



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

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 (Pro-ITD)

Faculty 4 / ZbwS

Presented by

Rishi Srinivasan Kanaka Sabapathy

Date:

Berlin, 14.02.2023

1st Evaluator: Dr. Helena Mihaljević

2nd Evaluator: Nico Schönnagel

Table of Contents

Abstract	5
List of Figures	6
List of Tables	8
List of Abbreviations	9
Chapters	
1 Introduction	10
1.1 Motivation	11
1.2 About Master Thesis.	13
1.3 Thesis Structure.	14
1.3.1 Research and Background.	14
1.3.2 Plan and Implementation	15
1.3.3 Ethics and Conclusion.	16
1.4 General Overview of Existing Tools and Processes	17
1.4.1 Slack	17
1.4.2 Dialogflow.	18
1.4.3 AWS.	18
1.4.4 ChatOps.	19
1.4.5 Confluence	20
1.4.6 Jira	20
1.4.7 Software Development Life Cycle	20
1.5 Thesis Approach	21
1.5.1 Theoretical Analysis	21
1.5.2 Natural Language Processing.	22
1.5.3 Cloud Computing.	23
1.6 Contributions	23

1.6.1 Theoretical Contributions	24
1.6.1.1 Business Process Automation	24
1.6.1.2 Microservices Architecture	25
1.6.2 Empirical Contributions.	26
1.6.2.1 Proof-of-Concept (PoC)	27
1.6.2.2 IT Security.. . . .	28
2 Status Quo	30
2.1 Implemented PoC	30
2.1.1 Requirements Document.	31
2.1.2 Architecture and Functionality	31
2.1.3 Conversational User Flows	34
2.1.4 Additional Requirements for Improvisation	35
2.2 Technical Overview of Existing Tools and Processes	36
2.2.1 Slack	36
2.2.2 Dialogflow	42
2.2.3 Processes	57
2.2.3.1 Applying Sick Leaves	58
2.2.3.2 Applying Vacation Leaves.	58
2.2.3.3 Submitting Invoices	59
2.2.3.4 Initiating Project Ideas.	59
2.2.3.5 Project Details	60
3 Development Plan	61
3.1 Tech Stack for Slackbot	61
3.2 Software Development Life Cycle (SDLC).	61
3.3 System Requirements Analysis.	63
3.3.1 Requirements Gathering	63

3.3.2 Requirements Analysis.	68
3.4 Conversational Design	69
3.4.1 History of Conversational AI	70
3.4.2 Designing Conversational AI	70
3.4.3 Conversational Storylines	71
3.4.4 Branching Storyline Route.	71
3.5 Step-by-Step Plan.	82
3.5.1 Iteration-1.	83
3.5.2 Iteration-2	84
3.5.3 Iteration-3.	88
4 Architecture	94
4.1 Importance of Architecture	94
4.2 Microservices Architecture	95
4.3 Slackbot Architecture	96
4.3.1 AWS Key Management Service (KMS)	97
4.3.2 AWS S3 Service	98
4.3.3 AWS Lambda Service.	98
4.3.4 AWS SNS Service.	104
4.3.5 List of AWS Services Used	105
4.3.6 Estimated Cloud Costs.	105
4.4 Cloud Application Security.	107
4.4.1 Unauthorised access to application functionality or data.	107
4.4.2 Exposed application services due to misconfigurations	108
4.5 End-to-End Testing	109
4.5.1 Benefits of End-to-End Testing	109
4.5.2 Test Results.	110

5	Ethical Concerns	112
	5.1 Large Language Models	112
	5.1.1 How do Large Language Models work	112
	5.1.2 Environmental costs from operating LLM.	113
	5.2 Data Protection in Cloud	113
6	Conclusion	115
	6.1 Missed Requirements	115
	6.2 Next Steps	116
7	References	118

ABSTRACT

Productivity plays a key role in an organisation's success. Companies often struggle to find solutions to increase workforce productivity. Inefficient business processes like poor accessibility to company wiki and inconspicuous workflows are usually at the forefront of the lack of productivity. Effective business processes dramatically increase productivity thereby making businesses more successful. This thesis aims to improve the productivity of OEV Online Dienste GmbH by enabling employee accessibility to the company wiki and enhancing visibility of workflows within the company by creating a streamlined process of automation in business processes. To achieve this goal, a comprehensive study of tools, methodologies and services, to improve existing business processes was performed. Following the study, empirical evidence was collected, for the hypothesis - "*automation improves business processes*" by performing theoretical analysis and prototyping a similar solution. The study provided the theoretical knowledge necessary to implement automation. Prototyping helped in getting hands-on experience on the technologies required to build the automation. Combining both theoretical and practical knowledge, an automation solution was developed using technologies and methodologies like cloud, chatOps, conversational AI, etc. The automated solution was implemented into the company's infrastructure and was tested end-to-end. Results of the end-to-end tests indicate that the automation solution worked as per requirements and helped in improving the productivity of the employees of the company by creating a streamlined process of business automation.

