****

**Analysis and Improving company documentation usage by developing a chatbot and integrating it with existing messaging platforms**

****

Master Thesis

Name of the Study Programme

Professional IT Business and Digitalization

**Faculty ZbwS**

from

Rishi Srinivasan Kanaka Sabapathy

Date:

Berlin, 14.02.2023

1st Supervisor: Dr. Helena Mihaljević

2nd Supervisor: Nico Schönnagel

**Table of Contents**

**Abstract iv**

**List of Figures v**

**List of Tables vi**

**List of Abbreviations vii**

**Chapters**

**1** **Introduction 8**

1.1 Motivation9

1.2 Thesis Approach. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

1.3 Contributions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

1.4 Thesis Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .20

**2** **Status Quo 23**

2.1 Requirement Document for PoC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.2 PoC Application Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2[5](#_heading=h.30j0zll)

2.3 PoC Application Functionality . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

2.4 Smart Slackbot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

2.5 Existing Processes and Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

**3** **Development Plan 35**

4.1 System Requirements Analysis 4

4.2 Conversational User flows [5](#_heading=h.30j0zll)

4.3 Step-by-Step Plan [5](#_heading=h.30j0zll)

**4** **Architecture 10**

5.1 Google DialogFlow **11**

5.2 Slack Application **12**

5.3 Amazon Web Services **13**

5.4 Smart Slackbot Architecture **14**

**5** **Infrastructure 15**

6.1 Microservices 16

6.2 Cloud Deployment 17

6.3 Infrastructure-as-a-Code 18

**6** **Implementation 20**

7.1 Application Implementation 21

7.2 Monitoring and Improvements 22

**7** **Conclusion 22**

**Statutory Declaration 23**

**References 24**

ABSTRACT  
This thesis aims to bring in  
the first part of the thesis introduces  
the second part

LIST OF FIGURES

Figure 1: Example of label (style sheet „Figure“)

LIST OF TABLES

Table 1: Example for a table (style sheet „Figure“)

LIST OF ABBREVIATIONS

NLU Natural Language Understanding

ML Machine Learning

IoT Internet of Things

AWS Amazon Web Services

LLM Large Language Models

NLP Natural Language Processing

AI Artificial Intelligence

PoC Proof of Concept

MSA Microservices Architecture

GPU Graphic Processing Unit

TPU Tensor Processing Unit

BPA Business Process Automation

IaC Infrastructure-as-Service

CDK Cloud Development Kit

UI User Interface

RPA Robotic Process Automation

ES Essentials

SDLC Software Development Life Cycle

CI Continuous Integration

CD Continuous Development

SRA System Requirement Analysis

FRD Functional Requirements Document

TRD Technical Requirements Document

# 

**CHAPTER**

ONE

INTRODUCTION

Organisations of all sizes struggle with the challenge of disconnected document processes, a pervasive problem whose negative impact cuts across all business functions [1]. With fast-changing global markets, companies struggle to break down silos and boost cross-functional collaboration [2]. This is especially a major concern in startups and small organisations where cross-functional collaboration becomes more challenging. Unlike companies like Google and Amazon which have dedicated teams with hundreds of employees for employee onboarding, training programs and support centres for providing assistance to employees regarding company processes, small organisations are often underemployed and overworked due to constraints in funding and scaling the business. People are expected to work across different functions at all times with little or no knowledge of the existing processes in the organisation. Startups and small organisations with fewer employees also face fierce competition and have little or no time to create a structured onboarding process for the employees. The lack of opportunity to create training programs or to have an automated onboarding system for all employees and train them across different business functions adversely impacts on the revenue, customer engagement and team productivity.

The basic principle to create efficiency, analyse the work being done, provide better customer service and innovate solutions is for the employees to fully understand the company processes and improve collaboration among teams. This includes creating a culture of automation which is essential to an organisation’s growth and success. Collaboration among teams is as important as automation or building quality softwares. This is because a company without its employees interacting with each other on a daily basis, for knowledge sharing or building relationships, cannot function as a team. This is why team collaboration tools like slack are used in many companies. Based on these fundamental principles, the key to avoiding bottlenecks and to enable smooth software development and flow of information among cross-functional teams is by automation through ChatOps. ChatOps is a collaboration model that connects people, tools, process and automation into a transparent workflow [3]. With the help of ChatOps the disconnected documents problem that companies face can be solved through continuous automation and collaboration in a transparent manner.

In this thesis, I a) explore the problems that *OEV Online Dienste GmbH* face due to disconnected document processes and roadblocks in team collaborations which affects productivity b) propose a solution by building an automated chatbot using cloud technologies, collaboration tools and Natural Language Understanding (NLU), to improve employees’ access to the knowledge base and onboarding processes at the company, c) implement the developed solution onto the company’s cloud native environment and integrate it efficiently into their existing infrastructure thereby making it accessible to all their employees.

## Motivation

Automation has been pivotal in changing how the world operates. From manufacturing industries to Amazon warehouses, from self-checkout stores to home automation, every aspect of our life has some amount of automation in it. Automation is also crucial to have a robust software development process in deciding how quickly and efficiently we build softwares. With the advent of digital technologies like Machine Learning (ML), Cloud Computing, and Internet of Things (IoT), we have plenty of resources at our disposal to build things that have never been built before.

Public cloud providers like Amazon Web Services (AWS) offer their massive infrastructure including computing power, scalability and reliability which is essential to building and deploying scalable software applications. This infrastructure is expensive and difficult to maintain in a private on-premise set up. Companies like Google have also opened up their Natural Language Processing (NLP) platforms like DialogFlow Essentials (ES) where you can build highly advanced chatbots and integrate it into any platform. Building private Large Language Models (LLM) such as Google’s Bidirectional Encoder Representations from Transformers (BERT), which is used to create sophisticated chatbot agents in DialogFlow ES, is close to impossible at the cost at which Google offers its DialogFlow services for public use. Finally, slack, despite being a massive instant messaging platform on its own which provides professional and organisational communication service to thousands of companies, offer a variety of possibilities to integrate these external services from AWS or Google DialogFlow ES into their platform. Not just infrastructure-based and collaboration-based advancements have come up in the tech environment in recent years but also architectural changes such as migrating from monolithic architecture to Microservices architecture (MSA) have modified the way we build modern scalable applications. As a software developer, these advancements fascinate me and have been my inspiration to harness these digital technologies in building scalable and reliable software applications that can help businesses or people make their lives easier.

This master thesis is one such opportunity for me to build a scalable software application using Slack, AWS and Google DialogFlow ES in order to assist *OEV Online Dienste GmbH* in improving their existing employee management system and documentation process. The problem that I am trying to solve at the companyis that their existing employee management system and corporate wiki does not offer clear visibility to their employees. For example, If an employee needs information regarding a project or a process, the employee has to manually scan through their entire corporate wiki to find the right documentation. Alternatively an employee has to ask another employee(s) for information thereby disrupting both employees’ productivity. This scenario exists not just for the knowledge base in their company wiki but also for other employee services such as applying leaves and holidays, submitting invoices, initiating a new project idea, etc.

In order to automate this process, improve knowledge base visibility and boost employee productivity, I am building an application called ‘***Smart Slackbot***’ that acts as a chatbot on the slack application and informs/redirects users to the right processes and documents within the company. When an employee mentions a particular keyword related to a process or a document on the company’s slack channel, the chatbot which is installed in the company’s slack workspace listens to the keyword mentioned by the employee, get activated and responds to the employee’s query inside the same slack channel. This is the basic functionality of the *Smart Slackbot* application. The goal of this application is to act as a knowledge distribution system guiding employees to information that they require at different stages of software development and also help employees find the right processes they want to initiate within the company. Thereby making life easier for both the employees and the company in the area of knowledge sharing by creating a streamlined process of automation.

