



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



DEVELOPMENT OF A CONTEXT KNOWLEDGE SYSTEM FOR MOBILE CONVERSATIONAL AGENTS

RAÚL LOZANO GARCIA

Thesis supervisor: QUIM MOTGER DE LA ENCARNACION (Department of Service and Information
System Engineering)

Degree: Bachelor Degree in Informatics Engineering (Software Engineering)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

29/06/2023

Abstract

A mobile conversational agent or chatbot is software that can perform tasks or services for a particular user or group. The main goal of this Final Degree Project is to develop a context knowledge system for mobile agents, as well as provide it with tools that allow it to be adapted dynamically. This system will allow the user to receive personalised suggestions of actions based on their context and preferences.

This project is developed in the A modality, which means it is associated with a university department. In this case, this project is linked to the Software and Service Engineering Group (GESSI) department from the Barcelona School of Informatics, Universitat Politècnica de Catalunya.

This system will expose feature integrations between different applications of a mobile device, allowing the user to perform actions in one application and receive suggestions of possible actions to be executed in another application, letting them complete that suggestion without having to explicitly open the application.

Resum

Un agent conversacional mòbil o chatbot és un programari que pot realitzar tasques o serveis per a un usuari o grup en concret. L'objectiu principal d'aquest Treball de Fi de Grau és desenvolupar un sistema de coneixement de context per a agents mòbils, així com proporcionar-li eines perquè pugui adaptar-se dinàmicament. Aquest sistema permetrà a l'usuari rebre suggeriments personalitzats d'accions basades en el seu context i preferències.

Aquest projecte es desenvolupa en la modalitat A, que significa que està associat a un departament universitari. En aquest cas, aquest projecte està vinculat al departament de Grup d'Enginyeria del Software i dels Serveis (GESSI) de la Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya.

Aquest sistema exposarà integracions de funcions entre diferents aplicacions d'un dispositiu mòbil, permetent a l'usuari realitzar accions en una aplicació i rebre suggeriments d'accions possibles per a ser executades en una altra, permetent-li completar aquesta acció sense haver d'obrir explícitament l'aplicació en qüestió.

Resumen

Un agente conversacional móvil o chatbot es un software que puede realizar tareas o servicios para un usuario o grupo en concreto. El objetivo principal de este Trabajo de Fin de Grado es desarrollar un sistema de conocimiento de contexto para agentes móviles, así como proporcionarle herramientas para que pueda adaptarse dinámicamente. Este sistema permitirá al usuario recibir sugerencias personalizadas de acciones basadas en su contexto y preferencias.

Este proyecto se desarrolla en la modalidad A, que significa que está asociado a un departamento universitario. En este caso, este proyecto está vinculado al departamento de Grupo de Ingeniería del Software y de los Servicios (GESSI) de la Facultad de Informática de Barcelona, Universitat Politècnica de Catalunya.

Este sistema expondrá integraciones de funciones entre diferentes aplicaciones de un dispositivo móvil, permitiendo al usuario realizar acciones en una aplicación y recibir sugerencias de acciones posibles para ser ejecutadas en otra, permitiéndole completar esa acción sin tener que abrir explícitamente la aplicación en cuestión.

Contents

1	Introduction	7
1.1	Contextualization	7
1.2	Concepts	9
1.3	Problem to be solved	10
1.4	Stakeholders	11
2	Justification	13
2.1	Examples of task-oriented conversational agents	14
2.2	Justification of the solution proposed	15
3	Scope	17
3.1	Goals	17
3.2	Requirements	18
3.2.1	Functional requirements	18
3.2.2	Non-functional requirements	19
3.3	Obstacles and risks	20
4	Methodology	21
4.1	Working methodology	21
4.2	Monitoring tools and validation	21
5	Time planning	23
5.1	Task description	24
5.2	Resources	26
5.3	Estimations and Gantt	27
5.4	Risk management: alternative plans and obstacles	30
6	Budget	31
6.1	Identification of costs	31
6.1.1	Human resources	31
6.1.2	Material resources	32
6.1.3	General resources	33
6.1.4	Contingencies	34
6.1.5	Unforeseen events	34
6.1.6	Final budget	34
6.2	Management control	35
7	Fundamentals	36

7.0.1	What is a Knowledge Graph?	36
7.0.2	Ontologies	37
7.0.3	Why use Knowledge Graphs?	37
7.1	Semantic Web	38
7.1.1	Resource Description Framework (RDF)	38
7.1.2	Semantic RDF Schema	39
7.1.3	SPARQL	40
8	Requirement specification	42
8.1	Motivational example	42
8.2	Functional requirements	43
8.2.1	User stories	43
8.3	Non-functional requirements	57
9	Design	60
9.1	Basic structure	60
9.2	Logical architecture	61
9.3	Design patterns	62
9.3.1	Dependency Injection	63
9.3.2	Singleton	63
9.4	Data model diagram	63
9.4.1	Entities and relations	65
10	Implementation process	68
10.1	System overview	68
10.2	Development resources	69
10.2.1	Technologies and frameworks used	69
10.2.2	Development tools	70
10.3	Data schema extension	70
10.3.1	Creation of the data schema extension	70
10.3.2	Domain constraints	73
10.4	Knowledge base operations	73
10.4.1	Entity definitions	73
10.4.2	User preferences	83
10.4.3	Suggestions computation	84
11	Testing	89
11.1	Knowledge base test dataset	89
11.2	Component testing	91
11.3	Integration testing	92
12	Sustainability	95
12.1	Initial milestone	96
12.1.1	Economic dimension	96
12.1.2	Environmental dimension	96
12.1.3	Social dimension	96
12.2	Final milestone	97
12.2.1	Economic dimension	97
12.2.2	Environmental dimension	98

12.2.3 Social dimension	99
13 Legislation	101
14 Final planning and budget	102
14.1 Estimations	102
14.2 Budget	106
15 Conclusions	107
15.1 Achievement of the goals	107
15.2 Technical competencies	108
15.3 Future work	110
15.4 Personal conclusions	110
A Swagger API documentation	111
References	114

List of Figures

1.1	Example of how the current knowledge base looks like	10
2.1	Example of how Google Assistant looks like	14
2.2	Example of how Siri looks like	15
2.3	Example of how the final knowledge base will look like	15
4.1	Example of a Git feature branching strategy	22
4.2	Snapshot of the student's board in Taiga	22
5.1	Gantt's diagram	28
5.2	Gantt's diagram	29
7.1	Example of a graph	36
7.2	Example of an RDF triple	39
7.3	Example of a schema.org schema	40
7.4	Example of a SPARQL query	40
8.1	Motivational example	42
9.1	3-layer architecture	61
9.2	Spring Boot flow architecture	62
9.3	Data model diagram	64
10.1	Example of how users were represented in the initial knowledge base . . .	69
10.2	Example of how the data for Alice and Bob users would be represented in the extended data schema	72
10.3	SPARQL query to request feature integrations from source features and previous user preferences	85
10.4	SPARQL query to request app integrations from a target feature and previous user preferences	86
10.5	SPARQL query to request source-target parameter integrations for selected app	87
10.6	SPARQL query to request custom parameters for selected app	88
11.1	Example conversation between the chatbot and the user	91
11.2	Component testing	92
11.3	Component testing results	92
11.4	Integration testing example	93
14.1	Gantt's diagram	104
14.2	Gantt's diagram	105

List of Tables

5.1	Summary table of the tasks	27
5.2	Project risks	30
6.1	Salaries of the roles involved in the project	31
6.2	Cost of the human resources involved in the project	32
6.3	Software resources used in the project.	33
6.4	Hardware costs	33
6.5	Unforeseen events costs	34
6.6	Final budget estimation	35
8.1	User Story 1: Add a feature integration	44
8.2	User Story 2: Read a feature integration	45
8.3	User Story 3: Update a feature integration	45
8.4	User Story 4: Delete a feature integration	46
8.5	User Story 5: Create a parameter	46
8.6	User Story 6: Read a parameter	47
8.7	User Story 7: Update a parameter	47
8.8	User Story 8: Delete a parameter	48
8.9	User Story 9: Add a parameter integration	48
8.10	User Story 10: Read a parameter integration	49
8.11	User Story 11: Update a parameter integration	49
8.12	User Story 12: Delete a parameter integration	50
8.13	User Story 13: Add an app integration	50
8.14	User Story 14: Read an app integration	51
8.15	User Story 15: Update an app integration	51
8.16	User Story 16: Delete an app integration	52
8.17	User Story 17: Request feature integrations from a source features and previous user preferences	52
8.18	User Story 18: Request app integrations from a selected target feature and previous user preferences	53
8.19	User Story 19: Request source-target parameter integrations for a selected app	53
8.20	User Story 20: Request custom parameters for a selected app	54
8.21	User Story 21: Design and integrate the data schema extensions	54
8.22	User Story 22: Manage Knowledge Base with CRUD operations	55
8.23	User Story 23: Define and formalise domain constraints for the integrations	55
8.24	User Story 24: Obtain personalised suggestions of actions	56
8.25	User Story 25: Obtain personalised suggestions of apps	56

8.26	User Story 26: Obtain parameters used for an integration	56
8.27	User Story 27: Obtain custom parameters	57
8.28	Non-functional requirement 1: Speed and Latency	57
8.29	Non-functional requirement 2: Reliability and Availability	57
8.30	Non-functional requirement 3: Scalability and Extensibility	58
8.31	Non-functional requirement 4: Access	58
8.32	Non-functional requirement 5: Privacy	58
8.33	Non-functional requirement 6: Project Planning	59
12.1	Final cost of the project	97
12.2	Maintenance cost of the project	98
14.1	Summary table of the tasks with their initial and final time estimations. .	103
14.2	Initial and final budget	106

Chapter 1

Introduction

The project “Development of a context knowledge system for mobile conversational agents” corresponds to my Final Degree Project to conclude my studies in Informatics Engineering, at the Barcelona School of Informatics, Universitat Politècnica de Catalunya. This work belongs to the Software Engineering specialization, so it aims to address a series of concepts and topics taught in this specialization. It is developed in the A modality, which means it is associated with a university department. In this case, this project is linked to the Software and Service Engineering Group (GESSI) department and it is directed by one of its members, Quim Motger de la Encarnación.

The main goal of this project is to develop a context knowledge system [1] for mobile agents, as well as provide it with tools that allow it to be adapted dynamically. This knowledge system will expose feature integration between different applications of a mobile device. In this project, we identify feature integration between two particular applications as the correlation of an action done by the user in a specific application and the resulting suggestion of possible action to be executed by the user in another application, based on the user’s context.

1.1 Contextualization

The popularity of ChatGPT and other large language models has been increasing lately, and this has provoked as a consequence an increase in interest in the natural language processing field (NLP). These models have been a revolution when it comes to mobile conversational agents, due to them being able to provide text generation capabilities, which is a very important aspect when it comes to conversational agents. This ability to make systems that are capable of both understanding and generating text is key in several fields, and this has made the interest in conversational agents arise.

A conversational agent [2], also known as a chatbot can be described as an entity that can simulate a real conversation a user would have with another human. It uses both Natural Language Processing (NLP) and Natural Language Understanding (NLU) in order to provide a proper conversation in natural language, so the user is able to understand.

There are several types of chatbots, such as *rule-based*, *generative*, *retrieval-based* or *task-oriented*. The chatbot developed in the Chatbots4Mobile project follows the task-oriented approach, due to its special characteristics related to the knowledge structure [3]. Also

known as declarative, these type of chatbots resolves an attempt to a specific task or the configuration of a specific task executed in a software environment. Evolving from basic rule-based NLP techniques to the appearance and revolution of deep learning models like recurrent neural networks and, more recently, transformer models, these chatbots are able to understand the user's intent and provide the user with the best possible answer to the question asked. This type of chatbot can be usually seen in customer service, where the user asks a question and the chatbot provides the user with the best possible answer to the question asked.

Recently, there has been a huge increase in the number of chatbots that are being developed, and this is due to the fact that the technology behind them is becoming more and more accessible to the general public. And thus, added to the revolution currently happening in the field of NLP, with transformer models such as BERT, GPT, etc..., makes conversational agents a very interesting topic to study and get involved in.

Other aspects that may attract and are attracting the attention of the public are:

- *Generative content*: the ability to generate text is a very important aspect when talking about chatbots, and it is also one of the most important aspects when it comes to the user experience. The ability to generate content considering the context of the user is what makes this project so interesting.
- *Personalisation*: the ability to provide a personalised experience to the user is also a very important aspect when it comes to chatbots. This is also one of the main goals of this project.
- *Adaptive style*: the ability to adapt the style of the conversation with the user is also a very important feature, especially in contexts where the chatbot may have more or less knowledge of the user, and it must adapt to that and offer the best possible information to them.

There is another type of chatbot which has high relevance for this project, and it is the *knowledge-based chatbot*. This type of chatbot is based on a knowledge base, which is a database that stores information about a particular or several topics, and it is used to provide the user with the best possible answer to what was asked.

However, all types of chatbots can be also combined, to make their functionalities even more powerful. For instance, if we consider both the *task-oriented* and the *knowledge-based* chatbot types when combined, we can obtain a really powerful chatbot. In this case, the final result would be a chatbot that is supported by a knowledge base, which would allow us to set, adapt, and modify the conversational experience the user will have with the agent applied to a particular context. This is the key of this project, enabling the definition of a knowledge base which will make a chatbot adapt to the user's necessities and context.

This project is part of a much wider project, which is named Chatbots4Mobile [4], and it is also carried by the GESSI. This bigger project consists of the full design and development of a conversational agent system, including both the design of the chatbot itself (what the user will see on their screens) as well as the development of a mobile app knowledge base for the apps that will be integrated and also the design and implementation of the data modelling and storage system. In this case, the project "Development of a context knowledge system for mobile conversational agents" will focus more on the latter;

although the knowledge base already exists, it was originally limited to model feature-oriented knowledge based on the NL processing of document-related data, designed to conduct deductive and inductive knowledge-generation techniques. In this thesis, we aim at extending the knowledge base to support the process of automatic enactment of mobile app features through the formalisation of the features exposed by those applications and the data required to enact those features, by using the conversational agent as a supporting agent for the user to conduct automatic enactment of those features.

1.2 Concepts

In this section, there will be a series of concepts that will be mentioned several times throughout the document, in order to clarify their meaning and also try to avoid any misunderstanding to the reader.

- **App:** A mobile application integrating a specific set of functionalities for a given domain.
- **App catalogue:** It is the complete set of apps that are stored in the knowledge base. This set of apps will be the core of the knowledge base to which the chatbot has access to support the conversational process and the execution of tasks (i.e., feature integration, see below).
- **App integration:** It reflects the correlation that exists between two apps (that is, a user first performed an action and after that, they performed another action in a different app). The first app will be considered as the one that exposes the initial action (*source*) and the second one will be the app that performs the final action (*target*).
- **Users:** People who use the applications available in the app catalogue and benefit from the suggestions the chatbot will provide.
- **Feature:** It is a functionality that corresponds to an action a user can perform in a particular app. For instance, in the case of a calendar app, a feature could be *create an event* or *delete an event*.
- **Feature integration:** It refers to the correlation that exists between two features (i.e., a functional requirement from the user's perspective [5]) from different applications. One action done in a particular application will *trigger* this feature integration, which will have another action from a different application as its *target*.
- **Parameter:** It is a value that is required to perform a particular action/feature. For instance, a feature may require a date and a name to be specified, so these two values will be considered as parameters.
- **Parameter integration:** It is the process associated with obtaining a parameter value that appears in the integration of a particular action and using it in another action integration. The parameter that we obtain the value from will be considered as the one which *populates* the parameter integration and the one which receives this value and uses it for itself will be the *target*.

To have a better understanding of the current state of the system's knowledge base, a visual representation of what has been created, as can be seen in Figure 1.1.

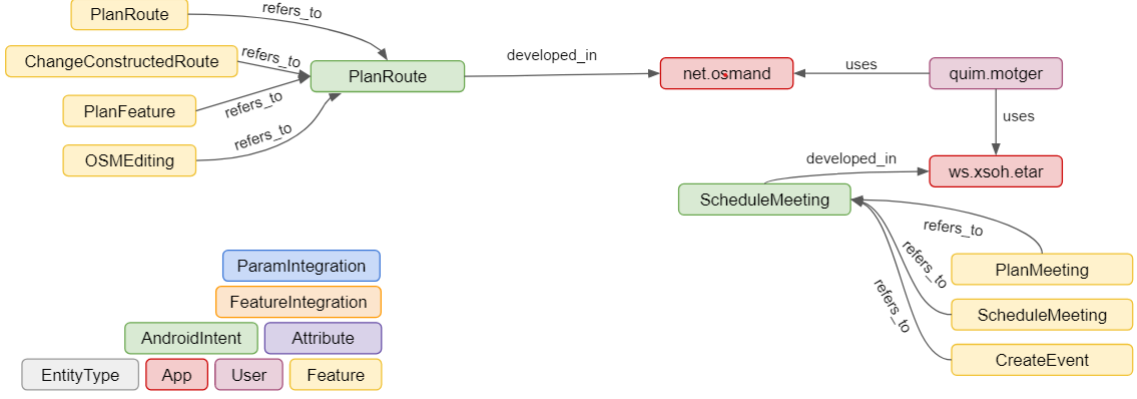


Figure 1.1: Example of how the current knowledge base looks like
Font: GESSI-Chatbots4Mobile

It currently has this structure due to it being limited to a particular context and a series of functionalities. It is only composed of a set of users, a set of mobile applications and a set of features that are associated with each one of them. The features are the ones that will be used to perform the feature integration and parameter integration processes. In this case, considering the example shown, the *PlanRoute* feature (also named *ChangeConstructedRoute*, *PlanFeature* or *OSMEditing*) is integrated into the *net.osmand* [6] application, a trail-tracking app including navigational and route-planning features. The *ScheduleMeeting* (*PlanMeeting*, *ScheduleMeeting* or also named *CreateEvent*) feature is integrated into the *ws.xsoh.etar* [7] application, a calendar app. The user *quim.motger* uses both apps, and the knowledge base extension proposed in this project will be responsible for linking the *PlanRoute* feature with the *ScheduleMeeting* one in an automatic way, allowing him to perform both actions by only performing one of them.

1.3 Problem to be solved

As mentioned in the previous section, this project will work on the design and development of an extension of an initial knowledge base, in order to make it facilitate the automated implementation of functionalities within mobile applications. At this moment, this initial knowledge base (database) scope is limited to modelling feature-oriented knowledge, so several changes are required.

Checking the current data model the system has, it can be seen that it is limited to the knowledge of the features at a granular level (a feature is an irreducible element). The concept of feature is to be extended with other aspects, such as the actions the user has done in the past, the context of the user, etc. This will allow the agent to have a much more complete knowledge of the user, and thus, be able to provide a much more personalised experience.

This existing knowledge base the agent has access to is not enough. As mentioned, the system has no notion of the context of the user, an aspect that can be key to providing an overall better experience to them.

In this context, we refer to user context as the preferences of the user as described in [8] towards a specific use/configuration of a set of mobile applications (i.e., an applications

catalogue).

In other words, *the current system does not have the formalisation of either features or feature integrations from a technical perspective, nor can provide a personalised experience to the user* (due to it being designed for other purposes), so there is a need to extend the current system with a more complex data model, in order to have some knowledge of the user's context.

In summary, it is required the design and development of several features to enhance the product that is being implemented, such as the integration of a context knowledge system or a much more complex data schema, so the user can have a fully personalised experience with the agent, as well as delivering a high-quality product the users can rely on and truly get a benefit from.

1.4 Stakeholders

The stakeholders are all the individuals or groups that are interested, benefited and also involved in the project. Taking all of them into account is key because they allow us to be able to define everything related to the project correctly. Taking this into account, it is a must to consider all of their opinions, requirements and desires. The different stakeholders that have been identified have been listed below.

Mobile app users

As has been described before, they are the ones who will truly obtain the benefits of using the conversational agents, by having one of them on their devices.

GESSI

GESSI is the research group that owns the project Chatbots4Mobile, which is the master project of this. We can identify several stakeholders in the department:

- **Product Owner:** it is the one responsible for maximizing the value that is being delivered as well as ensuring the delivery of the product. In this case, the whole department is considered the product owner, they want to obtain a high-quality final product from all of their projects, including this one.
- **Project coordinators:** are the individuals that are responsible for coordinating not only this project but also the other sub-projects that are being developed related to Chatbots4Mobile.
- **Final Degree Project's director:** Quim Motger de la Encarnación will be the person who will supervise and guide the project, to make it progress and keep the correct path.
- **Developer:** is the one responsible for implementing all the software, documenting the whole process of the project and assuring the correct functioning of it. In this case, there is only one developer, and it is the author of this Final Degree Project.

Mobile software developers

Not only entities that are close to the project will be benefited from it. In this case, there is a whole group of developers that work on the production of conversational agents that, by checking Chatbots4Mobile can be inspired and obtain another perspective of how conversational agents should work, and more.

Task-oriented and knowledge base chatbots NLP community

Of course, everyone with an interest in this type of chatbot will be keen on considering what has been developed in this thesis, as it is a really useful tool that can be used, for instance, as a reference for future projects.

Chapter 2

Justification

The pervasiveness of conversational agents in mobile devices (e.g., Siri, Google Assistant) is already a reality nowadays. However, the usage of these agents is limited to just being used whenever the user wants to, and this affects directly the experience people get from their assistants, negatively. In the following sections, some existing conversational agents will be mentioned, and also the justification of the presented solution will be presented.

Before justifying the existence of this thesis, it is important to mention that the scope of this thesis is limited to the extension of the knowledge base to support the design and development of the task-oriented chatbot; therefore, the design and development of the conversational system itself, especially in terms of dialogue management, intent recognition, entity extraction, etc., is not considered; it will be aimed towards the development of its knowledge base, which will improve the overall performance and capabilities of the chatbot.

The agent Chatbots4Mobile will offer will be a combination of a *task-oriented* and a *knowledge-based* agent, and it will be also *domain-specific*. In other words, it will be a task-oriented agent, as it will be able to perform a specific task (i.e., to provide the user with a list of suggestions of apps that can be useful for them), and it will be also knowledge-based, as it will have a knowledge base that will be used to store and retrieve the information needed to perform the task. Finally, it will be domain-specific, as it will be focused on the domain of mobile applications.

2.1 Examples of task-oriented conversational agents

Nowadays mobile conversational agents are part of most of our devices, although their usage is not as expanded as it could be. In this section, two examples of ubiquitous agents will be shown and, although not behaving as the one that we are going to develop, they are still interesting to see, since they represent the state of the art in terms of direct user support.

Google Assistant [9]

Google Assistant is a virtual assistant developed by Google, and it can be found and installed with ease on several platforms, and it is dedicated to mobile and home devices mainly. You can request via its application or by using voice commands several actions, such as searching for things on the internet, playing games, and also actions with Google apps, such as creating events, reminders, and more. If we compare what this assistant can do with what our chatbot will be able to do, we can see that Google's assistant enables the execution of a set of specific actions for which an entry gateway is provided, whereas our chatbot will be modelled to support the integration of functionalities between different applications.

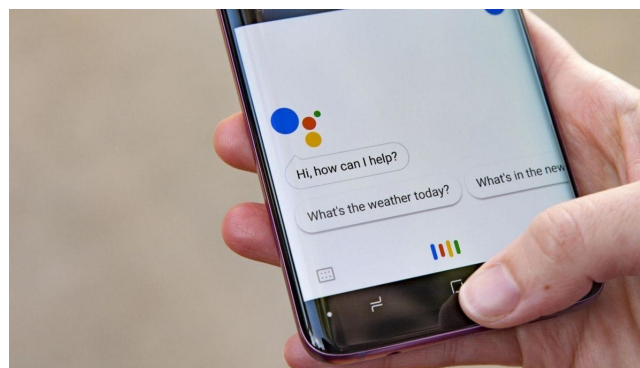


Figure 2.1: Example of how Google Assistant looks like
Font: Analytics Vidhya [10]

Apple's Siri [11]

Siri is the virtual assistant developed by Apple, which is a built-in feature for all iOS devices and it cannot be installed or accessed by any devices that use a different operating system. It can resolve your doubts by searching the internet as well, or also make some actions with Apple native applications, like setting alarms, reminders, and so on. It can be manually invoked or it can also be requested by voice commands. The chatbot will be able to perform actions as Siri does, with the difference that it will not be only performing actions with native applications, but with a much wider range of applications, the user might have installed. It also happens the same as with Google Assistant, that Siri is not able to integrate functionalities between different applications, which is one of the main features of our chatbot.



Figure 2.2: Example of how Siri looks like
Font: Macworld [12]

2.2 Justification of the solution proposed

The current state of Chatbots4Mobile’s knowledge base, as has been explained and shown previously (a representation of its initial state is shown in 1.1), is limited to the knowledge of the features at a granular level. What it is aimed is to extend the current system with a more complex data model, to make it support the formalisation of the application’s features, as well making it able to store information related to the user’s context. Both of these aspects will allow the agent to have much more complete knowledge of the user (in terms of actions they do/they have done in the past, preferences, etc.), and thus, be able to provide a much more personalised experience. An example of how the knowledge base will look after the addition of the new features is shown in 2.3.

