****

**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 7**

1.1 Motivation8

1.2 About Master Thesis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.3 Thesis Structure. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.3.1 Part I - Research and Background. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.3.2 Part II - Plan and Infrastructure. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.3.3 Part III - Implementation and Conclusion. . . . . . . . . . . . . . . . . . . . . . . .10

1.4 Collaboration Tools Used. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.1 Slack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.2 Dialogflow. . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.3 AWS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.4 ChatOps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.5 Microservice Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.5 Thesis Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.5.1 Theoretical Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.5.2 Natural Language Processing. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.5.3 Cloud Computing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.4.4 ChatOps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.1 Theoretical Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.1.1 Business Process Automation . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.1.2 Microservice Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.2 Empirical Contributions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.2.1 Proof-of-Concept . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.2.1.1 Workflow Automation . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

1.6.3 IT Security. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

**2** **Status Quo 22**

2.1 Requirement Document for PoC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

2.2 PoC Application Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.3 Slackbot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

2.5 Existing Processes and Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

**3** **Development Plan 33**

3.1 System Requirements Analysis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

3.2 Conversational Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

3.3 Step-by-Step Plan. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

**4** **Architecture 10**

4.1 Google Dialogflow **11**

4.2 Slack Application **12**

4.3 Amazon Web Services **13**

4.4 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-Code

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

BPMN Business Process Model and Notation

# 

**CHAPTER**

ONE

INTRODUCTION

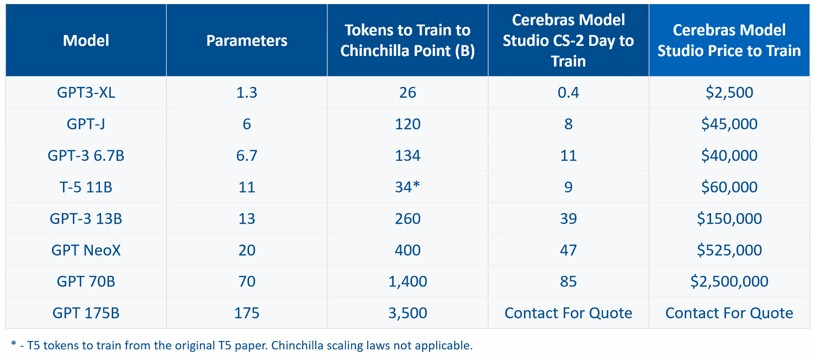
Organisations of all sizes struggle with the challenge of disconnected document processes, a pervasive problem whose negative impact cuts across all business functions [1]. With fast-changing global markets, companies struggle to break down silos and boost cross-functional collaboration [2]. This is especially a major concern in startups and small organisations where cross-functional collaboration becomes more challenging. Unlike companies like Google and Amazon who 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 employees, and to 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 [24]. This includes creating a culture of automation which is essential to an organisation’s growth and success [25]. Collaboration among teams is as important as automation or building quality software. 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 [26]. Based on these fundamental principles, the key inference is to avoid bottlenecks and enable smooth software development and flow of information among cross-functional teams by automation through chatOps. ChatOps is a collaboration model that connects people, tools, processes and automation into a transparent workflow [3]. The goal of the thesis is to prove that with the help of chatOps the disconnected documents problem that the company faces can be solved through continuous automation and collaboration in a transparent manner.

In this thesis, I first explore the problems that *OEV Online Dienste GmbH* face due to disconnected document processes and roadblocks in team collaborations which affects productivity. Then I 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. Following which I implement the developed solution into the company’s cloud native environment and integrate it seamlessly 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 are essential to building and deploying scalable software applications. This infrastructure is expensive and difficult to maintain in a private on-premise set up. The public cloud is significantly cheaper than the private cloud [27]. Companies like Google have also opened up their Natural Language Processing (NLP) platforms like Dialogflow where you can build highly advanced chatbots with lifelike conversational AI and integrate it into any platform [28]. Building private Large Language Model (LLM) such as Google’s Bidirectional Encoder Representations from Transformers (BERT), which is used to create sophisticated chatbot agents in Dialogflow [28] is close to impossible at the cost at which Google offers its Dialogflow services for public use. There are companies like Cerebras and Jasper that even offer their supercomputers as a service for Graphic Processing Unit (GPU) AI training [29]. Below is a sample price quotation for the cost of training your LLMs on a public supercomputer with Cerebras. Based on the model which needs to be trained, the costs could vary between $2500 to $2,5 million. 

This is really expensive as compared to Google Dialogflow costs which is around $0.002 per request [30]. Finally, slack, despite being a massive instant messaging platform on its own which provides professional and organisational communication service to thousands of companies [31], offer a variety of possibilities to integrate these external services from AWS or Google Dialogflow ES into their platform and send messages using webhooks [32]. Not only infrastructure-based and collaboration-based advancements have come up in the tech industry in recent years but there have also been architectural changes such as migrating softwares from monolithic architecture to Microservices architecture (MSA) have modified the way we build modern scalable applications [33]. 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.

## About Master Thesis

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 challenge that I am facing 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 company wiki in order 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, holidays or submitting invoices or initiating a new project idea, etc.

In order to automate this process, improve knowledge base visibility and boost employee productivity, I am building a slackbot application 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 slackbot application and the topic of my master thesis. 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 Structure (Write at last)

Now that the introduction, the motivation and an overview of the master thesis is covered, this section will explain the structure of the master thesis. The master thesis is structured into 3 parts in order to sequentially explain the thought process and execution involved in building the slackbot application starting from planning stage till monitoring stage.

#### Part I – Research and Background

Part I of the master thesis covers the research and background work involved in planning the slackbot application. This part has two chapters, **Chapter one and Chapter two**. Chapter one has 6 sections in total. The chapter starts by giving a brief introduction to the thesis. Following which the sections begin: ***Section 1.1*** explains the motivation behind choosing the topic for the master thesis. ***Section 1.2*** gives an overview of what is being built for the thesis. ***Section 1.3*** is to explain the structure of the thesis so that the readers can understand what each chapter and section is about. ***Section 1.4*** is to list and explain about the collaboration tools used in the company currently. ***Section 1.5*** is the approach used to write the thesis. ***Section 1.6*** talks about the contributions involved in writing the thesis. Chapter one is completed at the end of this section.

Chapter two has 4 sections in total. It begins with an introduction to what the chapter is about followed by ***Section 2.1*** which

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 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 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 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 slackbotapplication, 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 slackbotapplication. 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 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 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 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.

## Collaboration Tools used

This section consists of the list of collaboration tools used in the company currently and the ones that the master thesis is introducing into the company. These collaboration tools are discussed in this section along with their purpose.

### Slack

Slack is a

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 slackbotapplication. 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.

### Dialogflow

Google Dialogflow has 2 editions: Customer Experience (CX) and Essentials (ES). Essentials (ES) agent was chosen for the master thesis 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.

Dialogflow 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*.

### AWS

accordingly. It is also

### ChatOps

### Microservice Architecture

Microservices Architecture (MSA) was chosen over other types of architectures for the following reasons:

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.

I wanted to implement the DevOps ideology from the early stage of Software Development Life Cycle (SDLC) of the slackbotapplication 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.

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 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 slackbotapplication.

## Thesis Approach

The approach to building the slackbot application and ergo my thesis approach, is briefly divided into 3 parts:

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

### Theoretical Analysis

First is the theoretical analysis part where a comprehensive background study of the existing collaboration tools, corporate wikis, organisational process optimization methods and chatOps is performed through various research papers, journals and documentations. The reason why this analysis was performed before the design phase of the application’s architecture was to learn about the state-of-the-art technologies that are already available and are used in today’s market. Thereby selecting the most suitable tools and methodologies that would fit the requirements criteria in solving the company specific problem and not just a generalised solution. The documentations helped understand the capabilities and functionalities of existing collaboration tools. The research papers gave an in-depth knowledge of how certain methodologies like ChatOps can significantly improve an existing system through automation [4]. The theoretical analysis, as a whole, proved to be valuable as it offered an idea of how the system can be designed and also provided the scientific validation that certain methodologies and tools were successful in the past in solving a similar challenge.

### Natural Language Processing

Second part of the thesis approach is the Natural Language Processing (NLP) part where the best suited conversational Artificial Intelligence (AI) platform had to be chosen to build the chatbot. Once chatOps was selected as the methodology to building a conversational experience with the slackbot, there were two options available - Option 1 was creating the LLMs from scratch using a transformer model for the NLU task and building a chatbot on top of it. This would mean that LLMs had to be trained privately with large amounts of text data in order to have a good conversational experience with 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 the necessary large datasets were to be collected in order to train the LLMs privately, it would still have to be labelled and fine tuned regularly with new data in order to build a good responsive NLP based chatbot. Apart from these challenges, there was also the need for intensive computing power like Graphic Processing Units (GPUs) and Tensor Processing Units (TPUs) in order to train the LLMs quickly and efficiently.

Even if we assume that the necessary data was collected, labelled correctly and trained on new data regularly with the necessary computing power, there is no guarantee that the privately developed LLMs will outperform the existing transformer based models that are already available in the market like Google’s BERT or OpenAI’s GPT-3. All of this seemed time consuming, ineffective and more expensive when there were already good LLMs available for private use offered by public cloud providers like Google on their Google Cloud Platform. Therefore, option 2, which was to use an existing conversation AI that allows users to build chatbots on top of it. This was provided by Dialogflow as part of Google Cloud Platform service. The cost was also inexpensive as compared to building and training private LLMs. Therefore it was the clear choice considering the time, cost and efficiency in building the slackbot application.

At this point, the application architecture was already starting to take shape. It was 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.

### Cloud Computing

The third and final part of the thesis approach was 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, AWS was chosen as the preferred cloud platform to integrate these tools and services. This is mainly due two reasons - First, the company already has a strategic partnership with AWS. Almost all their public cloud infrastructure is in AWS. Therefore it is easier to build the application on their existing AWS infrastructure and integrate it seamlessly into their environment. Second, my personal expertise is also in AWS in the domain of cloud computing. AWS is secure, stable, reliable and highly configurable to suit personal requirements which means both the application architecture and AWS services can be configured in a way that would best suit the working of the application.

These 3 approaches form the basis of the master thesis. These contributions are further subdivided into two categories: Theoretical and Empirical and are explained in the following section.

## Contributions

In this section, the different types of contributions involved in writing the master thesis are discussed along with the explanation of how these contributions helped create the skeleton for the slackbot application.

