



INTERNATIONAL
HELLENIC
UNIVERSITY

Chatbots for Greek/EU Public Services

Karamitsos Ioannis

SID: 330610005

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile & Web Computing

MARCH 2019

THESSALONIKI – GREECE



INTERNATIONAL
HELLENIC
UNIVERSITY

Chatbots for Greek/EU Public Services

Karamitsos Ioannis

SID: 330610005

Supervisor:

Prof. Magnisalis Ioannis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile & Web Computing

MARCH 2019

THESSALONIKI – GREECE

Abstract

The advent of web applications that provide automated user assistance as well as the evolution in using conversational programs, has led to the development of new directions in online customer support.

By taking into consideration the fact that quick and flexible online assistance is nowadays crucial for engaging customers and after identifying the current gaps especially in public services, we decided to develop a conversational chatbot system based on public services for a greek web portal called ,diadikasies.gr, in order to help citizens find easily the desired service. Another critical objective of our thesis, was to annotate several public services from this web portal using a european standard called C.P.S.V (Core Public Service Vocabulary) and develop the chatbot system to retrieve these semantic data on real time in order to operate appropriately.

Karamitsos Ioannis

7/12/2018

Contents

Abstract	3
Contents	4
1 Introduction	6
1.1 Background	6
1.2 Objectives	7
1.3 How objectives were achieved	8
1.4 Outline of the dissertation	9
2 Literature review	10
2.1 Literature review objectives	10
2.2 Few words about chat bots	11
2.3 Research method that was applied	11
2.4 Digital libraries and records	12
2.5 Chatbots	13
2.5.1 Why are chatbots used	14
2.5.2 Chatbots vs search engine machines	14
2.5.3 Ideal chatbots	15
2.5.4 Few concerns on chatbots	16
2.5.5 Technical approaches, languages and tools	17
2.5.6 Where are they used	20
2.6 Chatbots in public services	22
2.7 Three chatbot systems for public services are evaluated	23
2.8 CPSV and why to apply it	25
2.9 Need for chatbot systems in public services	26
2.9.1 Web portal : Diadikasies.gr	27
2.9.2 Ideal requirements of a chatbot	28
2.10 Towards Research questions	29
3 Design and Architecture	31
3.1 Design and Architecture	31
3.1.1 Requirements of the system	32
3.1.2 General flow of the system	33
3.1.3 Use cases of the system	34
3.2 Tools, frameworks and languages that were used	38
3.2.1 Framework to develop the system	38
3.2.2 Botpress framework and JAVASCRIPT	40
3.2.3 CPSV tool (to create semantic information for public services)	41
3.2.4 RDF to JSON converter (tool)	42

3.2.4.1 what is JSON format	42
4 Implementation	43
4.1 design pattern and restrictions (due to the framework)	44
4.1.1 Flow diagram (a tool of the framework)	44
4.2 Semantic Data	45
4.2.1 Description fields for each service	45
4.2.2 How semantic information is created and stored	47
4.3 Chatbot system (how the system works)	48
4.3.1 Information retrieval (semantics) by the system	48
4.3.2 How this semantic information is used by the system	49
4.3.3 Limitation of services according to user's preference	52
4.3.4 System provides the link of the service	53
4.4 The conversation interface and the possible interactions	54
4.5 Proof of concept	55
5 Results	74
5.1 Objectives achieved	74
5.2 Answers to the Research Questions	75
6 Conclusion and future work	77
6.1 Conclusion	77
6.2 Future work	79
7 References and Bibliography	80
8 APPENDIX	84
8.1 Flow Diagram of the system	84
8.2 Installation of botpress and built-ins	94
8.3 Implementation of methods	97
8.4 Creation and storage of semantics data	117
8.5 Images	121
8.5.1	121
8.5.2	122

1 Introduction

1.1 Background

As customers tend to be more demanding with new technologies in terms of quickness and flexibility, technological companies inevitably follow that trend and strive to fulfill their demands. One of the most crucial subjects that nowadays more and more technological experts are concerned about, is the appropriate information flow towards the customers for the product or service customers are interested in.

Companies have to maintain techniques that prioritize real-time online assistance, since the necessity for enhancing online customer experience has become essential. Hence, topics such as time-consumption and user friendliness are of vital importance in automated user assistance services.

Various search engines are available and are generally straightforward to be used by users, such as Google, Yahoo and so many others, however these engines are not focusing on specific data. Thus, usually users cannot find what they are searching for, and furthermore these engines are not human friendly as they just respond to queries and not communicate with users.

The evolution of technology engendered the advent of chatbots, a technological invention that firstly appeared, as a concept, during mid-60s. Subsequential research indicated that chatbots satisfy the desire of users to interact with systems and elicit the information required fast and easy. Since chatbots provide real time assistance, they tend to become a prerequisite feature in recent web applications. Indeed, the majority of highly appreciated web applications are nowadays equipped with chatbot systems in order to enhance user experience.

Except from commercial applications, chatbots are also used by public service agencies in several countries, as they can support online customer assistance without any limitation every day and every hour of the week, efficiently and transparently.

However, as concluded based on existing research, only few of the existing chatbots (based on public services) are operating as expected. Thus, the majority of citizens do not seem to appreciate chatbots and they avoid using them in order to seek for public services.

1.2 Objectives

After taking into consideration the gap between public services and automated customer assistance, especially chatbots, this dissertation focused on developing a chatbot system for public services using semantic data based on C.P.S.V (Core Public Service Vocabulary) standard, to help users on their quest for the desired services. Hence, the established objectives of this dissertation are summarized in the following:

1. The development of a chatbot system that uses semantic information of Public services to guide users find the service they seek for (in this case for a web portal called diadikasies.gr).
2. The use of semantics (semantic data) should be based on C.P.S.V standard.
3. The selection and usage of the appropriate chatbot framework for the implementation of the system.
4. The real time retrieval of the necessary information by the system.

1.3 How objectives were achieved

1st step (framework selection)

The focus of this dissertation is to connect the world of chatbots with that of semantics in the field of public services. As a first step, the appropriate chatbot framework for this purpose was selected. This selection was based on the inspection and comparison of several chatbot frameworks while aiming to distinguish the most flexible and easily customized to facilitate and fulfill all the requirements of the developed chatbot in the context of this dissertation.

2nd step (annotation of services based on C.P.S.V)

Additionally, the already existing Greek public services were enhanced with semantic data. This was performed for a sample of services described in a portal (diadikasies.gr) through wiki pages/documents. Then, this sample (e.g. five services) was enhanced with metadata according to CPSV standard by utilizing a tool provided by EU (PSDescriptionCreator).

3rd step (system retrieves metadata)

All information of the aforementioned documents, data and metadata was uploaded on a server in the format of a JSON file. This structure was required in order to use and handle data through the selected chatbot framework. That way, the implemented chatbot retrieves the necessary information and can guide users in finding the desired services. Last but not least, the system was developed to be capable of being integrated within web applications and especially for the web portal of diadikasies.gr.

However, this integration is out of the scope of current work and is left as a future possible implementation.

1.4 Outline of the dissertation

Below there is the summary of the chapters to follow:

The first chapter is the introduction of this dissertation which refers to chatbots as entities and to the fundamental goals of this thesis.

In the second chapter the literature review for the chatbots is presented along with several aspects upon them. Namely, comparison of chatbots with other assistive technologies, research for chatbots in public services, and finally the justification for the need of a chatbot system in public services throughout the examination other existing systems and pointing their weak features.

The third chapter illustrates the tools, the frameworks, the programming languages and the design of the developed system. This chapter refers mainly on the technologies utilized for the implementation of this chatbot.

The fourth chapter, is about the implementation of the system. First of all, it focuses on the restrictions on the development process as it is based on a chatbot framework. Then it presents the process of the implementation, from the annotation of the services based on C.P.S.V standard until the sequence of the questions that the system addresses to the users to guide them properly. It also presents the user

interfaces and describes how users interact with them. Finally, a proof of concept for the developed system is demonstrated.

In the fifth chapter the results of this dissertation are presented as inferred based on the literature research and the development of the specified chatbot.

The last chapter summarizes the conclusion and the contribution of this dissertation. Furthermore, the limitations of this dissertation are presented as well as directions for future research on chatbots.

2 Literature review

2.1 Literature review objectives

This chapter describes the progress and the history of chatbots among with a comparison between other assistive technologies. Additionally, the implementation of chatbot systems in various environments and contexts, especially in public services, is briefly presented. Moreover, the next section deals with the comparison of three existing chatbot systems that have been developed for public services, and discusses their advantages and disadvantages. Based on this comparison, the proposed solution, for the chatbot system, is presented.

2.2 Few words about chat bots

The main purpose that humanity has given so much emphasis on technology, is that it is rapidly evolving in order to make people's life easier. Not only in materialistic world, such as handheld devices and so, but also in software systems. In this way , more and more systems have been developed in order to be able to 'communicate' in a direct way with the users, so as to help them find all the information they want easily and quickly. However, it is not as simple as it may seems. Nowadays, there is a tremendous amount of information, and this makes it difficult for users to find exactly what they want to from a database system, so they tend to navigate through search engines, lists and sites to find it. Respectively, it is also difficult for machines to extract data from big datasets to serve users' needs. This, is also referred by Pavel Kucherbaev's, Achilleas Psyllidis's and Alessandro Bozzon's research [5]. As the open data movement is rising, most the of municipal datasets remain siloed, making it almost impossible for users to handle. There are some implemented solutions in the form of standalone applications or web portals, but the majority of the citizens are unwilling to use them or they are not apprised of them.

A technology called chatbot which, according to Bayan Abu Shawar's and Eric Atwell's (2007) study[1], is not so new as it started in 1960, is gradually used to help and guide users to 'chat' with the system in order to find the appropriate information. As this study says , at first , the main goal was to see if chatbots could make users believe that they were real humans. However, later on and specifically in this decade, companies thought that this, in relative terms, old technology could significantly emerge in the HCI community, as AsbjørnFølstad&Petter Bae Brandtzaeg (2018)[2] said, and moreover to bridge that gap between data and the desire of users to be able to use it quickly and easily.

2.3 Research method that was applied

Chatbot technology is not as new as it seems. It was first introduced in the mid-1960s; however, the evolution of chatbots came up in the last decade. Its fundamental feature is the interaction between systems and users in a user-friendly way to cover any need. This work is a literature review that covers the topics of chatbot, public services and CPSV standard.

In order to conduct our literature review, we defined a specific method to follow as it is proposed by the majority of students and tutors.

- Firstly, by defining which are the keywords and search terms, so to make our research on libraries.
- Then, by selecting the e-libraries which will be used to our research.
- Finally, by selecting the publications that correspond to the criteria and, in the same time, to be relative to our subject.

2.4 Digital libraries and records

The research that was conducted, has been elaborated through specific electronic libraries and by using specific key words on search. The search keywords were:

- **Chatbots.** First of all, to give a definition and answer the defined research questions based on the literature.
- By concatenating keywords **Public Services e-government** using Boolean term OR (Public Services OR e-government).
- And finally, by concatenating the terms **(CPSV) standard OR semantics (CPSV)** to cover all the subjects for our review.

E – Libraries that used for the search are:

- SCOPUS
- ACM digital library

- IEEE
- EBSCO
- SpringerLink

2.5 Chatbots

Chatbots are automated software systems, based in Artificial Intelligence (AI), that communicate with users through messages [1]. Users interact with chatbots via a chat interface, usually by typing text messages and getting back a respective answer to their request or by selecting through multiple choice fields.

There are two types of chatbot:

1. Chatbot with functions based on rules - it responds only to specific commands, those it has been programmed for. And it is obvious that its capability is limited.
2. Chatbot with functions using machine learning - these chatbots are programmed in terms of artificial intelligence, as a consequence they do understand the language not just commands[1].

The most significant difference between these two types is that rule-based chatbots are not capable of learning from interactions with users to get smarter over time, while chatbots based on AI are able to learn as they interact with more and more users [4].

Most of big companies tend to develop smart bots in order to follow the new trends of technology. Such an example is Watson, the IMB bot which is a bot based on AI and is capable of learning. However, even smart bots can't respond to each question, thus some of them are programmed to send requests to respective human staff to do so[4].

Another interesting view [5] is that chatbots are internal developed systems in messenger applications and emulating conversations with users so as to provide them with specific services. Acting like conversational recommender systems based on user preferences.

2.5.1 Why are chatbots used

According to AsbjørnFølstad&Petter Bae Brandtzaeg (2018)[2] research, there is a significant technology boost in development of chatbots and other conversational interfaces. Service providers focus on this new technology as the growth in the mobile applications' market remains stable and people spend most of their online time on messaging platforms. It is a new way for customer engagement. Another factor, as this research says, is that chatbot technology is more efficient in customer service in comparison with other technologies and also less expensive.

Darlene Fichter and Jeff Wisniewsky [4] support another point of view and claim that chatbots are considered to be the best user experience for such interactions, as people prefer to communicate whenever it is possible. Users are emotional creatures and they yearn for human connection. As a consequence, bots cover a fundamental human need that no other technology before could do so. As it said "*The bot experience is one that more closely reflects the world as we experience it*", and that is what users desire.

Apart from this, their research also support that it is more comfortable for users to communicate with a chatbot as reference transactions are better, deeper and with lower frictions.

2.5.2 Chatbots vs search engine machines

According to Andreas Lommatzsch 2018 [7], responses from chatbots are based on a knowledge base for frequently requests, that saves important time from human experts to answer the same questions every time. In contrast with search engine machines, which are impersonal and determined systems, chatbots are always friendly, scalable and with a high level of answer quality.

Search engines do not provide users with integrated answers to each question, but they do provide them with lists of documents related with the asked question. And the problem here is that users have to search on their own from provided lists, most of the times, long ones, so as to access the desired information. And this is the most critical factor that chatbots are better than search engines, because they do save time by giving an appropriate answer, rather than providing with a list where user has to seek the right document.

2.5.3 Ideal chatbots

Darlene Fichter and Jeff Wisniewsky [4] defined a set of common characteristics that 'good' chatbots might have:

- First of all, they have to be scoped. This means they have to inform the user what they are capable of doing and what they cannot do to cover their needs. Something very crucial for users to avoid wasting their time.
- They must have a main purpose. To wit, they have to do specific acceptable actions that they are developed for. For example, schedule meetings help users in searching, and so on.
- 'Honesty' of chatbots. Unlike with people, chatbots cannot tell lies to users unless they are programmed to do so. There are cases where human agents avoid to answer directly or even give incorrect answers, although customers repeat a question or seek for details. It is impossible to occur such a situation via a chatbot and even if so, it will occur without intention. In conclusion, they should be programmed to answer directly and correctly.

- User-friendly chatbots. It is vital importance to develop a friendly contact with users while 'talking' with chatbots. There are chatbots with a (fake) sense of humor, something more stimulating for users to consider a bot even more friendly.
- And finally, it is very essential for a bot, when it cannot meet users' needs to fail with an adorable way in order to do not discourage users to use it again.

2.5.4 Few concerns on chatbots

On the other hand, there are some researches which claim that bots are not so ideal as they considered to be by the majority of the technological experts. As David Skerrett's (2017)[3] article says, a significant amount of chatbots fail to meet the requirements that have been specified. There are some specific reasons according to the article:

1. the AI and natural language processing (NLP) has not reach a level where respective products could be affordable and not time consuming. As a consequence, the majority of chatbots are being developed based on rules and not based on machine learning. This means these chatbots are very simple and can only response accordingly to users' input identified keyword. Unlike chatbots based on NLP, such as Alexa, they are limited by the variables that have been programmed for.
2. It is impossible for bots to manipulate human language as people are able to do so. People use different tone, facial expressions, slang, humor, also different social groups communicate in alternate ways. No matter how, the technology progresses cannot approach the human communication completely.
3. Since competition leads all economic and technology evolution, companies tend to build products in order to offer something innovative that no other can offer and, as consequence, many products are not based in user needs. That confuses users and tend to serve poor user experiences.
4. In contrast with young users, who are more adaptable to new technologies, middle- aged users usually do not like changes, so most of them are not familiar with

chatbots and are not willing to do so until now. Thus, many companies hesitate to invest in chatbots.

5. There are too many chatbots for use, hence it is hard to find the one than fits to each case. It is claimed that there are more chatbots than users who use them.

6. Escalation processes are missing from chatbots. If users cannot find what they desire using a chatbot, there is no alternate way to achieve it. On the other hand, when a user searches for information by talking to an employee, it is easier to find other ways to achieve this, if the employee cannot help by himself. It is a matter of quality in customer experience.

2.5.5 Technical approaches, languages and tools

Despite the approaches and design techniques that are deployed for chatbots, the fundamental purpose of them is to perform a conversation with humans, which would be in natural language form and human-like [22].

Pattern Matching

A small number of chatbots is using the pattern matching method, to apply suitable responses for users [24].

Since earlier chatbots, such as ELIZA, pattern matching approaches have been widely used and adopted for bot development [22]. All of these approaches are based on a basic common idea, but simultaneously, they can vary in their complexity [22]. Actually, the majority of the existing chatbot systems works on the pattern matching of inputs that are matched with a scripted response [22].

Based on the approach of using scripted responses, scientists keep adding new functionalities to existing chatbot development techniques and apply improvements

to them, so that chatbots can become more successful. As a result, computer scientists have introduced a number of different systems and approaches regarding the same problem [22].

Ontologies (semantic networks)

In chatbot systems, an ontology or semantic network is “a set of hierarchically and relationally interconnected concepts”, that can be applied, in order to resolve synonyms and other associations between these concepts [22]. The advantage of ontologies (or semantic nets) is that they provide the opportunity of a graphic interconnection of the concepts, enabling systems to conduct searches using special reasoning rules [22].

Artificial Intelligence Markup Language (AIML)

Based on the Pattern Recognition technology, the language AIML (Artificial Intelligence Markup Language) is adopted by computer scientists for the development of chatbot systems [25]. AIML is an XML-based interpreted language and it constitutes the major technology that is implemented in well-known bots, such as ALICE [22]. It mainly consists of input rules with respective outputs for each one of them [22] and it can be referred to as a tag-based language, which uses tags as identifiers that generate the chatbot’s code snippets and commands. AIML models patterns of dialogs and enables chatbots to respond to anything that is related to an associated pattern. However, in this case, chatbot systems do not have the ability to handle inputs that are beyond the associated patterns.

Thus, based on the aforementioned code snippet, in the question “Do you know who Abraham Lincoln is?” the system would answer “Abraham Lincoln was the US President during American civil war.”, since their name is included in the associated pattern [23].

Natural Language Processing (NLP)

Natural language processing (NLP) aims to take users' unstructured phrases and convert them into a structured output, containing natural language understanding (NLU) [29].

The structured data that is generated can be used in order to select the related answer [23]. Natural Language Processing includes some major steps that are presented below: Tokenization: Through NLP, tokenization is the process of breaking down a sentence into tokens that represent each of its parts (such as words, numbers, and punctuation marks) [29].

Named Entity Recognition (NER): In this step, the chatbot system searches for categories of words (for instance, the name of the product or the user's name) [20]. In the process of NER, the names of people and products will be extracted and labeled accordingly. NER-name pairs will be stored in the conversation's history so as to keep track of the context of the conversation [29].

Normalization: Through the normalization process, text is broken up into its component parts; sentences are broken into words and punctuation marks and words are broken into their phonemes (the smallest unit of sound in speech [22]) [29]. A chatbot system may process user's text in order to find ordinary spelling mistakes, which would result in a more human-like conversation when they are used by the chatbot inside the dialog [23].

Part of Speech (POS) tagging: POS tagging aims to label each word of the users' sentences with its part of speech (for instance, "noun", "adjective" etc.). Also, a word's label can be rule-based, which refers to the ability to create a manually-generated set of rules for ambiguous words. Dependency Parsing: In the dependency parsing step, a chatbot system searches in the user's text in order to find objects and subjects which may have some kind of dependency or relation [23].

2.5.6 Where are they used

Nowadays, there are many areas where chatbots are used to provide with certain services. As [5] says, for example, many applications related to pedestrian navigation, made a further step to help users walk through unvisited cities or find streets that they have never been before, by applying chatbot systems to boost that experience.

However, generally as Følstad&Brandtzaeg (2000) [2] said, as chatbot technology may provide more affordable, available and accessible services, it can be used as an essential tool for engaging with all type of customers, from commercial purpose to public services purpose.

such examples are:

Education

Innovation has effectively made its initial steps, making numerous chatbots whose reason differ contingent upon the correct zone of utilization in the education area.

There is plenty of examples to state this. Chaniago et al have developed a bot which empowers parents to check that their kids are in their school as they should have. Messenger also, through Buddy (a MOOC) proposes MOOC courses [9].

An associate being initiated by two fundamental parts, an Android application, and a server stage that has been developed for students is NLAST. The communication between students and machine can occur either orally or by text messaging. And its principle purpose is to enable understudies to find course material from an immense school repository, to get suggestions about the current learning material and to check the students' progress through an appraisal. This bot searches schools' data and responds respectively to the requests [10].

Chatbots also used to empower the learning process, by allowing students to make their questions to obtain information from websites, such as Wikipedia. This may occur either via speech or by text messaging, thus increasing user -friendliness[11].

Contribution of chatbots in the field of education does not have limitations, as bots can also collaborate with libraries and teaching staff to make their job a lot more painless. Bots can support library staff by answering simple requests quickly and efficiently[12].

In the same direction instructors can take advantage by using them, while they can interact with them by adding or deleting new material in order to make chatbots capable of serving any demand.

A respective example is Confucius, an internet-based bot, it is based on question-answer type system and aids in the learning process of students. It provides two modes, a discussion and a lecture mode, where for any wrong given answer, a discussion about the answer is taken place and then the correct answer is provided [13].