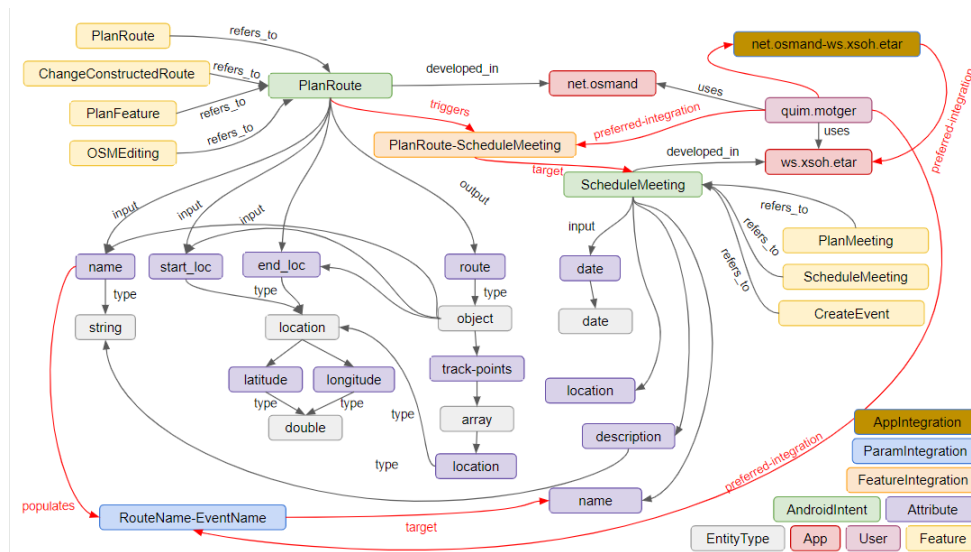


Figure 2.3: Example of how the final knowledge base will look like
Font: GESSI-Chatbots4Mobile

As can be appreciated, the new data model will have the existing entities (i.e., users, apps and features) the current system has, but it will be extended with the addition of feature integrations, which will correlate two applications. To develop these integrations, parameters will be used, and also parameter integration will be added. Last but not least, app integrations will be added as well to understand in a more general way which apps the user usually uses together. All of these extensions will be done in order to have a more complete system, which will be able to work in a custom way for each user.

Considering the example shown above, we can see that the user, the apps and the features these expose are the same as the ones in the initial knowledge base shown in 1.1. However, it can be seen that now each feature has a set of parameters that enables us to know what is required to perform the feature (e.g., the feature *PlanRoute* now has parameters that allow us to indicate the name, the starting location and the end location of the route we want to plan). Also, it can be seen that the concept of *integrations* appears as well, in this case, we can see a feature integration defined between the two apps called *PlanRoute-ScheduleMeeting*, which has the first one as the source feature and the second one as the target. A parameter integration called *RouteName-EventName* is also defined, which correlates the parameter *name* of the feature *PlanRoute* with the parameter *name* of the feature *ScheduleMeeting*.

One aspect that is worth mentioning is that with the desired solution the concept of cross-app feature integration is introduced. This concept can be defined as the ability to integrate the features of two or more applications. There are several examples of this concept, such as deep linking, SDKs, APIs, or more recent techniques such as MashReDroid [13]. However, the solution proposed in this thesis is different from the ones mentioned, as it aims to provide this integration by using a conversational agent.

Our contribution focuses on the development of a context knowledge base, which will allow the agent Chatbots4Mobile to offer a more personalised experience to the user.

With this, what it is aimed is to extend the current functionalities the project Chatbots4Mobile offers, enabling it to provide both an integration between several applications and a much more personalised experience in terms of suggestions displayed to the user.

So, it can be said that the main contribution of this thesis is the development of a context knowledge base, which will focus on contributing into two scopes: **(1) the cross-app feature integration**, and **(2) the task-oriented, domain-specific and knowledge-based conversational agents**, aiming for the delivery of a personalised experience to the user.

Chapter 3

Scope

In this chapter, the main goals of this project are defined, as well as all the requirements that have to be ensured in order to deliver a correct final product. Lastly, the possible obstacles and risks that may appear during the development process are analysed, in order to have awareness of what can happen and to avoid unforeseen circumstances.

3.1 Goals

This project has a clear goal defined, which then is divided into different sub-goals in order to facilitate tasking management, as well as being more capable to monitor everything is going according to plan.

The main goal of this project is to **integrate a feature-based Context Knowledge System (CKS) into an existing Knowledge Base (KB) in the field of mobile applications and the set of features these expose with the required data schema and business logic to allow the formalisation of these features and the potential personalized integration between functionalities from different applications.** To achieve this, a set of sub-goals have been defined:

G1 Design the data schema extension to support the formalisation of:

- (a) Mobile app features
- (b) Input and output parameters
- (c) App integrations
- (d) Feature integrations
- (e) Parameter integrations
- (f) User-preferred integrations

G2 Define and formalise the domain constraints of these feature integrations. That includes (1) setting a unique identifier for each feature, (2) defining the data is being sent before actually performing any action in the knowledge base and (3) ensuring the data that is trying to be added does fit with what is already stored in the knowledge base.

- G3 Integrate the data schema into the existing KB and extend the KB with the required business logic to populate the KB with CKS data (CRUD operations).
- G4 Extend the KB with advanced inductive requests based on the customisation and enactment of potential integrations for a given user.
- G5 Evaluate the KB extension through the integration of a conversational agent querying and modifying a specific KB instance.

3.2 Requirements

In this section all the functional and non-functional requirements have been defined for this project, considering the goals that have been mentioned. These requirements have great relevance, due to them being the ones defining a valid product.

It is important to mention that in this section the requirements are not detailed in a formal way, but in a more general way, in order to have a better understanding of what the system has to do. The formalisation of these requirements has been done and can be found in Chapter 8 *Requirement specification*.

3.2.1 Functional requirements

System management

- The system's current data schema will be extended to support a wider range of options, that is (1) mobile app features, (2) feature parameters and (3) potential integrations between apps, features and parameters.
- The system's Knowledge Base will be also extended with advanced inductive requests, and these will be the following:
 - Request feature integrations from source features and previous user preferences.
 - Request app integrations from a selected target feature and previous user preferences.
 - Request source-target parameter integrations for a selected app.
 - Request custom parameters for a selected app.

Administrator's requirements

- The users with the pertinent privileges will be able to design and extend data schema extensions which will be then integrated into the existing Knowledge Base, which are the ones recently mentioned.
- The users with the pertinent privileges will be able to populate the current Knowledge Base with Context Knowledge Base data with the required business logic. The operations that will be added to these types of data will be the **CRUD** ones:
 - Create
 - Read

- Update
- Delete

These CRUD operations will be defined for all the entities that are currently defined as well as the ones that will be added. So, the CRUDS are defined for the *users, applications, features, parameters, app integrations, feature integrations* and *parameter integrations* entities.

- The users with the pertinent privileges will be able to both define and formalise the domain constraints those potential integrations that will be added will have. Those constraints are applied at a business logic level, and they consist of (1) unique identifiers for each entity, (2) validation of the data being sent to the knowledge base and (3) ensuring data integrity. Those constraints will be explained more in detail later on.

Chatbot management

- The system will allow the chatbot to obtain suggestions of (1) target applications, (2) target features, (3) parameter integrations and (4) custom parameters based on the user's actions on an application from their device.
- The system will allow the chatbot to be able to receive feedback from the user to check if what is being displayed fits their requirements, by modifying the user's preferences they have previously defined accordingly.
- The system will allow the chatbot to know if a user has performed a particular action that it recommended, by applying the corresponding changes in the user's knowledge stored in the Knowledge Base.

3.2.2 Non-functional requirements

In terms of the Context Knowledge System that is going to be developed, the following non-functional requirements are the ones that the system must satisfy. The *Volere Requirements Specification Template* [14] has been used to define this.

- **12a. Speed and Latency Requirements:** the system has to be able to provide the requested information in a reduced amount of time.
- **12d. Reliability and Availability Requirements:** the system has to be always available for the chatbot, and it should be resistant to errors, it knows how to act when something fails.
- **12g. Scalability or Extensibility Requirements:** the system has to be able to scale (that is, in terms of the knowledge base size) its features without letting the users with no access to the conversational agent.
- **15a. Access Requirements:** access to this knowledge has to be restricted to only the developers and administrators of it, using credentials and having to access to the system through a *Virtual Private Network* (VPN).
- **15c. Privacy Requirements:** the user's data the system uses has to be protected and the application must assure its privacy, by having security measures such as the ones mentioned in the previous requirement.

- **21a. Project Planning:** to develop the system, every feature that is wanted to be implemented has to be defined before starting the process of development.

3.3 Obstacles and risks

During the development of this thesis, different obstacles and blockers may appear, and they might affect by delaying the completion of some features, forcing us to maybe not complete every goal set.

- **Bugs:** in every software development project there is a possibility of spending a significant amount of time dealing with bugs. Depending on the phase these appear, they can have a considerable impact and may affect our initial planning.[15]
- **Deadline of the project:** the whole project will last no more than 4 months, so if there are some problems or delays during the process, this will cause us to discard some aspects initially defined, due to the lack of more hours to complete them.
- **Lack of experience in the field:** I have no previous experience in developing a context knowledge system or with anything related to conversational agents: semantic web, not that much experience with graph databases nor RDF..., so if during the process the necessity of learning a particular technology appears, there will be no other option but to spend some time learning it reducing the amount of time available to actually work on the development of the project.

Chapter 4

Methodology

A correct definition of the working methodology it is going to be applied, the monitoring tools that will be used and the validation methods that will be chosen to control everything is going correctly is key in the development process of a thesis.

4.1 Working methodology

Due to the short amount of time given to complete this thesis (approximately 4 months), it has been decided to work in an *agile* [16] manner, in order to be able to organise the work in a clear way. For instance, the *Kanban* scheduling system has been chosen to organise what needs to be done. A series of tasks will be defined, and they will be distributed along the *sprints* previously scheduled, each one of them lasting two weeks. Once a *sprint* ends, there will be some retrospectives in order to see if everything that had been defined has been accomplished or if there have been some tasks that should be considered in the following iterations.

4.2 Monitoring tools and validation

On the one hand, in terms of validation, there will be, at least, a meeting with the director once every two weeks, which can be considered as both a *sprint planning* and a *sprint review* session. However, there will be constant communication with the director, and any other meetings will be scheduled if needed. The means of communication chosen have been mail and also using Slack [17].

On the other hand, some monitoring tools will be used to have everything organised and accessible, making it easy for the director and the department to check everything is going according to the plan:

Git [18]

It is the most popular distributed version control system in the market, and it is the one that GESSI have been using while working on the Chatbots4Mobile project. The *Git Feature Branch* workflow will be used, which allows us to divide the code into branches, enabling us to work on different features independently and then merge them all together in a clean way. Each branch will contain a feature (task) previously defined, and once it

has been implemented, it will be merged with the main branch, which contains the final code. This workflow can be seen in figure 4.1

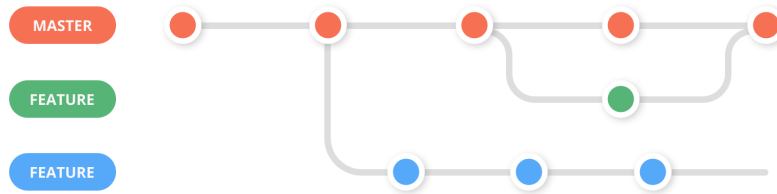


Figure 4.1: Example of a Git feature branching strategy
Font: Tencent [19]

GitHub [20]

It is a popular online hosting service for version control and development, which uses Git. The project's code will be uploaded and stored there, so it can be easily checked by any member that has access to it. The reason behind this choice is simple: it is also currently being used by other projects related to Chatbots4Mobile.

Taiga [21]

It is an open-source agile project management software which will be used in order to work on the project as desired, using *agile* methodologies. There we will have all the tasks and *sprints* defined, and by this interface Taiga offers everything will be organised immaculately. A snapshot of how the used board looks can be seen in figure 4.2.

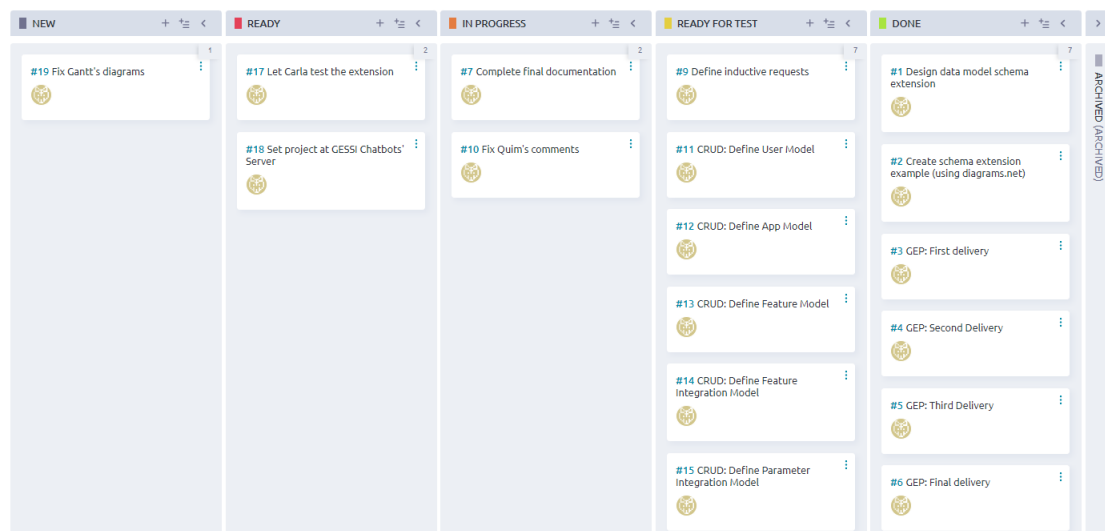


Figure 4.2: Snapshot of the student's board in Taiga

Chapter 5

Time planning

As has been previously mentioned, agile methodologies will be used in the development of this project, more specifically, the concept of *sprints* from the *Scrum* methodology has been chosen, in order to organise the work as well as to be able to track everything more efficiently. In the agile methodology, previous to the *sprints*, there is a pre-stage called *Inception*, which aims to identify the goals and also the risks of the project, as well as to perform the temporal planning, define the user stories and so on. Therefore, since February 9th, which is when the first meeting between the director and the student took place until March 9th, the project is considered to be in the inception stage.

Related to the *sprints*, they will be done every two weeks:

- ***Sprint 1:*** from March 9th to March 23rd
- ***Sprint 2:*** from March 23rd to April 6th
- ***Sprint 3:*** from April 6th to April 20th
- ***Sprint 4:*** from April 20th to May 4th
- ***Sprint 5:*** from May 4th to May 18th
- ***Sprint 6:*** from May 18th to June 1st
- ***Sprint 7:*** from June 1st to June 15th

Those *sprints* will last 2 weeks starting and ending on Thursdays, in which the *Sprint Planning* of the next iteration will be held and also both the *Sprint Review* and *Sprint Retrospective* from the previous iteration will be discussed, due to us adapting the *Scrum* methodology. For each of those meetings, the student will take some notes as a summary of the contents discussed, and those notes will appear as an appendix in the final report.

The remaining time, which takes from June 15th until the oral defence day (which can be between the 26th and the 30th of June), will be dedicated to the final phase of the project, which will consist of polishing the documentation, beautifying the code and preparing for the final presentation.

For this time planning, it is important to mention that weekends are considered as working days for me as well, due to them being the days in which I can dedicate more time for the project. They have been considered when estimating the scheduled time for each task.

5.1 Task description

To organise the tasks, two big groups have been defined:

- **Project management (PM):** all the tasks related to the project which are not directly related to the development of the solution. Both the meetings and documentation are included in this group.
- **Development (DEV):** all the tasks related to the development of the application. Due to the size of the project, the development tasks have been divided into smaller groups, based on the sub-goals previously defined: *Project set up (DEV0)*, *Data schema extension (DEV1)*, *Domain constraints (DEV2)*, *Data schema integration into the Knowledge Base (DEV3)*, *Knowledge Base extension (DEV4)* and *Knowledge Base evaluation (DEV5)*

Project management (PM)

- **PM1 - Context and scope:** define the main goal of the thesis, give previous context, explain the relevance and justification of the topic, how the work will be developed and with which tools. Also, possible risks and obstacles are defined.
- **PM2 - Temporal planning:** define the project's phases, the tasks to be done and the estimated time for each one. Also, the possible risks and obstacles are defined.
- **PM3 - Budget and sustainability:** define and describe the cost of the project as well as the budget.
- **PM4 - Initial documentation:** unify the documents generated in PM1, PM2 and PM3, obtaining an initial version of the thesis.
- **PM5 - Final documentation:** prepare the final version of the thesis.
- **PM6 - Oral defence preparation:** prepare the oral defence of the thesis.
- **PM7 - *Sprint* Planning:** meeting dedicated to choosing the tasks that will be done in the upcoming iteration.
- **PM8 - *Sprint* Review:** meeting dedicated to showing the job done in the last iteration and receiving feedback.
- **PM9 - *Sprint* Retrospective:** meeting dedicated to evaluating the work done in the last iteration and, if necessary, making changes.

Project set up (DEV0)

- **DEV0.1 - Learn the scientific fundamentals:** check the technologies that will be used in the project and learn the basic concepts of it, in order to be able to use them for the development of the solution without having to spend too much time learning them while developing. The technologies to be studied are *Java Spring Boot*, *SPARQL* and *RDF*.
- **DEV0.2 - Set up the development environment:** set up the development environment and analyse the initial contents of it, which will be used to develop the solution. The development environment is composed of the following tools: *Visual Studio*, *GitHub* and *GraphDB*.

Data schema extension (DEV1)

- **DEV1.1 - Identify requirements:** identify the requirements for the data schema extension.
- **DEV1.2 - Define entities and relationships:** define both the entities and relationships that will be included in the data schema extension.
- **DEV1.3 - Define attributes:** define the attributes of the entities that will be included in the extended data schema.
- **DEV1.4 - Create the data schema:** once everything is defined, add everything together and formally define the extended data schema, which will be used as a reference in order to implement it.

Domain constraints (DEV2)

- **DEV2.1 - Analyse potential constraints:** analyse the potential constraints that can be applied to entities, relationships and attributes that will be added to the data schema extension.
- **DEV2.2 - Formalise the constraints for each feature:** define the constraints for each feature integration.
- **DEV2.3 - Implement the constraints:** formally implement the constraints in the thesis.

Data schema integration (DEV3)

- **DEV3.1 - Analyse the existing KB structure:** analyse the existing KB structure and identify the required changes.
- **DEV3.2 - Identify required changes:** identify the required changes to the existing KB structure.
- **DEV3.3 - Implement changes to KB:** implement the required changes to the existing KB structure.
- **DEV3.4 - Create business logic for CRUD operations:** create the business logic for the CRUD operations.
- **DEV3.5 - Test the integration:** test the integration of the extended data schema into the existing KB.

Knowledge Base extension (DEV4)

- **DEV4.1 - Identify the types of inductive requests to be supported:** the chatbot will be able to perform inductive requests, which means that the user can ask the chatbot to perform actions that are not explicitly defined in the chatbot's knowledge base. It is a must to identify what are the expected types of inductive requests that the chatbot will be able to perform.
- **DEV4.2 - Design a method for customisation and enactment of potential integrations.**
- **DEV4.3 - Implement the method for supporting inductive requests.**

- **DEV4.4 - Test the extension:** test the Knowledge Base's extension with inductive requests.

Knowledge Base evaluation (DEV5)

- **DEV5.1 - CRUD operations testing:** test the CRUD operations of the extended data schema developed, to ensure every entity is being created, read, updated and deleted correctly.
- **DEV5.2 - Integrate the KB extension into the agent:** add the designed and implemented knowledge base extension into the current conversational agent.
- **DEV5.2 - Test the agent's ability to query and modify the KB:** check that the agent can perform the desired actions and see what happens in the knowledge base when the agent performs them.
- **DEV5.3 - Evaluate the results and make improvements as needed:** after testing, check if everything is working as expected and, if not, make the necessary changes.

5.2 Resources

Human resources

- **Quim Motger de la Encarnación (D):** the director of this thesis. His role is to guide the student across the different stages of the project, to accomplish the established goals.
- **Raúl Lozano García (S):** the student author of this thesis. His role is to both design and develop the defined functionalities.
- **Carla Campàs Gené (C):** student that, along with GESSI has developed some chatbot implementation tasks. She will allow the student author of the thesis to ease the integration process as well as the evaluation of the agent's knowledge base system.

Material resources

- **HP Omen (LAP1):** laptop which will be used to develop the project.
- **Dell Latitude 3520 (LAP2):** laptop which will be used to develop the project.
- **Mouse (M)**
- **Visual Studio Code (VS):** code editor to develop the project as well as to document the process.
- **GitHub and Git (GIT):** version controlling tools.

5.3 Estimations and Gantt

Task ID	Task Name	Effort	Dependencies	Resources
PM	Project management	100		
PM1	Context and scope	20		D, S, LAP1, LAP2, M, VS
PM2	Temporal planning	10	PM1	D, S, LAP1, LAP2, M, VS
PM3	Budget and sustainability	10	PM2	D, S, LAP1, LAP2, M, VS
PM4	Initial documentation	10	PM3	D, S, LAP1, LAP2, M, VS
PM5	Final documentation	20	PM4	D, S, LAP1, LAP2, M, VS
PM6	Oral defence preparation	20	PM5	D, S, LAP1, LAP2, M
PM7	<i>Sprint</i> Planning	5		D, S, LAP1, LAP2, M
PM8	<i>Sprint</i> Review	5		D, S, LAP1, LAP2, M
PM9	<i>Sprint</i> Retrospective	5		D, S, LAP1, LAP2, M
DEV0	Project set up	10		
DEV0.1	Learn the scientific fundamentals	5		S, LAP1, LAP2, M
DEV0.2	Set up the development environment	5		D, S, LAP1, LAP2, M, GIT
DEV1	Data schema extension	80	DEV0	
DEV1.1	Identify requirements	10		S, LAP1, LAP2, M, GIT
DEV1.2	Define entities and relationships	30	DEV1.1	S, LAP1, LAP2, M, GIT
DEV1.3	Define attributes	15	DEV1.1	S, LAP1, LAP2, M, GIT
DEV1.4	Create the data schema	25	DEV1.1	S, LAP1, LAP2, M, GIT
DEV2	Domain constraints	60		
DEV2.1	Analyse potential constraints	20		S, LAP1, LAP2, M
DEV2.2	Formalise the constraints for each feature	30	DEV2.1	S, LAP1, LAP2, M
DEV2.3	Implement the constraints	10	DEV2.2	S, LAP1, LAP2, M, VS
DEV3	Data schema integration	130	DEV2	
DEV3.1	Analyse the existing KB structure	20		S, LAP1, LAP2, M, VS, GIT
DEV3.2	Identify required changes	20	DEV3.1	S, LAP1, LAP2, M, VS, GIT
DEV3.3	Implement changes to KB	35	DEV3.2	S, LAP1, LAP2, M, VS, GIT
DEV3.4	Create business logic for CRUD operations	35	DEV3.3	S, LAP1, LAP2, M, VS, GIT
DEV3.5	Test the integration	20	DEV3.4	S, LAP1, LAP2, M, VS, GIT
DEV4	Knowledge Base extension	105	DEV3	
DEV4.1	Identify the types of inductive requests to be supported	10		S, LAP1, LAP2, M, VS, GIT
DEV4.2	Design a method for customisation and enactment of potential integrations	35	DEV4.1	S, LAP1, LAP2, M, VS, GIT
DEV4.3	Implement the method for supporting inductive requests	35	DEV4.2	S, LAP1, LAP2, M, VS, GIT
DEV4.4	Test the extension	25	DEV4.3	S, LAP1, LAP2, M, VS, GIT
DEV5	Knowledge Base evaluation	75	DEV4	
DEV5.1	CRUD operations testing	15		S, LAP1, LAP2, M, VS, GIT
DEV5.2	Integrate the KB extension into the agent	15		S, LAP1, LAP2, M, VS, GIT
DEV5.3	Test the agent's ability to query and modify the KB	35		S, LAP1, LAP2, M, VS, GIT
DEV5.4	Evaluate the results and make improvements as needed	10	DEV5.1, DEV5.2	S, LAP1, LAP2, M, VS, GIT