To briefly explain, there were 2 contributions that were detrimental for the thesis. They were:

* Theoretical Contributions
* Empirical Contributions

### Theoretical contributions

Theoretical contributions come from the theoretical analysis which was performed in *Section 1.5.1 - Theoretical Analysis*. The results of this study will be explained in this section.

#### Business Process Automation (BPA)

The first topic in theoretical analysis was *Business Process Automation (BPA)*. This study was performed to learn how BPA can be achieved in small organisations as the goal of the thesis is also to automate business processes 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 [34].

BPA provides certain benefits such as higher productivity, improved efficiency, less human error, allowing employees to focus on important things, reducing operating costs, etc to companies. Even though BPA provided these valuable benefits, it was important to consider the pitfalls that come along with automation in business. *Gartner Inc*, which is a highly reputed tech research and consulting firm listed the below two pitfalls as the top 2 automation mistakes to avoid in their article [5]. Therefore, with the help of theoretical contributions derived from the theoretical analysis, I was able to mitigate these 2 common pitfalls, as mentioned in Gartner article, that businesses usually face while adopting automation:

1. Falling in love with a single technology such as Robotic Process Automation (RPA) which is widely known for automation.
2. Believing that no code applications or external softwares can solve business problems that are specific to the company.

The goal was to avoid these pitfalls right from the beginning in order to build meaningful technologies that actually provided value to business. This was the reason why the plan for slackbot was to build a customised automation application that solves the company specific problem rather than a generic application. Other considerations from the theoretical analyses that became part of theoretical contribution includes:

###### 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, it was evident that by properly implementing automation to their existing process, which was not broken, can only make the company and employees more efficient.

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

The tools that I had selected for building slackbot include Slack, AWS, and Google Dialogflow ES. Slack and AWS, as mentioned in sections 1.4.1 and 1.4.3, were used by the company due to their strategic partnership and licensing. Google Dialogflow ES was the best available option for the company’s use case as discussed in section 1.4.2.

#### Microservices Architecture

After learning in-depth about BPA to implement the best practices in business process automation, the next step was to decide the architecture of my application. For the slackbot application, MSA was chosen as the final architecture due to the reasons which were already mentioned in *section 1.4.5*.

### Empirical Contributions

As theoretical contributions came from theoretical analysis, empirical contributions come from the practical work which was done before writing the thesis. This encompasses the two remaining concepts of the thesis approach: *Section 1.5.2 Natural Language Processing* and *Section 1.5.3 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, build applications and deploy it in the production environment where actual users can interact with the application. Before building the slackbot, I had developed a similar application as a Proof-of-Concept (PoC) for the OEV *Online Dienste GmbH* as part of a module in my study program. The same set of platforms used for slackbot was also used for building the PoC - Slack, AWS and Google Dialogflow ES. The PoC application provided the first empirical contribution with a solid proof that if implemented correctly the slackbot application will work in the production environment. The second empirical contribution which formed the basis for the thesis is in the domain of IT security. Both these empirical contributions are discussed in the following subsections.

#### Proof-of-Concept

The functionality of the PoC application was to initiate slack workflows through a chatbot installed in the company’s slack channel. The PoC application became part of the empirical contribution where I got hands-on experience in chatOps. ChatOps was a theoretical methodology that I had knowledge about through research papers and documentations but never had the opportunity to test its validity. This was until the PoC was built and chatOps was applied in production. It proved in practice how chatOps contributes to improving company processes. The PoC improved certain processes inside the company on a significant level which led to the definitive proof that “automation through chatOps helps solve the company’s disconnected documents problem and removes roadblocks in team collaboration” which was hypothesised in the Introduction section of this document. ChatOps also includes the topic of workflow automation which was another methodology tested in the PoC application.

#### Workflow Automation

Workflow refers to the series of activities needed to complete a task [35]. Workflow automation refers to implementing software that can complete tasks which are managed manually. Automation software company ‘*Zapier*’ surveyed and found that 94% of the small and medium-sized business workers said they perform repetitive, time-consuming tasks [8]. 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 crucial requirement for both the PoC application and the slackbot from the company’s perspective. In order to successfully implement workflow automation in the PoC application, the below 5 steps were followed which I learnt from the background research I had performed for building the PoC application [36]:

1. **Identify the process** - Taking a note of all the processes that needs 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 were successful in building the PoC on which automated certain workflows for the company. Therefore these proven 5 steps will also be followed to build the slackbot which also performs workflow automation in the slack application. The in-depth details of the PoC application, its requirements, slack workflows, etc are explained in chapter two of this document.

### IT Security

Second empirical contribution for my thesis is in the domain of IT security. One of the important concepts that was not too much focused while building the PoC was IT security. There was no implicit security system built to protect the PoC or the data which was shared across different platforms. The PoC was sustained only by the built-in security features of AWS services like AWS lambda and API Gateway, Google DialogFlow and Slack. Since it was only a PoC, it did not have any major impact due to lack of security systems but this is not the case with the slackbotapplication as it is a complete, scalable application that real users will interact with. Therefore, For the master thesis, there will be more focus on 3 areas of IT security - infrastructure, data and network as it is important to protect the application data and user data from intentional and accidental breach. As part of security measures, keeping the application as compact as possible within easily migratable cloud areas is the goal. This makes it easier to design a security system that covers all platforms used by the slackbot application.

Theoretical contributions may have provided the knowledge and information necessary to design the application but empirical contributions in the form of PoC gave both the practical experience and a definitive proof that by combining the tools and methodologies that was studied in theory, a complete microservice-based, secure, scalable, reliable application can be built and successfully deployed in production environment that will most importantly work in real time. Both theoretical and empirical contributions were important for the master thesis.

# 

**CHAPTER**

TWO

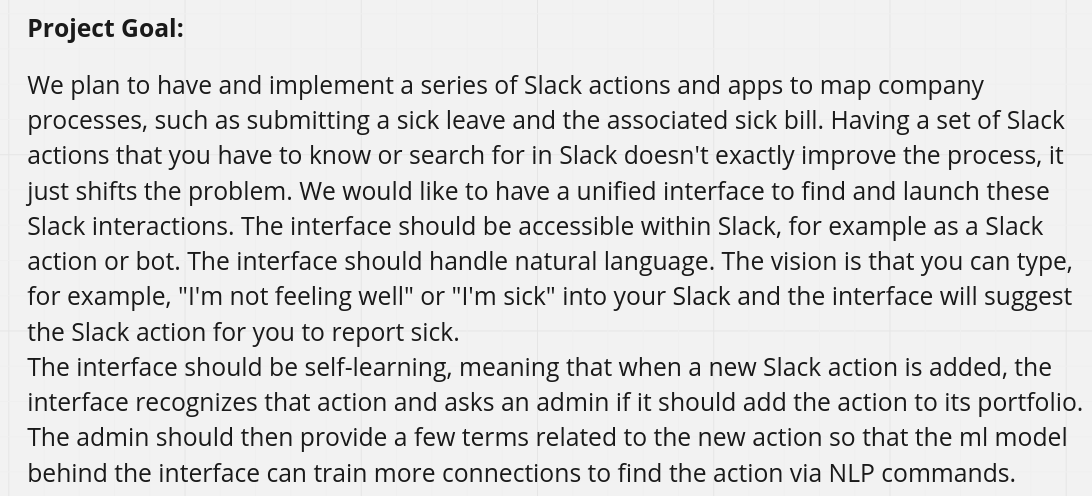
STATUS QUO

As discussed in *Section 1.6.2 Empirical Contributions*, the slackbotis not a completely new application but is rather an improvisation on an existing PoC. In this chapter, the existing system which formed the foundation for the slackbot will be discussed. The chapter will begin with an overview of the existing system, followed by the requirements document provided by the company for the PoC. In the final section of chapter two, I will discuss the existing processes and tools used in the company for maintaining their knowledge bas

## Existing System

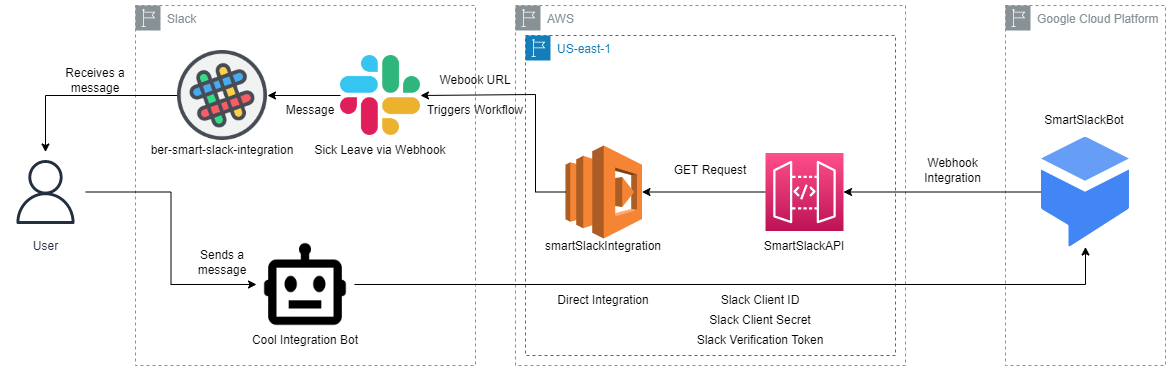
The PoC version of the slackbot had only 3 environments as part of its architecture - Slack, AWS and Google Dialogflow. The slack application was used as the frontend (or) User Interface (UI) for the employees of the company to interact with the application. The backend was created using AWS and Google Dialogflow. Dialogflow ES agent was used for the NLU part similar to the slackbotapplication. 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 was also built using MSA. All the services were loosely coupled, independently deployed and were scalable whenever necessary. The PoC was deployed to the production environment using the process of ***Infrastructure-as-Code*** (IaC). As the whole infrastructure is on AWS, I used ***AWS Cloud Development Kit*** (CDK) to deploy the infrastructure of PoC 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 brief overview of the software stack of the PoC application.

## Requirement Document for PoC

Below (*Figure 1*) is the project goal that was defined by the company as requirements for developing the PoC. 

As you can see (in the image above), the project requirement was 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 in slack. 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 a problem statement, the PoC application was developed. The architecture and functionality of the application is explained in the next section.

