}

```

Figure 23 – Iteration through objects

In each iteration, the script stores the information for the Regions, the Stations, the Companies, the Products and finally the mapping of Products to Stations with the prices and the latest timestamp of when these got updated by the Ministry. Figure 23 shows a small part of the script, displaying the process of storing the information for the Companies and the mapping Station - Product - Price.

Firstly, the SQL Insert queries are written and stored to PHP variables and then these queries get prepared to PDO objects. The execution happens in TRY – CATCH statements to avoid the interruption of the code's execution for data that are already stored to the database, like inserting the same station or product twice. Storing the same element twice would have caused an SQL error and as well, it would interrupt the execution and that is because the tables in the database have primary and unique keys which prevent duplicates, but this is going to be further explained later.

Also, the data returned from the cURL call can contain prices with values equal with 0 and therefore, those have been excluded from getting stored to the database by using an IF statement as shown in the figure above. The reason for those getting excluded is obvious. It is irrelevant and pointless to show these results to the users.

Throughout the whole process of iterating and storing the data, counters are kept for each element that got successfully inserted in the database and when the whole iteration process is done, the total numbers of the items stored are displayed to the browser's screen.

The initialization of the database has finished.

Total Regions: 1538
Total Stations: 4560
Total Companies: 21
Total Products: 207
Total Prices Mapped: 21452

Figure 24 – Total results of elements stored to the database

Continuing with the update file, the steps to fetch the data via a cURL call and transform them into data objects are the same. The difference here is that the prices of the gas stations' products need to be updated and also store new elements if there are any. These new elements can be anything, from a Greece's Region that hadn't been stored at the database initialization phase or new Stations or Companies or Products. For this reason, there are several checking functions and if statements to check whether the current object that is being processed at the current iteration is already stored or not. These checks happen consecutively for each object. Also, every event that is happening during the whole process, is being logged at a logger file.

```
foreach ($xml as $element) {
    // ----- Check for changes ----- //

    // Check if there are data for current station and the current product in the database
    $chkIfEdited = $pdo->prepare('SELECT * FROM station_product_map WHERE station_id = ? AND product_id = ?'); // Select
    current station and current product
    $chkIfEdited->execute([$element->station, $element->product->code]);
    $editResult = $chkIfEdited->fetch(PDO::FETCH_ASSOC);

    if(!$editResult) { // If no results found at all then insert the new entry in mapping table
        if($element->price != 0) { // Insert the new entry only if the price is not 0
            $currentFileContents = file_get_contents($loggerFile); // Get current logged content
```

Figure 25 – Code sample of update file

Some critical cases that are worth mentioning here are the following:

1. The object that is being processed is not stored as probably it was not existing during the process of initializing the database and therefore has to be stored but its price is 0. In this case the insertion is skipped and the process goes on to the next object.
2. The object that is being processed is already stored with a normal non 0 price and the new updated price is as well non 0. In that case the price's value and timestamp get updated for the existing entry in the database.
3. The object that is being processed is already stored with a normal non 0 price and the new updated price is 0. In that case, the existing outdated entry gets removed from the mapping of Station – Product table. The database's structure and the tables that consist it will be further explained on the next chapter.
4. The object that is being processed is already stored with a normal non 0 price and the new updated price is exactly the same as the one stored. In this case the script skips that object and continues to the next one.

During the whole update process, the script logs every event in the logger file and also outputs everything on the browser in case the update file is ran manually through a browser. Otherwise, if the update file is being ran from the cron job, then the logging happens to the file only.

```
Update found for station 21335 on product 1002. Previous price was 0.769 on 2019-10-30 08:28:51 and now is 0.745 on 2019-11-20 09:16:04.
New Station added with id: 21339
New entry found for station 21339 on product 72. Price is 1.895 on 2019-11-16 07:42:49.
New entry found for station 21339 on product 74. Price is 1.428 on 2019-11-16 07:43:28.
New entry found for station 21339 on product 75. Price is 1.08 on 2019-11-16 07:45:09.
New entry found for station 21339 on product 79. Price is 1.498 on 2019-11-16 07:44:26.
New entry found for station 21339 on product 1073. Price is 1.668 on 2019-11-16 07:40:57.
Update found for station 21343 on product 79. Previous price was 1.385 on 2019-11-05 07:34:12 and now is 1.395 on 2019-11-16 05:01:41.
Update found for station 21346 on product 23. Previous price was 1 on 2019-11-02 12:42:46 and now is 1.02 on 2019-11-18 09:14:10.
New entry found for station 21346 on product 936. Price is 1.05 on 2019-11-18 09:16:06.
Update found for station 21346 on product 937. Previous price was 1 on 2019-11-14 07:23:05 and now is 1.02 on 2019-11-18 09:14:19.
Update found for station 21346 on product 938. Previous price was 1 on 2019-11-14 07:18:14 and now is 1.02 on 2019-11-18 09:14:29.
New DD added with id: 16040700
New Station added with id: 21347
New entry found for station 21347 on product 51. Price is 1.638 on 2019-11-21 07:33:32.
New entry found for station 21347 on product 54. Price is 1.397 on 2019-11-21 07:35:29.
New entry found for station 21347 on product 55. Price is 1.13 on 2019-11-21 07:36:35.
New entry found for station 21347 on product 59. Price is 1.467 on 2019-11-21 07:35:54.
Update found for station 21350 on product 11. Previous price was 1.545 on 2019-11-15 04:06:09 and now is 1.538 on 2019-11-21 03:55:43.
Update found for station 21350 on product 12. Previous price was 1.655 on 2019-11-15 04:06:23 and now is 1.648 on 2019-11-21 03:55:55.
Update found for station 21350 on product 13. Previous price was 1.895 on 2019-11-15 04:06:31 and now is 1.945 on 2019-11-21 03:56:04.
Update found for station 21350 on product 17. Previous price was 1.395 on 2019-11-15 04:06:59 and now is 1.305 on 2019-11-21 03:56:38.
Update found for station 21352 on product 23. Previous price was 1.034 on 2019-11-15 05:03:30 and now is 1.035 on 2019-11-21 04:43:39.
Update found for station 21352 on product 936. Previous price was 1.034 on 2019-11-15 05:03:46 and now is 1.035 on 2019-11-21 04:43:49.
Update found for station 21352 on product 937. Previous price was 1.034 on 2019-11-15 05:03:56 and now is 1.035 on 2019-11-21 04:43:58.