E-commerce

More and more companies are aware nowadays for the advantages of chatbots so they tend to enhance their systems with sophisticated chatbots in order to increase their profits in online commerce. Except from Amazon's Echo and Alexa that is applied on many devices to empower user friendliness, also Facebook's Messenger can now interact with its customer and even IKEA has a similar assistant called Anna[9] to help customers search for their products.

A meaningful obstacle for municipal authorities to serve citizens is how to inform them appropriately and quickly without any frictions. And as open data gets larger over the time, it is very tough for public services to help each citizen upon their

request, thereat, several of them start using the technology of chatbot to provide better solutions. This idea was adopted from brand name companies, while plenty of them started applying chatbots in their web platforms to attract more users and make them more pleased[5].

In this case, we are going to use chatbot technology for similar purpose, as this dissertation has to do with chatbots in public services. So, while we elaborate this literature review, we are also searching for chatbot technologies to apply the appropriate one for our occasion.

2.6 Chatbots in public services

As it seems from an older publication [6] made in 2011, neither technology nor software systems were in such a level to support public services to use 'virtual agents' (chatbots) to boost the experience and the delightness of customers at these early days. Although, market and the majority of the organizations recognized the importance of chatbot technology for all types of information management systems. This technology was not established respectively.

And as this research claims, the most significant reasons were:

lack of appropriate algorithms to support a smooth and meaningful communication between customer and the system, underdeveloped virtual systems, machine learning had not evolved so much yet.

However, this publication [6], states the necessity of chatbots in all kinds of public services. It refers that after a research among informatics and information management experts who were seeking for a solution in communication between public administrations and clients, there are many theoretical aspects that support the technology of chatbots as an ideal solution. Two basic purposes for applying information technology on public services are, to increase management efficiency and information processing.

Bartosz Kopka also stated the basic advantages and disadvantages of this technology upon public services,

Advantages:

- System becomes more attractive
- Useful for customers' searching attempts
- Reduced costs in customer service

Disadvantages:

- Time consuming as it needs updates regularly in order to handle new data
- Cannot mimic exactly the human behavior

Bartosz Kopka's review on a general term 'software agent' for that occasion defined some rules:

- Independency and autonomy. System works independently without direct human interruption. Can be initiative by asking questions, suggesting possible answers and even denying at special circumstances requests.
- Reactivity and proactivity. Understanding any changes in its environment and reacting properly.
- Be accessible and effective unceasingly. Every hour of the day, each day.

2.7 Three chatbot systems for public services are evaluated

Singapore's chatbot system for public services [30]

By examining Singapore's chatbot system for public services, we realised that it is a very basic system which covers only simple scenarios to support Singapore citizens and it operates more like a search engine machine rather than a chatbot. So , it is not human-friendly and does not stimulates user's interest to use it.

It is also based on static data, so it does not retrieve any changes that take place for public services [30].

So we could mention that it is not so helpful for users and it is almost out of date.

Poland's chatbot system for public services [7]

The system is a little bit more useful than Singapore's system and it is easy to understand that even from its user interface which is more friendly. It supports small talk and is able to correct typos [31]. In our tests the system provided useful answers for about 2/3 of our questions. For many questions it did not deliver a direct answer however explained who to contact in order to get an answer.

An integrated Chatbot system for public services (Andreas Lommatzsch in 2018) [7]

Here we will present an integrated chatbot system for public services of Berlin that was developed by Andreas Lommatzsch, the best chatbot system as it is evaluated in our research.

Andreas Lommatzsch developed a chatbot system very efficient and after a lot of search as it is clear on his work, in order to support citizens of Berlin to search easily the desired information for public services.

This system is capable of answering a broad spectrum of questions and it can obviously lead a citizen to find the appropriate information as it provides correct answers for the most of the question that we asked.

It supports also other popular languages such as English , provides a variety of possible answers for specific questions in order to be more human friendly and its user interface is user-friendly.

2.8 CPSV and why to apply it

CPSV-AP standard is data model used for the description of public services in European countries. It was proposed by European commission and it is offered to European countries in order to standardize the semantics for catalogues of public services. And this is very meaningful for both users-citizens to seek any information they want to and also for machines as metadata are machine - readable.

In that way, more and more countries start using this model such as Estonia, Finland, Belgium, Italy and Netherlands. To create a user-centric and interoperable catalogue of public services for any administration office. CPSV-AP stands for Core Public Service Vocabulary Application Profile. A critical advantage is that it is open source, thus any public organization is capable of doing any custom changes to cover their needs.

As it is stated by Alexandros Gerontas, Efthimios Tambouris and Konstantinos Tarabanis in 2018 [8], CPSV is a European Standard developed to model public service descriptions and bring consensus on the basic PS metadata across EU Member States.

Its purpose is not to record any feature of each property of PS all over Europe but to "*capture the minimal, global characteristics/attributes of an entity in a generic, country and domain neutral fashion. It can be represented as Core Vocabulary using different formalisms (e.g. XML, RDF, JSON)*" as a simplified model.

Both new public service models and existing public services can be designed or mapped respectively based on CPSV. That is why CPSV is proposed as the linking element between PS models, to keep semantic interoperability in an appreciated level.

In this [8] publication, they have defined some benefits for both users and public administrations by using CPSV standard:

Advantages for citizens:

- Better access for all citizens, including those with special needs such as moving difficulties.
- Saving time and money while citizens do not have to visit an administration office or talking with an agent.
- Reliability of citizens on public services.

Advantages for public administrations:

- higher quality of service as less and less people is going to wait to be served.
- organizations will cooperate with each other more efficiently.
- descriptions of public services may be used for applications related with these public services.

2.9 Need for chatbot systems in public services

In the previous sections, public services and semantics upon them were studied in the context of chatbots. Furthermore, different tools and technologies available for chatbot development and implementation were discussed.

As deducted from all the above nowadays, there is indeed the appropriate technology (such as A.I, machine learning e.t.c) to support well structured chatbots and there is also high demand for such systems in public services. However, only few countries (Singapore, Poland) currently provide chatbot systems for public services and all of these systems are not sufficient enough to cover citizen's needs, since they mostly operate as searching machines. A significant exception, was Andreas Lommatzsch's system [7] for Berlin administration that was implemented in 2018 as already indicated in this research.

Therefore, we decided to develop a chatbot system for a greek web portal, called diadikasies.gr, which provides public service information. Obviously, in most European countries there is a lack of such a system and Greece is not an exception. The advantage of the proposed system is that it is based on real time data in contrast to Singapore's and Poland's systems which are based on static data. Furthermore, similarly to the Berlin's chatbot, the proposed system is user-friendly aiming in enhancing user experience. Last but not least, the proposed system adheres to the European Union's principles on the sector of public services, since this system utilizes semantics and metadata based on CPSV standard.

2.9.1 Web portal : Diadikasies.gr

Diadikasies.gr is a web portal that was introduced by Greek government in order to provide a collection of public services for Greek citizens. Its model is based on Wikipedia (thus the definition of wiki, which is an open source tool for a big collection of archives in a web page). It is continuously enriched with new public sector services and processes by public servants who are responsible for the delivery of the services offered by their administrative authority.

As there is a big amount of public services in this web portal and it is continuously enriched with new ones, it tends to be difficult and unpleasant for citizens to seek the services they need. Thus a development of a chatbot to enhance this experience is of vital importance.

In our case, we will annotate five services that exist in the web portal in order to prove (for educational purpose) the proper operation of the chatbot that we are going to develop.

(For purpose of our thesis we will pass the names of the services in English)

This five services are:

1st service

Native name of the service: “Εγγραφή ανηλίκου αδήλωτου¹”

¹source:https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%86%CE%AE_%CE%B1%CE%BD%CE%AE_%CE%BB%CE%B9%CE%BA%CE%BF%CF%85_%CE%B1%CE%B4%CE%AE%CE%BB%CF%89%CF%84%CE%BF%CF%85

2nd service

Native name of the service: “Έγκριση συμμετοχής υπαλλήλου σε επιμορφωτικό σεμινάριο²”

3rd service

native name of the service: “Είσπραξη ημερήσιων τακτοποιητέων εισπράξεων³”

4th service

native name of the service: “Βεβαίωση μόνιμης κατοικίας⁴”

5th service

native name of the service: “Δικαιολογητικά για ανέργους⁵”

2.9.2 Ideal requirements of a chatbot

In order to fulfill the requirements set in the context of this thesis, it is considered that a chatbot system should aim to offer:

- a user-friendly system that will help users-citizens to search efficiently desired information about public services.
- a variety of possible answers to imitate a human-to-human-based conversation.

² source:

https://diadikasies.gr/%CE%88%CE%B3%CE%BA%CF%81%CE%B9%CF%83%CE%B7_%CF%83%CF%85%CE%BC%CE%BC%CE%B5%CF%84%CE%BF%CF%87%CE%AE%CF%82_%CF%85%CF%80%CE%B1%CE%BB%CE%BB%CE%AE%CE%BB%CE%BF%CF%85_%CF%83%CE%B5_%CE%B5%CF%80%CE%B9%CE%BC%CE%BF%CF%81%CF%86%CF%89%CF%84%CE%B9%CE%BA%CF%8C_%CF%83%CE%B5%CE%BC%CE%B9%CE%BD%CE%AC%CF%81%CE%B9%CE%BF

³ source:

https://el.diadikasies.gr/%CE%95%CE%AF%CF%83%CF%80%CF%81%CE%B1%CE%BE%CE%B7_%CE%97%CE%BC%CE%B5%CF%81%CE%AE%CF%83%CE%B9%CF%89%CE%BD_%CE%A4%CE%B1%CE%BA%CF%84%CE%BF%CF%80%CE%BF%CE%B9%CE%B7%CF%84%CE%AD%CF%89%CE%BD_%CE%95%CE%B9%CF%83%CF%80%CF%81%CE%AC%CE%BE%CE%B5%CF%89%CE%BD

⁴ source:

https://diadikasies.gr/%CE%A7%CE%BF%CF%81%CE%AE%CE%B3%CE%B7%CF%83%CE%B7_%CE%92%CE%B5%CE%B2%CE%B1%CE%AF%CF%89%CF%83%CE%B7%CF%82_%CE%9C%CF%8C%CE%BD%CE%B9%CE%BC%CE%B7%CF%82_%CE%9A%CE%B1%CF%84%CE%BF%CE%B9%CE%BA%CE%AF%CE%B1%CF%82

⁵ source:

https://el.diadikasies.gr/%CE%94%CE%B9%CE%BA%CE%B1%CE%B9%CE%BF%CE%BB%CE%BF%CE%B3%CE%B7%CF%84%CE%B9%CE%BA%CE%AC_%CE%B3%CE%B9%CE%B1_%CE%91%CE%BD%CE%AD%CF%81%CE%B3%CE%BF%CF%85%CF%82

- a multilingual system to support other languages catering for international target audience.
- description of metadata based on CPSV standard so as the system will be capable to operate in other european systems if required in the future.
- data that will be retrieved by the system in real time.

2.10 Towards Research questions

Summarizing, in this chapter the theoretical and conceptual context of this dissertation was established. Additionally, the requirements and the goals of the proposed system were presented. Based on the aforementioned, the following research questions were formulated:

Research Questions:

RQ1: Is it possible to annotate easily and transparently already existing public services with semantics related to chatbots?

Easily and transparently means by using already existing standards and without intervening in the already existing public service descriptions.

RQ2: Is it possible to train a (preferably open source based) chatbot to help a citizen find a service according to public service metadata?

This possibility has to be demonstrated by a proof of concept implementation.

These RQs direct the design and implementation choices of ours, that are presented in the next chapters. In chapter 5, we try to answer whether we have dealt with them adequately, or not, where the results and contribution of this dissertation are illustrated.

3 Design and Architecture

In this chapter the functional requirements of the system are presented. Then, the general flow of the operation of the system along with use case scenarios are illustrated. And finally, the last section presents the tools, the programming languages and the frameworks that were used to develop the desired chatbot.

3.1 Design and Architecture

The most important step to develop a web application, and generally a software system, is to design its implementation properly and follow the plan accordingly. If the design of a system is not well structured, it is possible that the development of the system will fail to fulfil the requirements. That is why, most companies invest on the designing part and spend a lot of time on this procedure [32].

With regards to the design and architecture of the developed chatbot not only functional requirements were taken into consideration but also features such as usability and user-friendliness.

3.1.1 Requirements of the system

After an extensive research on chatbot systems focused on public services, several useful design techniques were chosen to be applied to our web application, aiming at the creation of an efficient and usable system. The chosen framework used to develop the chatbot (Botpress) was the ideal one to facilitate the implementation of all the functional requirements and desired features.

In paragraph 2.9.2 the ideal requirements of a chatbot were presented based on the comparison of other existing chatbots in the context of public services. In this section, the high level requirements of the developed chatbot are presented. Obviously, these requirements tend to be similar to the ideal ones.

Requirements of the system:

- The focal point of the developed system is to guide users-citizens that use diadikasies.gr, a web portal that provides information about public services, to find the appropriate information efficiently and quickly.

- The system achieves interaction with users by asking a sequence of questions. The system leads users to the appropriate service by utilizing users' input, i.e. their answers to the questions asked. That way the range of the possible answers that the chatbot can provide to users is gradually decreased and only relevant to the users' quest answers are provided.

- Next to each question, respective buttons (suggestions) should be displayed on the user interface to allow users choose one of them, if any of these suggestions satisfy their request. What is more, users' are also able to type a more custom and specific request if none of the suggested one fulfills their requirement.

- Descriptions of each public service will be created based on CPSV standard (such as name of the service, language of the script, country of the provided service, keywords based on the name of the services, and type of the service), in other words semantic metadata.
- The required information (data) and semantic metadata should be retrieved by the system on real time. That way the system does not contain hardcoded data and thus the system can easily be integrated on many platforms and systems.
- These metadata (descriptions of the services) will be stored in a specific server, to be retrieved by the system on real time.
- When a user ends up with a desired response of the system, the system should provide the option to the user to be easily redirected on the page of this service in diadikasies.gr web portal.

For the purpose of this thesis, five services based on CPSV standard will be annotated in order to prove the appropriate operation of the system. However, if the chatbot system will be delivered to greek public authorities in the future, public servants should follow the same process in annotating services that are required to be managed by this system.

3.1.2 General flow of the system

The system retrieves the semantic data from the server each time a user initiates a conversation with the chatbot. Then the system asks a sequence of questions with provided suggestions (suggestion-buttons are based on the semantic data of the services) to limit the range of services according to users' preferences. Finally, the system asks the user about the services he/she seeks for and after user's choice, it

provides the link of the appropriate service on the web portal (diadikasies.gr) to let user navigate to that page.

The sequence of questions refers to language, nation and type of the services successively, in order to limit the possible range of documents according to user's choices. If the suggestions-buttons upon a question do not satisfy the user, the latter is allowed to type his/her own request and the system will respond respectively whether there is a match or not.

3.1.3 Use cases of the system

The provided use case (figure 1) indicates the interactions among users and the system. Specifically, the main actions of the users are to choose the provided suggestions-buttons by the chatbot or to apply a request in the input field.

In figure 1 are also illustrated the actions that public servants should apply to create metadata for services so as the latter can be retrieved by the system.

Namely, to annotate a public service based on C.P.S.V standard, and to store this semantic information to a specific server.

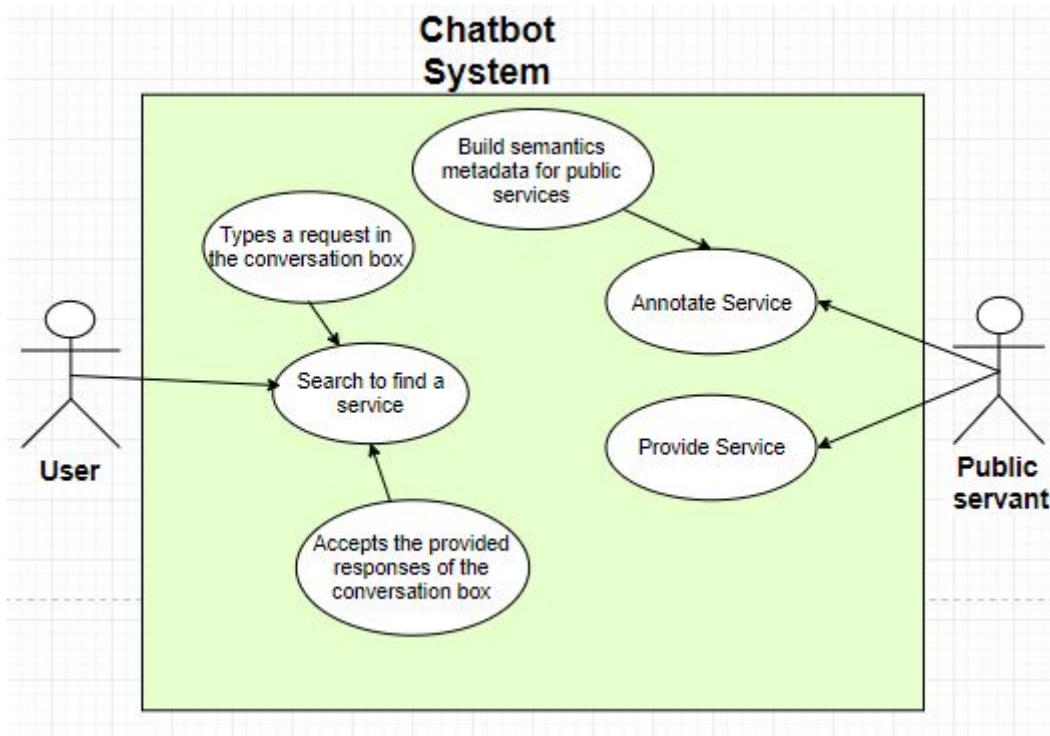


Figure 1. Use case

In the next sections two use case scenarios are deployed. The first scenario describes a citizen who uses the chatbot system as an assistant in order to find a service. The second scenario presents how a public servant annotates a service based on CPSV standard and stores it in a JSON file on the server.

Use case scenario for citizen

As previously stated, the system is based on a sequence of questions to limit the range of possible services. It is obvious that on the provided scenario below, the user responds to the provided questions in order to find the desired service.

- the User initiates a conversation with the chatbot by activating the conversation box.

- Chatbot replies with a question which asks the user to select one of the provided buttons with the languages that are used in the public services.
- the user clicks on the button with the desired language.
- Chatbot replies with a question about the nation of the public services that the user may be interested in, by providing suggestions (in the form of buttons) with the appropriate countries.
- the user selects the country that he/she is interested in.
- Chatbot then asks the user to choose what type of services he/she is seeking for.
- the user selects the preferred type of services.
- Chatbot then asks the user to type which service he/she wants to find.
- the user types in the text field of the conversation box the desired service.
- Chatbot provides the services that are relevant to the input.
- User selects the service by pressing the respective button.
- Chatbot provides a link to ‘diadikasies.gr’ web portal that leads to this service.

A demonstration of two real scenarios between user and the system is provided in the next chapter (4.5 proof of concept).

Use case scenario for public servant

- The public servant creates the semantics for the provided service in a CPSV platform (output is on RDF format). CPSV is used in order to denote the semantics to be used for the descriptions of the public services by the chatbot application. The process for the creation and the storage of such information is described in the next chapter (4.1.1) and it is also provided with technical details in Appendix (8.4).

- Moreover, public servant converts the created metadata to JSON format via an RDF to JSON converter tool. The chatbot technically requires JSON files in order to analyze real-time data and make decisions. So, we use a specific tool described in next paragraphs in order to convert the previous step's RDF description to JSON format.

- Finally, the public servant stores the JSON file with the metadata of the service on the server. These files should be stored on an online server in order to facilitate online and real-time functionality of the chatbot.

3.2 Tools, frameworks and languages that were used

3.2.1 Framework to develop the system

After an extensive research on the existing chatbot frameworks, the Botpress framework was selected for the development of the chatbot. It was considered to be the most suitable one compared to others as illustrated in (TABLE 1.1). Botpress is an open-source developing framework that facilitates any changes in its source code files and therefore, it is highly flexible. Moreover, it is also designed to retrieve information, i.e. fetch data and metadata, on real time, a feature that the remaining frameworks could not yet support. Last but not least, the Botpress framework is the most appreciated one among developers, and the Botpress's community is very large and supportive. Thus, it is feasible to overcome any obstacles during development since someone can easily find support and assistance online. Hence, we decided to develop the system based on Botpress framework to take advantage of all these benefits.

	Botpress	Amazon Lex	Google Dialog Flow
open-source	x		
support data on real time	x		
support static data	x	x	x
adjustable	x		
user-friendly interface	x	x	x

Table 1. Comparison between chatbot frameworks

About Botpress



Figure 2. Botpress logo (Source:

<https://startupbeat.com/the-lightning-pitch-botpress-the-ultimate-bot-framework/25322/>)

Botpress is a chatbot framework that many companies use to support their applications in order to let users 'communicate' easily and efficiently with their systems. It is open source, thus developers can adapt it to fit in their requirements.

Additionally, Botpress is very flexible and facilitates fetching and managing data.

What is more, it can be deployed to almost any well known platform or application.

Finally, there is no need to deploy Botpress on any platform on early steps, a fact that provides resilience and flexibility during design and development phases. This means that the developed chatbot can be finally deployed and integrated on platforms after designing and implementing the basic dialog flows and functionalities.

3.2.2 Botpress framework and JAVASCRIPT



Figure 3. Javascript logo (Source: <https://www.computerhope.com/jargon/j/javascript.jpg>)

Botpress is developed exclusively with Javascript and it can be modified by using Javascript. This means that all the implementation will be based on this programming language. The most popular and the most preferred programming language nowadays in web applications [26]. It is considered as an interpreted programming language, which is mainly used for enhancing web page functionality and interactivity [27]. Software written in JavaScript, which is deployed in HTML documents, provides useful client-side computation facilities and access to the client system, making web pages more interactive, engaging and responsive [28].

Nowadays, the overwhelming majority of modern websites use this language, and all major web browsers on tablets, smartphones and desktops, include JavaScript interpreters, making this language the most ubiquitous programming language in the history [26].