## PoC Application

The below architecture (*Figure 2*) shows how the PoC was built using slack app, Dialogflow ES and just 2 AWS services - Lambda and API Gateway. There are only 3 environments - Slack, AWS and GCP. These 3 environments integrate together to form the PoC application. 







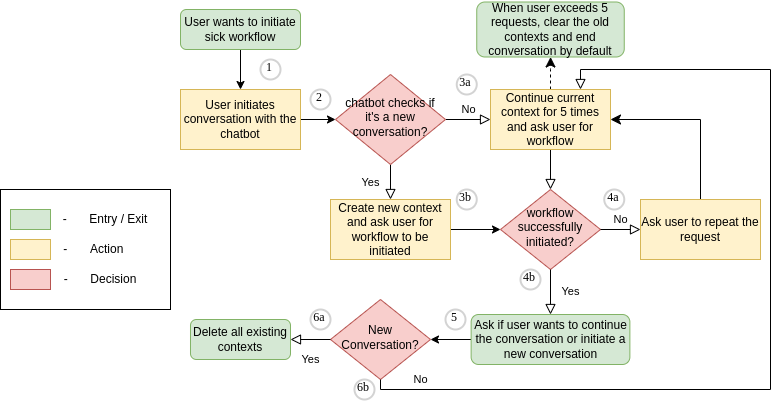
Its functionality can be summarised in 5 simple steps. These steps are also marked in the architecture diagram above from 1-5 in chronological order:

1. When the user wants to initiate a slack workflow (For example a sick workflow), the user initiates 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 agent by sharing the slack app’s *client ID*, *client secret*, and *verification token*. These 3 tokens are unique to each slack app and are generated when a new slack app is created in the slack app homepage [35]. These 3 tokens provide the basic security to the custom slack app. When users send a message to the slack app, the message is sent directly to the Dialogflow ES agent. The ES agent understands that a user is initiating a conversation based on its NLU capabilities and responds to the message.
3. The response from the Dialogflow is sent directly to the AWS through a webhook integration. 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 lambda service. The API Gateway service acts as the webhook link between Dialogflow and AWS. The API Gateway, in this case, is a REST API that is allowed to perform 2 methods - *GET* and *POST*. The API Gateway’s REST API is connected to the Dialogflow agent so that the chatbot’s response is sent directly from the GCP environment to AWS. The ES agent sends the user message with other conversational parameters 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. AWS lambda function is used as the compute service that has the code to decide whether or not to initiate the slack workflow that the user had requested. The lambda function is written in Python 3.8.
4. Once the lambda function receives the JSON payload, the code written inside the lambda function decides to initiate a workflow or not based on the value of a json parameter - ‘*workflow*’. This ‘*workflow*’ parameter is sent from the ES agent to lambda through API gateway. There are 3 workflows that I had created in Dialogflow, each workflow for one use case - Sick, Vacation and Invoice. These 3 use cases were provided as the project requirement from the company. The lambda function checks if the ‘*workflow*’ parameter coming from Dialogflow has any of the 3 above mentioned values. Based on the value of ‘*workflow*’, the specific workflow is initiated. If the ‘*workflow*’ parameter value does not match any of the 3 values then the response is sent directly to the user without initiating any slack workflow. The user receives a response based on Dialogflow agent’s advanced NLU capabilities. The lambda service is only used to trigger the slack workflows by invoking the slack workflow’s webhook URLs. Slack workflows have their own webhook URLs which when invoked initiates the workflow in the slack channel. Lambda sends all the responses received from the Dialogflow 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 directly on the slack channel. If for some reason the workflow is not initiated then the user also gets the response sent by Dialogflow 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 existing system that was created before starting the master thesis.

## Conversational User Flows in Existing System

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



The flowchart is created to display the end-to-end interaction in the PoC. The flowchart is also colour coded for easy comprehension of the dialogue structure between the user and the highly advanced NLP based ES agent. There are 3 major components in the flowchart above:

1. The green component denotes the entry or exit points where the user initiates or ends the conversation with the chatbot.
2. Yellow rectangular components are the actions taken by the Dialogflow agent based on user input or existing context of the conversation.
3. Pink diamond components denote the decision taken by the ES agent based on the user actions.

The flow of conversation in the flowchart is also numbered from 1 - 6 for the readers to understand the dialogue flow. Flow points 3, 4, and 6 are split into (a) and (b) because these flow decisions are based on the binary choice of ‘*yes*’ or ‘*no*’ made by the chatbot based on the user inputs or on the existing context of the conversation. When a conversation ends, all the contexts of the conversation are deleted and a new context is created for each new conversation.

## Why Slackbot

The reason for the PoC being just the first step and not a full application is because it is only semi-automated. Even though the workflow initialisation process in the backend is automated. It is only after the user initiates the conversation with the chatbot. Therefore it is not as effective as full automation. Users have to manually go to the slack app and initiate a conversation to perform the slack workflow initiation. This was a concern for both parties involved - the company and myself which led to the motivation of developing a fully automated version of the PoC 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.

## Existing Processes and Tools

In this section, the existing processes and tools used in the company for documenting business processes, team collaboration, and maintaining knowledge base about projects, teams, organisational procedures and other essential information about the company are discussed.

### 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 *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 *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 *slackbot* will be automating the below 5 use cases as part of the 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. Some employees don’t know such a workflow even exists for applying sick leaves. Some experienced employees might have also forgotten how they initiated the sick workflow as they 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 the employees to search and find where and how to initiate the krankmeldung workflow even if they are aware of this 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 slackbotis trying to solve. Slackbot, when activated, provides the user with 2 buttons - one for redirecting the user to the confluence page 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 in the company 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 slackbotfor this use case is similar to applying sick leaves. 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 slackbotfor this use case will be to provide 2 buttons similar to the previous 2 use 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 slackbotfor this use case will be to redirect the users from the general slack channel where everyday conversations happen to the specific slack channel created for the purpose of sharing new project 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 previous 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 need, but the same problem of finding the confluence page among multiple confluence pages exists. At the same time searching for the project details is a time consuming and tedious task even inside the project details confluence page. The role of *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 *slackbot* will fetch the details of this project number from the confluence page and display it directly to the user. The user does not have to go to the confluence page and search for the project details.

Chapter one and two have covered all the non technical aspects of the master thesis. Chapter three, four and five will explain the technical aspects of the slackbotapplication including the requirement analysis, step-by-step plan for building the application, architecture and individual components used for building the application.

# 

**CHAPTER**

THREE

DEVELOPMENT PLAN

An overview of the *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 *slackbot* application:

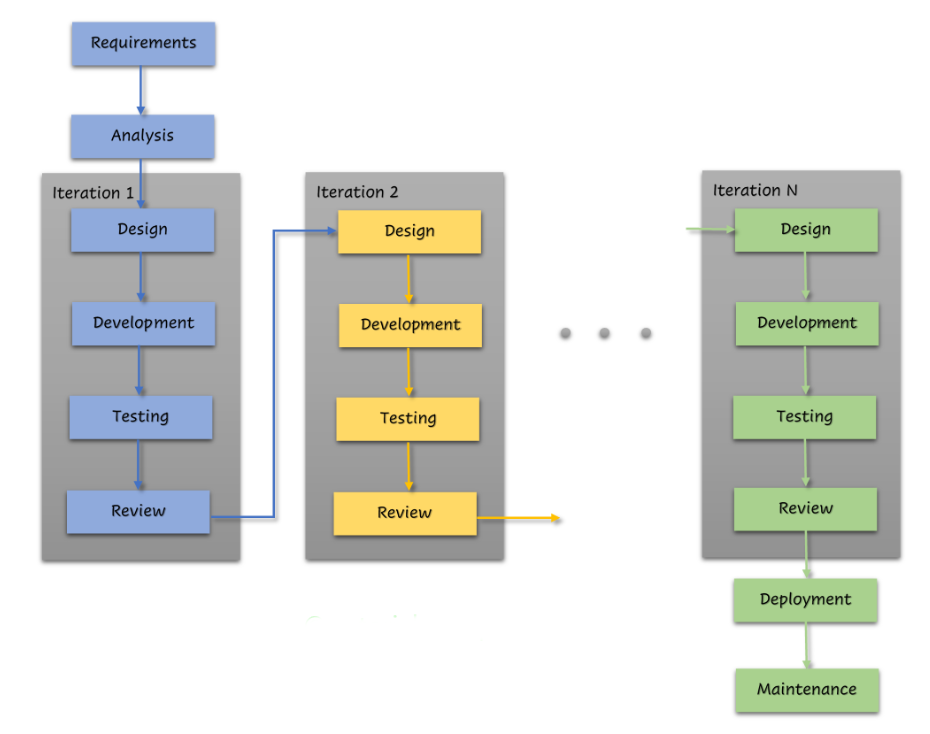
* Platforms used - Slack, AWS and Google Dialogflow ES
* Architecture - Microservices Architecture (MSA)
* Methodology - ChatOps, SDLC
* Tech Stack - Python, Typescript, HTML, Javascript, CSS

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 *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 *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 *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 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 *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.



If the above image is not visible or unclear, it can also be found in the github repository link:<https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/iterative-sdlc.png>

## 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 is subdivided into two steps: Requirements Gathering and Requirements Analysis.

#### **Requirements Gathering**

Requirements Gathering is usually done in collaboration with the stakeholders in order to collect, record and categorise the requirements. This process consists of 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 *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 *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 *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 *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.