Update found for station 21352 on product 938. Previous price was 1.034 on 2019-11-15 05:04:04 and now is 1.035 on 2019-11-21 04:44:07.
Update found for station 21359 on product 17. Previous price was 1.425 on 2019-11-15 03:51:47 and now is 1.335 on 2019-11-21 03:55:06.
Update found for station 21361 on product 1241. Previous price was 0.747 on 2019-10-29 16:31:14 and now is 0.757 on 2019-11-18 09:05:48.
Update found for station 21367 on product 1041. Previous price was 0.739 on 2019-11-15 04:19:33 and now is 0.75 on 2019-11-21 04:30:18.
New Station added with id: 21369
New entry found for station 21369 on product 41. Price is 1.559 on 2019-11-16 14:38:32.
```

Figure 26 – Update.php file output on browser

3.2.2 Database Schema

In this part, the designing and structure of the database is going to be described in detail. First of all, by observing the elements that the cURL call returns, it was clear from the beginning that the database will be consisted of 4 tables. These tables are the Regions table which is named “dd” in the database to keep consistency and keep the same name as the one returned in the cURL response. The other 3 tables are the Companies, the Products and the Stations. In these tables, all the relevant information for the Regions, Companies, Products and Stations are being stored.

Apart from those 4 tables, there was a need to also include one more which was named “station_product_map”. In this table, a mapping of products to stations is being kept and for every entry there is the price and timestamp details. Therefore, this table is the biggest comparing to the rest having approximately 23.000 rows. To give some insights for the rest of the tables as well, the Stations table has about 4800 rows, where each row represents an actual station, the Products table has about 210 rows, the Companies table has 21 rows and the Regions table has about 1600 rows.

Apart from thinking of how and where the data will be stored, rules had also to be included and applied on these tables to prevent errors and duplicate values that would serve false information to the users. To achieve this, primary and unique keys to the tables were created. The primary key of the Stations table is the Station ID, for Products is the product ID, for Companies is the company ID and so on. The mapping table on the other hand, has a combined unique key consisted of 2 primary keys. These keys are the Station ID and Product ID primary keys.

The figure below shows a visual representation of the database and how each table connects to the other.

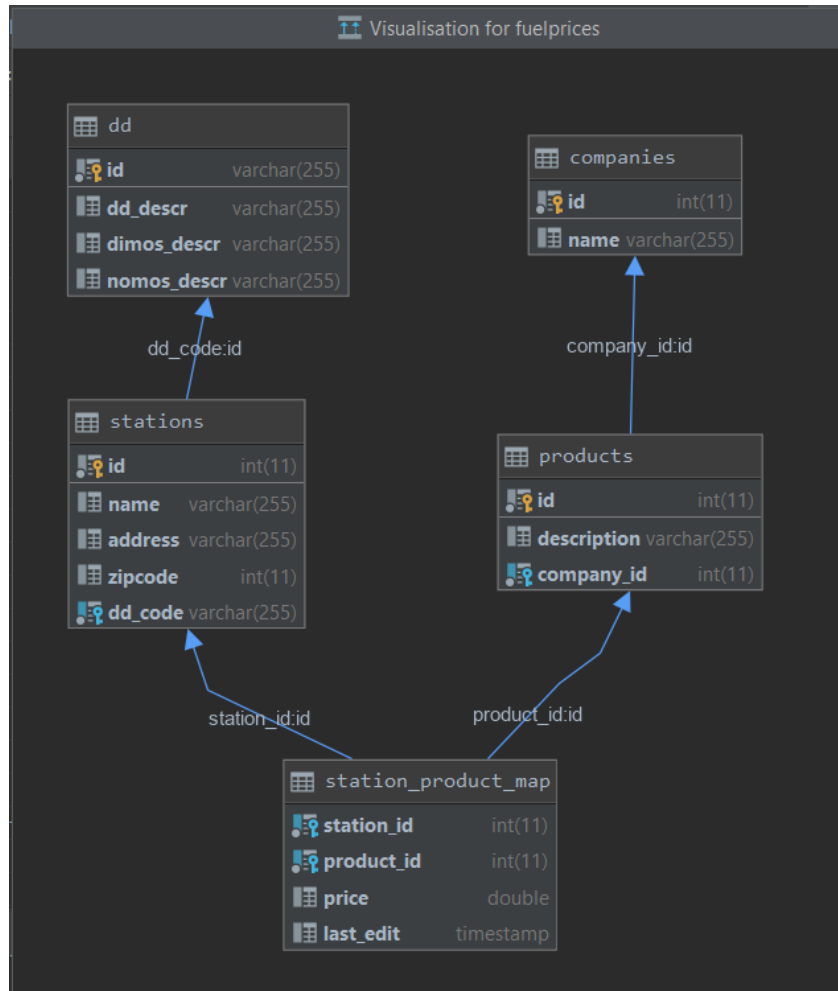


Figure 27 – Visual representation of the database schema

	id	dd_descr	dimos_descr	nomos_descr
1	01010100	Δ.Δ.Μεσολογγίου	ΔΗΜΟΣ ΙΕΡΑΣ ΠΟΛΗΣ ΜΕΣΟΛΟΓΓΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
2	01010200	Δ.Δ.Αγίου Γεωργίου	ΔΗΜΟΣ ΙΕΡΑΣ ΠΟΛΗΣ ΜΕΣΟΛΟΓΓΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
3	01010600	Δ.Δ.Ευνοχωρίου	ΔΗΜΟΣ ΙΕΡΑΣ ΠΟΛΗΣ ΜΕΣΟΛΟΓΓΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
4	01020100	Δ.Δ.Αγγελοκάστρου	ΔΗΜΟΣ ΑΓΓΕΛΟΚΑΣΤΡΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
