Mestrado em Engenharia Informática 2017/2018
Estágio
Relatório Final

Android App for Enterprise Bots

Leonardo Filipe da Fonseca e Silva
lfdfes@student.dei.uc.pt

Orientador DEI:
Vasco Pereira
vasco@dei.uc.pt

Orientador WIT:
Paulo Sousa
paulo.sousa@wit-software.com

Data: 02 de Julho de 2018

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologias
Universidade de Coimbra
Pólo II, Pinhal de Marrocos, 3030-290 Coimbra
Tel: +351239790000 | Fax: +351239701266 |
info@dei.uc.pt

WIT Software, S.A.
Centro de Empresas de Taveiro
Estrada de Condeixa, 3045-508 Taveiro, Coimbra
Tel: +351239801030 | Fax: +351239801039 |
info@wit-software.com

Estagiário:
Leonardo Filipe da Fonseca e Silva
lfdfes@student.dei.uc.pt

Orientador do DEI:
Vasco Pereira
vasco@dei.uc.pt

Orientador da WIT:
Paulo Sousa
paulo.sousa@wit-software.com

Júri Arguente:
Jorge Cardoso
jorgecardoso@dei.uc.pt

Júri Vogal:
António Correia
dourado@dei.uc.pt

Abstract

In a world that every day is more technologic, lots of time is still lost, in companies, in small tasks that are not automated.

Enterprise apps have the goal of raising a company's productivity, either by facilitating communication or simplifying the way that some tasks are performed.

This project has the goal of developing features that are valuable on an enterprise app. The developed features are related to conference calls and make use of virtual assistants (chatbots). Chatbots can be useful when it comes to planning tasks, because they allow the user to interact in a more natural way to him.

The developed features are based on a combination of chatbot interaction and visual elements that speed up the tasks. The scope goes from the conference's schedule to the management of ongoing conferences and it is integrated with a communications app that already existed in the company where the internship took place.

Keywords: Conference calls, Chatbot, Android, Enterprise Apps, RCS, Calendar

Resumo

Num mundo cada vez mais tecnológico, ainda se perde muito tempo nas empresas em pequenas tarefas que não estão automatizadas.

As aplicações de contexto empresarial têm como objetivo aumentar a produtividade dos funcionários de uma empresa, quer seja facilitando a comunicação ou tornando mais simples a forma como algumas tarefas são realizadas.

Este projecto tem como objectivo desenvolver funcionalidades que tenham valor em aplicações de contexto empresarial. As funcionalidades desenvolvidas estão relacionadas com chamadas em conferência e tiram partido das capacidades de assistentes virtuais (*chatbots*). Os *chatbots* podem ser bastante uteis em tarefas de planeamento devido ao facto de permitirem uma interação por parte do utilizador mais natural para este.

As funcionalidades desenvolvidas baseiam-se numa combinação de interação com chatbots e elementos visuais que agilizam as tarefas. O trabalho desenvolvido contém funcionalidades que vão desde a marcação de conferências até à gestão de chamadas a decorrer, e está integrado com uma app de counicações que já existia na empresa onde o estágio se realizou.

Palavras-Chave: Chamadas em conferência, Chatbot, Android, Aplicações de contexto empresarial, RCS, Calendário

Acknowledgements

The biggest professional challenge that I have ever had, and the best one too.

To my **mom and dad**, for all the support, for the values that you have though me, for all the sacrifice you have made in benefit of my education. You are responsible for the person I am today and I hope that makes you both proud.

To **my brother** and **PP** for being such nice guys, for the friendship, good advices and for all the weekend nights of companionship.

To **all my family and friends** for the good times, advices and support.

To professor **Vasco Pereira** for being so dedicated, for being always available and for all the help in this document.

To **Paulo Sousa** for giving me the opportunity to take this internship at WIT, for the guidance and the advices during the various phases of the project.

To **Jorge Sousa** for being always available to help me and for the assistance in keeping my progress on schedule.

To all the **DEI teachers** for all the knowledge shared and for always being available to help and teach me.

To all the people at **WIT** that helped me and that always did it with a smile on the face.

Finally, to the most wonderful person I have ever met, **Inês**. For all the support, for all the advices, for believing in me and for making me a better person every day.

Thank you!

Contents

List of Figures

List of Tables

Acronyms

Acronym	Description
AIML	Artificial Intelligence Markup Language
API	Application Programming Interface
APK	Android Package
BPMN	Business Process Model and Notation
GSMA	Groupe Spécial Mobile Association
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MTBF	Mean Time Between Failures
NLP	Natural Language Processing
NoSQL	Non Structured Query Language
OTT	Over the Top
RCS	Rich Communication Services
SDK	Software Development Kit
SMS	Short Message Service
ToS	Threshold of Success
UI	User Interface
VoIP	Voice over Internet Protocol
XML	Extensible Markup Language

Chapter 1
Introduction

This document describes the planning and work developed during the curricular internship of the Masters in Informatics Engineering. The internship took place at WIT Software, S.A. during the 2017/2018 academic year. It was supervised by Paulo Sousa, Head of Software Development at WIT Software, S.A. and Vasco Pereira, PhD Professor at Department of Informatics Engineering of the University of Coimbra.

1.1 WIT-Software

WIT Software, S.A.[1] is a company, founded in 2001 as a spin-off of the University of Coimbra, that creates solutions and products for the mobile telecommunications industry.

Throughout the years, the company has developed several products focused on various areas. One of the fields where the company stands out is the Rich Communication Services (RCS), where it is one of the world's top players. The company has developed an app called RCS+, which will serve as a base for the software product that will be developed during the internship.

1.2 Motivation

Every company wants to see its productivity growing. For that, employees can't spend their work time with planning tasks, filling forms or scheduling reunions. It is in any company's best interests that employees make the most of the work hours they have.

Enterprise apps are developed to help companies improve their methods. And every day new enterprise apps are launched that help companies to raise their productivity.

In the year 2016, Portugal had a productivity of 68.9, considering a scale where 100 is the average productivity per work hour of the 28 countries of the European Union[2]. This list is led by Luxembourg with a productivity of 182 and Bulgaria comes on the last place with a productivity of 44.7. Portugal comes on the 18th place.

This becomes worse if we see that on average, a Portuguese employee works more hours per week than an employee in any of the top countries in productivity[3].

So, if Portuguese employees work more hours and produce less, it urges to raise productivity in Portuguese companies.

Since so much time is lost to planning tasks, reducing the time of those tasks would improve the quality of the working time. More efficient than filling forms or use computer programs to perform those tasks would be to tell "someone" using natural language to do it. That is where a new agent acting as a virtual assistant could be helpful. An assistant may be used to help in the accomplishment of the tasks. This allows the user to perform the task in a much more natural way to him. For example, when a user wants to book a room, usually it is necessary to fill out some form, with this new agent, a user just needs to tell it something like "I want to book a room for tomorrow at 2 pm" or "What is the availability of the rooms for tomorrow?" and most parts of the tasks are performed by the agent and not by the user, therefore, saving time and allowing the user to focus his work time on actual work.

One of the tasks in which the productivity can be improved is the management of conference calls. From the scheduling to the call itself.

To schedule a conference call, one has to check his availability to know when to schedule the call, send invitations to all the participants and make the actual call. That is much more time consuming than it should be. During the conference call, most of the time we don't know who is already participating or who have already left the call. People talk at the same time, people talk and then realize that no one heard because they were on mute… Conference calls can be complicated.

But this is one of the cases where a virtual assistant could be helpful. The assistant could show us our agenda so we know our availability. Then it could send the invitations to the guests. And during the call it could show who is participating in a given moment, show who is muted or allow the conference manager to perform simple actions like mute or remove participants.

That is what has been developed during the internship. The use of an agent to manage conference calls.

WIT already had a conference call system. But it was not widely used in the company. The lack of its success was due to the complexity on scheduling the conferences and to the fact that the conferences were reduced to "simple phone-calls". That led the users to opt-in to use other services to make the conference calls.

1.3 Goals

1.3.1 Personal goals

For this internship, two main personal goals were established.

The first one was to consolidate all the knowledge acquired during the last years, first in the bachelor's in informatics engineering and then in the masters.

The other goal was to grab the opportunity given by WIT Software and the University of Coimbra and develop a project that was valuable for the company and that allowed me to finish my master's with success.

1.3.2 Project goals

The main goal of this project was to use chatbots in order to simplify conference calls' related tasks that are performed in companies and that are much more time consuming than they should be. The internship was focused on simplifying conference calls, from its scheduling to the calls' management.

For that, a communications app produced by the company was used. That app allows the user to perform phone calls and trade messages using the network instead of the traditional services provided by telecommunication companies. The goal is to create a conference calls' module using chatbots to facilitate its scheduling features and operation. The conference calls' related features are relevant for a future enterprise version of the app.

The goals were fully achieved during the internship. Using the app, it is now possible to use a bot in order to easily schedule, modify or cancel conference calls and, it is possible to manage ongoing calls in which the user is participating.

1.4 RCS

Smartphones are extremely common nowadays. This kind of mobile phones brought new possibilities to communicate through IP. Over the Top (OTT) apps like Skype[7] or Facebook Messenger[8] use the operators' infrastructures to provide quick, and most of the times free, access to this kind of communication.

Telecommunication operators see OTT platforms as a threat to their income because users opt to communicate using the OTT apps instead of the traditional communication services provided by the operators.

In order to fight this competition, a group that represents the interest of operators called *Groupe Spéciale Mobile Association* (GSMA)[9] created the RCS program, which would allow the operators to provide enriched services (VoIP calls, group messages, file share…).

The app that served as a base for this project (RCS+) follows the RCS specifications.

1.5 Structure

This document is structured as follows:

- **Chapter 1 Introduction** – This section introduces the project, the motivations for its development and its goals.
- **Chapter 2 State of the Art** – This section contains an analysis of the competitors for the project and a presentation of the used technologies.
- **Chapter 3 Methodology and Work Plan** – This section presents the methodology of work adopted, as well as the work planning, the definition of the expected outcome of this project and an analysis of the factors that might become a risk for the project's success.
- **Chapter 4 Requirements** – This section presents the requirements elicitation of the project.
- **Chapter 5 Architecture** – This section explains the architecture of the project.
- **Chapter 6 Implementation** – This section describes the work done for the development of the defined features.
- **Chapter 7 Verification & Validation** – This section describes the work done for the verification and validation of the developed features
- **Chapter 8 Conclusion** – This section has an overview of the entire internship and the progress that has been made.

Chapter 2
State of the art

In this chapter, the state of the art in enterprise apps that support conference calls' management and in bots that allow event's schedule is analyzed. The analysis of the products that compete with the features to be developed allows to look into each functionality and see if it brings value to the project or if there are better alternatives.

2.1 Competitors

There are two types of competitors for this project. The first type, are the services that allow the user to perform conference calls. The second one, are the chatbots and virtual assistants that support actions related to the schedule of events.

2.1.1 Apps supporting conference calls

The first type of competitors comprises mobile apps that can provide solutions for conference calls' management.

For each of the apps, the main features related to conference calls were analyzed:

Figure 2-1: Workplace by Facebook logo[10] **Workplace by Facebook**	• Who is on the conference call • Participants detailed information

Table 2-1: Workplace by Facebook app details[11]

Workplace by Facebook is an enterprise version of the Facebook app, which allows the employees of a company to communicate, share contents and make calls, both audio and video.

Figure 2-2: Skype for Business logo[12] ***Skype for Business***	• Who is speaking • Who is muted • Who is on the call

Table 2-2: Skype for Business app details[13]

Skype for Business is an app focused on video and audio calls, but also allows message and files sharing.

Figure 2-3: Cisco WebEx Teams logo[14] **Cisco WebEx Teams**	• Who is muted • Who is speaking • Who is on the conference call

Table 2-3: Cisco WebEx Teams app details[15]

Cisco WebEx Teams allows the users to schedule and make conference calls, with support for both audio and video.

Figure 2-4: Zoom Cloud Meetings logo[16] **Zoom Cloud Meetings**	• Who is muted • Who is on the call

Table 2-4: Zoom Cloud Meetings app details[17]

Zoom Cloud Meetings it's an app developed by Zoom, focused on enterprise communications.

The following were considered the main features regarding conference calls on mobile apps:

- **Who is speaking** – Know which participants are speaking at a given moment during a call
- **Who is muted** – Know which participants are muted on a given moment during a call
- **Who is on the call** – Know which participants are on a call on a given moment
- **Participants information** – Provide access to detailed information about the participants
- **Remove participants** – The ability to remove a participant from a conference call

Although the last one is not present in the analyzed apps, it was considered to be relevant to the context of the project.

Table 2-5 indicates the functionalities that will be included on this project.

Who is speaking	Who is muted	Who is on the call	Participants detailed information	Remove participants
✗	✓	✓	✓	✓

Table 2-5: Ongoing conference functionalities to be developed

All the functionalities considered relevant for an ongoing conference call will be implemented on this project, with the exception of the "Who is speaking". Although it is a nice feature to have, in the context of this project it would not have the desired behavior. Since the warnings of the conference's events will arrive on the app through chatbot messages, their delivery time may vary. With that constraint, the functionality would not work in the correct way. The remaining functionalities are not affected by this constraint and will be included on the project.

2.1.2 Chatbots

The other kind of competitors for this project are chatbots or virtual assistants that allow actions related to the schedule of events.

A chatbot is a computer program that uses natural language in order to interact with users. Most of the times, the main goal of chatbots is to simulate human behavior, thereby, trying to pass the Turing test[4].

Turing test consists of a human evaluator, having a textual conversation with another human and with a machine. The machine passes the test if the first human cannot reliably tell which one is the machine[4].

Eliza was the first known chatbot. It was developed during the 1960's[5] and its main purpose was to simulate the responses that a therapist might have during an intimate conversation. Although it's conversational capacities were not great, it still lead some people to think that they were actually talking with a human, during the first minutes of dialog.

Chatbots are much more evolved these days and can be found in lots of different "places" and with lots of different purposes. Most instant messaging platforms support chatbots nowadays, from the simpler ones to the more complex ones. Lots of services use chatbots in customer support. And there are also virtual assistants like Apple's Siri, Amazon's Alexa and Microsoft's Cortana. Some of these chatbots use sophisticated natural language processing (NLP) techniques and other use simpler methods like matching patterns.

Taking into account that the app that served as a base for the project supports chat interaction with chatbots, it was considered that the capabilities provided by this app fit perfectly on the definition of "agent" that was needed for the project.

In the field of chatbots, lots of progress has been made in the last few years. The number of agents that are capable of human interaction using natural language keeps increasing, and nowadays, almost every device comes with a personal virtual assistant.

The following were considered the most relevant players acting on this field. For each one of them, the major positive and negative aspects related to the schedule of events were analyzed.

Figure 2-5: Amazon's logo[18] **Amazon's Alexa**	• Voice control support • NLP support for the schedule of meetings • Only supports 1 on 1 meetings • Verification of both calendars (if integrated) • Easy events modification

Table 2-6: Amazon's Alexa[19]

Figure 2-6: Microsoft's logo[20] **Microsoft's Cortana**	• NLP support for the schedule of meetings • Verification of all participant's calendars (if integrated) • Easy events modification

Table 2-7: Microsoft's Cortana[21]

Figure 2-7: X.AI's logo[22] **X.AI's Amy/Andrew**	• NLP support for the schedule of meetings • Possibility to add participants to the events • Verification of all participants' calendars (if integrated) • Integration with Slack, Zoom, Outlook…

Table 2-8: x.ai's Amy/Andrew[22]

Figure 2-8: Apple's logo[23] **Apple's Siri**	• Voice control support • NLP support for the schedule of calendar events • Easy events modification • Strong error proof details

Table 2-9: Apple's Siri[24]

The analysis of these competitors helped in the perception of what aspects are really important when it comes to the schedule of events.

Since the project to be developed is focused on conference calls, it is important to support the addition of more than one invitee per event.

One of the strong aspects of some of the solutions analyzed is the voice control action. This feature allows the schedule to be simple and fast. Since voice control will not be supported on this project, it is necessary to find a way that allows the schedule to be equally simple and fast.

Textual interaction will be part of the solution adopted, but if all the interaction is done in a textual way, the schedule will be slower than desired. To complement the textual communication, graphical components will be used. This will allow users to interact in a more spontaneous way.

Calendar verification is another important aspect when it comes to schedule events. The solution developed will comprise calendar verification of the user who is scheduling the conference. The verification will be done on the user's Google Calendar[25], once access is granted to the bot. Verification of all participants' calendars in order to seek for a time slot available for all, is a relevant functionality, but, it will not be supported in this project. This choice was made because not everyone uses the same calendar service. In a near future, WIT will develop its own calendar service, and then, this functionality may be developed and integrated.

Another important aspect revealed during this analysis is the importance of having a simple way to edit the created events. Users regularly have to postpone, anticipate or simply cancel conference calls, and this should be as simple as it possibly can. In order to have a simple event modification system, the same approach used for the schedule will be adopted (the use of graphical components).

One detail that was also considered relevant was the techniques used by Apple's Siri to avoid mistakes. For example, if a user is asking to schedule an event for "tomorrow" and it's close to midnight, Siri will ask the user to confirm the date. This and other details will be adopted on this project's solution.

The following were considered the main features regarding the schedule of events on a chatbot:

- **Calendar verification** – Check if the user doesn't have anything else scheduled for the pretended time period
- **Voice control** – Interact with the chatbot using voice
- **NLP support** – Use NLP on the bot in order to interact with the user
- **Simple event modification** – Provide an efficient way to modify or cancel conference calls
- **Mistakes avoidance** – Have procedures that allow the avoidance of mistakes triggered by misinterpretation of user's inputs

Table 2-10 indicates the functionalities that will be included on this project.

Calendar verification	Voice control	NLP support	Simple event modification	Mistakes avoidance
✓	✗	✓	✓	✓

Table 2-10: Conference schedule features to be developed

2.2 WIT RCS Suite

The RCS+ app, on which the developed software will be integrated, is part of the WIT RCS Suite (Figure 2-).

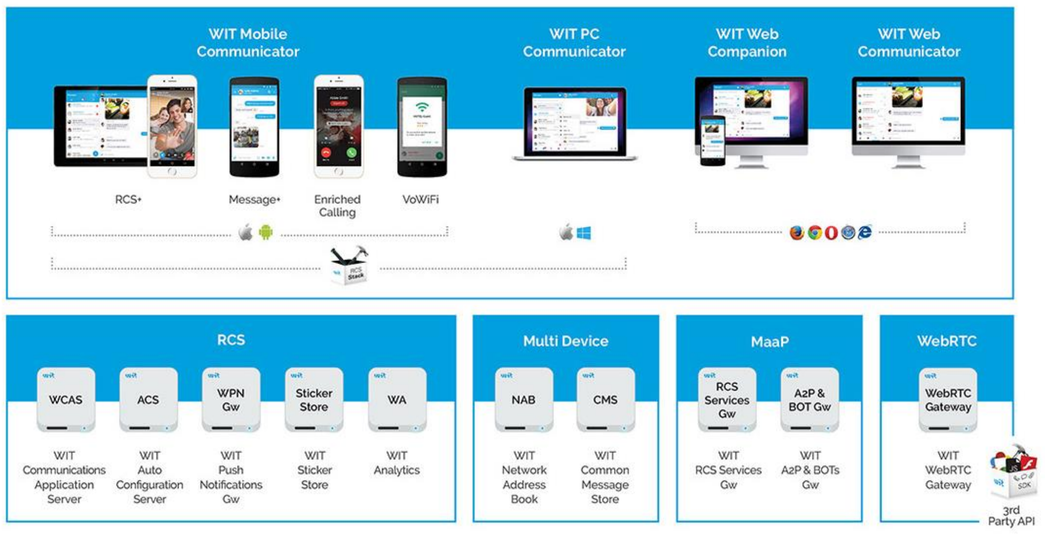


Figure 2-1: WIT RCS Suite

This Suite also includes components like WIT PC Communicator, which uses the WebRTC protocol and WIT Communications Application Server, which implements the RCS specifications, among others.

The RCS+ is an app that follows the RCS specifications. Among other things, this app allows the user to make Voice over Internet Protocol (VoIP) calls, send and receive messages or interact with chatbots. All of this using the network instead of the traditional communication services provided by telecommunication companies.

The RCS+ app has the following main features (among others):

- VoIP calls
    - Voice calls
    - Video calls
    - Audio and video messages
    - Content Share in calls

- Messages
    - Single chat
    - Group chat
    - Content share
    - Chatbots

This app is one of the biggest projects at WIT, and it has been developed for many years now. The app keeps evolving and new versions are released. The work developed during this internship was based on the most recent version at the time of its start, version 4.4.

RCS+ is the main focus of the project. New features were added to it, some of them completely new, and others taking advantage of functionalities that the app already had.

2.3 Features to be developed

Having analyzed the competitors for this project, it was necessary to define what would be developed and integrated in the existing app. The app supports chat messages and the use of *chatbots*, which will be used for the call scheduling and management.

Table 2-11 contains a generic description of the major elements that were defined.

Feature	Description
Call Scheduling	This module allows the user to schedule and modify conference calls, based on the user's calendar availability and allowing the user to invite his colleagues for the calls.
During Call Management	During a call, the user will be able to view all the call's events and also to see detailed information about a participant
Calendar Tab	The user will be able to see his calendar and use it to view scheduled conference calls or schedule new ones.
LDAP Integration	The app will synchronize the WIT workers' contacts so the user can add them to conference calls.

Table 2-11: Features to be developed

By the end of the project, a user should be able to ask the bot to schedule a call. Should be also possible for a user to ask the bot to show his calendar with his events. On the calendar view (with the user's current calendar events) the user must be able to select a slot to schedule a conference call and then, choose, from a list of contacts, the participants. On the calendar view, the user should also have the possibility to edit a conference call's details.

When an operation is done, the user should receive feedback from the bot, telling if the operation was successful or not.

While on a conference call, the user should have access to the call's basic information (duration and number of participants). This information should be on an expandable component. When expanded, the component should contain information about the call's participants, alongside with information about who is muted/unmuted. When the user selects a participant, information about that person should be displayed.

A tab containing the user's calendar should be present on the main menu of the RCS+ app. On this calendar, the user should be able to see his upcoming events and have the possibility to schedule conference calls and edit them.