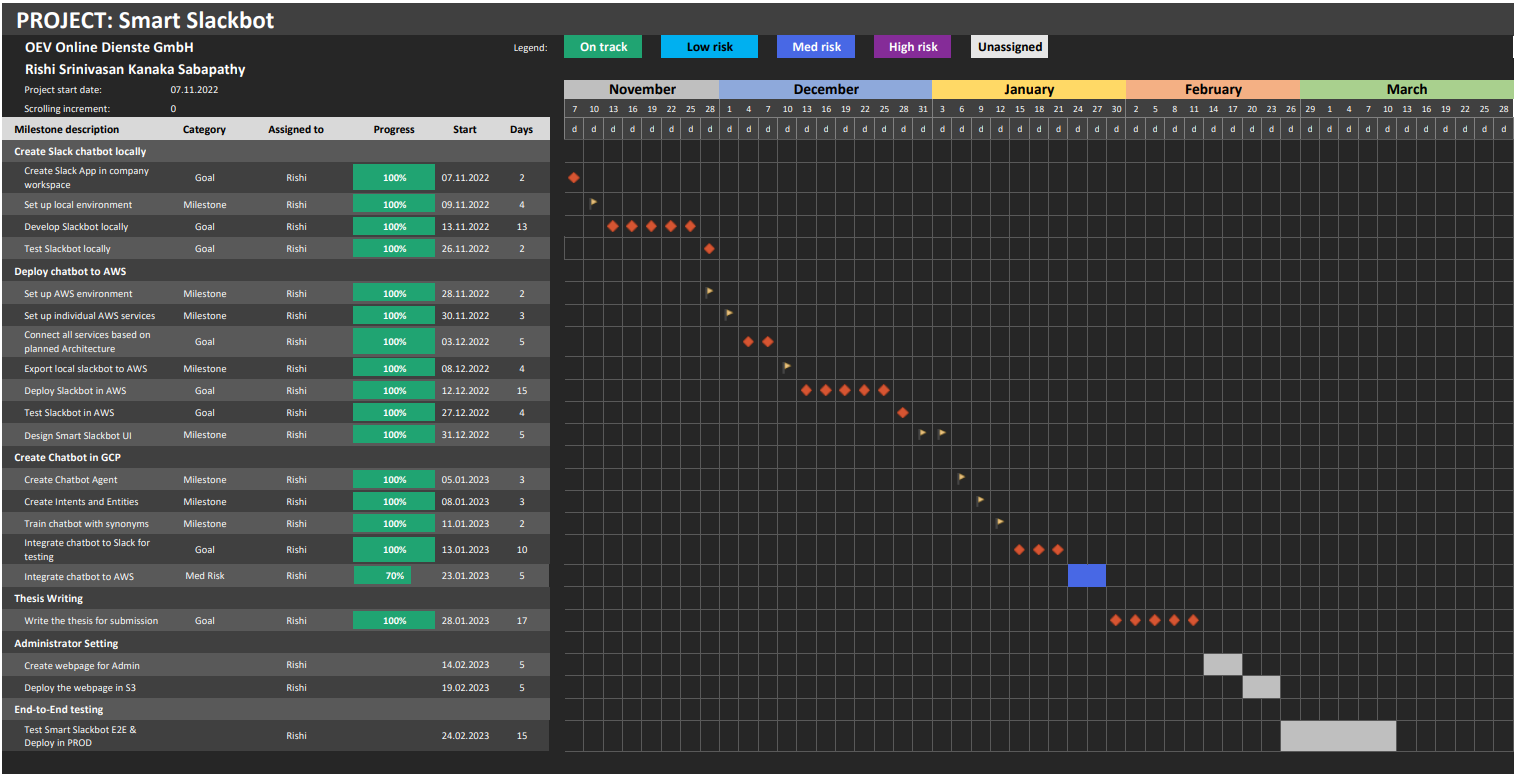
On the other hand, system requirements are technical information that are more clearly and rigorously written. These requirements are written within the development team to get a clear overview of the technical specifications of the software to be developed. User requirements are collected in a document 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 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 used in projects to track the progress of all tasks in the project. Below (img no.) is the Gantt chart I created to track all the tasks for developing *slackbot*.



###### 

###### 

###### 

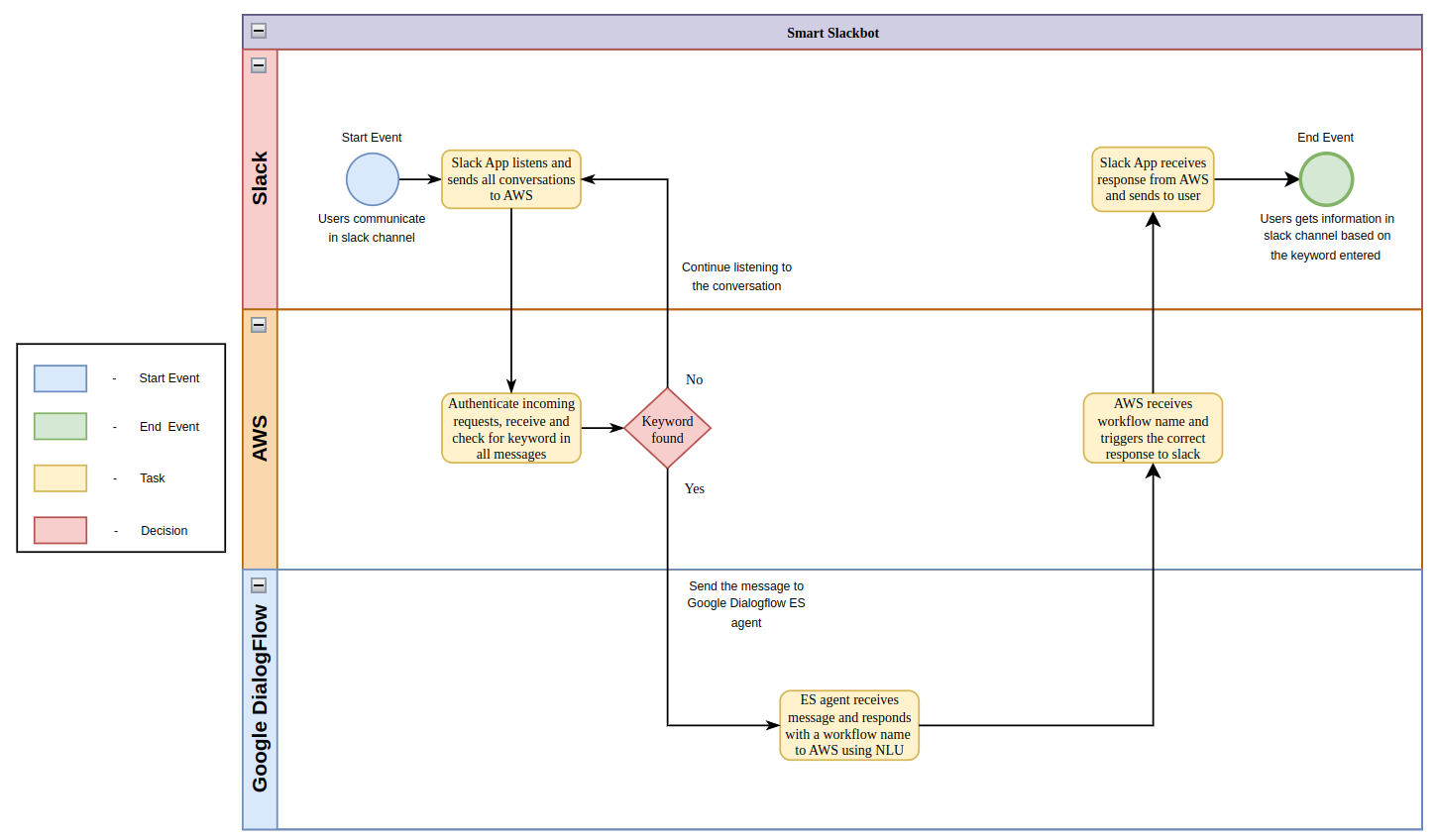
If the above image is not visible or unclear, it can also be found in the github repository link:<https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/gantt-chart.png>

On the left of the above Gantt chart, you can find information such as: Milestone description which explains the task briefly, Category of the task whether it is a goal or milestone or risk, Team member to whom the task is assigned to, status or progress of the task followed by start date and days required to finish the task. Goals and Milestones are very similar and are often used interchangeably in project management. Goals are the results that a team or a person achieves within the specified timeline and milestones are the important measurable checkpoints reached in order to achieve the goals. On the right side of the above chart, you can find the visual representation of the tasks’ scheduled timelines along with the current status - completed or at risk. The red diamonds are for goals and yellow flags are for the milestones. They signify that the task is completed on the date specified at the top along with the month. Blue and purple bars show that a task is at risk and it is not completed on time. Various shades of blue differentiate between low and medium risks. Purple is for high risk tasks. Currently there is one medium risk task that needs to be completed. Grey bars signify that the task is unassigned and is planned for the future.

###### Step 5: Sign off

Once a final decision is made on the requirements, ensure that you get a confirmation from the key stakeholders regarding the accepted requirements. This is done to ensure that there are no changes or uncontrolled growth in the scope of the project.

#### **Requirements Analysis**

Requirements Analysis, on the other hand, is for the development team to analyse the collected requirements. This provides a clarification on what is expected from the stakeholders and how to build a system that helps meet those requirements. There are many requirements analysis techniques that can be used to understand the requirements. For *slackbot*, I used the Business Process Model and Notation (BPMN) technique for requirements analysis. BPMN is a flow chart method that models the steps of a planned business process from end to end. It visually depicts a detailed sequence of business activities and information flows needed to complete a process. Below is the BPMN flowchart I created for modelling the processes of *slackbot*:

In the above flowchart, you can understand that there are 3 environments: Slack, AWS and Google Dialogflow, and the business process for *slackbot* transcends between these 3 environments. The start event and end event defines the endpoints of the process. The business process begins at ‘Start Event’ and ends at ‘End Event’. The flow of information is unidirectional when the ‘*Keyword found*’ decision is ‘*Yes*’ and the flow of information becomes a loop when the ‘*Keyword found*’ decision is ‘*No*’.

After gathering and analysing the requirements necessary for building the application, we have to design the conversational user flows in Google Dialogflow. This is important because the conversational user flows define the way in which the users interact with the Dialogflow ES agent and the response users receive from the Dialogflow ES agent is also largely influenced by the conversational user flows. I explain how the user flows are designed in Dialogflow for the slackbot application in the next section.

## Conversational Design

Conversations are an interactive way of communication or social interaction between two or more human beings. Conversations, for centuries, have been structural in building relationships and exchanging ideas. Since the evolution of Artificial Intelligence (A.I), people have questioned the machine’s ability to have human-like conversations. This is one of the main themes in Alan Turing’s seminal paper, in 1950, “*Computing Machinery and Intelligence*” [18], in which he proposed an experiment to evaluate the intelligence of A.I systems based on their ability to generate conversations that cannot be distinguished from humans. This is called the Turing Test and it was one of the most significant tests used in the past to evaluate an A.I system’s ability to think. Alan Turing proposed that if a machine can have conversation like humans, then they can think like humans but A.I systems of today are not the same as they were 70 years ago hence many A.I systems have managed to pass the Turing test without actual human-like intelligence [19]. Since the development of Large Language Models (LLMs), the conversational A.I has transformed from rudimentary chatbots that gave pre-fed answers with no intelligence in Alan Turing time to transformer-based deep learning models with some intelligence that can recognize, summarise, translate, predict and generate texts based on the knowledge gained from massive datasets.