5	01030100	Δ.Δ.Αγρινίου	ΔΗΜΟΣ ΑΓΡΙΝΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
6	01030200	Δ.Δ.Αγίου Κωνσταντίνου	ΔΗΜΟΣ ΑΓΡΙΝΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
7	01030400	Δ.Δ.Δοκιμίου	ΔΗΜΟΣ ΑΓΡΙΝΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
8	01030500	Δ.Δ.Καλυβίων	ΔΗΜΟΣ ΑΓΡΙΝΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
9	01030600	Δ.Δ.Καμαρούλας	ΔΗΜΟΣ ΑΓΡΙΝΙΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
10	01040100	Δ.Δ.Αιτωλικού	ΔΗΜΟΣ ΑΙΤΩΛΙΚΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
11	01040300	Δ.Δ.Σταμνάς	ΔΗΜΟΣ ΑΙΤΩΛΙΚΟΥ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
12	01050100	Δ.Δ.Κανδήλας	ΔΗΜΟΣ ΑΛΥΖΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
13	01050400	Δ.Δ.Μύτικα	ΔΗΜΟΣ ΑΛΥΖΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
14	01060100	Δ.Δ.Αμφιλοχίας	ΔΗΜΟΣ ΑΜΦΙΛΟΧΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
15	01060200	Δ.Δ.Αμπελακίου	ΔΗΜΟΣ ΑΜΦΙΛΟΧΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
16	01060500	Δ.Δ.Κεχρινιάς	ΔΗΜΟΣ ΑΜΦΙΛΟΧΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
17	01060600	Δ.Δ.Λουτρού	ΔΗΜΟΣ ΑΜΦΙΛΟΧΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ
18	01060900	Δ.Δ.Σπάρτου	ΔΗΜΟΣ ΑΜΦΙΛΟΧΙΑΣ	ΝΟΜΟΣ ΑΙΤΩΛΙΑΣ ΚΑΙ ΑΚΑΡΝΑΝΙΑΣ

Figure 28 – Screenshot of Regions (dd) table

	id	name
1	1	BP
2	2	SHELL
3	3	EKO
4	4	ΑΝΕΞΑΡΤΗΤΟ ΠΡΑΤΗΡΙΟ
5	5	ΑΙΓΑΙΟ (AEGEAN)
6	6	ΑΡΓΩ
7	7	ΑVIN
8	9	CYCLON
9	11	ΕΛΙΝΟΙΑ
10	13	ΕΤΕΚΑ
11	14	JETOIL
12	15	ΚΑΟΙΛ
13	16	ΚΜΟΙΛ
14	18	MEDOIL
15	19	REVOIL
16	20	SILKOIL
17	23	SHELL
18	24	ΤΡΙΑΙΝΑ
19	25	WIN
20	26	VALIN
21	27	BEESTATION J.V.

Figure 30 – Screenshot of Companies table

	id	description	company_id
1	4	Diesel Κίνησης	27
2	5	Diesel Θέρμανσης	27
3	6	BP Unleaded 95	1
4	7	BP Ultimate 100	1
5	8	BP Super LRP	1
6	9	BP Super Diesel	1
7	10	BP Heat	1
8	11	Shell Αμόλυβδη Ενισχυμένη 95 οκτανίων	2
9	12	Shell Αμόλυβδη 95 V-Power	2
10	13	Shell Αμόλυβδη 100 V-Power Racing	2
11	14	Shell Νέα Σούπερ	2
12	15	Shell Πετρέλαιο Κίνησης Extra	2
13	16	Shell Μείγμα Κίνησης Βιοντίζελ Extra	2
14	17	Shell V-Power Diesel	2
15	18	Shell Πετρέλαιο Θέρμανσης	2
16	19	EKO Kinitron Unleaded 95 Plus	3
17	20	EKO Kinitron Unleaded 100 Speed	3

Figure 29 – Screenshot of Products table

	id	name	address	zipcode	dd_code
1	2	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΛΕΝΟΡΜΑΝ	ΛΕΝΟΡΜΑΝ 187-191 ΑΘΗΝΑ	10442	A1010400
2	3	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΡΕΤΖΙΚΙΟΥ	4,5 ΧΛΜ. ΘΕΣ/ΝΙΚΗΣ-ΑΣΒΕΣΤΟΧΩΡΙΟΥ, ΚΟΙΝΟΤΗΤΑ ΠΕΥΚΩΝ - ΘΕΣ/ΝΙΚΗ	57010	54420400
3	4	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΒΡΙΑΗΣΣΙΩΝ 2	ΑΝΑΠΑΥΣΕΩΣ 41, ΒΡΙΑΗΣΣΙΑ	15235	A1100100
4	5	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΗΡΑΚΛΕΙΟΥ 2	ΧΡΙΣΤΟΜΙΧΑΛΗ ΞΥΛΟΥΡΗ 74, ΗΡΑΚΛΕΙΟ	71500	91010100
5	6	ΕΡΜΗΣ	ΑΜΕΡ. ΕΡΥΘΡΟΥ ΣΤΑΥΡΟΥ 212, ΚΑΒΑΛΑ	65201	55010100
6	7	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΠΕΡΑΙΑΣ ΘΕΣ/ΝΙΚΗΣ	ΘΕΣ/ΝΙΚΗΣ 15, ΠΕΡΑΙΑ ΘΕΣ/ΝΙΚΗΣ	57019	54170100
7	8	ΜΥΡΤΕΑ ΑΕ ΥΠ/ΜΑ ΗΡΑΚΛΕΙΟΥ 1	ΙΚΑΡΟΥ 42, ΗΡΑΚΛΕΙΟ	71306	91010100
8	9	ΜΥΡΤΕΑ ΑΕ ΥΠ/ΜΑ ΛΑΜΙΑΣ	3 ΧΛΜ.ΛΑΜΙΑΣ-ΑΘΗΝΑΣ, ΛΑΜΙΑ	35300	06011100
9	10	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΛΙΜΟΥ	ΛΕΩΦ. ΠΟΣΕΙΔΩΝΟΣ (ΕΟΤ ΑΛΙΜΟΥ), ΑΛΙΜΟΣ	17562	A1340100