Table 5.1: Summary table of the tasks
Font: Own elaboration

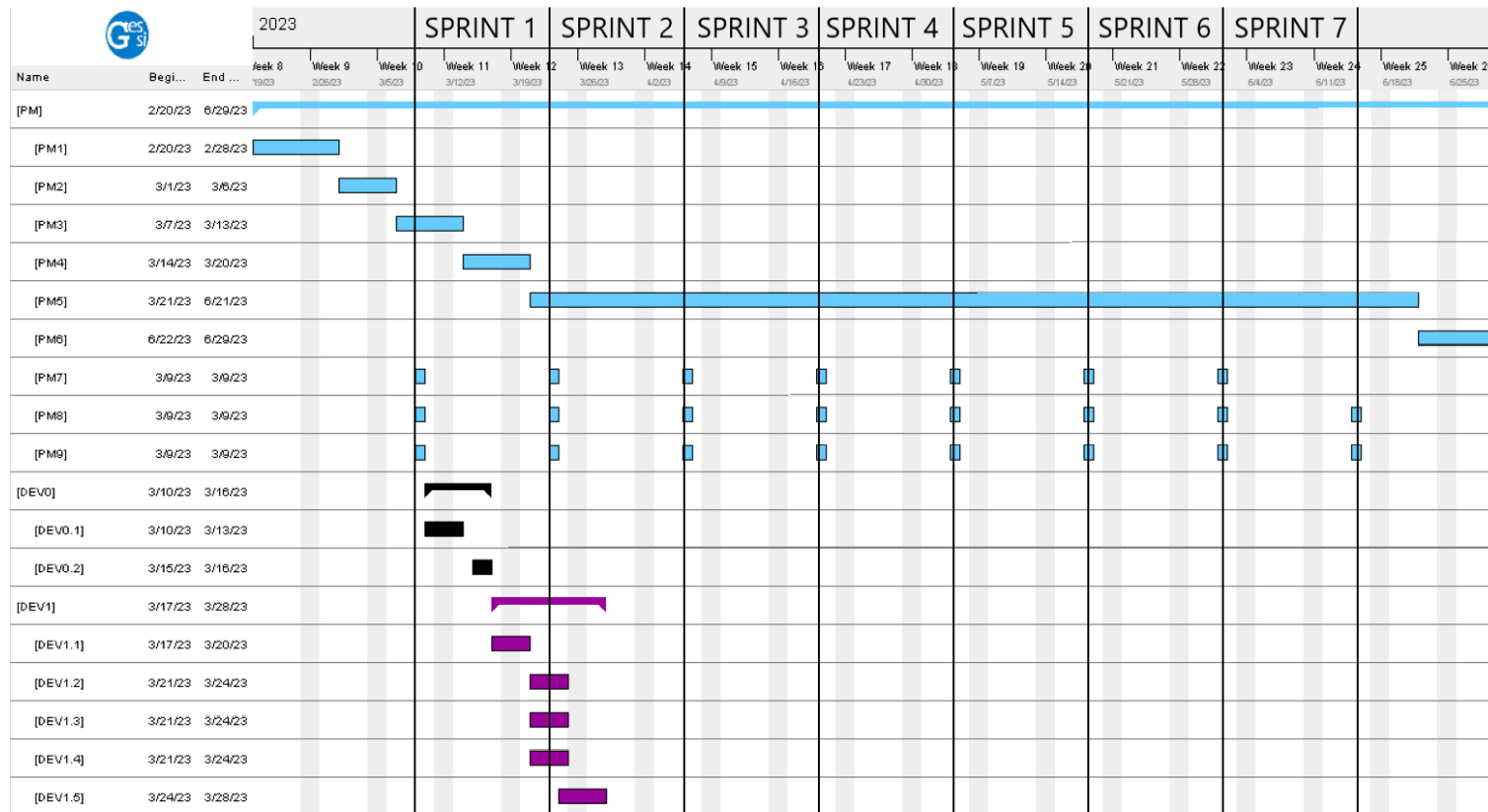


Figure 5.1: Gantt's diagram

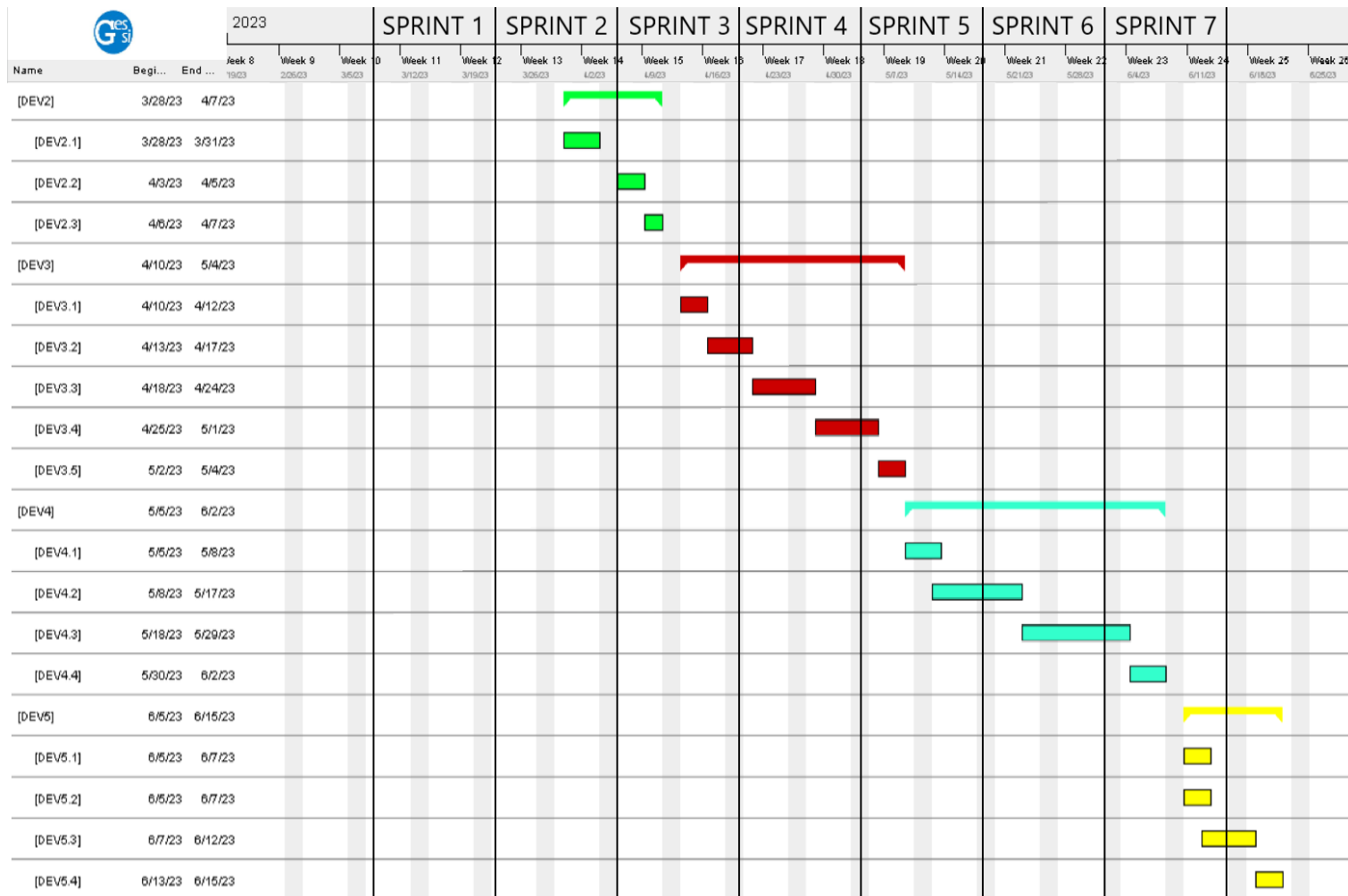


Figure 5.2: Gantt's diagram

5.4 Risk management: alternative plans and obstacles

In this chapter, the possible risks that may appear during the development of the project are listed. For each of them, the impact level and the mitigation are also specified.

Possible risk	Impact level	Mitigation
Deadline of the project	Medium	Design of a high-quality and detailed time planning
Lack of experience in the field	High	Assignment of more hours in tasks where the developer lacks experience
Bugs	Medium	Addition of testing and previous consideration of incompatibilities

Table 5.2: Project risks
Font: Own elaboration

Before mentioning the possible risks, it is important to mention that when preparing the project planning those possible risks were taken into account, when estimating the time needed for each task. However, it is relevant to say that this estimated time added to each task may be shorter or longer than the real-time needed to complete the task, depending on the risks involved.

Firstly, the time-related risk is one possible problem that we may face, since it determines if the project can be successfully delivered or not. Due to this, proper initial planning and a rigorous following of it are required.

Secondly, the lack of knowledge of some techniques or technologies used in the project may cause delays in the development, which might affect the final delivery of the project, obtaining a final product less complete. For instance, some of the technologies used in the project are new or not very well known by myself, such as working with GraphDB or using SPARQL queries. In this case, I will maybe have to dedicate more time to particular tasks that require the use of these technologies (maybe even more than the estimated time in the initial planning), an aspect that may affect not only the planning but may also affect the scope of the project.

Finally, bugs may appear during the development of the project. Depending on the phase in which they appear, they may have a considerable impact and may affect the initial planning. If they appear to be critical, they will stop the development of the project until they are solved, and if they are not that critical, they will be tried to solve in the next *sprint* iteration.

Chapter 6

Budget

In this chapter, all the costs associated with the development of the project are described. Human resources, general expenses and taxes are taken into account, obtaining an approximate budget for the work.

6.1 Identification of costs

6.1.1 Human resources

To be able to estimate how much the cost related to human resources will be, it is important to set both the salary per hour and the number of hours per person involved in the project. After considering this, table 6.1 it is defined the salary per hour of each member implicated in the project. Those salaries have been estimated by checking the average salary of each role in Barcelona, Spain, and the site where the information has been taken is *Glassdoor*[22]. The gross salary per hour has been taken, and the Social Security (SS) has been added to it.

Role	Gross salary (hour) + SS
Project Lead [PL]	27,10 €
Solution Architect [SA]	26 €
Software Architect [A]	24,70 €
Developer [D]	13,20 €
Tester [T]	17,20 €

Table 6.1: Salaries of the roles involved in the project
Font: Own elaboration

Once the salaries have been defined, the cost per task can be defined as well, as it is shown in table 6.2. For each task, each of the involved roles has been set with an approximate number of hours of dedication.

Task ID	Task Name	PL	SA	A	D	T	Cost
PM	Project management	30h	0h	0h	100h	0h	2.199 €
PM1	Context and scope	2	0	0	20	0	318,20 €
PM2	Temporal planning	2	0	0	10	0	186,20 €
PM3	Budget and sustainability	1	0	0	10	0	159,10 €
PM4	Initial documentation	0	0	0	10	0	132 €
PM5	Final documentation	5	0	0	20	0	399,50 €
PM6	Oral defence preparation	5	0	0	20	0	399,50 €
PM7	<i>Sprint</i> Planning	5	0	0	5	0	201,50 €
PM8	<i>Sprint</i> Review	5	0	0	5	0	201,50 €
PM9	<i>Sprint</i> Retrospective	5	0	0	5	0	201,50 €
DEV0	Project set up	2h	0h	0h	10h	0h	186,20 €
DEV0.1	Learn the basic concepts	0	0	0	5	0	66 €
DEV0.2	Set up the development environment	2	0	0	5	0	120,20 €
DEV1	Data schema extension	0h	17h	38h	25h	0h	1.710,60 €
DEV1.1	Identify requirements	0	2	8	0	0	249,60 €
DEV1.2	Define entities and relationships	0	10	20	0	0	754 €
DEV1.4	Define attributes	0	5	10	0	0	377 €
DEV1.5	Create the data schema	0	0	0	25	0	330 €
DEV2	Domain constraints	0h	17,5h	42,5h	0h	0h	1.504,75 €
DEV2.1	Analyse potential constraints	0	7,5	12,5	0	0	503,75 €
DEV2.2	Formalise the constraints for each feature	0	10	20	0	0	754 €
DEV2.3	Implement the constraints	0	0	10	0	0	247 €
DEV3	Data schema integration	0h	10h	30h	90h	15h	2.449 €
DEV3.1	Analyse the existing KB structure	0	5	15	0	0	500,50 €
DEV3.2	Identify required changes	0	5	15	0	0	500,50 €
DEV3.3	Implement changes to KB	0	0	0	35	0	462 €
DEV3.4	Create business logic for CRUD operations	0	0	0	35	0	462 €
DEV3.5	Test the integration	0	0	0	5	15	324 €
DEV4	Knowledge Base extension	0h	3h	7h	75h	20h	1.584,90 €
DEV4.1	Identify the types of inductive requests to be supported	0	3	7	0	0	250,90 €
DEV4.2	Design a method for customisation and enactment of potential integrations	0	0	0	35	0	462 €
DEV4.3	Implement the method for supporting inductive requests	0	0	0	35	0	462 €
DEV4.4	Test the extension	0	0	0	5	20	410 €
DEV5	Knowledge Base evaluation	0h	0h	0h	45h	30h	1.110 €
DEV5.1	CRUD operations testing	0	0	0	15	0	198 €
DEV5.2	Integrate the KB extension into the agent	0	0	0	15	0	198 €
DEV5.3	Test the agent's ability to query and modify the KB	0	0	0	5	30	582 €
DEV5.4	Evaluate the results and make improvements as needed	0	0	0	10	0	132 €
Total		30h	47,5h	117,5h	335h	65h	10.744,45 €

Table 6.2: Cost of the human resources involved in the project
Font: Own elaboration

6.1.2 Material resources

In terms of material resources, they have been divided into two categories: software and hardware. The software resources are the ones that are needed to develop the project, and the hardware resources are the ones that are needed to run the project. In the following subsections, the software and hardware resources are described in detail.

Software

For each of the software used, since they offer a free version or a student version, for the estimation of the budget any piece of it will be considered. In table 6.3 all the software resources used are listed.

Software resource	Price	Amortization
Visual Studio Code [23]	0€	0€
GitHub/Git [24]	0€	0€
GraphDB [25]	0€	0€
Total		0€

Table 6.3: Software resources used in the project.

Font: Own elaboration

Hardware

For the development of this thesis, basic hardware will be used. It will be composed of two laptops and a mouse. The Tax Office allows to amortize the cost of the hardware in 3-4 years, so the amortization will be computed using the following formula:

$$\text{Amortization (€)} = (\text{Cost (euros)} / ((\text{Lifetime (years)} * \text{Annual workdays} * 8 \text{ hours})) * \text{Project dedication (hours)})$$

Resource	Price	Lifetime	Usage time	Amortization
HP Omen ax001ns [26]	1.199 €	4 years	535 hours	79,86 €
Dell Latitude 3520 [27]	1.124 €	4 years	535 hours	74,87 €
Logitech MX Master 3 [28]	20 €	4 years	535 hours	1,33 €
Total				156,06€

Table 6.4: Hardware costs

Font: Own elaboration

6.1.3 General resources

Although the project is not going to be developed in a company, the resources that are going to be used are the same as in a company. So, there are two costs that have to be considered:

- **Electricity:** Although the cost of electricity varies daily, in order to simplify the estimation of the budget, the cost of electricity at the time of writing this document is 0.19746 €/kWh [29], and that value will be the one used for the estimation of the budget. Considering that the project will be developed over 128 days, the consumption of the room where the developer will work is approximately 140W, and also that the developer will work 4 hours per day, the cost of the electricity is estimated as follows:

$$\text{Electricity cost (€)} = (0.19746 \text{ €/kWh}) * (140\text{W}) * (4\text{h/day}) * (128 \text{ days}) = 14,15 \text{ €}$$

- **Internet:** The monthly cost of the internet service the developer has is around 80€. Considering that the project will be developed over 128 days, the cost of the internet is estimated as follows:

$$\text{Internet cost (€)} = 80\text{€} * (128 \text{ days} / 30 \text{ days}) = 341,33 \text{ €}$$

In conclusion, the total cost of those general resources is $14,15 + 341,33 = \mathbf{355,48\text{€}}$.

6.1.4 Contingencies

The contingency cost is an amount of money that is added to the project's budget, in order to cover those unforeseen events that have not been anticipated. This cost is calculated as a percentage of the total value of the budget, and it is usually determined by the sector and the level of detail of the budget. In software projects, it is usually between 10 and 20% of the total budget[30]. Just to set a value inside this range, a contingency cost of 15% has been defined.

6.1.5 Unforeseen events

Previously, in section 5.4, the possible risks that may appear during the development of the project were listed, as well as the mitigation for each of them. This mitigation has also a cost, and that is why it has to be considered when estimating the budget of the project. Its cost has been estimated using the following formula:

$$\text{Cost (€)} = \text{Probability (\%)} * \text{Fixer salary (€/h)} * \text{Dedicated time (hours)}$$

Applying this, the following estimations are obtained:

Risk	Probability	Dedicated time	Cost
Deadline of the project	25%	30h	99 €
Lack of experience in the field	80%	50h	528 €
Bugs	40%	30h	158,40 €
Total			785,40 €

Table 6.5: Unforeseen events costs
Font: Own elaboration

It is important to note that the responsible for working on the mitigation of the risks will be the developer, so the costs have been estimated using the salary of the developer.

6.1.6 Final budget

Finally, after identifying the resources and estimating the cost of each of them, the final budget of the project is obtained, as can be seen in table 6.6.

Cost	Value
Human resources	10.744,45 €
Material resources	156,06 €
General resources	355,48 €
Contingencies	1577,47 €
Unforeseen events	785,40 €
Total cost	13.618,86 €

Table 6.6: Final budget estimation
Font: Own elaboration

6.2 Management control

Once the budget has been estimated, some metrics must be defined to help detect possible deviations that may occur. For this reason, after each *sprint*, these metrics and the real costs obtained will be used in order to identify any deviation in advance or, if not, to check if the initial budget estimation is being fulfilled.

The metrics established are the following:

- Hours per task deviation:

$$(\text{Estimated hours} - \text{Real hours}) * \text{Real cost}$$

- Human resources:

- Cost of human resources per task deviation:

$$(\text{Estimated cost} - \text{Real cost}) * \text{Real hours}$$

- Human resources cost deviation:

$$(\text{Estimated cost per hour} - \text{Real cost per hour}) * \text{Real hours}$$

- General resources cost deviation:

$$\text{Estimated general resources cost} - \text{Real general resources cost}$$

- Total contingency cost deviation:

$$\text{Estimated unforeseen expenses} - \text{Real unforeseen expenses}$$

- Total cost deviation:

$$\text{Estimated total cost} - \text{Real total cost}$$

- Total hours deviation:

$$\text{Estimated total hours} - \text{Real total hours}$$

Chapter 7

Fundamentals

In this chapter, the fundamentals of the project are explained, in order to understand what the extension will be technologically based on.

7.0.1 What is a Knowledge Graph?

A Knowledge Graph is a knowledge base that uses a graph data model to store knowledge [31]. Before explaining what a Knowledge Graph is, it is necessary to explain how a graph data model works.

A graph data model is a data model that stores data in the form of a graph [32]. What is a graph? It is a data structure that consists of a set of nodes and a set of edges, which are the ones that connect the nodes. An example of a graph can be seen in figure 7.1.

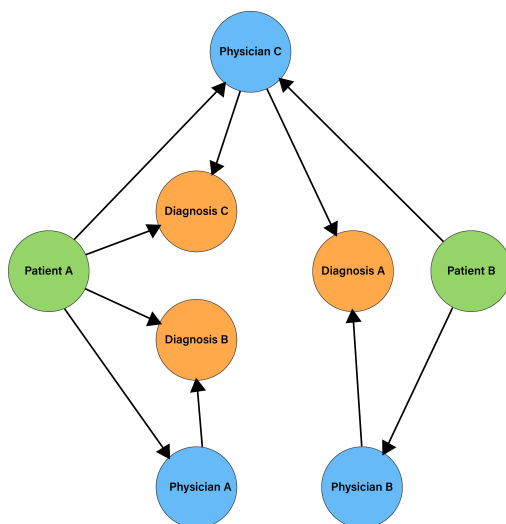


Figure 7.1: Example of a graph
Font: Plum Flower Software [33]

As can be seen, a node represents an entity and an edge represents a relationship between two entities. Those relationships can be directed or undirected. In the example, the

relationship between the nodes is directed, because it is only possible to go from one direction to the other, but not the other way around.

This information represented as a graph has to be stored somewhere, and that is where another term comes into play: the knowledge base. A knowledge base, long story short, is a centralised repository of organised information. It is used to store information in a way that allows us to retrieve it in a fast way. Taking into consideration the thesis context, the knowledge base is the database that stores the information the chatbot uses.

Last but not least, it is necessary to mention another feature of the knowledge graphs that is key to understanding how they work: the integration of deduction and/or induction knowledge techniques. Deduction and induction are two types of reasoning that are used to, given some information, derive new information from it [34].

So, in other words, a Knowledge Graph is a graph data model that stores knowledge, and it combines different aspects from some data management paradigms, these being (1) the *database* (it stores the data), (2) the *knowledge base* (it possesses formal semantics) and (3) the *graph* (it represents the data in a graph) [35].

7.0.2 Ontologies

It has been mentioned that in this thesis a Knowledge Graph is used, as it has been also said that they follow a graph data model. However, it is also necessary to mention how the information the graph contains is structured and understandable for everyone. That is why ontologies are something to take into account.

An ontology is a formal representation of knowledge as a set of concepts within a domain, as well as the relationships that exist between those concepts [36]. It defines a common vocabulary which enables a good organisation of knowledge, both in a structured and systematic way. By using ontologies, knowledge sharing between systems, apps, and also stakeholders is facilitated.

They are usually represented using some kind of ontology language, such as Web Ontology Language (OWL) [37] or Resource Description Framework (RDF) [38]. A more detailed explanation of these languages (RDF in particular) will be given later in section 7.1.1.

To sum up, ontologies are used to define the structure of the information that is stored in the Knowledge Graph, and by using them it is possible to make some actions such as automated reasoning, knowledge sharing or addition of new knowledge in a more efficient way.

7.0.3 Why use Knowledge Graphs?

After analysing what has been previously mentioned, it can be concluded that this type of database, taking into account the thesis' context, is the best option to store the information about the chatbots, due to several reasons:

- *The information about the features and user preferences is not structured, so using a relational database is not the best option.* A knowledge graph is a smarter option because it allows one to store information in a more flexible way.

- *The information about the features and user preferences is not only stored but also related.* The users have, for instance, a lot of features related to them, and those features are related to other features, and so on. That is why it is necessary to store all that information in a way that allows us to retrieve it easily. In this case, a knowledge graph is the best option, because it allows us to store the information in a way that allows us to retrieve it easily.
- *The information about the features and user preferences is not complete.* They are constantly changing, so it is necessary to be able to add new information to the database without having to modify the previous one. Due to the flexibility of the knowledge graphs, it is possible to add new information without applying any changes to the existing one, so it is one of the main reasons why a knowledge graph is the best option for this project.

7.1 Semantic Web

The Semantic Web (also known as Web Semantic or Web 3.0) can be defined as an extension of the current World Wide Web (WWW), which looks for adding more meaning to content that is available on the Internet, in order to make it more understandable for machines and, therefore, more useful for humans [39].

The semantic is a branch of linguistics dedicated to the study of meaning, reference and truth [40]. For instance, a word can have different meanings depending on the context it is used in, and words that are similar in one context can be different in another.

Therefore, it can be said that what the Semantic Web does is add semantics to the current Web contents, so that machines can have a better understanding of the information, enabling them to have knowledge about the context and, as a consequence, to be able to have more accurate interpretations of the information in question.

The first person to use the term *Semantic Web*, Tim Berners-Lee, explained that, in the context of the Semantic Web, the term *semantic* reflects that the data is manageable by machines, and the term *web* reflects a dynamic realm of interconnected objects with mappings from URIs to objects [41].

7.1.1 Resource Description Framework (RDF)

As has been said, RDF [42] is an ontology language, and it is used to represent information. It is indeed the standard language for representing information in a graph, as it is mentioned in the World Wide Web Consortium (W3C) website [43].

The usage of an ontology language allows to have standard semantics and syntax for:

- Concept definitions.
- Relationship definitions.
- Constraints.

RDF is a graph-based data model, and it is composed of triples. These triples are composed of three elements:

- **Subject:** the resource that is being described.

- **Predicate:** the property that is being described.
- **Object:** the value of the property.

Below, in figure 7.2, an example of how a triple should be formed is shown. In order to have a complete understanding, let's set an example: *"The kid uses a mobile phone"*. In this case, the subject would be *"The kid"*, the predicate would be *"uses"* and the object would be *"mobile phone"*. So, using triples for representing information as the one in the example, an understandable graph can be created, for both humans and machines.

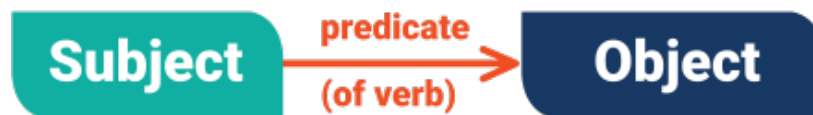


Figure 7.2: Example of an RDF triple
Font: Ontotext

It is important to mention that the nodes in an RDF graph can be either resources, literals or blank nodes. For instance, resources are the ones that are identified by a URI, and they can be either a subject or an object, and literals are the ones that represent values, and they can be either a subject or an object.

In conclusion, RDF is a very important part of the Semantic Web, as it is the standard language for representing information in a graph, and it is used for representing information in a way that is understandable for both humans and machines. For this thesis, RDF is used for representing the information that is contained in the Knowledge Base, more specifically in the Knowledge Graph.

7.1.2 Semantic RDF Schema

Another key aspect of this thesis and closely related to the Semantic Web and more specifically to RDF is the Semantic RDF Schema [44], a schema that is used for representing information within a graph. It is a very important part of the Semantic Web, as it provides a solid standard for representing what is inside a graph, enabling us to have a clear understanding of the information that is contained in it. There are several schemas that are used in the Semantic Web, such as FOAF (Friend of a Friend) [45], which is used for representing people, DCT (Dublin Core Terms) [46], which is used for representing metadata, or Schema.org, which is the one that has been used in this thesis.

Schema.org [47] is a collaborative project between Google, Microsoft, Yahoo and Yandex, and it is a collection of schemas that can be used to markup web pages in order to help search engines and other systems to have a better understanding of the information contained in web pages, graph data and other types of documents.

Schema.org is a very important part of this thesis, as it is the schema that is used to represent the data our knowledge base contains. There are several reasons why it has been chosen, and that is because it provides (1) standardised vocabulary to represent information in, for instance, a Knowledge Graph, (2) interoperability between different systems and (3) semantic annotations for web pages. An example of a schema that is used in this thesis is shown below, in figure 7.3. It is the schema used to represent the

users of the system. It has several properties defined, such as the address. It can be seen, from the perspective of object-oriented programming, as a class, and the properties as the attributes of the class.

Property	Expected Type	Description
Properties from Person		
additionalName	Text	An additional name for a Person, can be used for a middle name.
address	PostalAddress or Text	Physical address of the item.
affiliation	Organization	An organization that this person is affiliated with. For example, a school/university, a club, or a team.
alumniOf	EducationalOrganization or Organization	An organization that the person is an alumni of. Inverse property: alumni
award	Text	An award won by or for this item. Supersedes awards.
birthDate	Date	Date of birth.
birthPlace	Place	The place where the person was born.