## Thesis Approach

The approach to building the ‘*Smart Slackbot*’ application and ergo my thesis approach, is briefly divided into 3 parts:

1. Theoretical Analysis
2. Natural Language Processing
3. Cloud Computing

First is the theoretical analysis part where I perform a comprehensive background study of the existing collaboration tools, corporate wikis, organisational process optimization methods and chatOps through various research papers, journals and documentations. The reason why I performed this theoretical analysis before jumping in to design my application’s architecture was to learn about the existing technologies that are already available and used in today’s market so that I can select the most suitable tools and methodologies that would fit the requirement criteria to solve the problem specific to the company and not to develop a generalised solution.

The documentations helped me understand the capabilities and functionalities of existing collaboration tools. The research papers gave me an in-depth knowledge of how certain methodologies like ChatOps can significantly improve an existing system through automation [4]. The theoretical analysis proved to be valuable as it gave me an idea of how I can design my system and also provided the scientific validation that certain methodologies and tools have proved to be successful in the past in getting the desired results out of the application. In the next section - *1.3 Contributions*, I explain how I decided to use slack as the collaboration tool, MSA as the architecture and chatOps as the methodology for developing the *Smart Slackbot* application.

Second part of the thesis approach is the Natural Language Processing (NLP) where I had to choose the right conversational Artificial Intelligence (AI) platform to build my chatbot. After choosing chatOps as my methodology to create a conversational experience with my application, I had two options available - create my own LLMs using a transformer model from scratch, for the NLU task, and build a chatbot on top of it. This would mean that I had to train my LLMs with large amounts of text data in order to have a good conversational experience for the users. This required vast amounts of text data, both in German and in English since the application is developed in both languages. Even if I had managed to collect the large datasets required to train my LLM, I would still have to label them and fine tune the model regularly with new data in order to build a good responsive NLP based chatbot. Apart from these challenges, I would also need intensive computing power like Graphic Processing Units (GPUs) and Tensor Processing Units (TPUs) in order to train the LLMs quickly and efficiently.

All of this seemed to be time consuming, ineffective and more expensive when there are really good LLMs available for public use offered by public cloud providers like Google on their Google Cloud Platform. Let us assume that I managed to collect the necessary data, performed labelling on the datasets and trained the LLMs on new data regularly with the necessary computing power. Even then there is no guarantee that my LLMs will outperform the existing transformer based models that are already available in the market like Google’s BERT.

Therefore, I chose to go with the second option which is to use an existing LLM that allows me to create a chatbot on top of it. I chose Google DialogFlow ES because it offered both the LLM and also the ability to build a chatbot agent. DialogFlow ES has one of the best LLMs available in the market - BERT, arguably one of the best in the world, for performing the NLU task in the background. DialogFlow ES uses Google’s BERT which makes it really efficient in a conversation as it retains the context of the conversation and responds accordingly. It is also able to distinguish between synonyms and captures the true meaning of a sentence. The cost was also inexpensive as compared to building my own LLMs. Therefore the second option was the clear choice considering the time, cost and efficiency in building the *Smart Slackbot* application.

At this point, my application architecture was already starting to take shape. I had decided that the collaboration tool would be Slack, the methodology would be ChatOps, the NLP platform would be Google Cloud Provider (GCP) and the chatbot would be DialogFlow ES.

The third and final part of the thesis approach is to bring together all these services and tools under one roof in a public cloud environment where you can integrate them seamlessly and create a scalable, reliable, MSA based application. Although there are many public cloud providers in the market like GCP, Microsoft Azure and Oracle cloud, I chose AWS as my cloud platform to integrate these tools and services. This is because of two main reasons - First, the company already has a strategic partnership with AWS. Almost all their public cloud infrastructure is in AWS. Therefore it is easier for me to build my application on their existing AWS infrastructure and integrate it seamlessly into their environment. Second, I personally prefer using AWS because I had prior experience working in AWS for my personal projects. AWS is secure, stable, reliable and highly configurable to suit personal requirements which means I can modify my architecture in a way that would best suit my application. AWS is also cheaper as compared to other large cloud providers like Azure and GCP for the advanced services that they offer. Hence it was an obvious choice to go ahead with AWS as my cloud platform.

These 3 approaches form the basis of my thesis and for the *Smart Slackbot* application. The contributions from these 3 approaches are subdivided into two categories and explained in the next section.

## Contributions

In this section, I will explain what types of contributions were involved in writing my master thesis and how these contributions helped me create the skeleton for the *Smart Slackbot* application.

To briefly explain, there are 2 contributions that were detrimental for my thesis:

1. Theoretical Contributions
2. Empirical Contributions

###### **Theoretical contributions**

Theoretical contributions come from the theoretical analysis which I had already mentioned in the previous section - *1.2 Thesis Approach*. I had performed the theoretical analysis before building the *Smart Slackbot* application to understand and learn about the existing collaboration tools, corporate wikis, organisational process optimization methods and chatOps by the means of research papers, journals and documentations. I will explain the results of this study in this section. I started with the topic of ***Business Process Automation (BPA)*** to learn how it can be achieved in a small organisation as the goal of my thesis is to automate business processes in order to ensure more efficiency in the organisation. BPA describes a situation where a business process is executed without any human intervention - when a task is implemented through software and is executed behind the scenes, on the schedule or automatically.

BPA provides certain benefits to the company such as higher productivity, improved efficiency, less human error, allows employees to focus on important things and reduces operating costs. Even though BPA provides these valuable benefits to organisations, it was important for me to consider the common pitfalls that come along with automation in business. I had cautiously gone through the common pitfalls that businesses usually face while adopting automation: a) falling in love with a single technology such as Robotic Process Automation (RPA) which is widely known for automation, b) believing that no code applications or external softwares can solve business problems that are specific to the company.

Gartner Inc, which is a highly reputed tech research and consulting firm listed the above mentioned pitfalls as the top 2 automation mistakes to avoid in one of their articles [5]. My goal was to avoid these pitfalls right from the beginning in order to build meaningful technology that actually provides value to business. This is the reason why I started building a customised automation application that solves the company’s specific problem. My other considerations from the theoretical analyses include:

1. Not automating broken processes

The current process of knowledge sharing at the company is already detailed and well executed. The only process that was missing from making it more efficient was automation. Therefore, I was confident that by properly implementing automation to their existing process, which was not broken, will only make the company and employees more efficient.

1. Picking automation tools that correspond to the company’s business model

The tools that I had selected for building *Smart Slackbot* include Slack, AWS, and Google DialogFlow ES. Slack and AWS, as already mentioned, were used by the company due to their strategic partnership and licensing. Google DialogFlow ES was the best available option to build my NLP based chatbot and it was also cost effective due to its pay-as-you-go pricing model. These tools correspond to the company’s business model and also provided me the flexibility to customise based on the needs of the company. This is why even though it is the same cloud provider - GCP, I did not choose DialogFlow Customer Experience (CX) agent which is a more advanced and sophisticated conversational AI chatbot as compared to DialogFlow Essentials (ES) agent which is the smaller version among the two. For the company’s business requirement DialogFlow ES was more than enough. I will delve more deeper into their differences in the next chapter in *section 2.5 - Google DialogFlow*.