10	12	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ 3ΧΛ ΕΔΕΣΣΑΣ-ΘΕΣΣ	3 ΧΛΜ ΕΔΕΣΣΑΣ-ΘΕΣ/ΝΙΚΗΣ, ΕΔΕΣΣΑ	58200	59010800
11	13	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΘΗΝΑΣ 43 ΚΑΒΟΥΡΙ	ΑΘΗΝΑΣ 43, ΚΑΒΟΥΡΙ	16673	A2080100
12	14	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΙΓΑΛΕΟ	Λ.ΚΗΦΙΣΙΟΥ 90, ΑΙΓΑΛΕΟ	12241	A1060100
13	15	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΛΕΞΑΝΔΡΟΥΠΟΛΗΣ	ΛΕΩΦ ΔΗΜΟΚΡΑΤΙΑΣ 46, ΑΛΕΞ/ΠΟΛΗ	68100	71010100
14	16	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΛΦΕΙΟΥ 36	ΑΛΦΕΙΟΥ 36, ΠΥΡΓΟΣ	27100	14010100
15	17	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΜΑΡΥΝΘΟΥ	ΑΜΑΡΥΝΘΟΣ, ΑΜΑΡΥΝΘΟΣ ΕΥΒΟΙΑΣ	34006	04030100
16	18	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΡΓΟΥΣ ΝΑΥΠΛΙΟΥ 44	ΑΡΓΟΥΣ ΝΑΥΠΛΙΟΥ 44	21	11010100
17	19	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΡΓΥΡΟΥΠΟΛΕΩΣ 89	ΑΡΓΥΡΟΥΠΟΛΕΩΣ 89,ΑΡΓΥΡΟΥΠΟΛΗ	16451	A1090100
18	20	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΡΜΕΝΟΙ ΡΕΘΥΜΝΟΥ	ΑΡΜΕΝΟΙ, ΡΕΘΥΜΝΟ	74100	93010200
19	21	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΣΠΡΟΥΡΓΟΥ	17 ΧΛΜ. ΕΟ ΑΘΗΝΩΝ-ΚΟΡΙΝΘΟΥ, ΑΣΠΡΟΥΡΓΟΣ	19300	A3030100
20	23	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠ/ΜΑ ΑΦΙΔΑΝΩΝ	27 ΧΛΜ ΑΘΗΝΩΝ-ΛΑΜΙΑΣ, ΑΦΙΔΑΝΑΙ	19014	A2650100
21	26	ΕΡΜΗΣ ΑΕΜΕΕ ΥΠΟΚ/ΜΑ ΒΟΛΟΥ	ΛΑΡΙΣΗΣ 212, ΒΟΛΟΣ	38334	43010100

Figure 31 – Screenshot of Stations table

	station_id	product_id	price	last_edit
1	2	11	1.558	2019-11-15 04:43:06
2	2	12	1.668	2019-11-15 04:43:15
3	2	13	1.945	2019-11-21 06:59:28
4	2	16	1.318	2019-11-15 04:43:36
5	2	17	1.32	2019-11-21 07:04:16
6	3	11	1.568	2019-11-15 06:25:23
7	3	12	1.678	2019-11-15 06:25:39
8	3	13	1.945	2019-11-20 22:03:13
9	3	16	1.338	2019-11-15 06:27:51
10	3	17	1.348	2019-11-20 22:03:34
11	4	11	1.518	2019-11-21 05:35:44
12	4	12	1.628	2019-11-21 05:35:50
13	4	13	1.945	2019-11-21 05:35:56
14	4	16	1.298	2019-11-15 05:59:00
15	4	17	1.308	2019-11-21 05:36:09
16	5	11	1.598	2019-11-15 04:08:37
17	5	12	1.708	2019-11-15 04:08:54
18	5	13	1.945	2019-11-21 04:16:42
19	5	15	1.375	2019-11-15 04:09:11
20	5	16	1.375	2019-11-15 04:09:19
21	5	17	1.385	2019-11-21 04:17:17

Figure 32 – Screenshot of Mapping table

3.2.3 PHP Rest API

The acronym API stands for Application Programming Interface which is an interface or a communication protocol or sometimes also described as a “contract” between a client and a server. The reason for the API implementation was for the Chatbot to be able to get data from the database. In other words, an API is a URL end point, from which a client can request data, similar to what has been done at the cURL call process where the gas stations and prices information is requested from the Ministry’s URL end point.

The API has been implemented in PHP. More or less, the API is PHP classes for reading and accessing the database and finally returning the requested information. Since an API is an end point, this means that anyone knowing of its existence, can access it and use it. Therefore, Basic Auth using username and password has been included to secure the API and allow access to authorized clients only. Otherwise, the back end would be at risk of attacks or misuse.