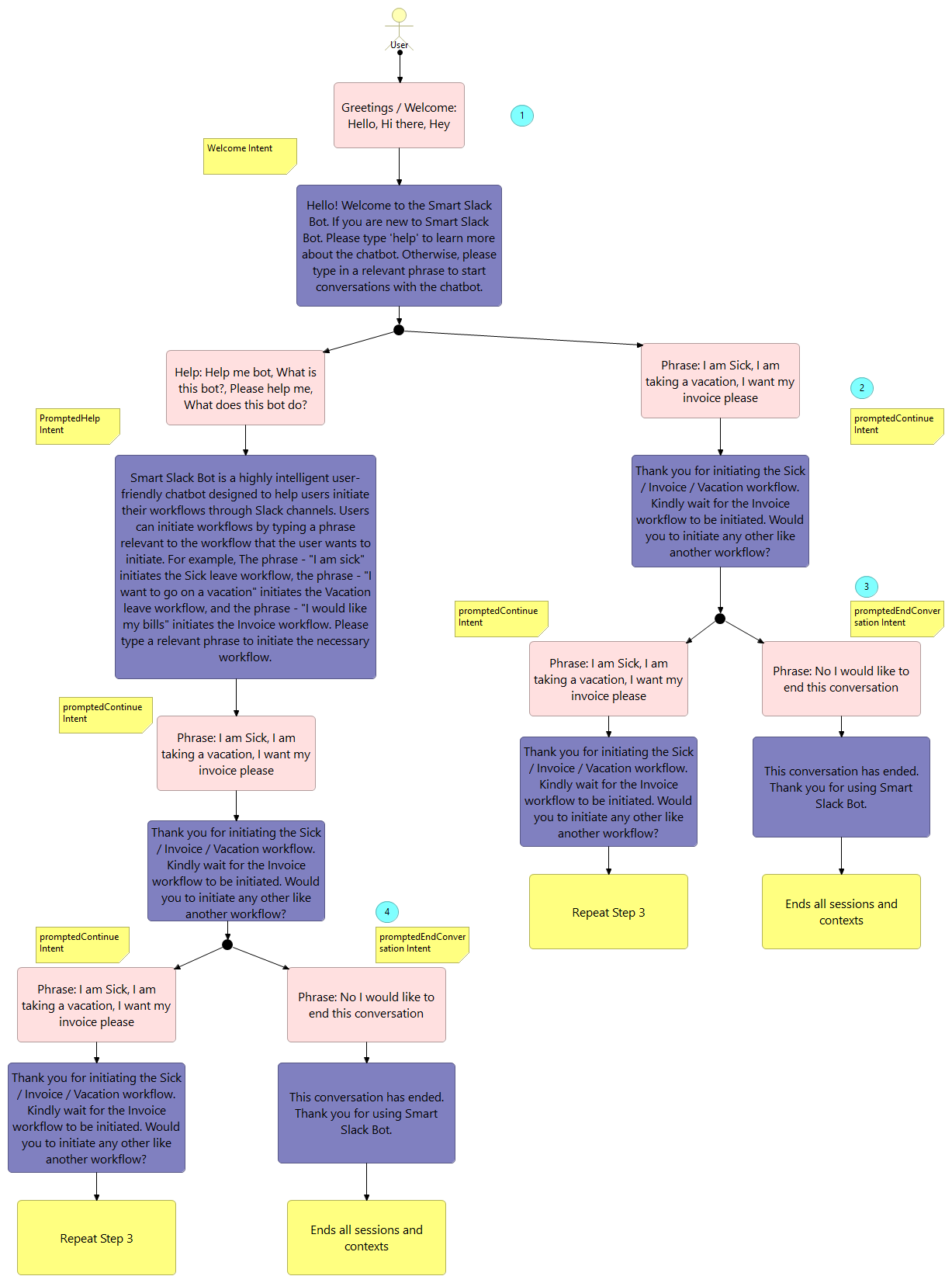
Conversational A.I is the study of techniques for software agents that can engage in natural conversational interactions with humans. Conversational A.I is the brain that powers virtual agents or chatbots. Since the purpose of chatbots is to solve human problems or help humans solve their own problems we can design the conversations on how humans and chatbots can interact with each other. This is called conversational storylines [21]. Designing conversational storylines is the first step to building a chatbot as this determines how the chatbot will respond to each interaction with a human. Since the chatbot or Dialogflow ES agent is a key player in *slackbot*, I will be explaining in this section how I designed the conversational storyline for our chatbot.

In making a conversational storyline, the first step is determining the overall theme of the conversation. The theme for our conversation is to determine the type of workflow that users want to initiate in the company. So the goal of the chatbot is to have a conversation with the users and predict what workflow the users want to initiate. Now that we have determined the overall theme of the conversation. Next step is to determine each scene and the characters to illustrate each scene. This helps clearly define the roles of chatbot and the users in each scenario where the chatbot is used by the users. Below is how I set them both for our use case:

| Scene Number | Scene | Character |
| --- | --- | --- |
| 1 | Sick Leave | Sick Employee |
| 2 | Vacation Leave | Employee going on vacation |
| 3 | Submit Bills / Invoices | Employee requesting information |

*Table 1: Chatbot Storylines*

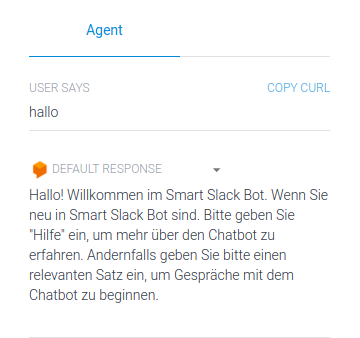
There are 3 scenarios where the Dialogflow ES agent is used in *slackbot* currently. These 3 scenarios form the scenes and the character is the role which the users play in each of these scenes for the chatbot to understand what the user wants in each scene. After creating a conversational storyline, the next step is to sketch the storyline branching route. The goal of branching storylines is to anticipate what the user wants and navigate the user through the available options present in the chatbot which are closest to what the user wants. This is done through giving ‘prompts’ to the users about the available options so that the users can choose the closest option that they need. Below is the branching storyline diagram that I created for *slackbot*. The diagram illustrates how the conversation user flows happen between the Dialogflow ES agent and the user.

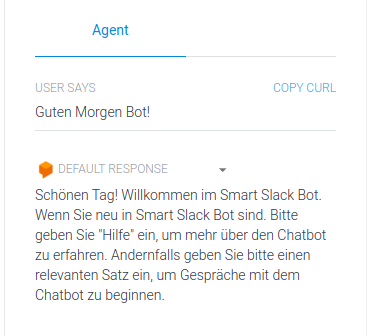




In the above image (img no), the conversations are colour coded for better understanding. The user entries are denoted in pink while ES agent’s responses are denoted in violet. The yellow boxes represent intents, contexts and processes. I will explain these terminologies in detail separately when I talk about Google Dialogflow in chapter 4. In this section, I only discuss how the conversation is structured between the users and the agent. As you can see in the above branching diagram, there are 4 steps involved in the conversation. These steps are highlighted in Teal blue.

###### Step 1:

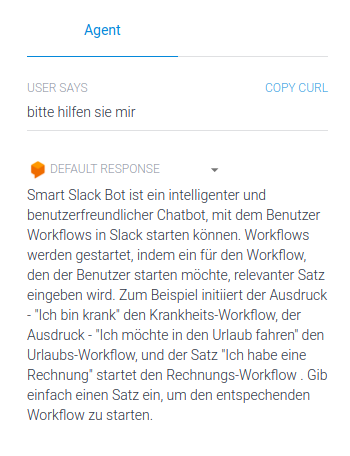
All conversations with the agent begin with a greeting or welcome from the user. The Dialogflow agent is dormant unless the user initiates a conversation with a greeting or a welcome message. The greeting can be anything - Hi, hello, hey, Good Morning, etc. Since the Dialogflow agent uses NLU in the background to understand conversations, they will be able to recognise any welcome message from the user by using the “Welcome Intent” set in Dialogflow ES agent. The Welcome Intent determines what types of welcome messages the agent will be able to comprehend and what are the types of responses that the agent will send to the user. After receiving the welcome message from the user, the agent responds with an introductory message. This introductory message is also set in the “Welcome Intent” of the Dialogflow agent. 

Since the agent is developed both in English and in German, I will show the conversational user flows in both languages. The english version is available in the branching diagram (image no.) and the german version is a screen capture directly from the Dialogflow console shown above (image no) and below (image no)

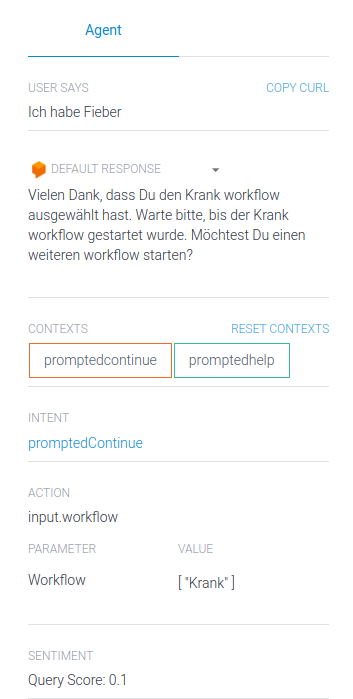
As seen in the images (1) and (2), the chatbot responds differently to different welcome messages from the users. This is because both these responses are stored in the welcome intent and the chatbot uses NLU to decide the most appropriate response to the user’s input.

###### Step 2:

The second step in the conversational user flow comes with 2 options. First option is when the user is new to the chatbot and has no idea what the chatbot does. In this case, the user can ask the chatbot about itself before asking it to perform an action. As you can see in the branching diagram, the chatbot explains to the user what it is created for, what it can do and how users can interact with it. This is part of the “promptedHelp Intent”. The responses are registered in the promptedHelp intent and similar to the welcome intent the chatbot chooses the most appropriate response and sends it to the user based on the user input. Below is an example of the same promptedHelp intent response in German.