Figure 7.3: Example of a schema.org schema
Font: Schema.org

7.1.3 SPARQL

Another important part of the Semantic Web is SPARQL. It is a query language for RDF, and it is used for querying RDF graphs. It is a very important part of the Semantic Web, as it allows one to query the information that is contained in the Knowledge Graph. It has a syntax that is very similar to SQL, and it is composed, mainly of two parts: the *SELECT* clause and the *WHERE* clause.

The *SELECT* clause is used for specifying the variables that are going to be retrieved from the query, and the *WHERE* clause is used for specifying the conditions that the results must meet. Below, in figure 7.4, an example of a SPARQL query performed in the developed Knowledge Base is shown.

```
PREFIX schema: <https://schema.org/>
SELECT ?id ?email
WHERE {
    ?user a schema:Person .
    ?user schema:identifier ?id .
    ?user schema:email ?email
}
```

Figure 7.4: Example of a SPARQL query
Font: Own elaboration

As can be seen in the figure, the *SELECT* clause is used to specify that the id and the email properties are the ones that will be retrieved from the query, and the *WHERE* clause is used to specify that the results must be the ones that have the *Person* type.

It is important to highlight SPARQL due to its importance in both the Semantic Web and this thesis. It allows us to give the chatbot the information from the Knowledge Base that it requires to perform its tasks. For instance, if the chatbot needs to know the name of a person, it can perform a SPARQL query to the Knowledge Base, and it will retrieve the information that it needs.

Chapter 8

Requirement specification

In this chapter, all the requirements the system has to meet are both specified and described. A requirement can be defined as a condition, capability or expectation that the system in question must satisfy.

These sets of requirements have been divided into two big groups: functional and non-functional requirements.

8.1 Motivational example

For a better, clearer perspective on the set of functional and non-functional requirements depicted in this section, we present a motivational example of a case study exemplifying the expected behaviour of the knowledge base system extension. This example can be found in figure 8.1.

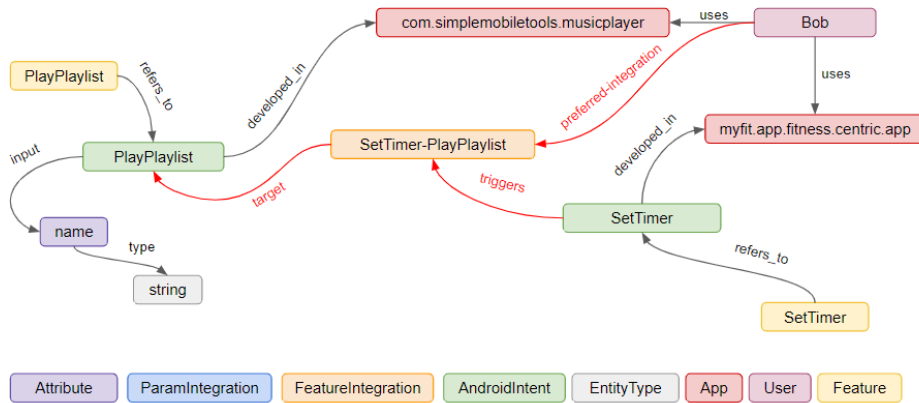


Figure 8.1: Motivational example

Let's consider the following scenario: there is a user called Bob, who has a set of apps installed on his phone, where one of them is a music app (for instance, *Simple Music Player* [48]) and another one is a clock app that has a timer functionality (for instance, the *MyFit* [49] app). Bob usually works out and when he is going to start he sets a timer

in order to keep track of the rest time between sets. Bob also has a playlist on the music app that he always listens to when working out, called "Calorie Burner".

So, Bob is used to doing the following order:

1. Open the *MyFit* app (*MyFit.openApp()*)
2. Set a timer (*AlarmApp.setTimer()*)
3. Open the music app (*MusicPlayer.openApp()*)
4. Search the playlist (*MusicPlayer.searchPlaylist("Calorie Burner")*)
5. Play the playlist (*MusicPlayer.searchPlaylist("Calorie Burner").play()*)
6. Start working out

Bob has been doing this for a while, and with the knowledge system that is being developed, the chatbot will be able to learn about the user's context and patterns, such as the one mentioned above ¹. So now, with this extended knowledge system, the chatbot will be able to get the trigger that the user Bob has performed the *AlarmApp.setTimer()* action, and then the chatbot will consult the knowledge that exists about Bob, and based on his apps, usual actions and context, the chatbot will be able to suggest Bob play the particular playlist he always listens when working out, so Bob does not have to do it manually. The chatbot then, will perform by itself all three actions mentioned above (*MusicPlayer.openApp()*, *MusicPlayer.searchPlaylist("Calorie Burner")* and *MusicPlayer.playPlaylist("Calorie Burner")*), reducing the number of steps Bob has to do to achieve the same result.

That is what this project is about, to develop a knowledge system that is able to learn about the user's context and patterns, and then be able to suggest the user with the best possible integrations between the apps the user has installed on their phone, in order to make them have a better experience.

8.2 Functional requirements

The functional requirements are the ones that define the functionality of a system, and they can describe the behaviour of the software as well.

8.2.1 User stories

A user story [50] is a general description of a software feature, specified from the perspective of the final user. It has to define what the user exactly wants and the reason why. Its goal is to make understandable what the user requires from the system, and what is valuable for them. These user stories always follow the following structure:

As a [User Role], I want to [Goal], so that [Reason].

On each of the elements that are between brackets is a field that has to be filled with the corresponding information. For them to be defined correctly each one of them must answer the following questions:

¹This learning is out of the scope of this project. In this thesis, the focus is on collecting this learning, modelling it, updating it and returning it to the chatbot.

System management

- **User Role:** For whom is the feature being developed? The role of the user has to be specified, and it can be a final user, administrator, etc.
- **Goal:** What is the user trying to achieve? The action that the user wants to perform has to be specified, and it can be a request, creation, modification, etc.
- **Reason:** What will the user obtain by achieving the goal? The reason why the user wants to act has to be specified.

It is also important to mention that, to know when a user's story is fully accomplished, an *acceptance criteria* [51] has to be defined.

US1: Add a feature integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to add feature integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none">• The system receives a JSON body which follows the data model schema design.• The system, if the feature integration already exists in the data schema, throws a 409 error.• The system, if the body is not valid, throws a 400 error.• The system, if the body is valid, adds the feature integration to the knowledge base.

Table 8.1: User Story 1: Add a feature integration

US2: Read a feature integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to read feature integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if a specific feature integration is not specified, will let the chatbot know all the information stored in the knowledge base about all the feature integrations. • The system, if a specific feature integration is specified, if it exists, will let the chatbot know all the information stored in the knowledge base about that feature integration. • The system, if a specific feature integration is specified, if it does not exist, will return a 404 error.

Table 8.2: User Story 2: Read a feature integration

US3: Update a feature integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to update feature integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the specified feature integration does not exist, throws a 404 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, updates the feature integration in the knowledge base.

Table 8.3: User Story 3: Update a feature integration

US4: Delete a feature integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to delete feature integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if the specified feature integration does not exist, throws a 404 error. • The system, if the specified feature integration exists, throws a 204 message and deletes the feature integration from the knowledge base, including all the relations it had.

Table 8.4: User Story 4: Delete a feature integration

US5: Create a parameter	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by adding parameters
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the parameter already exists in the data schema, throws a 409 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, adds the feature integration to the data schema.

Table 8.5: User Story 5: Create a parameter

US6: Read a parameter	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to read parameters
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if a specific parameter is not specified, will let the chatbot know all the information stored in the knowledge base about all the parameters. • The system, if a specific parameter is specified, if it exists, will let the chatbot know all the information stored in the knowledge base about that parameter. • The system, if a specific parameter is specified, if it does not exist, will return a 404 error.

Table 8.6: User Story 6: Read a parameter

US7: Update a parameter	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to update parameters
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the specified parameter does not exist, throws a 404 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, updates the parameter in the knowledge base.

Table 8.7: User Story 7: Update a parameter

US8: Delete a parameter	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to delete parameters
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if the specified parameter does not exist, throws a 404 error. • The system, if the specified parameter exists, throws a 204 message and deletes the parameter from the knowledge base, including all the relations it had.

Table 8.8: User Story 8: Delete a parameter

US9: Add a parameter integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by adding parameter integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the parameter integration already exists in the data schema, throws a 409 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, adds the feature integration to the data schema.

Table 8.9: User Story 9: Add a parameter integration

US10: Read a parameter integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to read parameter integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if a specific parameter integration is not specified, will let the chatbot know all the information stored in the knowledge base about all the parameter integrations. • The system, if a specific parameter integration is specified, if it exists, will let the chatbot know all the information stored in the knowledge base about that parameter integration. • The system, if a specific parameter integration is specified, if it does not exist, will return a 404 error.

Table 8.10: User Story 10: Read a parameter integration

US11: Update a parameter integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to update parameter integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the specified parameter integration does not exist, throws a 404 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, updates the parameter integration in the knowledge base.

Table 8.11: User Story 11: Update a parameter integration

US12: Delete a parameter integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to delete parameter integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if the specified parameter integration does not exist, throws a 404 error. • The system, if the specified parameter integration exists, throws a 204 message and deletes the parameter from the knowledge base, including all the relations it had.

Table 8.12: User Story 12: Delete a parameter integration

US13: Add an app integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by adding app integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the app integration already exists in the data schema, throws a 409 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, adds the feature integration to the data schema.

Table 8.13: User Story 13: Add an app integration

US14: Read an app integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to read app integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if a specific app integration is not specified, will let the chatbot know all the information stored in the knowledge base about all the app integrations. • The system, if a specific app integration is specified, if it exists, will let the chatbot know all the information stored in the knowledge base about that app integration. • The system, if a specific app integration is specified, if it does not exist, will return a 404 error.

Table 8.14: User Story 14: Read an app integration

US15: Update an app integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to update app integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system receives a JSON body which follows the data model schema design. • The system, if the specified app integration does not exist, throws a 404 error. • The system, if the body is not valid, throws a 400 error. • The system, if the body is valid, updates the app integration in the knowledge base.

Table 8.15: User Story 15: Update an app integration

US16: Delete an app integration	
As an	<i>Administrator</i>
I want to	Extend the existing data schema by making it able to delete app integrations
So that	It is able to support a wider range of options that allows the chatbot to have more information about the user.
Acceptance criteria	<ul style="list-style-type: none"> • The system, if the specified app integration does not exist, throws a 404 error. • The system, if the specified app integration exists, throws a 204 message and deletes the app from the knowledge base, including all the relations it had.

Table 8.16: User Story 16: Delete an app integration

US17: Request feature integrations from a source features and previous user preferences	
As an	<i>Administrator</i>
I want to	Extend the existing data schema with the possibility to request feature integrations from source features and previous user preferences
So that	The chatbot capabilities are improved, and it can offer the user a more personalised experience.
Acceptance criteria	<ul style="list-style-type: none"> • The system has a new endpoint that receives a feature and a user id. • The system returns a list of features that has a high probability of being integrated with the feature received, based on user preferences. • The system returns a 200 message if the feature and user id are valid. • The system returns a 404 message if the feature or user id is not valid.

Table 8.17: User Story 17: Request feature integrations from a source features and previous user preferences

US18: Request app integrations from a selected target feature and previous user preferences	
As an	<i>Administrator</i>
I want to	Extend the existing data schema with the possibility to request app integrations from a selected target feature and previous user preferences
So that	The chatbot capabilities are improved, and it can offer the user a more personalised experience.
Acceptance criteria	<ul style="list-style-type: none"> • The system has a new endpoint that receives a source and a target feature and a user id. • The system returns a list of apps that are capable of performing the target feature, basing the results on the user preferences. • The system returns a 200 message if the source, target and user id are valid. • The system returns 404 if any of the specified does not exist.

Table 8.18: User Story 18: Request app integrations from a selected target feature and previous user preferences

US19: Request source-target parameter integrations for a selected app	
As an	<i>Administrator</i>
I want to	Extend the existing data schema with the possibility to request source-target parameter integrations for selected app
So that	The chatbot capabilities are improved, and it can offer the user a more personalised experience.
Acceptance criteria	<ul style="list-style-type: none"> • The system has a new endpoint that receives a JSON body specifying the source app and feature, and the target app and feature. • The system returns a list of parameters integrations that are possible between the two desired features, if any. • The system returns a 200 message if the JSON body is valid, 404 if the specified apps and/or features do not exist.

Table 8.19: User Story 19: Request source-target parameter integrations for a selected app

US20: Request custom parameters for a selected app	
As an	<i>Administrator</i>
I want to	Extend the existing data schema with the possibility to request custom parameters for selected app
So that	The chatbot capabilities are improved, and it can offer the user a more personalised experience.
Acceptance criteria	<ul style="list-style-type: none"> • The system has a new endpoint that receives a JSON body specifying the source app and feature, and the target app and feature. • The system returns the target feature's parameters that do not have a parameter integration with the source feature ones (that is, the ones that must be explicitly specified by the user). • The system returns a 200 message if the JSON body is valid, 404 if the specified apps and/or features do not exist.

Table 8.20: User Story 20: Request custom parameters for a selected app

Administrator's management

US21: Design and integrate the data schema extensions	
As an	<i>Administrator</i>
I want to	Design and extend data schema extensions
So that	They can be then integrated into the existing Knowledge Base and allow the change in the chatbot's behaviour
Acceptance criteria	<ul style="list-style-type: none"> • The data schema designs must be compatible with what the Knowledge Base already has. • The data schema designs must be designed so they can be then used to perform personalised actions for the users. • The chatbot adapts to the new data schema correctly, changing its behaviour accordingly.

Table 8.21: User Story 21: Design and integrate the data schema extensions

US22: Manage Knowledge Base with CRUD operations	
As an	<i>Administrator</i>
I want to	Populate the Knowledge with Context Knowledge Base data, including the required business logic, using CRUD operations
So that	I can manage the chatbot's knowledge in an effective and centralised way
Acceptance criteria	<ul style="list-style-type: none"> • The system must provide Create, Read, Update and Delete (CRUD) operations for managing the Context Knowledge Base data. • The system must ensure the correct functioning of the CRUD operations. • The Knowledge Base, after performing one of the CRUD operations must reflect the pertinent changes, and the chatbot must be able to use the newest state of the data.

Table 8.22: User Story 22: Manage Knowledge Base with CRUD operations

US23: Define and formalise domain constraints for the integrations	
As an	<i>Administrator</i>
I want to	Both define and formalise domain constraints for potential integrations
So that	The correct alignment between the new integrations and the requirements can be ensured.
Acceptance criteria	<ul style="list-style-type: none"> • The system provides a way to define domain constraints for the integrations. • The domain constraints are imposed when integrating any applications, features or parameters.

Table 8.23: User Story 23: Define and formalise domain constraints for the integrations

Chatbot management

US24: Obtain personalised suggestions of actions	
As a	<i>Chatbot user</i>
I want to	Ask the chatbot to give me possible suggestions of actions
So that	I can have other actions suggested in case the initial recommendation it has given me does not fit what I want.
Acceptance criteria	<ul style="list-style-type: none">• The chatbots offers the user suggestions of actions to integrate with their initial action.• The chatbot, if the user asks for more suggestions, gives them other suggestions computed on their preferences, and updates the user profile accordingly.

Table 8.24: User Story 24: Obtain personalised suggestions of actions

US25: Obtain personalised suggestions of apps	
As a	<i>Chatbot user</i>
I want to	Ask the chatbot to give me possible suggestions of applications
So that	I can have other apps suggested in case the initial recommendation it has given me does not fit what I want.
Acceptance criteria	<ul style="list-style-type: none">• The chatbots offers the user suggestions of apps with which they can integrate the feature the user has chosen.• The chatbot, when the user chooses an app, updates the user profile accordingly.

Table 8.25: User Story 25: Obtain personalised suggestions of apps

US26: Obtain parameters used for an integration	
As a	<i>Chatbot user</i>
I want to	Ask the chatbot to give me the data it will use to complete the action it has recommended
So that	I can check everything that is going to be defined using the data from the action I made is correct or not.
Acceptance criteria	<ul style="list-style-type: none">• The chatbots offers the user the parameter integrations that will be used to complete the action. This means that it will show the user the data that will be used from its initial action to complete the suggested action.• The chatbot, when the user wants to change some of this data, updates it accordingly.

Table 8.26: User Story 26: Obtain parameters used for an integration

US27: Obtain custom parameters	
As a	<i>Chatbot user</i>
I want to	Know what parameters are left to specify for the target action
So that	I can define them with the correct values and let the chatbot act for me.
Acceptance criteria	<ul style="list-style-type: none"> The chatbots offers the user the parameters from the target action that have not been defined using the data from the initial action (the ones that do not have parameter integration).

Table 8.27: User Story 27: Obtain custom parameters

8.3 Non-functional requirements

The non-functional requirements are those that specify criteria that can be used to judge the operation of a system, rather than specific behaviours. In this section, the non-functional requirements of the system are presented.

Requirement	<i>Speed and Latency</i>
Description	The system has to be able to provide the requested in a reduced amount of time.
Justification	The system must be able to give the chatbot the information it requests quickly, so the final user has an efficient and seamless experience
Satisfaction condition	The requirement will be considered completed if the requests the chatbot requests are provided in less than 5 seconds.

Table 8.28: Non-functional requirement 1: Speed and Latency

Requirement	<i>Reliability and Availability</i>
Description	The system has to be always available for the users, and it should be resistant to errors.
Justification	The system must be highly reliable and available for the chatbot and the final users at all times, in order to avoid any possible disruption and be able to ensure constant access to the system.
Satisfaction condition	The system must be prepared to deal with any type of errors that can appear, and keep functioning correctly. In case of an interruption, the system must be prepared to handle it quickly and be back in a reduced amount of time.

Table 8.29: Non-functional requirement 2: Reliability and Availability

Requirement	<i>Scalability and Extensibility</i>
Description	The system must be able to scale its features without letting the chatbot and the users with no service.
Justification	The system has to be able to be extended without having an impact on the users' experience.
Satisfaction condition	The system should support its scalability, allowing the addition/-subtraction of features without causing any downtime or restriction when accessing the service.

Table 8.30: Non-functional requirement 3: Scalability and Extensibility

Requirement	<i>Access</i>
Description	The access to this service must be restricted.
Justification	The knowledge that is stored in the Knowledge Base has to be accessible only to authorized personnel: administrators and developers, to ensure the integrity of the service as well as securing the data.
Satisfaction condition	The access to the system should be restricted to admins and developers only. Some sort of authentication mechanisms must be added to strengthen the security of both the data and the Knowledge Base.

Table 8.31: Non-functional requirement 4: Access

Requirement	<i>Privacy</i>
Description	The user's data must be protected and the system must ensure its privacy.
Justification	The system must treat the data inside the Knowledge Base (especially all the related to the users) with uttermost privacy to accomplish privacy regulations and ensure the confidentiality of the user's information.
Satisfaction condition	The system must ensure user data protection. To achieve this, it must follow all the pertinent privacy regulations as well as having this data protected by some security measures, such as authorized access.

Table 8.32: Non-functional requirement 5: Privacy

Requirement	<i>Project Planning</i>
Description	Every feature that is intended to be done, it must have been previously defined.
Justification	A good project planning is a must when trying to develop a system. Making sure that all the required features are defined initially before starting the development process is key.
Satisfaction condition	All the desired features must be clearly defined, described and documented before starting the development. Also, time planning must be done and an overall consideration of all the aspects that involve the project must be treated.

Table 8.33: Non-functional requirement 6: Project Planning

Chapter 9

Design

In this chapter, the design of the knowledge base that is developed in this project is explained. Both the physical and logical architecture of the application are described, as well as the design patterns that have been used to develop it. Finally, the data model diagram is shown.

9.1 Basic structure

Considering the Chatbots4Mobile project as a whole, the physical structure of the solution is composed of a mobile phone that uses the chatbot and the knowledge base as a backend.

- **The mobile phone:** it is the device that the user uses to interact with the chatbot. It is the device that triggers the chatbot, and it is the device that receives suggestions from the chatbot.
- **The chatbot:** it is the application that the user interacts with. It is the one that receives the user's requests, and it is the one that provides the user with suggestions.
- **The chatbot's NLU component:** it is the component of the chatbot that is responsible for understanding the user's requests. It is the one that receives the user's requests, and it is the one that provides the chatbot with the user's requests in a structured way.
- **The knowledge base:** it is the backend of the application. It is the one that receives the requests from the chatbot, and the source of the information for it to provide the user with suggestions.

This thesis is focused on the knowledge base, as it is the one that is being developed in this project.

This architecture follows the 3-layer architecture, which is a software architecture pattern that separates the concerns of the application into three interconnected layers, as shown in Figure 9.1.

- **Presentation layer:** this layer is the one that interacts with the chatbot. It is responsible for receiving the user's requests, and it is also responsible for showing the results to the user.

- **Business layer:** this layer is the one that contains the business logic of the application. It is responsible for processing the user's requests, and it is also responsible for providing the results to the presentation layer.
- **Data layer:** this layer is the one that contains the data of the application. It is the one that stores the data, and it is the one that provides the data to the business layer.

In this case, the presentation layer is the mobile phone, more specifically the chatbot, the business layer and the data layer are both in the knowledge base.

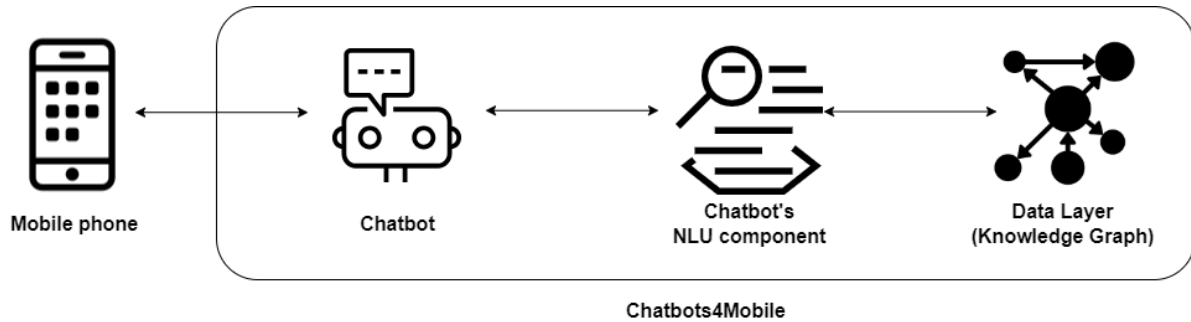


Figure 9.1: 3-layer architecture
Font: Own elaboration

It is important to mention that the following sections are going to focus on the knowledge base, more specifically on the Knowledge Graph, as it is the one that is being developed in this project.

9.2 Logical architecture

In terms of the logical architecture that has been chosen for the development of the knowledge-based system, due to the fact that Java Spring Boot is the framework that has been chosen for the development of the system, the most common architecture has been followed.

In this case, the Spring Boot flow architecture is the one shown in figure 9.2. This architecture, as can be seen, has a very simple flow, which is the following:

1. The client is the one who makes the HTTPS request to the server, although he is not the one explicitly who makes the request, but the chatbot.
2. The request is received by the knowledge base, arriving at the controllers, which are the ones that, based on the request, map the request to the corresponding service.
3. The service layer is the one doing the business logic, it is the one which will prepare the data to be sent to the client. In this case, the service will call the repository layer, which is the one that will access the database and perform the operations that are needed, as well as use the models defined, which are the ones that will be used to represent the data.
4. Once the service has the data, it will return it to the controller, which will be the one that will return the data to the client.

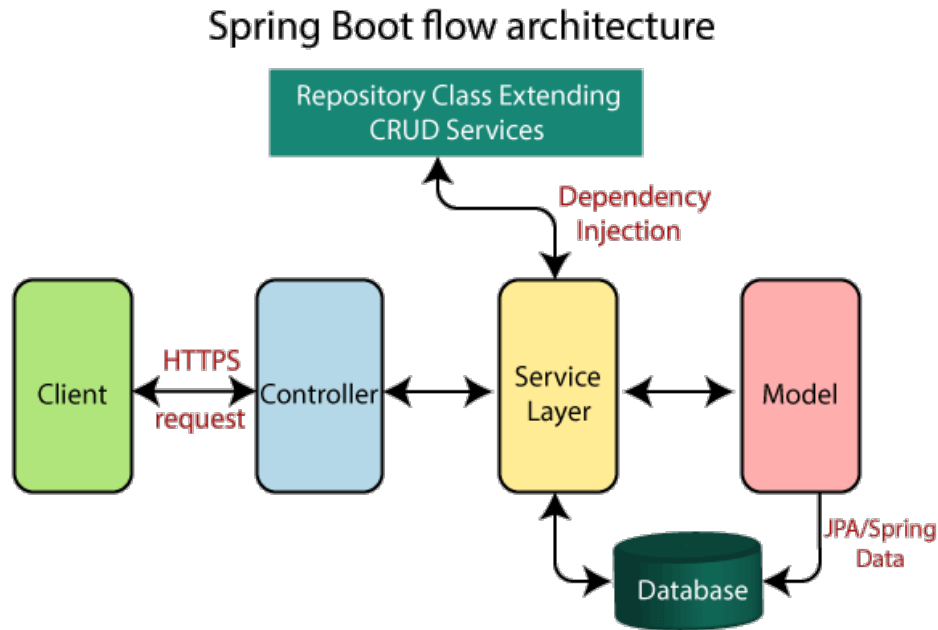


Figure 9.2: Spring Boot flow architecture
Font: javaTpoint [52]

Spring Boot is an extension of the Spring framework [53], which is a framework used as well to develop web applications, and it offers several modules that can be used to develop different parts of the application, such as the Spring Data module, which is the one that is used to access the database, or Spring MVC, which is the one that is used to develop the controllers and the services.