The API is consisted of two PHP classes. The first one is the Prices class which is responsible for reading the database based on the queries coming from the client and returning the responses back to the clients. The second class is the Database which is responsible for managing the connection to the database, by either opening or closing a connection. Finally there is an end point file which is called “read” and this is the one through which the clients pass their queries to the API.

The figure below shows a call to the API. In this call, there is a request to the end point’s URL “http://local/fuelprices/rest_api/api/prices/read.php” for the fuel prices of gas stations in Greece’s region named “Νομός Σερρών” for the product “Αμόλυβδη 95”. The response returned contains 15 gas stations and those are listed by the lowest price first to the highest at the end.

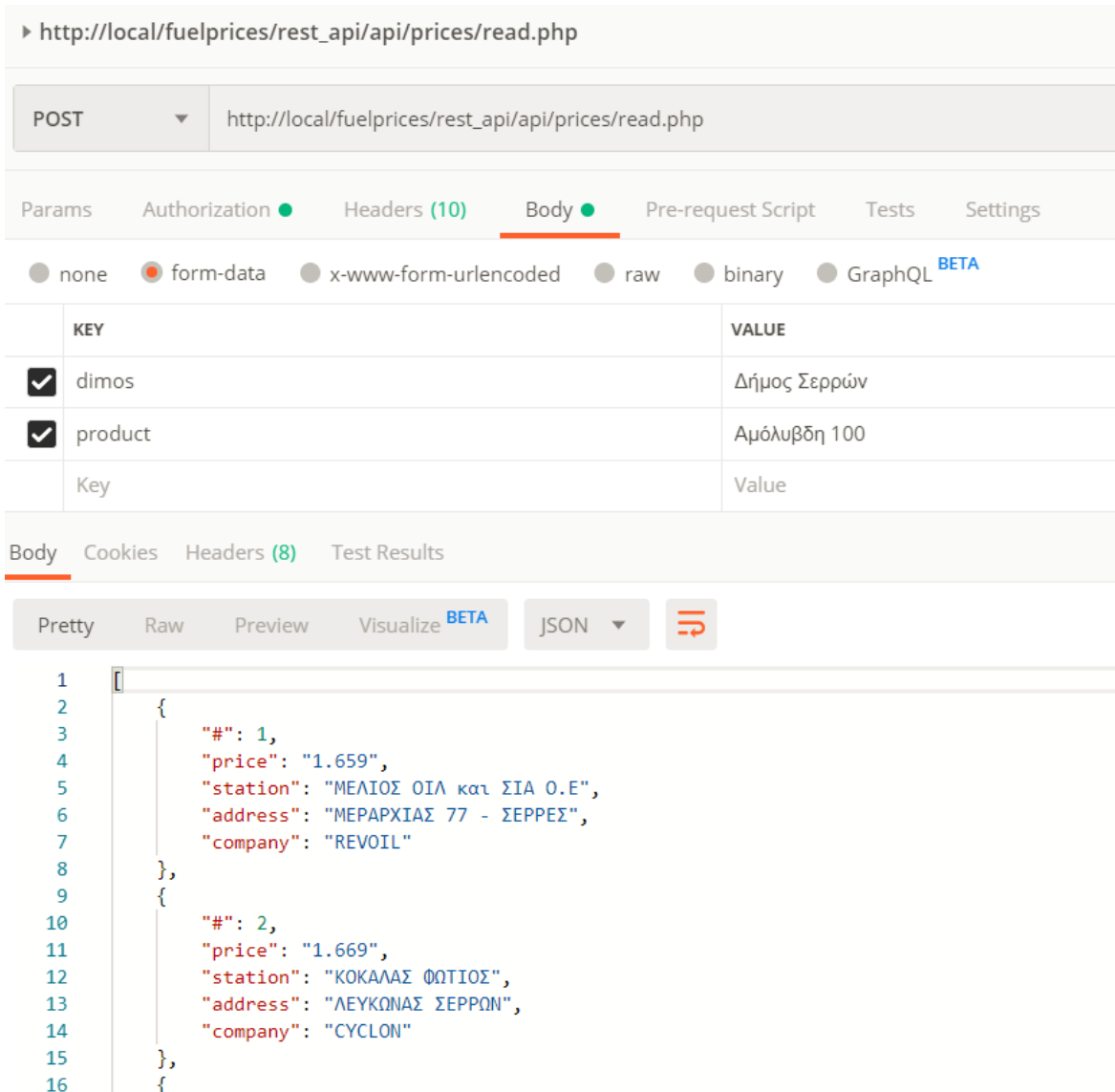


Figure 33 – Screenshot of Postman after call to the API endpoint

Notice that the API call is a POST type and therefore the parameters are being passed in the body of the message and not through the URL, as it would be the case for GET type requests.

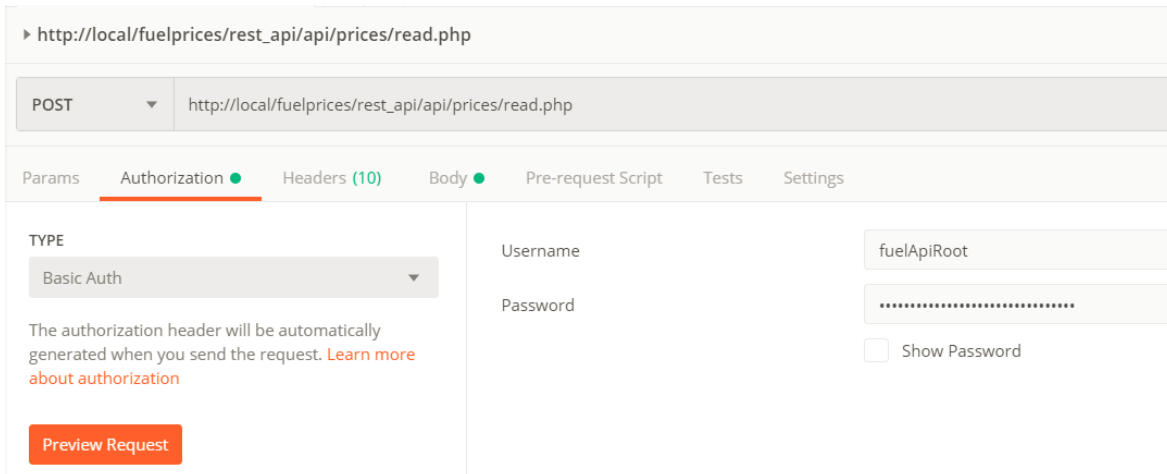


Figure 34 – Basic Authorization type to perform the call

The figure above shows the credentials username and password that have been set to perform the call.

To create this authorization mechanism, two files have been included in the API's root folder. These files are an `.htaccess` and an `.htpasswd` file respectively.

3.2.4 The Chatbot prototype

Finally, in this last part of the implementation, all the things that have been done to implement the Chatbot prototype will be described.

After having created the whole aforementioned infrastructure, the thinking and designing of how the actual product will look like, takes place. In this case, the product is a Chatbot or better an Assistant Agent. The goal from the beginning was to create a service to feed accurate data to that assistant agent and give precise answers to users about fuel prices on selected locations and specific fuel products. Therefore, after having implemented the back-end service, Botsociety platform was then used to design the interaction flow between the user and the agent.

There may be several things that a user could say to request fuel prices information like for example "I need fuel" or "I'm out of fuel" or "Find me the closest gas stations" and further on, the agent could respond several things back to the user instead of simply giving direct answers. For example, the agent may ask first the location of the user and after the user responds back to the agent about their location, the agent may then ask about which fuel

product the user would like to find information for and so the user could again respond back to the agent by saying “Unleaded 95”.

So, simply like that, there may actually be lots of conversational paths which the users may follow by interacting with the assistant agent until they get the information they need. Therefore, some of those paths were designed in Botsociety to have a better view of how the interaction may happen and which paths connect to each other and where.