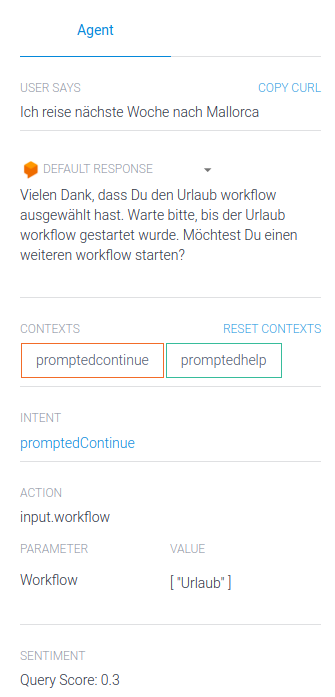
Second option for the users to choose from is directly asking the chatbot to perform an action. This is the “promptedContinue Intent”. If the user is already familiar with the chatbot, the user does not necessarily have to go through the promptedHelp Intent as the user already knows how the chatbot works. Therefore the user can directly converse with the chatbot. Based on the scenes that we had created for the agent earlier (*Table 1*), the user can ask the chatbot to perform 3 different actions: initiate sick workflow or vacation workflow or invoice workflow. These 3 workflows are the scenes created for the chatbot in its current version. Once the user asks the chatbot to perform an action, the promptedContinue intent sends a response to the user saying the workflow has been successfully initiated and asks if the user wants to continue the conversation or not. Example of this intent is shown in the branching diagram in English and below (image no) in German



As you can see in the above screen capture, the chatbot is smart enough to understand different ways in which the user can express that they want to initiate a sick workflow. In the above example the user does not mention the word “Krank” but the chatbot is smart enough to understand that “Fieber” refers to the Krank workflow and thus initialises the krank workflow. In the above image you can also see the other information captured by the chatbot such as: Contexts, Intent, Parameter and Sentiment. In contexts, the chatbot shows what other contexts are possible for the user next. The user can continue the conversation here which means the next context is promptedContinue or the user can ask for help with the promptedHelp context. In intent, the chatbot has recognised the user input as promptedContinue intent. If the user had asked for help the chatbot would recognise it as promptedHelp intent. Parameter is the entity set by me to recognise the action which the chatbot should perform. The entities for *slackbot* are Workflow and endConversation. These entities captured from the user input define the actions that the chatbot will perform. In the above example, the entity is correctly identified as Krank and hence Krank workflow will be initiated. The last detail is the Sentiment which is the confidence score of the NLP engine. In this conversation, the chatbot is 100% confident that the user is requesting for the sick workflow to be initiated. This information is captured for every input entered by the user. The chatbot will only confirm that the workflow has been successfully initiated when it recognises what the user wants. If it does not recognise the user input it will ask the user to repeat again. I will explain about contexts, intents, entities in more detail in chapter four. With this, step 2 is completed.

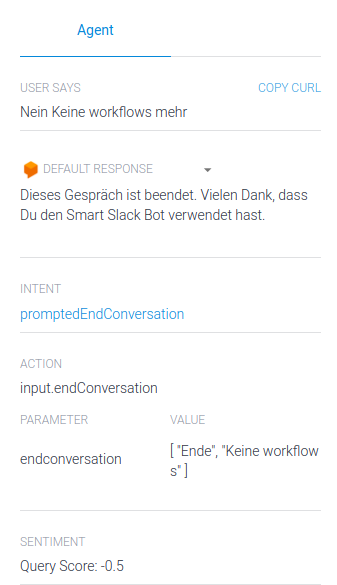
###### Step 3:

Step 2 ends with the chatbot asking the user for a response where the user can either continue the conversation by asking the chatbot to initiate another workflow or end the conversation if the needs of the user are satisfied. Step 3 covers this. If the user wants to continue and initiate another workflow then the user can proceed by directly asking the chatbot to perform an action. This is similar to the second option of step 2 in the branching diagram. An example of this is shown in the branching diagram in English with the intent detected as “promptedContinue intent”. Below the same scenario is shown in German.



In the above example, you can see that I have not used the word “urlaub” but the chatbot is able to identify the correct workflow that I want to initiate using the verb “Reisen” and the entity is also detected as [“Urlaub”] in parameter value. The user can go ahead and continue the conversation for a maximum of 5 times after which all the contexts are deleted meaning the chatbot forgets everything and the user needs to start fresh. This is an example of continuing the conversation.

Second option in Step 3 is if the user wants to end the conversation then the user can ask the chatbot to end this conversation. An English example is shown in the branching diagram as promptedEndConversation Intent and the German example is shown below.



For ending the conversation, the chatbot uses the second entity ‘endconversation’ whereas the previous examples had the ‘workflow’. This is to differentiate between users asking to initiate a workflow and users choosing to end the conversation. As you can see in the image above, the user says “Nein Keine workflows mehr” and the chatbot understands that the user wants to end the conversation. In Parameter, you can see that the chatbot has recognised that the user wants to end conversation with the two words mentioned in the user input: “Ende” and “Keine workflows”. The intent used to recognise the ‘endconversation’ entity is the “promptedEndConversation Intent”. This is also captured in the intent variable in the above image. The chatbot then deletes all the contexts and ends the current session.

###### Step 4:

When the user is new to the chatbot as shown in step 2, and once the user learns about the chatbot the user can directly initiate an action from this point. This is denoted as step 4 in the branching diagram. Step 4 is basically step 3 attached to the end of the first option of step 2. The user can perform the same actions mentioned in step 3 in this step.

This is the conversational user flow created for *slackbot* using the Dialogflow ES agent in Google Cloud Platform. In the next section, I will talk about the step-by-step plan of building the *slackbot* application.

## Step-by-Step Plan

*slackbot* is a much bigger application than the PoC which I built as a prototype in the previous semester. Therefore, careful planning was involved in building the application so that the application is built on time and as per the stakeholder requirements. First step of the plan was to design the architecture of the application but to design the full architecture of the application, I needed to know the individual architectures of Slack, AWS and Google Dialogflow ES first. Therefore I studied the individual application, their architecture and services offered by the 3 applications. From the previous PoC I had some knowledge about how the slack, aws and Dialogflow can be connected together but this time the requirement and hence the way these 3 applications are integrated would also be different. This is because the basic requirement of how a conversation should happen was different between the PoC and *slackbot*.

In the architecture diagram of the PoC (Figure ), the slack app was directly connected to Dialogflow ES agent through a direct integration. This was possible because the user had to manually initiate a conversation and send a message to the slack app which then sent it to the ES agent. For *slackbot*, the slack app should always be active and listen to all conversations happening in a slack channel. Therefore a direct integration to Dialogflow agent would not work. I had to research and find a suitable way in which the slack app can listen to all conversations and become active upon listening to a particular keyword. I found a framework created by slack that allows the developers to create an app that can be activated based on a message sent by the users in a slack channel. The framework is called **Bolt** and it can be used to build custom slack apps [22]. My expertise is in python programming language therefore I used the Bolt for python sdk [23] for developing my slack app. Once the slack app was custom built it was able to listen to a particular conversation and respond accordingly. I tested the app locally and it was able to respond to only one particular keyword “Krank'' or “Urlaub”. Since I was following the Iterative SDLC model for building my app, I built iterative versions of a working application. There were a total of X iterations for *slackbot*.

###### Iteration-1

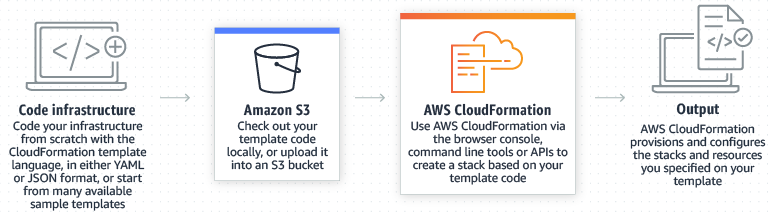
The first iteration was to design, build, test and review the local version of the application running on a local IDE in my machine. The first iteration was working as expected. Hence I decided to move the locally running app infrastructure to the cloud. In Iteration 1, the python script written in *smart\_slackbot.py* file was a basic code which initialised the slack app and listened to a particular keyword in the slack channel. Below is the first version of the code written for Iteration-1:

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **import** os  **from** slack\_bolt **import** App  **from** slack\_bolt.adapter.socket\_mode **import** SocketModeHandler  *# Initializes your app with your bot token and socket mode handler*  app **=** App(token**=**os.environ.get("SLACK\_BOT\_TOKEN"))  *# Listens to incoming messages that contain "hello"*  **@**app.message("krank")  **def** **message\_hello**(message, say):  *# say() sends a message to the channel where the event was triggered*  say(f"Hey there <@{message['user']}>!")  *# Start your app*  **if** \_\_name\_\_ **==** "\_\_main\_\_":  SocketModeHandler(app, os.environ["SLACK\_APP\_TOKEN"]).start() |
| --- | --- |

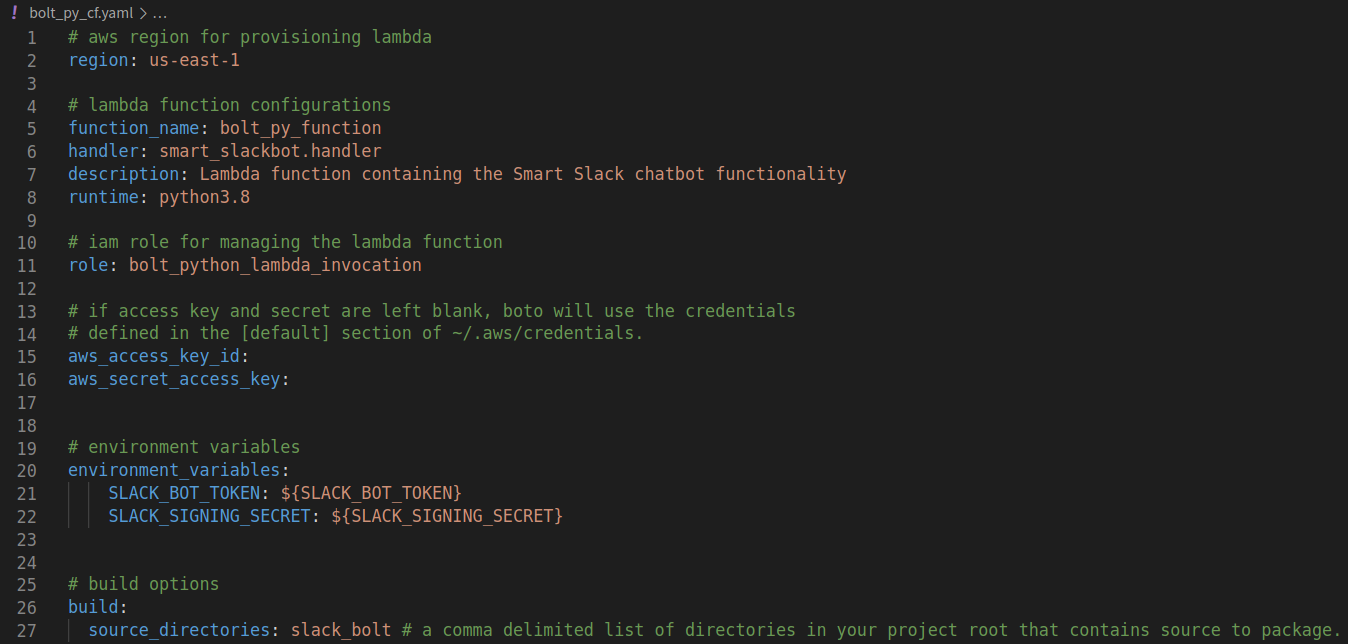
The above script just initialises the app (line 15 & 16) and listens for the word ‘krank’ (line 13) in the slack channel. When a user mentions the word ‘krank’, the chatbot responds in the same channel with the message (line 12) by tagging the user. To initialise/start the app, the code gets the ‘SLACK\_BOT\_TOKEN’, a unique token generated for each slack app in the Slack application, from the environment variables.