2.4 Technologies

Android Studio[26] was the chosen platform for the Android development. This choice was based on the fact that this was the platform used by the more experienced colleagues at WIT, and therefore, was the platform where the RCS+ app was developed, making it simpler to integrate the developed software with the remaining app. Also, as it was used by the colleagues, it was the platform on which was easier to obtain help when needed.

For the bot, the technologies used were Business Process Model and Notation (BPMN)[27], for the flows that the conversations should follow, and Artificial Intelligence Markup Language (AIML)[28] for dialogs that follow stimulus-response interactions. These technologies were adopted because they were the ones used by the company in the bots' area.

Dialogflow platform was used as an Natural Language Processing (NLP) interpretation tool. This tool was already used on other WIT's bots, hence its choice.

2.4.1 BPMN

BPMN is a graphical representation for specifying business processes in business process models. This language allows the creation of paths that can be followed during the execution of a program. This language is composed of several types of elements. The elements used in this project are described in further detail below.

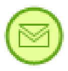
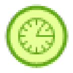
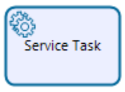
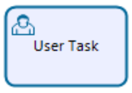
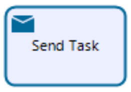
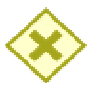
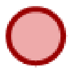
Start Events		Starts a flow. It is triggered by the arrival of a message. Once a pattern matching the start event arrives, the flow is started and the path will be followed until an "end event" is found.
		Starts a flow. It is triggered with a time loop. Can be triggered for example every 5 minutes. Once a pattern matching the start event arrives, the flow is started and the path will be followed until an "end event" is found.
Tasks		A task that represents that some "backend" work has to be done before proceeding to the next step on the flow.
		This task requires an input by the user. The flow will stop and wait for this input.
		Some output is sent to the user.
Gateways		It's used to fork and merge paths. It can have several types, but for this project, only exclusive gateways were used. An exclusive gateway executes only one of the paths that it is merging or forking.
End Events		Represents the end of a flow.

Table 2-11: BPMN events

2.4.2 AIML

AIML is an XML based markup language used for the creation of natural language software agents.

In AIML there are two fundamental "tags".

- **Pattern –** The pattern that may be equal to the user input
- **Template –** The response associated with a pattern.

In this work, AIML and BPMN were used together. If a user's input matches an AIML pattern, the BPMN start event associated with this pattern will be triggered, starting a BPMN flow.

BPMN's "send tasks" can also trigger AIML patterns. When that happens, the template or one of the templates associated with that pattern will be sent to the user.

2.4.3 Natural Language Processing

NLP is a field of computer science that studies how it is possible for a machine to understand and handle natural language in order to have a "more natural" interaction with a human user. Languages like English or Portuguese are easier to be understood by humans, but they cannot be understood by machines. On the other hand, languages like Java were developed to be interpreted by machines, but they are not usable for humans as a communication tool.

Since humans communicate with each other through natural language, the easiest way for a human to communicate with a machine is also the use of natural language. In order to perform this interaction, tools were developed that allow the interpretation of the intents behind each user's input.

Since the main focus of this internship was not NLP, DialogFlow[6] platform was used in order to perform the natural language interpretation by the chatbot's side on the project.

Chapter 3
Methodology and Work Plan

This section is dedicated to the presentation and description of the methodology adopted, as well as the work plan to be followed during the internship. The work plan includes a planning of the tasks that were done during the course of time, the definition of the final success scenario and a presentation of the risks that might interfere with that scenario. Planning is a very important part of this project, as it allows to define personal milestones to be accomplished and it gives a better control of one of the most important resources of this project – time.

3.1 Methodology

Every software project must follow the methodology that fits better its needs. In this case, an AGILE methodology may be more adequate, since, the project's scope was not closed. During the project's life cycle, new requirements may be included. An adaptation of SCRUM[29] was adopted.

In SCRUM, at the beginning of a project, the features that will be implemented in the product are defined and joined in what is called the product backlog. During the course of the project, new features can be added to the backlog and others can be removed. The development phase is divided into cycles that are called sprints. For each sprint, a set of tasks from the product backlog are selected and will compose the sprint backlog. The duration of the sprints should be defined according to the necessities of each project. On each sprint, there is a set of formal events. The first one is the sprint planning, on which the tasks for the sprint backlog are selected. Tasks are selected taking into account the time and effort that they take to complete in order for the sprint not to have too many tasks nor too few. During the sprint, every day there is a meeting called daily scrum, on which the team elements talk about what they've done since the last meeting what they plan to do until the next. And then, typically in the last day of the sprint, there are the sprint review and the sprint retrospective, on which the team talks about the final product of the sprint, and about how the sprint went.

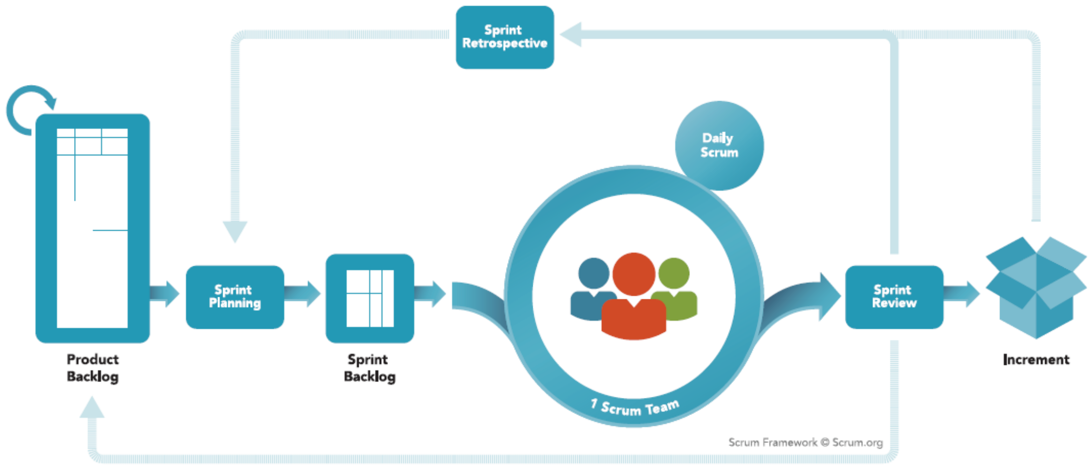


Figure 3-1: SCRUM Framework

As the project is developed only by the intern, there isn't a clear definition of roles. And in the methodology adopted, there aren't formal meetings. Instead, the intern's tutor is always aware of how the work is going and any problem is reported immediately. This is possible because the intern and its tutor work side by side.

3.2 Work plan

Table 3-1 gives an overview of the work plan to be followed during the internship. The complete plan can be found in Appendix A.

1st Semester	State of the Art	September/October
	Requirements	October/ November
	Architecture	November
	Development "Call Scheduling Module"	December/January
	First Presentation	January
2nd Semester	Bot integration	February
	Verification and Validation	February
	Development "During Call Module"	February/March/April
	Verification and Validation	May
	Development "Call Tab"	May/June
	Verification and Validation	June
	Functional Tests	June
	Final Presentation	July

Table 3-1: Work Plan

3.3 Threshold of Success

In order to know if a project was succeeded, there must be a criterion to measure against. The Threshold of Success represents the boundary between success and failure of a project.

For this project the following Threshold of Success (ToS) was defined:

- Have all Must Have requirements (RequirementsChapter 4) accomplished by the end of the project.

The definition of what is success in a given scenario increases the chance of achieving it because it allows us to keep straight following the road that leads to the success scenario defined.

3.4 Risk Management

In every software project, there are risks that should be managed. This process allows the minimization of their possible negative impact. An incorrect handle of the risks might lead to a project failure. Risk management should be a continuous process, because, as the project evolves, new risks might appear and some might be mitigated.

A risk is an event that can have a negative effect on a software project.

The identifications of the risks is the first step of the risk management, followed by analyzing, planning and monitoring.

3.4.1 Risk metrics

As for the metrics to measure the risks, there is the impact that they will have on the project in case of happening and the probability, which represents how likely they are to occur.

- Impact
  - Low – ToS can be achieved without much extra work
  - Medium – ToS can be achieved but with extra work
  - High – ToS isn't likely to be achieved
- Probability
  - Low – probability is less than 30%
  - Medium – probability is between 30% and 70%
  - High – probability is more than 70%

After the identification and evaluation of the risks, a risk matrix is used to ease the understanding of their influence on the project. The matrix used is divided into 9 categories with different colors. The green zone is for the risks that represent a low danger for the project, the yellow, for the ones that represent a medium danger, the orange is for the ones that represent a high danger and the red zone is for the most critical risks.

🟩	• Low danger for the project
🟨	• Medium danger for the project
🟧	• High danger for the project
🟥	• Critical danger for the project

3.4.2 Risks

Risk 1 – Android learning	
Description	The intern has never worked on mobile platforms before; might cause some difficulty in the adaptation to the project
Impact	High
Probability	High
Mitigation Plan	• Study of the Android programming techniques • Experienced colleagues at WIT Software are available to help

Table 3-2: Risk 1 - Android learning

Risk 2 – Adaptation to RCS+	
Description	The app to be developed is based on an existing one; might take some time to understand how the existing app is structured
Impact	High
Probability	Medium
Mitigation Plan	• Study of the existing documentation • Experienced colleagues at WIT Software are available to help

Table 3-3: Risk 2 - Adaptation to RCS+

Risk 3 - Estimations	
Description	The intern doesn't have much work experience; might lead to bad effort estimations
Impact	High
Probability	Low
Mitigation Plan	• Help of supervisors in the definition of the work plans

Table 3-4: Risk 3 - Effort estimations

Risk 4 – Android devices and versions	
Description	The number of different devices that run Android and the number of Android versions is quite big; In some devices, the app might not run correctly or may have some bugs.
Impact	Medium
Probability	High
Mitigation Plan	• Use different devices for tests • Try to cover the maximum Android versions with the code components used

Table 3-5: Risk 4 - Android devices and versions

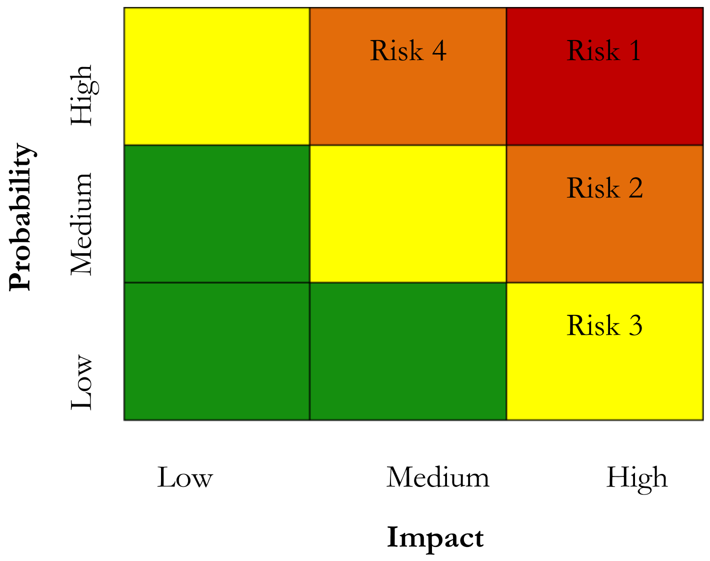


Figure 3-2: Risk Matrix

As can be seen in the risk matrix, risks 1, 2 and 4 are in the orange and red zones, which means that are the most likely to damage to the project's evolution. For that risks, the mitigation plans must be followed carefully in order to lower the impact that they might have on the project.

As for the remaining risk, although it has a low probability, it has a high impact, and so, due to the low number of risks, the mitigation plan will be followed carefully for this risk also.

Chapter 4
Requirements

One of the first steps of a software project is the requirements analysis, and it is one of the most important too. Before doing something, it is important to decide what to do and how to do it.

This process will almost always help in the understanding of the problem, allowing the developer to have a clearer vision of it and minimizing the chances of failure. The first step of the requirements elicitation of this project was the definition of User Stories. The choice of this method is related to its simplicity and that fact that it allows the developer to view the problem under the perspective of all the personas that have interest on a certain feature.

4.1 User Stories

User Stories are very high-level requirements. They should be short, simple, and be written in the perspective of the person who desires the capabilities described. Typically, they are written according to the formula "*As a <person who desires a capability> I want to <goal> so that I can <reason>*".

The definition of User Stories is useful to understand the point of view of the personas involved on the project and to proceed to the functional requirements definition.

The way the User Stories are written, allows any team member to estimate the cost of a task in a very easy and quick way.

User Stories			
ID	**As a**	**I want to**	**So that I can**
US1	User	Schedule a conference call	Talk with other people
US2	User	Invite contacts	Choose who is participating in a conference call
US3	User	Know if a date is available for a call	Schedule a call
US4	User	View my calendar	Check my availability to schedule a conference call

User Stories			
ID	**As a**	**I want to**	**So that I can**
US5	User	Receive a confirmation after a call schedule	Know it was successfully scheduled
US6	User	Have my co-workers contacts on the contact list	Invite them to a conference call
US7	User	Receive a reminder when a call is about to start	Not forget about a conference that I am supposed to be in
US8	User	Know who is participating in a conference call	Know who I'm talking to
US9	User	Mute a participant on a call that I have created	Mute a participant
US10	User	Unmute a participant on a call that I have created	Unmute a participant
US11	User	Remove a participant from a call that I have created	Remove a participant
US12	User	Know who left a conference call	Know who still is on the conference call
US13	User	See information about a person that is on a conference call	Know who I'm talking with
US14	User	See who is muted on a conference call when I am a guest	Know who is muted
US15	User	I want to receive a notification when I am invited to a conference	Know when I have a conference

Table 4-1: User Stories

4.2 Functional Requirements

After the definition of the User Stories, it is easier to write the functional requirements of the project. This second approach allows the division of the user stories in specific tasks, which are better guidelines for the development process.

Each requirement is composed by an ID, a description and a priority field. The prioritization of the requirements was made according to the principles of the MoSCoW method[30], which consists on the distribution of the requirements through the following four categories:

- **Must Have:** requirements that are critical for the success of the project
- **Should Have:** requirements that are important, but not mandatory for the success of the project
- **Could Have:** requirements that are desirable but not necessary for the project
- **Won't Have:** requirements that will not be included in this version of the project

Table 4-2 contains the project's functional requirements.

Functional Requirements		
ID	**Description**	**Priority**
FR1	User can select a date for a call while scheduling it	Must Have
FR2	User can select participants for a call while scheduling it	Must Have
FR3	User can ask the bot to schedule a call specifying a date and an hour	Must Have
FR4	User can ask the bot to schedule a call specifying only a date	Should Have
FR5	User can ask the bot to schedule a call without specifying a date or an hour	Must Have
FR6	User can ask the bot to view his calendar	Must Have
FR7	User can provide a name for a call	Could Have
FR8	User can cancel a call he has scheduled	Must Have
FR9	User can edit the date of a call he has scheduled	Must Have

Functional Requirements		
ID	**Description**	**Priority**
FR10	User can see who are the invited participants for a call	Must Have
FR11	User can edit the participants of a call he has scheduled	Must Have
FR12	User can see the number of participants on an ongoing call	Should Have
FR13	User can see who is on the call on a given moment	Must Have
FR14	User can see an ongoing call's duration	Must Have
FR15	User can see who is muted/unmuted on a conference call	Must have
FR16	User can mute/unmute participants on a call that he has created	Must Have
FR17	User can see call participant's information	Must Have
FR18	Bot sends message saying if an action was successfully done	Must Have
FR19	Bot sends message saying that a call is about to start	Must have

Table 4-2: Functional Requirements

4.3 Nonfunctional requirements

As the functional requirements define what the app does, nonfunctional requirements define how the app works.

Non-Functional Requirements		
ID	**Description**	**Priority**
NFR1	**Data consistency** – The conference calls' data stored on the Asterisk server must be consistent with the data on the bot's database and with the data written on the user's Google Calendar. To achieve this, all the data writing must be done in a transactional way.	Must Have
NFR2	**Reliability** – At least 40 hours of Mean Time Between Failures (MTBF), where a failure can be a conference call that is not really scheduled on the calendar although a positive feedback was received by the user, a conference call that the user is not warned about, or, the user receiving information that is not accurate about a call during that call.	Must Have

Table 4-3: Nonfunctional Requirements

Chapter 5
Architecture

This chapter contains an explanation of the system's architecture. A good definition of the system's architecture is an important step in a software project's lifecycle. The architecture will define how the system will work. A good architecture should be simple, complete and precise. If not, the developers may have a wrong interpretation of it and problems may arise from there.

In order to achieve the desired easily interpreted architecture, various views were made. Each view represents a different vision of the problem or of part of it. This approach was chosen over having just one big diagram of the all project because when only one diagram is defined for the architecture of such a big system, that diagram, even if well designed, will almost always lead to misinterpretations. On the other hand, the approach adopted allows the person who reads the architecture to view it from different perspectives and to have more detailed information about each component of the system.

5.1 Overall System

Figure 5-1 represents the overall architecture of the system.

On this section it is possible to see that the RCS+ app, which is one of the two main components of the project, communicates through HTTP with the other main component, the WIT's Bot Platform. This communication channel is the only one used by the RCS+ app. All the services necessary to provide the information for the app are reached through the bot's platform, where the bot is.

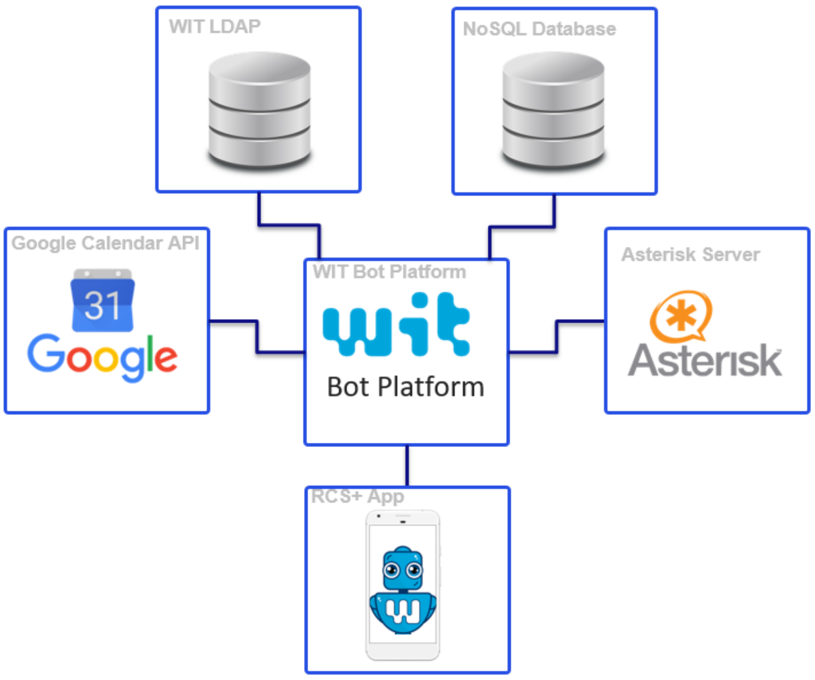


Figure 5-1: System's overall architecture

The RCS+ app sends chat messages to the WIT's bot platform, and receives messages from it. The bot, on the WIT's bot platform communicates with several services in order to perform the tasks related to conference calls.

The bot has a NoSQL database where data like bot's users, conferences or lunch orders are stored. On this database, all conferences are saved on a collection named "conferences". Each conference on this collection is composed of the following fields.

- Id – a unique identifier for the object;
- Title – the title of the conference;
- Date – the start date of the conference;
- End – the end date of the conference;
- Number – the conference identifier on the Asterisk server;
- SecurityPin – the number to be dialed by users to join the conference
- ModeratorPin – the number to be dialed by the conference owner;
- Description – a description comprising all the conference's main information;
- Participants – the group of participants that are invited to the conference;

The bot also communicates with an Asterisk server. This server is used to perform VoIP calls. The communication with this server occurs during several operations. When a conference is scheduled, a request is sent to the Asterisk server. Here, the most important aspect is the conference identifier, a number that can be used in order to join the conference that was created (once it starts). When the conference starts, that identifier is used by the bot in order to send a request to the Asterisk server, so that a call to join that conference is performed to the users' phones. Those are situations where the bot sends requests to the Asterisk server. But there are also some cases where it is the Asterisk server that performs the communication to the bot. That happens when a conference event occurs. The events that can trigger that communication are the following.

- Conference starts
- Conference ends
- User joins conference
- User leaves conference
- User is muted
- User is unmuted

Those events are communicated to the bot in order to be multicasted to all of the conference's participants.

There are also communications with the Google Calendar API service. Requests are sent in order to perform both reading and writing operations. The calendar reading is the operation that occurs more frequently, since requests to update the calendar are sent, for example, always that the user opens the calendar tab on the RCS+ app. The writing operations are performed when it is necessary to schedule, modify or delete a conference.

The other service used by the bot is the LDAP server. Communication with this service is necessary in order to identify the conferences' participants.

5.2 RCS+

The RCS+ app is composed of many modules. Figure 5-2 presents a diagram with the app's main modules, identifying the one that was created and the modified ones.

The conference calls module was entirely created during the internship. This module manages most of the things that are related to the conference calls.

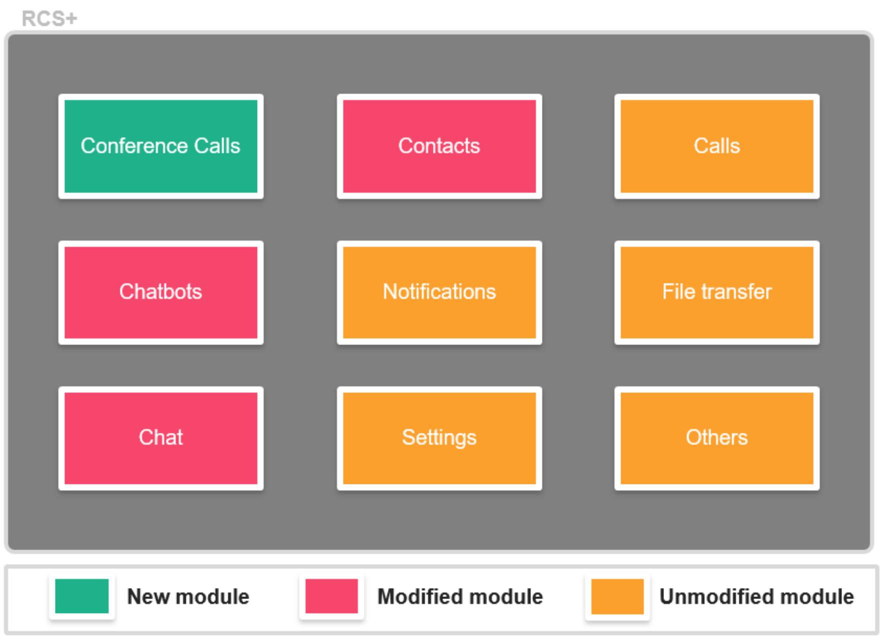


Figure 5-2: RCS+ modules

The architecture of the conference calls module can be seen in Figure 5-3, as well as its interaction with the remaining modules.