In this case, Spring Boot uses these modules that have been mentioned before. The architecture of Spring Boot is the same as the architecture of Spring MVC, but it is simplified, due to the fact that DAO (Data Access Object) and DAOImpl (Data Access Object Implementation) are not used, and the repository layer is used instead.

This type of architecture allows separating the functionality of the application into different layers, which allows a more modular application, which is easier to maintain and extend, as well as to test.

9.3 Design patterns

A design pattern can be described as a common approach to a recurring problem. Design patterns are a way to describe a solution to a problem that occurs in a specific context. They are not a finished design but a template, which can be used on several occasions when the necessities of the application allow it.

In this case, most Java Spring Boot projects usually follow a set of design patterns, and this project is not an exception. Two of the most common design patterns have been used, and will be explained next.

9.3.1 Dependency Injection

Dependency injection is a design pattern that allows the creation of loosely coupled modules. This means that the modules are not dependent on one another, the dependency itself is injected, rather than created explicitly within the class, allowing to have better testability and a much more modular design.

9.3.2 Singleton

One of the most common design patterns that can be found and used in several projects of different types is the singleton pattern. This pattern is used when there is the necessity of having one and only one instance of a class.

In this case, in Java Spring Boot, the singleton pattern is used to be able to manage shared resources. In this project, this has been used quite regularly. For instance, the repository classes are defined as singletons implicitly, when annotating them with the `@Repository` annotation. This is done as well with the controllers and the services. An example of how this is done can be seen in the following code snippet:

```
@org.springframework.stereotype.Repository
public class UserRepository {
    ...
}
```

9.4 Data model diagram

The data model diagram is a diagram that shows the different entities that are going to be used in the project, and the relations between them.

Due to the database that is going to be used, the data model diagram is not going to be a relational diagram, but a NoSQL diagram. This means that the relationships between the entities are not going to be represented as foreign keys but as references to other entities.

Although an example of how the data model diagram is going to look has been shown in figure 2.3, it may be quite difficult to see the general structure of the diagram. For this reason, a simplified version of the diagram is shown in figure 9.3. In this one, each entity has been defined so that it is easier to see the relations between them.

PREFIX schema <<https://schema.org>>
PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>
PREFIX rdfs: <<http://www.w3.org/2000/01/rdf-schema#>>

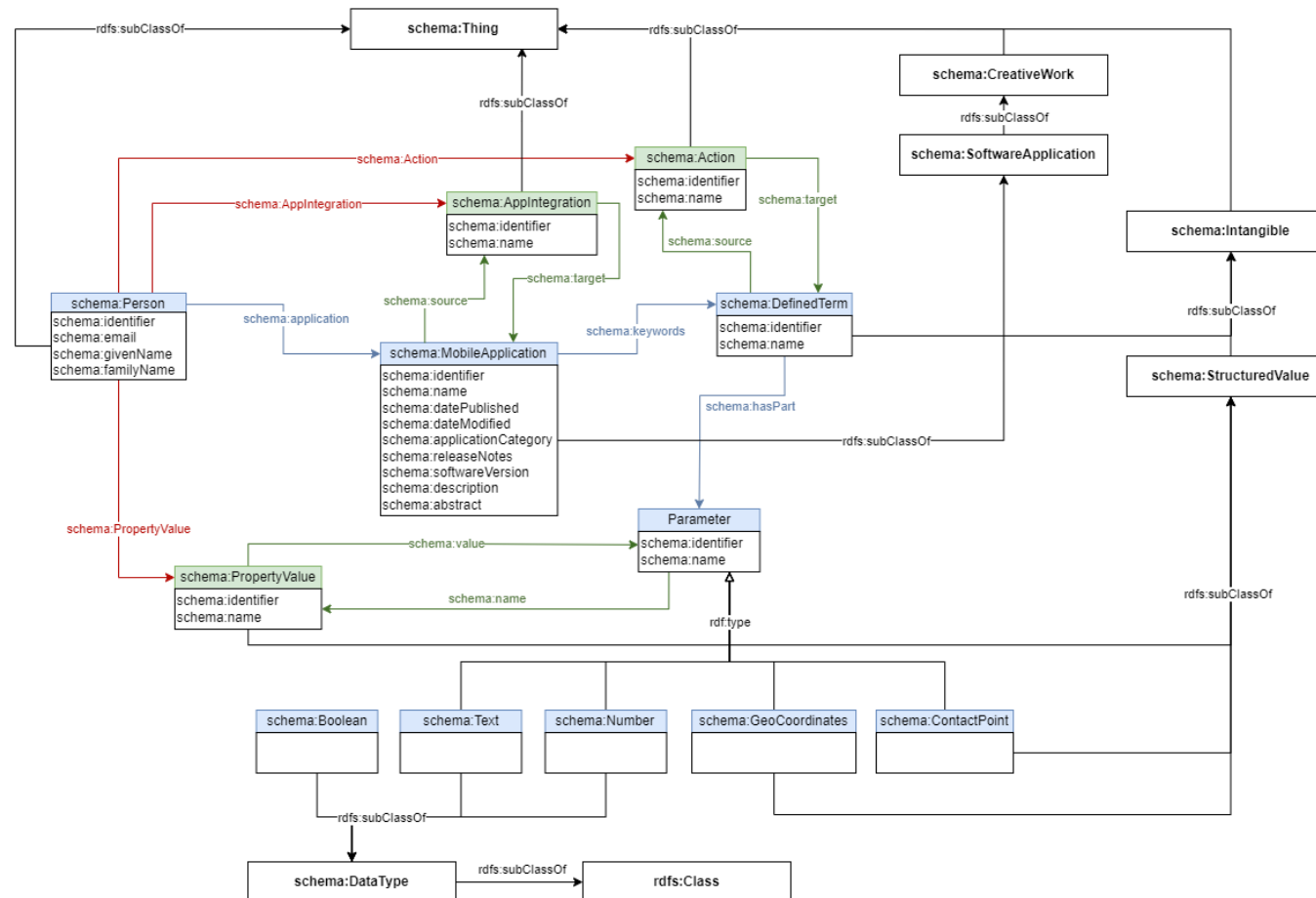


Figure 9.3: Data model diagram
Font: GESSI-Chatbots4Mobile

9.4.1 Entities and relations

Just to be able to fully understand the project's structure, the different entities (with their attributes) and relations are detailed, as well as the schema.org type for each of them is specified. For each entity, the domain constraints applied for each attribute (if any) are also specified. Those constraints are the ones that the data must follow in order to be valid, and due to our knowledge base being schema-less, all the restrictions are at the business logic level.

- **User (schema:Person)**: It represents a user of the application. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the user. Its domain constraint is that it must be unique, there cannot be two users with the same identifier.
 - **schema:email**: This is the email of the user. Its domain constraint is that it must be a mail address (it must contain an @ symbol).
 - **schema:givenName**: This is the given name of the user.
 - **schema:familyName**: This is the family name of the user.

And the following relations:

- **schema:application**: It connects the user with an App that he/she uses.
 - **schema:Action**: It connects the user with a preferred feature integration that he/she has.
 - **schema:PropertyValue**: It connects the user with a preferred parameter integration that he/she has.
 - **schema:AppIntegration**: It connects the user with a preferred app integration that he/she has.
- **App (schema:MobileApplication)**: It represents an application that the user uses. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the application. Its domain constraint is that it must be unique, there cannot be two applications with the same identifier.
 - **schema:name**: This is the name of the application.
 - **schema:datePublished**: This is the date when the application was published. Its domain constraint is that it must be a date which follows the Month/Day/Year format. (e.g. 12/31/2020)
 - **schema:dateModified**: This is the date when the application was last modified. Its domain constraint is that it must be a date which follows the Month/Day/Year format. (e.g. 12/31/2020)
 - **schema:applicationCategory**: This is the category of the application. Its domain constraint is that it has a set of possible values, and only those values are allowed. Those values are: *Trail Tracking*, *Sports Activity*, *POI Report*,

Calendar, GPS/Maps, Weather, Air quality, Instant Messaging, Task Manager, Notes.

- **schema:releaseNotes**: This is the release notes of the application.
- **schema:softwareVersion**: This is the version of the application.

And the following relations:

- **schema:description**: This is the description of the application.
- **schema:abstract**: This is the summary of the application.
- **schema:keywords**: It connects the app with a feature that it exposes.
- **Feature (schema:DefinedTerm)**: It represents a feature of an application. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the feature. Its domain constraint is that it must be unique, there cannot be two features with the same identifier.
 - **schema:name**: This is the name of the feature.

And the following relations:

- **schema:hasPart**: It connects the feature with a parameter that it has.
- **Parameter (schema:Boolean / schema:Number / schema:Text / schema:GeoCoordinates / schema:Contact)**: It represents a parameter of a feature. Depending on the type of the parameter, it will be defined as one of the specified types. This is limited but the idea is to allow the extension of the types based on the app's domain and the features they expose and covered by the chatbot. To support that only those types are allowed, the domain constraint is that when creating a parameter its type has to be explicitly specified, and it has to be one of the presented types. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the parameter. Its domain constraint is that it must be unique, there cannot be two parameters with the same identifier.
 - **schema:name**: This is the name of the parameter.
- **AppIntegration (schema:AppIntegration)**: It represents the integration of two apps. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the app integration. Its domain constraint is that it must be unique, there cannot be two app integrations with the same identifier (that is that there cannot be two app integrations with the same source and target apps).
 - **schema:name**: This is the name of the app integration.

And the following relations:

- **schema:source**: This is the source app of the integration. Its domain constraint is that it must be an app that exists in the system.

- **schema:target**: This is the target app of the integration. Its domain constraint is that it must be an app that exists in the system.
- **FeatureIntegration (schema:Action)**: It represents the integration of two features from different apps. It has the following attributes:
 - **schema:identifier**: This is the unique identifier of the feature integration. Its domain constraint is that it must be unique, there cannot be two feature integrations with the same identifier (that is that there cannot be two feature integrations with the same source and target features).
 - **schema:name**: This is the name of the feature integration.

And the following relations:

- **schema:source**: This is the source feature of the integration. Its domain constraint is that it must be a feature that exists in the system.
- **schema:target**: This is the target feature of the integration. Its domain constraint is that it must be a feature that exists in the system.
- **ParameterIntegration (schema:PropertyValue)**: It represents the integration of two parameters from two features. It has the following attribute:
 - **schema:identifier**: This is the unique identifier of the parameter integration. Its domain constraint is that it must be unique, there cannot be two parameter integrations with the same identifier (that is that there cannot be two parameter integrations with the same source and target parameters).
 - **schema:name**: This is the name of the parameter integration.

And the following relations:

- **schema:name**: This is the source parameter of the integration. Its domain constraint is that it must be a parameter that exists in the system.
- **schema:value**: This is the target parameter of the integration. Its domain constraint is that it must be a parameter that exists in the system.

Chapter 10

Implementation process

In this chapter, all the processes related to what has been implemented are described. The process is divided into three main parts: the original project's state, the data schema extension and the knowledge base extension.

As it has been previously mentioned, this Final Degree Project has focused on the development of the knowledge base extension, that is, the back-end knowledge base for the chatbot. Therefore, everything that is mentioned in this chapter is related to the back-end, and images of real examples of how the data is structured in the knowledge base are shown.

10.1 System overview

The initial knowledge base was provided by the Chatbots4Mobile research project infrastructure, and what could be seen there was a database without the formalisation of the concept of a feature from the software engineering perspective (e.g. parameters, integrations), nor anything related to working with the user's information (e.g. user preferences). The data was composed, as it has been mentioned and shown in the section *1.2 Concepts*, of (1) a catalogue of mobile applications known by the chatbot, (2) a set of users with any kind of information about them (in terms of neither attributes nor preferences) and (3) a set of features that the mobile applications have (they are the ones that trigger the chatbot to enter into action). An example of how a user was represented in the knowledge base is shown in figure 10.1². These shown nodes are the ones that have a direct relationship with our thesis, although there are more types of entities defined, they are not inside this project's scope, so only the relevant ones for our case are displayed. As it can be seen, the knowledge base did not have any information related to the formalisation of a feature, and that was the main goal of this project.

By checking the image and understanding that Schema.org is the vocabulary that is used to represent the data in the knowledge base, it can be seen that the user is represented as a *schema:Person* and has a relation with the *schema:MobileApplication* called *com.strava* [54], which exposes several features (*schema:keywords*), as well as it also has a description of the application itself.

²All the images that will be shown of the knowledge base represent unreal users that were created for testing purposes. Any real user has been shown in order to protect and maintain their privacy.

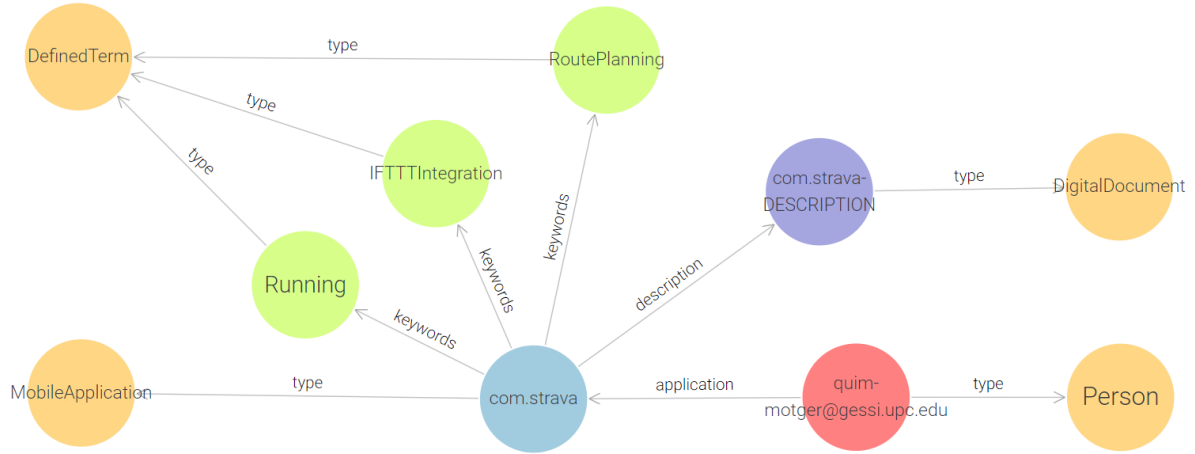


Figure 10.1: Example of how users were represented in the initial knowledge base
Font: Own elaboration

The main goal of this project, as a reminder, was to extend the knowledge base in order to make the conversational agent able to offer a personalised experience to the user, by considering their preferences and actions performed across the apps the chatbot is aware of, and that is done by (1) formalising both features and feature integrations and (2) personalizing the feature integrations.

Therefore, after considering the initial state of the knowledge base and having a clear idea of what the goal of this project defined by GESSI (owner of the Chatbots4Mobile parent project) was, the next step was to define the new schema that allowed us to move from the initial database to a *context-aware* knowledge base.

10.2 Development resources

10.2.1 Technologies and frameworks used

- **Java:** a high-level, class-based, object-oriented programming language developed and maintained by Oracle. [55]
- **SPARQL:** a query language for databases that retrieve data stored in the Resource Description Framework (RDF) format. [56]
- **Java Spring Boot:** an open source Java-based framework used to create a micro Service, developed by Pivotal Software, Inc. [57]
- **RDF:** a standard model for data interchange on the Web. In this case, it is used to represent the data of the application. [38]
- **RDF4J:** a Java framework for processing RDF data. It is distributed under the terms of the Eclipse Public License, and it is well-integrated with the Spring Framework. [58]
- **Springdoc:** a framework that allows the automatic generation of Swagger documentation for the Spring Boot REST API. [59]

- **JUnit**: a testing framework for the Java programming language. [60]

10.2.2 Development tools

- **Visual Studio Code**: a source-code editor developed by Microsoft for Windows, Linux and macOS. [23]
- **Git**: a distributed version-control system for tracking changes in source code during software development. [20]
- **GraphDB**: an RDF triplestore developed by Ontotext [25]. For this project, it is used to:
 - Store the RDF data.
 - Execute SPARQL queries.
 - Manage the RDF data.
 - Visualize the RDF data.

10.3 Data schema extension

Once the initial knowledge base was analysed, it was time to start considering what should be done in order to extend it and transform it into a context knowledge base.

10.3.1 Creation of the data schema extension

The first thing that was required to do was to redefine the data schema. There were several questions that were emerging and that needed to be answered. The first question was: **With what should the knowledge base be extended to support feature formalisation (including feature integrations) and user preferences?** The answer to this question was not easy, due to the fact that some redesign of the data schema was required.

After having some discussions with the project's director, two main actions were decided to be done. The first one was to complement the existing data schema by (1) adding parameters to each feature, in order to be able to represent the different values that they can have and facilitate the integration between them and (2) adding the possibility of representing the user preferences.

The first action was done by defining that each feature would have a set of input/output parameters that would define the different values that the feature requires to be performed (if any). With this, we obtain a much more precise representation of the features, and we can now define the integrations between them in a much more accurate way.

The second action was decided to be done by adding new entities to the data schema that would allow us to represent for each user what preferences he has got in terms of applications and in terms of specific features these apps expose. This way, the agent would be able to know about the user's historical integrations and would be able to recommend him new ones based on that. There were 3 types of integrations that were considered and finally added:

- **Application to application:** this type of integration would let us know what applications the user usually integrates, without specifying the features that are integrated (multiple features can be integrated between two applications). For instance, there can be a user that usually plans a route in Strava and then creates an event in Google Calendar [61] with the route's information, or they can also add photos of a route into Strava and wants to add to a Google Calendar event the photos that were added to Strava. We will end up obtaining that the user has an app integration between Strava and Google Calendar defined.
- **Feature to feature:** this type of integration would let us know what features the user usually integrates, without specifying the applications that expose those features (multiple applications can expose the same feature). For instance, there can be a user that usually plans a route in Strava when they go hiking and then creates an event in Google Calendar with the route's information, or they can also plan a route to make tourism using Google Maps and wants to add to a Google Calendar event the route that was planned. We will end up obtaining only one feature integration which involves the act of planning a route and creating an event, no matter what applications are used to do so. By specifying the features that are integrated, we can know more about the user's preferences and recommend him a much more accurate integration.
- **Parameter to parameter:** this type of integration would let us automate the process of integrating two applications' features (depending on the features that are being integrated all the required parameters for the target feature can be defined totally defined if the values obtained from the source feature are enough, partially otherwise), so the user action would be reduced to just confirm the integration or, in some cases, having to specify fewer parameters than if the action was done manually by them.

With this in mind, we were able to make the design of the new data schema. The result of this design was shown in section 9.4 *Data model diagram*.

The second question was related to the first one: **What can be used to represent the new entities and relations?** Considering the fact that both RDF and schema.org were used in the initial knowledge base, it was decided to keep using them in the extended version. The reason for this is that they are the most used technologies in the semantic web, as was explained in chapter 7 *State of the art*. So, the next step was to analyse what schema from schema.org could be used in our scenario, in order to avoid the creation of new ones when possible. The final result of this analysis was previously shown along with the data model diagram. We finally managed to reuse resources from schema.org to represent the new entities and relations, except for the *AppIntegration* entity and the *source* feature in a feature integration. With this new schema, the chatbot would be able to have more information about the user and more possible actions to make whenever the user performed any action. Now an example of how the data for some users would be represented can be seen in figure 10.2.

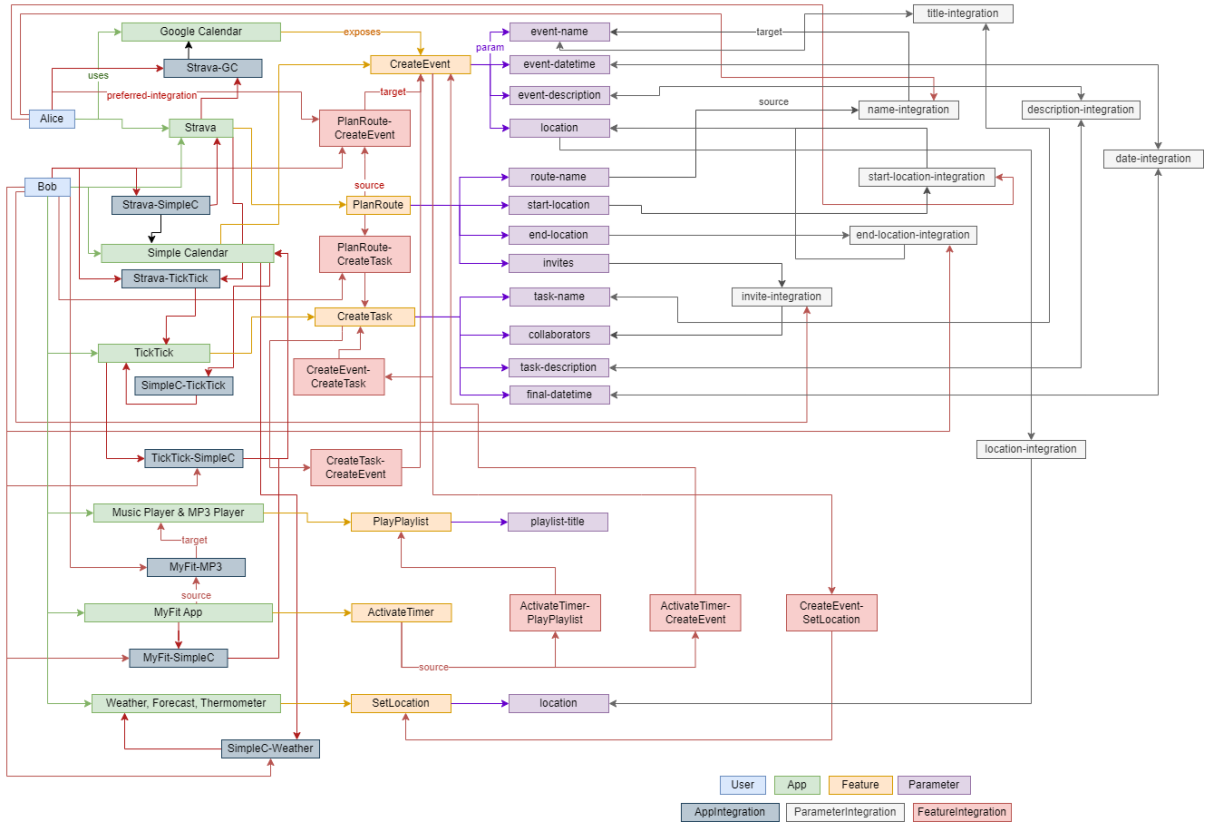


Figure 10.2: Example of how the data for Alice and Bob users would be represented in the extended data schema
Font: Own elaboration

Let's consider the user Bob again, and let's suppose that he usually performs the following: he usually plans a route in Strava and then he creates an event in Simple Calendar with the route's information. Now, with this new data schema, we can check that this information is represented in the knowledge base in several ways. First, we can see that there is an *AppIntegration* entity that represents the integration between Strava and Google Calendar. Then, we can see that there is a *FeatureIntegration* entity that represents the integration between the features *PlanRoute* and *CreateEvent*. Finally, we can see that there is a *ParameterIntegration* entity that represents the integration between the parameters *route-name/event-name* and *start-location/location* from these features. We can see that Bob has those integrations as preferred, due to the chatbot having now the ability to store what the user prefers. With this, whenever Bob performs the planning of a route in Strava, the chatbot will estimate which is the most possible suggestion he would like to receive based on his preferences, and in case of doing it, he would use the parameter integrations defined to be able to automatically defines the one which is possible (the ones that are the target of the parameter integrations, the rest should be provided by the user). Now, the process of integrating two applications' features can be estimated with the knowledge base's data and it can be done in an efficient way, letting the user just perform the action and confirm the integration.

It is important to clarify that the responsible for when and how the user preferences are updated is the chatbot. The extended knowledge base simply provides a flexible mechanism to add/update/delete these preferences and persist them so that their access is both clear and efficient. The decision-making part is beyond the scope of this work.

10.3.2 Domain constraints

The domain constraints are the rules that are going to be used in the knowledge base in order to insert new data, as well as to make sure that the data the chatbot will consume follow always the same format and there are no ambiguities or errors in it.

The domain constraints defined for the extended data schema were the following:

- **Unique identifier:** each entity in the knowledge base has an identifier defined. This identifier has to be unique for each entity, so there cannot be two entities with the same identifier. With this, we make sure that the data is not duplicated and that the data is consistent. It is important to clarify that the identifiers have not been generated, but specific properties that are unique for each entity have been used instead, allowing us to facilitate the accessibility to the data.
- **Data completeness:** each entity in the knowledge base has to follow the data schema that was defined for it. This schema has some properties that are required and some that are optional. With this approach, we expose a compromised solution with respect to a full schemaless knowledge graph by forcing the presence and validity of some properties which are required for the correct modelling - and therefore, execution - of feature integrations.
- **Data integrity constraints:** the data that is inserted in the knowledge base has to be consistent with the data that is already in it. This means that the relations between the entities have to be consistent and that the entities that are related to each other have to exist on both sides of the relation. This is done in order to make sure the chatbot uses data that exists and does not try to compute suggestions with inexistent data.

There are some more constraints that are usually followed for every database, such as handling the null values or the data types, but the most relevant ones that make our system work and make it scalable are the ones that were explained above.

10.4 Knowledge base operations

In this section, all the API operations that are related to the knowledge base are described. Also, the API documentation generated using Springdoc in Swagger [62] can be found in appendix A.

10.4.1 Entity definitions

Users

In order to be able to work with users, the CRUD operations have been defined.