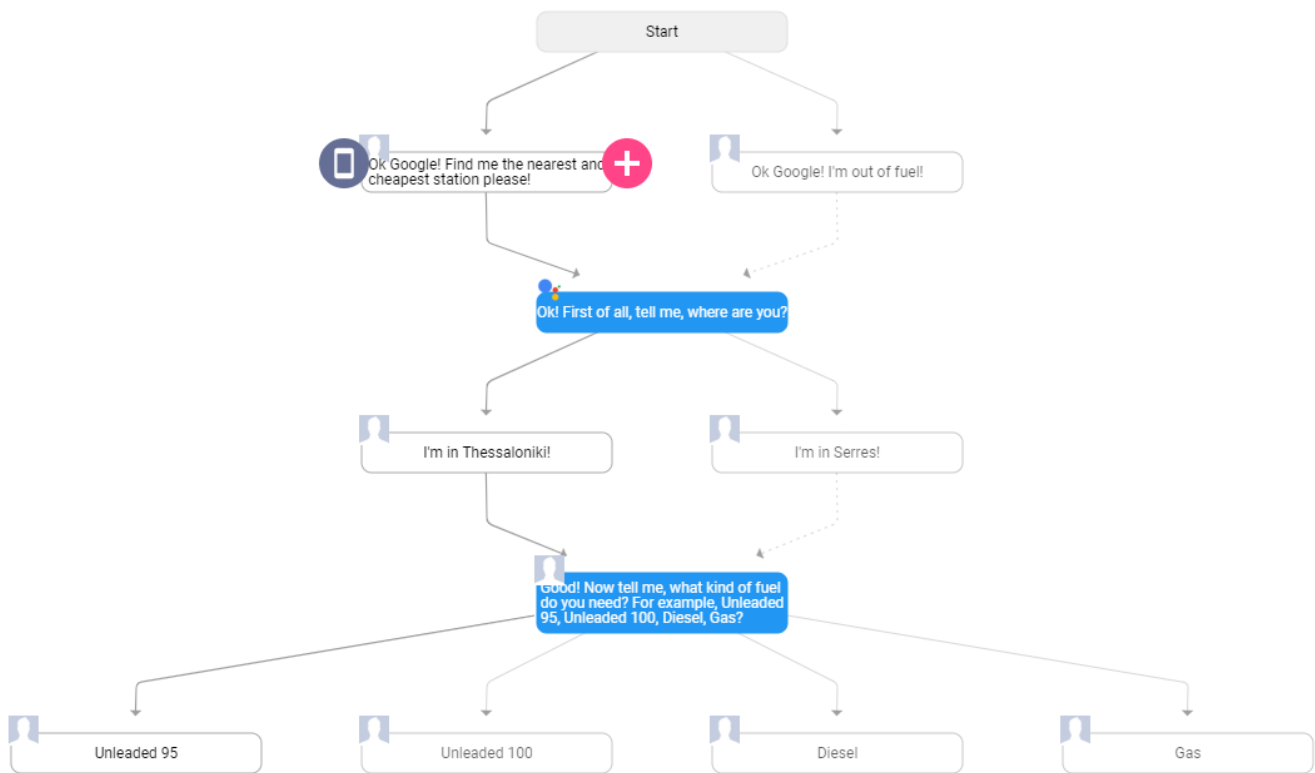


Figure 35 – User - agent interaction flow diagram exported by Botsociety tool

The figure above shows some path examples that the users could follow to get the fuel prices they need. Start, represents the initiation of the session by the users between them and the agent. The initiation may happen by saying “Ok Google” on an android device. Then, the flow diagram shows 2 paths that the users could follow which afterwards connect to the same node again, since at that point, it is important for the agent to know the user’s location. At that point, a different implementation was considered, where the agent gets the users’ location by their device’s GPS services but for that to happen, the users have to grant to the agent access to their device’s location permissions. The granting of access happens only once but there may be users who don’t want to grant access to the agent and therefore there must be an alternative path for that case, the one that have already been implemented and in which the agent asks from the users to share their location. Then, the users respond back to

the agent of their location and the agent asks back which type of fuel they would like to find prices information about. Finally, the users answer back the fuel product that they desire and they get the information they need.

To note here, there could be countless paths in the flow diagram but for the purposes of this dissertation and to also have a clean and direct flow diagram, it has been kept simple to what the figure shows.

After finally having created a complete flow that suit the needs of the implementation, the option to export this design to Dialogflow or Rasa is provided by Botsociety, which are both platforms for advanced development of chatbots. The Dialogflow platform was chosen which is a Google product and feels more familiar. The export is useful in general terms but there was a lot of work needed to be done to correct things that didn't happen as expected.

First of all, regarding the Dialogflow platform, for an assistance agent to function there are several things that need to be done. The most important thing is the configuration of the Intents. The intents are the intentions or in other words, the logical functions that the agent does. For example, the getting of the fuel prices is one intent. Getting the user's location could be a different intent and so on. The retrieval of fuel prices has been implemented in one intent only since it only serves one purpose and that is the retrieval of fuel prices by doing only one request. This of course could have been done in different ways as well by including more than one intents.

The correct configuration of an intent requires training phrases which will train the agent to understand the way that it should work and function.

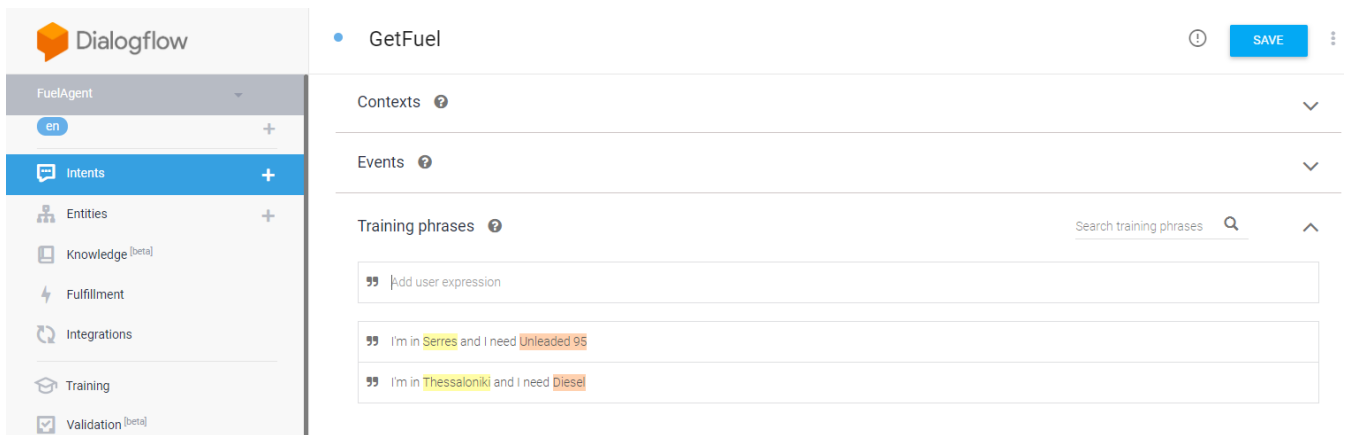


Figure 36 – Agent's main intent named GetFuel

Figure 36 shows the training phrases that were included in the agent's main intent which is named GetFuel. These training phrases have a form of "I'm in <location> and I need <product>". After adding only two phrases, the agent understood that at the positions of the yellow and orange words it will expect a keyword for each one of the colors respectively.