The various components of the conference calls module have connections between them but also with components of other modules.

The **calendar tab** element on the conference calls module, is managed by the app's tab manager. It is an Android's fragment, and it contains a calendar view element. It also contains the ongoing call expandable component and it is connected to the chat module, so it can receive the updated calendar events of the user's calendar.

The **calendar view** is the calendar that allows the user to select time slots to schedule conferences. It is used on the calendar tab and on chat messages sent by chatbots. A calendar view has an ArrayList of events and an event has an ArrayList of participants. From the calendar view, it is possible to start the scheduling of a conference call.

The **ongoing call manager** is connected to the chat history manager, this connection exists so that when the app starts, a search for ongoing conference calls is performed. It also communicates conference calls' events to the conference call expandable component. And, through the chat module, it sends messages to perform actions like mute or unmute conference participants.

The **chatbot chat message calendar** is a type of message to support the use of the calendar view on a chat message that is sent by a bot.

The **WIT contact manager** receives the messages sent by the bot containing the information related to the WIT's workers contacts. It turns that information into contacts and adds them to the contact's list.

The chatbots' module comprises a variety of chat message types. The message types allow for example the use of sound messages, messages with maps to share locations or messages with videos. A new type of message was created to support the use of the calendar view on a chat. The chatbot chat is the component where it is decided how to treat each chatbot message type. This component was modified in order to decide what to do with the new message types defined.

The chat module contains the chat manager, the component where the chat messages arrive. This component was modified in order to identify the type of message that arrives and forward it to the right place. A message is forwarded to the chatbots' module if it is a chatbot type message, to the ongoing call's listeners if it is an ongoing call event or to the contact's module if it is a message containing the contacts to be added to the contact list.

On the contact's module, the WIT's contacts manager was created in order to process the contacts that arrive via bot message and the android contact manager was modified in order to perform the contact integration.

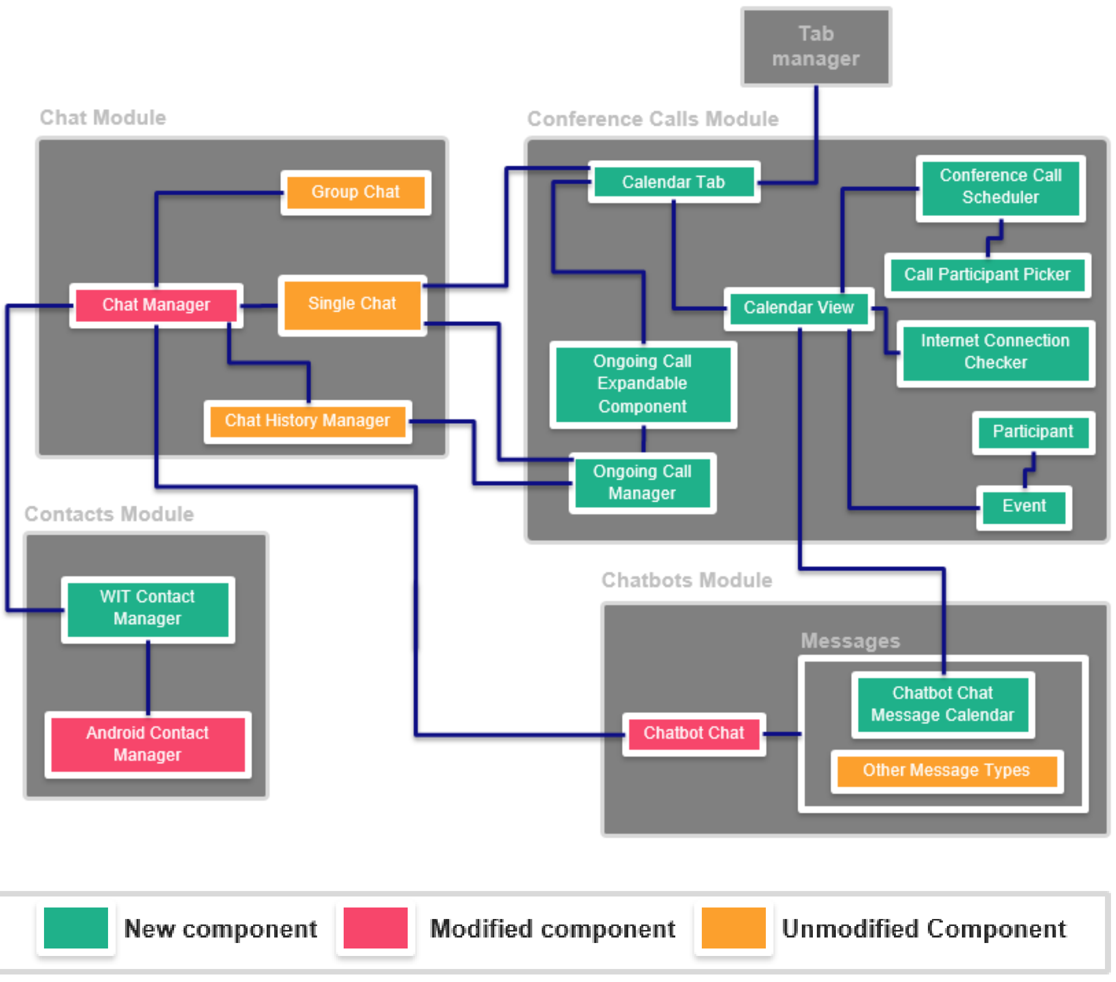


Figure 5-3: RCS+ created and modified elements

When a conference call is happening, a component with the conference details is visible on the app. This component is updated in runtime by chat messages that come from the bot. When the app is running, there are listeners that are triggered every time a message related to a conference call arrives. This approach works well when the conference events happen while the app is running. Nevertheless, if the app is restarted, this method of getting the events does not work, because the events that had been already received, will not be re-sent, and when a new event arrives, it will not be recognized because as far as the app "knows" there are no conferences happening.

In order to solve this problem, a chat verification needs to be done on the app start. The calendar tab is one of the places where the ongoing call component can be found. Since this tab is initialized when the app starts, the chat verification can be done here.

The chat history is stored on a database, and it's accessed by the app through the COMLib. The COMLib is a communication application operating with a stack protocol, which provides communication components like messages, to the RCS+ App. It follows a service oriented architecture, in order to be well organized and easily configurable. When the app starts, the calendar tab, sends a request to the COMLib to get the bot's chat conversation. When the response comes, the messages are analyzed and if there is a message related to a conference start that is not followed by a message related to a conference end, it means that there is a conference happening. After processing a message related to a conference start, a search occurs for other conference events, like participant entrances or exits.

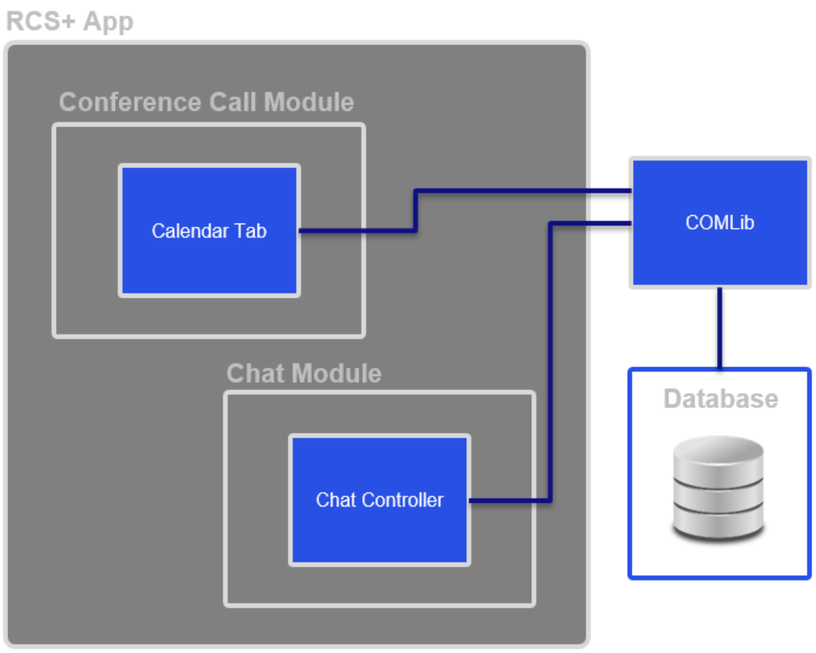


Figure 5-4: RCS+ and COMLib interaction

5.3 WIT Bot Platform

Figure 5-5 allows the understanding of the whole bots communication structure inside WIT's network.

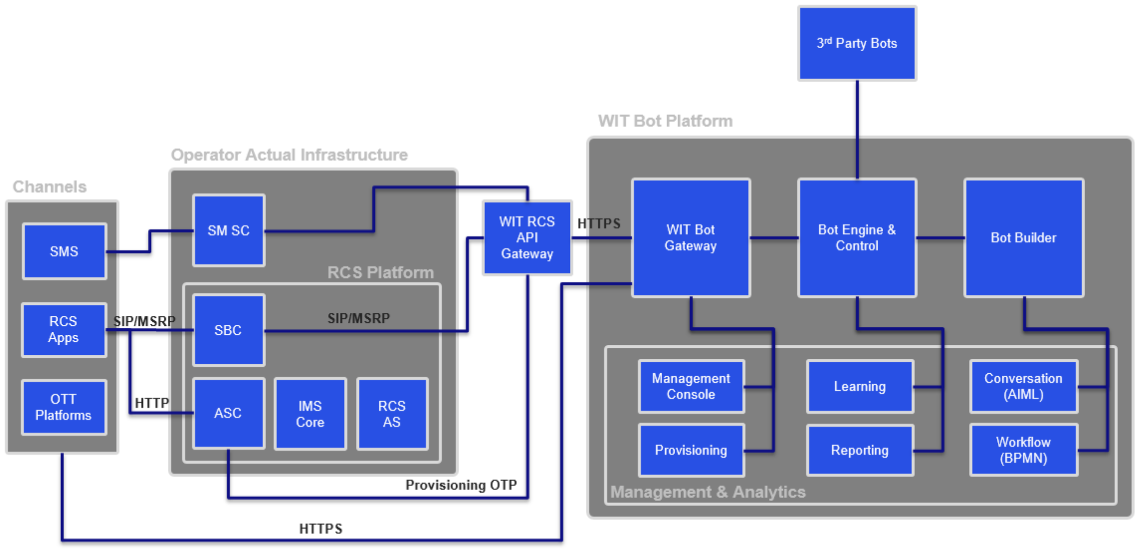


Figure 5-5: WIT Bot Platform

As the diagram shows, bots can be used through various channels. Either through SMS, Over the Top (OTT) platforms like Facebook Messenger[8] or the RCS apps.

The requests to the WIT Bot Platform are made using the HTTP protocol. Those requests may come directly from the OTT platforms or through the WIT RCS API Gateway, which serves as intermediate for SMS and for RCS apps.

WIT Bot Platform is where all the bot's artificial intelligence is done. This server is composed of 3 major components.

- **WIT RCS Bot Gateway** – This component is responsible for receiving HTTP requests and sending them to the proper next step. It is also responsible for sending the HTTP answers back to their origin. This component also adapts the response contents, having in consideration what is supported by the client.

- **Bot Engine & Control** – The component that allows the integration with 3rd party bots, that allows bots to have access to users' information and that includes the module of the conversation engine. When the bot receives a message, it decides the flow of actions to be taken.

- **Bot Builder** – Here, is created and stored the bots conversational capabilities. These conversation capabilities are defined with BPMN workflows and AIML files.

5.2.1 Witty Bot

The Witty Bot is composed of five main components. Three of them were not modified at all. The "book room" element that allows the booking of rooms by the bot's users, the "lunch" component that allows bot's users to order lunch and the "who is" element that allows the users to search for people.

Among other things, the "witty main" component manages the bot's main menu. This menu was modified in order to contain the conference related features.

The "conference call manager" was the main focus of the work performed on the bot's side. This component is divided into two main parts, the "schedule" part, and the "look for conferences" part. The "schedule" part already existed on a previous version. However, it had to be done from scratch, in order to support the new functionalities. This module sends requests to the Asterisk server, in order to schedule the calls. When a call is scheduled or modified, it is also saved on the bot's own database (a NoSQL database). When a schedule is performed, it is also necessary to communicate with the LDAP service, in order to identify the conference participants and with the Google Calendar API so it can be scheduled also on the user's own calendar.

The "look for conferences" part performs features related to conference calls that do not involve creation or modification. Tasks like checking if a call is about to start or treating ongoing call's events are done by this module.

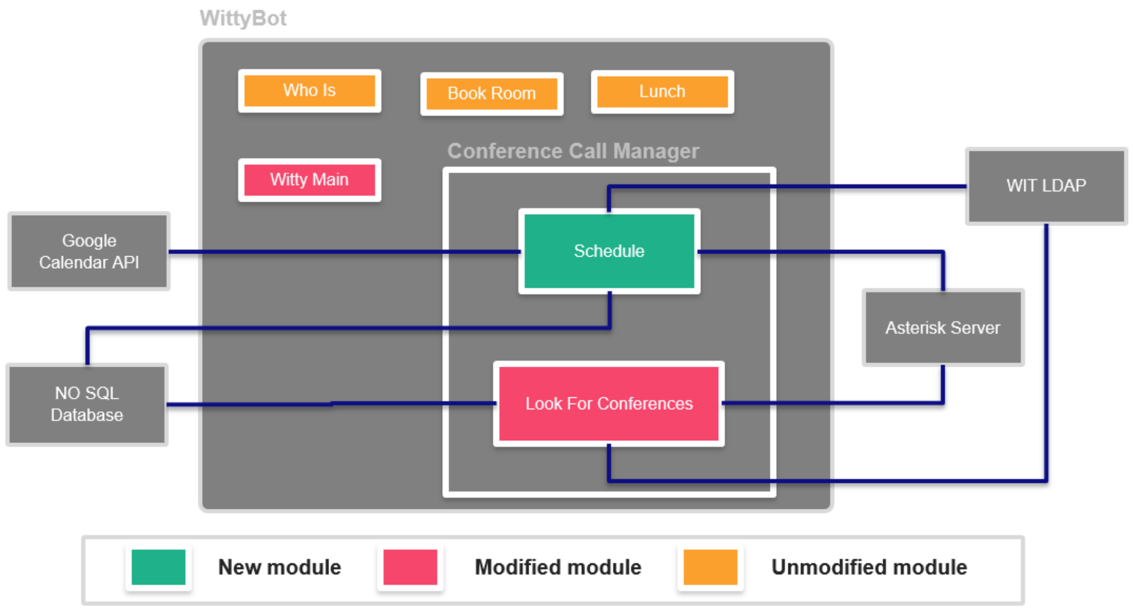


Figure 5-6: Witty Bot modules

5.2.2 Schedule a conference

Figure 5-7 shows the sequence of steps followed to schedule a conference on the bot. Messages sent by the RCS+ app arrive on the RCS Endpoint of the bot. Here, the message is analyzed in order to check if it matches any pattern defined. In the case of the schedule of a conference, it will not match any pattern. In these cases, messages are sent to the DialogFlow platform in order to understand the user's intention. On the platform, uses NLP to extract intentions and elements out of inputs. In the case of the schedule of a message, it will find a date, an hour, a title and phone numbers of participants. The elements found are then sent back to the bot, ready to be treated by the schedule module.

The first thing done by the schedule module is to search the participant's phone numbers, detected in the NLP analysis, on WIT LDAP. After this, besides the phone numbers, there will be the emails and names of the participants.

When all the necessary information for the schedule of the conference is gathered, it can be scheduled. Firstly, it is scheduled on the Asterisk server. After the confirmation of this server, the conference is also scheduled on the bot's database and in the end, in the user's Google Calendar.

After the schedule of the conference call, the schedule module composes a confirmation message to be sent to the user. This message is prepared on the message constructor and is sent to the user by the RCS Endpoint.

If an error occurs while scheduling on the Asterisk server, it will not be scheduled on the remaining places. This approach contributes to increasing the app's reliability, which is one of the defined nonfunctional requirements.

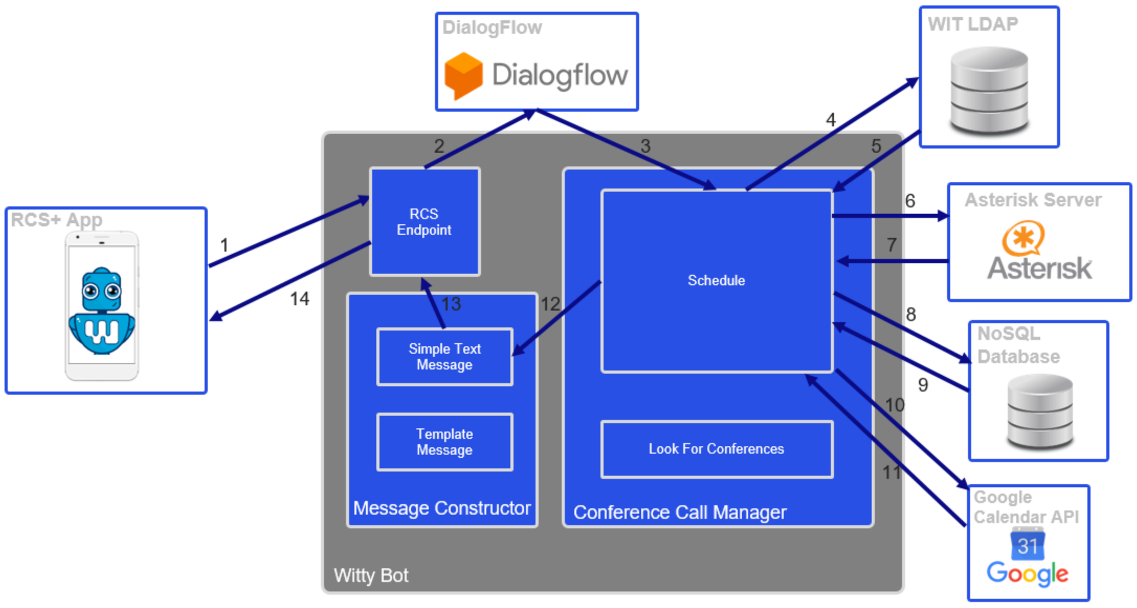


Figure 5-7: Schedule a conference call flow

5.2.3 Calendar Message

The calendar message is used in two situations. It is used to keep the calendar tab updated and to display the user's calendar on a chat message when he asks for it.

In the first case, the request is sent automatically by the app when the "onResume" method of the calendar tab is called.

When the message arrives on the bot, it is analyzed in order to search for pattern matches. In this case, it will match the pattern defined to update the calendar tab. It is sent to the "Look for conferences" module and there, a request is sent to the Google Calendar API. The response will be a JSON message containing the user's calendar events. This JSON cannot be sent directly back to the app, because it contains more information than it is necessary and it does not follow the structure required to the messages that are accepted by the RCS Gateway.

On the message constructor, a new JSON message is defined. For each calendar event, the following fields are added to the message.

- **Title** – This field represents the event's title.
- **Start date** – The day and hour when the event starts
- **End date** – The day and hour when the event ends
- **Participants** – The phone number of the participants of the event
- **Description** – Through the description, the app will be able to identify if the event corresponds to a conference call or not
- **Link** – The link to the event on the Google Calendar service

The JSON message needs to have also a field identifying the message type.
The message will be sent to the app by the RCS endpoint. When the message arrives on the RCS+ app, the chat manager processes it and identifies it as a message to update the calendar tab. A listener is triggered in order to re-direct this message to the calendar tab fragment. In the tab, the events that came on the message are displayed on the calendar view.

The other situation where the calendar view is used is when the user asks to see his calendar on the chat. In this case, when the message arrives on the bot it will not match any pattern defined. It is then analyzed on the DialogFlow platform in order to get the intent. The remaining flow of actions in the bot's side is the same as the one described before for the calendar tab update. Once the RCS+ app receives the message, the chat manager processes it and sends it to the chat history manager where a chat history entry is created. That history entry is identified as a JSON containing calendar events and is then treated on the chatbots module. In this module, a chatbot chat message calendar is created. This message consists on a calendar view, with the events that came on the JSON message. After all this steps, the message is ready to be displayed on the chatbot chat.
Figure 5-8 represents the flow followed when the user asks to see his calendar.

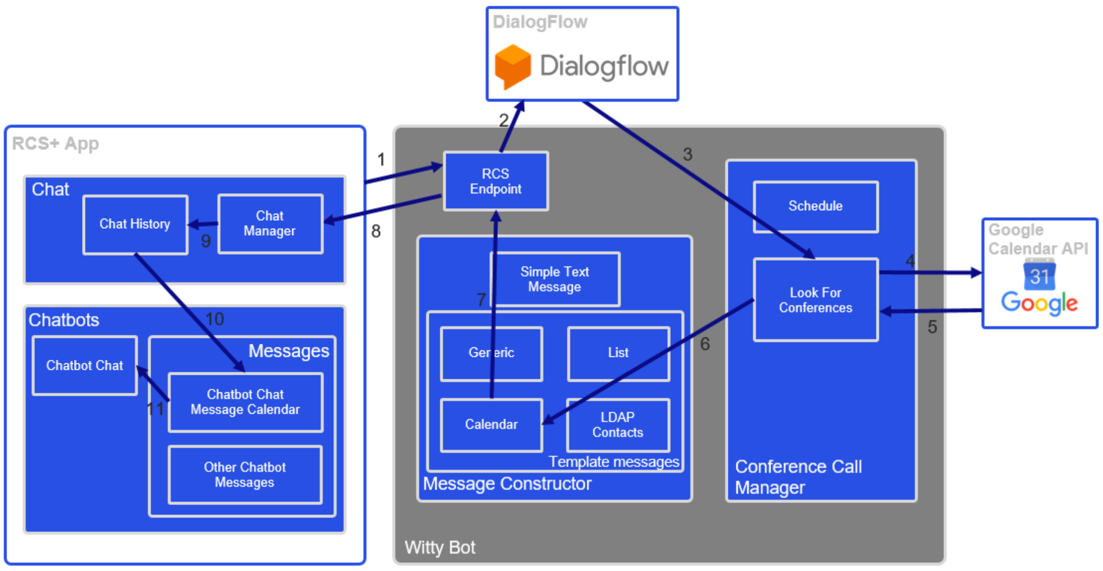


Figure 5-8: Calendar message flow

5.2.4 LDAP Integration

The integration of the WIT's LDAP contacts is triggered when the app starts. A request is sent to the bot in order to start this process. When the message arrives on the bot, it matches a pattern and does not need to be analyzed by the NLP platform. It is then sent a request to the WIT LDAP in order to get the contacts of all the users. The answer received is a JSON message containing all the information about the LDAP contacts. This message is then sent to the message constructor in order to build a JSON template message that is not blocked by the RCS gateway. The JSON to be built contains the following fields.