After learning in-depth about BPA to implement the best practices in business process automation, I had to decide the architecture of my application. At this point, I had already worked on both Monolithic Architecture (back as a C# developer in India) and Microservices Architecture (for my projects at HTW Berlin). The argument between the two architectures on which is better is never ending in the tech community but the most simplest answer in my opinion is - “It depends on your application”. Both architectures have their merits and demerits. It is up to the solutions architect and the product owner to decide which architecture suits better to serve the needs of their application. As the cloud solution architect for the *Smart Slackbot* application, I chose MSA over monolithic architecture for the following reasons:

1. Each service in MSA can be independently developed, updated, deployed, scaled or replaced. This freedom is not possible in monolithic architecture since all the services in monolithic architecture are built as a single unified unit. Splitting independent services or scaling them up or down was not possible in monolithic architecture. This was an important consideration in building our application since in the future, if the company decides to scale up or modify a part of the application, for example, change the conversational AI platform from Google DialogFlow ES to another product that better suits the company’s needs then they don’t have to break the whole architecture. They can just replace the DialogFlow ES API with the new API and the application will function perfectly well. This makes MSA agile and the best option for an ever changing tech environment.
2. I wanted to implement the DevOps ideology from the early stage of Software Development Life Cycle (SDLC) of the *Smart Slackbot* application as it provides valuable insights into the production environment in which the application will be deployed. Thereby helping in improving the software quality [6]. MSA emerged from a common set of DevOps ideology that came into existence in companies like Netflix who were the first high profile companies to migrate from monolithic architecture to MSA in 2009. Since we have discussed in depth about the advantages of automation, it only makes sense to implement automation in also building the application. This includes fully automated pipelines from code repository in Github to production. Automation includes release pipelines with [continuous integration (CI)](https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration), automated testing, and [continuous delivery (CD)](https://learn.microsoft.com/en-us/devops/deliver/what-is-continuous-delivery). I wanted to build the application by following these DevOps ideologies thereby automating the whole application life cycle from planning to monitoring.
3. Monolithic Architecture works best for applications that face lesser loads as compared to applications that experience higher loads. For applications facing higher loads, MSA is the most suited type of architecture. Therefore to build scalable applications that don't require major modifications in the future, MSA is preferred over monolithic architecture [7]. The *Smart Slackbot* application may not experience higher load at present but in future there might be an increase in the users and hence the application may experience a higher load. Therefore it is knowledgeable to build a scalable application that doesn’t fail under higher loads. Also it is important to build an application that doesn't need too many modifications to accommodate more users in the future. Hence I decided that MSA is the best architectural choice for building the *Smart Slackbot* application.

###### **Empirical Contributions**

As the theoretical contributions come from theoretical analysis performed as part of my thesis approach in *section 1.2*, empirical contributions come from the practical work that I performed before writing my master thesis. This encompasses the two remaining concepts of the thesis approach: Natural Language Processing and Cloud Computing. Pure theoretical analyses may provide a hypothesis but there is no definitive proof that these concepts will work in real-life applications. This is where empirical work is so important to test the hypothesis and build applications and deploy it in the production environment where actual users can interact with your application. Before building the *Smart Slackbot* application, I had developed a similar application as a Proof of Concept (PoC) for the same company as part of my master program in one of the study modules. I used the same platforms for the PoC application that I am using for the *Smart Slackbot* application - Slack, AWS and Google DialogFlow ES. The functionality of this PoC application was to initiate slack workflows through a chatbot installed in a slack channel. I will explain in detail about the PoC application, PoC requirement, slack workflows, etc in chapter two of my thesis document.

This PoC became part of the empirical contribution in which I got hands-on experience in two main concepts - chatOps and IT security. Firstly, chatOps was a theoretical methodology that I had studied only through research papers and documentations but had never tested its validity. This was the case until I applied it personally in building the PoC application. Once the PoC application was successfully developed and deployed in the production environment it became evident to me how much chatOps contributes to improving the company process. The PoC application improved certain processes inside the company on a significant level which led to the definitive proof of my theory - “automation through chatOps is the solution to solving the company’s disconnected documents problem and removing any roadblocks in team collaboration” which I had hypothesised in chapter one - Introduction of my master thesis document. ChatOps also includes the topic of workflow automation which was another methodology tested in my PoC application.

Workflow refers to the series of activities needed to complete a task. Workflow automation refers to implementing software that can complete tasks which are managed manually. Automation software company ‘Zapier’ surveyed and found that 94% of small and medium-sized business workers said they perform repetitive, time-consuming tasks. Zapier also noted that 90% of knowledge workers found that automation improved their jobs and 66% said automation has made them more productive [8]. Automating workflows in the slack application is a significant project requirement for both the PoC application and the *Smart Slackbot* application from the company.

To successfully implement workflow automation in the PoC application, I followed the below 5 steps that I had learnt as part of the background research I performed before building the PoC application:

1. **Identify the process** - Taking a note of what all processes need to be automated in the organisation is the first step. Not all processes need to be automated. Only successful and working processes that add business value to the organisation needs to be automated. Automating broken processes is a waste of both time and resources for the automation engineer and also for the company because no business value is added at the end of the automation process.
2. **Map out workflow** - Mapping the entire workflow from end-to-end which needs to be automated is a crucial step to designing the automation software so that the software covers the entire workflow.
3. **Define business goals** - This is to understand the business goal in automating the workflow. As the person designing the automation software, you need to find the answer to the question “What does the business gain by automating this workflow?”. This helps in defining the business goal for the company.
4. **Research, choose and implement** - Analysing the existing softwares, tools, methodologies available in the market and selecting the right ones required to automate the workflows.
5. **Drive continuous improvement** - Even after the workflows are automated, continuous monitoring and feedback gathering helps improve the workflow automation.

These steps proved to be successful in building the PoC application which automated certain workflows for the company. Therefore I will be following these proven 5 steps to build the *Smart Slackbot* which also performs workflow automation in the slack application.

Second empirical contribution which formed the basis for my thesis is in the domain of IT security. One of the important concepts that I had really not focused on while building the PoC application was IT security. There was no implicit security system built to protect the PoC application or the data which was shared across different platforms. Since it was only a PoC, it did not have a major impact due to lack of security systems. This is not the case with the *Smart Slackbot* application as it is a complete, scalable application that real users will interact with. Therefore, I will be focusing more on all 3 types of IT security - infrastructure, data and network as it is important to protect the application data and user data from intentional and accidental breach. One of the steps that I will be following is to keep the software as compact as possible within easily migratable cloud areas, therefore it is easy for me to design a security system that covers all platforms used by *Smart Slackbot*.

Theoretical contributions may have provided me with the knowledge and information necessary to design my application but empirical contributions in the form of building the PoC application gave me the practical experience and definitive proof that by combining the tools and methodologies that I had studied in theory, I can build a full scale application that will work in the production environment. Both theoretical and empirical contributions formed the foundation for my master thesis.

## Thesis Structure

The master thesis is structured into 3 parts in order to sequentially explain the thought process and execution involved in building the *Smart Slackbot* application starting from planning stage till monitoring stage.

#### Part I – Research and Background

Part I of my master thesis covers the research and background work involved in planning the *Smart Slackbot* application. This part has two chapters, **Chapter one** gives a brief introduction to my thesis and explains the motivation behind choosing the topic for my thesis. Following the motivation, the thesis approach and thesis contributions are explained in detail in sections following motivation. Since it is a real-world project where the application will be actually deployed in the production environment and real-life users will interact with it, a lot of planning has to be done before the application is built. So I performed a comprehensive study of existing tools and methodologies. The results of these studies are only explained in ***section 1.2*** and ***section 1.3***. Chapter one ends with ***section 1.4***, the current section which describes the structure of my thesis. In **chapter two**, following a thorough literature study and background analysis, I explain the status quo of the *Smart Slackbot* application.

The prototype version of this application, as already mentioned, was already created in the previous semester as a Proof of Concept (PoC). This PoC was a much smaller version than the actual *Smart Slackbot* application which I am building for my master thesis. In chapter two, I explain what the existing PoC application does, covering the requirement document in ***section 2.1***, the PoC application’s architecture in ***section 2.2*** and its functionality in ***section 2.3***. Next, in ***section 2.4***, I will be explaining why the smart slackbot application was necessary despite having the PoC version of the application. In the final section of part I, ***section 2.5***, the existing processes and tools used in the company for collaboration and knowledge sharing is discussed. **Part I** ends with this section.