3.2.3 CPSV tool (to create semantic information for public services)

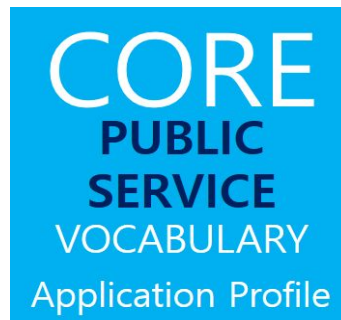


Figure 4. C.P.S.V - A.P logo (Source: <https://joinup.ec.europa.eu/solution/core-public-service-vocabulary-application-profile>)

CPSV is considered to be a remarkable standard in European Union for e-public services and it is promoted to be used by all countries in the EU. The official authorities of European Nations also provide a public service description creator tool ‘PSDescriptionCreator’, a very useful tool that does not require any technical skills by its users. This tool was utilized to annotate each public service of the web portal based on CPSV standard.

In the context of this thesis, five metadata files related to five public services were created in order to examine that the system operates properly. In the future, if this system is going to be used by public services, public servants may easily follow the same procedure.

3.2.4 RDF to JSON converter (tool)



Figure 5. Easy RDF tool logo (src: <http://www.easyrdf.org/converter>)

The output from the CPSV creator tool is in RDF format. However, it is considered to be better to convert this metadata information to JSON format as the BOTPRESS framework can handle these information with ease when this format is used.

So, after examining several RDF to JSON converters, we ended up that the best choice is to use “easy RDF” converter.

3.2.4.1 what is JSON format

But what exactly is JSON? JSON (JavaScript Object Notation), is a minimal, readable format for structuring data. It is used primarily to transmit data between a server and web application, as an alternative to XML. Nowadays is preferred from the

majority of the developers as it is easy for humans to read and write and also for machines to parse and generate data in JSON format. In addition, it can be used in almost all well-known languages and gradually, it will replace all other formats, such as XML which tends to become outdated.

4 Implementation

In this chapter, there is a thorough analysis of the steps that have been taken in order to develop the chatbot.

Additionally, the pattern and the restrictions for the development of the system are described.

More specifically, it is presented how the flow of the system is defined, how the semantic data required for the services were created and stored in the appropriate server, and how the system retrieves these data. Moreover, it is demonstrated how the system manages to help users find the desired services by using semantic data, followed by a proof of concept to state the proper operation of the system.

4.1 design pattern and restrictions (due to the framework)

Because the development was based on a specific framework (Botpress), there were some restrictions on the development of our system. First of all, it is all based on Javascript, as Botpress is javascript-based, so no other programming language could be applied for the development of the chatbot system.

Secondly, even though Botpress is considered to be an open source framework, this only applies in regards to its functionalities. This means that its user interface is not adjustable (not open-source) and thus no changes were applied in the front-end part. Furthermore, the source code of the framework has a specific structure, and we made our changes based on this structure. More precisely, the flow of the system and the way it handles each case is defined by a tool of the framework called ‘flow diagram’. The flow diagram consists of nodes. Each node has specific methods and according to the result of each method, the flow moves on to the next node.

The required methods by our system were all defined in a specific file of the system called ‘actions.js’. This file is used by the ‘flow diagram’ tool in order to “read” and use these methods. The ‘actions.js’ file and the implemented methods are presented in Appendix 8.3 ‘implementation of methods’ paragraph.

4.1.1 Flow diagram (a tool of the framework)

By referring the term flow of the system, we imply the sequence of any possible operation the system can perform based on users’ actions.

The Flow diagram tool of the Botpress, depicts the pattern that the bot is going to follow for each case during execution. Botpress provides this tool-platform, where developers can edit every part and every case of the program, along with all

alternative cases, and also view the whole flow of the program. The flow diagram specifies the execution process and structure of the bot, i.e. the operation flow of the bot. In this flow, and sequentially in the flow diagram, everything that is required and will be used has to be declared, i.e., functions, methods, messages, texts, etc, specifying thus where, when and how things are going to be applied and used. The whole view of the flow diagram and the way that it works is explained technically in Appendix 8.1 ‘flow diagram’.

4.2 Semantic Data

In this section, a brief description is deployed on how services are annotated based on CPSV standard. Basically, this section refers to the keywords that will be used to describe the services. Additionally, it is presented the way that these data is converted to a readable, by our chatbot system, format (JSON) and where do these data are stored.

4.2.1 Description fields for each service

As mentioned before, five indicative services based on CPSV standard were annotated with semantic data. These semantic information was utilized by the chatbot in order to guide users properly to the desired service.

For each service six semantic description fields were applied, as they considered to be the most critical to describe a public service. More description fields can be added, if necessary, in the future by the Greek public authorities that may use the system.

‘PSDescriptionCreator’ is the tool that aids in defining the appropriate values to these six fields for each service. The output of this process is an RDF text having

this semantic information. Noted that this form of semantic data is not readable by the system.

The aforementioned description fields for each service are:

- **Identifier:** this field stores the url of the link for the specified service in the web portal.
- **Name:** In this field , the title of the service is declared.
- **Description:** A description of the specific service is stated in this field.
- **Keyword:** This field stores one or more types of services in order to properly describe the nature of the provided service.
- **Language:** The language that the service is written on.
- **Spatial:** This field defines for which nation this service exists.

It is considered that these six semantic descriptions for each service are capable of providing the appropriate information to the system to guide users efficiently. The language, the nation and the type of the service for the respective questions that are provided to the users, and the identifier (url) to provide the link and definitely the name of the service.

(Although most of the service are in Greek language, we annotated them in English in the context of this thesis).

In the following figure 6, these description fields are illustrated, as defined in the PSDescriptionCreator tool.

Identifier	https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%86%CE%AE
Name	registration of an adult demonstrator
Description	registration of the municipality of an undeclared person
Keyword +	municipality
Sector +	<input type="text"/>
Type +	<input type="text"/>
Language +	English
Status	<input type="text"/>
Spatial +	GRC-Greece

Figure 6. PSDescriptionCreator tool

4.2.2 How semantic information is created and stored

This semantic information is formed in RDF format as an output by the ‘PSDescriptionCreator’ tool (see Appendix 8.5.1 image). However we are converting this information to JSON format with the converter tool (RDF to JSON converter) because it is a lot easier and efficient for the chatbot to manage information in JSON format.

Converting from RDF to JSON

By using the converter tool (see Appendix 8.5.2 image), the output of this conversion is the semantic information in JSON format. That is, what is required by the chatbot to be managed.

The output is being stored on the server of I.H.U (<http://imagnisa.labs.ihu.edu.gr/>) and from now on, the system is capable of retrieving this information.

The whole process is shown analytically with technical details in Appendix (8.4 paragraph ‘Creation and storage of semantics data’).

4.3 Chatbot system (how the system works)

The system is developed in order to retrieve semantic data from a server each time a user initiates a conversation. The objective of this chatbot is to guide a user in finding the desired service. This is achieved by asking the user a sequence of questions and providing few possible suggestions on each question. By this interaction between system and user, the system limits the range of possible services and provides the remaining services when user finally asks for a service. Finally when user selects a service, the system provides the respective link of the service on the web portal diadikasies.gr.

The whole process is explained technically in details in Appendix (8.3 paragraph).

4.3.1 Information retrieval (semantics) by the system

The developed chatbot retrieves the data (semantic information data) for the services from a server (in our case the hosting server can be found in the following link <http://imagnisa.labs.ihu.edu.gr>), every time a user activates the conversation system (on real time).

It is developed that way, so as to retrieve up-to-date data, i.e. any possible changes to existing services or even additions regarding new services. This is the most significant advantage, since no other chatbot system for public services is capable of doing so, at least not yet.

After fetching the information for the services, the system stores all data locally, for reasons of efficiency and quickness.

This process is described in more technical details, with images and comments upon source code, in Appendix 8.3.

4.3.2 How this semantic information is used by the system

After the system has retrieved all required data, a series of questions are being asked to a user. The system provides several possible answers that were constructed upon the fetched data. Users' answers provide the vital to the chatbot input in order to gradually filter the available services and provide the more relative ones to the users' quests.

Questions that the system asks

This is the sequence of the questions that the system asks to the users in order to filter the range of services and lead users to the desired service.

1st question about language (with provided answers):

Chatbot asks user about the language of the desired service. Alongside, the chatbot checks the language field from the stored (semantics) data, and provide buttons-answers with the value of those languages.

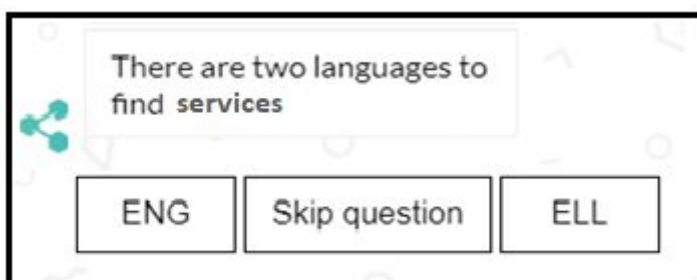


Figure 7. Provided suggestions (buttons) for language

2nd question about country (with provided answers):

Via this question chatbot asks the user for which country's services he/she is interested in. Similar to the previous question, the system checks the Spatial (country) field about the values of the services in order to provide the respective buttons-answers to the user.



Figure 8. Provided suggestions (buttons) for nation

3rd question about type of service (with provided answers):

This question gives the chance to the user to limit significantly the range of possible services by declaring the type of service that seeks for. Similar to the previous two questions, the chatbot provides buttons-answers after checking the field of type in the semantics data.

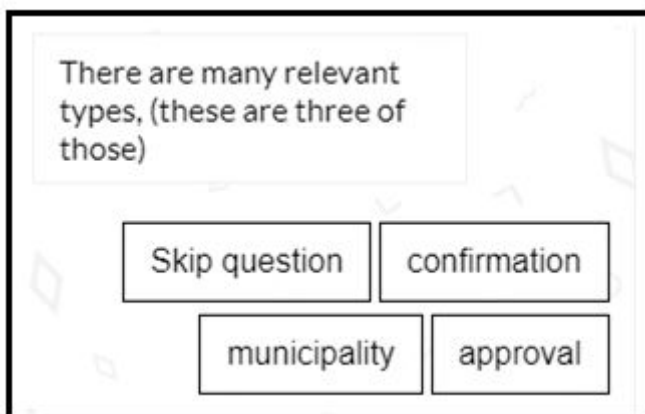


Figure 9. Provided suggestions (buttons) for type of service

The methods that were implemented to generate the questions along with the provided suggestions-buttons for language, nation and type of service respectively are presented in Appendix 8.3, paragraph, '**showQuestionforLanguage**' , '**showQuestionforNation**' , '**showQuestionforType**'.

Towards a user-friendly interface

In case there are many different values in a field upon a question (i.e many types of services), it will be very unpleasant and a bad experience for a user to view many buttons-answers to the conversation box of the system. Thus, there is a limit of maximum five buttons-answers to be provided in the interface for each question. This refinement occurs by showing the values that exist more times in the semantics data.

What happens with the other values beyond the five most found

Although the other values are not shown as buttons, users have the opportunity to type a reply to the question (input field) if the provided buttons are not covering their needs. Thus, the system process the user's input and responds accordingly.

How the system 'reads' user's input

The system splits user's submitted input to unique words and checks whether these words match or not with the semantic descriptions, i.e. the retrieved data. That way the system investigates whether there is a better suggestion for the user or not, and if so, it responds accordingly.

This implementation is illustrated in details in Appendix 8.3, paragraph implementation of methods

4.3.3 Limitation of services according to user's preference

The system receives user's respond for every question. Then, it filters the services, based on the semantic data of each one of them, that do not match with the received reply. Thus, after user's respond on these three questions, the services are filtered based on language, nation and type of service and the most relevant ones to the user emerge.

Skip button

However , user is also given the change to avoid answering a question (even all of them) by skipping it.

For each question, an additional 'skip question' button is provided to allow a user not to answer this question if he/she wishes to.

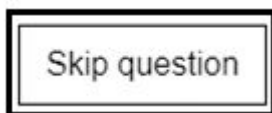


Figure 10. Skip question Button

4.3.4 System provides the link of the service

So far the chatbot has filtered the available services based on user's preferences regarding the language, the country and the type of service. The fourth question asks the user to type what he/she is looking for from the available services. Based on user's input the chatbot searches in the filtered services if there is a match to user's requirements. In case there are available services the chatbot provides a list with the relevant services. When the user clicks on one of the provided buttons (with the name of the service) the system provides the link of that service on the web portal (diadikasies.gr) , so that user may navigate to the page by clicking the link.

an example of a service (attestation of permanent residence) and its link below.

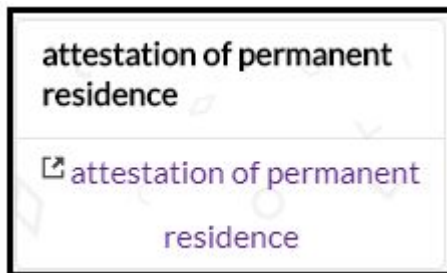


Figure 11. Provided link for the respective service

4.4 The conversation interface and the possible interactions

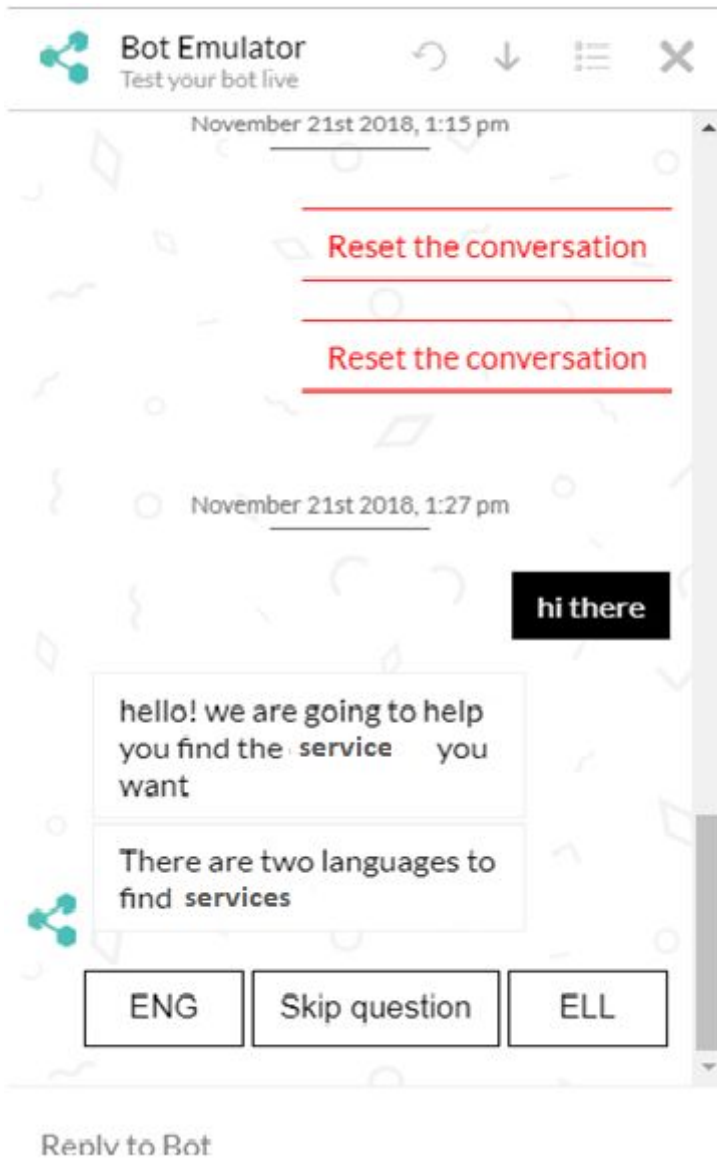


Figure 12. indicative image from the conversation interface

This is the user interface of the Botpress framework when the chatbot box is opened. The main actions that a user can apply are:

- to open the chatbot box to initiate a conversation with the system.
- to close the conversation box.
- to refresh and start a new conversation.
- to download the conversation in a text file.
- to press a button (suggestion) that cover his/her need. The chatbot provides buttons according to the semantics data and the responses of the user.
- to type a request in the input field. If provided buttons-suggestions do not cover user's needs, user may respond to the system with a text.

4.5 Proof of concept

In this section , a demonstration is provided to prove that the prototype system operates as expected. By utilizing the chatbot, a user can find the required information about the services he/she is concerned about. Based on user's replies , the chatbot gradually limits the range of the available services and finally, in the case that a user accepts the suggested service, the system provides the respective link on the web portal. Two different scenarios are presented below, providing thus an overview of the system's operation, and indicating how alternative situations are managed by the chatbot.

First scenario

Brief story:

A user starts a new conversation with the chatbot in order to find the appropriate information for public services on the web portal (diadikasies.gr). After typing a starting message to activate the bot, he/she selects the English language by clicking the appropriate button. Then the user skips the question about desired nation by clicking the 'Skip question' button. On the next chatbot's suggestion regarding the type of the service, the user clicks on a button referring to confirmation. Thus, the system limits the range of metadata and information according to the user's choices.

The last question asks the user to type what is the service in need. The chatbot matches user's input to the titles of the remained services, and provides suggestions with the most relevant services. The user clicks on the button, i.e. on one of the provided suggestions, with 'permanent residence' title. Sequentially, the chatbot provides two options, one with a button referring to the title that matches to the search, and the second is a button to search again. As the user accepts the found service, by clicking the respective button, the chatbot opens the link on the web portal for that service in a new browser tab.

step1

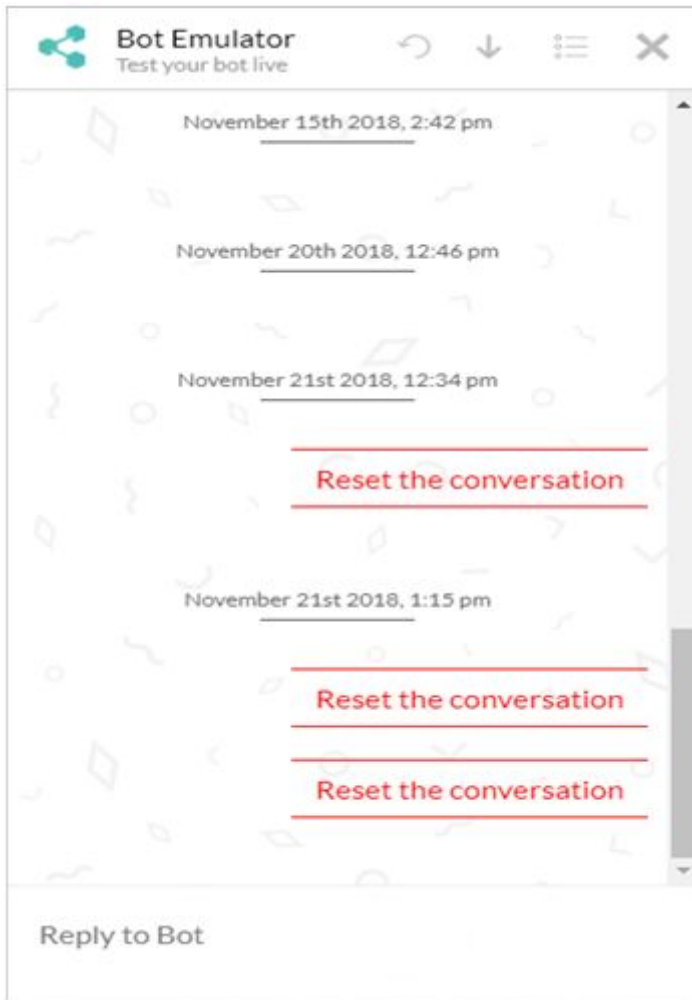
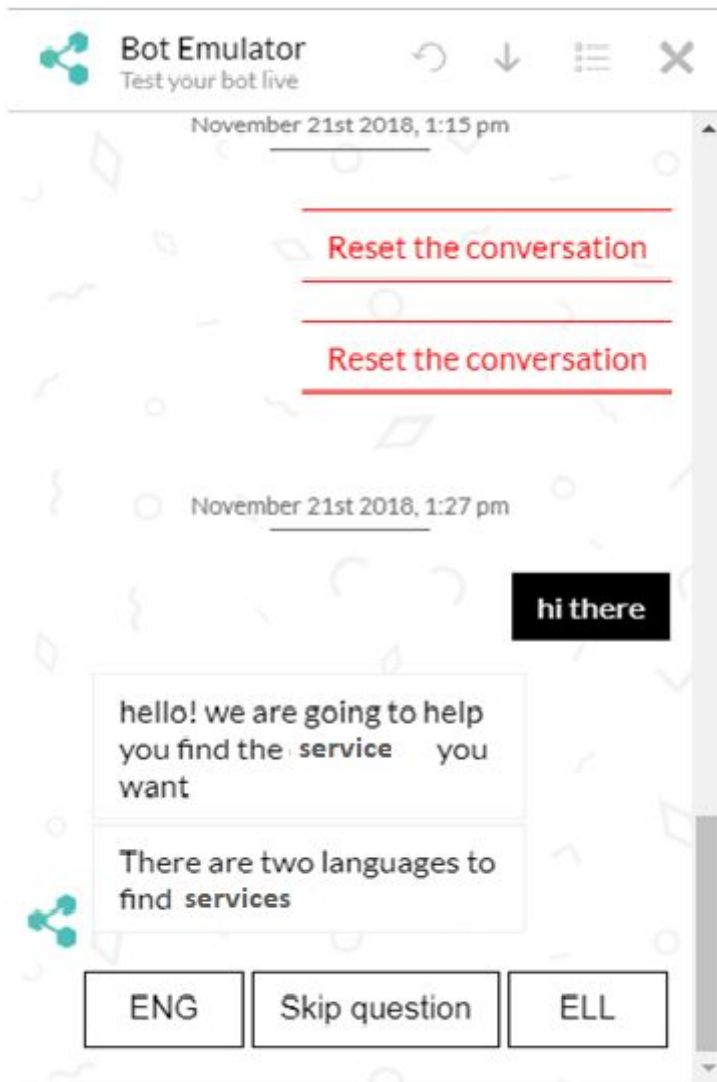


Figure 13. Initial chatbot interface

This is the interface where users interact with the chatbot. Obviously, the chatbot has been reset (Figure 13) in order to start a new session (conversation).