- **Name** – the name of the contact
- **Phone Number** – the phone number of the contact
- **Email** – the email of the contact
- **Job** – the job of the contact

A field identifying the message type is also added to the JSON message. And then, the RCS Endpoint sends the message to the RCS+ app.

When the message arrives on the app, the chat manager identifies it and passes it to the contacts module. There, the WIT contact manager parses the JSON message, and creates contacts. This contacts will have a field identifying that they represent a WIT worker. This field will allow the app to filter the contacts when only the WIT workers' contacts need to be displayed. After the creation of the contacts, the Android contact manager add them to the app's contact list.

Figure 5-9 shows the flow of this process.

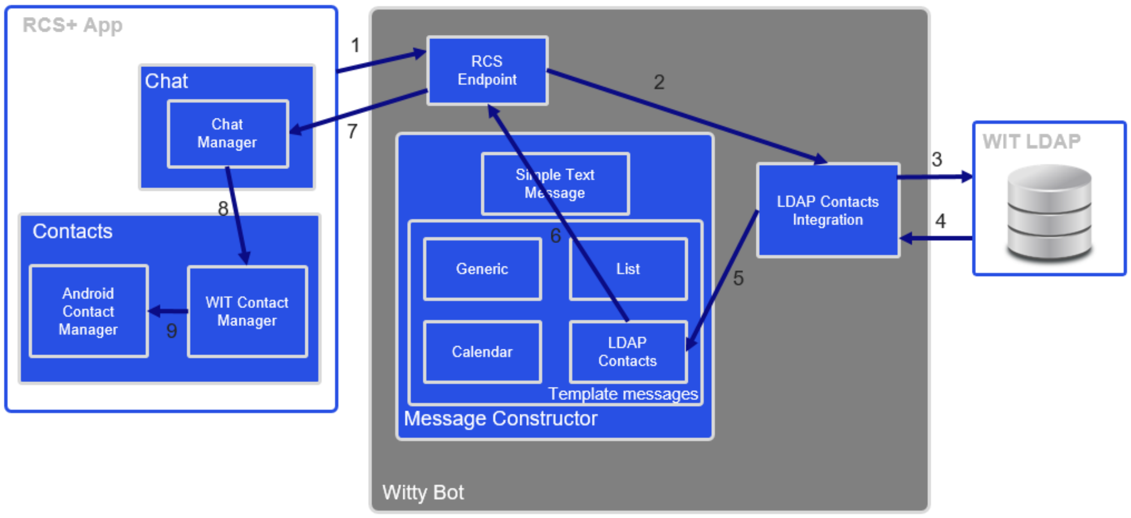


Figure 5-9: LDAP integration flow

5.2.5 Ongoing Call

Unlike the other processes, this one is not triggered by the RCS+ app. It occurs when a message arrives on the bot coming from the Asterisk server. There are five types of messages that can start this process.

- **Start of a conference** – a conference starts, this happens when the first participant joins
- **End of a conference** – a conference ends, this happens when the last participant leaves the conference
- **Participant entrance** – someone joins the conference
- **Participant exit** – someone leaves the conference
- **Participant muted** – someone is muted by the conference owner
- **Participant unmuted** – someone is unmuted by the conference owner

When any of those messages arrives on the bot, it is received on the Asterisk Endpoint. The message will always contain the call id and if it is related to a participant, contains also the participant's id. The message is passed to the Look for conferences module and there, a request is sent to the bot's database, in order to get the calls participants. After that, a text message is broadcasted to all participants.

On the app, the messages are received by the chat manager. Then, two separated paths are followed. The message follows the regular trail to be displayed on the chat window and it is also sent to the listener that redirects it to the ongoing call manager. Once it gets there, it is analyzed in order to determine the type of message, the call's id and, if is the case, the associated contact.

To end this process, the information is passed to the calendar tab, where it is received by the ongoing call expandable list component, and, if it the case, also passed to the ongoing call expandable list child.

When a message represents the start of a conference, the ongoing call component is turned visible, and it is removed when a message represents the end of a conference.

Figure 5-10 represents the flow that is followed when an event of this type happens.

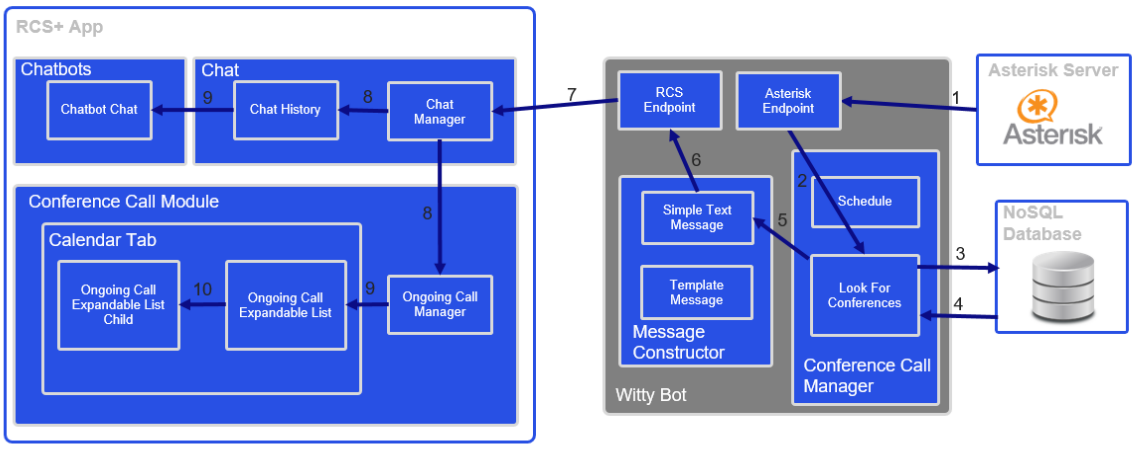


Figure 5-10: Ongoing call events flow

Chapter 6
Implementation

This chapter gives an overview of the implementation done during the internship. During the first semester, the implementation was focused on the call schedule module, which comprises everything related to the scheduling of a conference call.

The second semester started with the conclusion of the work developed during the first semester and proceeded with the development of the app's new calendar tab, followed by the ongoing conference call's features. The last major part developed was the LDAP integration.

The implementation consisted mostly of two very different parts. The development of the screen of the app and the development of the bot's side.

The definition of the screens' layout was an important aspect of this project. Since the work developed was meant to be integrated into an existing app, the new functionalities layout should be coherent with the remaining screens.

Two different approaches were adopted to design the screens. The first one was used in the development of the calendar view.

For this component, the approach was to develop and then keep editing it until its design was coherent with the app.

The remaining screens were defined by the WIT designers' team. This is the standard process used by the company when it comes to layout design definition.

For this to be done, a document must be written with a request, explaining the goals, the constraints and the context of what is needed.

The methodology adopted for the requests consisted on the definition of requirements explaining what the goal to be achieved is, and the definition of simple mockups that represent an idea of what is pretended with the requirement.

The first thing to have in mind when writing a request is that the person who is reading the request is not familiarized with the app. So, the request needs to be very detailed and clear.

During the project, two requests were sent to the designers' team.

The first one was related to the "Schedule call screen" and the "Ongoing call component". The second one was related to images for the bot's main menu.

Both requests and respective answers can be consulted on Appendix B – Requests to Designers.

On the first document, there was a request for the definition of a presenter on the call. When the request was analyzed, the concept of presenter was dropped and the role of "owner" took its place. This happened because the role of presenter was considered not be relevant and there was a lack that would be filled by the creation of the "owner" of the conference. The owner role brought organization to the ongoing conference call. Before the creation of the conference's owner, any participant would be able to execute actions like mute and unmute other participants. This could lead to situations that would not correspond to the desired result.

With an "owner" (the user who creates the conference), the actions are restricted to only one user. This way, the conference environment will be more controlled and people will be focused on what really matters.

6.1 RCS+ App

This section contains the information about the functionalities developed and the way that they work on the RCS+ app's side.

6.1.1 Call Schedule Module

The first major outcome of the intern was the conference call scheduling module. At first, for simplification of the development, this module was developed as a standalone app.
The integration with the RCS+ app occurred when every feature was already developed and working correctly.
This module consists of three parts, the calendar view, the addition of participants and the modification of a conference call's data.

For purposes of reducing failure possibility, internet connection verifications are made when an operation is done. This way, the conference call scheduling is only possible when the device is connected to the internet, avoiding that changes are made locally and not remotely. Either a change is immediately made both locally and remotely or it is not done at all. This choice was made in order to achieve a bigger MTBF which is one of the nonfunctional requirements of the project. And since all the operations are made through the bot, a connection is needed in order to send the messages.

6.1.1.1 Calendar View

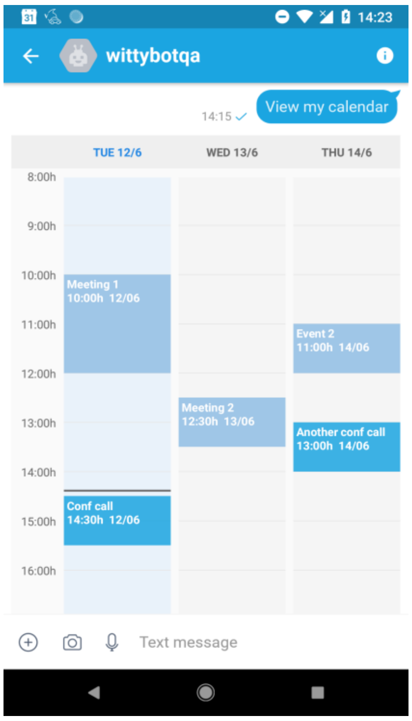
In this sub-module, the user has access to a calendar view of one day, three days or all week. This component is based on an open source calendar view for Android. That calendar view was modified in order to adapt it to the project's context. The following adaptions were done:

- Addition and removal of events;
- Synchronization with Google calendar;
- Disable the possibilities of adding events in the past or on weekends;

This component is shown when a JSON template message containing calendar events is received from the bot. Once that message is received, the calendar view is displayed. On that calendar view, the events that correspond to a conference call are represented by a different color. When an event that is not a conference call is selected, a message is shown saying that the event cannot be opened by the app. On the other hand, when the event is a conference call, it is opened in order to be consulted or edited.

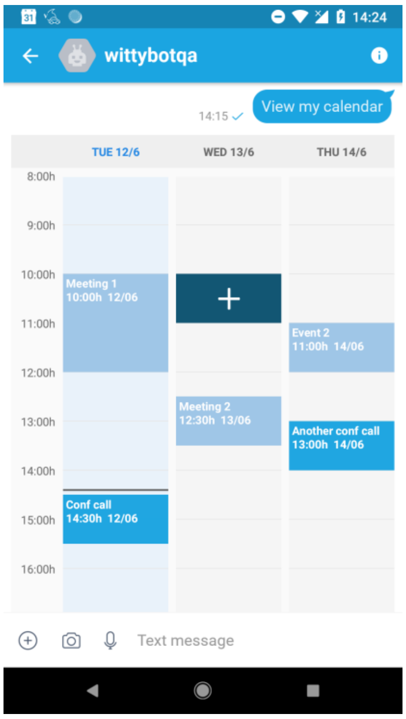
If instead, the goal is to schedule a new conference call, the user will press an empty slot. Once that happens, a temporary icon is created on the pressed slot. In order to schedule, the selection must be confirmed, by pressing the temporary icon.

On the calendar, only events that represent conference calls can be opened and edited. In order to know which events are conference calls, an extra field is added to conferences when they are scheduled. That field will indicate that the event is a conference call and it will also contain the conference's id. It is necessary to save the conference's id in order to perform modifications. Events are displayed on the calendar with two different colors. The two colors allow the user to identify which events are conference calls. If the user tries to open an event that is not a conference call, a message is shown saying that the event cannot be opened by the app.



View my calendar

Figure 6-1: Calendar view



View my calendar

Figure 6-2: Calendar view with add event icon

6.1.1.2 Call Schedule

Once a calendar slot is picked, a new screen is presented to the user. This new screen contains information about the conference.

In order to keep the design coherent with the remaining screens of the app, this screen was based on a mockup made by the WIT designer's team.

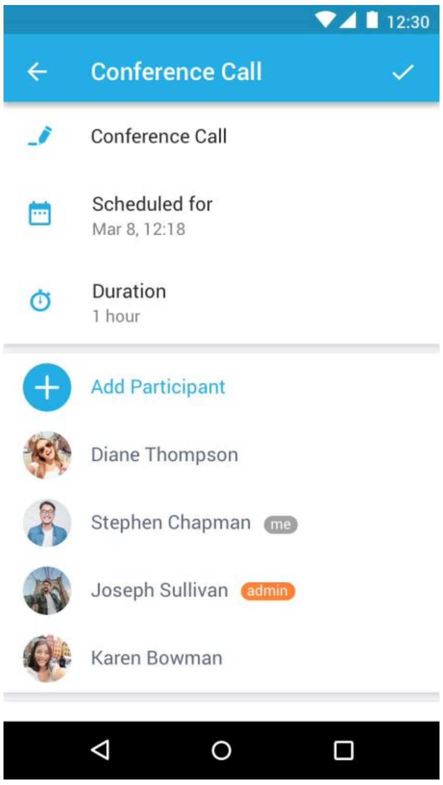


Figure 6-3: Schedule call screen mockup

If the slot picked by the user was empty, the intention is to schedule a new conference call. So, the date related field is pre-filled with the date corresponding to the calendar slot picked. There is also a field for the conference's title and one for its participants. There is an option to add participants and one to cancel the call. In this case, this last option will be disabled, because the user is creating a new conference.

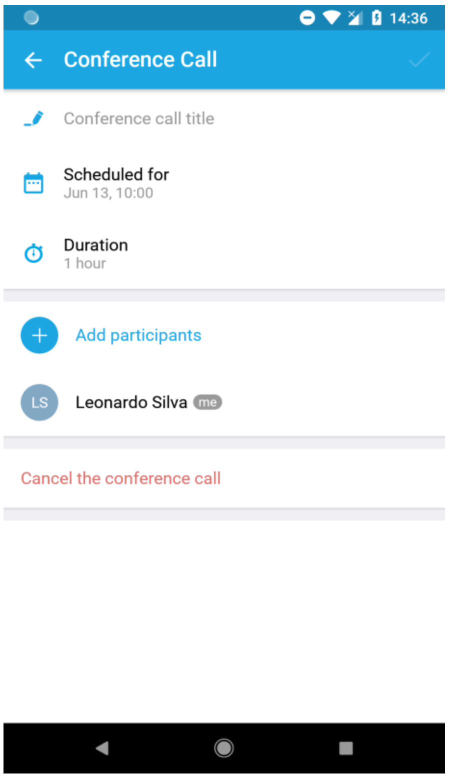


Figure 6-4: Create conference call screen

On the other hand, if the slot picked contains a conference call, this screen will provide the possibility to edit it. Current title, date, duration and participants of the call are shown on this screen. And, since the call already exists, the option to cancel it is available.

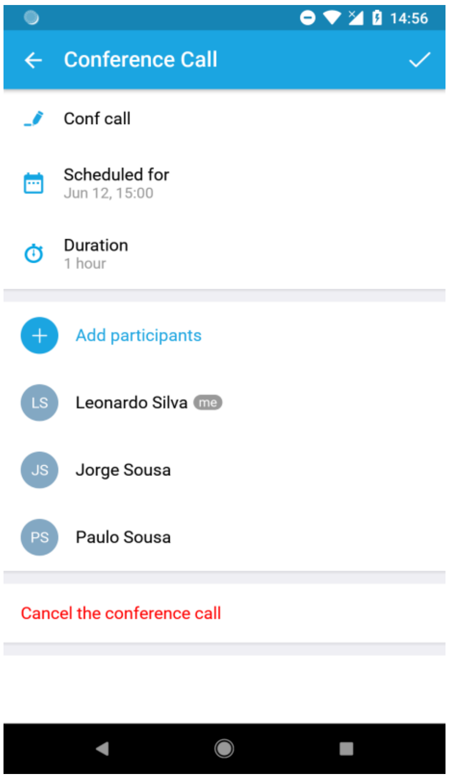


Figure 6-5: Edit conference call screen

Once the user is satisfied with the changes made, he can select the "Done" options on the right top corner of the screen and a message will be sent to the bot containing all the information related to the changes made by the user.

6.1.1.3 Participants Addition

The addition of participants to a conference call is part of the scheduling process. This was done, in conformity with the contact pickers of the RCS+ app, for purposes of design consistency.

On this participant picker, only contacts that work at WIT are displayed. This filter was made based on the LDAP integration that is described in section 6.1.4.

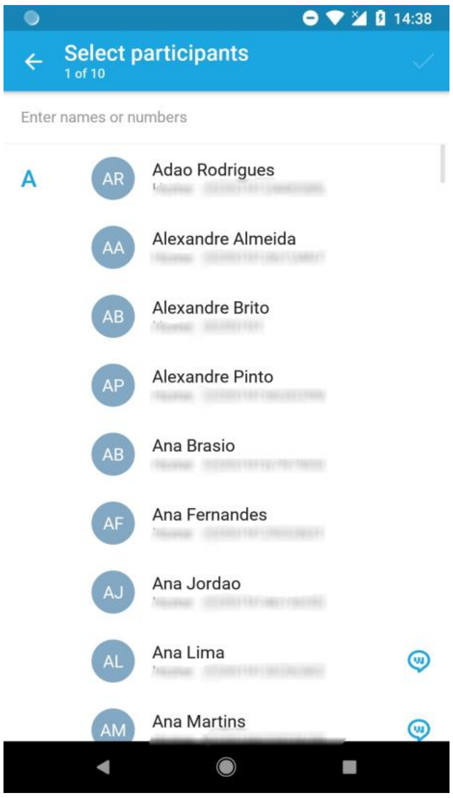


Figure 6-6: Participants Selection 1

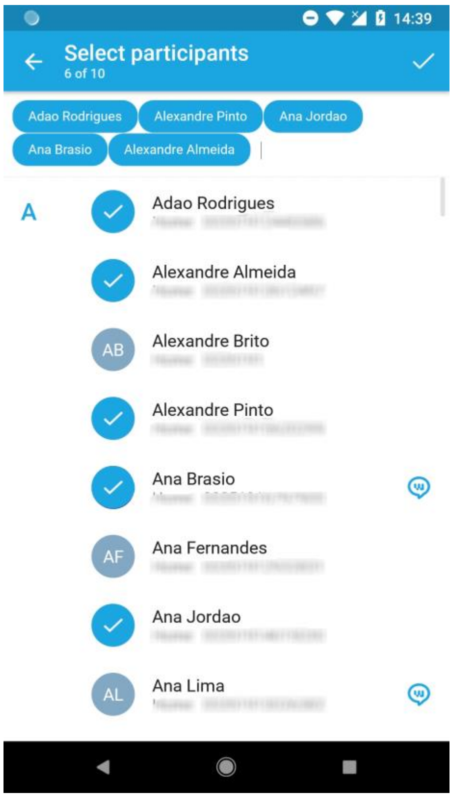


Figure 6-7: Participants Selection 2

6.1.1.4 Participants Removal

The removal of conference call's participants a feature of the scheduling process. The process of removal was based on a mockup defined by the WIT designer's team. It is possible to remove one or more participants at a time. Figures 6-8 and 6-9 show the mockup and the final implementation.

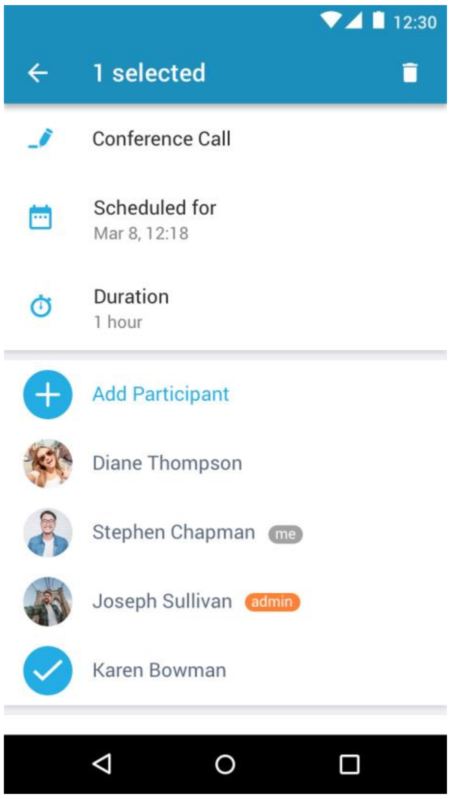


Figure 6-8: Participants removal (mockup)

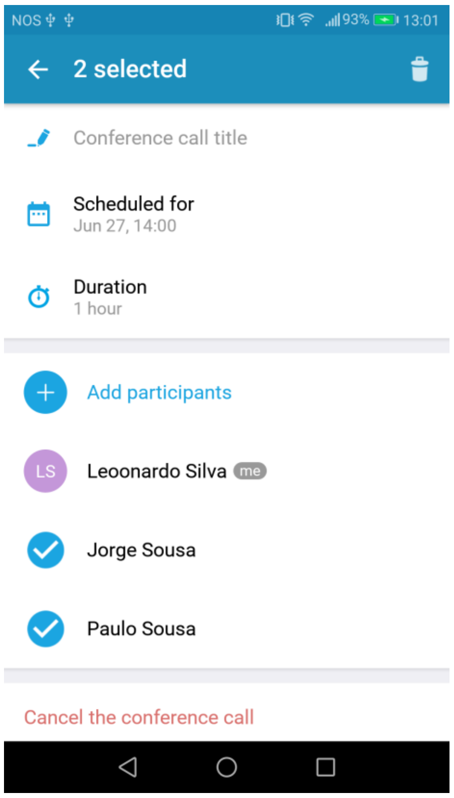


Figure 6-9: Participants removal

6.1.2 During Call Module

The During Call module allows the user to have access to an ongoing conference call's events.