The keywords had to be marked and given specific types, to make the agent understand what kind of keywords these will be, which in this case, both are @sys.any. This converted these keywords from normal words to parameters, through which data will be passed to the API call that will happen after the user’s question to the agent. The @sys.any type means that this parameter is of any type and should be correctly configured by the developer. In this case, the yellow keyword was marked to be the location and the orange keyword to be the product.

Action and parameters

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	locat	@sys.any	\$locat	<input type="checkbox"/>
<input type="checkbox"/>	prod	@sys.any	\$prod	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

[+ New parameter](#)

Figure 37 – Intent’s parameters location and product

To give a brief example, let’s say the user initiates a session with the Assistant agent by calling it with the phrase “Ok Google! I’m in Thessaloniki and I need fuel” or more precisely “Ok Google! I’m in Thessaloniki and I need diesel” or “... and I need unleaded 95” or “... and I need gas”. The agent then would immediately make a call to the RESTful API, that was implemented on the third part, by passing the parameters \$locat and \$prod which in this case may be Thessaloniki and Unleaded 95 respectively. The API is programmed to be able to receive requests at any time, find for answers in the database that has also been implemented in the second part, retrieve these answers and return them back to the Assistant Agent and then the agent will finally either display these answers to the user’s screen or will pronounce them.

Continuing, after having created some training phrases and also having specified parameters to pass data to the API, the configuration of the connection between the agent and the API must be done by specifying the API’s URL end point and handling the data coming from the parameters correctly. To do this, Dialogflow has a different section called Fulfillment in which there are two options which are to either enable a Webhook where a URL end point has to be set, or to enable the Inline Editor which is a more advanced way to configure the connection to an external URL end point by writing custom code in Node.js.

For the purpose of this dissertation, the Inline editor was chosen to configure the connection and make the whole setup cover the needs.

The screenshot displays the Dialogflow Fulfillment configuration page. On the left is a sidebar with the 'Fulfillment' option selected. The main content area is titled 'Fulfillment' and includes a 'Webhook' section which is currently disabled. Below it is the 'Inline Editor', which is enabled and contains a code editor. The code in the editor is as follows:

```
30
31 return axios.post('https://fuelpricebot.com/rest_api/api/prices/read.php', {
32     dimos: locat,
33     product: prod
34 }).then((response) => {
35     response.data.map(dataObject => {
36         agent.add('Address: '+dataObject.address+' | Price: € '+dataObject.price);
37     });
38 }, (error) => {
39     console.log(error);
40 });
41 }
42
43 // Run the proper function handler based on the matched Dialogflow intent name
44 let intentMap = new Map();
```

At the bottom of the code editor, there is a 'DEPLOY' button and a note indicating the last deployment was on 11/25/2019 at 18:57.

Figure 38 – Fulfillment configuration to perform the call to our API

The figure above shows the Javascript code written in a Node.JS platform. Axios JS HTTP client was used to perform the call to the RESTful API. As shown on lines 32 and 33, the parameters \$locat and \$prod are being passed as data in order to get the correct answers back and on line 31 the API's URL is being specified.

Now that everything has been set up correctly, it's time out to test the whole implementation. The scraper is deployed on a server and the database has been created as well, the initialization of the database has been made and the updating is happening every 5 minutes via cron jobs. The API is also ready to respond to any request and the Assistant Agent is ready to be used.

Simulator

The screenshot shows a chat interface for a bot named "Fuel Price Bot Test". The interface includes a title bar with a close button (X) and the bot's name. The chat history consists of several messages:

- A button labeled "Talk to Fuel Price Bot Test".
- A system message (speaker icon) stating: "Here's the test version of Fuel Price Bot Test. Hello! How can I help you?"
- A user message (grey bubble) stating: "I'm in Thessaloniki and I need unleaded 95".
- A system message (speaker icon) providing two addresses and prices: "Address: ΔΕΛΦΩΝ 100 ΘΕΣΣΑΛΟΝΙΚΗ | Price: € 1.498" and "Address: 10xlmΘΕΣ/ΝΙΚΗΣ-ΜΟΥΔΑΝΙΩΝ | Price: € 1.525".
- A user message (grey bubble) stating: "I'm in Serres and I need gas" with a keyboard icon.
- A system message (speaker icon) providing two addresses and prices: "Address: 1 ΧΙΛ ΣΕΡΡΩΝ ΝΙΓΡΙΤΑΣ | Price: € 0.769" and "Address: 1οΧΛΜ ΣΕΡΡΩΝ-ΝΕΟΧΩΡΙΟΥ | Price: € 0.775".

At the bottom of the chat, there is a "Suggested input" section with a "cancel" button. Below that is an "Input" field containing the text "Try typing or saying 'Talk to Fuel Price Bot Test'." with a microphone icon on the right.

Figure 39 – Final stage Usage of the whole implementation

The figure above shows the successful functioning of the whole implementation. Google's Actions Console, provides a test section where tests can be performed on the agent that has been implemented in Dialogflow. As shown above, a request for the cheapest gas stations in Thessaloniki for Unleaded 95 is done in the first call and the cheapest stations in Serres for Gas in the second call.

Chapter 4: Discussion and results

In this chapter of the dissertation, it is going to be discussed what the final feeling is, after successfully completing the whole implementation, what goals that had been set in the beginning have been achieved and if there were any tackles or problems that came up in the meanwhile.

4.1 Goals and implementation targets

The goals that had been set, were very specific from the beginning and it was already known beforehand what had to be implemented. These goals generally speaking were, to create an assistant agent that will be spoken to and will return results for the cheapest and closest gas stations according to the user's location. Specifically speaking, these goals were the creation of a Scraper that will scrape the data from the Ministry's of Development and Competitiveness end point, the creation of a Database that will be the storage, where the scraper will save and update the scraped data, the creation of an API service to retrieve data from the database externally from the Assistant Agent and finally the Assistant Agent or in other words, the Chatbot that will respond to users' queries.

After setting these four major goals, there were tasks that needed to be done to start achieving all of them one by one, step by step. To achieve the first goal, first, there was a need to communicate with the Ministry and let them know that for the implementation of this dissertation, access to their data was required. After providing the access that was needed, the implementation process of scraper began, which after some good and focused thinking, scoping and designing, has been implemented successfully and was running stable even from the first tests.

For the second goal to be achieved, the most critical requirement was concentrated thinking and precise designing of the database, the tables, the rules and the restrictions between the tables, to successfully store the data avoiding cases of duplicate inputs or false and corrupted data. As a result, by following this tactic, the database was created successfully and was functioning perfectly even from the first initialization.