At the beginning, the user has to type a message, just to let the bot know that the user decided to start a conversation (chat). User's messages are being typed in the area at the bottom as suggested by the system ("Reply to Bot").

step 2



Reply to Bot

Figure 14. Question about language with provided suggestions

The user has typed "hi there" , and the chatbot replied with two messages. The first one is a welcome message addressed to the user ("hello! we are going to help you find the service you want"), whilst the second one informs the user that there are information about services in two different languages.

The second message is displayed after the system retrieves the semantics metadata from the server, storing them locally in the system and analyzing the languages these services are written in.

The user now is required to take a decision based on the three choices provided by the chatbot (Figure 14).

The user can select the English (ENG) or the Greek (ELL) language, depending on the language the service in need is written in. If a user is not sure about the language of the service, then he/she can simply choose to skip this question.

Except of these choices, the user is also allowed to type another language in the textfield of the interface. In that case, the system will check whether the specified language matches to an existing service or not, and will inform the user about the result with an appropriate message (this is shown in the next scenario).

step 3

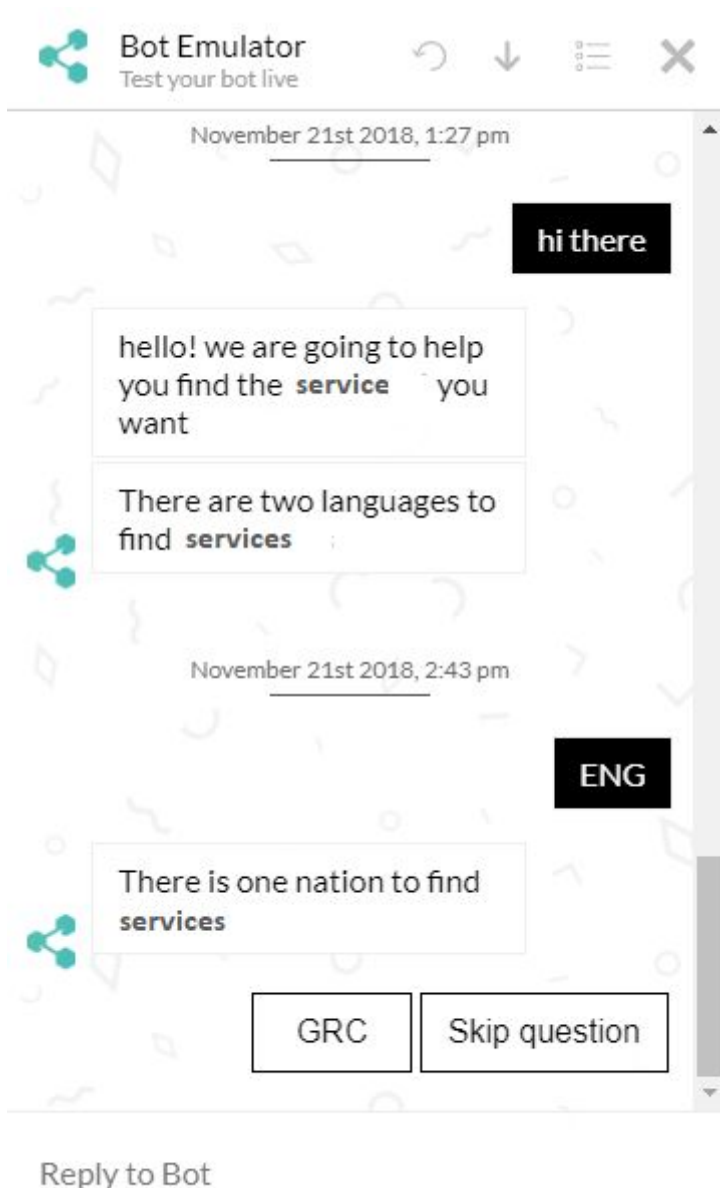


Figure 15. Question about nation with provided suggestions

In this case, the user clicked on 'ENG' button, so the system narrows the search to the services written in English (according to the language metadata semantic field). The chatbot then pops up a new question, asking the user to choose for which nation's services he/she is interested in.

The chatbot also provides suggestions regarding the nations that the services are referenced to (in the above case just one nation was found). In this scenario, the user chooses to skip the question. As a consequence, the system does not filter the services based on the nation metadata field (Figure 15).

step 4

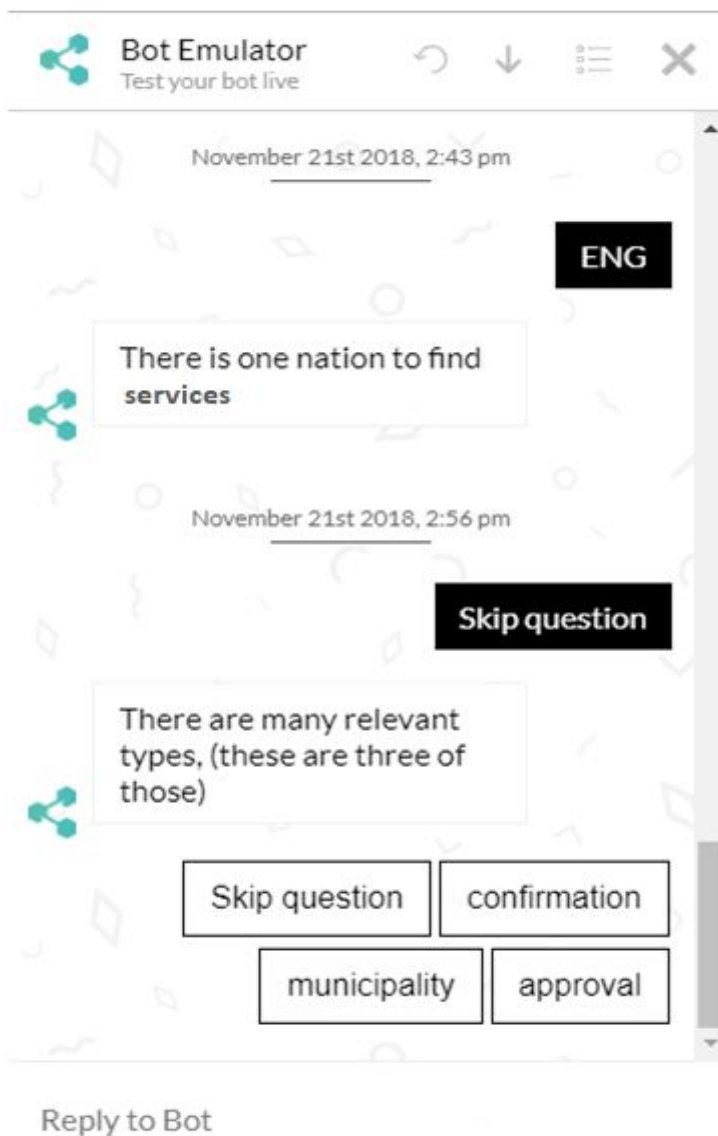


Figure 16. Question about type of service with provided suggestions

Afterwards, the bot provides the top three⁶ types of services that were found. The user can choose one of them, skip this question, or request another type of service (Figure 16).

step 5

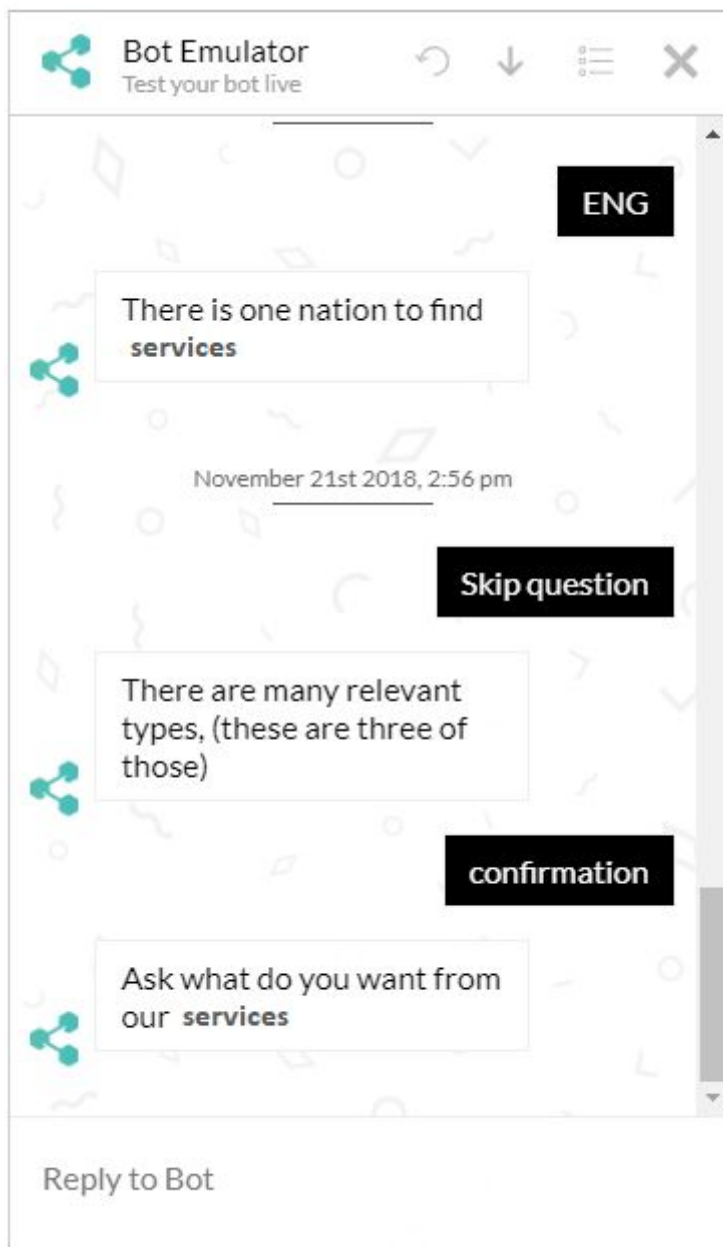


Figure 17. System asks user to search for the desired service

⁶ The top three services are formed based on the number of the available services for each type.

The user in this scenario, has chosen a service of type 'confirmation'.

The system based on user's choice, has limited the range of services to those related to confirmation. After this sequence of questions, the chatbot prompts the user to explicitly make a request of the desired service (Figure 17).

step 6

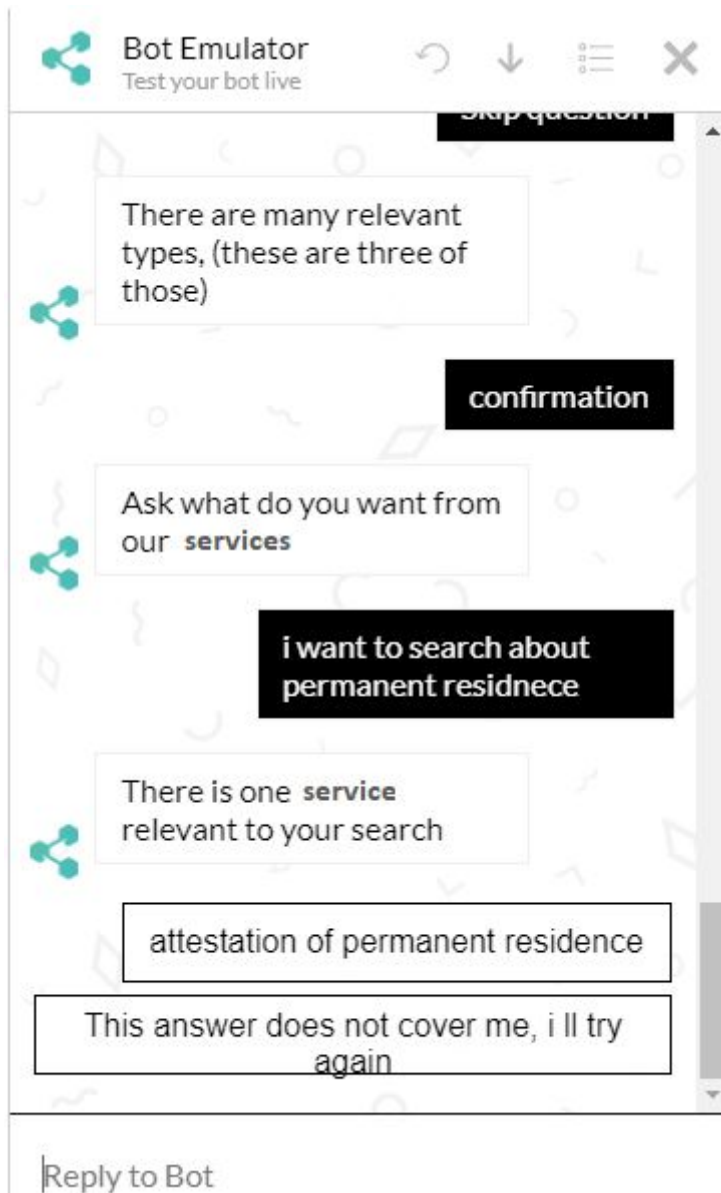


Figure 18. System provides response for the required service

The user typed "i want to search about permanent residence" in the input field. The system retrieved the services that match to this request and replied with a message

saying "There is one service relevant to your search". Additionally, the chatbot provides two buttons: one with the title of the retrieved service and one that indicates that the provided answer is not the desired one ("This answer does not cover me, i will try again") (Figure 18).

step 7

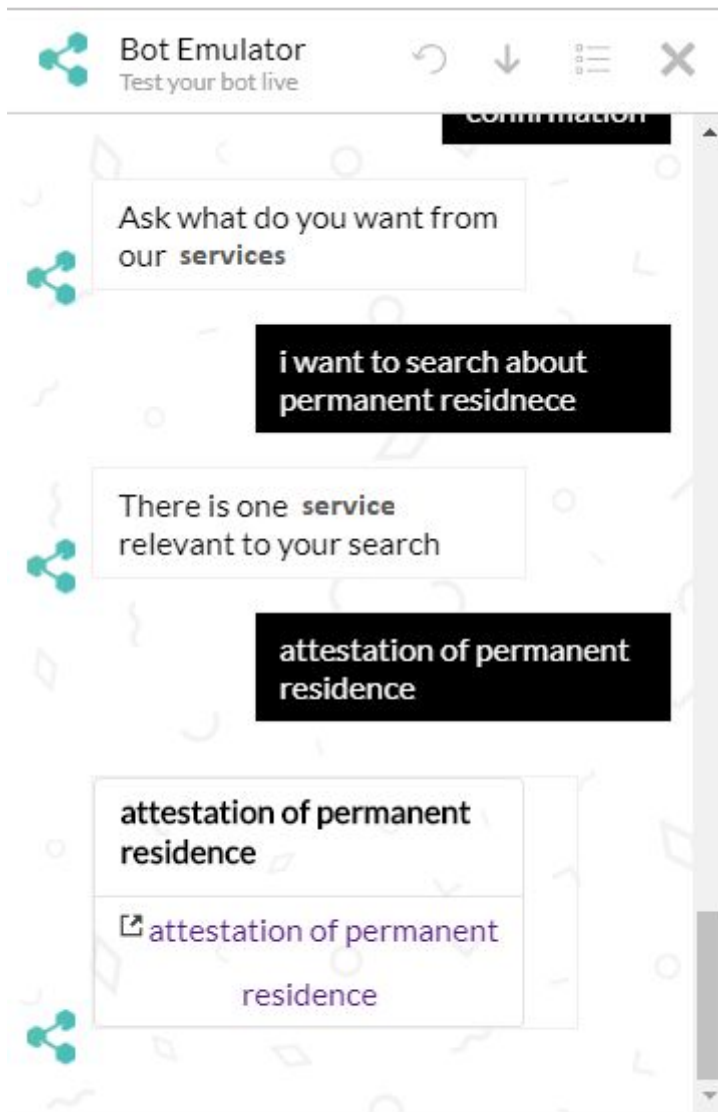


Figure 19. System provides link for the respective service

User clicked on 'attestation of permanent residence' button, therefore system displays the link of the selected service to the user to help the latter navigate to the web portal (Figure 19).

The screenshot shows the website interface for the 'Βεβαίωση μόνιμης κατοικίας' (Attestation of permanent residence) service. The page features a header with the Wikis logo and navigation links. The main content area includes a search bar, a list of 'Δικαιολογητικά' (Documents), a 'Σημειώσεις' (Notes) section, and 'Παρατηρήσεις σχετικά με την έκδοση βεβαίωσης μόνιμου κατοικίας' (Remarks regarding the issuance of permanent residence attestation). The 'Δικαιολογητικά' section lists requirements such as 'Αστυνομική ταυτότητα' (Police ID card), 'Αντίγραφο λογαριασμού Δ.Ε.Κ.Ο.' (Copy of DEKO account), and 'Αίτηση - Υπεύθυνη Δήλωση' (Application - Solemn Declaration). The 'Σημειώσεις' section notes that the 'Αστυνομική ταυτότητα' cannot be used as a sole proof. The 'Παρατηρήσεις' section provides additional instructions for the DEKO account and the applicant's status.

Figure 20. The page of the service in diadikasies.gr

Finally, the system opens the link in a new browser tab, after the user has clicked on the link (Figure 20).

Second scenario

Brief story:

As before, the chatbot is activated after a welcome message by a user. After that, the user types a request about Spanish language and the system replies with a negative response. Then, the user requests English language by typing a new message. The chatbot after matching the input to one of the available languages that the services are written in, provides two options: to accept the matching language or to try again. In the next suggestion, about nation, the user clicks on the provided button for Greece. Subsequently the system moves on to the last suggestion about the type of services. On this step, the user types a message about deletions and the bot replies negatively. So, the user this time chooses the provided button about municipality. Based on user input so far, the system has made the limitations on the range of services. Moving on, the chatbot asks the user to type the services he/she is looking for. The user asks for services relevant to receipts, and similarly as before, the chatbot presents two options. The first option is to accept the matching title of the found service, and the second option is to try again with a new request. Finally, as the user accepts the found match, the chatbot provides the link for that service in a new browser tab.

step 1

First of all, the user resets the previous conversation and initiates a new one by typing "hi", as depicted in figure 21. Similarly to the previous scenario, the chatbot welcomes the user and provides the options to select a language.

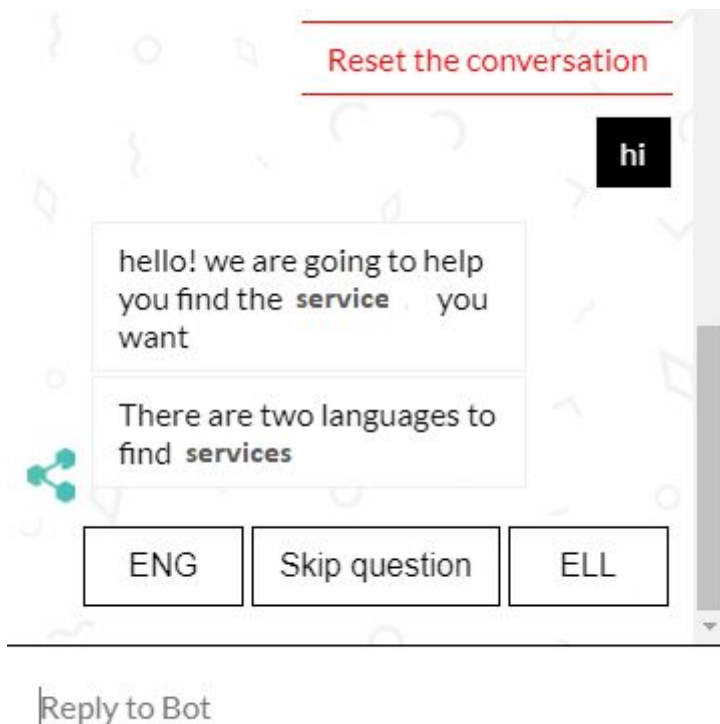


Figure 21. Chatbot interface provides suggestions for language

step 2

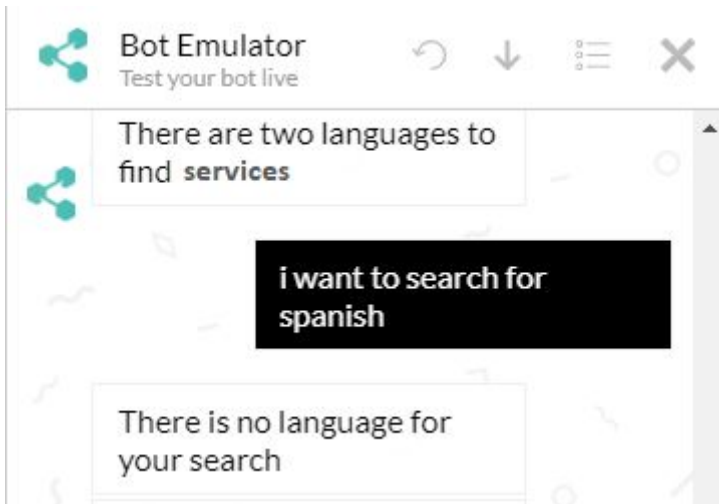


Figure 22. System asks to user's custom request

The user makes a request for services in Spanish language, however, since there are no services written on this language, the system replied with a message saying "There is no language for your search" (Figure 22).

step 4

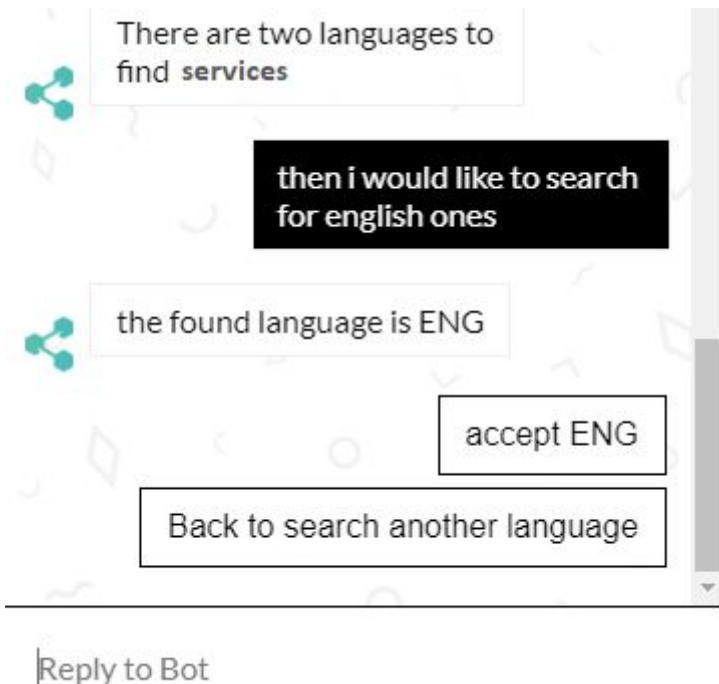


Figure 23. System's suggestions on user's custom request

Then, the user makes a new request regarding services in the English language.