During a conference call, a user should be able to see who is participating in a conference call at a given moment. He should also see the duration of the conference. If the user is the owner of the conference, he should also be able to mute, unmute and remove active participants from the call.

For this module, an expandable component was added to the top of the screen. When this component is collapsed, it shows the number of active participants and the call's duration. The design of this component was based on the mockup (figure 6-10) created by the WIT designers' team. Figure 6-11 represents the final result on the collapsed component.

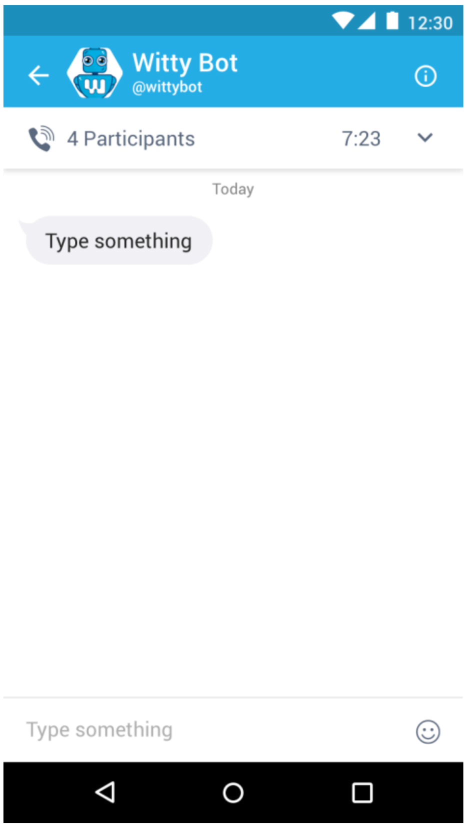


Figure 6-10: Ongoing conference call component collapsed (mockup)

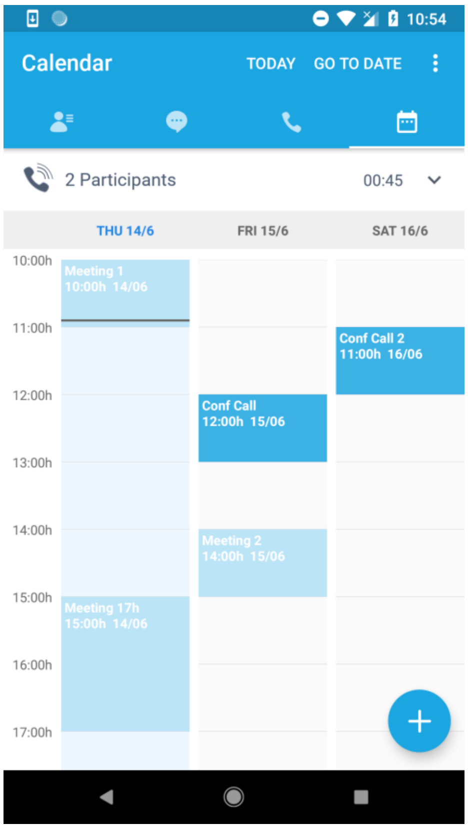


Figure 6-11: Ongoing conference call component collapsed

Once the component is expanded the user is able to see who the active participants are.

There are two labels on the component, the "me" label, that marks the user itself and the "owner" label. If the "me" and the "owner" label are on the same user, that user will be able to click the "remove participant icon" in order to remove a user from the call, and can also click the "mute/unmute icon" in order to mute or unmute a user, depending on its current state. Once again, the element was based on the mockup shown in figure 6-12. The final result of the expanded element of the conference's owner can be seen in figure 6-13.

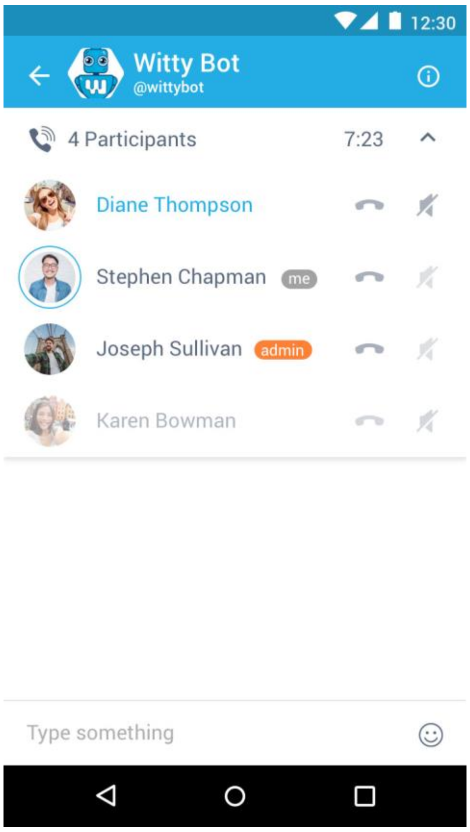


Figure 6-12: Ongoing conference call component expanded (owner view mockup)

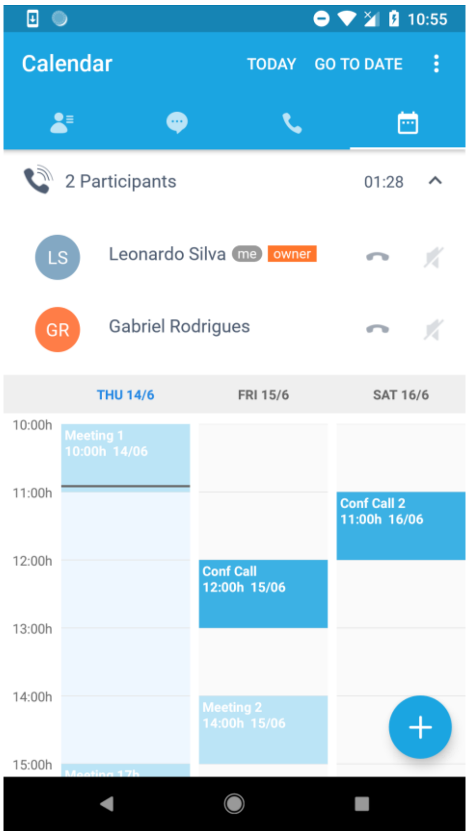


Figure 6-13: Ongoing conference call component expanded (owner view)

If the labels are on different participants, it means that the user is not the owner, and he will be only able to see the mute/unmute state of the participants. Both the mockup and the final result can be seen in figures 6-14 and 6-15 respectively.

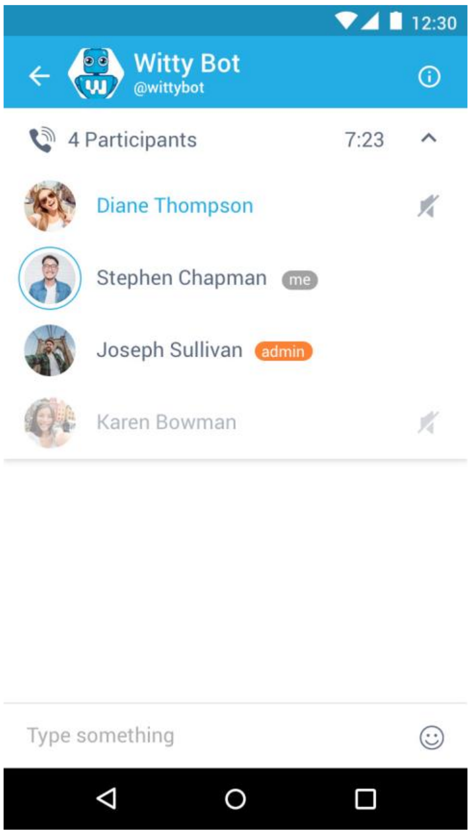


Figure 6-14: Ongoing conference call component expanded (normal view mockup)

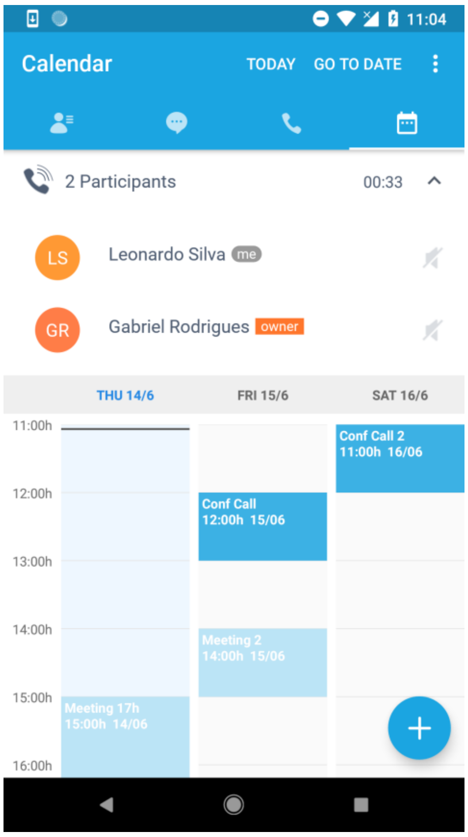


Figure 6-15: Ongoing conference call component expanded (normal view)

A participant leaves the call when he hangs up the phone or when he is kicked by the conference's owner. When a participant leaves the conference, the ongoing call component must be updated. In order to make it clear for the user that someone left the conference, the participant's information is not removed right away when he leaves. The number of participants on the top of the bar is updated right away, but the participant's field stays for five more seconds with its transparency changed. Figure 6-16 shows the screen on the moment between the exit of the participant and the field removal.

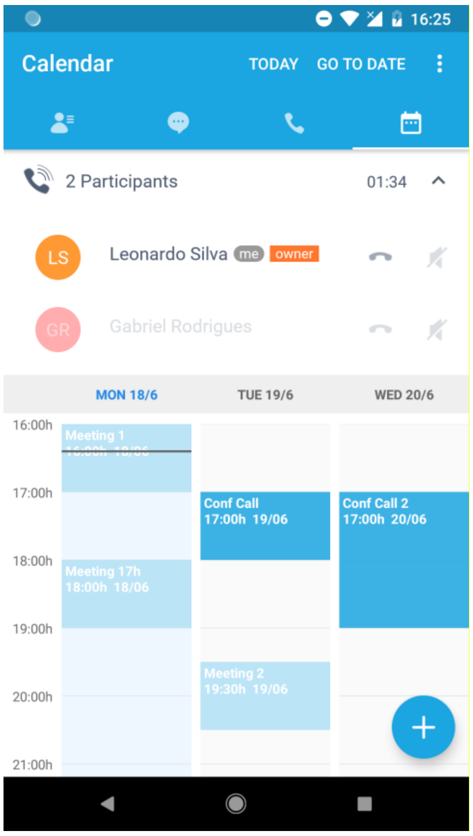


Figure 6-16: Ongoing call component - Participant left conference

At the beginning of the internship, one of the requirements defined for this module was to show when a user is speaking. During the development phase, it turned clear that this feature could not be implemented. The conference's events are sent by the bot. And the delivery time of the bot's messages on the app may vary a lot. In order to be well functioning, this feature would require a very small delivery time. As this delivery time cannot be guaranteed, this feature was left behind.

6.1.3 Conference Call's Tab

This feature consisted on the addition of a new tab to the RCS+ app's main menu. This new tab allows the user to view its calendar without having to ask the bot for it. And also allows the user to create new conferences or edit the existing ones.

This tab has all the functionalities of the calendar view used on chat messages and the possibility to create a new call by pressing the floating action button on the screen's bottom right corner. This way of creating a conference will open the schedule screen that, by default will be filled with the next available time slot for a conference call.

In order to keep this calendar view updated, a request is sent to the bot every time the tab is selected. A JSON template is sent back by the bot containing all the events on the user's calendar.
For consistency purposes, an icon for the tab was developed by the WIT designers' team.

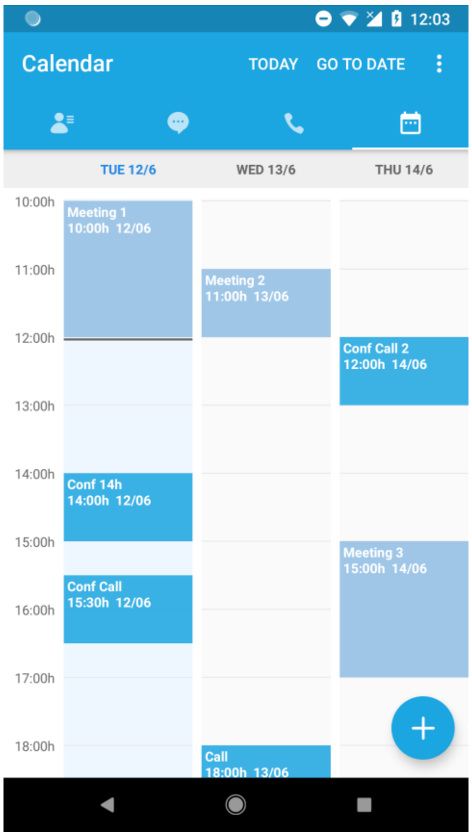


Figure 6-17: Conference Call's tab

6.1.4 LDAP Integration

This feature consists on the integration of the contacts of the WIT's workers with the RCS+ app. Although it has an impact on all the functionalities developed, it was the last major feature to be developed.

Once the app is launched, a request is sent to the bot asking for the LDAP integration. The response comes as a JSON template, containing information about all of the WIT's workers. All this contact are added to the app's contact list.

The app's contact list is created every time that the app is launched. It reads the phone's contact list and adds every contact. The contact list of the app is not stored anywhere, so, the list is created every time when the app is launched. That is why the message asking for the integration has to be always sent when the app starts.

The contacts that come from the bot's response, are added with the "organization" field saying "WIT Software". On the contact's tab, a filter was created, in order to be possible to display only the WIT's workers contacts

When a conference is being scheduled, the same filter is applied to the participant picker. This way, only WIT's workers' contacts can be added to a conference call.

This feature has also impact on the ongoing conference call's top bar. For example, when a user joins a conference call, a message is received on the app with the person's phone number. The LDAP integration makes it possible to associate every phone number received during a conference call with a contact.

6.1.5 Message treatment

Some of the messages received on the app need to have a different treatment than the regular ones. These messages are the JSON templates.

The template containing the user's calendar needs to be displayed as a graphical component. When it arrives, the app needs to recognize that this message is a template and that it is not to be displayed in a textual way. Then, a "ChatbotChatMessageCalendar" object is created and the properties of the message are attributed to the object. Then, this object is displayed on the bot's chat screen.

On the other hand, the template containing the LDAP contact's information does not need to be displayed either in a textual way neither graphically. The app needs to recognize that it is a contacts list template, and create "Contact" objects with the information on the template, add those contacts to the app's contact list and then, don't display the message at all.

6.2 Bot

This section contains the information about the work developed on the bot's side.
On the bot, every functionality starts with the definition of a BPMN diagram. This diagram defines the flow that the conversation will follow. Figure 6-18 shows one simple BPMN flow, representing the update of the events of the calendar tab. All the BPMN diagrams developed can be consulted on *Appendix C – BPMN flows*.

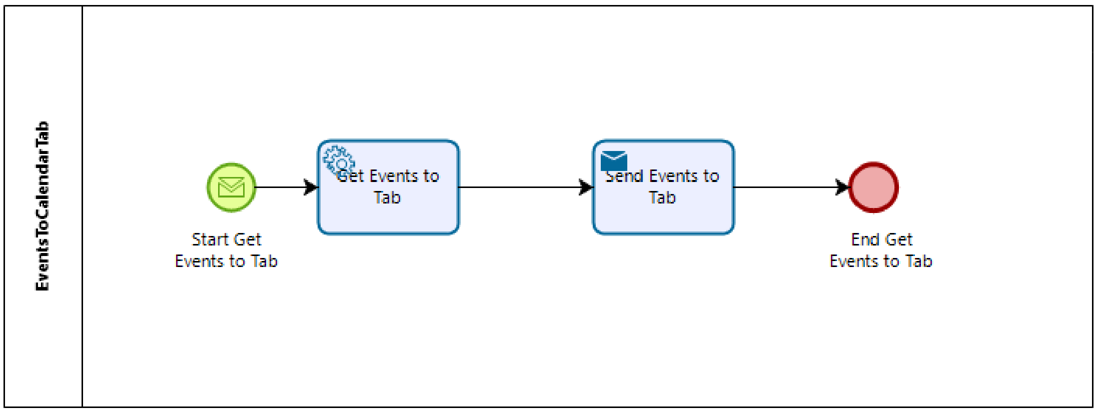


Figure 6-18: Update calendar tab's events - BPMN diagram

This BPMN flow is triggered when a request arrives on the bot matching the pattern defined to the "Start Get Events to Tab" event. From there, the flow follows to the "Get Events to Tab". This is a *service task*. This kind of task is used to perform "backend" processes. In this example, this task sends a request to the Google Calendar service to get the user's calendar events and waits for its response. Once the response arrives, the task builds a JSON object with the events and the finishes. The flow continues with the "Send Events to Tab" task, which is a *send task*. This type of task sends answers to the users. In this case, the task will send the JSON object built by the previous task. The flow ends with the "End Get Events to Tab" event, an *end event*.

Once the diagram is done, the remaining needed files can be generated based on it. It is necessary to generate an AIML file and a JavaScript file.
The AIML can be modified in order to define templates for the bot's answers and associate them with user inputs.
The JavaScript is used to define what is done in each BPMN task.

6.2.1 Conference Call Schedule

On the bots side, the schedule module is the most complex module developed.

On the BPMN file of this feature, there are several "start events" that can be triggered in order to start one flow.

The main "start event" is the "Start schedule". This start event is triggered when the user sends a message asking to schedule a call. The input is analyzed by the platform DialogFlow, to check if it corresponds to request to schedule a conference call. If it does, this start event is triggered. If the request already contains a valid date and an hour, the bot will send a message to the user with the information that was contained in the request. The user will then have the opportunity to add participants and a title to the call.

If the request contains a date but it is not valid, a message will be sent saying that it is not possible to schedule a conference on the date mentioned. If it contains a valid date and an invalid hour, it will be necessary to check if it is possible to suggest an alternative hour for the conference. If it is, a message will be sent to the user saying that it was not possible to schedule a conference on the hour indicated, but it is possible to schedule to the next hour available. If the user accepts the alternative hour, the schedule will continue from then, if he does not, the schedule is aborted.

If the request does not contain a date and an hour, the user is asked if he wants to check his calendar. If he does, a JSON template message is sent with the user's calendar events. If he does not, the schedule will have by default the next available time slot.

Another "start event" is the "Show My Calendar". The input is also analyzed by the DialogFlow platform and if it contains an intent to see the calendar, it is triggered. This start event will lead to the sent of a JSON template message with the user's calendar events.

The events described are the ones that lead the user to the call schedule screen. Once the "done" button on that screen is pressed, a request is sent to the bot containing the details of the conference to be scheduled.

The bot starts by trying to schedule the call on the asterisk server (the server that manages the WIT's conference calls) if it succeeds, the bots proceeds to save the call on its database. To finish the process, if both previous steps were completed with success, the bot saves the call on the user's Google Calendar. If any of the steps fail, the user will receive a message saying that it was not possible to schedule the call.

There are two other "start events" on this BPMN file. One to cancel conference calls and another to edit them. Both of them are triggered by a pattern. If the request sent by the app matches the pattern defined for the start events, the cancelation or the edition starts.

As an extra, it was added support to schedule messages through the Facebook messenger's platform. The request that comes from the messenger platform cannot be processed in the same way that the RCS+ app ones are. For instance, it is not possible to show the calendar view on the Facebook messenger's platform. Some adaptations were made in order to make it functional. On the Facebook messenger's platform, all the interactions had to textual in this case.

6.2.2 Ongoing Conference Call

Although the majority of the bot's start events are triggered by user's inputs or messages that are received, there are some that are triggered on a time basis. An example of that is the events that check if the user has a conference call about to start.

Every five minutes, the bot checks on its database if there any conference call about to start. If there is, a message is sent to the user. This message is sent with three quick replies that the user can select. The first is to join the call. If it is selected, the bot sends a request to the asterisk server and the user will receive a call to join the conference. Another quick reply available is to tell that the user is late for the call. The bot will send a message to the conference's participants telling that the user will be late, if this reply is selected. The last quick reply is to say that the user cannot participate in the call. When it is selected, the bot sends a message to the conference's participants saying that he cannot participate on the call.

The asterisk server sends a message to the bot every time an event happens on the call.

When the call starts, the bot receives a message from the asterisk server, and it multicasts the message to all the participants. The same happens every time a new user joins or leaves the call.

On the RCS+ app, the owner of the conference can ask to mute, unmute or remove a participant. When the bot receives any of those requests. The bot parses this information and sends a request to the asterisk server. Once the confirmation is received, the message is sent to all the conference's participants.

6.2.3 Events to RCS+ Calendar Tab

The "start event" of this flow is triggered when the bot receives a message that matches a certain pattern. That pattern is sent by the RCS+ app every time the calendar tab is selected.

Once the message is received, the bot will get the events from the user's Google Calendar. The user needs to have granted access to its calendar.

Once the bot gets the events, it builds a JSON template with them. Not all the information received from the Google Calendar's API is used. The JSON will only contain the event id, the event title, start and end timestamp and time zone, the participants and the description.

The JSON template is then sent to the user, in order to be parsed and displayed on the calendar view.

6.2.4 LDAP Contacts Integration

Like the previous one, this flow is triggered when the bot receives a message matching a certain pattern. The message that matches the pattern is sent by the RCS+ app every time that the app is launched.

When the message is received, the bot sends a request to LDAP server, asking for the information of all the contacts.