The third goal was the API creation which was an easy task as it simply required some PHP coding to retrieve data from the database and a correct structure of files and folders to compose the RESTful API. This goal was achieved successfully as well by doing some online research to find out what the best practices of building an API in PHP are.

The last and final goal was the creation of the Chatbot. Thankfully, Google supports Dialogflow with a lot of documentation that assisted the whole implementation process of the Chatbot prototype and also to configure it and perform the connection between it and the API successfully without any issues.

Finally, all of the goals were successfully achieved and the final outcome is a great software that could also serve other kind of requests as well and not only price retrieval of gas stations.

4.2 Challenges and problems tackled

The first and critical problem that had to be tackled was even before getting into the implementation phase of writing the actual code. This problem occurred by the Ministry's website from which data had to be scraped and collected. The problem was a change that had been done in its security structure and was not open to scraping by third parties. This was very critical since the whole implementation was based on the data that would be gathered by their website through our scraper.

What had to be done then was to communicate with their support and let them know about the needs to successfully complete this dissertation. This issue held the implementation process back for 2 to 3 weeks but thankfully, they granted the access that was needed, bypassing passwords and authentication measures. Therefore, it was possible to start testing and building the scraper and finally continue with the implementation.

Another challenge that came up was the need of a live server. For the API to be available at any time, there was a requirement of a live webserver on which, the scraper's files, the database and the API's files were going to be hosted and serve answers to users' calls. Thankfully, the accompanying supervisor, Dr. Stergios Tegos and their company named Enchanted, provided the server that was needed to continue with the implementation.

Rather than these two critical problems that were directly affecting the implementation process, there weren't any other issues to slow things down.

4.3 Limitations of current work

The only limitation was time. The short period of 6 months available to complete this dissertation was enough to just create a Chatbot prototype with only the basic functions rather than an intelligent agent being able to handle all types of requests and users' unquities. For that reason, the Chatbot is limited simply to understand phrases like "I'm in -location- and I need -product-". Anything else rather than that would just not work and no

results would be responded back to the users. Having said that, the chatbot is not clever enough to use the user's device location from the GPS sensor and therefore the user must also specify their location to the agent.

4.4 Future research

Since this Chatbot is limited to basic functions only, there is a lot of future research and implementation that could be done. To begin with the most basic stuff, it can be further trained to understand many more phrases, like for example "I'm out of fuel" or "Get me to a fuel pump" and other similar. It can also be smarter and locate the user by the GPS sensor as has been already mentioned. It can also store the users' product preference by the first query that the users would make, having as a result a really clever agent that would already know the location and product type and it would only need the initial phrase to trigger and activate it.

Not only that but, this whole implementation could also support other kind of information retrieval as well and not only fuel prices. For example, the exact same implementation, could serve information about nearby pharmacies that are open even at midnight hours. There may be many more scenarios as well.

Chapter 5: Conclusions

Concluding, having fully went through the whole implementation of building a Chatbot, from the literature review to the final deliverable, what has been realized is that there's a tremendous momentum with which Chatbots are growing. Chatbots start to exist everywhere, from simple websites, to mobile and wearable devices, to cars, to telephone centers and the list goes on. Chatbots will probably replace most of the apps that people use and have been using since the first smartphones appeared. An app always require taps and more time consuming tasks to get what you need where Chatbots not, you simply have to speak to them.

These are all natural beneficial outcomes due to the artificial intelligence's growth. The number of new features and applications that surfaced with the AI's growth is countless. Assistance agents may become a vital service in the future and may also replace lots of actual job positions that are currently occupied by humans.

Bibliography

- [1] Irfan Ahmad, How Much Data Is Generated Every Minute?, <https://www.socialmediatoday.com/news/how-much-data-is-generated-every-minute-infographic-1/525692/>, 2018.
- [2] Jizhou Huang, Ming Zhou, Dan Yang, Extracting Chatbot Knowledge from Online Discussion Forums, IJCAI, 2007.
- [3] Abdul-Kader, Sameera A., John Woods, Survey on Chatbot Design Techniques in Speech Conversation Systems, International Journal of Advanced Computer Science and Applications, 2015.
- [4] Ron Artstein, Laure Vieu, Decalog 2007: Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, pages 83–90. Trento, Italy, 2007.
- [5] Bayan Abu Shawar, Eric Atwell, Different measurements metrics to evaluate a chatbot system, pages 89-96, NAACL-HLT, Rochester, NY, 2007.
- [6] Bayan Abu Shawar, Eric Atwell, Chatbots: Are they Really Useful?, LDV-Forum, 2007.
- [7] S. Quarteroni, S. Manandhar, Designing an interactive open-domain question answering system, Natural Language Engineering, Cambridge University Press, 2009.
- [8] Marie-Claire Jenkins, Richard Churchill, Stephen Cox, Dan Smith, Analysis of User Interaction with Service Oriented Chatbot Systems, HCI, 2007.
- [9] Menal Dahiya, A Tool of Conversation: Chatbot, International Journal of Computer Sciences and Engineering, 2017.
- [10] Adnan Rehan, 9 Best Chatbot Development Frameworks to Build Powerful Bots, <https://geekflare.com/chatbot-development-frameworks/>, 2019.
- [11] Dan Jurafsky, Conversational Agents, <https://web.stanford.edu/class/cs124/lec/chatbot.pdf?fbclid=IwAR2Wbadl1KwiJNBRsJo1-RivdXAt6hAq44nw375HrUqmuOHsv3QL-wyrRjk>, 2019.
- [12] Discoverbot, Your Guide to Bot-Building Frameworks, <https://discover.bot/bot-talk/guide-to-bot-frameworks/>, 2019
- [13] KPMG, The rise of Chatbots and Virtual Assistants, <https://smarttech.kpmg.nl/hubfs/Downloads/Digital%20Advisor/digital-advisor-whitepaper-rise-of-chatbots-virtual-assistants.pdf>, 2019.

- [14] Anush Fernandes, The Best Open Source Chatbot Platforms in 2019, <https://blog.verloop.io/the-best-open-source-chatbot-platforms-in-2019/>, 2019.
- [15] Wikipedia, cURL, <https://en.wikipedia.org/wiki/CURL>, 2019.
- [16] NPM, axios, <https://www.npmjs.com/package/axios>, 2019.
- [17] Wikipedia, Dialogflow, <https://en.wikipedia.org/wiki/Dialogflow>, 2019.