The system checked if the requested language is a match to at least one of the services. Having found a match to the English language the system informs the user and provides two options: either to accept the English language or make a new search regarding the language of the services (figure 23).

step 5



Figure 24. System provides suggestions for nation

The user has accepted the English language, therefore the system proceeds to the next step , asking about the nation that the services are related to (Figure 24).

step 6



Figure 25. System provides suggestions for type of service

Similar to the previous scenario, the user clicks on 'GRC' button as he/she wants to search for services related to Greece. Sequentially, the bot moves to the next step indicating that there many types of services that are related to Greece. The user can either select one of the options provided or type what is needed (Figure 25).

step 7

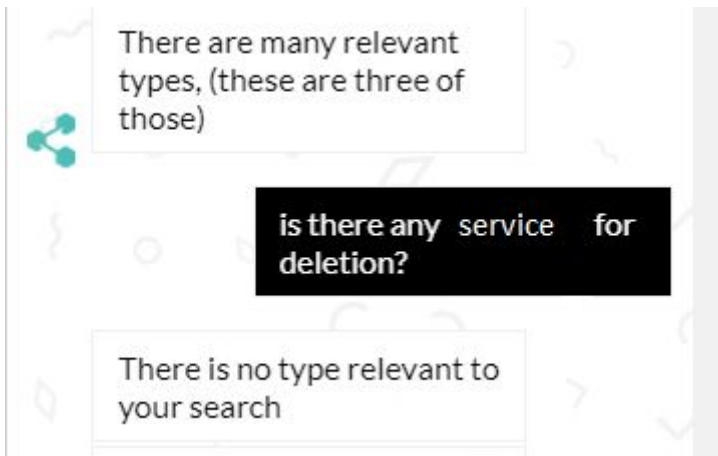


Figure 26. System responds negatively in user's request

In this case, the user asked for services relative to deletion, but the system did not find any respective match. Hence, the chatbot replies negatively and displays the same results as before (Figure 26).

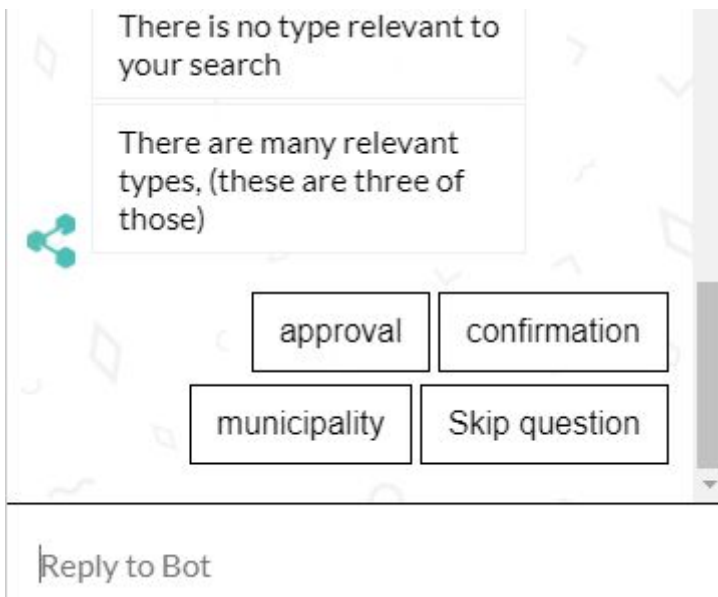


Figure 27. System provides suggestions-buttons for type of service

step 8

In this scenario, the user selects the 'municipality' button, and then the chatbot displays the message "ask what do you want from our documents", as illustrated in the following figure 28.



Reply to Bot

Figure 28. System asks user to type about desired service

step 9



Reply to Bot

Figure 29. System provides suggestions for requested service

The user makes a request regarding services that reference to receipts. After a thorough search, the chatbot found one service relevant to receipts and provided two options: one to accept the match found , and one to search again (Figure 29).

step 10

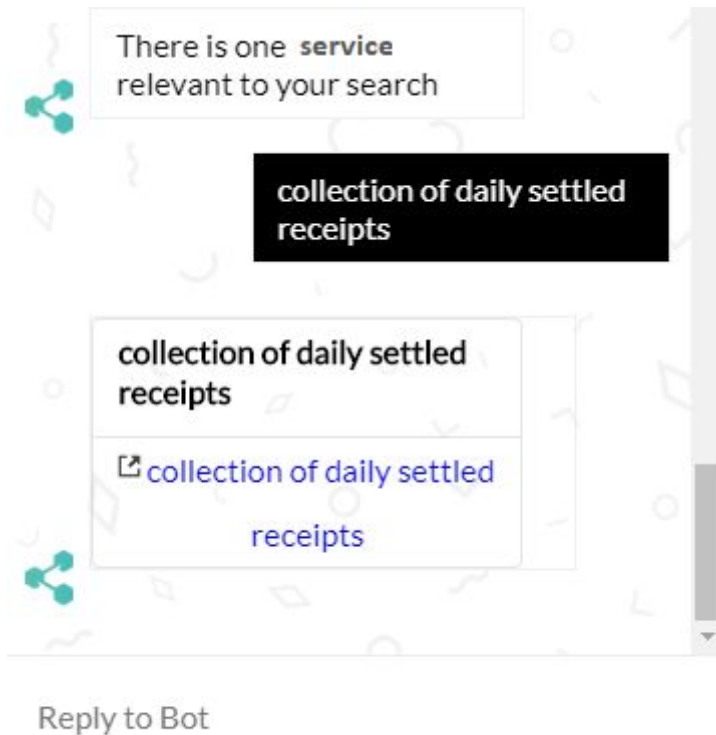


Figure 30. System provides link for the respective service

The user accepts the found service and consequently the chatbot provides the link and the title of the service to the user (Figure 30).

step 11

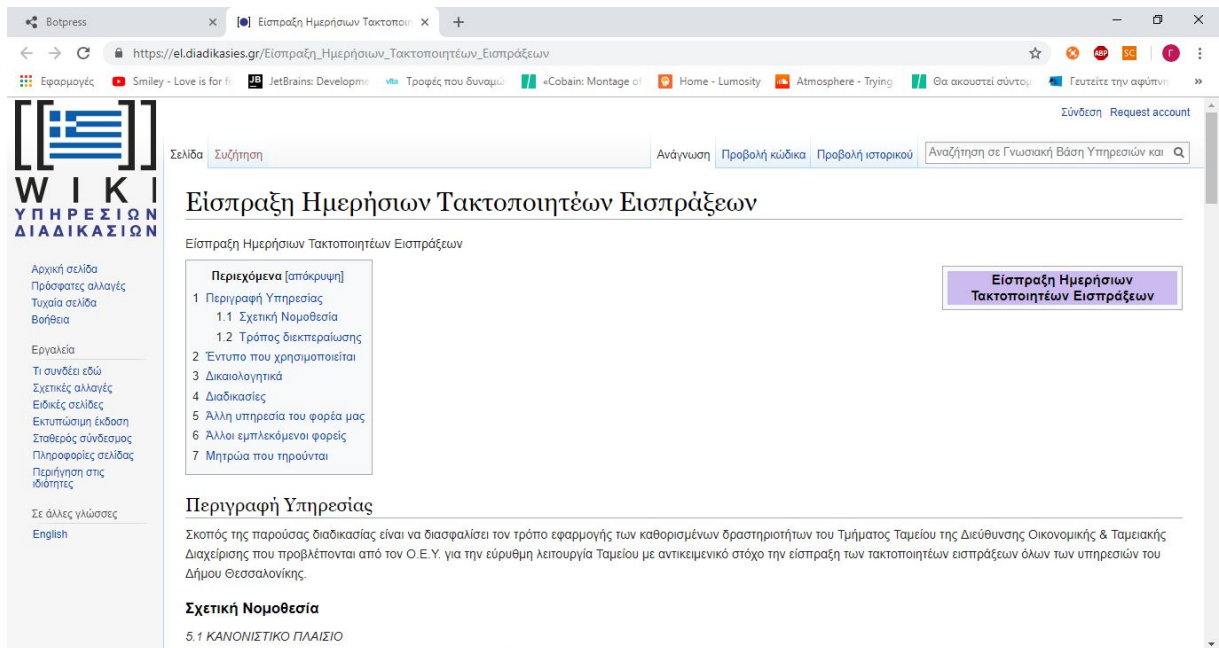


Figure 31. The web page for the respective service in diadikasies.gr

Finally, this is the web page where the link leads the user to, after the user has clicked on the provided link (Figure 31).

5 Results

In this chapter, the objectives (1.2 Objectives) that were achieved are presented, and the research questions that were defined in chapter 2 (paragraph 2.9 Towards Research Questions) are sufficiently answered.

5.1 Objectives achieved

After implementing the system according to the needs that are stated in this thesis about chatbot systems for public services, it is evident that the following objectives were accomplished.

- Successful implementation of a prototype Chatbot system that uses semantic information of Public services in order to assist a citizen-user in finding the desired public services.

The proper operation of the system is proved in chapter 4.

- Semantic metadata (semantics) were based on the C.P.S.V standard. Five indicative public services from “diadiaksies.gr” web portal were annotated with the use of the ‘public service description creator’ tool.
- Chatbot is developed based on the open source Botpress framework that is developed in Javascript. After examining several chatbot frameworks, we ended up that this was the most suitable framework.

- The system is capable of providing information about services written in other languages. (in our case, four out five services were written in English, and one was written in greek).
- It was proved that the botpress framework is a suitable and extendable framework for the development and implementation of a chatbot system for public services. this was also proved throughout the demonstration of a proof of concept for the system.
- A critical advantage of the developed Chatbot is that it can retrieve data on real time. This is important because, if this system is going to be eventually used by the web portal in the future, it can and will retrieve always updated data, i.e. any change or any new added service that a public servant will provide to the server.
- It was also proved that the CPSV standard can be utilised by chatbot systems for public services, something that was not implemented before. This fact is crucial, since European Nation supports and promotes this standard (CPSV) among European countries in order to be manageable all these metadata by european authorities.

5.2 Answers to the Research Questions

RQ1: Is it possible to annotate easily and transparently already existing public services with semantics related to chatbots?

Easily and transparently means by using already existing standards and without intervening in the already existing public service descriptions.

RQ2: Is it possible to train a (preferably open source based) chatbot to help a citizen find a service according to public service metadata?

This possibility has to be demonstrated by a proof of concept implementation.

(research questions were formulated in 2.10 towards research questions)

Answer to the first question:

We achieved to annotate public services with semantics related to them by using already existing technology. Semantics for the public services that we annotated are based on C.P.S.V standard. By using a specific tool 'public service description creator', five indicative public services were annotated in order to prove the adequate operation of our system.

In addition, there were no intermediate changes or other external tools to accomplish this task as the provided tools for the C.P.S.V standard are suitable for annotating public services and using these metadata to a chatbot system.

As it is mentioned before, this is the first chatbot system to retrieve metadata for public services based on C.P.S.V standard and this is important, since European Nation promotes the use of this standard for all public authorities of the European Union

Answer to the second question:

Through the demonstration of the proof of concept (4.5 paragraph 'Proof of concept') for the chatbot system that we developed, it is evident that the system operates adequately. Two scenarios were deployed in the proof of concept showing how the system manages different actions by users in order to guide them properly to find the desired service.

Hence, it is definitely achievable to train a chatbot system to retrieve semantic information of Public services based on C.P.S.V standard, and utilize the retrieved data in order to help a citizen-user find the service he/she seeks for.

6 Conclusion and future work

This chapter summarizes our thesis's contributions and conclusions and outlines the directions for future work and research. It is divided in two sections; the first one depicts the contribution of the current work and the second one, indicates the potential future research that can be conducted as an extension of the current work.

6.1 Conclusion

Based on existing research it is evident that Chatbot are currently one of the best assistive technology. By utilising artificial intelligence and machine learning, chatbots can become very powerful and highly enhance user experience.

In this thesis, we addressed the problem of creating an automated web assistant for a web portal for public services having a conversational interface and a client-side architecture.

After comparing existing Chatbots, that were implemented focusing on public services, we identified several features that are still missing, such as retrieval of real time data, multilingual support, usage of CPSV standard, proper guidance and assistance to users in finding the desired services. Thus, we proposed and developed a state-of-the-art Chatbot system that integrates all the aforementioned features and is capable of fulfilling users' increased needs.

Furthermore, we stated the advantages of using C.P.S.V standard in order to annotate public services. Indeed, we used the C.P.S.V standard to annotate the public services. This was a critical objective for our dissertation, as there was no similar implementation of a chatbot system based on this standard.

And finally, after comparing existing chatbot frameworks we selected and used for the implementation of our Chatbot system the botpress framework. This decision was based on the unique features this framework provides ,i.e. flexibility, adaptability, community support, and on the fact that in comparison to other frameworks this is by far the most notorious.

Summarizing, after the demonstration of a proof of concept for the developed chatbot system, it is evident that the system operates as it is designed to do, and actually helps users in finding the desired services. Concluding, we adequately implemented a system that our research showed was necessary for efficient customer assistance for public services, since all the other existing chatbots that were examined, were not capable of satisfying user's needs properly.

6.2 Future work

In this section, we propose several possible steps that may be applied in this chatbot system, which can extend the existing scope of the application.

One of the most crucial features that could be added in the system is speech recognition. Speech recognition constitutes an emerging trend in modern applications and specifically in virtual assistants. It would be a great enhancement, regarding user experience, to implement speech recognition in an assistive oriented Chatbot system. As technology is improving, toolkits that are related to speech recognition are becoming more efficient and robust. Hence, a potential addition of speech recognition, would further boost the human computer interaction and fasten the time needed to accomplish the whole order process.

Moreover, the system could store user's preferences based on their previous searches, in order to provide popular suggestions. That way, if a relevant search occurs again, this will save a lot of time from users and make the system even more user-friendly.

Finally, since the system is adjustable because it is based on an open source framework (Botpress), it could be modified for other web portals or even other web applications just by training the chatbot system accordingly.

All these features are considered to be suitable for any kind of Chatbot systems, and would probably become necessary in the next years for such a system.

7 References and Bibliography

- [1] Shawar, B. A., & Atwell, E. (2007, April). Different measurements metrics to evaluate a chatbot system. In *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies* (pp. 89-96). Association for Computational Linguistics.
- [2] Følstad, A., Brandtzaeg, P. B., Feltwell, T., Law, E. L., Tscheligi, M., & Luger, E. A. (2018, April). SIG: Chatbots for Social Good. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (p. SIG06). ACM.
- [3] Skerrett, D. (2017). Seven Reasons Why Most Chatbots Launched in 2017 Are Dead on Arrival.
- [4] Fichter, D., & Wisniewski, J. (2017). Chatbots Introduce Conversational User Interfaces. *Online Searcher*, 41(1), 56-58.
- [5] Kucherbaev, P., Psyllidis, A., & Bozzon, A. (2017, August). Chatbots as conversational recommender systems in urban contexts. In *Proceedings of the International Workshop on Recommender Systems for Citizens* (p. 6). ACM.
- [6] Kopka, B. Theoretical aspects of using virtual advisors in public administration.
- [7] Lommatzsch, A. (2018, June). A Next Generation Chatbot-Framework for the Public Administration. In *International Conference on Innovations for Community Services* (pp. 127-141). Springer, Cham.8. On Using the Core Public Sector Vocabulary (CPSV) to Publish a "Citizen's Guide" as Linked Data
- [8] Gerontas, A., Tambouris, E., & Tarabanis, K. (2018, May). On using the core public sector vocabulary (CPSV) to publish a citizen's guide as linked data.

In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age* (p. 9). ACM.

[9] Klopfenstein, L.C., Delpriori, S., Malatini, S. and Bogliolo, A., 2017, June. The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (pp. 555-565). ACM.

[10] Fonte, F.A.M., Nistal, M.L., Rial, J.C.B. and Rodríguez, M.C., 2016, April. NLAST: A natural language assistant for students. In *Global 12 Engineering Education Conference (EDUCON), 2016 IEEE* (pp. 709-713). IEEE.

[11] Kumar, M.N., Chandar, P.L., Prasad, A.V. and Sumangali, K., 2016, December. Android based educational Chatbot for visually impaired people. In *Computational Intelligence and Computing Research (ICCIC), 2016 IEEE International Conference on* (pp. 1-4). IEEE.

[12] Vincze, J. and Vincze, J., 2017. Virtual reference librarians (Chatbots). *Library Hi Tech News*, 34(4), pp.5-8.

[13] Hsieh, S.W., 2011. Effects of cognitive styles on an MSN virtual learning companion system as an adjunct to classroom instructions. *Journal of Educational Technology & Society*, 14(2), p.161.

[14] Kerlyl, A., Hall, P., & Bull, S. (2007). Bringing chatbots into education: Towards natural language negotiation of open learner models. In *Applications and Innovations in Intelligent Systems XIV* (pp. 179-192). Springer, London.

[15] Rubin, V. L., Chen, Y., & Thorimbert, L. M. (2010). Artificially intelligent conversational agents in libraries. *Library Hi Tech*, 28(4), 496-522.

- [16] McNeal, M., & Newyear, D. (2013). Chatbots: automating reference in public libraries. In *Robots in Academic Libraries: Advancements in Library Automation* (pp. 101-114). IGI Global.
- [17] Allison, D. (2012). Chatbots in the library: is it time?. *Library Hi Tech*, 30(1), 95-107.
- [18] Kernaghan, K. (2012). Anywhere, Anytime, Any Device: Innovations In Public Sector Self-Service Delivery.
- [19] Tambouris, E. (2018). Using Chatbots and Semantics to Exploit Public Sector Information. *EGOV-CeDEM-ePart 2018*, 125.
- [20] Salem, S., Ojo, A., Estevez, E., & Fillottrani, P. R. (2018, September). Towards a Cognitive Linked Public Service Cloud. In *Working Conference on Virtual Enterprises* (pp. 430-441). Springer, Cham.
- [21] Fillottrani, P. R. (2018). Towards a Cognitive Linked Public Service Cloud *Collaborative Networks of Cognitive Systems*, 534, 430.
- [22] Bradesko, L., & Mladenic, D. (2012). A Survey of Chabot Systems through a Loebner Prize Competition.
- [23] "How do Chatbots work? A Guide to the Chatbot Architecture - Maruti Techlabs," Maruti Techlabs Pvt. Ltd., [Online]. Available: <https://www.marutitech.com/chatbots-work-guide-chatbot-architecture/>. [Accessed 14 9 2017]
- [24] S. A. Abdul-Kader and D. J. Woods, "Survey on Chatbot Design Techniques in Speech Conversation Systems," 2015.
- [25] M. d. G. B. Marietto, R. V. d. Aguiar, G. d. O. Barbosa, W. T. Botelho, E. Pimentel, R. d. S. França and V. L. d. Silva, "ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE: A BRIEF

TUTORIAL," 2013.

[26] Flanagan and David, JavaScript: The Definitive Guide, 6th Edition, O'Reilly Media, 2011.

[27] C. Yue and H. Wang, "Characterizing Insecure JavaScript Practices on the Web," 2009

[28] D. Yu, A. Chander, N. Islam and I. Serikov, "JavaScript Instrumentation for Browser Security," 2007.

[29] J. Cahn, "CHATBOT: Architecture, Design, & Development," 2017.

[30] A. Chia. 5 reasons to use the gov.sg bot, March 2017. Blog of Singapore Government, <https://www.gov.sg/news/content/5-reasons-to-use-the-gov-sg-bot>.

[31] <https://inteliwise.com/products/ai/egov-virtual-assistant/>

[32] Rossi, G., Schwabe, D., & Guimarães, R. (2001). Designing personalized web applications. *WWW*, 1, 275-284.