Once the answer is received, a JSON template is built with the name, phone number, e-mail and job of all the contacts. Then, it is sent to the user. Once it is received on the RCS+ app, it is parsed and the contacts are added to the app's contact list.

6.2.5 JSON Templates Constructor

The bot can send various types of messages. The most common are text messages. There are also quick replies, which allow the user to choose from a set of options the answer that he pretends to give. And there are template messages, which are JSON objects.

The JSON templates allow the bot to send information that will be displayed in a different way by the receiver (most of the times it will compose graphical component).

For this project, two new template types were created. The first one was created to send the user's calendar events. The other allows the bot to send the LDAP's contacts. Both of the templates can be adapted in order to send other types of content, although they are optimized for this kind of information.

6.2.6 Other aspects

In order to achieve the goals proposed on the non-functional requirements, some particularities were implemented. The first requirement says that the data stored on the Asterisk server, on the NoSQL database and on the user's Google calendar must be consistent.

To do this, all the schedules, modifications and cancelations are performed in a transactional way. Either a change is performed in all 3 platforms or it is not performed in any of them. This aspect is also important for the second requirement because it reduces the possibility of the user to receive a positive feedback when some error occurred performing the action.

Also to increase the system's reliability, the verifications for conferences that are about to start are performed every minute. A conference call is considered to be "about to start" when there are less than ten minutes left to its start. Doing this verification every minute, even if a connection error occurs during a verification, more verifications will be performed and the chances to have at least one that is successful is bigger.

Chapter 7
Validation & Verification

Quality assurance is a very important phase of a project.
This stage allows the developer to **validate** whether the product satisfies the requirements that led to its development. The validation of the product, in this case was done by the internship's supervisor, as he represents the client of the product, deciding if each feature is implemented in the way that serves best the required functionalities.
Besides validation, this phase is also composed by the **verification** of the developed product. Verification consists on the evaluation of the developed features in order to check whether they satisfy the functional requirements and to verify if there are any errors.
This phase can be done either during the software development or at its end. In this case, the validation and verification was a continuous process. In order to validate the developed functionalities, regular meetings were scheduled with the internship's supervisors. For the verification, several tests were made during the development of each feature. At the end of the development, a more formal verification took place. That verification started with the definition and evaluation of the functional tests.

7.1 Functional Tests

Functional tests are a type o black-box testing. This kind of tests are based on the project's specifications and with no knowledge of the implemented code.

For each test, there is an input and an expected output. If the final result is equal to the expected result, the test is considered to be passed with success. Otherwise, there is something that needs to be corrected.

Each functional test is composed by an id, a sequence of steps to perform in order to make the test, a description of the expected result and a result.

Table 7-1 represents an example of a functional test. This example consists on the cancelation of a conference call.

FT39	**Cancel a conference call**
Steps	• Select a conference call on the calendar view • Select the "Cancel conference call" field
Expected Result	○ The app sends a message to the bot saying to cancel the call ○ The call is deleted from the bot's database ○ The call is removed from the user's google calendar ○ User receives a message saying that the call was successfully canceled
Result	**SUCCESS**

Table 7-1: Functional test 39 - Cancel a conference call

In order to test the various features developed, 71 different tests were defined, distributed over six categories. Appendix D – Functional Tests, contains the complete list of tests and its details.

After the development of each major module, a test phase dedicated to the features related with that module occurred. During those test phases, some tests have failed and modifications were made in order to correct the errors found. In the final stage of the internship, a new verification phase took place, covering all the modules. Table 7-2 contains the results of this final tests.

Category	Success	Failed
Schedule Calls	43	0
Call About to Start	4	0
Ongoing Call	12	0
LDAP Integration	2	0
Calendar Tab	7	0
General	3	0
TOTAL	71	0

Table 7-2: Functional Tests results

7.2 Calendar View Tests

One of the tasks performed during the app's usage is reading the user's Google Calendar events and present them on the app's calendar view. On the calendar view, only the events that represent conference calls can be opened.

In order to test this feature, several calendar variants were used. For each test, the calendar would vary the number of events, the number of events that correspond to conference calls or not and the possibility to have simultaneous events.

For each calendar, the following tests were performed:

- Select an empty slot to schedule a conference
- Select an event that is not a conference call
- Select an event that is a conference call
- Perform a modification on a conference call

Not all the tests are applicable to all the calendar distribution used. For example, in the case where the calendar does not have any event, it is not possible to select events or make modifications on a conference.

With this test, it is possible to verify if all the events are identified with the correct type (conference call or not) and to check if all conference calls are being identified with the correct id (which is essential to perform modifications on a scheduled conference). It allows also to verify the behavior of the calendar with different loads.

If all the tests are well succeeded, the result is considered to be correct (✓), otherwise, the test failed (✗).

Number of events	Contains events at the same time	Number of events that are conference calls	Number of events that are not conference calls	Tests results
0	No	0	0	✓
1	No	0	1	✓
1	No	1	0	✓
5	No	5	0	✓
5	No	0	5	✓
5	Yes	1	4	✓
10	No	4	6	✓
10	Yes	5	5	✓
20	No	2	18	✓
20	Yes	8	12	✓
40	Yes	8	32	✓
100	Yes	15	75	✓

Table 7-3: Calendar view tests

7.3 Usability Tests

Usability tests are particularly important on this project. It is vital for the project that the features are simple, easy to use and improve the velocity of execution of the tasks. So, it is important to evaluate how the users perform the tasks on the app.

For these tests, some users were asked to perform certain tasks. The goal of this task is to find problems in the app's layout and to find out if the app's interface is intuitive enough.

Five users were selected to perform the usability tests. This number of users was chosen taking into account Jakob Nielsen's research[31]. According to Nielsen, there is no need to use more than five users to perform usability tests. The use of more users does not compensate. Everything that the first user does will be new. The second user will perform some of the things that the first user did, and some new things. The third will do some things that the first user did, some things that the second user did and some new things. Each user will perform less and less new things. With 5 users, typically, 80% of the UI problems are detected. The suggestion coming from the article says that instead of performing usability tests with more users, it pays off to perform more usability tests with fewer users.

For each of the following Use Cases, a usability test was performed.

- Schedule a conference call
- Join a call as conference "owner" and mute a participant

For each use case, the number of steps that the user took to complete the task was measured.

Schedule a conference call

For this use case, the app is given to the user right after start, and it is asked to him to schedule a conference call "today".

Minimum required steps:

- Go to calendar tab
- Select a slot
- Confirm the selection
- Give a title to the call
- Select the "Add participants" button
- Add a participant
- Select "Done" button

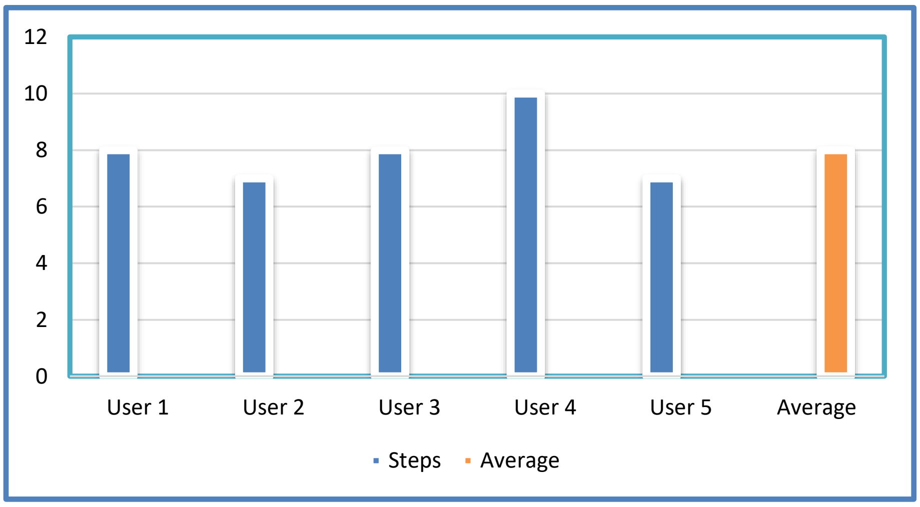


Figure 7-1: Schedule a conference - Usability test results

The results of this use case are considered good. For the completion of the task, there was a minimum of 7 required steps. Even though there was a user who performed 10 steps (3 more than the minimum), the average of completion was 8 steps, an excellent result.

Join a call as conference "owner" and mute a participant

This use case is started with a call to the user's phone. That call will allow him to join the conference.

Minimum required steps:

- Accept the call
- Go to the ongoing call component
- Expand component
- Select the "mute button" of a participant

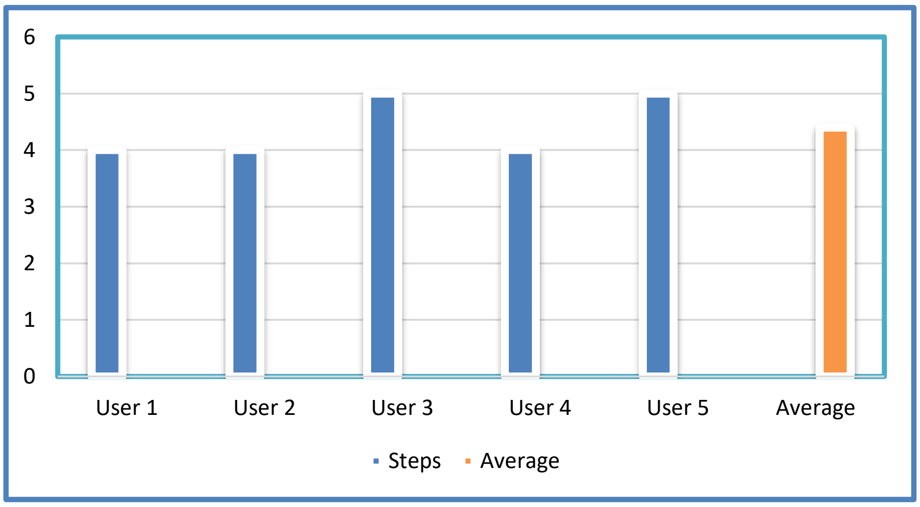


Figure 7-2: Join call and mute participant - Usability test results

The results of this use case are very good. There was a minimum of 4 required steps to complete the task, and the average number of steps needed by the users was 4.4. This indicates that the UI on this use case is explicit and intuitive.

Chapter 8
Conclusion

This last section contains an overview of the work developed during the internship, some considerations regarding the project's success and possible future work and finally, a personal analysis of the internship.

8.1 Overview

The main goal of this internship was to develop features that could use bots in order to be valuable in an enterprise context. Since that topic was far too broad, there was a need to reduce the range and the internship focused on features related to conference calls.

WIT had already a very complete communications app that allowed the use of chatbots. So it turned evident that it was a good choice to use that app as a base and add value to it.

The first challenge of the internship came with the fact that this was a project to be integrated with an app that already existed. So, a study of the app was required in order to understand how it was structured and how could the new functionalities be incorporated in that structure.

The study of the state of the art allowed to see what the best functionalities present in the competitors were, both in terms of enterprise apps that comprise conference calls related features and bots that have also conference call related capabilities.

After that analysis, the decision of what features would be implemented and what features would be adaptable or not to the context of the project took place.

With the definition of the functionalities to be developed concluded, the requirements elicitation followed. The definition of the requirements was a long process, initiated with the writing of User Stories that later gave place to Functional and Non Functional Requirements.

Based on those requirements, an architecture for the solution was designed. Until the end, the output of this phase suffered some modifications, but, in general, the initially designed architecture does not differ much from the last versions.

The implementation of the features occurred mostly during the second semester and was always complemented with phases of validation and verification. To assure the validity of the work done, several meetings with the internship's tutors were held. The feedback from those meetings was always taken into account and some changes arose from there. The verification was also always performed in order to check if the developed features were working as suppose.

In the final stage of the project, a new verification phase took place, covering all the work developed during the internship.

8.2 Success Evaluation

The threshold of success is one of the success criteria defined. It said that the project was considered to be successful if all the "Must Have" requirements were completed by the end of the project. Not only the "Must Have" requirements were completed, but all the requirements defined in table 4-2 were. So, the threshold of success was achieved, and, therefore, from this point of view, the project was well-succeeded.

This allows saying that from a perspective of completion, the project was succeeded. But there are other aspects in which the project's success can be evaluated.

If the developed features have anomalies that affect the use of them, the project cannot be considered successful. So, another metric that can be used to test the success of the project is the percentage of tests that were well succeeded. For the project, 71 functional tests were defined. In this set of 71, there are tests related to all the developed features, trying to cover all the project's scope. The tests revealed a success rate of 100%, being, this way, an indicator of the project's success.

8.3 Future Work

Regarding what was defined for this project, there is not much left to do. All the requirements were implemented and the results match the proposed goals. Although, during the state of the art analysis, there were some interesting features present on some competitors that were not adopted to the project due to the fact that they were either out of the internship's scope or there were limitations. Examples of this are the voice control support and the indication of who is speaking in a given moment. The voice control support would be a nice feature to have, because, one of the main goals of the project was to make tasks easier and quicker. That was achieved with textual interaction and visual components, but if voice control support was added, the results would be even more satisfactory. The "Who is speaking" was also a nice feature to have on the project, but, due to the limitations of the response time of the bot, it was not implemented. In the future, it would be nice to find a workaround for this problem and implement this feature.

8.4 Final Considerations

This internship allowed me to grow both personally and professionally. This was without any doubt, the biggest challenge that I have ever faced and I am glad to say that the goals that I have established at the beginning of the project were achieved. It is a great feeling to start a project from scratch, pass through all the development phases and finish with a complete product.

I had the opportunity to put in practice many of the things that I have learned throughout the years of study and I have had also the chance to deepen my knowledge in many technologies and to learn some others, and I am sure that this experience will contribute to my future as a software engineer.

The great environment of the company, and in particular, the team that welcomed me allowed me to face every day with joy and will to give my best to this project.

The importance of the organization and documentation of a project was also something that proved to be essential. That is something that I had already learned from several teachers but in this project, I had the opportunity to experience it. The adaptation to the RCS+ app was greatly facilitated by the well commented code and the organization of the project. On such a big application, it was easy to let it turn into a confusion of files and functionalities. But, the fact that it was well organized and documented allowed me to get my way around it easily. And it inspired me also to follow the same line with my code.

Generally, this internship was a great experience and I am sure it will allow me to be a better person and professional.

References

1. WIT Software | Witness the difference. (n.d.). https://www.wit-software.com/ (accessed January 10, 2018).

2. PORDATA - Produtividade do trabalho, por hora de trabalho (UE28=100). (n.d.). https://www.pordata.pt/Europa/Produtividade+do+trabalho++por+hora+de+trabalho+(UE28+100)-1992 (accessed January 7, 2018).

3. Eurostat - Tables, Graphs and Maps Interface (TGM) table. (n.d.). http://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&language=en&pcode=tps00071&plugin=1 (accessed January 9, 2018).

4. R. Epstein, The Quest for the Thinking Computer. *AI Magazine*, **13** (1992) 81. https://doi.org/10.1609/AIMAG.V13I2.993.

5. J. ~veizenba Um, ELIZA A Computer Program For the Study of Natural Language Communication Between Man And Machine. (n.d.).

6. Dialogflow. (n.d.). https://dialogflow.com/.

7. Skype | Ferramenta de comunicação para chamadas e conversas gratuitas. (n.d.). https://www.skype.com/ (accessed June 22, 2018).

8. Messenger. (n.d.). https://www.messenger.com/ (accessed January 19, 2018).

9. GSMA. (n.d.). https://www.gsma.com/.

10. Workplace do Facebook - Muda a forma como trabalhas. (n.d.). https://www.facebook.com/workplace (accessed January 8, 2018).

11. Workplace by Facebook – Aplicações Android no Google Play. (n.d.). https://play.google.com/store/apps/details?id=com.facebook.work (accessed January 8, 2018).

12. Skype para Empresas. (n.d.). https://www.skype.com/pt/business/ (accessed January 8, 2018).

13. Skype for Business for Android – Aplicações Android no Google Play. (n.d.). https://play.google.com/store/apps/details?id=com.microsoft.office.lync15 (accessed January 8, 2018).

14. Cisco Webex | Cisco Spark is now Cisco Webex Teams. (n.d.). https://www.webex.com/products/teams/index.html (accessed June 18, 2018).

15. Cisco Webex Teams (ex-Cisco Spark) – Aplicações no Google Play. (n.d.). https://play.google.com/store/apps/details?id=com.cisco.wx2.android (accessed June 18, 2018).

16. Video Conferencing, Web Conferencing, Webinars, Screen Sharing - Zoom. (n.d.). https://zoom.us/ (accessed January 8, 2018).

17. ZOOM Cloud Meetings – Aplicações Android no Google Play. (n.d.). https://play.google.com/store/apps/details?id=us.zoom.videomeetings (accessed January 8, 2018).

18. Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs &amp; more. (n.d.). https://www.amazon.com/.

19. Amazon Alexa. (n.d.). https://developer.amazon.com/alexa?cid=a.

20. Home Page Oficial da Microsoft. (n.d.). https://www.microsoft.com.

21. Personal Digital Assistant - Cortana Home Assistant - Microsoft. (n.d.). https://www.microsoft.com/en-us/cortana.

22. x.ai – AI Personal Assistant Who Schedules Meetings For You. (n.d.). https://x.ai/.

23. Apple. (n.d.). https://www.apple.com/.

24. iOS - Siri - Apple. (n.d.). https://www.apple.com/ios/siri/.

25. Google Calendar. (n.d.). https://google.com/calendar.

26. Android Studio. (n.d.). https://developer.android.com/studio/index.html (accessed January 15, 2018).

27. BPMN Specification - Business Process Model and Notation. (n.d.). http://www.bpmn.org/ (accessed January 18, 2018).

28. B. S. Richard W A L L A C E, The Elements of AIML Style. (2003).

29. Homepage | Scrum.org. (n.d.). https://www.scrum.org/ (accessed January 18, 2018).

30. D. Clegg & R. Barker, *Fast-track : a RAD approach* (Addison-Wesley Pub. Co., 1994).

31. Why You Only Need to Test with 5 Users. (n.d.). https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (accessed June 21, 2018).

Appendices

Appendix A – Work Plan

This appendix contains the planning of the work developed during the first semester of the internship and the work to be developed during the second semester.

1st Semester

State of the art	
20/09/2017 – 20/10/2017	• Definition of the project's main goals • Analysis of the state of the art ○ Competitor apps[30] ○ Comparative analysis • Analysis of the WIT RCS Suite • Definition of the features to be developed

Table A-1: Planning - State of the Art

Requirements	
23/10/2017 – 03/11/2017	• Definition of User Stories • Definition of Functional Requirements • Definition of Nonfunctional Requirements

Table A-2: Planning – Requirements

Analysis of the RCS+ Android app and definition of the architecture	
06/11/2017 – 17/11/2017	• Analysis of the RCS+ Android app code • Definition of the interaction between the app and the new modules to be developed

Table A-3: Planning – Architecture

Sprint 1	
20/11/2017 – 7/12/2017	• Search of an open source Calendar View; • Modification of the chosen calendar ○ Addition and removal of events ○ Synchronization with Google Calendar ○ Disable the possibility of schedule events on weekends and in the past ○ Disable the possibility to schedule events without internet connection •

Table A-4: Planning - Sprint 1

Sprint 2	

11/12/2017 – 22/12/2017	• Select participants to a conference call
	• Modify a conference call's details
	  o Edit day
	  o Edit Hour
	  o Edit Duration
	  o Edit Title
	  o Edit participants

Table A-5: Planning - Sprint 2

Sprint 3	
27/12/2017 – 10/01/2018	• Integration with the RCS+ app

Table A-6: Planning - Sprint 3

Sprint 4	
10/01/2018 – 30/01/2018	• Bug fixing and improvements to the work developed • Writing of the first version of the internship report and preparation of the internship presentation

Table A-7: Planning - Sprint 4

2nd Semester

Sprint 5	
31/01/2018 – 16/02/2018	• Integration of the Call Scheduling module with the bot • Validation and basic testing of the Call Scheduling module • Preparation for the development of the During Call module

Table A-8: Planning - Sprint 5

Sprint 6	
February/March	• Development of the ongoing conference call's basic information element

Table A-9: Planning - Sprint 6

Sprint 7	
March	• Development of the ongoing conference call's detailed information o Participants o Who is muted o Type of device used

Table A-10: Planning - Sprint 7

Sprint 8	
March/April	• Development of the ongoing conference call's detailed information o Who is speaking o Who is presenter

Table A-11: Planning - Sprint 8

Sprint 9	
April	• Development of the ongoing conference call's participant information o Name o Phone Number o E-mail

Table A-12: Planning - Sprint 9

Sprint 10	
May	• Validation and basic testing of the During Call module • Preparation for the development of the Conference Call's tab

Table A-13: Planning - Sprint 10

Sprint 11	
May/June	• Development of the Conference Call's tab

Table A-14: Planning - Sprint 11

Sprint 12	
June	• Validation and basic testing of the Conference Call's tab

Table A-15: Planning - Sprint 12

Sprint 13	
June/July	• Overall testing of the developed product • Writing of the final version of the internship report

Table A-16: Planning - Sprint 13

Appendix B – Requests to Designers

Request 1

App Android for Enterprise Bots – UI Requirements

Request to WIT Designers

Overview: The goal is to create a prototype version of the RCS+ app with additional features for users working in companies. Those features will be related to conference calls and triggered via bots.

A user should be able to schedule conference calls through a bot as well as be aware of an ongoing conference call's events also though a bot.

The new components should follow the style of the RCS+ app.

Ongoing conference call: During a conference call, if a user opens the bot's conversation he should be able to see the call's basic information. This should happen on any device, even if it is not the one that is being used for the call.

Requirement #1:

If the user is participating on a conference call, the bot's conversation screen should contain a fixed element (ex: a bar) with the call's duration and the number of active participants at the moment. This component should be expandable.

When the component is **not expanded**, it should show the following elements:

- Number of active participants

- Duration
- Icon indicating that the component is expandable

When the component is **expanded**, it should show the following elements:

- Number of active participants
- Duration
- Icon indicating that the component is collapsible
- Information about each active participant
    - RCS+ contact icon
    - Name
    - Speaker status
        - Is speaking
        - Is not speaking
        - Is muted
- Who is the current presenter

When a user joins the call, the component's information should be updated.