Considering that a *User* object looks like the following:

```
1 {
2   "identifier*": "string",
3   "email*": "string",
4   "givenName*": "string",
5   "familyName*": "string",
6   "apps": [
7     "string"
8   ],
9   "preferredFeatureIntegrations": [
10    "string"
11  ],
12  "preferredParameterIntegrations": [
13    "string"
14  ],
15  "preferredApps": [
16    "string"
17  ]
18 }
```

Description	Get all the users
Method	GET
Endpoint	/users
Responses	200: User[]

Description	Get a specific user by ID
Method	GET
Endpoint	/users/:id
Parameters	id: User's unique identifier
Responses	200: User 404: Not Found

Description	Create a user
Method	POST
Endpoint	/users
Body	User
Responses	201: Created 400: Bad Request 409: User already exists

Description	Update a user
Method	PUT
Endpoint	/users/:id
Parameters	id: User's unique identifier
Body	User
Responses	200: Updated 404: Not Found

Description	Delete a user
Method	DELETE
Endpoint	/users/:id
Parameters	id: User's unique identifier
Responses	204: No Content 404: Not Found

Mobile Applications

In order to be able to work with mobile applications, the CRUD operations have been defined.

Considering that a *MobileApplication* object looks like the following:

```

1 {
2   "name*": "string",
3   "identifier*": "string",
4   "description*": "string",
5   "summary*": "string",
6   "releaseNotes*": "string",
7   "applicationCategory*": "AppCategory",
8   "datePublished*": "string",
9   "dateModified*": "string",
10  "softwareVersion*": "string",
11  "features*": [
12    "string"
13  ]
14 }
```

Description	Get all the mobile apps
Method	GET
Endpoint	/apps
Responses	200: MobileApplication[]

Description	Get a specific mobile app by ID
Method	GET
Endpoint	/apps/:id
Parameters	id: Mobile application's unique identifier
Responses	200: MobileApplication 404: Not Found

Description	Create a mobile app
Method	POST
Endpoint	/apps
Body	MobileApplication
Responses	201: Created 400: Bad Request 409: MobileApplication already exists

Description	Update a mobile app
Method	PUT
Endpoint	/apps/:id
Parameters	id: Mobile application's unique identifier
Body	MobileApplication
Responses	200: Updated 404: Not Found

Description	Delete a mobile app
Method	DELETE
Endpoint	/apps/:id
Parameters	id: Mobile Application's unique identifier
Responses	204: No Content 404: Not Found

App Integrations

In order to be able to work with the integration of Apps, the CRUD operations have been defined.

Considering that an *AppIntegration* object looks like the following:

```
1 {  
2   "identifier": "string",  
3   "name": "string",  
4   "sourceApp*": "string",  
5   "targetApp*": "string"  
6 }
```

Description	Get all the app integrations
Method	GET
Endpoint	/apps/integrations
Responses	200: AppIntegration[]

Description	Get a specific app integration by ID
Method	GET
Endpoint	/apps/integrations/:id
Apps	id: App integration's unique identifier
Responses	200: AppIntegration 404: Not Found

Description	Create an app integration
Method	POST
Endpoint	/Apps/integrations
Body	AppIntegration
Responses	201: Created 400: Bad Request 409: AppIntegration already exists

Description	Update an app integration
Method	PUT
Endpoint	/apps/integrations/:id
Apps	id: App integration's unique identifier
Body	AppIntegration
Responses	200: Updated 404: Not Found

Description	Delete an app integration
Method	DELETE
Endpoint	/apps/integrations/:id
Apps	id: App integration's unique identifier
Responses	204: No Content 404: Not Found

Features

In order to be able to work with features, the CRUD operations have been defined.

Considering that a *Feature* object looks like the following:

```

1 {
2   "identifier": "string",
3   "name*": "string",
4   "parameters*": [
5     "string"
6   ]
7 }
```

Description	Get all the features
Method	GET
Endpoint	/features
Responses	200: Feature[]

Description	Get a specific feature by ID
Method	GET
Endpoint	/features/:id
Parameters	id: Feature's unique identifier
Responses	200: Feature 404: Not Found

Description	Create a feature
Method	POST
Endpoint	/features
Body	Feature
Responses	201: Created 400: Bad Request 409: Feature already exists

Description	Update a feature
Method	PUT
Endpoint	/features/:id
Parameters	id: Feature's unique identifier
Body	Feature
Responses	200: Updated 404: Not Found

Description	Delete a feature
Method	DELETE
Endpoint	/features/:id
Parameters	id: Feature's unique identifier
Responses	204: No Content 404: Not Found

Feature Integrations

In order to be able to work with the integration of features, the CRUD operations have been defined.

Considering that a *FeatureIntegration* object looks like the following:

```

1 {
2   "identifier": "string",
3   "name": "string",
4   "sourceFeature*": "string",
5   "targetFeature*": "string"
6 }
```

Description	Get all the feature integrations
Method	GET
Endpoint	/features/integrations
Responses	200: FeatureIntegration[]

Description	Get a specific feature integration by ID
Method	GET
Endpoint	/features/integrations/:id
Parameters	id: Feature integration's unique identifier
Responses	200: FeatureIntegration 404: Not Found

Description	Create a feature integration
Method	POST
Endpoint	/features/integrations
Body	FeatureIntegration
Responses	201: Created 400: Bad Request 409: FeatureIntegration already exists

Description	Update a feature integration
Method	PUT
Endpoint	/features/integrations/:id
Parameters	id: Feature integration's unique identifier
Body	FeatureIntegration
Responses	200: Updated 404: Not Found

Description	Delete a feature integration
Method	DELETE
Endpoint	/features/integrations/:id
Parameters	id: Feature integration's unique identifier
Responses	204: No Content 404: Not Found

Parameters

In order to be able to work with parameters, the CRUD operations have been defined.

Considering that a *Parameter* object looks like the following:

```

1 {
2
3   "identifier*": "string",
4   "name*": "string",
5   "type*": : "ParameterType"
6 }
```

where ParameterType can have 5 possible values: [Number, Boolean, Text, GeoCoordinates, ContactPoint].

Description	Get all the parameters
Method	GET
Endpoint	/parameters
Responses	200: Parameter[]

Description	Get a specific parameter by ID
Method	GET
Endpoint	/parameters/:id
Parameters	id: Parameter's unique identifier
Responses	200: Parameter 404: Not Found

Description	Create a parameter
Method	POST
Endpoint	/parameters
Body	Parameter
Responses	201: Created 400: Bad Request 409: Parameter already exists

Description	Update a parameter
Method	PUT
Endpoint	/parameters/:id
Parameters	id: Parameter's unique identifier
Body	Parameter
Responses	200: Updated 404: Not Found

Description	Delete a parameter
Method	DELETE
Endpoint	/parameters/:id
Parameters	id: Parameter's unique identifier
Responses	204: No Content 404: Not Found

Parameter Integrations

In order to be able to work with the integration of parameters, the CRUD operations have been defined.

Considering that a *ParameterIntegration* object looks like the following:

```
1 {  
2   "identifier": "string",  
3   "sourceParameter*": "string",  
4   "targetParameter*": "string"  
5 }
```

Description	Get all the parameter integrations
Method	GET
Endpoint	/parameters/integrations
Responses	200: ParameterIntegration[]

Description	Get a specific parameter integration by ID
Method	GET
Endpoint	/parameters/integrations/:id
Parameters	id: Parameter integration's unique identifier
Responses	200: ParameterIntegration 404: Not Found

Description	Create a parameter integration
Method	POST
Endpoint	/parameters/integrations
Body	ParameterIntegration
Responses	201: Created 400: Bad Request 409: ParameterIntegration already exists

Description	Update a parameter integration
Method	PUT
Endpoint	/parameters/integrations/:id
Parameters	id: Parameter integration's unique identifier
Body	ParameterIntegration
Responses	200: Updated 404: Not Found

Description	Delete a parameter integration
Method	DELETE
Endpoint	/parameters/integrations/:id
Parameters	id: Parameter integration's unique identifier
Responses	204: No Content 404: Not Found

10.4.2 User preferences

Several operations have been defined in order to add and remove preferences from a user.

App integration preferences

Description	Add a preferred app integration to a user
Method	POST
Endpoint	/users/:id/integrations/apps
Parameters	id: User's unique identifier
Body	AppIntegration
Responses	201: Created 404: Not Found

Description	Delete a preferred app integration to a user
Method	DELETE
Endpoint	/users/:id/integrations/apps
Parameters	id: User's unique identifier
Responses	204: No Content 404: Not Found

Feature integration preferences

Description	Add a preferred feature integration to a user
Method	POST
Endpoint	/users/:id/integrations/features
Parameters	id: User's unique identifier
Body	FeatureIntegration
Responses	201: Created 404: Not Found

Description	Delete a preferred feature integration to a user
Method	DELETE
Endpoint	/users/:id/integrations/features
Parameters	id: User's unique identifier
Responses	204: No Content 404: Not Found

Parameter integrations preferences

Description	Add a preferred parameter integration to a user
Method	POST
Endpoint	/users/:id/integrations/parameters
Parameters	id: User's unique identifier
Body	ParameterIntegration
Responses	201: Created 404: Not Found

Description	Delete a preferred parameter integration to a user
Method	DELETE
Endpoint	/users/:id/integrations/parameters
Parameters	id: User's unique identifier
Responses	204: No Content 404: Not Found

10.4.3 Suggestions computation

In order to be able to offer suggestions to the users based on their actions, the following operations have been defined.

The first operation that has been defined allows us to obtain, based on the action the user has just made (i.e., the feature enacted in the user's smartphone device), the list of features that can be integrated with the one that has been used. Although all the features that can be integrated with the one that has been used are returned, it is a sorted list, where the first feature is the one that is most likely to be integrated with the one that has been used, according to the user preferences (more specifically, the preferred feature integrations are considered the most likely to be integrated, and then the app integrations are taken into account). The endpoint of this operation is the following:

Description	Request feature integrations from source features and previous user preferences
Method	GET
Endpoint	/users/:id/integrations/features/:source
Parameters	id: User's unique identifier source: feature that acts as the source of the integration
Responses	200: Feature: string[] 404: Not Found

An example of a SPARQL query that this query has defined to obtain the features basing specifically on the user preferences is shown in figure 10.3.

SPARQL Query & Update ⓘ

Editor only Editor and results Results only ⓘ

Table Raw Response Pivot Table Google Chart Download as

Filter query results Showing results from 1 to 1 of 1. Query took 0.1s, minutes ago.

	appName
1	"Simple Calendar"

```

1 PREFIX schema: <https://schema.org/>
2 SELECT DISTINCT ?appName
3 WHERE {
4   ?user a schema:Person .
5   ?user schema:identifier 'bob@essi.upc.edu' .
6   ?user schema:application ?app .
7   ?app a schema:MobileApplication .
8   ?app schema:name ?appName .
9   ?app schema:keywords ?feature .
10  ?feature a schema:DefinedTerm;
11         schema:identifier 'CreateEvent'
12 }

```

Figure 10.3: SPARQL query to request feature integrations from source features and previous user preferences
Font: Own elaboration

The second operation that has been defined allows us to obtain, based on a source and a target feature, the list of apps that can perform the target feature. Although all the apps that have the target feature defined are returned, it is a sorted list, where the first app is the one that is most likely to be used by the user, according to their preferences (more specifically, the preferred apps that expose this feature are considered the most likely to be used, and then the apps that the user uses that expose this feature are taken into account). The endpoint of this operation is the following:

Description	Request app integrations from a target feature and previous user preferences
Method	GET
Endpoint	/users/:id/integrations/apps/feature/:source/:target
Parameters	id: User's unique identifier source: feature that acts as the source of the integration target: feature that acts as the target of the integration
Responses	200: MobileApplication: string[] 404: Not Found

An example of an SPARQL query that this call has defined to obtain the apps that expose the given target feature basing on the user preferences is shown in figure 10.4.

SPARQL Query & Update ⓘ

Editor only

Editor and results

Results only

⊞

Table

Raw Response

Pivot Table

Google Chart

Download as

Unamed ×

Unamed ×

Unamed ×

Unamed ×

Unamed ×

Unamed ×

⊕

Unamed ×

Unamed ×

Unamed ×

Unamed ×

Unamed ×

```

1 PREFIX schema: <https://schema.org/>
2 SELECT DISTINCT ?appName WHERE {
3 {
4   ?user a schema:Person .
5   ?user schema:identifier 'bob@essi.upc.edu' .
6   ?user schema:AppIntegration ?appIntegration .
7   ?appIntegration a schema:AppIntegration .
8   ?appIntegration schema:source ?app .
9   ?app a schema:MobileApplication .
10  ?app schema:name ?appName .
11  ?app schema:keywords ?feature .
12  ?feature a schema:DefinedTerm;
13           schema:identifier 'PlanRoute'
14 } UNION
15 {
16   ?user a schema:Person .
17   ?user schema:identifier 'bob@essi.upc.edu' .
18   ?user schema:AppIntegration ?appIntegration .
19   ?appIntegration a schema:AppIntegration .
20   ?appIntegration schema:target ?app .
21   ?app a schema:MobileApplication .
22   ?app schema:name ?appName .
23   ?app schema:keywords ?feature .
24   ?feature a schema:DefinedTerm;
25           schema:identifier 'CreateEvent'
26 }
27 }

```

Run

keyboard shortcuts

Filter query results

Showing results from 1 to 2 of 2. Query took 0.1s, moments ago.

	appName
1	"Strava_Run_Ride_Hike"
2	"Simple Calendar"

Figure 10.4: SPARQL query to request app integrations from a target feature and previous user preferences
Font: Own elaboration

The third operation that has been defined allows us to obtain the list of parameter integrations that can be performed between two features from two different apps. The endpoint of this operation is the following:

Description	Request source-target parameter integrations for selected app
Method	GET
Endpoint	/parameters/integrations/request
Body	<pre>{ "sourceApp": "string", "sourceFeature": "string", "targetApp": "string", "targetFeature": "string" }</pre>
Responses	200: ParameterIntegrations: string[] 404: Not Found

An example of an SPARQL query that this call has defined to obtain the parameter integrations between two features from two different apps is shown in figure 10.5.

SPARQL Query & Update ⓘ

Unnamed X Unnamed X Unnamed X Unnamed X Unnamed X Unnamed X

```

1 PREFIX schema: <https://schema.org/>
2 SELECT ?sourceParam ?targetParam
3 WHERE {
4   ?sourceApp a schema:MobileApplication;
5             schema:identifier 'com.strava';
6             schema:name ?sourceAppName;
7             schema:keywords ?sourceFeature.
8   ?sourceFeature a schema:DefinedTerm;
9                 schema:identifier 'PlanRoute'.
10  ?targetApp a schema:MobileApplication;
11            schema:identifier
12            'com.simplenobiletools.calendar';
13            schema:name ?targetAppName;
14            schema:keywords ?targetFeature.
15  ?targetFeature a schema:DefinedTerm;
16                 schema:identifier 'CreateEvent'.
17  ?sourceFeature schema:hasPart ?sourceParam.
18  ?targetFeature schema:hasPart ?targetParam.
19  ?parameterIntegration a schema:PropertyValue;
20                        schema:name ?sourceParam;
21                        schema:value ?targetParam.
22 }
```

Editor only Editor and results Results only

Table Raw Response Pivot Table Google Chart Download as

Filter query results Showing results from 1 to 6 of 6. Query took 0.1s, moments ago.

	sourceParam	targetParam
1	https://schema.org/Text/route-name	https://schema.org/Text/event-name
2	https://schema.org/Text/route-name	https://schema.org/Text/event-name
3	https://schema.org/GeoCoordinates/start-location	https://schema.org/GeoCoordinates/location
4	https://schema.org/GeoCoordinates/start-location	https://schema.org/GeoCoordinates/location
5	https://schema.org/GeoCoordinates/end-location	https://schema.org/GeoCoordinates/location
6	https://schema.org/GeoCoordinates/end-location	https://schema.org/GeoCoordinates/location

Figure 10.5: SPARQL query to request source-target parameter integrations for selected app
Font: Own elaboration

The fourth and last operation that has been defined allows us to obtain the list of parameters that must be specified by the user when integrating two features from two different apps, due to them not having any parameter integration defined for this case. The endpoint of this operation is the following:

Description	Request custom parameters for selected app
Method	GET
Endpoint	/parameters/integrations/request/custom
Body	<pre>{ "sourceApp": "string", "sourceFeature": "string", "targetApp": "string", "targetFeature": "string" }</pre>
Responses	200: Parameters: string[] 404: Not Found

An example of an SPARQL query that this call has defined to obtain the parameters that must be specified by the user when integrating two features from two different apps is shown in figure 10.6.

SPARQL Query & Update ⓘ

Editor onlyEditor and resultsResults only

TableRaw ResponsePivot TableGoogle ChartDownload as

Filter query resultsShowing results from 1 to 2 of 2. Query took 0.1s, moments ago.

	targetParamId
1	"event-name"
2	"location"

1234567891011121314151617181920212223

```
1 PREFIX schema: <https://schema.org/>
2 SELECT DISTINCT ?targetParamId
3 WHERE {
4   ?sourceApp a schema:MobileApplication;
5             schema:identifier 'com.strava';
6             schema:keywords ?sourceFeature .
7   ?sourceFeature a schema:DefinedTerm;
8                 schema:identifier 'PlanRoute'.
9   ?sourceFeature schema:hasPart ?sourceParam.
10  ?sourceParam schema:identifier ?sourceParamId.
11  ?targetApp a schema:MobileApplication;
12            schema:identifier 'com.simplermobiletools.calendar';
13            schema:keywords ?targetFeature .
14  ?targetFeature a schema:DefinedTerm;
15                schema:identifier 'CreateEvent'.
16  ?targetFeature schema:hasPart ?targetParam.
17  ?targetParam schema:identifier ?targetParamId.
18  ?paramInt a schema:PropertyValue;
19            schema:name ?sourceInt;
20            schema:value ?targetInt.
21  ?sourceInt schema:identifier ?sourceParamId.
22  ?targetInt schema:identifier ?targetParamId
23 }
```








Figure 10.6: SPARQL query to request custom parameters for selected app
Font: Own elaboration

Chapter 11

Testing

In this chapter, all the testing that has been done on this project will be mentioned and explained, as well as the results obtained after performing the tests.

There are four types of testing levels that can be done to a software system: *unit testing*, *integration testing*, *system testing* and *acceptance testing* [63]. Due to the scope of this project, only the first two types of testing have been done, due to the current state of the Chatbots4Mobile general project, which is still in development.

11.1 Knowledge base test dataset

There have been several aspects that have been considered when evaluating the functionality of what has been implemented:

- **Test dataset:** In order to check the functionality of the CRUD operations defined, they have been used to define a dataset with which other functionalities could be tested. This dataset is the one from figure 10.2 and has the following:
 - **Users:** 2 mock users have been defined (*Alice and Bob*).
 - **Apps:** 7 apps have been defined, most of them being reused from the original dataset given. A total of 6 **features** are defined, and each of those applications has, at least, a **feature integration** defined for its feature exposed, having a total of 7. These apps and their features (with their parameters) are the following:
 - * **Strava:**
 - *Plan a route: allows the user to plan a route (e.g. running, cycling, etc.) specifying the name, where it starts and where it ends, and the persons that are invited to join (if any).*
 - *Parameters: [route-name, start-location, end-location, invites]*
 - * **Google Calendar:**
 - *Create an event: allows the user to create an event specifying its name, date, a brief description and its location.*

- *Parameters: [event-name, event-datetime, event-description, location]*
- * **Simple Calendar:**
 - *Create an event: allows the user to create an event specifying its name, date, a brief description and its location.*
 - *Parameters: [event-name, event-datetime, event-description, location]*
- * **TickTick:**
 - *Create a task: allows the user to create a task specifying its name, a brief description, the collaborators (i any) and the final date and time.*
 - *Parameters: [task-name, task-description, collaborators, final-datetime]*
- * **Music Player & MP3 Player:**
 - *Play a playlist: allows the user to play a specific playlist.*
 - *Parameters: [playlist-title]*
- * **MyFit:**
 - *Activate a timer: allows the user to start a timer.*
- * **Weather, Forecast, Thermometer:**
 - *Set your location: allows the user to set its location in the app.*
 - *Parameters: [location]*

With this dataset, we were able to add integrations between those entities and then add them as Alice and Bob's preferred integrations, so we could have a real case scenario.

- **Simulation of the system:** Once the dataset was defined, we simulated the interaction of users with the chatbot. This way, we could perform the operations the agent should call to see if we could obtain the expected results. The following figure 11.1 reflects a possible conversation between the chatbot and the user that would require access to the knowledge base as well as the computation of the suggestions.

Chatbot: Hi there! I noticed you used <Strava> to <plan a route>. As a suggestion, you could <create a task> with the generated data.

Bob: Not really. What else could I do?

Chatbot: Here is a list of suggested features you could do now: <create an event>.

Bob: OK let's <create an event>.

Chatbot: Would you like to <create an event> with <Google Calendar>?

Bob: Which other apps can I use?

Chatbot: Here is a list of suggested apps you could use to <create an event>: <Simple Calendar>.

Bob: Ok use <Simple Calendar>.

Chatbot: To <create an event> with <Simple Calendar>, I will use the following data:

- <event-name> as <route-name>
- <location> as <start-location>

Bob: Change the event's <location> to the route's <end-location> and set the <event-name> to <"Lunch after hiking">.

Chatbot: To <create an event> with <Simple Calendar>, I am missing the following details:

- <event-datetime>
- <event-description>

Bob: Schedule the event on <February 23rd>, it will take place at <12:30 p.m>.

Figure 11.1: Example conversation between the chatbot and the user

In this case, the knowledge base has the logic to perform the pertinent operations and provide the chatbot with the information the user needs in each case: actions (features), apps, etc.

With this, we were able to perform an evaluation of what was implemented. Exactly what testing was performed and the reasons why as well as the results obtained will be explained next.

11.2 Component testing

Unit testing is a low-level type of testing, consisting in testing individual functions/methods from a class, a module or a component, so they are performed at a granular level. Component testing's granularity, on the other hand, has a higher granularity, as it tests a whole software component, which is a set of classes, modules, frameworks, etc. that are interconnected and work together to achieve a common goal [64].

This testing has been done using the *JUnit* framework, which is a pretty popular framework to perform unit testing in Java. What has been tested are the CRUD operations of all the models, as well as the calls that allow the chatbot to set the user's preferences and provide them with suggestions (ergo everything that has been defined) in order to see that all works as expected. This has been done due to the importance of letting the chatbot to work only with valid and well-structured data, as well as to avoid any type of data corruption.

The tests have been defined for the controllers, which are the classes that handle the HTTP requests. Due to them being the initial point of the application, it has been considered enough to test them, as they are the ones that call the services, and the services are the ones that call the repositories, which are the ones that perform the CRUD operations. An example of how these tests are defined can be seen in figure 11.2 and the results can be seen in figure 11.3. All of them can be found in the project's repository [65], in the *src/test* folder.

```

@ParameterizedTest
@Order(1)
@CsvSource({
    //a test user is created, it should return 201
    "testingID, 201 CREATED",
    //an already existing user is trying to be created, it should return 409
    "quim-motger@gessi.upc.edu, 409 CONFLICT"
})
public void testCreateUser(String username, String expectedStatus){
    User user = new User(username, email:"testmail@mail.com", givenName:"testGivenName", familyName:"testFamilyName");
    ResponseEntity<User> response = restTemplate.postForEntity(baseUrl + "/users", user, responseType:User.class);
    assertEquals(expectedStatus, response.getStatusCode().toString());
}

@Test
@Order(2)
public void testGetAllUsers() throws Exception {
    ResponseEntity<List<User>> response = restTemplate.exchange(
        baseUrl + "/users", HttpMethod.GET, requestEntity:null,new ParameterizedTypeReference<List<User>>() {});
    List<User> users = response.getBody();
    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertNotNull(users);
    assertThat(users.size()).isGreaterThan(other:0);
}

```

Figure 11.2: Component testing
Font: Own elaboration

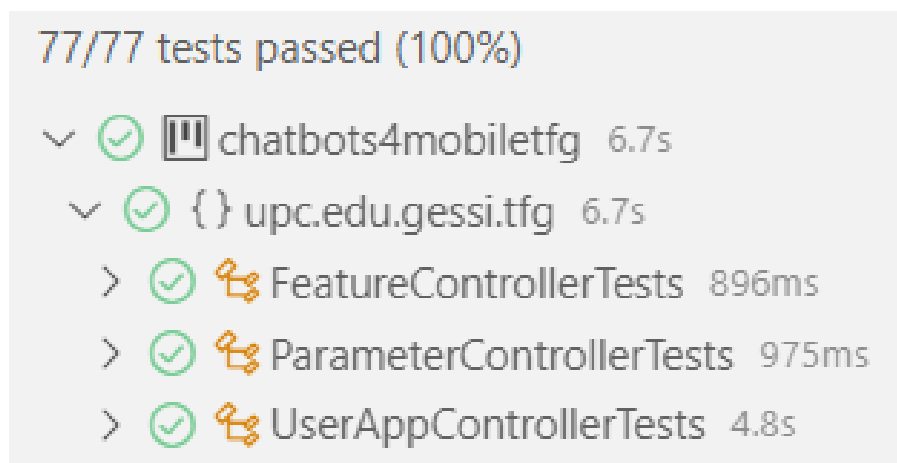


Figure 11.3: Component testing results
Font: Own elaboration

11.3 Integration testing

Integration testing is a type of testing that is based on testing the interaction and the correct integration of two different software systems or components. Its goal is to ensure the correct working of the system as a whole, as well as to detect whatever defects (errors or bugs) that may arise from the interaction of the components [66].