8 APPENDIX

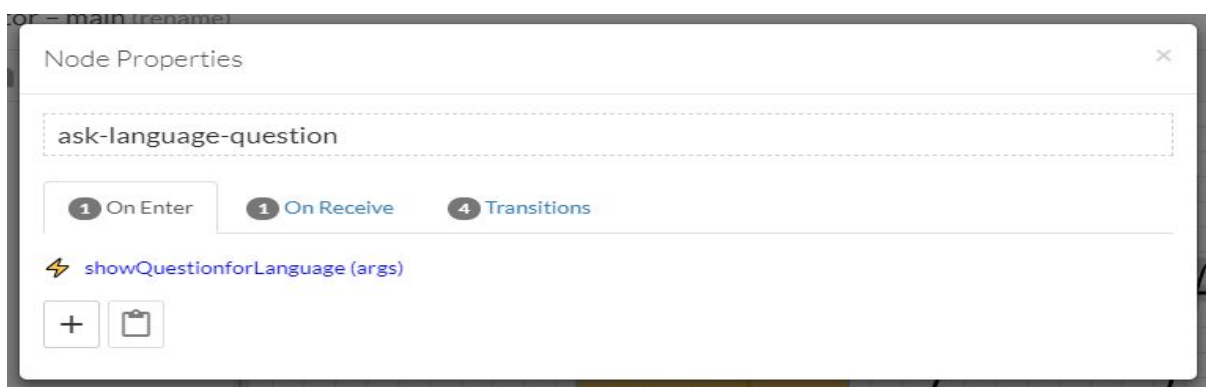
8.1 Flow Diagram of the system

The Flow diagram shows the pattern that the bot is going to follow for each case during execution. Botpress provides a tool or even better a platform, where developers can edit every part and every case of the program, and also view the whole flow of the program. The flow diagram specifies the execution process and structure of the bot, i.e. the operation flow of the bot. In this flow, and sequentially in the flow diagram, everything that is required and will be used has to be declared, i.e., functions, methods, texts, etc, specifying thus where, when and how things are going to be applied and used.

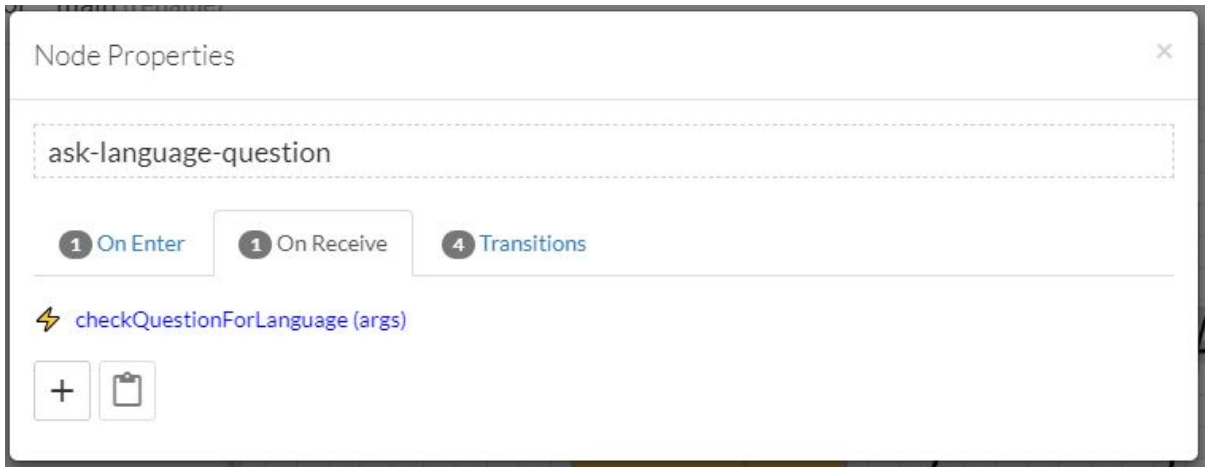
How to pass methods in node

By clicking on a node, an interface pops up which enables user to pass any method that exists in actions.js file or passing a text from the files in 'content data' folder, in our case these files are text.json and trivia.json. An example is given below on 'ask-language-question' node.

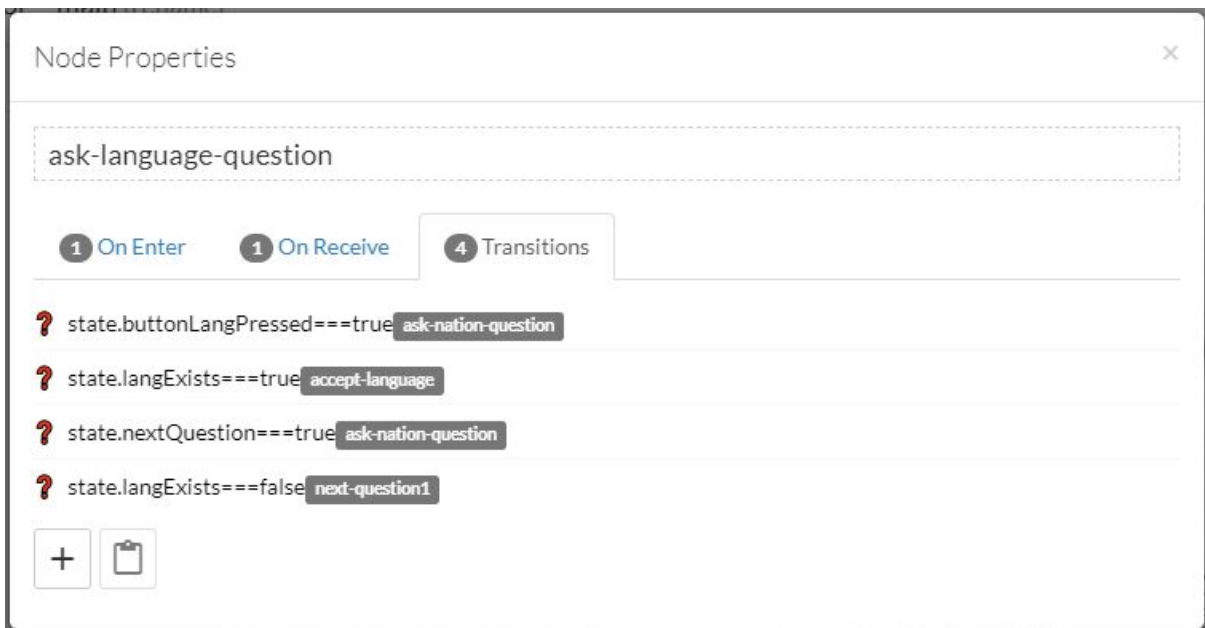
Here we clicked on 'ask-language-question' node and this is the shown interface



We have set showQuestionforLanguage method to be implemented while entering this node. However we can pass any method we want from actions.js and we can also pass multiple methods. In the same way we can pass text messages from the content data files.

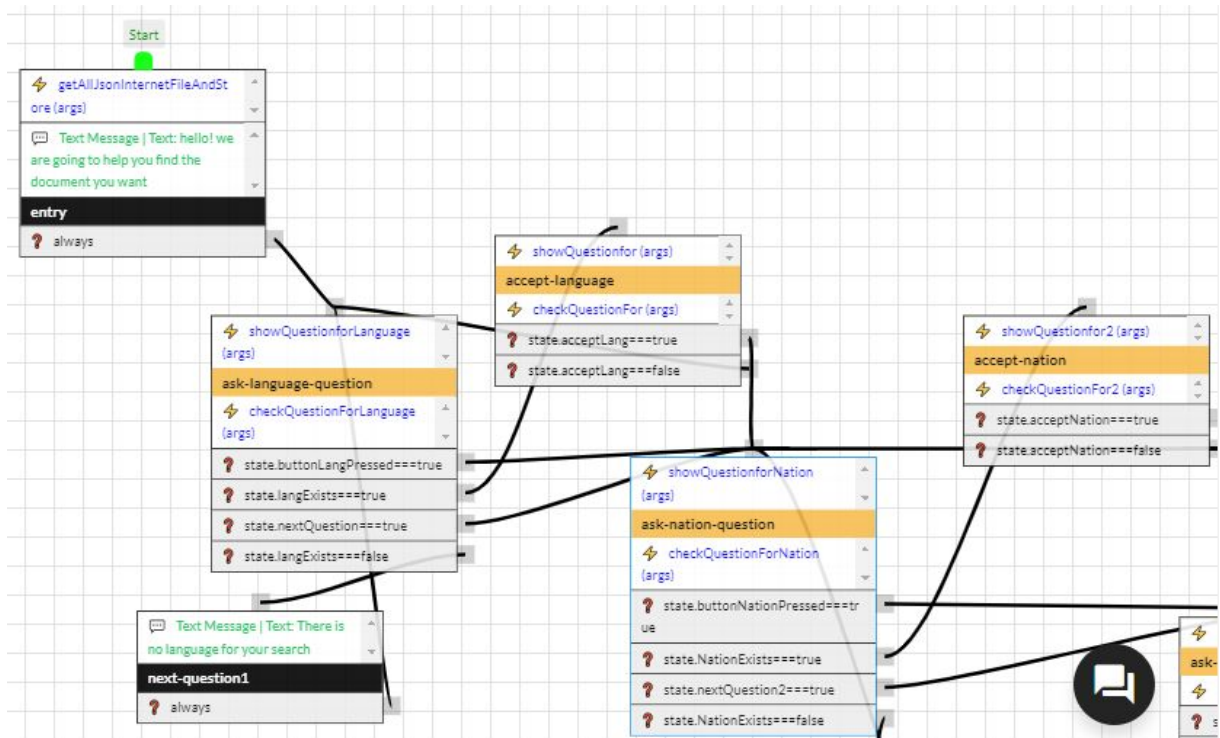


On receive panel, we have passed checkQuestionForLanguage method and as it mentioned before, multiple methods or text messages can be passed.



This is the transition panel where according to the state of the shown variables the flow will pass to the next nodes. In our case, the state of these variables and in all nodes is changing via the operation of the methods. More explanations will be given in the implementation chapter.

Diagram of our bot



First Node: starting

Text: "hello! we are going to help you find the document you want"

Method : getAllJsonInternetFileAndStore

getAllJsonInternetFileAndStore method requests the json file from the server and stores data to a specific array called allList. After executing the above method ,the bot shows a message to user saying, "hello! we are going to help you find the document you want".

Transition: uniuue

The transition occurs each time and it leads to the next node called 'ask-language-question'.

Second Node: ask-language-question

Methods: showQuestionForLanguage, checkQuestionForLanguage

one method while entering the node ,called 'showQuestionForLanguage' which shows a message to the user to press the desired button or inserting his/her own request.

one method while receiving a response from user, called 'checkQuestionForLanguage' which checks if the response from user matches with an element from the data of the array.

Transitions: 4

- 1) if a choice from the shown buttons is selected the flow moves to 'ask-nation-question' node.
- 2) if user's input matches with an element from the array it leads to 'accept-language' node.
- 3) if user selects 'skip question ' button it moves to 'ask-nation-question' node.
- 4) if user's input does not match with an element from the array it leads to 'next question' node wchich shows a message about this failure and returns the flow to the 'ask-language-question' again.

Third node: accept-language

Methods: showQuestionFor, checkQuestionFor

one method on entering node, called 'showQuestionFor' which addresses a question to the user, whether or not the user accepts the found element as a match to the specified input provided by the later.

one method on receiving input, called 'checkQuestionFor' which checks whether the user accepted or not the answer.

Transitions: 2

- 1) if user accepts the answer the flow moves to 'ask-nation-question' node.
- 2) if user rejects the answer the flow returns to 'ask-language-question' node.

Fourth node: ask-nation-question

Methods: showQuestionForNation, checkQuestionForNation,

one method while entering the node ,called 'showQuestionForNation' which shows a message to the user allowing either pressing the desired button of those provided or typing a new request.

one method while receiving a response from user, called 'checkQuestionForNation' which checks if the response from the user is a match to an element from the data of the array.

Transitions: 4

- 1) if a choice from the shown buttons is selected the flow moves to 'ask-type-question' node.
- 2) if user's input matches with an element from the array it leads to 'accept-nation' node.
- 3) if user selects 'skip question ' button it moves to 'ask-type-question' node.
- 4) if user's input does not match with an element from the array it leads to 'next question2' node which shows a message about this failure and returns the flow to the 'ask-nation-question' again.

Fifth node: accept-nation

Methods: showQuestionFor2, checkQuestionFor2

one method on entering node, called 'showQuestionFor2' which addresses a question to the user, whether or not the user accepts the found element as a match to the specified input provided by the later.

one method on receiving input, called 'checkQuestionFor2' which checks whether the user accepted or not the answer.

Transitions: 2

- 1) if user accepts the answer the flow moves to 'ask-type-question' node.
- 2) if user rejects the answer the flow returns to 'ask-nation-question' node.

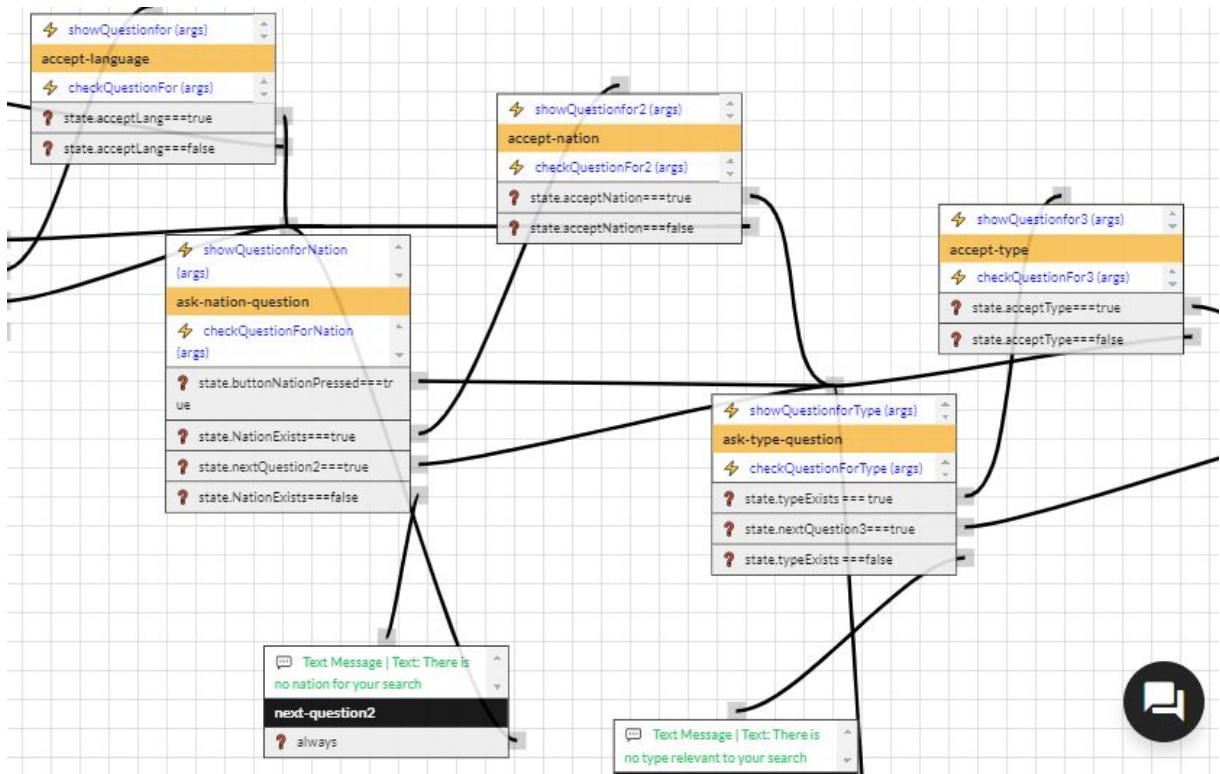
Sixth node: next-question1

Text: "There is no language for your search"

Transition: 1

The transition occurs each time and it leads to the previous node called 'ask-language-question'.

part 2



Seventh node: next-question2

Text: "There is no nation for your search"

Transition: 1

The transition occurs each time and it leads to the previous node called 'ask-nation-question'.

Eighth node: ask-type-question

Methods: showQuestionForType, checkQuestionForType

one method while entering the node ,called 'showQuestionForType' which shows a message to the user allowing either pressing the desired button of those provided or typing a new request.

one method while receiving a response from user, called 'checkQuestionForType' which checks if the response from user matches to an element from the data of the array.

Transitions: 4

- 1) if a choice from the shown buttons is selected the flow moves to 'ask-question' node.
- 2) if user's input matches with an element from the array it leads to 'accept-type' node.
- 3) if user selects 'skip question ' button it moves to 'ask-question' node.
- 4) if user's input does not match with an element from the array it leads to 'next question3' node which shows a message about this failure and returns the flow to the 'ask-type-question' again.

Ninth-node: accept-type

Methods: 'called showQuestionFor3, checkQuestionFor3

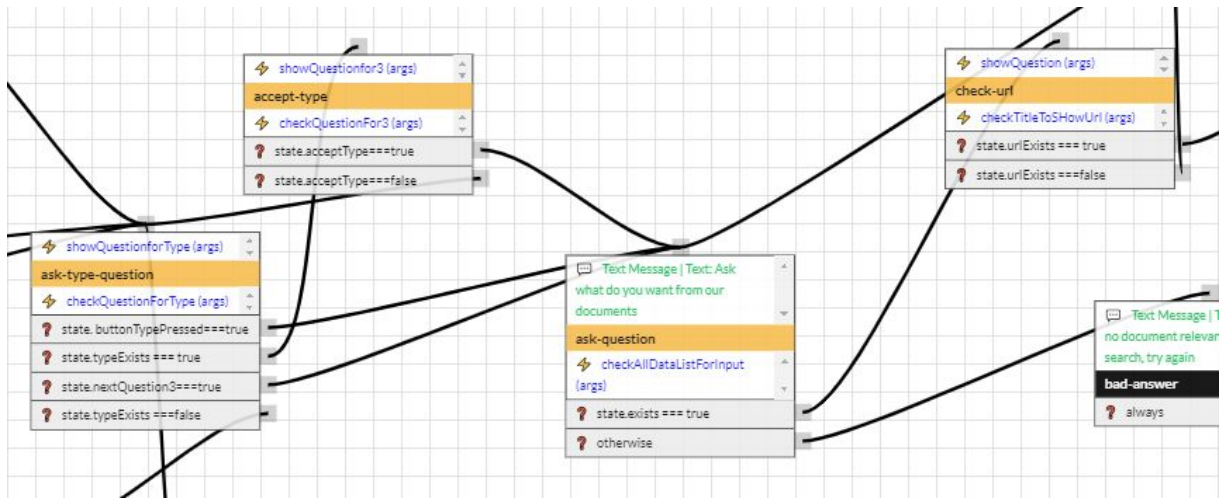
one method on entering node, 'called showQuestionFor3' which addresses a question to the user, whether or not the user accepts the found element as a match to the specified input provided by the later.

one method on receiving input, called 'checkQuestionFor3' which checks whether the user accepted or not the answer.

Transitions: 2

- 1) if user accepts the answer the flow moves to 'ask-question' node.
- 2) if user rejects the answer the flow returns to 'ask-type-question' node.

part 3



Tenth node: ask-question

Text: "Ask what do you want from our documents"

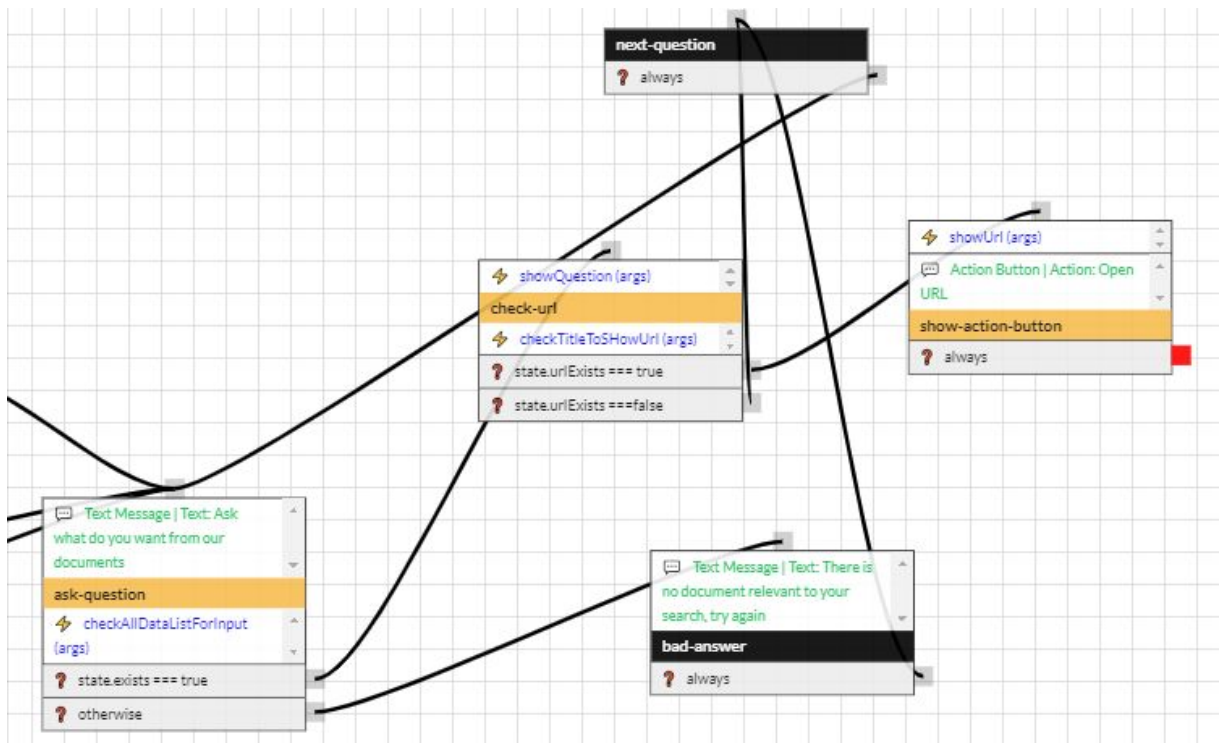
a message saying "Ask what do you want from our documents" to the user on entering the node.

Method: checkAllDataListForInput

one method called 'checkAllDataListForInput' to check if user's input matches with an element from the source data of the array.

Transitions: 2

- 1) It leads to 'check-url' node if the inserted data match with the source data.
- 2) Else the second condition leads to 'bad-answer' node.



Eleventh node: check-url

Method: showQuestions, checkTitleToShowUrl

one method on entering node, called 'showQuestions' which shows the relevant documents to the user's search.