If the component is collapsed, the number of participants should be updates. If it is expanded, the participant's details should appear and the component's size should be adapted if necessary. The new user's information should have an indication, for a few seconds, showing that he just entered the call (for example, another colour).

When a user leaves the call, his information shouldn't be removed right away, but instead, stay for a few seconds with some indication showing that he's no longer on the call and then, be removed. Nevertheless, the number of participants on the top of the component should be updated right away.

On this component it should be possible to mute and unmute a participant and change the current presenter. There can be only one presenter at a time.

All the participants and all the icons should be clickable.

When expanded, if the number of participants, is big enough to not fit on the screen, it should be possible to scroll on the component, in order to access all the participants' information. If the number of participants is not big enough to fill the screen, the bot's conversation should remain visible on the back.

Figures 1 and 2 show an example of what is pretended for this requirement.

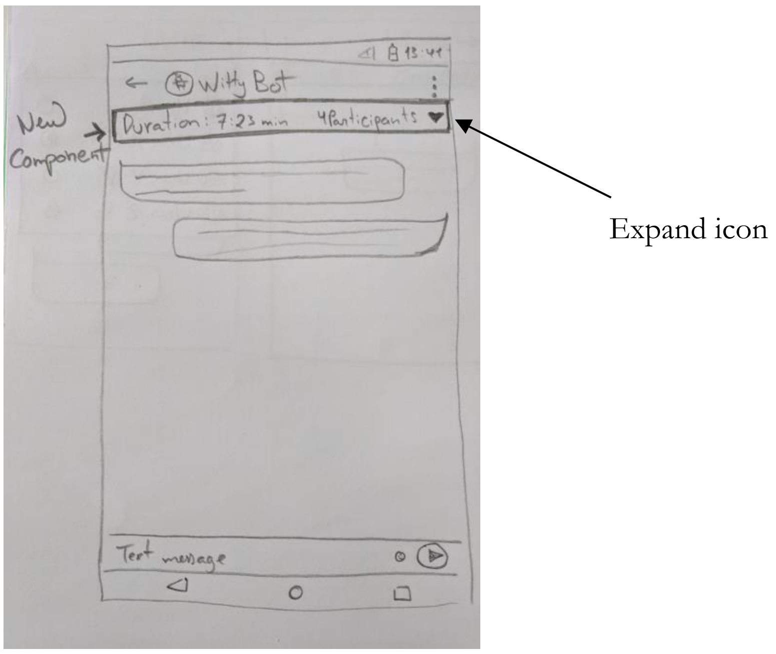


Figure B-0-1: Ongoing conference element collapsed sketch

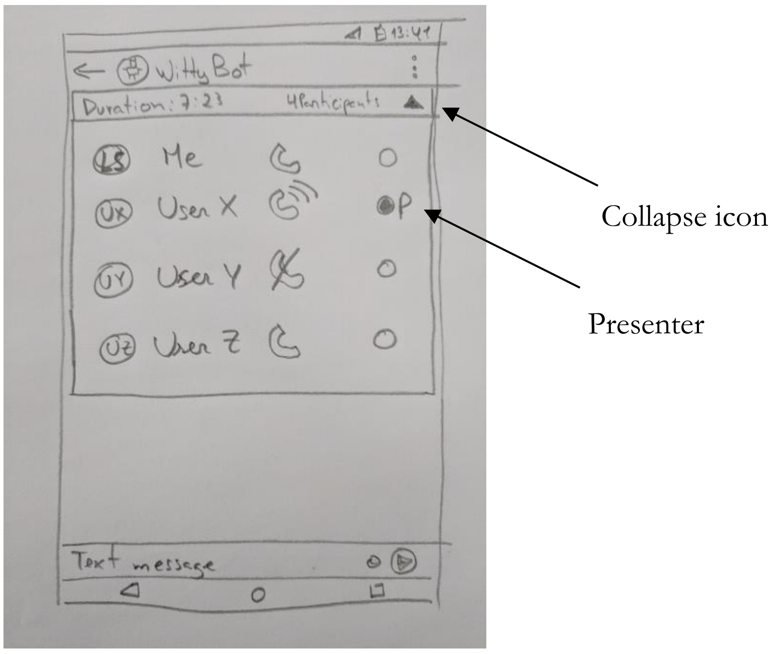


Figure B-0-2: Ongoing conference element expanded sketch

Modify conference call's details: It should be possible for a user to modify future call's details.

Currently, when the user asks the bot to schedule or modify a conference call, a calendar view is returned (figure 3). Here, the user can select an available slot to schedule a new call or select a call to change its details.

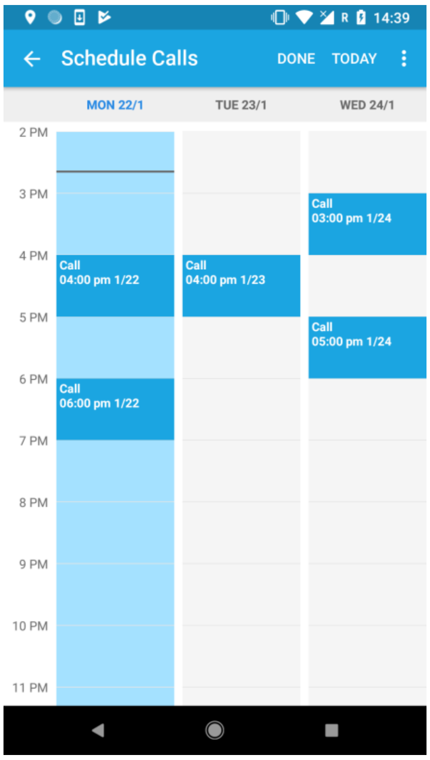


Figure B-0-3 - Calendar View

Requirement #3:

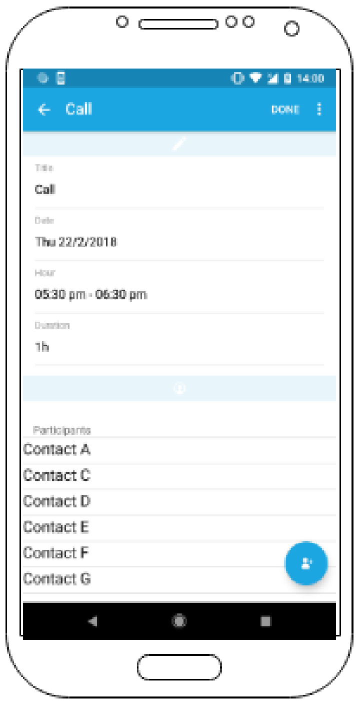
On the calendar view, when a conference call is selected, a screen with the call's details should be presented. This screen should contain the following elements:

- Conference Name
- Conference Date
- Conference Hour
- Conference Duration
- List of the conference invited participants

The user should be able to edit the call's name as well as its date and time. It should be also possible to remove and add participants.

It should be also possible to cancel the conference call. In order to allow that action, a button to cancel should be present (even if it is on the toolbar options).

Figure 4 shows an idea of what is pretended for this screen.



Title
Call

Date
Thu 22/2/2018

Hour
05:30 pm - 06:30 pm

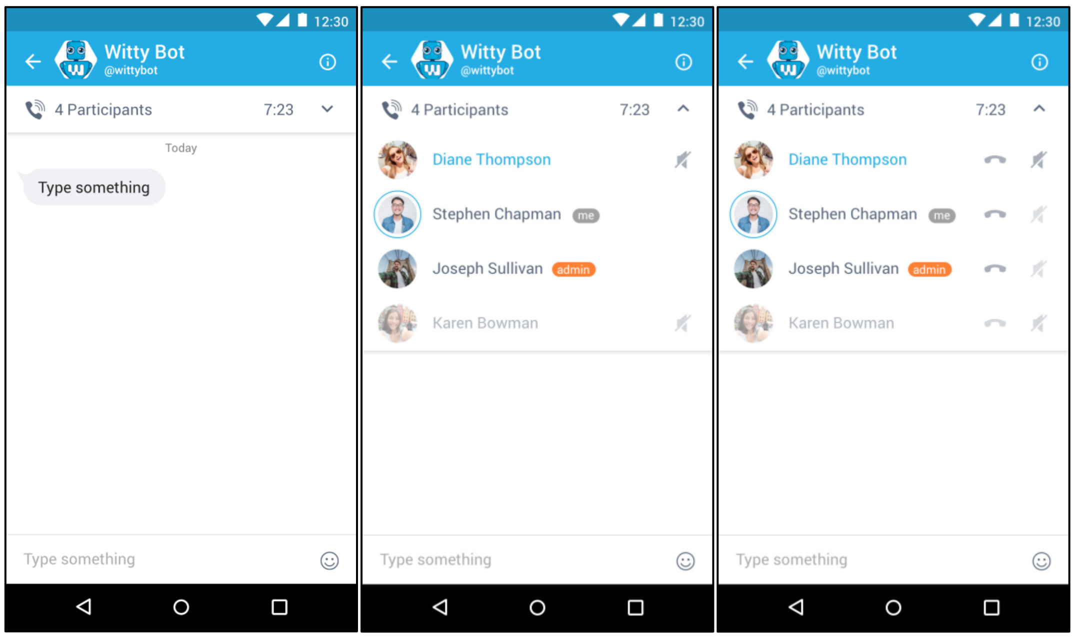
Duration
1h

Participants
Contact A
Contact C
Contact D
Contact E
Contact F
Contact G

Figure B-0-4: Edit conference plan

Answer to Request 1

Ongoing Conference Call


12:30
Witty Bot
@wittybot
4 Participants 7:23
Today
Type something
Type something

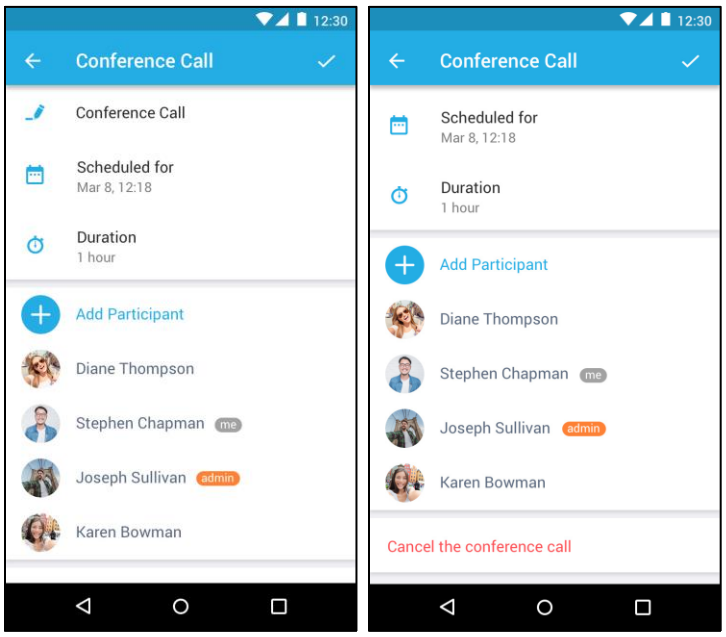
12:30
Witty Bot
@wittybot
4 Participants 7:23
Diane Thompson
Stephen Chapman me
Joseph Sullivan admin
Karen Bowman
Type something

12:30
Witty Bot
@wittybot
4 Participants 7:23
Diane Thompson
Stephen Chapman me
Joseph Sullivan admin
Karen Bowman
Type something

Normal User View Admin View

Figure B-0-5: Ongoing conference designers' proposal

Conference Call Details


12:30
Conference Call
Conference Call
Scheduled for
Mar 8, 12:18
Duration
1 hour
Add Participant
Diane Thompson
Stephen Chapman me
Joseph Sullivan admin
Karen Bowman

12:30
Conference Call
Scheduled for
Mar 8, 12:18
Duration
1 hour
Add Participant
Diane Thompson
Stephen Chapman me
Joseph Sullivan admin
Karen Bowman
Cancel the conference call

Figure B-0-6: Schedule conference designers' proposal

Deleting Participant

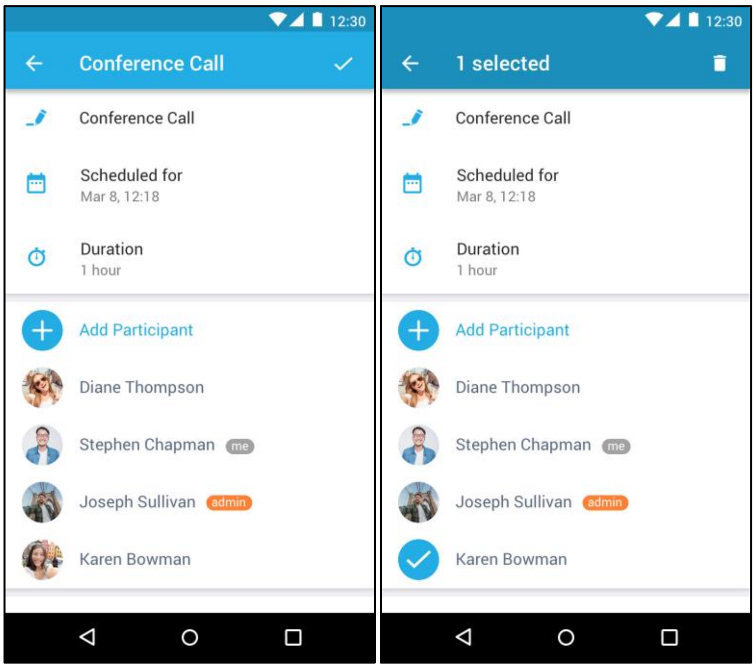


Figure B-0-7: Remove participant designers' proposal

Request 2

App Android for Enterprise Bots – Witty-bot images

Request to WIT Designers

Overview: The goal is to create a visual component to the witty bot's new functionalities.

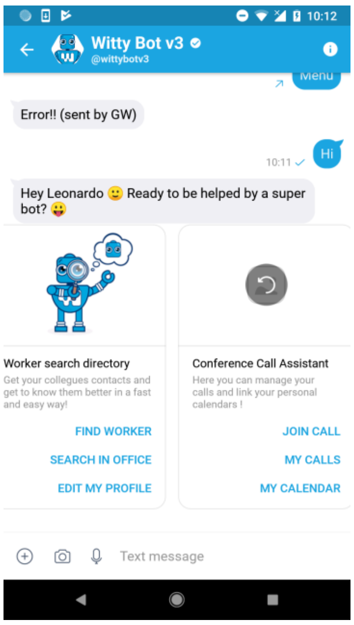
Requirement #1:

On the witty bot's menu, every set of functionalities has an image related.

A new set of functionalities was developed and it needs its own image. The new functionalities are related to conference calls.

Schedule of calls, management of ongoing calls and calendar synchronization are among the functionalities available on this new set of features.

Image 1.1 shows one of the sets of functionalities currently available on the menu (worker search directory) and the new one (that needs an image) related to the conference calls.



Error!! (sent by GW)

10:11 Hi

Hey Leonardo 🙂 Ready to be helped by a super bot? 😛

Worker search directory
Get your collegues contacts and get to know them better in a fast and easy way!

Conference Call Assistant
Here you can manage your calls and link your personal calendars !

Figure B-0-8: Witty Bot's main menu

<u>Requirement #2:</u>

When a user asks to see his upcoming calls, a carousel list (like the bot's menu) is shown where each element on the carousel represents a conference call.

For this call components there should also be an image.

Image 2 shows the component without image.