#### Part II – Plan and Infrastructure

Part I was all about the background and research work involved in creating the *Smart Slackbot* application, Part II is about the step-by-step building plan to develop the application. This includes requirement analysis and conversational user flows with DialogFlow chatbot. After discussing the building plan, I explain the architecture of all the services that are involved in building the *Smart Slackbot* application. This includes discussing the architecture of individual components like the Google DialogFlow, Slack, AWS. This is important to understand how each of these individual components function on their own and the reason why they were developed in the first place. Finally I discuss the complete Smart Slackbot application architecture, how all these individual components come together to form a single microservice based application. I also explain the role of individual components and what their contributions are in developing the application. Following which in ***section 3.5***, I will discuss in brief about the two external platforms used for building the *Smart Slackbot* application - Google DialogFlow as it is essential to understand what Google DialogFlow is and how it works. I finish Part I of my master thesis with ***section 3.6*** in which I explain the Slack architecture and its role as the frontend of the *Smart Slackbot* application

#### Part III – Implementation and Conclusion

Part I and Part II talked about building the application. Part III is all about implementation and conclusion. In the final part of the thesis, I talk about how the application after being built gets deployed into the company’s infrastructure. Implementation of the application is as important as building the application as this decides how well the application integrates into the company and if the application actually solves the problem that I am trying to solve. Apart from the implementation plan and final conclusion, I also talk about how the application can be monitored and improvised once the application is deployed to the production environment.

# 

**CHAPTER**

TWO

STATUS QUO

The *Smart Slackbot* is not a completely new application but is rather an improvisation on an existing Proof of Concept (PoC) that I had built in my previous semester. The PoC version of the *Smart Slackbot* had only 3 environments as part of its architecture: Slack Application, AWS and Google DialogFlow ES.

The slack application was used as the frontend (or) user interface for the employees of the company to interact with the PoC application. The backend was created using AWS and Google DialogFlow ES. DialogFlow ES was used for the NLU part which is similar to the current *Smart Slackbot* application. AWS was used for the infrastructure, computing and API services that were pivotal to building the backend of the application. The frontend of slack and the backend of AWS and DialogFlow ES were connected through webhook integrations which were triggered based on events.

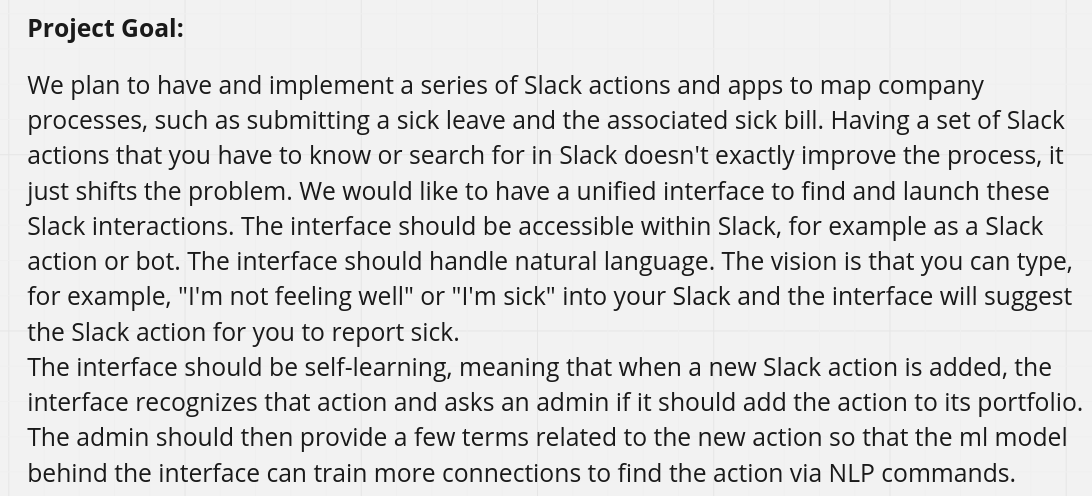
The PoC application was also built using Microservices Architecture (MSA). All the services that were built for the PoC were loosely coupled, independently deployed and were scalable whenever necessary. The PoC application was deployed to the production environment using the process of ***Infrastructure-as-Service*** (IaC). As the whole infrastructure is on AWS, I used ***AWS Cloud Development Kit*** (CDK) to deploy the whole infrastructure of the application using code. AWS Lambda function served as the computing service in the backend. It was written in python 3.8. The CDK stack which served as IaC was written in Typescript. This is the tech stack of the PoC application.

Before explaining in detail about the PoC application, there are few terminologies that need to be explained in order to understand all the components involved in both the PoC and the *Smart Slackbot* application. Slack in itself, is a massive instant messaging platform as mentioned in section *1.1*. Slack also offers in-builts apps which can be installed into any slack channel to perform certain tasks [9]. They can be found and installed from the app directory of the Slack application. Some examples of slack apps are: Google Calendar, GitHub, AWS etc. Apart from in-built apps, slack also allows users to build their own customised apps to suit the users’ specific needs [10].

For the PoC and for my thesis, I built a custom Slack App inside the slack platform which was used as the frontend for users to interact with the application. In the case of PoC, this slack app was used to perform the task of workflow initiation by triggering the workflow when a user requests for a particular workflow [11]. The next terminology related to the PoC and my thesis is slack workflows [12]. Slack workflows are a series of actions and reactions that can be triggered by the user manually or by an external application. These workflows guide users to perform step-by-step activities inside the slack application. I will explain in-depth about the slack apps and the slack workflows in Part II of the thesis document.

## Requirement Document for PoC

Below (img no.) is the project goal that was defined for creating the PoC application.



As you can see in the image above, the project requirement was provided by the company to create an application that implements a series of slack actions like initiating slack workflows. The goal was to initiate the right workflow automatically without users having to manually click on the workflow buttons. The project goal also requested for an app that can perform NLP actions and understand what the users say without the users having to explicitly mention a keyword to initiate the workflows. Based on the project requirement provided by the company as the problem statement, I came up with the PoC application whose architecture is as shown below:

## PoC Application Architecture





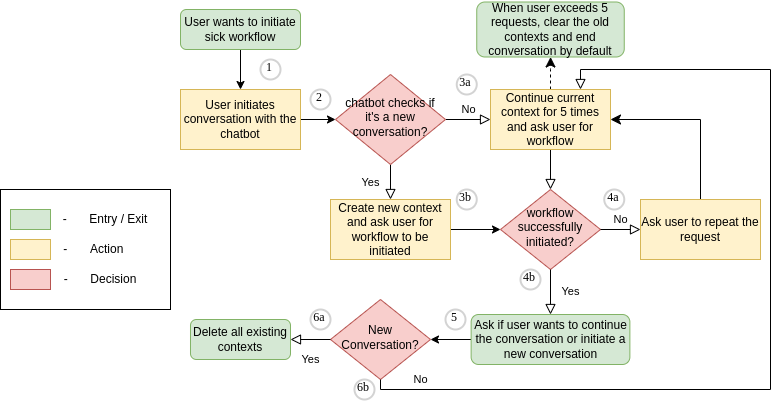


## PoC Application Functionality

The above architecture (img.no) shows how the PoC version of *Smart Slackbot* was built using just 3 cloud services apart from the slack application itself. There are 3 environments: Slack, AWS and GCP. Slack is responsible for the frontend or User Interface (UI). AWS provides the infrastructure necessary for the compute and API services. GCP provides the NLP engine necessary for the chatbot to function. These 3 environments integrate together to form the PoC application. Its functionality can be summarised in 5 steps. These steps are also marked in the architecture diagram above:

1. When users want to initiate a slack workflow (Example: a sick workflow), they initiate a conversation with the slack app installed in the company’s slack channel by sending a message to the slack app. A “sick workflow” is a custom built workflow in slack which informs users on the steps to be followed to apply for a sick leave in the company.
2. The slack app which is a custom built app, named ‘*Cool Integration Bot*’ is integrated directly with the DialogFlow ES chatbot by sharing the slack app’s client ID, client secret, and verification token. This information is unique to each slack app and is generated when a new slack app is created in the webpage: [*api.slack.com*](https://api.slack.com/). When users send a message to the slack app, the message is sent directly to the DialogFlow ES chatbot. The DialogFlow ES agent understands that a user is initiating a conversation based on its NLU capabilities and responds to the message sent by the user.
3. The response from the DialogFlow chatbot is sent directly to the AWS environment through a webhook integration and not to the user in the slack app. This is because a decision needs to be made whether to initiate a slack workflow or not based on each message sent by the user through the slack app. This decision is made in the AWS environment. The AWS environment includes 2 cloud services: Amazon API Gateway service and AWS Lambda service. API gateway acts as the webhook link between DialogFlow ES agent and AWS environment. The API Gateway, in this case, is a REST API that is allowed to perform 2 methods - *GET* and *POST*. The REST API URL is provided to DialogFlow ES so the agent’s response is sent directly from the GCP environment to AWS. The DialogFlow sends the user message along with other conversational data as a *POST* method to the API Gateway. The API Gateway is connected to AWS Lambda service which gets invoked on receiving the JSON payload from the API Gateway as POST method. Now the conversational data is sent from GCP to AWS lambda. AWS lambda function is used as the compute service that has the code to decide whether to initiate the slack workflow or not that the user had requested. The lambda function is written in python 3.8.
4. Once the AWS lambda function receives the JSON payload, the decision to initiate a workflow or not is made in the lambda function based on the value of a json parameter value - ‘*workflow*’. This ‘*workflow*’ parameter is sent from DialogFlow ES to AWS lambda and based on the value of this parameter, the lambda function decides whether to initiate the workflow or not. There are 3 workflows that I had created in DialogFlow ES, each for one for a separate use case - Sick, Vacation and Invoice. These 3 use cases were provided as a project requirement by the company. The lambda function checks for the 3 previously mentioned use case values - sick or vacation or invoice in the ‘*workflow*’ parameter coming from DialogFlow and initiates the respective workflow based on the ‘workflow’ parameter value. If the workflow parameter value does not match any of the 3 use case values then the response is directly sent to the user without initiating any slack workflow. The response to be sent to the user in this case is decided by the DialogFlow ES agent using its advanced NLU capabilities. AWS lambda service is only used to trigger the slack workflows by invoking the slack workflow’s webhook URL. Slack workflows have their own webhook URLs which when invoked initiates the slack workflow in the slack channel. AWS lambda service sends all the responses received from the DialogFlow ES to the user in the slack app through another webhook integration. This is irrespective of whether the slack workflow is initiated or not.
5. The user who initiated the conversation in the slack app awaits for a response and the response comes through the lambda function. If the workflow is successfully initiated then the user will see the workflow steps directly on the slack app. If for some reason the workflow is not initiated then the user also gets the response sent by the DialogFlow agent asking for the user to repeat their request or to end the conversation if the workflow is successfully initiated and the user is satisfied. This is the end-to-end functionality of the PoC application that I had created before starting my master thesis.

Below is a flowchart diagram (diagram no) explaining the conversational flow between the user and the DialogFlow ES agent.



The flowchart is created to display the end-to-end interaction between the user, slack app, AWS and the DialogFlow ES agent. The flowchart is also colour coded for easy comprehension of the dialogue structure between the user and the highly advanced NLP based DialogFlow ES agent. There are 3 major components in the flowchart below: The green component denotes the entry/exit points where the user initiates or ends the conversation with the chatbot. Yellow rectangular components are the actions taken by the DialogFlow agent based on user input or existing context of the conversation. Pink diamond components denote the decision taken by the DialogFlow ES agent based on the user actions. The flow of conversation in the flowchart is also numbered (from 1 to 6) for the readers to understand the dialogue flow between the user and the chatbot. Flow points 3,4, and 6 are split into 2 as (a) and (b) because these flow decisions are based on the binary choice of ‘yes’ or ‘no’ decisions taken by the chatbot based on the user responses or existing context in the conversation. When a conversation ends, all the context of the conversation is deleted and a new context is created for each new conversation.

## Smart Slackbot

The reason for the PoC version being just the first step and not a full application is because it is only semi-automated. Even though the slack workflow initiation process, in the backend (after a user initiates the conversation with the chatbot), is automated, it is not as effective as full automation. Users still have to manually go to the slack app and initiate a conversation to perform the slack workflow initiation. This was a concern for both the company and for me which led to the motivation of building a fully automated version of the PoC application - the *Smart Slackbot* application, which can perform multiple tasks such as knowledge distribution by redirecting users to the correct information they need and also by providing the option to initiate slack workflows similar to the PoC version. In the next and final section of chapter one, I will discuss the existing processes and tools used in the company for maintaining their knowledge base.

## Existing Processes and Tools

I will discuss the existing processes and tools used by the company for documenting business processes, team collaboration, and maintaining knowledge base about projects, teams, organisational procedures and other essential information about the company.

### Confluence:

The company uses “*confluence*” as its corporate wiki. Confluence is a web-based corporate wiki developed by the Atlassian software company [13]. Confluence is used widely in companies around the world to break down team silos and optimise everything in one place. It is an important player in the knowledge management domain and is prevalent in the area of software development. The company’s knowledge base regarding projects, budgets, team members, onboarding and offboarding employees, etc is persisted in the company’s confluence website. Confluence is also used for writing and maintaining ‘*How-to*’ articles regarding projects, like how to use AWS, how to write clean code, information about committing to and maintaining version control systems, etc by the company. Apart from technical knowledge, the confluence also contains information about employee services such as applying leaves/holidays, sending invoices for reimbursement of bills, employee onboarding steps, etc. In the context of *Smart Slackbot* application, the company confluence is the main source of destination where the user is redirected to whenever a request is made through the slack application.

### Jira:

Jira is a work management tool developed by the same organisation that developed Confluence, Atlassian [14]. Jira is mainly used as a tool for team collaboration. *OEV Online Dienste GmbH* uses Jira for ticket management, logging and resolving bugs, project management activities such as organising agile meetings, maintaining kanban boards, etc. Since *Smart Slackbot* is developed as a project within the organisation therefore it follows the same documentation methodologies as followed in other projects of the company.

### Processes:

Apart from the collaboration tools and the corporate wiki used by the company, it is necessary to briefly explain how the organisational process for the below 5 use cases happen at the organisation. This is because the current version of the *Smart Slackbot* will be automating the below 5 use cases as part of my master thesis:

* Applying sick leaves (Krankmeldung)
* Applying vacation leaves (Urlaub Erfassen)
* Submitting invoices (Rechnungen)
* Initiating project ideas (Meine Idee)
* Requesting project details (Projekte)

###### Applying Sick Leaves

The sick leaves can be applied in the company using slack workflows. There is a slack workflow called *Krankmeldung* which is created in a private slack channel of the company where one can initiate the workflow and the workflow explains the consecutive steps necessary for the users to complete the sick leave application. The problem in this process is that new employees or employees who have not applied sick leaves before have no idea about the slack workflow. People may not know such a workflow even exists for applying sick leaves. Some experienced employees might also forget how they initiated the sick workflow as people don’t apply for sick leaves quite often. The employees may also not be aware of which slack channel(s) this workflow is enabled in. Hence it becomes both time consuming and tedious for employees to search and find where and how to initiate the krankmeldung workflow even if they are aware of the workflow. There is a confluence page called Krankmeldung which explains the process of how krankmeldung workflow works but finding this confluence page among multiple confluence pages in the company wiki is a separate challenge for the employees.

This is the problem that the Smart *Slackbot* applicationis trying to solve. *Smart Slackbot*, when activated, provides the user with 2 buttons: one for redirecting the user to the confluence page containing the information on how the krankmeldung workflow works and the second button to initiate the workflow directly from the chat window so that the user does not have to search for the krankmeldung workflow.