In this project, due to it being part of a much wider system, the knowledge base that has been developed, in the end, has to be integrated with the rest of the system (that is, the chatbot may be able to use it to offer recommendations to the user). Therefore, integration testing has been carried out to ensure that the knowledge base works correctly with the rest of the system. The integration tested is based on the integration between

the knowledge base and the NLU chatbot component, trying to ensure the end-to-end functionality of the system.

This has been done with the help of a chatbot developer (explained in 5.2), who has been the one responsible for integrating the knowledge base with the chatbot and checking that everything works correctly. If there was any problem, the student was there to help and was responsible for solving whatever problem that may have arisen.

An example of how the integration testing has been carried out can be seen in the following figure 11.4.

```

Chatbot: Hi there! I noticed you used <Strava> to <plan a route>. As a suggestion, you could <create a task> with the generated data.
Bob: Not really. What else could I do?
GET .../users/bob@essi.upc.edu/integrations/features/plan_route
Result = { create_event }
Chatbot: Here is a list of suggested features you could do now: <create an event>.
Bob: OK let's <create an event>.
Chatbot: Would you like to <create an event> with <Google Calendar>?
Bob: Which other apps can I use?
GET .../users/bob@essi.upc.edu/integrations/apps/feature/com.strava/com.google.android.calendar
Result = { com.simplmobiletools.calendar }
Chatbot: Here is a list of suggested apps you could use to <create an event>: <Simple Calendar>.
Bob: Ok use <Simple Calendar>.
GET .../parameters/integrations/request          Body: { "sourceApp": "com.strava", "sourceFeature": "plan_route",
                                                "targetApp": "com.google.android.calendar", "targetFeature": "create_event" }

Result = { [event-name, route-name], [location-start-location] }
Chatbot: To <create an event> with <Simple Calendar>, I will use the following data:
- <event-name> as <route-name>
- <location> as <start-location>
Bob: Change the event's <location> to the route's <end-location> and set the <event-name> to <"Lunch after hiking">.
GET .../parameters/integrations/request/custom    Body: { "sourceApp": "com.strava", "sourceFeature": "plan_route",
                                                "targetApp": "com.google.android.calendar", "targetFeature": "create_event" }

Result = { event-datetime, event-description }
Chatbot: To <create an event> with <Simple Calendar>, I am missing the following details:
- <event-datetime>
- <event-description>
Bob: Schedule the event on <February 23rd>, it will take place at <12:30 p.m>.

```

Figure 11.4: Integration testing example

This example follows the previously shown example of a chatbot conversation (see figure 11.1). In this case, we can see how the chatbot interprets the user's message and with the input received and considering the user profile, it queries the knowledge base to obtain the recommendations that the user is asking for. Those calls to the knowledge base are the inductive requests that have been defined in the previous section (see subsection 10.4.3). So, the scenario goes as follows:

1. The chatbot gets triggered by Bob's action of planning a route using Strava, and asks him if he wants to create a task.
2. Bob says no, and asks for other suggestions.
3. The chatbot interprets the message and queries the knowledge base, requesting feature integrations considering *Plan a route* as the source feature and considering Bob's preferences (first GET call). Then, the chatbot receives the recommendations from the knowledge base and shows them to Bob.
4. Bob chooses to create an event.
5. The chatbot asks if he wants to use Google Calendar to create an event.
6. Bob asks for other suggestions.

7. The chatbot queries the knowledge base again, requesting app integrations considering *Plan a route* as the source feature, *Create an event* as the target feature and considering Bob's preferences (second GET call). Then, the chatbot receives the recommendations from the knowledge base and shows them to Bob.
8. Bob chooses to create an event using Simple Calendar.
9. The chatbot queries the knowledge base again, requesting all the parameters that are integrated between these two features from these two apps (third GET call). Then, the chatbot receives the results from the knowledge base and shows them to Bob.
10. Bob, once those parameters are shown, customises them to his liking (if needed).
11. The chatbot queries the knowledge base again, requesting the parameters that are lacking of a value for the integration between these two features from these two apps (fourth GET call). Then, the chatbot receives the results from the knowledge base and shows them to Bob.
12. Bob specifies the desired values, and the chatbot creates the event in Simple Calendar.

Chapter 12

Sustainability

After completing the EDINSOST2-ODS survey, I have been able to think over and realise the importance sustainability has in a project.

Firstly, related to the economic dimension, I have been working and getting better at it during my specialization. I am aware of the importance of analysing the possible costs of a project, as well as studying the market and its viability. I am aware of the importance of making a deep analysis based on the data when evaluating a project. The economic impact is crucial, including a deep study of what the costs and the potential returns are key aspects when evaluating the viability of a project.

Moreover, the usage of open-source technologies within the project can have an impact when it comes to the estimation of the costs, as well as the potential for community contributions (although this estimation is not as easy to compute as the previous one). Long story short, focusing on these aspects can help us when deciding whether the project is feasible or not.

Secondly, related to the environmental dimension, I have realised the underestimated impact hardware and software have, particularly computers containing toxic metals. That is why recycling these tools at the end of their lifespan is a must. Although it does not affect my thesis work, this training in sustainability has helped me to cover aspects of this type. Regarding my thesis work, I want to highlight the low environmental impact it has compared to other disciplines which are really close to this one, like Language Models (LLMs) in general. By focusing on taking into account the environmental impact throughout my research, including resource utilization or energy efficiency, I intend to contribute to ending up having a much more environmentally conscious approach in the field.

Finally, related to the social dimension, I have realised the importance of the impact a project can have on society. There are several dilemmas that can arise when developing a project that involves the usage of data (e.g. privacy, security, etc.). Another aspect I have found really key is the usage of open-source technologies mentioned earlier; contributing to the open-source community is a great way to give back to society. Lastly, regarding my thesis work, I want to emphasise the importance of the impact it can have on the end users: the development of a system which can offer a fully personalised experience to the user makes it a great tool for society.

12.1 Initial milestone

12.1.1 Economic dimension

Regarding PPP, have you estimated the cost of the realization of the project?

In the *Budget* chapter, an estimation of the project's costs has been made, taking into account all the factors that surely intervene as well as possible factors that may appear during the development. Once all the costs have been estimated, the project's viability has been analysed, taking into account the possible deviations that may occur during the development. After this analysis, it has been concluded that the project is financially viable, due to the fact that the estimated total cost (deviations included) is not extremely high, and the value of the project is high enough to justify the investment.

12.1.2 Environmental dimension

Regarding PPP, have you estimated the environmental impact that the realization of the project will have?

Due to the nature of the project (software development), the environmental impact will be minimal. Nevertheless, the impact related to the use of the hardware is considered, such as the electricity consumption, as well as the impact that might be caused by the use of a virtual machine deployed in a server hosted by the university.

12.1.3 Social dimension

Regarding PPP, what do you think you will get from the realization of this project?

I consider that this project will help me to improve my skills, not only in terms of programming but also in terms of organization, due to the fact that I will be the one in charge of the project (along with the director) and I will have to manage precisely my time. It will also allow me to apply all the concepts and knowledge I have learned so far in the degree.

Regarding lifespan, how will the project improve sociality's quality of life?

It will mainly save users some time, by offering actions they may want to perform after a particular action, saving them that process.

Regarding lifespan, is there a real need for the project?

From GESSI's point of view, there is a real need for this product, since, as it has been commented in the contextualization, this tool would allow the Chatbots4Mobile project to expand its features, and would offer a new service to the users, which will be very useful.

12.2 Final milestone

12.2.1 Economic dimension

Regarding PPP, have you quantified the cost (human and material resources) of carrying out the project? What decisions have you taken to reduce the cost? Have you quantified these savings?

Both the human and material resources have been quantified in the project (check chapter *14 Final planning and budget*). Finally, the human resources have been reduced to a single person, only myself acting as the developer of the project. In terms of material resources, the only costs considered have been the ones related to the hardware and software as well as the electricity consumption and the Internet connection (general resources). This cost is the one that was estimated initially, and taking into account that finally there has been no payment for the human resources, the cost of the project has been reduced drastically, as can be seen in table 12.1.

Cost	Value
Human resources	4.422,00 €
Material resources	156,06 €
General resources	355,48 €
Total cost	4.933,54 €

Table 12.1: Final cost of the project
Font: Own elaboration

We can conclude that there has been a huge saving in the cost of the project, compared to what was initially estimated. The total saving has been 8.487,32 €, which is a 63,25% of the initial cost.

Regarding PPP, has the planned cost been adjusted to the final cost? Have you justified the differences (lessons learned)?

If we compare what was initially estimated with the final cost, we can see that there is a huge difference between them. The initial cost was 13.420,86€, and the final cost has been 511,54€. This difference is due to the fact that the human resources, which were the ones that inflated the initial cost, have been drastically reduced, due to me being the only one developing the project. There have been no unforeseen events either, so the amount of money initially estimated has been also removed.

Regarding lifespan, what is the estimated lifetime cost of the project? Could this cost be reduced to make it more viable?

The cost across the lifetime of the project may be higher than the current one. I will not be the one maintaining the project, so the necessity of hiring personnel to do it arises. These people will also need a computer to work with, so the cost of that has to be considered as well. If we consider that the material resources costs are the same as the ones considered in the development, adding the cost of the personnel, the cost of the project would be the one that can be seen in table 12.2. Only a developer has been considered.

Cost	Value
Material resources	156,06 €
General resources	355,48 €
Developer	27.456 €
Total cost	27.967.54 €/year

Table 12.2: Maintenance cost of the project
Font: Own elaboration

It is important to mention that the maintenance cost of the project is computed annually, considering that there will be a developer working full time maintaining/improving the project.

Regarding lifespan, has the cost of adjustments/upgrades/repairs during the lifetime of the project been taken into account?

Yes, the cost of adjustments/upgrades/repairs has been taken into account. The cost of the developer previously detailed is the one that has been considered for this. He/She will be the one responsible for the maintenance of the project-

Regarding risks, could scenarios occur that would undermine the viability of the project?

There can happen two things: the first one is that the project is not used, and the second one is that the project goes viral and is used by a lot of people. In the first case, the cost of the project will not be affected; with the existing resources everything will be covered and depending on the number of users, the viability of the system can be considered. In the second case, the cost of the project will increase, as the amount of resources needed will increase as well. The cost of the project will be higher, but the income will be higher as well, so the viability of the project will not be affected.

12.2.2 Environmental dimension

Regarding PPP, have you quantified the environmental impact of your project? What measures have you taken to reduce it? Have you quantified this reduction?

In terms of environmental impact, the development of this project is not very significant. This thesis is based on the development and extension of software that is already in use, so the resources that have been used are the ones that are already used for it (for example, the server that hosts the knowledge base and the electricity used to power it), and there has been no real addition of resources that have made this impact increase. The resources used for this development have been one computer (there have been two computers used, but not at the time so technically it is one computer), and the electricity used to power it (more information about this can be checked in sections 5.2 and 6.1.3). The computers are both laptops, so they are not very powerful computers, and they are not on all the time. Another resource that is important to mention is the virtual machine deployed in ESSI's server, which is used to host the knowledge base. This virtual machine is also not very powerful in terms of computational power, so its impact is also low. The total impact of this project resides on the laptop itself as well as the software that has been

used to develop it. There has been no reduction so there is not a quantification of it available.

Regarding PPP, if you had to redo the project, would you be able to reduce the resources used?

I do not believe that there is a way to reduce the resources used for this project, as the resources used are the mandatory ones to develop it, there is nothing extra that has been used. The resources used are not very powerful, and the energy consumption derived from the development of this project has been restricted to the original plan, so the level of impact is really low especially when compared to projects from similar areas, such as LLMs or other projects that require a lot of computational power.

Regarding lifespan, what resources do you estimate that the project will consume during its useful life? What would be the environmental impact of these resources?

The resources that will be used across the existence of this project are practically the same ones used for the initial development of it. There will be the necessity of using a laptop/ computer to add/fix/deploy new features, as well as the electricity used to power both the laptop in use and the server (which will be also a resource used) that hosts the knowledge base. It is important to note that the resources, when they reach the end of their lifespan, will be replaced by new ones, in order to avoid the impact of the project to increase.

So, in terms of the environmental impact of these resources, laptop usage will be reduced, ergo the impact of it will be reduced. The impact of the server, however, will remain the same as it is for the development, but it will not increase as the server will not be replaced by a more powerful one.

Regarding lifespan, will the project allow to reduce the usage of other resources? Globally, will the usage of the project improve or worsen the ecological footprint?

Due to the project being an extension of an already existing software, the ecological footprint will not be affected by it. The project will not allow reducing the usage of other resources, as it is not a project that is meant to be used by the general public, but by a specific group of people that are already using the software that this project is based on.

Regarding risks, could there be scenarios that would increase the ecological footprint of the project?

I do believe that the project could increase its ecological footprint, but the scenario that may provoke that is very unlikely. Maybe there could be an impact if the chatbot started to be used by a large number of people, meaning that a much more powerful server would be needed to host the knowledge base, but the amount of users that should appear in order to make this happen is very high, so I do consider that it can be discarded.

12.2.3 Social dimension

Regarding PPP, has the realisation of this project involved significant personal, professional or ethical reflections of the people involved?

The realisation of this project has not involved significant nor personal or ethical reflections of the people involved. A professional reflection, however, has been made by myself, as it has been the first time that I have worked with a project contained inside a bigger one, and also the usage of several new technologies has been a challenge that has been overcome.

Regarding lifespan, who will benefit from the use of the project? Are there any groups that may be disadvantaged by the project? To what extent?

The potential users of the chatbot will, after system and acceptance validation benefited from this new knowledge system, as they will be receiving suggestions from the chatbot that they already used, offering them new options that they may find useful. This new knowledge system adds different functionalities to the chatbot, and that may call the attention of the general public, adding new users to the system.

To the best of our knowledge, any group may feel disadvantaged or rejected by the project, as the knowledge system's information is not biased in any way, and it is not intended to be used in a way that may harm any group of people.

Regarding lifespan, to what extent does the project solve the problem initially posed?

The project solves the problem initially posed in a way that the chatbot is now able to offer suggestions to the users, based on the information that it has received from the user's context (previous actions, new actions, etc.).

Regarding risks, could there be scenarios that would make the project detrimental to a particular segment of the population?

There are no scenarios that would make the project detrimental to a particular segment of the population, as the information that the knowledge system receives is not biased in any way.

Regarding risks, could the project create some kind of dependency that would leave users in a position of weakness?

There is the possibility that the system may not work in terms of adding/updating/deleting the information from a user, making the user's suggestion static in the sense that they will be always the same until the system is fixed. But I will not consider this as a dependency, as the user will always receive suggestions using the existing data, so the service will not be interrupted.

Chapter 13

Legislation

In this chapter, both the laws and the regulations that the application complies with are mentioned. Firstly, regarding the libraries and frameworks that are used in the development of the project, all of them are open source and have different licenses, such as MIT [67], Apache 2.0 [68] and EPL [69]. All of them share some characteristics, such as the fact that they are permissive licenses, which means that they allow the use of the software in any context, including commercial and non-commercial. They also allow the modification of the software and the distribution of the modified versions, as long as the license is included in the modified version. In our case, the license under which our code is published is the Apache 2.0 one.

Secondly, regarding the database, the service uses some data which was taken from other projects related to Chatbots4Mobile, such as applications and features. This data has been taken from public sources or has been created by the department, so there is no problem related to that either, since that in this data collection process, the involved researchers and developers aimed at guaranteeing compliance with respect to the regulations from the original data sources, making sure that its distribution and usage for the purposes covered in this thesis were authorized. User data does not compromise the General Data Protection Regulation (GDPR) [70] because, in the context of the thesis, everything is mocked, so there is no real user data being used.

Lastly, this project is not being released into any platform as a product in particular, it is serving as a service to another project. This bigger project is still in development, so that is why it is not being released as a product.

Chapter 14

Final planning and budget

Once the project has been developed, it is necessary to go back to the original time planning and budget to see if everything has been done as how it was initially defined, or if any changes have been made. In this chapter, both the final planning and the final budget will be detailed, and compared with the initial ones in case of any changes.

14.1 Estimations

Regarding the estimations, the initial estimations were detailed in section 5.3 "*Estimations and Gantt*". There, the estimations for the different tasks were detailed, as well as the Gantt diagram, which showed the time distribution that would be tried to be followed to perform each of the tasks.

Although the initial estimations were made without having a clear idea of the complexity of each task, the final estimations have been quite similar to the initial ones, as can be seen in table 14.1. There have been some tasks that have taken a little longer than expected, but others have taken less time than expected, so the final estimations have been quite similar to the initial ones. The final Gantt diagram can be seen in figures 14.1 and 14.2, where the final distribution of the tasks across the different weeks can be seen.

Task ID	Task Name	Estimated Effort (h)	Final Effort (h)	Difference (h)	Dependencies
PM	Project management	100	100	0	
PM1	Context and scope	20	20	0	
PM2	Temporal planning	10	10	0	PM1
PM3	Budget and sustainability	10	10	0	PM2
PM4	Initial documentation	10	10	0	PM3
PM5	Final documentation	20	20	0	PM4
PM6	Oral defence preparation	20	20	0	PM5
PM7	<i>Sprint</i> Planning	5	5	0	
PM8	<i>Sprint</i> Review	5	5	0	
PM9	<i>Sprint</i> Retrospective	5	5	0	
DEV0	Project set up	10	25	+15	
DEV0.1	Learn the basic concepts	5	15	+10	
DEV0.2	Set up the development environment	5	10	+5	
DEV1	Data schema extension	80	77.5	-2.5	DEV0
DEV1.1	Identify requirements	10	15	+5	
DEV1.2	Define entities and relationships	30	25	-5	DEV1.1
DEV1.3	Define attributes	15	12.5	-2.5	DEV1.1
DEV1.4	Create the data schema	25	25	0	DEV1.1
DEV2	Domain constraints	60	60	0	
DEV2.1	Analyse potential constraints	20	20	0	
DEV2.2	Formalise the constraints for each feature	30	30	0	DEV2.1
DEV2.3	Implement the constraints	10	10	0	DEV2.2
DEV3	Data schema integration	130	132.5	+2.5	DEV2
DEV3.1	Analyse the existing KB structure	20	22.5	+2.5	
DEV3.2	Identify required changes	20	20	0	DEV3.1
DEV3.3	Implement changes to KB	35	35	0	DEV3.2
DEV3.4	Create business logic for CRUD operations	35	35	0	DEV3.3
DEV3.5	Test the integration	20	20	0	DEV3.4
DEV4	Knowledge Base extension	105	108	+3	DEV3
DEV4.1	Identify the types of inductive requests to be supported	10	15	+5	
DEV4.2	Design a method for customisation and enactment of potential integrations	35	38	+3	DEV4.1
DEV4.3	Implement the method for supporting inductive requests	35	30	-5	DEV4.2
DEV4.4	Test the extension	25	25	0	DEV4.3
DEV5	Knowledge Base evaluation	75	75		DEV4
DEV5.1	CRUD operations testing	15	15	0	
DEV5.2	Integrate the KB extension into the agent	15	15	0	
DEV5.3	Test the agent's ability to query and modify the KB	35	35	0	
DEV5.4	Evaluate the results and make improvements as needed	10	10	0	DEV5.1, DEV5.2

Table 14.1: Summary table of the tasks with their initial and final time estimations.
Font: Own elaboration

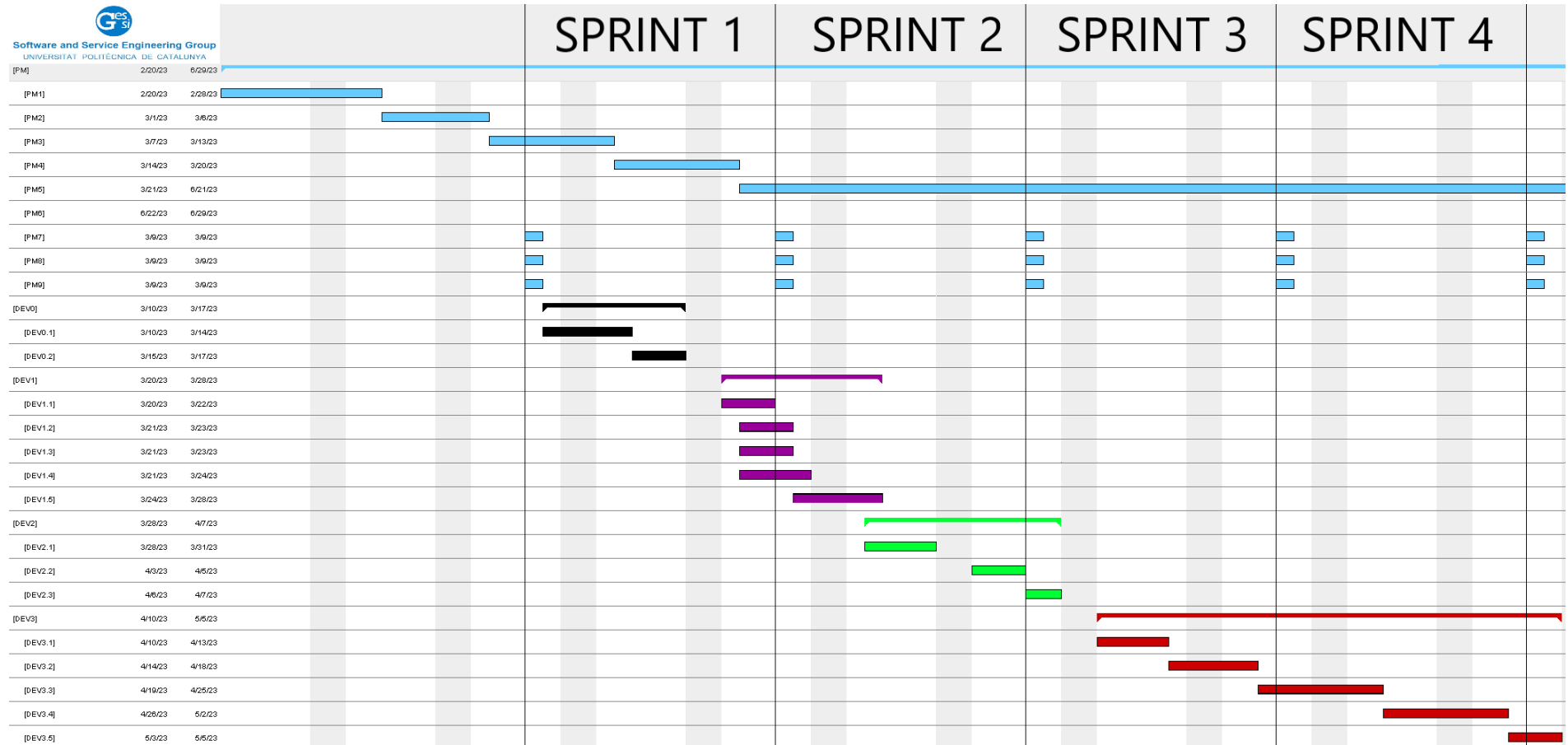


Figure 14.1: Gantt's diagram

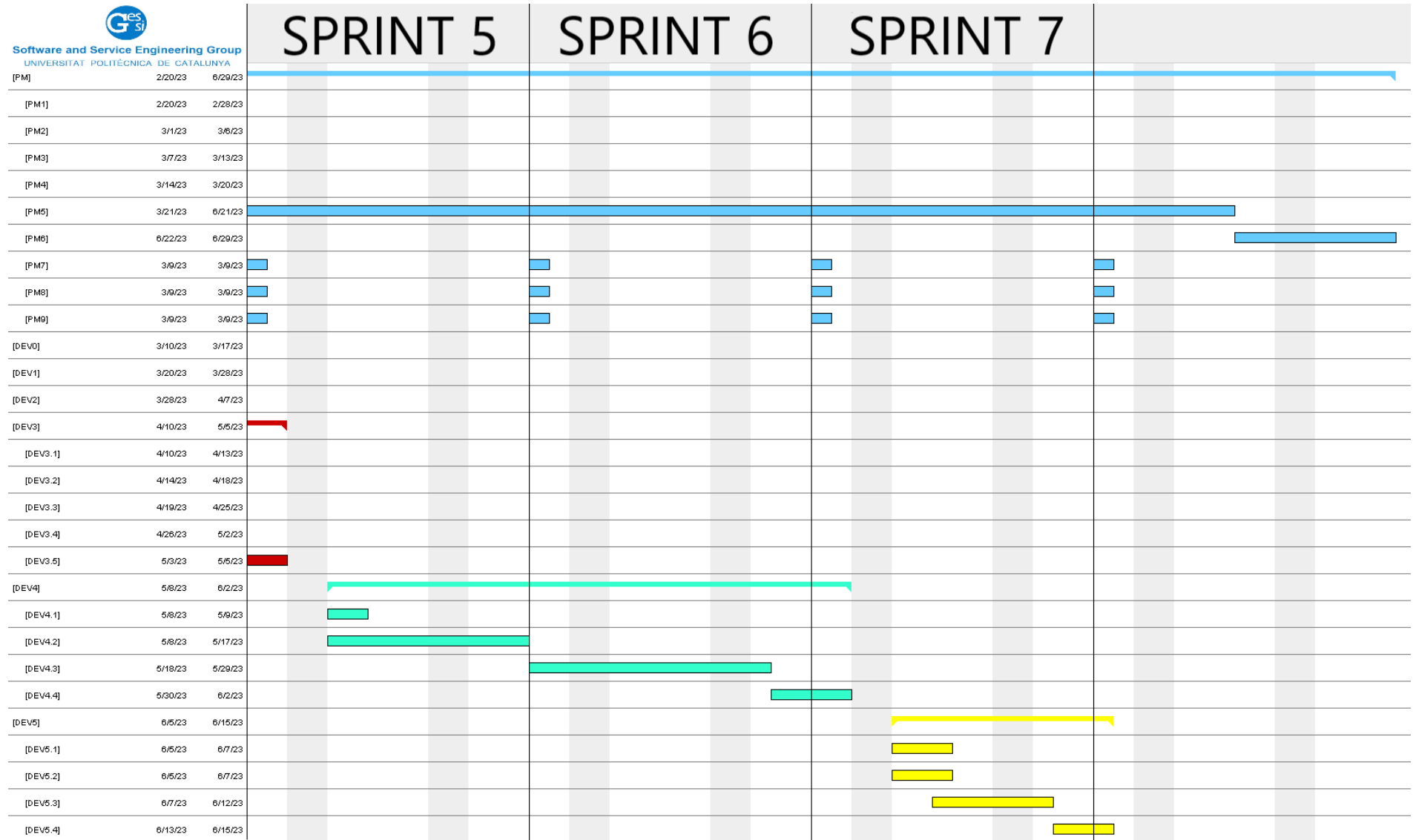


Figure 14.2: Gantt's diagram

14.2 Budget

Regarding the budget, the initial budget was estimated and detailed in section 6 "*Budget*". It ended up being a total of 13.420,86 €, after considering all the resources and expenses that were going to be necessary for the development of the project.