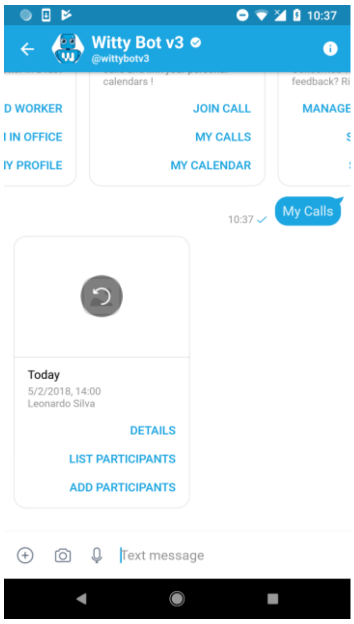


Figure B-0-9: Conference call list element

Answer to Request 2

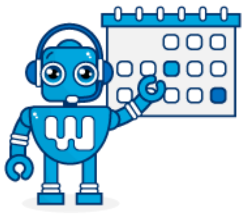


Figure B-0-10: Conference call image for Witty Bot's menu

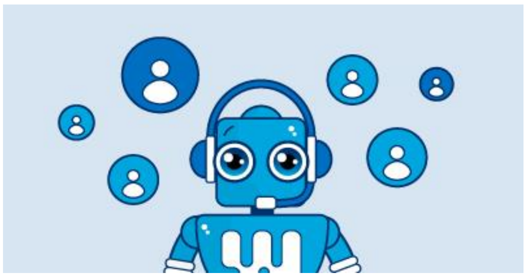


Figure B-0-11: Conference call list image

Appendix C – BPMN Flows

Schedule Conference Call Flow

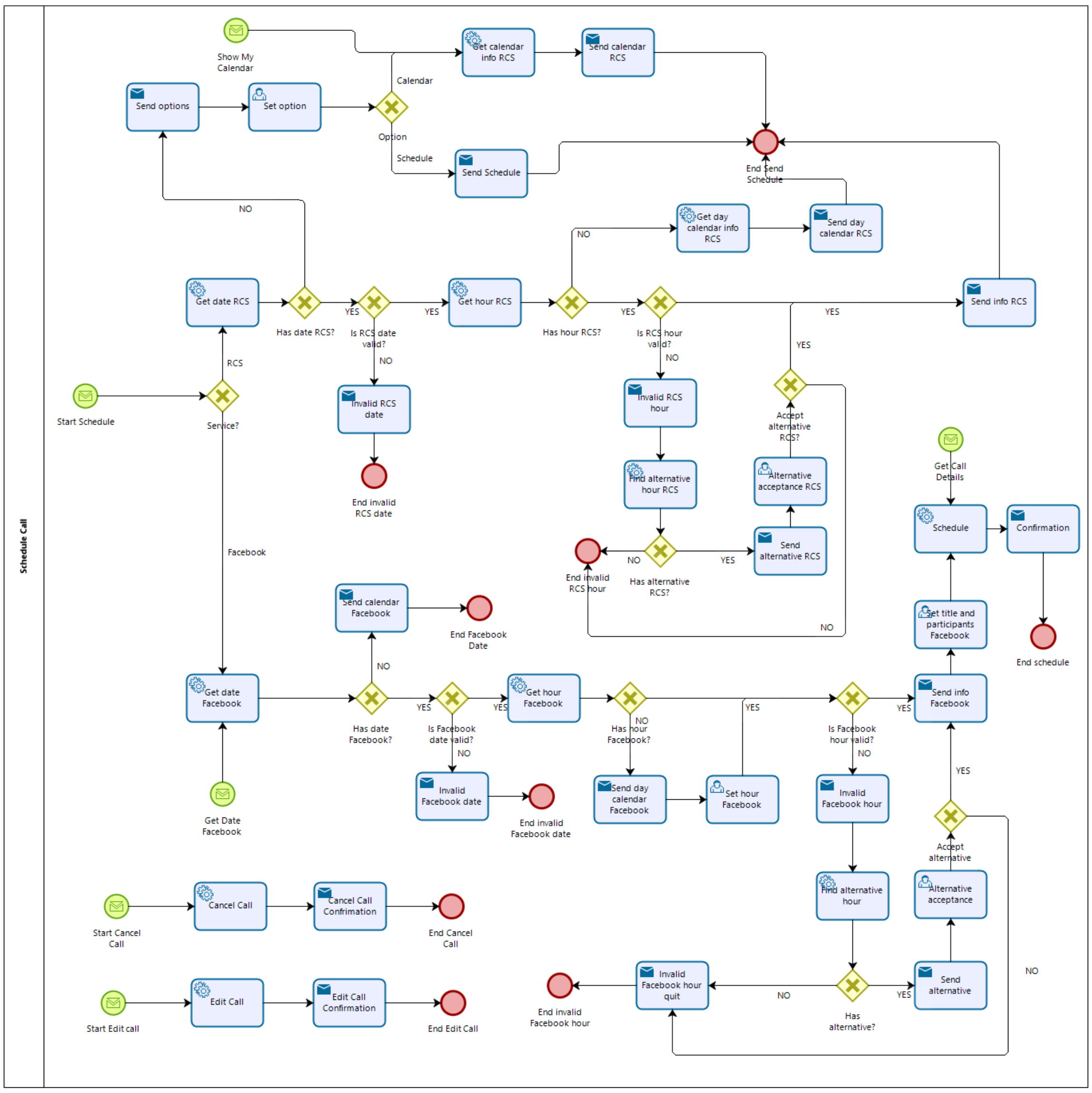


Figure C-0-1: Schedule conference BPMN diagram

Ongoing Conference Call Actions Flow

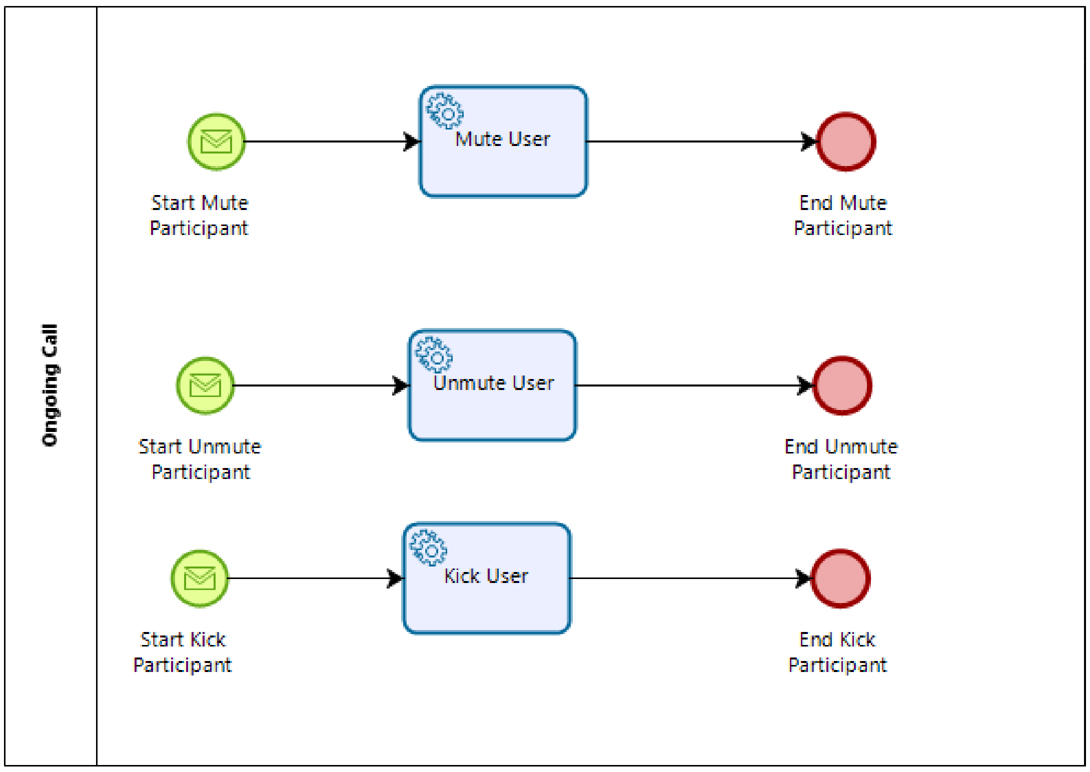


Figure C-0-2: Ongoing conference BPMN diagram

Calendar Tab Events Flow

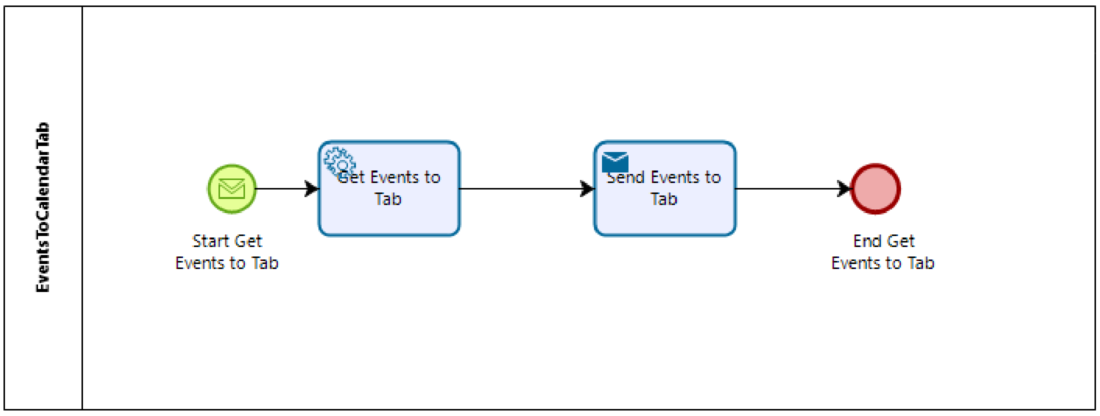


Figure C-0-3: Calendar tab events BPMN diagram

LDAP Integration Flow

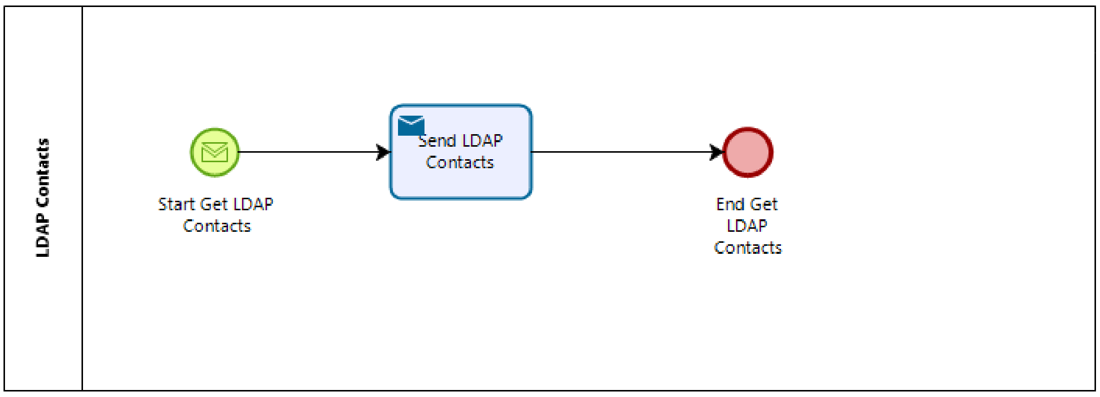


Figure C-0-4: LDAP integration

Appendix D – Functional Tests

SCHEDULE CALLS

FT1	**Ask the bot to schedule a call to a specific date (numeric) and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call for 25/5/2018 at 4pm)
Expected Result	○ Bot returns a message (invisible to the user) with commands to open the call scheduler screen with the date and hour values specified by the user
Result	SUCCESS

Table D-1: Functional Test 1

FT2	**Ask the bot to schedule a call to a specific date (non-numeric) and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call for tomorrow at 4pm)
Expected Result	○ Bot returns a message (invisible to the user) with commands to open the call scheduler screen with the date and hour values specified by the user
Result	SUCCESS

Table D-2: Functional Test 2

FT3	**Ask the bot to schedule a call to a past date (numeric) and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call for 25/5/2017 at 4pm)
Expected Result	○ Bot returns a message saying that the date specified is invalid
Result	SUCCESS

Table D-3: Functional Test 3

FT4	**Ask the bot to schedule a call to a past date (non-numeric) and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call for yesterday at 4pm)
Expected Result	○ Bot returns a message saying that the date specified is invalid
Result	SUCCESS

Table D-4: Functional Test 4

FT5	Ask the bot to schedule a call for today for an hour that has already passed
Steps	• Open bot's chat • Send message (e.g. Schedule a conference for today at 9 am)
Expected Result	○ Bot returns a message saying that the date specified is invalid ○ Bot asks the user if he wants to schedule the call to the next available slot (if it is 2:27pm, the bot suggests to schedule to 2:30pm)
Result	SUCCESS

Table D-5: Functional Test 5

FT6	Ask the bot to schedule a call for today at a valid hour
Steps	• Open bot's chat • Send message (e.g. Schedule a call for today at 5pm)
Expected Result	○ Bot returns a message (invisible to the user) with commands to open the call scheduler screen with the date and hour values specified by the user
Result	SUCCESS

Table D-6: Functional Test 6

FT7	Ask the bot to schedule a call for a specific day (with no hour)
Steps	• Open bot's chat • Send message (e.g. Schedule a call for tomorrow)
Expected Result	○ Bot returns a message (invisible to the user) with commands to open the call scheduler screen with the date value specified by the user (and by default to 9:00 am, which can later be modified by the user)
Result	SUCCESS

Table D-7: Functional Test 7

FT8	Ask the bot to schedule a conference without date and hour
Steps	• Open bot's chat • Send message (e.g. Schedule a call)
Expected Result	○ Bot returns a message with two quick replies for the user to choose. ○ View calendar ○ Schedule
Result	SUCCESS

Table D-8: Functional Test 8

FT9	**Select quick reply "View my calendar" after ask to schedule a call with no date and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call) • Select the "View my calendar" quick reply
Expected Result	○ Bot returns a message with a template for the user's calendar and its events
Result	SUCCESS

Table D-9: Functional Test 9

FT10	**Select quick reply "Schedule" after ask to schedule a call with no date and hour**
Steps	• Open bot's chat • Send message (e.g. Schedule a call) • Select the "Schedule" quick reply
Expected Result	○ Bot returns a message (invisible to the user) with commands to open the call scheduler screen with the next valid date and hour
Result	SUCCESS

Table D-10: Functional Test 10

FT11	**Ask the bot to see the calendar**
Steps	• Open bot's chat • Send message (e.g. View my calendar)
Expected Result	○ Bot returns a message with a template for the user's calendar and its events
Result	SUCCESS

Table D-11: Functional Test 11

FT12	**Select a future date while on the call scheduler screen**
Steps	• Select the date field • Select a future date
Expected Result	○ Hour selector is presented
Result	SUCCESS

Table D-12: Functional Test 12

FT13	**Select today's date while on the call scheduler screen**
Steps	• Select the date field • Select today's date
Expected Result	○ Hour selector is presented
Result	SUCCESS

Table D-13: Functional Test 13

FT14	**Select a past date while on the call scheduler screen**
Steps	• Select the date field • Select a past date
Expected Result	○ A toast message is presented saying that the selected date is not valid
Result	SUCCESS

Table D-14: Functional Test 14

FT15	**Select a future hour after the selection of today's date while on the call scheduler screen**
Steps	• Select the date field • Select today's date • Select a future hour
Expected Result	○ The date and hour are properly selected and are presented in the date field on the call scheduler screen
Result	SUCCESS

Table D-15: Functional Test 15

FT16	**Select a past hour after the selection of today's date while on the call scheduler screen**
Steps	• Select the date field • Select today's date • Select a past hour
Expected Result	○ A toast message is presented saying that the selected hour is not valid
Result	SUCCESS

Table D-16: Functional Test 16

FT17	**Select a duration while on the call scheduler screen**
Steps	• Select the duration field • Select a duration value (between zero and 24 hours)
Expected Result	○ The duration is properly selected and is presented in the duration field on the call scheduler screen
Result	SUCCESS

Table D-17: Functional Test 17

FT18	**Select an invalid duration while on the call scheduler screen**
Steps	• Select the duration field • Select duration zero hours and zero minutes
Expected Result	○ The minimum valid duration value is considered (half an hour).
Result	SUCCESS

Table D-18: Functional Test 18

FT19	**Give a title to a call while on the call scheduler screen**
Steps	• Select the title field • Insert a title
Expected Result	○ The title is properly given and it is presented on the title field on the call scheduler screen
Result	SUCCESS

Table D-19: Functional Test 19

FT20	**Select the "Add Participants" field while on the call scheduler screen**
Steps	• Select the "add participants" field
Expected Result	○ The WIT's contact list is presented
Result	SUCCESS

Table D-20: Functional Test 20

FT21	**Select the "Add Participants" field when the number of participants added is the limit (10)**
Steps	• Select the "add participants" field
Expected Result	○ Nothing happens (the field is blocked and with a different color)
Result	SUCCESS

Table D-21: Functional Test 21

FT22	**Select a participant's field while on the "add participants screen"**
Steps	• Select a contact's field
Expected Result	○ The contact's icon changes ○ The contact's name is added to the top of the screen
Result	SUCCESS

Table D-22: Functional Test 22

FT23	**Select a participant's field that is already selected while on the "add participants screen"**
Steps	• Press a selected contact's field
Expected Result	○ The contact's icon turns to its original ○ The contact's name is removed from the top of the screen
Result	SUCCESS

Table D-23: Functional Test 23

FT24	**Write the phone number of a contact while on the "add participants screen"**
Steps	• Write a phone number of a contact on the "add participants screen"
Expected Result	○ The contact's icon changes ○ The contact's name is added to the top of the screen
Result	SUCCESS

Table D-24: Functional Test 24

FT25	**Try to add own phone number to the participants list**
Steps	• Write own phone number
Expected Result	○ A message is displayed saying that the action is not possible
Result	SUCCESS

Table D-25: Functional Test 25

FT26	**Write the name of a contact on the contact filter**
Steps	• Write the name of a contact
Expected Result	○ Only the contacts that contain the written character sequence are displayed
Result	SUCCESS

Table D-26: Functional Test 26

FT27	**Select the "Cancel call" field while creating a call**
Steps	• Select the "cancel call" field
Expected Result	o Nothing happens (the field is blocked and with a different color)
Result	SUCCESS

Table D-27: Functional Test 27

FT28	**Select a participant of a conference while on the schedule conference screen**
Steps	• Select a participant
Expected Result	o The participant's icon changes o The toolbar's color changes o The "done" icon disappears o The "remove" icon appears o The toolbar title changes and indicates the number of participants that are selected
Result	SUCCESS

Table D-28: Functional Test 28

FT29	**Select a participant of a conference while on the schedule conference screen after having already selected another participant**
Steps	• Select a participant
Expected Result	o The participant's icon changes o The toolbar title, indicating the number of selected participants, changes
Result	SUCCESS

Table D-29: Functional Test 29

FT30	**Select a participant after having already selected it**
Steps	• Select a participant that is selected
Expected Result	o The participant's icon changes back to its original o The toolbar is adapted
Result	SUCCESS

Table D-30: Functional Test 30

FT31	**Select the "go back" option while on the schedule conference screen after having participants selected**
Steps	• Select the "go back" options
Expected Result	o The selected participants' icons change back to the original o The toolbar's color changes o The toolbar title changes o The "remove" icon disappears o The "done" icon appears
Result	SUCCESS

Table D-31: Functional Test 31

FT32	**Select the "remove" icon while on the schedule conference screen after having selected participants**
Steps	• Select the "remove" option
Expected Result	o The selected participants are removed o The toolbar's color changes o The toolbar title changes o The "remove" icon disappears o The "done" icon appears
Result	SUCCESS

Table D-32: Functional Test 32

FT33	**Select the "Done" button while scheduling a call**
Steps	• Select the "Done" button
Expected Result	o A message is sent to the bot with the calls information o The call is scheduled on the asterisk server o The call is scheduled on the bot's database o The call is scheduled on the user's calendar
Result	SUCCESS

Table D-33: Functional Test 33

FT34	**Select an event that is not a conference call on the calendar view**
Steps	• Select an event that is not a conference call on the calendar view
Expected Result	o A toast message is shown saying that the event is not a conference call and it cannot be opened on the app
Result	SUCCESS

Table D-34: Functional Test 34

FT35	Select an event that is a conference call on the calendar view
Steps	• Select an event that is a conference call on the calendar view
Expected Result	o The event is opened on the edit conference call screen
Result	SUCCESS

Table D-35: Functional Test 35

FT36	Select an event on the calendar view without having internet access
Steps	• Select an event on the calendar view
Expected Result	o A message is shown saying that the operation is not possible without an internet access
Result	SUCCESS

Table D-36: Functional Test 36

FT37	Try to select an empty slot on the calendar view without having internet access
Steps	• Select an empty slot on the calendar view
Expected Result	o A message is shown saying that the operation is not possible without an internet access
Result	SUCCESS

Table D-37: Functional Test 37

FT38	Select the floating action button on the calendar tab
Steps	• Press the button on the tab's bottom right corner
Expected Result	o The "schedule conference call screen" is opened filled with the next available time slot
Result	SUCCESS

Table D-38: Functional Test 38

FT39	Cancel a conference call
Steps	• Select a conference call on the calendar view • Select the "Cancel conference call" field
Expected Result	o The app sends a message to the bot saying to cancel the call o The call is deleted from the bot's database o The call is removed from the user's google calendar o User receives a message saying that the call was successfully canceled
Result	SUCCESS

Table D-39: Functional Test 39

FT40	**Edit a conference call's participants**
Steps	• Select a conference call on the calendar view • Change the call's participants • Select the "Done" button
Expected Result	○ The app sends a message to the bot with the new information of the call ○ The call's information is updated on the bot's database ○ The call's information is updated on the user's google calendar ○ User receives a message saying that the call was successfully edited
Result	

Table D-40: Functional Test 40

FT41	**Edit a conference call's date**
Steps	• Select a conference call on the calendar view • Change the call's date • Select the "Done" button
Expected Result	○ The app sends a message to the bot with the new information of the call ○ The call's information is updated on the bot's database ○ The call's information is updated on the user's google calendar ○ User receives a message saying that the call was successfully edited
Result	

Table D-41: Functional Test 41

FT42	**Select the participant "me" field on the "schedule conference screen"**
Steps	• Select the participant "me" on the "schedule conference screen"
Expected Result	○ The user's profile is opened with option to edit it
Result	SUCCESS

Table D-42: Functional Test 42

FT43	**Select another participant's field on the "schedule conference screen"**
Steps	• Select another participant on the "schedule conference screen"
Expected Result	○ The participant's details are shown
Result	SUCCESS

Table D-43: Functional Test 43

Call about to start

FT44	A call is about to start
Steps	• A call is scheduled
Expected Result	○ Bot sends a message saying that a call is about to start with the following quick replies: ○ Call me ○ I'm running late ○ Cannot make it
Result	SUCCESS

Table D-44: Functional Test 44

FT45	A call is about to start and user selects "call me" quick reply
Steps	• A call is scheduled • Bot sends a message with quick replies • User selects "Call me" quick reply
Expected Result	○ User receives a call to join the call
Result	SUCCESS

Table D-45: Functional Test 45

FT46	A call is about to start and user selects "I'm running late" quick reply
Steps	• A call is scheduled • Bot sends a message with quick replies • User selects "I'm running late" quick reply
Expected Result	○ Bot sends a message to the call's participants saying that the user is running late
Result	SUCESS

Table D-46: Functional Test 46

FT47	A call is about to start and user selects "Cannot make it" quick reply
Steps	• A call is scheduled • Bot sends a message with quick replies • User selects "Cannot make it" quick reply
Expected Result	○ Bot sends a message to the call's participants saying that the user cannot make it to the call
Result	SUCCESS

Table D-47: Functional Test 47

Ongoing call

FT48	User joins call
Steps	• A call starts • User receives a call to join the call
Expected Result	○ User joins the call ○ The ongoing call top bar turns visible on the app
Result	SUCCESS

Table D-48: Functional Test 48

FT49	User leaves call
Steps	• User is participating on a call • User hangs up the phone
Expected Result	○ User leaves the call ○ The ongoing call top bar turns transparent and disappears after 5 seconds.
Result	SUCCESS

Table D-49: Functional Test 49

FT50	User joins call again after leaving
Steps	• A call starts • User joins the call • User leaves the call • User joins the call
Expected Result	○ User joins the call ○ The ongoing call top bar adapts its content correctly
Result	SUCCESS

Table D-50: Functional Test 50

FT51	Another user joins call
Steps	• User is participating on a call • Another user joins the call
Expected Result	○ The ongoing call top bar adjusts the number of active participants ○ The ongoing call top bar adapts its size and add the field of the new participant
Result	SUCCESS

Table D-51: Functional Test 51

FT52	**Another user leaves call**
Steps	• User is participating on a call • Another user leaves the call
Expected Result	o The ongoing call top bar adjusts the number of active participants o The field associated to the user turns transparent and after 5 seconds it is removed and the ongoing call's top bar size is adapted
Result	SUCCESS

Table D-52: Functional Test 52

FT53	**User is participating on a call and he is the owner**
Steps	• User is participating on a call
Expected Result	o The ongoing call's top bar includes the options to mute/unmute participants and the options to kick participants
Result	SUCCESS

Table D-53: Functional Test 53

FT54	**User is participating on a call and he is not the owner**
Steps	• User is participating on a call
Expected Result	o The ongoing call's top bar includes the icons of muted/unmuted participant but does not include the option to kick participants
Result	SUCCESS

Table D-54: Functional Test 54

FT55	**Owner mutes participant**
Steps	• User presses the icon to mute a participant
Expected Result	o The participant is muted
Result	SUCCESS

Table D-55: Functional Test 55

FT56	**Owner unmutes participant**
Steps	• User presses the icon to unmute a participant
Expected Result	o The participant is unmuted
Result	SUCCESS

Table D-56: Functional Test 56

FT57	Owner kicks participant
Steps	• User presses the icon to kick a participant
Expected Result	○ The participant is removed from the call
Result	SUCCESS

Table D-57: Functional Test 57

FT58	Guest presses the icon to mute a participant
Steps	• Guest presses the icon to mute a participant
Expected Result	○ Nothing happens
Result	SUCCESS

Table D-58: Functional Test 58

FT59	Guest presses the icon to unmute a participant
Steps	• Guest presses the icon to unmute a participant
Expected Result	○ Nothing happens
Result	SUCCESS

Table *D-59*: Functional Test 5*9*

Ldap integration

FT60	Ldap integration
Steps	• Start the app
Expected Result	○ The app sends a request to the bot for the ldap contacts integration ○ The bot sends a template message with the contacts of WIT's workers ○ The contacts are added to the app's contact list
Result	SUCCESS

Table D-60: Functional Test 60

FT61	WIT's contacts filter
Steps	• Select the WIT's contact filter on the contact list
Expected Result	○ Only the WIT's workers contacts are shown
Result	SUCCESS

Table D-61: Functional Test 61

Calendar Tab

FT62	Go to calendar tab
Steps	• Go to calendar tab
Expected Result	○ App sends a request to the bot for the user's calendar events ○ Bot send's a template message with the user's calendar events ○ The tab's calendar is displayed with the user's calendar events ○ The tab's icon turns white
Result	SUCCESS

Table D-62: Functional Test 62

FT63	Leave calendar tab
Steps	• Leave calendar tab
Expected Result	○ Tab's icon turns blue
Result	SUCCESS

Table *D-63*: Functional Test 6*3*

FT64	"Go to date" on calendar tab
Steps	• Select the "go to date" button on the calendar tab • Select a date on the date picker presented
Expected Result	○ The calendar view adapts its content to the date selected
Result	SUCCESS

Table D-64: Functional Test 64

FT65	“Go to today” on calendar tab
Steps	• Select the “Today” button on the calendar tab
Expected Result	○ The calendar view adapts its content to “today”
Result	SUCCESS

Table D-65: Functional Test 65

FT66	Select “Day view” on calendar tab
Steps	• Select the “Day view” button on the calendar tab
Expected Result	○ The calendar view adapts its content and shows only one day (today)
Result	SUCCESS

Table D-66: Functional Test 66

FT67	Select “3 day view” on calendar tab
Steps	• Select the “3 day view” button on the calendar tab
Expected Result	○ The calendar view adapts its content and shows three days (today and the next two days)
Result	SUCCESS

Table D-67: Functional Test 67

FT68	Select “Week view” on calendar tab
Steps	• Select the “Week view” button on the calendar tab
Expected Result	○ The calendar view adapts its content and shows seven days (today and the next six days)
Result	SUCCESS

Table D-68: Functional Test 68

General

FT69	User starts app for the first time
Steps	• Start the app
Expected Result	○ Bot sends a message asking the user to register his email
Result	SUCCESS

Table D-69: Functional Test 69

FT70	**User registers in the bot**
Steps	• User sends a message with his email
Expected Result	○ Bot analyses if it is a "WIT email" ○ Bot sends a message with a link for the user to insert his credentials
Result	**SUCCESS**

Table D-70: Functional Test 70

FT71	**Log in on bot**
Steps	• Open link to insert credentials
Expected Result	○ Bot registers user on its database ○ Bot sends a personalized greeting and the main menu
Result	**SUCCESS**

Table D-71: Functional Test 71