###### Iteration-2

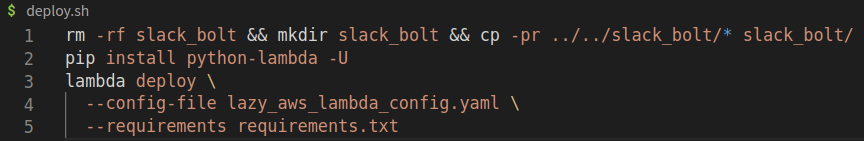
Next step of the plan was to find a way to integrate the slack app into AWS. The compute service of AWS - AWS Lambda is a powerful service that can manage the loads that the slack app receives from the users from the slack application. I know this from theory by studying the documents of AWS and also practically by implementing and testing the integration of AWS lambda along with the slack app for the PoC. Therefore I knew AWS lambda was the right choice to connect to the slack app for both reliability and scalability. For this, I had to include all the Bolt framework packages into AWS Lambda along with the python code. I used a Python-Lambda library to deploy my local code along with all Bolt framework packages to AWS lambda. The Python-Lambda library helps streamline the python - AWS lambda connections from your local machine to AWS. This package has a deploy command which will evaluate the virtual environment where the project is created and identify the required project dependencies. It will package these dependencies along with your lambda handler function to a zip file and upload it to AWS Lambda. I also wanted to write the Infrastructure-as-Code (IaC) for creating a lambda function locally and then deploying the service on AWS. For this, I created an AWS cloudformation template. AWS Cloudformation lets you model, provision, and manage AWS and third-party resources by treating infrastructure as code. Below is an illustration of how AWS Cloudformation works from creating the template in your local machine to getting deployed in AWS.



Code infrastructure is written in YAML or JSON format. I wrote it in YAML because it is easier for me to write in YAML than in JSON. When the YAML code is executed locally using python-lambda library, a cloudformation stack is created in AWS and the stack takes care of creating the services which we had written as code in our AWS cloudformation template YAML file. Below is the AWS cloudformation template in ***bolt\_py\_cf.yaml*** file that I had written to create my lambda function and deploy it in AWS:



There are some mandatory properties defined in the config file like region, function\_name, handler, description, role, and runtime. Whereas other properties like aws\_access\_key\_id, aws\_secret\_access\_key, SLACK\_BOT\_TOKEN and SLACK\_SIGNING\_SECRET are specific to our project and are not mandatory to writing a cloudformation template. Below is the deploy script that I had used to create my lambda function in the local environment and deploy all the code and its dependencies directly to AWS lambda.



This shell script creates a folder and copies all the dependencies created in the virtual environment to the folder, then it instals the earlier mentioned library ‘*python-lambda*’ and deploys the cloudformation template along with the requirements file to AWS. The cloudformation template file has all the necessary information for creating the lambda function in AWS.

Now that we have deployed the code to AWS, the third step was to connect the slack app with AWS to test if our application is working properly from the cloud. There were two options available to connect AWS lambda with slack app - First one was to connect the AWS lambda and slack app through a REST API created and provisioned through the Amazon API Gateway service. This is the traditional method and provides the loose coupling advantage as part of the microservice architecture. Second option was to directly connect AWS lambda to slack app using the AWS lambda function url. There are certain advantages of using function url over API gateway which I will discuss in section 4.4.

Once the slack app and AWS were connected using the function url, I had to test my application from the cloud. This was one of the challenging parts in the project because when I tested the code which was working in Iteration-1 (in local machine), it did not work in AWS despite being the same code. AWS lambda could not recognise the dependency packages of ***slack\_bolt*** uploaded along with the python script due to version mismatch. This code was working in the local environment in Iteration 1 but the same code did not work from AWS for Iteration-2. It took me a few days, as you can see from the Gantt chart(img no) which tracked my progress on a daily basis as part of the project management, to figure out the problem. I had to go through a lot of github issues and stackoverflow queries to understand what the problem was. I was finally able to triangulate the root cause of the problem. The code did not work in AWS because the python version that I had downloaded and was working in my local machine was Python 3.10 so all the dependencies created for the Bolt framework were for Python 3.10 but AWS lambda does not have the provision to use the latest python version. Only Python 3.8 is available (cloud formation img, line 8) in AWS lambda. Therefore when I pushed my code to AWS, both the python and bolt framework dependencies were running in Python 3.10 version. Therefore It did not work. I had to downgrade my local environment to Python 3.8, download all the python packages and bolt dependencies for Python 3.8 version and then deploy it in AWS lambda. This finally worked and I was able to listen to my slack app from AWS. This was iteration 2.

###### Iteration-3

Since the app was working from the cloud and was able to react to the exact keyword. The next step was to connect the Dialogflow ES agent to the slack app. Therefore, the app can react to not just the exact keywords but also to similar words used in our workflows - Krank, Urlaub and Rechnung. This step included getting Dialogflow ES agent involved with either the slack app directly or to AWS. I tried to build both versions’ architecture to see which suits better. Connecting Dialogflow to AWS offered more advantages because it allowed me to customise the app and it also made AWS lambda into a middleware between slack and Dialogflow.

There were two ways to connect AWS lambda and Dialogflow ES agent. Either through an API gateway as I did for the PoC (POC arch no.) but that was an additional AWS service to the existing architecture which impacted both the cost and the simplicity of the architecture. Therefore I chose to use the Google Dialogflow API which is used to build conversational interfaces. Dialogflow API. This API can be invoked from the lambda function when a user messages in the slack channel.

finally worked and I was able to listen to my slack app from AWS. This was iteration 2

|  |
| --- |

# 

**CHAPTER**

FOUR

ARCHITECTURE

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

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.

## 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 slackbot.

This is a significant task since the backend of *slackbot* is entirely dependent on the Dialogflow ES agent from Google. In the next section, I will give an overview of Google Dialogflow, its architecture, types of agents available and in deeper context how I built the agent for slackbot application.

This is why Dialogflow ES agent is smart and advanced than other chatbots as

After that it is made a sample conversation with the expected response taken based on the reference sources described earlier.

When the workflow is initiated successfully the chatbot is configured in such a way that it sends the payload to AWS Lambda along with the workflow name that we have asked the chatbot to trigger

## Slack Application

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

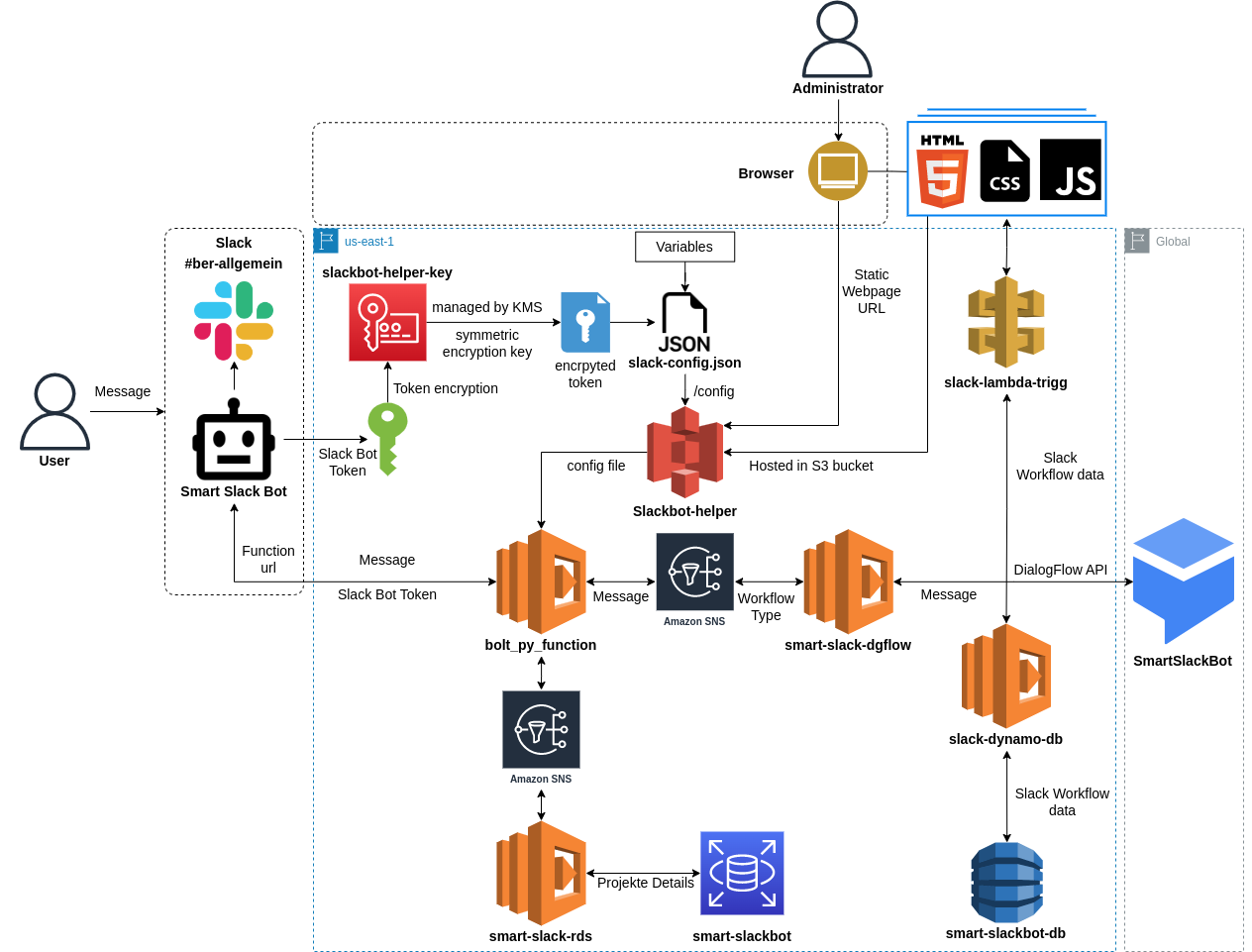
About Slack, Slack app, integration with AWS and Google Dialogflow