###### **Applying Vacation Leaves**

Vacation requests are made similar to the sick leaves by using a slack workflow called *Urlaubsmeldung*. Hence, the same problem as *Krankmeldung* applies to vacation requests too. There is a separate confluence page called Urlaub Erfassen containing information on how to apply for vacation leaves and what are the steps involved that an employee must follow in order to record their vacation requests. So the role of *Smart Slackbot* for this use case is similar to applying sick leaves. *Smart Slackbot* uponactivation will provide 2 buttons, one to redirect the users on the slack channel to the confluence page, the other to initiate Urlaubsmeldung workflow directly.

###### Submitting Invoices

Submitting invoices for reimbursement of company expenses can be done by sending the bills to one of the employees in the company. There is a process defined in a confluence place called Rechnungen that explains how much can be reimbursed without further approvals and how many approvals are needed for purchases exceeding a certain value. The role of *Smart Slackbot* for this use case will be to provide 2 buttons similar to the previous 2 cases but for submitting invoices there are no workflows built in slack. Therefore, the first button will be to redirect the users to the Rechnungen confluence page and the second button will be to directly contact the employee responsible for rechnungen through slack.

###### Initiating Project Ideas

New project ideas are essential for companies to develop innovative solutions and build on their existing portfolios. When an employee comes up with an idea for a new project, be it an internal or external project, it is essential for the company to heed and discuss them in order to explore any value in the idea. For this purpose, a separate slack channel is created by the company where new project ideas can be submitted and discussed. The role of *Smart Slackbot* for this use case will be to redirect the users from the general slack channels where everyday conversations happen to the specific slack channel created for the purpose of sharing new ideas. This is done so that the ideas can be shared and discussed by the people concerned and at the same time the conversation is not lost in the everyday conversations of the general slack channel.

###### Project Details

All the previous use cases had a task for the user - either to submit or to apply, but the project details use case is different from the other use cases. In project details, the user requests for information regarding a specific project that is ongoing in the company. The project details for all projects being developed by the company are stored in the confluence page called *Projekte* where users can go and find the information for the project they are looking for, but the same problem of finding the right confluence page among multiple confluence pages exist and at the same time searching for the project details is a time consuming and tedious task even inside the project details. The role of *Smart Slackbot* is to help automate this task byretrieving the information from the confluence page and sharing it on the slack channel directly with the user. When a user mentions a particular project number in the slack channel, the *Smart Slackbot* will fetch the details of this project number and display the details directly to the user. The user does not have to go to the confluence page and search for the project details.

I have covered all the non technical aspects of my master thesis - the *Smart Slackbot* application in *Part I* of the document. In Part II, I will explain the technical aspects of the *Smart Slackbot* application including the requirement analysis, step-by-step plan for building the application, architecture of the application and individual components used for building the application.

# 

**CHAPTER**

THREE

DEVELOPMENT PLAN

An overview of the *Smart Slackbot* application from a non technical perspective was provided in the first two chapters of my thesis document. This was to give an introduction to the application and to provide the necessary background information. Apart from the background, findings and analyses of both the theoretical and empirical contributions were also discussed in the previous chapters. We enter chapter three of my thesis document with the below finalised technical configurations from the previous chapters for the *Smart Slackbot* application:

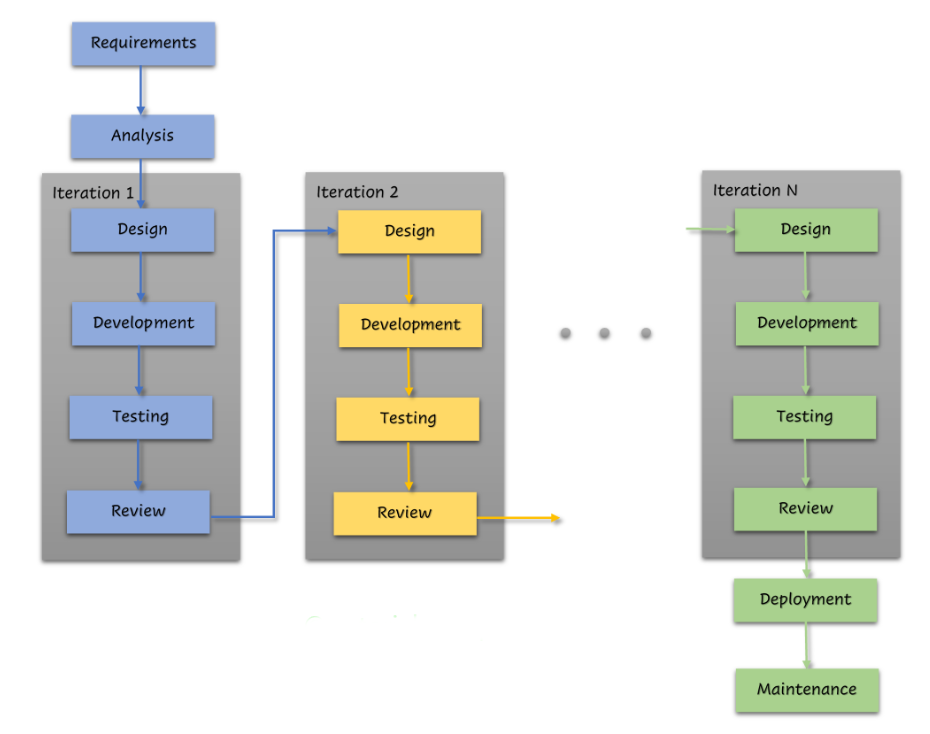
* Platforms used - Slack, AWS and Google DialogFlow ES
* Architecture - Microservices Architecture (MSA)
* Methodology - ChatOps, SDLC
* Tech Stack - Python, Typescript, HTML, Javascript, CSS

In this chapter, I will be focusing on how the application was developed by following a methodology that is most commonly used in modern software engineering - the Software Development Life Cycle (SDLC). In section 3.1, I discuss the SDLC model I had used for developing *Smart Slackbot* and why I used that model. After explaining the SDLC model, I introduce the topic of System Requirements Analysis (SRA) in section 3.2 where I list out the requirements analysis I performed for *Smart Slackbot*. Following SRA, conversational user flows had to be decided. This is because the decision to structure how the conversation should be between the user and the DialogFlow ES agent needs to be made as early as possible. This will help in defining the role of the DialogFlow ES agent in the whole application. It will also help in finalising the architectural design of the application. The final section of this chapter will cover the step-by-step plan that I followed for building the *Smart Slackbot* application.

## **Software Development Life Cycle (SDLC)**

The SDLC is a structured process that enables production of high quality, low cost software in the shortest possible production time. The goal of SDLC is to produce superior software that meets and exceeds customer expectations and demands. SDLC defines and outlines a detailed plan with phases, that each encompass their own process and deliverables. Adherence to the SDLC enhances development speed and minimises project risks and costs associated with alternative methods of production. There are various types of SDLC models that can be used to develop software. Some of the common ones are Waterfall model, Iterative model, V-shaped model, Spiral model, Agile model, etc. The SDLC model I used for developing Smart Slackbot is the iterative model. Iterative models provide faster results, require less up-front information, and offer greater flexibility [15]. This is because the development team does not require the full specifications or requirements of the product for building it. Instead, development begins by specifying and implementing just parts of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model [16].

The iterative model of SDLC can usually include 6 phases: Requirements Gathering & Analysis, Design, Implementation, Testing, Deployment, Review and Maintenance. As you can see in the image below (img no.), Requirements Gathering & Analysis is the first stage of iterative SDLC model, followed by multiple iterations of software development, testing and review until the product is finalised for production. When the desired end product is developed in the Nth iteration, the product is deployed to the production environment and the monitoring & maintenance phase begins. This is the sequence of the steps that I had followed in developing the *Smart Slackbot*. One of the major steps that I will explain as part of the iterative SDLC methodology is System Requirements Analysis which is the first phase. This is explained in the next section.