A method on receiving user's input , called 'checkTitleToShowUrl' that checks user's choice on given documents and finding the respective url.

Transitions: 2

- 1) url of user's choice exists it leads to 'show-action-button' node.
- 2) Second condition returns to 'ask-question' node.

Twelfth node: show-action-button

Action button: Open URL

An action button has been set on entering node, its action is to provide a link and redirect user to the appropriate web location by opening a new tab.

Thirteenth node: bad-answer

Text: "There is no document relevant to your search, try again"

Transition: 1

The transition occurs each time and it leads to the 'next-question' node.

Fourteenth node: next question

Transition: 1

Always leading to 'ask-question' node

8.2 Installation of botpress and built-ins

first of all, botpress runs on node.js server so i installed node.js ,8.12.0 version for 64bit windows 10 to support its proper operation.

Then I followed the documentation of botpress to install install its 10.46 version as explained below.

1-step. open a command prompt to get to the directory i prefer to install it.

2-step. typing

```
npm install -g botpress
```

in command prompt to install botpress

3-step. creating a new directory to set bot in and then getting to this directory

```
mkdir bot
```

```
cd bot
```

4-step. initialization of bot by setting name of bot, version and name of author

```
botpress init
```

5-step.

```
npm install
```

to install all the dependencies.

6-step. Once the dependencies are installed, start the bot using:

```
npm start
```

After following these steps bot starts to load, it will take a couple of time for the first as it has to initiate data and dependencies.

When the loading process finishes, it shows a message (in green), saying that the bot is launhed.

As it is shown below

```

> botpress start
19:31:35 - debug: [DB Janitor] Added table "logs"
19:31:35 - info: Starting botpress version 10.37.1
19:31:35 - info: [DB Janitor] Started
19:31:35 - info: [Ghost Content Manager] (transparent) Initialized
19:31:35 - debug: [Ghost Content Manager] (transparent) Added root folder media, doing nothing.
19:31:35 - debug: [Ghost Content Manager] (transparent) Added root folder src\flows, doing nothing.
19:31:35 - info: [Skills] Initiated
19:31:39 - verbose: [Renderers] Enabled for webchat.
19:31:39 - info: Loaded @botpress/channel-web, version 10.37.1
19:31:39 - info: Loaded 1 modules
19:31:39 - info: [Skills] Loaded 0 skills
19:31:39 - debug: [Ghost Content Manager] (transparent) Added root folder src\content_data, doing nothing.
19:31:39 - debug: Loading data for text from text.json
19:31:39 - info: Read 8 item(s) from text.json
19:31:39 - debug: Loading data for trivia from trivia.json
19:31:39 - info: Read 24 item(s) from trivia.json
19:31:39 - debug: Loading middleware: rendering.instrumentation
19:31:39 - debug: Loading middleware: hear
19:31:39 - debug: Loading middleware: webchat.sendMessagees
19:31:39 - debug: Loading middleware: fallback
19:31:44 - debug: Loading data for builtin_text from builtin_text.json
19:31:44 - debug: Loading data for builtin_image from builtin_image.json
19:31:44 - info: Bot launched. Visit: http://localhost:3000
19:31:44 - debug: Loading data for builtin_single-choice from builtin_single-choice.json
19:31:44 - debug: Loading data for builtin_card from builtin_card.json
19:31:44 - debug: Loading data for builtin_action-button from builtin_action-button.json
19:31:44 - info: Read 1 item(s) from builtin_action-button.json
19:31:44 - debug: Loading data for builtin_carousel from builtin_carousel.json
19:31:44 - debug: Loading data for builtin_raw from builtin_raw.json
19:32:35 - debug: [DB Janitor] Running tasks
19:32:35 - debug: [DB Janitor] Running for table "logs"
19:33:35 - debug: [DB Janitor] Running tasks
19:33:35 - debug: [DB Janitor] Running for table "logs"

```

From now on, user can navigate in the botpress by opening a browser and passing the link that is provided on the message, <http://localhost:3000>.

XMLHttpRequest built-in

XMLHttpRequest built-in was installed in botpress in order to send requests to the server to retrieve the required data.

1-step. Typing in command line :

```
npm i xmlhttprequest
```

After restarting botpress the built-in methods and classes could be used properly.

8.3 Implementation of methods

The project was implemented in Javascript Language, as botpress is also written on this language, Thus i did not take it into consideration to use any other language. Except this, there are some fundamental files of JSON format, where most of the texts and messages that are displayed on the interface are stored.

Botpress's structure is very easy to be used by a developer and even by a user with limited knowledge on programming. The most basic files to be modified by a developer are, 'actions.js' , 'index.js' and 'renderers.js',all three of them, written in Javascript. On the other hand, subfolders 'content' and 'flaw' are in JSON format and consist of text messages and information about each node respectively.

All methods that are going to be used, are declared in the 'actions.js' file. In order for the flow diagram to use each one, to the appropriate nodes while entering a node or on receiving an input inside the node.

Another crucial file is 'index.js', where all buit-in methods and renderers are registered and in the same time flaw diagram's event llistener is declared.

Finally , in 'renderers.js' file, all methods related to the rendering of the text messages are declared.

In this chapter, we are going to focus to the methods used in our bot ,in order to cover all the given requirements. Hence we will explain in details , 'actions.js' file, step by step.

explanation of installed built-ins

```
// i create an object called xhr with XMLHttpRequest constructor to handle the response from the server
var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
var xhr = new XMLHttpRequest();
const _ = require('lodash')
// this value is used to make the request from the server to get the necessary data.
var syncRequest = require('sync-request');
```

xhr variable is used handle the response from server. It is an object created by the XMLHttpRequest constructor.

_ is a variable to shuffle the answers/buttons of the interface.

syncRequest variable is declared in order to make the request for the necessary data to the server.

explanation of global variables Lists and global variables

```
//in this array i pass the data from the documents that are matching with use
var temporaryList = [];
//a list with not important words that its input should be cleared
const axrhstesLexeis = ["how", "ho", "why", "wy", "what", "for", "is", "are", "will",
//all data that are being pulled from server in order to be handled by the bo
var allList= [];

//in this array all distinct languages are passed in order to show them as po
var listWithLanguagesToShow = [];
//in this array all distinct nations are passed in order to show them as poss
var listWithNationToShow = [];
//in this array all distinct types of documents are passed in order to show t
var listWithTypeToShow = [];
//this is the first array after user limiting data by his/her choice on langu
var afterLanguageList = [];
//this is the second array after user limiting data by his/her choice on nati
var afterNationList = [];
//this is the third array after user limiting data by his/her choice on type
var afterTypeList = [];

var checkTitleList = []|

//these three variables keep the matched language,nation and type with user's
var langu = "";
var nat = "";
var typ = "";
```

all these lists are arrays and here is the declaration for all of them. A brief explanation is given on each method.

explanation of methods that are used in the flow diagram

all methods that flow diagram can use are declared inside ,module.exports brackets in order to be used by another Javascript file.

'getAllJsonInternetFileAndStore' method

```
module.exports = {  
  // this method requests and stores the data from the server to allList array in order to be managed from the botpre  
  getAllJsonInternetFileAndStore: async (state, event) => {  
    // clearing the list in case user resets the conversation with botpress, to avoid requesting data multiple times  
    allList = [];  
    //The await expression causes async function execution to pause until a Promise is resolved,  
    //that is fulfilled or rejected, and to resume execution of the async function after fulfillment.  
    //When resumed, the value of the await expression is that of the fulfilled Promise  
    // await code here  
    var result = await getDataFromServerToText ();  
    // code below here will only execute when await getDataFromServerToText() finished loading  
    makeArrayFromText (result);  
    return {  
      ...state, // we clone the existing state  
      count: 0, // we then reset the number of questions asked to `0`  
      score: 0 // and we reset the score to `0`  
    }  
  },  
}
```

allList is being reset in case ,the user resets a previous conversation, to avoid duplications.

In variable result, which the function waits to be completed (await), all data from server are being stored.

'getDataFromServerToText' method gets that request to text.

```
//it handles data according to their type from xhr response  
function readBody(xhr) {  
  var data;  
  if (!xhr.responseType || xhr.responseType === "text") {  
    data = xhr.responseText;  
  } else if (xhr.responseType === "document") {  
    data = xhr.responseXML;  
  } else {  
    data = xhr.response;  
  }  
  return data;  
}  
  
// this function returns a promise object ,that means it has to  
function getDataFromServerToText () {  
  return new Promise(function (resolve) {  
    xhr.onreadystatechange = function() {  
      if (xhr.readyState == 4 && xhr.status == 200) {  
        resolve(readBody(xhr));  
      }  
    }  
  })  
}  
xhr.open('GET', getRequestAPI() , true);  
xhr.send(null);  
});  
}]
```

Then , makeArrayFromText (result) call, retrieves that text and makes an array with each json file title.

```
function makeArrayFromText (r) {
  var txt = [];
  var txt2 = [];
  var txt3 = [];
  txt = [];
  txt = r.split(".json\\>");
  txt.shift();
  txt2 = [];
  for (var i=0; i<txt.length; i++){
    txt2.push(txt[i].split("</a>"));
  }
  txt3 = [];
  for (var j=0; j<txt2.length; j++) {
    txt3.push(txt2[j][0].trim());
  }
  requestForEachJson(txt3);
}
```

```
function requestForEachJson (list) {
  var res = [];
  res = [];
  for (var i=0; i<list.length; i++){
    var request = getRequestAPI().concat(list[i]);
    res.push(JSON.parse(syncRequest('GET', request).body));
  }
  //console.log(res);
  for (var z=0; z<res.length; z++) {
    var kathejson = res[z];
    var j = kathejson[3];
    var url = j["@id"];
    var title = j["http://purl.org/dc/terms/title"][0]["@value"];
    var description = j["http://purl.org/dc/terms/description"][0]["@value"];
    var keywords = [];
    var k = j["http://www.w3.org/ns/dcat#keyword"]
    for (var w=0; w<k.length; w++) {
      keywords.push(k[w]["@value"]);
    }
    var lang = j["http://purl.org/dc/terms/language"][0]["@id"].substring(58);
    var nation = j["http://purl.org/dc/terms/spatial"][0]["@id"].substring(55);
    alllist.push({"title": title,"url": url,"description": description,"keywords": keywords,"lang": lang,"nation": nation });
  }
}
```

'requestForEachJson' method makes a request for every json file in the server and pass the data formatted in allList array. This array has all the metadata needed for the documents. Finally, 'getAllJsonInternetFileAndStore' method returns the state of the flow. It was declared as async in order to 'wait' the data on the await call.

'showQuestionforLanguage' method

```
showQuestionforLanguage: async (state, event) => {  
  
  var countlang=0;  
  var listl=[];  
  listWithLanguagesToShow = [];  
  
  for (i=0; i<allList.length; i++) {  
    listl.push(allList[i].lang);  
  }  
  //s auth th lista krataw mono distinct values  
  listWithLanguagesToShow = [...new Set(listl)];  
  countlang = listWithLanguagesToShow.length;  
  
  if (countlang==1) {  
    event.user.lang1 = listWithLanguagesToShow[0];  
    const messageSent = await event.reply('#!trivia-2erwa3');  
  } else if (countlang==2) {  
    event.user.lang1 = listWithLanguagesToShow[0];  
    event.user.lang2 = listWithLanguagesToShow[1];  
    const messageSent = await event.reply('#!trivia-qf4w5b');  
  } else if (countlang==3) {  
    event.user.lang1 = listWithLanguagesToShow[0];  
    event.user.lang2 = listWithLanguagesToShow[1];  
    event.user.lang3 = listWithLanguagesToShow[2];  
    const messageSent = await event.reply('#!trivia-qr4e2e');  
  } else if (countlang>3) {  
    event.user.lang1 = listWithLanguagesToShow[0];  
    event.user.lang2 = listWithLanguagesToShow[1];  
    event.user.lang3 = listWithLanguagesToShow[2];  
    const messageSent = await event.reply('#!trivia-ar4d7w');  
  }  
  
  return {  
    ...state  
  }  
},
```

This method also declared as asynchronous ,having two parameters ,the state of the conversation and the event that has occurred before. It stores in an array called 'listWithLanguagesToShow' all the distinct languages found in documents, and according to array's length it provides the respective message with its buttons.

'checkQuestionForLanguage' method

part A

```
checkQuestionForLanguage: async (state, event) => {  
  var langExists = false;  
  langu = "";  
  var nextQuestion = false;  
  var buttonLangPressed = false;  
  // mhdenizw th lista se periptwsh pou kanw xanagurisw sthn idi  
  afterLanguageList = [];  
  
  if (event.text.length<3) {  
    langExists = false;  
  } else {  
    //spaw th protash se strings mesa se ena array  
    var substringList = event.text.split(" ");  
  
    //tha chekarw kathe string apto array an einai mono ena gramma  
    for (var c=0; c<substringList.length; c++) {  
      //afairw apo osa strings exoun to "?"  
      if (substringList[c].includes("?")) {  
        substringList[c] = substringList[c].replace('?', '');  
      }  
      //afairw apo osa strings exoun to "."  
      if (substringList[c].includes(".")) {  
        substringList[c] = substringList[c].replace(/./g, '');  
      }  
      //afairw apo osa strings exoun to ","  
      if (substringList[c].includes(",")) {  
        substringList[c] = substringList[c].replace(/,/g, '');  
      }  
  
      //chekarw poies lexeis apo to input einai idies me tou list  
      for (var y=0; y<axrhstesLexeis.length; y++) {  
        if (substringList[c]==axrhstesLexeis[y]) {  
          | substringList.splice(c, 1);  
        }  
      }  
    }  
  }  
}
```

part B

```

for (var i = 0; i < substringList.length; i++) {
  //kanw kathe lexh me kefalala k mono 3 grammata gia na kanei match me to lang p einai 3
  substringList[i] = substringList[i].toUpperCase();
  if (substringList[i].length > 3) {
    substringList[i] = substringList[i].substring(0,3);
  }
}
substringList.forEach(function(item, index, object) {
  if ((item.length > 3) || (item.length < 2) || (item === '') || (item === ' ')) {
    object.splice(index, 1);
  }
});
//if user's input matches with an element langExists gets true and langu gets the value
for (var e=0; e<allList.length; e++) {
  for (var w=0; w<substringList.length; w++) {
    if (allList[e].lang.includes(substringList[w])){
      langExists = true;
      langu = allList[e].lang;
    }
  }
}
}
}

```

part C

```

for (i=0; i<listWithLanguagesToShow.length; i++) {
  if (event.text==listWithLanguagesToShow[i]) {
    buttonLangPressed = true;
    for (var p=0; p<allList.length; p++) {
      if (allList[p].lang.includes(event.text)) {
        afterLanguageList.push({"title": allList[p].title,"url": allList[p].url,"description": allList[p].description,"keywords": allList[p].keywords,"lang": allList[p].lang});
      }
    }
  }
}
// If user presses 'Skip question' button, nextQuestion gets true so flow will move to showQuestionforNative
if (event.text=="Skip question") {
  nextQuestion = true;
}
return {
  buttonLangPressed,
  nextQuestion,
  langExists,
  langu
}
}
}

```

It is also an asynchronous method with the same parameters with the previous method, however this method takes the user's input and it is processing it. It splits each string ,removing commas, full stops ,question marks. It is also removing common words not helpful for the matching. And finally it checks three conditions, if there is any matching with the data from the allList array in language key, if any button indicating a language was clicked, and finally ,if 'Skip question' button was clicked.

'showQuestionfor' method

```

showQuestionfor: async (state, event) => {
  event.user.lang = langu;
  const messageSent = await event.reply('#!trivia-wex2r4');
  return {
    ...state
  }
},

```

'showQuestionfor' method will be called in the flow diagram only if user' input matches with a found language , to ask user for accepting it or not.

'checkQuestionFor' method

```

checkQuestionFor(state, event) {
  var acceptLang = false;

  if (event.text != "accept " + langu) {
    acceptLang = false;
  } else {
    for (var p=0; p=allist.length; p++) {
      if (allist[p].lang.includes(langu)) {
        acceptLang = true;
        afterlanguageList.push({"title": allist[p].title,"url": allist[p].url,"description": allist[p].description,"keywords": allist[p].keywords,"lang": all
      }
    }
  }
  return {
    acceptLang
  }
},

```

checks user's response on the previous message, if he/she accepts the found language, acceptLang variable turns to true in order to lead the flow to the next question for nation and in the same time limiting the array by removing elements with different language.

'showQuestionforNation' method


```

showQuestionforNation: async (state, event) => {
  var countNat=0;
  var listn=[];
  listWithNationToShow = [];

  //an sth prohgomeneh erwthsh pathsei o xrhsths NEXT QUESTION to
  if(afterLanguageList.length==0) {
    afterLanguageList = allList;
  }
  //edw pernaw ola ta dedomena gia lang apo th lista pou exw ola
  for (i=0; i<afterLanguageList.length; i++) {
    listn.push(afterLanguageList[i].nation);
  }
  //s auth th lista krataw mono distinct values
  listWithNationToShow = [...new Set(listn)];
  countlang = listWithNationToShow.length;

  if (countlang==1) {
    event.user.nat1 = listWithNationToShow[0];
    const messageSent = await event.reply('#!trivia-sz4dq3');
  } else if (countlang==2) {
    event.user.nat1 = listWithNationToShow[0];
    event.user.nat2 = listWithNationToShow[1];
    const messageSent = await event.reply('#!trivia-wz1sq5');
  } else if (countlang==3) {
    event.user.nat1 = listWithNationToShow[0];
    event.user.nat2 = listWithNationToShow[1];
    event.user.nat3 = listWithNationToShow[2];
    const messageSent = await event.reply('#!trivia-3wzad8');
  } else if (countlang>3) {
    event.user.nat1 = listWithNationToShow[0];
    event.user.nat2 = listWithNationToShow[1];
    event.user.nat3 = listWithNationToShow[2];
    const messageSent = await event.reply('#!trivia-f2ga4d');
  }
  return {

```

Its operation is similar with 'showQuestionforLanguage' method, while it is also storing distinct nation values in a respective array, called listWithNationToShow, and according to the length of the array, it shows the appropriate message with its choices.

'checkQuestionForNation' method

part A

```

checkQuestionForNation(state, event) {

    nat = "";
    var NationExists = false;
    var nextQuestion2 = false;
    var buttonNationPressed = false;
    // mhdenizw th lista se periptwsh pou kanw xanagurisw sthn idia
    afterNationList = [];

    if (event.text.length<3) {
        NationExists = false;
    } else {

        //spaw th protash se strings mesa se ena array
        var substringList = event.text.split(" ");

        //tha chekarw kathe string apto array an einai mono ena gramma
        for (var c=0; c<substringList.length; c++) {

            //afairw apo osa strings exoun to "?"
            if (substringList[c].includes("?")) {
                substringList[c] = substringList[c].replace('?', '');
            }
            //afairw apo osa strings exoun to "."
            if (substringList[c].includes(".")) {
                substringList[c] = substringList[c].replace(/./g, '');
            }
            //afairw apo osa strings exoun to ","
            if (substringList[c].includes(",")) {
                substringList[c] = substringList[c].replace(/,/g, '');
            }
            //diagraw to string an exei ena gramma moni
            if (substringList[c].length==1) {
                substringList.splice(c,1);
            }
        }
    }
}

```

part B

```

for (var y=0; y<axrhstesLexeis.length; y++) {
  if (substringList[c]==axrhstesLexeis[y]) {
    substringList.splice(c , 1);
  }
}

//kanw kathe lexh me kefalaia k mono 3 grammata gia na kanei match me to lang p
substringList[c] = substringList[c].toUpperCase();

if (substringList[c].length>3) {
  substringList[c] = substringList[c].substring(0,3);
}
}
substringList.forEach(function(item, index, object) {
  if ((item.length>3) || (item.length<2) || (item === '') || (item === ' ')) {
    object.splice(index, 1);
  }
});
//an sth prohgoumenh erwthsh pathsei o xrhsths NEXT QUESTION tote h lista tha nai
if(afterLanguageList.length==0) {
  afterLanguageList = allList;
}

for (var p=0; p<afterLanguageList.length; p++) {
  for (var w=0; w<substringList.length; w++) {
    if (afterLanguageList[p].nation.includes(substringList[w])) {
      NationExists = true;
      console.log("NationExists is " + NationExists)
      nat = afterLanguageList[p].nation;
    }
  }
}
}
}

```

part C

```

for (i=0; i<listWithNationToShow.length; i++) {
  if (event.text==listWithNationToShow[i]) {
    buttonNationPressed = true;

    for (var p=0; p<afterLanguageList.length; p++) {
      if (afterLanguageList[p].nation.includes(event.text)) {
        afterNationList.push({"title": afterLanguageList[p].title,"url": afterLanguageList[p].url,"description": afterLanguageList[p].description,"keywords": aft
      }
    }
  }
}

if (event.text=="Skip question") {
  nextQuestion2 = true;
}

return {
  buttonNationPressed,
  nextQuestion2,
  NationExists,
  nat
}
}
}

```

As said before for the 'showQuestionforLanguage' method, it also occurs with 'checkQuestionForNation' method. It is similar with the 'checkQuestionForLanguage' method. Its purpose is to check three conditions. Firstly, checking if the user's input matches with any value of the elements. The second condition is to check if the user clicked the 'Skip question' button and finally if he/she clicked on any suggested choice about nations and if so, limiting the range of documents.

'showQuestionfor2' , 'checkQuestionFor2' methods

```
showQuestionfor2: async (state, event) => {
  event.user.nation = nat;
  const messageSent = await event.reply('#!trivia-qrz1r6');
  return {
    ...state
  }
},

checkQuestionFor2(state, event) {
  var acceptNation = false;

  if (event.text !== "accept " + nat) {
    acceptNation = false;
  } else {
    for (var p=0; p<afterLanguageList.length; p++) {
      if (afterLanguageList[p].nation.includes(nat)) {
        acceptNation = true;
        console.log("acceptn is " + acceptNation)
        afterNationList.push({"title": afterLanguageList[p].title,"url": afterLanguageList[p].url})
      }
    }
    console.log(afterNationList);
    return {
      acceptNation
    }
  }
},
```

Operating in the same way as 'showQuestionfor' and 'checkQuestionFor' methods.
But for the case of nation on this step.