## Amazon Web Services

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

## slackbot Architecture

Architecture of entire application end-to-end

****

# 

1. **CHAPTER**
2. FIVE
3. 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

# 

**CHAPTER**

Six

THEORETICAL BACKGROUND

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

## Large Language Models

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

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

## Data Ethics and Data Privacy concerns

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

# 

**CHAPTER**

Seven

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

# 

**CHAPTER**

Eight

REFERENCES

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

1. Webster, M. (2015). Last Accessed: April 2015. Bridging the Document Disconnect in Sales, An IDC White Paper, Sponsored by Adobe. Adobe. [Link](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. [Link](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. [Link](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). [Link](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. [Link](https://www.gartner.com/en/articles/10-automation-mistakes-to-avoid)
6. Goel, A. (2021). Part 4 Including DevOps in microservices migration. Capgemini Engineering. [Link](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. [Link](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. [Link](https://zapier.com/blog/state-of-business-automation-2021/)
9. Slack Inc. Guide to apps in slack. [Link](https://slack.com/help/articles/360001537467-Guide-to-apps-in-Slack)
10. Slack Inc. Planning your slack app. [Link](https://api.slack.com/start/planning)
11. Slack Inc. Create more advanced workflows using webhooks. [Link](https://slack.com/help/articles/360041352714-Create-more-advanced-workflows-using-webhooks)
12. Slack Inc. Guide to workflow builder. [Link](https://slack.com/help/articles/360035692513-Guide-to-Workflow-Builder)
13. Confluence (software). Last Accessed: November 24, 2022. Wikipedia. [Link](https://en.wikipedia.org/wiki/Confluence_(software))
14. What is Jira used for?. Last Accessed: January 28, 2023. Atlassian Corporation. [Link](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. [Link](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. [Link](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. [Link](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)
18. Turing, A. M. (2009). Computing machinery and intelligence (pp. 23-65). Springer Netherlands. [Link](https://watermark.silverchair.com/lix-236-433.pdf?token=AQECAHi208BE49Ooan9kkhW_Ercy7Dm3ZL_9Cf3qfKAc485ysgAAAtEwggLNBgkqhkiG9w0BBwagggK-MIICugIBADCCArMGCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQMqw-FZrdacfQ9ew1CAgEQgIIChO9V7MJuPK47sZ-LLdBDt81VPRohUAmO0wW83XicDn10H-eYj-tSiBUPZsPzRyVYajG1oVSFLaxcj1RWUABD8OF6T0hP3O5SA1tBg-6Q_b9Xoq_QF4Dr3U_80IigOzhcvaSQ5AOHfPFvXI67-l_ZWqB8V0yAHIQeiKmSHo7-4DgxUdgSMOkvZRGo5E9_pCOIRElJFbkAkNYymmfuX9Wf0vXrzTyyIJx8OGXFYhin17nQecdi51yxUdIt0Ni-Cxf0DBwKiUEmenStL6-iWWf1NsnES-hOhk9rPP5sB40V6dwqEsCQC8Z8Wcd8Mm0XP0HO4cjQaWCxbjll5snwINconeFViXCgpFpJnZcy6D4NytI2_ivMx70V3CfQxgnSHxOHqgk1ryezTpxqGS9sfq6lKNgvXnYQY22410ce5Ta_ap-E_hi2UYXaPXw14rqDCQrYS3_xDvB6GNiXfl0o2vXSGkwy-O2J8P1jt2XbWS661uoZRg6jqzMF2AO52RwJABXsbTcvCgPmd4lmFt7B3DLTEpa5cGiVzImA-KeFi7JekZ4Nidmtk2DFjo314K8fGGwhJk2ViXrYsu1_rUJ3PB-gA55UtsiN85mZQ8mR1oNNLX6jPZgAJH4UA6WB9JwXrSQ2joSJp4AWgyIW2829fGp3A909dEvGy2ANjY27ztdXySbbGAjwbwQh3xanik961l1w-pYIhW5GUHhJfDNqq8xk2hgKS-fWjAHI0X_X6TdDfNFp33DKAOwLL7fxOIqZDFPj8n06Q4tywqYQpBGraQ1KquKkQps-rx3CQ80RMlBgCgCOOL94YP8yMCHna9bviQeyg57nP5z958DpUGoE1434neiGiGpf)
19. Oremus, W. (2022). Last Accessed: June 17, 2022. Google’s AI passed a famous test — and showed how the test is broken. Washington Post. [Link](https://www.washingtonpost.com/technology/2022/06/17/google-ai-lamda-turing-test/)
20. Hill, J., Ford, W. R., & Farreras, I. G. (2015). Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations. Computers in human behavior, 49, 245-250. [Link](https://d1wqtxts1xzle7.cloudfront.net/41445230/Hill__Ford____Farreras__2015-libre.pdf?1453499997=&response-content-disposition=inline%3B+filename%3DHill_Ford_and_Farreras_2015.pdf&Expires=1675537386&Signature=a2FDOhL0NsX9u0YdqWbA1Tr9J1QBGGNlsZIC7sadV6kii7ab9HRYJre5VJa1cCN-Fbvlxa~Xg34OqNJ7NZz~lTOOM0VziKGFrMTjkLwojXqVwAc2cBUK4yZ-A4l8skG~Dw3ziqB3J4c-5giRmfx~lQV5C~tqlUR5PZIniL8n7p~rjop2j2IiBJky2oYmTyBGtFPTsJjCFdeT0sxyNasIof5TvtSBMnjfjTlu-ca4VKw48DtMmS6xMs3D62DuV3IIhzz6Q8CLHj1QEUUiaLQ-zga6wY6KOUDlIBSqN2cU8aAYFhWCZQmPktxA4D5guAyrHjRleQbhR4ep8G~0fAHs-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)
21. Muhammad, A. F., Susanto, D., Alimudin, A., Adila, F., Assidiqi, M. H., & Nabhan, S. (2020, September). Developing English conversation chatbot using dialogflow. In 2020 International Electronics Symposium (IES) (pp. 468-475). IEEE.  
    <https://www.researchgate.net/profile/Aliv-Faizal-M/publication/346358785_Developing_English_Conversation_Chatbot_Using_Dialogflow/links/629f6122416ec50bdb13b3bc/Developing-English-Conversation-Chatbot-Using-Dialogflow.pdf>
22. Slack Inc. The Bolt family of SDKs. [Link](https://slack.dev/bolt-python/concepts)
23. Slack Inc. Getting started with Bolt for Python. [Link](https://slack.dev/bolt-python/tutorial/getting-started)
24. 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. [Link](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)
25. William, J. (2021). Last Accessed: February 10, 2021. How to bring a culture of automation in your organisation. Forbes Inc. [Link](https://www.forbes.com/sites/forbesbusinesscouncil/2021/02/10/how-to-bring-a-culture-of-automation-into-your-organization/?sh=17be76397c65)
26. Gaba, I. (2022). September 13, 2022. What is the Slack app and how can it improve productivity?. Simplilearn. [Link](https://www.simplilearn.com/tutorials/slack-tutorial/what-is-slack-app-uses-to-improve-productivity)
27. Linardatos, D. (2021). Last Accessed: February 14, 2021. TCO in private and public cloud. Cambrian Technologies. [Link](https://www.cambriantechnologies.com/blog/2021/02/14/comparing-tco-in-private-and-public-cloud/#:~:text=The%20public%20cloud%20is%20significantly,used%20components%20as%20managed%20services.)
28. Google Cloud. Dialogflow. [Link](https://cloud.google.com/dialogflow)
29. Prickett Morgan, T. (2022). Last Accessed: December 1, 2022. Counting the costs of training Large Language Models. The Next platform. [Link](https://www.nextplatform.com/2022/12/01/counting-the-cost-of-training-large-language-models/)
30. Google Cloud. Dialogflow Pricing. [Link](https://cloud.google.com/dialogflow/pricing#es-agent)
31. Slack Inc. About us. [Link](https://slack.com/about)
32. Slack Inc. Sending Messages through Incoming Webhooks. [Link](https://api.slack.com/messaging/webhooks)
33. Ponce, F., Márquez, G., & Astudillo, H. (2019, November). Migrating from monolithic architecture to microservices: A Rapid Review. In 2019 38th International Conference of the Chilean Computer Science Society (SCCC) (pp. 1-7). IEEE. [Link](https://www.researchgate.net/profile/Gaston-Marquez-2/publication/335716451_Migrating_from_monolithic_architecture_to_microservices_A_Rapid_Review/links/5d778d8292851cacdb2e2e23/Migrating-from-monolithic-architecture-to-microservices-A-Rapid-Review.pdf)
34. Lawton, G. (2022). Last Accessed: April 2022. Business Process Automation (BPA). TechTarget. [Link](https://www.techtarget.com/searchcio/definition/business-process-automation)
35. Pratt, M.K.(2022). Last Accessed: April 2022. Workflow. TechTarget. [Link](https://www.techtarget.com/searchcio/definition/workflow)
36. Pratt, M.K.(2022). Last Accessed: April 2022. Workflow Automation. [Link](https://www.techtarget.com/searchcontentmanagement/definition/workflow-automation)
37. Gaba, I. (2022). September 13, 2022. What is the Slack app and how can it improve productivity?. Simplilearn. [Link](https://www.simplilearn.com/tutorials/slack-tutorial/what-is-slack-app-uses-to-improve-productivity)
38. 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>

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>