## System Requirements Analysis

Requirements are necessary attributes defined for a software prior to the efforts of developing its design. System Requirements Analysis (SRA) is a structured or organised methodology for identifying an appropriate set of resources to satisfy a system need and the requirements for those resources that provide a sound basis for the design of those resources [17]. SRA acts as a transformation between the customer's needs and the design developed by the architect. SRA is the first phase of the iterative SDLC model and is performed at the very beginning of a project. SRA process consists of the below 5 steps:

###### Step 1: Identifying Key Stakeholders and End-users

This is the first step of the requirements gathering or analysis phase. In this step, we identify and define who the stakeholders are and who the end users will be for the project. In the case of *Smart Slackbot*, the stakeholders are the company that sponsors the project and my master thesis since the company defines the scope of the project. The end users will be the employees of the company as *Smart Slackbot* is an internal application used within the company to improve the knowledge base accessibility and team collaboration. The goal of the project is to cover the scope defined by the stakeholders and should satisfy the needs of the end users. In this step we have identified who the stakeholders and end users are for *Smart Slackbot*.

###### Step 2: Capture Requirements

In this step, we capture the requirements from the stakeholders and the end users. Both their inputs are necessary for defining the scope of the project before starting the project. This can be done by holding one-on-one interviews with the stakeholders and end users. This helps in gathering the requirements directly. We can also ask for use cases from the end users as this provides the walkthrough of the application end-to-end from the eyes of the users. Third option is to build prototypes similar to the requirements of the users as this will help address feasibility issues and identify roadblocks ahead of time. For *Smart Slackbot*, I performed both one-on-one interviews with the stakeholders from the company and also built a prototype as a PoC (as explained in chapter 2). These two methods helped me capture requirements accurately that are necessary for the project.

###### Step 3: Categorise Requirements

The capture requirements step includes capturing both user requirements and system requirements. These two requirement documentations combined together help in officially registering the agreed requirements for the project between the developer and the stakeholders. User requirements document is a non technical document and system requirements document is a technical document.

User requirements are created to define the idea behind building the software. They do not have any technical information and are often written in simple business language. The user requirements should not define how the system works but rather should state the clear purpose of the software to be developed. This document is written by the developer and is approved by the stakeholder. The development process does not begin without getting the final goal of the product approved. This is because without a clear purpose or a goal, the software developed could have various functionalities but miss out on what it was intended to do. This is precisely stated in the user requirements document which has the clearly defined goal approved by both the stakeholders and the developer. Therefore all the work dedicated to building the software can be directed toward this goal and any modifications suggested by the stakeholders gets added to the later versions of the software and not the initially approved end product.

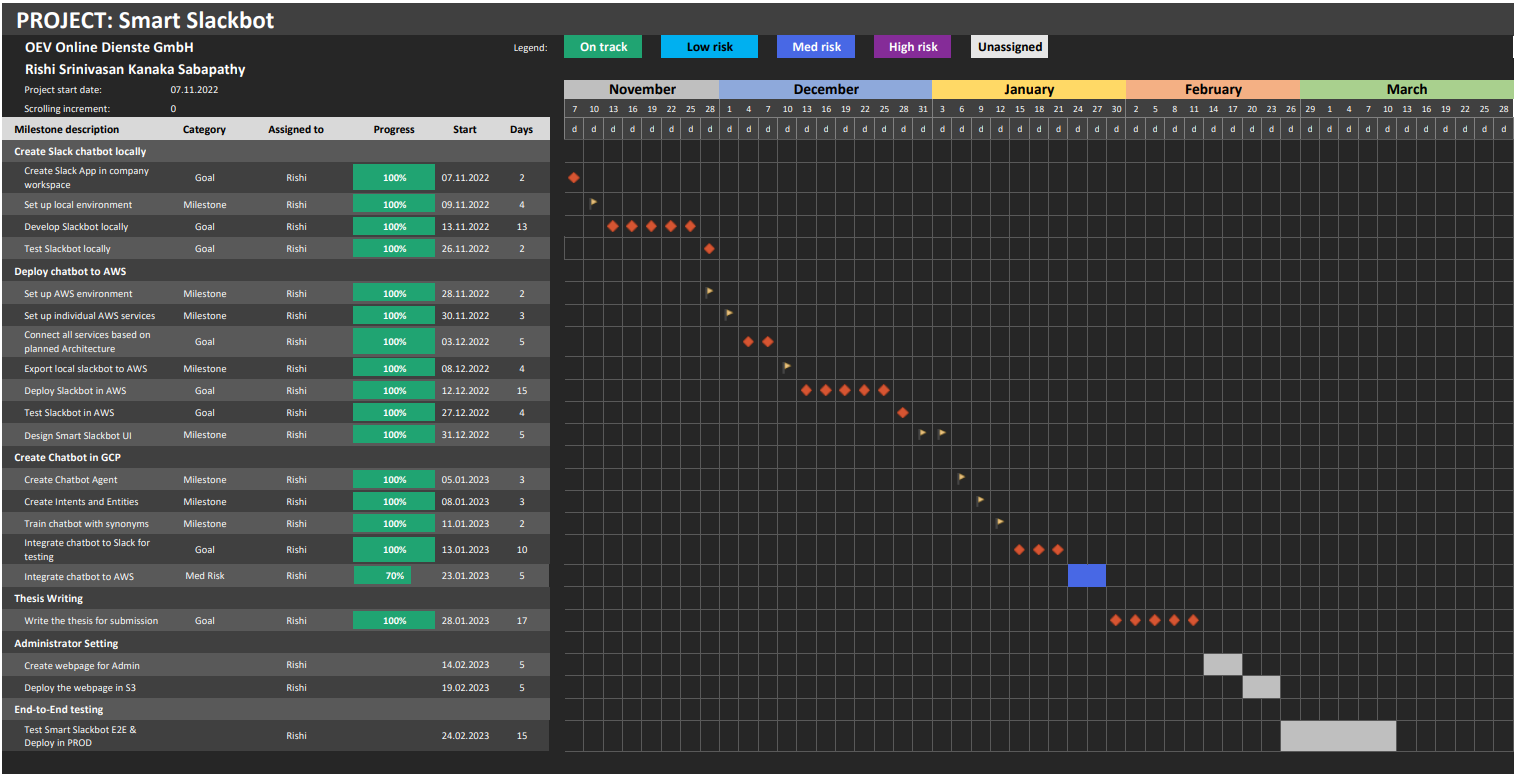
On the other hand, system requirements are technical information that are more clearly and rigorously written. These requirements are written within the development team to get a clear overview of the technical specifications of the software to be developed. User requirements are collected in a document called Functional Requirements Document (FRD) and system requirements are collected in a document called Technical Requirements Document (TRD). Both FRD and TRD are important documents in building software because they define the requirements and scope of the project at the same time they also define the technologies and components used to cover the requirements and scope. Both FRD and TRD are created for the Smart Slackbot. Since they are separate documents they are stored in a public repository and can be found in the github repository link below:

FRD:

TRD:

###### Step 4: Interpret and Record Documents

Once the necessary requirements are categorised, determine which ones are achievable within the time period and which requirements are planned for the consecutive versions of the software. This confirms the feasibility of the project and allows the developer and the stakeholders to discuss potential risks involved in the project. The finalised requirements and time frames for each requirement helps in building the Minimum Viable Product (MVP) and also the full version of the software in the planned time period and as per the stakeholder requirements. This can be done using the Gantt charts. Gantt charts are



###### 

###### 

###### 

###### Step 5: Sign off

UML Diagrams

UML diagrams shows how objects in a system or classes within code interact with each other. They show in sequence of events by ordering the events that are taking place in a system.

## Conversational Userflows

How the conversation is designed for the application. If a user types “abc” then the chatbot responds with "xyz".