'showQuestionforType' method

```
showQuestionforType: async (state, event) => {

  var countType=0;
  var listt=[];
  listWithTypeToShow = [];

  //an sth prohgoumenh erwthsh pathsei o xrhsths NEXT QUESTION tote h lista
  if(afterNationList.length==0) {
    afterNationList = afterLanguageList;
  }
  //edw pernaw ola ta dedomena gia lang apo th lista pou exw ola ta data a
  for (var i=0; i<afterNationList.length; i++) {
    for (var j=0; j<afterNationList[i].keywords.length; j++) {
      listt.push(afterNationList[i].keywords[j]);
    }
  }
  //s auth th lista krataw mono distinct values
  listWithTypeToShow = [...new Set(listt)];
  countType = listWithTypeToShow.length;
  if (countType==1) {
    event.user.type1 = listWithTypeToShow[0];
    const messageSent = await event.reply('#!trivia-az2rq5');
  } else if (countType==2) {
    event.user.type1 = listWithTypeToShow[0];
    event.user.type2 = listWithTypeToShow[1];
    const messageSent = await event.reply('#!trivia-4as3e8');
  } else if (countType==3) {
    event.user.type1 = listWithTypeToShow[0];
    event.user.type2 = listWithTypeToShow[1];
    event.user.type3 = listWithTypeToShow[2];
    const messageSent = await event.reply('#!trivia-a3s13t');
  } else if (countType>3) {
    event.user.type1 = listWithTypeToShow[0];
    event.user.type2 = listWithTypeToShow[1];
    event.user.type3 = listWithTypeToShow[2];
    const messageSent = await event.reply('#!trivia-q7d41h');
  }
  return {
    ...state
  }
},
```

It also works in the same way as 'showQuestionforLanguage' and 'showQuestionforNation' ,but in this step, the bot provides options to the user about the type of the documents.

'checkQuestionForType' methods

part A

```
checkQuestionForType(state, event) {  
  typ = "";  
  var typeExists = false;  
  var nextQuestion3 = false;  
  var buttonTypePressed = false;  
  // mhdenizw th lista se periptwsh pou kanw xanagurishw sthn  
  afterTypeList = [];  
  
  if (event.text.length<3) {  
    typeExists = false;  
  } else {  
  
    //spaw th protash se strings mesa se ena array  
    var substringList = event.text.split(" ");  
  
    //tha chekarw kathe string apto array an einai mono ena gra  
    for (var c=0; c<substringList.length; c++) {  
  
      //afairw apo osa strings exoun to "?"  
      if (substringList[c].includes("?")) {  
        substringList[c] = substringList[c].replace('?', '');  
      }  
      //afairw apo osa strings exoun to "."  
      if (substringList[c].includes(".")) {  
        substringList[c] = substringList[c].replace(/./g, '');  
      }  
      //afairw apo osa strings exoun to ","  
      if (substringList[c].includes(",")) {  
        substringList[c] = substringList[c].replace(/,/g, '');  
      }  
  
      //diagrafw to string an exei ena gramma mono  
      if (substringList[c].length==1) {  
        substringList.splice(c,1);  
      }  
    }  
  }  
}
```

part B

```

for (var y=0; y<axrhstesLexeis.length; y++) {
    if (substringList[c]==axrhstesLexeis[y]) {
        substringList.splice(c , 1);
    }
}
}
substringList.forEach(function(item, index, object) {
    if ((item.length<2) || (item === '') || (item === ' ')) {
        object.splice(index, 1);
    }
});
//an sth prohgoymenh erwthsh pathsei o xrhsths NEXT QUESTION tote h lista
if(afterNationList.length==0) {
    afterNationList = afterLanguageList;
}

for (var p=0; p<afterNationList.length; p++) {
    for (var t=0; t<afterNationList[p].keywords.length; t++) {
        for (var w=0; w<substringList.length; w++) {
            if (afterNationList[p].keywords[t].includes(substringList[w])) {
                typeExists = true;
                typ = afterNationList[p].keywords[t];
            }
        }
    }
}
}
}

```

part C

```

for (i=0; i<listWithTypeToShow.length; i++) {
    if (event.text==listWithTypeToShow[i]) {
        buttonTypePressed = true;

        for (var p=0; p<afterNationList.length; p++) {
            for (var k=0; k<afterNationList[p].keywords.length; k++){
                if (afterLanguageList[p].keywords[k].includes(event.text)) {
                    afterTypeList.push({"title": afterNationList[p].title,"url": afterNationList[p].url});
                }
            }
        }
    }
}

if (event.text=="Skip question") {
    nextQuestion3 = true;
}
}
return {
    buttonTypePressed,
    typeExists,
    typ,
    nextQuestion3
}
},

```

'showQuestionfor3' , 'checkQuestionFor3' methods

```

showQuestionFor3: async (state, event) => {
  event.user.type = typ;
  const messageSent = await event.reply('#!trivia-ske4t2');
  return {
    ...state
  }
},

checkQuestionFor3(state, event) {
  var acceptType = false;

  if (event.text != "accept " + typ) {
    acceptLang = false;
  } else {

    for (var p=0; p<afterNationList.length; p++) {
      for (var t=0; t<afterNationList[p].keywords.length; t++) {
        if (afterNationList[p].keywords[t].includes(typ)) {
          acceptType = true;
          afterTypeList.push({"title": afterNationList[p].title, "url": afterNationList[p].url
        }
      }
    }
  }
  return {
    acceptType
  }
},

```

Operating exactly as 'showQuestionFor2' and 'checkQuestionFor2' methods. But for the case of type on this step.

'checkAllDataListForInput' method

part A


```

checkAllDataListForInput(state, event) {
  //trabaw to input tou xrhsth
  var inp = event.text;
  //spaw th protash se strings mesa se ena array
  var substringList = inp.split(" ");

  //tha chekarw kathe string apto array an einai mono ena gramma
  for (var c=0; c<substringList.length; c++) {

    //diagraw to string an exei ena gramma mono
    if (substringList[c].length==1) {
      substringList.splice(c,1);
    }
    //afairw apo osa strings exoun to "?"
    if (substringList[c].includes("?")) {
      substringList[c] = substringList[c].replace('?', '');
    }
    //afairw apo osa strings exoun to "."
    if (substringList[c].includes(".")) {
      substringList[c] = substringList[c].replace('.', '');
    }
    //afairw apo osa strings exoun to ","
    if (substringList[c].includes(",")) {
      substringList[c] = substringList[c].replace(',', '');
    }
  }

  //chekarw poies lexeis apo to input einai idies me tou list , a
  for (var x=0; x<substringList.length; x++) {
    for (var y=0; y<axrhstesLexeis.length; y++) {
      if (substringList[x]==axrhstesLexeis[y]) {
        substringList.splice(x , 1);
      }
    }
  }
}

```

part B

```

//deleting null values from substringList
substringList.forEach(function(item, index, object) {
  if ((item === '') || (item === ' ')) {
    object.splice(index, 1);
  }
});
//to thetw kathe fora arxika false , k uparxei kapoia lekh antistoixh me t input to k
var exists = false;

//an sth prohgomienh erwthsh pathsei o xrhsths NEXT QUESTION tote h lista tha nai ke
if(afterTypeList.length==0) {
  afterTypeList = afterNationList;
}

//chekarw apo to arxeio p tha exw ola ta dedomena, edw peiramatika to allDataList, an
for (var j=0; j<afterTypeList.length; j++) {
  for (var w=0; w<substringList.length; w++) {
    //if (allDataList[j].title.includes(inp)){
    if (afterTypeList[j].title.includes(substringList[w])){
      exists = true;
      temporaryList.push({"title": afterTypeList[j].title,"url": afterTypeList[j].url
    }
  }
}
return {
  ...state, // We clone the state
exists,
count: state.count + 1
}
},

```

This method, processes the user's input by splitting it and removing all unnecessary data firstly. Then it checks every substring if matches with any title of the remained documents and finally stores to another array, called temporaryList, all the elements that doing so.

'showQuestion' method

part A

```
showQuestion: async (state, event) => {  
  // i initialize them for each search to find how many answers have been  
  var countResponses = 0;  
  event.user.title1 = "";  
  event.user.title2 = "";  
  event.user.title3 = "";  
  event.user.title4 = "";  
  event.user.title5 = "";  
  event.user.title6 = "";  
  
  event.user.url = "";  
  
  countResponses = temporaryList.length;  
  
  if(countResponses==1) {  
    event.user.url = temporaryList[0].url.split("/").join("\\");  
    event.user.title1 = temporaryList[0].title;  
  
    const messageSent = await event.reply('#!trivia-U_3sZz')  
  } else if (countResponses==2) {  
    event.user.title1 = temporaryList[0].title;  
    event.user.title2 = temporaryList[1].title;  
    const messageSent = await event.reply('#!trivia-DEiIPP')  
  } else if (countResponses==3) {  
    event.user.title1 = temporaryList[0].title;  
    event.user.title2 = temporaryList[1].title;  
    event.user.title3 = temporaryList[2].title;  
    const messageSent = await event.reply('#!trivia-wImL3f')  
  } else if (countResponses==4) {  
    event.user.title1 = temporaryList[0].title;  
    event.user.title2 = temporaryList[1].title;  
    event.user.title3 = temporaryList[2].title;  
    event.user.title4 = temporaryList[3].title;  
    const messageSent = await event.reply('#!trivia-zsfQQm')
```

part B

```
    } else if (countResponses==5) {
      event.user.title1 = temporaryList[0].title;
      event.user.title2 = temporaryList[1].title;
      event.user.title3 = temporaryList[2].title;
      event.user.title4 = temporaryList[3].title;
      event.user.title5 = temporaryList[4].title;
      const messageSent = await event.reply('#!trivia-dd3RQE')
    } else if (countResponses>=6) {
      event.user.title1 = temporaryList[0].title;
      event.user.title2 = temporaryList[1].title;
      event.user.title3 = temporaryList[2].title;
      event.user.title4 = temporaryList[3].title;
      event.user.title5 = temporaryList[4].title;
      event.user.title6 = temporaryList[5].title;
      const messageSent = await event.reply('#!trivia-9Jn6J2')
    }

    checkTitleList = temporaryList;

    //mhdenizw xana to temporaryList
    temporaryList = [];

    return {
      ...state, // We clone the state
      isCorrect: true, //gia na trexei sunexeia!!!
      count: state.count + 1, // We increase the number of questions we
    }
  },
```

Depending on matching elements, bot via this method suggests the titles of documents that are relevant to the search.

'checkTitleToSHowUrl' method

```

checkTitleToShowUrl: async (state, event) => {
  var urlExists = false;

  for (var c = 0; c < checkTitleList.length; c++) {
    if (event.text==checkTitleList[c].title) {
      //edw pairnei to swsto index to rightPosition gia na emfanizei
      rightPosition = c;
      urlExists = true;
    }
  }
  return {
    urlExists,
  }
},

```

It checks that the user's input is on of the titles of the elements in the array and stores the index of that element to get its url in the next method.

'showUrl' method

```

showUrl: async (state, event) => {
  event.user.url = checkTitleList[rightPosition].url.split("/").join("\\");
  event.user.title7 = checkTitleList[rightPosition].title;

  checkTitleList = [];

  return {
    ...state
  }
},

```

After all, on this step it passes the according title and the url of the found document in the variables of the action button.

'render' method

```

render: async (state, event, args) => {
  if (!args.renderer) {
    throw new Error('Missing "renderer"')
  }

  await event.reply(args.renderer, args)
}

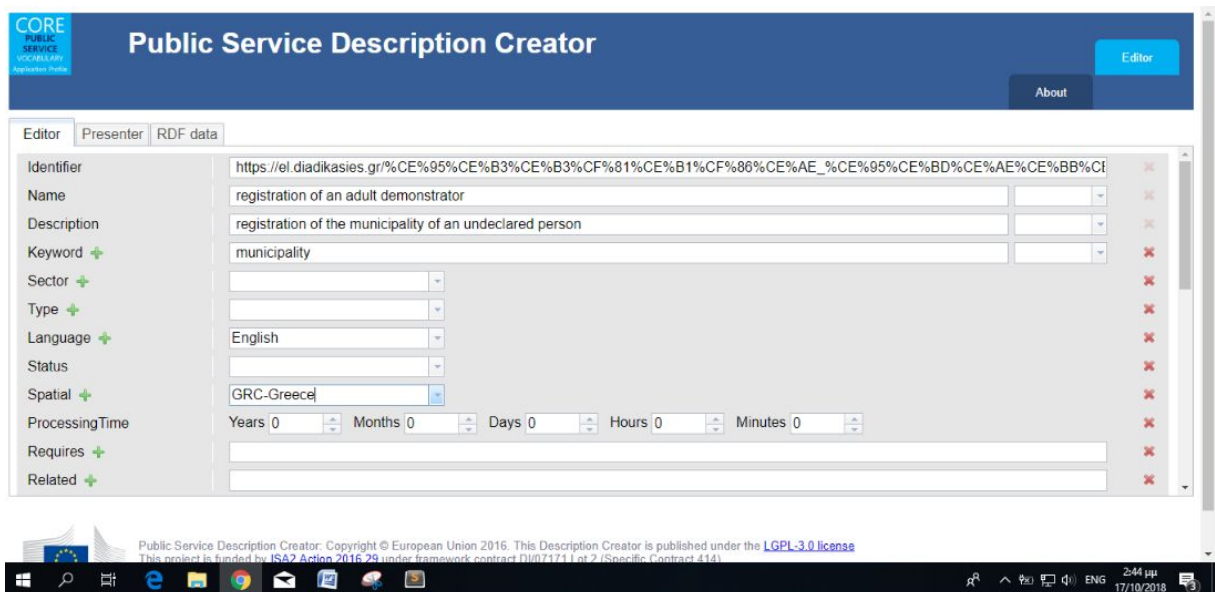
```

All messages that are stored in JSON files in the content_data subfolder, are rendered by this method to be shown on the interface.

8.4 Creation and storage of semantics data

step 1

This is the interface of the public service description creator that we work on, and as it is obvious the significant fields for our case are fulfilled, such as identifier, where we pass the url of each document, its name, description, keywords, language and nation.



The screenshot shows the 'Public Service Description Creator' web application. The interface includes a header with the 'CORE PUBLIC SERVICE VOCABULARY Application Profile' logo and a navigation bar with 'Editor', 'Presenter', and 'RDF data' tabs. The 'Editor' tab is active, displaying a form with the following fields:

- Identifier: https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%80%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CF
- Name: registration of an adult demonstrator
- Description: registration of the municipality of an undeclared person
- Keyword: municipality
- Sector: (empty dropdown)
- Type: (empty dropdown)
- Language: English
- Status: (empty dropdown)
- Spatial: GRC-Greece
- ProcessingTime: Years 0, Months 0, Days 0, Hours 0, Minutes 0
- Requires: (empty text field)
- Related: (empty text field)

At the bottom of the interface, there is a footer with the text: 'Public Service Description Creator. Copyright © European Union 2016. This Description Creator is published under the [LGPL-3.0 license](#). This project is funded by [ISA2 Action 2016.29](#) under framework contract D1/071711 and 21/Specific Contract 414.' The Windows taskbar at the bottom shows the date as 17/10/2018 and the time as 2:44 PM.

step 2

By selecting 'RDF data' button on the menu of the interface, automatically it provides all this information in RDF format, as shown below.

RDF formation provided by the tool

```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:ns1="http://www.w3.org/ns/dcat#">
    <rdf:Description
rdf:about="https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B
1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B9%CE%B
A%CE%BF%CF%85_%CE%91%CE%B4%CE%AE%CE%BB%CF%89%CF%84
%CE%BF%CF%85">
    <dcterms:identifier>https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81
%CE%B1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B9
%CE%BA%CE%BF%CF%85_%CE%91%CE%B4%CE%AE%CE%BB%CF%89%
CF%84%CE%BF%CF%85</dcterms:identifier>
    <rdf:type rdf:resource="http://purl.org/vocab/cpsv#PublicService"/>
    <dcterms:title>registration of an adult demonstrator</dcterms:title>
    <dcterms:description>registration of the municipality of an undeclared
person</dcterms:description>
    <ns1:keyword>municipality</ns1:keyword>
    <dcterms:language
rdf:resource="http://publications.europa.eu/resource/authority/language/ENG"/>
    <dcterms:spatial
rdf:resource="http://publications.europa.eu/resource/authority/place/GRC"/>
    </rdf:Description>
</rdf:RDF>

```

step 3

In this step, we pass this RDF data in the Json converter in order to do the conversion from RDF to Json data.

Input Data:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:ns1="http://www.w3.org/ns/dcat#">
  <rdf:Description
    rdf:about="https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81
%CE%B1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B
```

or Uri:

Input Format:

Output Format:

Raw output

Clear Submit

Output

Number of triples parsed: 7

Output

Number of triples parsed: 7

```
[{"@id": "http://publications.europa.eu/resource/authority/language/ENG"}, {"@id": "http://publications.europa.eu/resource/authority/place/GRC"}, {"@id": "http://purl.org/vocab/cpsv#PublicService"}, {"@id": "https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B9%CE%8A%CE%B8%CF%85_%CE%91%CE%B4%CE%AE%CE%B8%CF%89%CF%84%CE%B8%CF%85", "http://purl.org/dc/terms/identifier": [{"@value": "https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B9%CE%8A%CE%B8%CF%85_%CE%91%CE%B4%CE%AE%CE%B8%CF%89%CF%84%CE%B8%CF%85"}], "@type": ["http://purl.org/vocab/cpsv#PublicService"], "http://purl.org/dc/terms/title": [{"@value": "registration of an adult demonstrator"}], "http://purl.org/dc/terms/description": [{"@value": "registration of the municipality of an undeclared person"}], "http://www.w3.org/ns/dcat#keyword": [{"@value": "municipality"}], "http://purl.org/dc/terms/language": [{"@id": "http://publications.europa.eu/resource/authority/language/ENG"}], "http://purl.org/dc/terms/spatial": [{"@id": "http://publications.europa.eu/resource/authority/place/GRC"}]}
```

step 4

Finally, we store the result from the output field in a Json file and we upload it in the server

this is where we upload each Json file

Refresh Cut Copy Paste Rename Delete Chmod Logout

/public_html/cpsv

Name	Size	Date	Time	User	Group	Permissions
Approval of an employee's participation in a training seminar.json	2KB	18/10/18	18:13	imagnisa	imagnisa	-rw-r--r--
aithmata.politwn.json	1KB	07/11/18	21:16	imagnisa	imagnisa	-rw-r--r--
attestation of permanent residence.json	1KB	07/11/18	18:13	imagnisa	imagnisa	-rw-r--r--
collection of daily settled receipts.json	1KB	07/11/18	18:13	imagnisa	imagnisa	-rw-r--r--
registration of an undeclared person.json	1KB	18/10/18	14:05	imagnisa	imagnisa	-rw-r--r--

New Folder New File Upload Files Upload Folder

Host: 127.0.0.1 User: imagnisa Upload Limit: 1GB

here is a picture of the subfolder called cpsv in the server, where our sample of documents for the purposes of the dissertation are stored.

← → ↻ ⓘ Μη ασφαλής | imagnisa.labs.ihu.edu.gr/cpsv/

Εφαρμογές

Index of /cpsv

- [Parent Directory](#)
- [Approval of an employee's participation in a training seminar.json](#)
- [aithmata.politwn.json](#)
- [attestation of permanent residence.json](#)
- [collection of daily settled receipts.json](#)
- [registration of an undeclared person.json](#)

8.5 Images

8.5.1

Here we have an indicative image of such a description for a service ('PSDescriptionCreator' tool)

The screenshot shows the 'Public Service Description Creator' web application. The interface includes a header with the CORE logo and a navigation bar with 'About' and 'Editor' buttons. Below the header, there are tabs for 'Editor', 'Presenter', and 'RDF data'. The main area is a form with the following fields:

Identifier	<input type="text" value="https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81%CE%B1%CF%80%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CF"/>	✕
Name	<input type="text" value="registration of an adult demonstrator"/>	✕
Description	<input type="text" value="registration of the municipality of an undeclared person"/>	✕
Keyword	<input type="text" value="municipality"/>	✕
Sector	<input type="text"/>	✕
Type	<input type="text"/>	✕
Language	<input type="text" value="English"/>	✕
Status	<input type="text"/>	✕
Spatial	<input type="text" value="GRC-Greece"/>	✕
ProcessingTime	Years <input type="text" value="0"/> Months <input type="text" value="0"/> Days <input type="text" value="0"/> Hours <input type="text" value="0"/> Minutes <input type="text" value="0"/>	✕
Requires	<input type="text"/>	✕
Related	<input type="text"/>	✕

At the bottom of the page, there is a footer with the text: "Public Service Description Creator. Copyright © European Union 2016. This Description Creator is published under the [LGPL-3.0 license](#). This project is funded by IS42 Action 2016 29 under framework contract 0107171 Lot 2 (Specific Contract 414)". The Windows taskbar at the bottom shows the date and time as 17/10/2018, 2:44 μμ.

8.5.2

Input Data:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:ns1="http://www.w3.org/ns/dcat#">
  <rdf:Description
    rdf:about="https://el.diadikasies.gr/%CE%95%CE%B3%CE%B3%CF%81
%CE%B1%CF%86%CE%AE_%CE%95%CE%BD%CE%AE%CE%BB%CE%B
```

or Uri:

Input Format:

Output Format:

Raw output

Output

Number of triples parsed: 7

RDF format to be converted in JSON format