The final budget, however, has been drastically reduced, due to the fact that the project has been developed by a single person, and not by a team of people, as it was initially planned. Considering that the person who has developed the project has not been paid, the final budget has been simplified to only the material and general resources. So, a comparative table between the initial and final budget can be seen in table 6.6.

Cost	Initial	Final
Human resources	10.546,45 €	4.422,00 €
Material resources	156,06 €	156.06 €
General resources	355,48 €	355,48 €
Contingencies	1577,47 €	237.60 €
Unforeseen events	785,40 €	0.00 €
Total cost	13.420,86 €	5.408,74 €

Table 14.2: Initial and final budget
Font: Own elaboration

Chapter 15

Conclusions

15.1 Achievement of the goals

In this section, the goals that were set at the beginning of the project (check section 3.1) as well as the level of accomplishment of those goals are shown. We can say that all of them have been achieved successfully, and the level of accomplishment is high, as all the goals have been achieved completely.

G1: *Design the data schema extension to support the formalisation of: (a) Mobile app features, (b) Input and output parameters, (c) App integrations, (d) Feature integrations, (e) Parameter integrations, and (f) User preferred integrations.*

The data schema extension was thoroughly studied and designed, in terms of how to make it efficient (use only the required to not have to make a huge extension) and easy to apply to the existing knowledge base. This design included the definition of the different entities that were required, as well as the formalisation of the different features, and the different integrations. The formalisation of all of those entities, attributes, and relations was done following the Schema.org standard (see 7.1.2).

G2: *Define and formalise the domain constraints of these feature integrations.*

The domain constraints were defined and formalised and, due to the knowledge base being schema-less, the constraints were applied in the business logic of the knowledge base. This was done to ensure the uniqueness of the different entities, data completeness, and data integrity.

G3: *Integrate the data schema into the existing KB and extend the KB with the required business logic to populate the KB with CKS data (CRUD operations).*

For each of the entities that were defined in the extended data schema (including the ones that were already defined in the knowledge base), the CRUD operations were implemented. This was done to be able to create, read, update, and delete the different entities. Apart from that, the business logic was also implemented to ensure the domain constraints that were defined in G2. To ensure the correct functioning of the operations, unit tests were created and executed (see 11.2).

G4: *Extend the KB with advanced inductive requests based on the customisation and enactment of potential integrations for a given user.*

The advanced inductive requests were implemented and tested using two different approaches (see 11.2 and 11.3). The advanced inductive requests consisted of the following:

- Request feature integrations from source features and previous user preferences. 8.17
- Request app integrations from a selected target feature and previous user preferences. 8.18
- Request source-target parameter integrations for a selected app. 8.19
- Request custom parameters for a selected app. 8.20

These advanced inductive requests allow the chatbot to be able to perform personalised suggestions to the user and offer them the best possible integration based on their preferences and context.

G5: *Evaluate the KB extension through the integration of a conversational agent querying and modifying a specific KB instance.*

The knowledge base extension was evaluated by integrating it with the chatbot. This integration was done using REST, and the chatbot was able to perform the different operations that were defined in G4. The integration was tested and the results were satisfactory (see 11.3).

15.2 Technical competencies

Hereunder, the level of accomplishment of the technical competencies that were set at the beginning of the project is shown:

CES1.1: To develop, maintain and evaluate complex and/or critical software systems and services. [In-depth]

This competence has been achieved and developed during the whole process of creating the extension of the knowledge base. All the related to how the requests are processed, how the data should be stored and how the data should be retrieved has been. Not only the development of new features/extensions has been made, but also the required to maintain and have everything tested and working.

CES1.2: To solve integration problems in the function of the strategies, standards and available technologies [A little bit]

This competence has been achieved in the sense that the integration of the new features has been made in a way that it is easy to integrate with the rest of the system. The communication between the chatbot (the one which makes the requests) and the knowledge base has been made using REST. Apart from that, when defining which schema.org properties should be used, the ones that were already used in the knowledge base were taken into account and have been tried to avoid the creation of new ones, reusing the ones that were already defined. This has been followed, however, the creation of new ones was required in some cases.

CES1.4: To develop, maintain and evaluate distributed services and applications with network support. [Enough]

This competence has been achieved due to the fact that the knowledge base is a distributed service, meaning that the chatbot and the knowledge base are not on the same server, allowing the knowledge base to be used by other services. Also, the maintenance of the initial knowledge base has been made, and the evaluation of the new features has been made, with the goal to have in the knowledge base the most recent information possible.

CES1.5: To specify, design, implement and evaluate databases. [In-depth]

In order to achieve this competence in depth, a structured process has been followed. First, the analysis of the initial knowledge base was made. Then, the specification of the new features and goals was made, and once this was done, the design of the extended structure was made. Once the design phase was over, the implementation came and finally, the evaluation of the new features was added. This process has been followed for each of the new features that were added to the knowledge base.

Other aspects such as how to retrieve and compute the desired information from the knowledge base have been also taken into account, and have been tried to be done in the least amount of requests possible, to have a better performance.

CES1.6: To administrate databases (CIS4.3). [A little bit]

An administration of the database has been made, however, it has not been done in a professional way, but in a way that it is possible to have the knowledge base running and working. The administration of the database has been made in the sense that the data has been stored in a way that it is possible to retrieve it, and the data has been stored in a way that it is possible to be used by other services, making it understandable and easy to use for both machines and humans.

CES1.7: To control the quality and design tests in the software production [Enough]

There has been some testing defined which allows testing the new features that have been added to the knowledge base. Component testing has been made, which allows us to control that the request is being received and all the flow that was defined is being followed. Also, the integration testing has been made, which allows us to test that the new features are being integrated nicely by the chatbot. With this testing defined, we have been able to ensure that the new knowledge base is working as expected.

CES2.1: To define and manage the requirements of a software system. [Enough]

This competence has been achieved by (1) identifying, (2) defining and (3) managing both the functional and the non-functional requirements that the new knowledge base must have. The identification of the stakeholders was also made for each case.

CES3.2: To design and manage a data warehouse. [A little bit]

In order to achieve this competence, the requirements of the new features that were added to the knowledge base were defined. The structure of the nodes, the properties these nodes should have, the relations between the nodes, etc. were defined, using the schema.org vocabulary (when possible, if there was no other option, a new schema.org vocabulary was created). Also, the inductive requests were defined in a way that took the

most profit from the knowledge base's structure, allowing to have a much clear request defined and a much better performance.

15.3 Future work

This project has served as an extension of what was currently developed in the Chatbots4Mobile project and there is still some work to be done. The thesis scope was limited to the user preferences and the context understood as the *set of apps installed in the user's device + the set of features used*. Currently, the responsibility of the decision-making of these preferences is 100% on the chatbot but, over time, a hybrid approach can be studied so that historical user data (e.g., integrations made throughout their use with a set of applications) can also be used to apply decision criteria on when and how to update these preferences, beyond the feedback collected in user interactions with the chatbot, which is what chatbot should now do.

15.4 Personal conclusions

I would like to start this section by thanking my thesis director Quim, who has been a really key piece when it comes to the resolution of this project. He has been always worried about how I was doing, helped me with the new technologies/difficulties that I have been facing across the development of the knowledge system, and also I would like to highlight his availability; he has been always looking forward to knowing how everything was going and I am thankful for that.

By developing this project, I have gained several pieces of knowledge that I may put to use maybe not in the immediate future but in the following years of my career. Having to deal with a non-relational database, defining everything following schemas, how to handle this type of data, learning SPARQL to be able to obtain the data I was creating, etc. are technologies and actions that will be done again in the future certainly and I am really happy to have gained this knowledge by making something useful for someone (GESSI).

Last but not least, I am proud of how I was able to overcome whatever difficulty I was facing. I had several problems with making the knowledge base initially work, I had zero experience with most of the technologies I had to use, and I had to learn how to use them by myself. I am really happy with the result I have obtained, and I am really happy to have learned that there is no problem that cannot be solved and that if you put in enough effort, you will be able to overcome it.

Appendix A

Swagger API documentation

The screenshot displays the Swagger UI for the 'Extended Knowledge Base API'. At the top, there's a header with the Swagger logo, a search bar containing 'api-docs', and an 'Explore' button. Below the header, the API title 'Extended Knowledge Base API' is shown with a link to the 'api-docs'. A 'Servers' dropdown menu is set to 'http://localhost:8080 - Generated server url'. The main content area lists several API groups, each with a set of endpoints and their corresponding HTTP methods:

- Users CRUD** (CRUD operations for users):
 - GET /users/{id} Get a user by id
 - PUT /users/{id} Update a user
 - DELETE /users/{id} Delete a user
 - GET /users Get all users
 - POST /users Create a user
- Feature integrations CRUD** (CRUD operations for feature integrations):
 - GET /features/integrations/{id} Get a feature integration by id
 - PUT /features/integrations/{id} Update an existing feature integration
 - DELETE /features/integrations/{id} Delete an existing feature integration
 - GET /features/integrations Get all feature integrations
 - POST /features/integrations Create a new feature integration
- User preferences** (Operations for managing user preferences):
 - POST /users/{id}/integrations/parameters Add a preferred parameter integrations for a particular user
 - DELETE /users/{id}/integrations/parameters Delete a preferred parameter integrations for a particular user
 - POST /users/{id}/integrations/features Add a preferred feature integrations for a particular user
 - DELETE /users/{id}/integrations/features Delete a preferred feature integrations for a particular user
 - POST /users/{id}/integrations/apps Add a preferred app integrations for a particular user
 - DELETE /users/{id}/integrations/apps Delete a preferred app integrations for a particular user
- Apps CRUD** (CRUD operations for mobile applications):
 - GET /apps/{id} Get an app by id
 - PUT /apps/{id} Update an app
 - DELETE /apps/{id} Delete an app
 - GET /apps Get all apps
 - POST /apps Create an app
- App integrations CRUD** (CRUD operations for app integrations):
 - GET /apps/integrations/{id} Get an app integration by id
 - PUT /apps/integrations/{id} Update an app integration
 - DELETE /apps/integrations/{id} Delete an app integration
 - GET /apps/integrations Get all app integrations
 - POST /apps/integrations Create an app integration
- Parameters Integrations CRUD** (CRUD operations for parameters integrations):
 - GET /parameters/integrations/{id} Get a parameter integration by id

PUT	/parameters/integrations/{id}	Update a parameter integration	⌵
DELETE	/parameters/integrations/{id}	Delete a parameter integration	⌵
GET	/parameters/integrations	Get all parameters integrations	⌵
POST	/parameters/integrations	Create a parameter integration	⌵
Features CRUD <small>CRUD operations for features</small>			⌵
GET	/features/{id}	Get a feature by id	⌵
PUT	/features/{id}	Update an existing feature	⌵
DELETE	/features/{id}	Delete an existing feature	⌵
GET	/features	Get all features	⌵
POST	/features	Create a new feature	⌵
Parameters CRUD <small>CRUD operations for parameters</small>			⌵
GET	/parameters/{id}	Get a parameter by id	⌵
PUT	/parameters/{id}	Update a parameter	⌵
DELETE	/parameters/{id}	Delete a parameter	⌵
GET	/parameters	Get all parameters	⌵
POST	/parameters	Create a parameter	⌵
User stories <small>User stories operations</small>			⌵
GET	/users/{id}/integrations/features/{sourceFeature}	USER STORY #1: Request feature integrations from source features and previous user preferences	⌵
GET	/users/{id}/integrations/apps/feature/{sourceFeature}/{targetFeature}	USER STORY #2: Request app integrations from selected target feature and previous user preferences	⌵
GET	/parameters/integrations/request	USER STORY #3: Request source-target parameter integrations for selected app	⌵
GET	/parameters/integrations/request/custom	USER STORY #4: Request custom parameters for selected app	⌵
Schemas			⌵
<div> <div>User ⌵</div> <div> <div>identifier*</div> <div>email*</div> <div>givenName*</div> <div>familyName*</div> <div>apps</div> <div>preferredFeatureIntegrations</div> <div>preferredParameterIntegrations</div> <div>preferredApps</div> </div> <div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> </div> </div>			
<div> <div>Parameter ⌵</div> <div> <div>identifier*</div> <div>name*</div> <div>type*</div> </div> <div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> </div> </div>			
<div> <div>ParameterIntegration ⌵</div> <div> <div>identifier</div> <div>name</div> <div>sourceParameter*</div> <div>targetParameter*</div> </div> <div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> </div> </div>			
<div> <div>Feature ⌵</div> <div> <div>identifier*</div> <div>name*</div> <div>parameters</div> </div> <div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> </div> </div>			
<div> <div>FeatureIntegration ⌵</div> <div> <div>identifier</div> <div>name</div> <div>sourceFeature*</div> <div>targetFeature*</div> </div> <div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> <div>⌵ [...]</div> </div> </div>			

```
}  
}
```

```
App {  
  name* > [...]  
  identifier* > [...]  
  description* > [...]  
  summary* > [...]  
  releaseNotes* > [...]  
  applicationCategory* > [...]  
  datePublished* > [...]  
  dateModified* > [...]  
  softwareVersion* > [...]  
  features* > [...]  
}
```

```
AppIntegration {  
  identifier  
  name > [...]  
  sourceApp* > [...]  
  targetApp* > [...]  
}
```

```
RequestParamterIntegration {  
  sourceApp* > [...]  
  sourceFeature* > [...]  
  targetApp* > [...]  
  targetFeature* > [...]  
}
```

References

- [1] Hayet Brabra Sara Bouguelia et al. *Context Knowledge-aware Recognition of Composite Intents in Task-oriented Human-Bot Conversations*. URL: https://www.researchgate.net/publication/361388749_Context_Knowledge-aware_Recognition_of_Composite_Intents_in_Task-oriented_Human-Bot_Conversations (visited on 02/28/2023).
- [2] Eric Griffing. *What is a Conversational Agent?* URL: <https://www.dashbot.io/blog/conversational-agent> (visited on 02/24/2023).
- [3] Jordi Marco Quim Motger Xavier Franch. *Software-Based Dialogue Systems: Survey, Taxonomy, and Challenges*. URL: <https://dl.acm.org/doi/full/10.1145/3527450> (visited on 06/11/2023).
- [4] Software and Service Engineering Group. *Chatbots4Mobile*. URL: <https://gessi.upc.edu/en/projects/chatbots4mobile> (visited on 02/24/2023).
- [5] Jordi Marco Quim Motger Xavier Franch. *Integrating Adaptive Mechanisms into Mobile Applications Exploiting User Feedback*. URL: https://link.springer.com/chapter/10.1007/978-3-030-75018-3_23 (visited on 02/28/2023).
- [6] OsmAnd. *OsmAnd*. URL: <https://osmand.net/> (visited on 06/19/2023).
- [7] Etar-Group. *Etar-Calendar*. URL: <https://github.com/Etar-Group/Etar-Calendar> (visited on 06/19/2023).
- [8] Sara Bouguelia et al. "Context Knowledge-Aware Recognition of Composite Intents in Task-Oriented Human-Bot Conversations". In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. Vol. 13295 LNCS. 2022, pp. 237–252. DOI: 10.1007/978-3-031-07472-1_14.
- [9] Google. *Google Assistant, your own personal Google*. URL: <https://assistant.google.com/> (visited on 02/26/2023).
- [10] Analytics Vidhya. *Ok Google! Speech to Text in Python with Deep Learning in 2 Minutes*. URL: <https://www.analyticsvidhya.com/blog/2021/09/ok-google-speech-to-text-in-python-with-deep-learning-in-2-minutes/> (visited on 06/11/2023).
- [11] Apple. *Siri - Apple*. URL: <https://www.apple.com/siri/> (visited on 02/26/2023).
- [12] Macworld. *How to stop Siri listening to you*. URL: <https://www.macworld.com/article/673641/how-to-stop-siri-listening-to-you.html> (visited on 06/11/2023).
- [13] others J.Chen X.Peng. *MashReDroid: enabling end-user creation of Android mashups based on record and replay*. URL: <https://link.springer.com/article/10.1007/s11432-019-2646-2> (visited on 06/12/2023).
- [14] Volere. *Volere Requirements Specification Template*. URL: <https://www.volere.org/templates/volere-requirements-specification-template/> (visited on 02/26/2023).

- [15] Techopedia. *Software Bug*. URL: <https://www.techopedia.com/definition/24864/software-bug-> (visited on 02/24/2023).
- [16] Atlassian. *What is Agile?* URL: <https://www.atlassian.com/agile> (visited on 02/24/2023).
- [17] Slack. *Slack is your digital HQ — Slack*. URL: <https://slack.com/intl/en-gb/> (visited on 02/27/2023).
- [18] Git. *Git*. URL: <https://git-scm.com/> (visited on 02/24/2023).
- [19] Tencent. *Flow*. URL: <https://cloud.tencent.com/developer/article/1824154> (visited on 06/11/2023).
- [20] GitHub. *GitHub*. URL: <https://github.com/> (visited on 02/24/2023).
- [21] Taiga. *Taiga: Your opensource agile project management software*. URL: <https://www.taiga.io/> (visited on 02/24/2023).
- [22] Glassdoor. *Glassdoor Job Search — You deserve a job that loves you back*. URL: <https://www.glassdoor.com/> (visited on 03/07/2023).
- [23] Visual Studio Code. *Visual Studio Code - Code Editing. Redefined*. URL: <https://code.visualstudio.com/> (visited on 03/12/2023).
- [24] GitHub. *Pricing · Plans for every developer - GitHub*. URL: <https://github.com/pricing> (visited on 03/12/2023).
- [25] Ontotext. *GraphDB Download*. URL: <https://www.ontotext.com/products/graphdb/download/> (visited on 03/12/2023).
- [26] PC Expansion. *Hp Omen 15 Ax001ns*. URL: <https://www.pcexpansion.es/hp-omen-15-ax001ns.php> (visited on 03/09/2023).
- [27] Dell. *Portátil Latitude 3520*. URL: <https://www.dell.com/es-es/shop/port%C3%A1tiles-dell/port%C3%A1til-latitude-3520/spd/latitude-15-3520-laptop> (visited on 03/07/2023).
- [28] Logitech. *RATÓN INALÁMBRICO M185*. URL: <https://www.logitech.com/es-es/products/mice/m185-wireless-mouse.910-002235.html> (visited on 03/07/2023).
- [29] Selectra. *Precio de la luz por horas*. URL: <https://tarifaluzhora.es/> (visited on 03/07/2023).
- [30] Brian Potts. *How to Define your ERP Contingency Budget*. URL: <https://www.thirdstage-consulting.com/how-to-define-your-erp-contingency-budget/#:~:text=Most%20contingency%20budgets%20are%20way,to%20go%20forward%20for%20approval>. (visited on 03/12/2023).
- [31] IBM. *What is a Knowledge Graph?* URL: <https://www.ibm.com/topics/knowledge-graph#:~:text=A%20knowledge%20graph%2C%20also%20known,the%20term%20knowledge%20%E2%80%9Cgraph.%E2%80%9D> (visited on 06/09/2023).
- [32] Thomas Frisendal. *Graph Data Modeling*. URL: <http://graphdatamodeling.com/> (visited on 06/09/2023).
- [33] Plum Flower Software. *An intro to graph databases in healthcare*. URL: <https://plumflowersoftware.com/blog/post/84/An+intro+to+graph+databases+in+healthcare> (visited on 06/11/2023).
- [34] Conjointly. *Deduction and Induction*. URL: <https://conjointly.com/kb/deduction-and-induction/> (visited on 06/12/2023).
- [35] Ontotext. *What is a Knowledge Graph?* URL: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/> (visited on 05/31/2023).
- [36] Tom Gruber. *Definition of Ontology*. URL: <https://tomgruber.org/writing/definition-of-ontology> (visited on 06/09/2023).

- [37] W3C. *OWL*. URL: <https://www.w3.org/OWL/> (visited on 06/09/2023).
- [38] W3C. *RDF - Semantic Web Standards*. URL: <https://www.w3.org/RDF/> (visited on 05/20/2023).
- [39] W3C. *Semantic Web*. URL: <https://www.w3.org/standards/semanticweb/> (visited on 06/09/2023).
- [40] Wikipedia. *Semantics*. URL: <https://en.wikipedia.org/wiki/Semantics> (visited on 06/09/2023).
- [41] Ontotext. *What is the Semantic Web?* URL: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/> (visited on 05/31/2023).
- [42] Carbon LDP. *The Advantages of Resource Description Framework (RDF)*. URL: <https://carbonldp.com/blog/2017/12/18/the-advantages-of-resource-description-framework-rdf/> (visited on 06/09/2023).
- [43] W3C. *Semantic Web*. URL: <https://www.w3.org/standards/semanticweb/> (visited on 05/31/2023).
- [44] Aidan Hogan. *RDF Schema (RDFS) and Semantics*. URL: https://link.springer.com/chapter/10.1007/978-3-030-51580-5_4 (visited on 06/12/2023).
- [45] Libby Miller Dan Brickley. *FOAF Vocabulary Specification*. URL: <http://xmlns.com/foaf/0.1/> (visited on 06/12/2023).
- [46] DublinCore. *DCMI Metadata Terms*. URL: <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/> (visited on 06/12/2023).
- [47] Schema.org. *Schema.org*. URL: <https://schema.org/> (visited on 05/31/2023).
- [48] F-Droid. *Simple Music Player*. URL: <https://f-droid.org/en/packages/com.simplmobiletools.musicplayer/> (visited on 06/12/2023).
- [49] Pratyush-Avi. *MyFit-App-A-fitness-Centric-App*. URL: <https://github.com/Pratyush-Avi/MyFit-App-A-fitness-Centric-App> (visited on 06/12/2023).
- [50] Coursera. *What is a User Story?* URL: <https://www.coursera.org/articles/what-is-user-story> (visited on 05/29/2023).
- [51] ProductPlan. *Acceptance Criteria*. URL: <https://www.productplan.com/glossary/acceptance-criteria/> (visited on 05/29/2023).
- [52] JavaTpoint. *Spring Boot Architecture*. URL: <https://www.javatpoint.com/spring-boot-architecture> (visited on 06/11/2023).
- [53] Spring. *Spring*. URL: <https://spring.io/> (visited on 06/02/2023).
- [54] Strava. *Strava*. URL: <https://www.strava.com/> (visited on 06/13/2023).
- [55] Java. *Java — Oracle*. URL: <https://www.java.com/en/> (visited on 05/20/2023).
- [56] W3C. *SPARQL Query Language for RDF*. URL: <https://www.w3.org/TR/rdf-sparql-query/> (visited on 05/20/2023).
- [57] VMware. *Spring Boot*. URL: <https://spring.io/projects/spring-boot> (visited on 05/20/2023).
- [58] Eclipse Foundation. *Eclipse RDF4J*. URL: <https://rdf4j.org/> (visited on 05/20/2023).
- [59] springdoc-openapi. *OpenAPI 3 Library for spring-boot*. URL: <https://springdoc.org/> (visited on 05/20/2023).
- [60] JUnit. *JUnit*. URL: <https://junit.org/junit5/> (visited on 06/01/2023).
- [61] Google. *Google Calendar*. URL: <https://workspace.google.com/products/calendar/?hl=en> (visited on 06/17/2023).
- [62] Swagger. *Swagger*. URL: <https://swagger.io/> (visited on 06/03/2023).
- [63] Guru99. *Levels of Testing*. URL: <https://www.guru99.com/levels-of-testing.html> (visited on 06/11/2023).

- [64] Atlassian. *Types of software testing*. URL: <https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing> (visited on 06/01/2023).
- [65] duplant1s. *Chatbots4MobileTFG*. URL: <https://github.com/duplant1s/chatbots4mobiletfg> (visited on 06/17/2023).
- [66] javaTpoint. *Integration testing*. URL: <https://www.javatpoint.com/integration-testing> (visited on 06/03/2023).
- [67] Open Source Initiative. *The MIT License*. URL: <https://opensource.org/licenses/mit/> (visited on 05/20/2023).
- [68] The Apache Software Foundation. *APACHE LICENSE, VERSION 2.0*. URL: <https://www.apache.org/licenses/LICENSE-2.0> (visited on 05/20/2023).
- [69] Eclipse Foundation. *Eclipse Public License - v 2.0*. URL: <https://www.eclipse.org/legal/epl-2.0/> (visited on 05/20/2023).
- [70] GDPR. *General Data Protection Regulation (GDPR)*. URL: <https://gdpr-info.eu/> (visited on 05/29/2023).