## Step-by-Step Plan

Explain step-by-step plan on how the application will be built. Timeline, sprint plan, Continuous revisions and modifications

# 

1. **CHAPTER**
2. FOUR
3. ARCHITECTURE

Overview of the importance of having a clear well defined architecture. Changes from previous architecture.

## Google DialogFlow

Architecture of Google DialogFlow

About the NLP platform which can be integrated into a conversational user interface

In this section, I discuss what Google DialogFlow is and how it is configured for our Smart Slackbot.

## Slack Application

Architecture of Slack Application and Chatbots in Slack (not needed to define why Slack)(explain why slack, why aws, why dialogflow)

About Slack, Slack app, integration with AWS and Google DialogFlow

## Amazon Web Services

Architecture of AWS services used and how it connects to both Slack and Google DialogFlow

## Smart Slackbot Architecture

Architecture of entire application end-to-end

**IMPLEMENTATION**

Implementation Plan and how the application will be integrated into our slack channels

## Application Implementation

How users can interact with it, multiple ways it can be deployed

## Monitoring and Improvements

Continuous Monitoring and Continuous improvements

NLU

Data ethics in Dialogflow and AWS, GDPR, prof.Helena questions

# THEORETICAL BACKGROUND (Nico: move it to last)

Publications, Journals and Research papers studied for this thesis. Multiple sources mentioned and choosing a side to support and why

## Scientific Research on Company Wiki

What are the advantages and disadvantages in using the company wiki. Existing research work on company wiki

## Natural Language Understanding

About NLU in general, research on LLM, Advantages, biases and disadvantages in conversations

## Microservices (if necessary)

Rename and think about it later

# CONCLUSION

Conclude the thesis and state how the application will improve the company process (or not!)

# Statutory Declaration

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper.

I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content.

I am aware that the violation of this regulation will lead to failure of the thesis.

**Rishi Srinivasan Kanaka Sabapathy**

Student’s name Student’s signature

**S0581565** **14.02.2023**

Matriculation number Berlin, date

# REFERENCES

Below is the list of references used in my thesis document. These references refer to scientific papers, journals, documentations and articles from the internet. I have used the APA citation style for citing the references consistently throughout the thesis document.

1. Webster, M. (2015). Last Accessed: April 2015. Bridging the Document Disconnect in Sales, An IDC White Paper, Sponsored by Adobe. Adobe.  
   <https://esign.adobe.com/rs/345-TTI-184/images/idc-bridging-the-document-disconnect-sales.pdf>
2. Schaubroeck, R., Holsztejn Tarczewski, F., Theunissen, R. (2016). Last Accessed: March 3, 2016. Making collaborations across functions a reality. McKinskey Quarterly - McKinskey & Company.   
   <https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/making-collaboration-across-functions-a-reality>
3. Regan, S. (2016). Last Accessed: January 22, 2016. What is ChatOps? A guide to its evolution, adoption, and significance. Atlassian. <https://www.atlassian.com/blog/software-teams/what-is-chatops-adoption-guide>
4. Adamek, Á., Farkas, K., & Szabó, G. (2022). Last Accessed: May 29, 2022. Cloud Native ChatOps Development. HÍRKÖZLÉSI ÉS INFORMATIKAI TUDOMÁNYOS EGYESÜLET (hte).  
   <https://www.hte.hu/documents/10180/4734660/Adamek_Adam-Dolgozat.pdf>
5. Goasduff, L. (2022). Last Accessed: April 19, 2022. Top 10 Automation mistakes to avoid. Gartner, Inc. <https://www.gartner.com/en/articles/10-automation-mistakes-to-avoid>
6. Goel, A. (2021). Part 4 Including DevOps in microservices migration. Capgemini Engineering.   
   https://capgemini-engineering.com/us/en/insight/part-4-including-devops-in-the-microservices-migration/
7. Gos, K., & Zabierowski, W. (2020, April). The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE.   
   https://www.researchgate.net/profile/Wojciech-Zabierowski/publication/341956559\_The\_Comparison\_of\_Microservice\_and\_Monolithic\_Architecture/links/5edf80fe299bf1d20bdb24e2/The-Comparison-of-Microservice-and-Monolithic-Architecture.pdf
8. Zapier Editorial Team. (2021). Last Accessed: April 20, 2021. Zapier Inc. Zapier report: The 2021 state of business automation. https://zapier.com/blog/state-of-business-automation-2021/
9. Slack Inc. Guide to apps in slack. https://slack.com/help/articles/360001537467-Guide-to-apps-in-Slack
10. Slack Inc. Planning your slack app. https://api.slack.com/start/planning
11. Slack Inc. Create more advanced workflows using webhooks.  
    https://slack.com/help/articles/360041352714-Create-more-advanced-workflows-using-webhooks
12. Slack Inc. Guide to workflow builder.  
     https://slack.com/help/articles/360035692513-Guide-to-Workflow-Builder
13. Confluence (software). Last Accessed: November 24, 2022. Wikipedia.   
    https://en.wikipedia.org/wiki/Confluence\_(software)
14. What is Jira used for?. Last Accessed: January 28, 2023. Atlassian Corporation.   
    <https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for>
15. Munassar, N. M. A., & Govardhan, A. (2010). A comparison between five models of software engineering. International Journal of Computer Science Issues (IJCSI), 7(5), 94. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3a4a2cb2328e2f416be0be012e5d580975943554#page=115>
16. Rastogi, V. (2015). Software development life cycle models-comparison, consequences. International Journal of Computer Science and Information Technologies, 6(1), 168-172. https://d1wqtxts1xzle7.cloudfront.net/40003520/ijcsit2015060137-libre.pdf?1447530769=&response-content-disposition=inline%3B+filename%3D37\_SDLC\_comparison\_consequences.pdf&Expires=1675318114&Signature=edv9WD-Trk5YpgSgE3IqCg2251S4tAtgJLRqkXXKCan1p8rEZWMYLmTte-3r439BMV1RcWUioEBXxoTm6Q~h3TsIRiY1MvK5nMzKYCGz~1tDkeA3WFLBBDDY-Gy7s9yigjCBh7-1jDKrcARSu8604yxl0-n9xNhiOdUXvbyurB7CR9cZbzW-dWrbIZawSpg9B5lfO0mxbsUewn7Pp2Aq-BP8XlIoFUOvrHi3YUch0rsM-o6XppkJY9TcQAoauUkznTlCUrQe-yOPkZduzEvQVwDgmLr-uhF1YEi~09hP~vYPM-DYXF9bFHes8CIHQH49BTCZehdDGIDYTRNtTEtKFQ\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
17. Grady, J. O. (2010). System requirements analysis. Elsevier.  
    https://books.google.de/books?hl=en&lr=&id=FkpqAnHUNLYC&oi=fnd&pg=PP2&dq=requirements+analysis&ots=\_SjXCtvuO\_&sig=KF64GWBAglec3N5CSL6ue1nftn4&redir\_esc=y#v=onepage&q=requirements%20analysis&f=false
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Last Accessed: December 4, 2017. Attention is all you need. Advances in neural information processing systems, 30.   
    <https://arxiv.org/pdf/1706.03762.pdf>
20. Romao, M., Costa, J., & Costa, C. J. (2019). Last Accessed: June 22, 2019. Robotic process automation: A case study in the banking industry. In 2019 14th Iberian Conference on information systems and technologies (CISTI) (pp. 1-6). IEEE.   
    <https://www.researchgate.net/profile/Carlos-Costa/publication/334482701_Robotic_Process_Automation_A_Case_Study_in_the_Banking_Industry/links/6034f0c2299bf1cc26e4c057/Robotic-Process-Automation-A-Case-Study-in-the-Banking-Industry.pdf>

Al-Debagy, O., & Martinek, P. (2018, November). A comparative review of microservices and monolithic architectures. In 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI) (pp. 000149-000154). IEEE. <https://arxiv.org/pdf/1905.07997.pdf>