Keywords: Automation, workflows, cloud, chatOps, conversational AI

LIST OF FIGURES

Figure 1	Cerebras pricing model for Large Language Model Training
Figure 2	Project Requirement Document for PoC
Figure 3	PoC Architecture
Figure 4	Conversational User Flow for PoC
Figure 5	Basic Information of Smart Slack Bot
Figure 6	App Credentials for Smart Slack Bot
Figure 7	Slack Bot Token for Smart Slack Bot
Figure 8	Slack App Token for Smart Slack Bot
Figure 9	App Scopes for Smart Slack Bot
Figure 10	Bot Events Subscriptions for Smart Slack Bot
Figure 11	Redirect URLs for Smart Slack Bot
Figure 12	Dialogflow ES agent
Figure 13	SmartSlackBot Intents
Figure 14	Default Fallback Intent Responses
Figure 15	Default Welcome Intent & Contexts
Figure 16	Default Welcome Intent - Training Phrases
Figure 17	Default Welcome Intent - Responses
Figure 18	promptedContinue Intent - Contexts
Figure 19	promptedContinue Intent - Training Phrases
Figure 20	promptedContinue Intent - Response
Figure 21	promptedHelp Intent - Context
Figure 22	promptedHelp Intent - Training Phrases
Figure 23	promptedHelp Intent - Response
Figure 24	promptedEndConversation Intent - Contexts
Figure 25	promptedEndConversation Intent - Training phrase
Figure 26	promptedEndConversation Intent - Responses
Figure 27	Entities
Figure 28	Workflow Entity
Figure 29	endConversation Entity
Figure 30	Fulfillment Enabled
Figure 31	Iterative SDLC Model
Figure 32	Gantt Chart for Project Tracking
Figure 33	Business Process Model and Notation Diagram

Figure 34	Branching Storylines for dialogflow ES Agent
Figure 35	Dialogflow Demo - Welcome Intent
Figure 36	Dialogflow Demo - Welcome Intent for different input
Figure 37	Dialogflow Demo - Help Intent
Figure 38	Dialogflow Demo - promptedContinue Intent for Krank
Figure 39	Dialogflow Demo - promptedContinue Intent for Urlaub
Figure 40	Dialogflow Demo - promptedEndConversation Intent
Figure 41	AWS Cloudformation Stack Deployment Process
Figure 42	AWS Cloudformation template to provision AWS Lambda
Figure 43	Shell script code for deploying local codebase to AWS
Figure 44	System Design Architecture for Slackbot - High level overview
Figure 45	Postman - Dialogflow API test
Figure 46	Postman - Dummy function test
Figure 47	Slackbot Architecture
Figure 48	Slash command for Smart Slack Bot
Figure 49	Slackbot - E2E Testing Results
Figure 50	Slackbot Application v2.0

LIST OF TABLES

Table 1	Chatbot Storylines
Table 2	List of AWS services used and their costs
Table 3	Estimated Cloud Costs for Slackbot

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
GCP	Google Cloud Platform
PoC	Proof-of-Concept
MSA	Microservices Architecture
GPU	Graphic Processing Unit
BPA	Business Process Automation
IaC	Infrastructure-as-Code
CDK	Cloud Development Kit
SDK	Software Development Kit
UI	User Interface
RPA	Robotic Process Automation
SDLC	Software Development Life Cycle
CI	Continuous Integration
CD	Continuous Development
SRA	System Requirement Analysis
BPMN	Business Process Model and Notation
ChatOps	Chat Operations
DevOps	Development Operations
ITOps	IT Operations
E2E	End-to-End
SNS	Simple Notification Service
BERT	Bidirectional Encoder Representations from Transformers
AI	Artificial Intelligence
IRC	Internet Relay Chat
CX	Customer Experience
ES	Essentials

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][pg.2]. 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 their 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. This includes creating a culture of automation which is essential to an organisation's growth and success [4]. 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 [5]. 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 [6]. The goal of this master thesis is to prove that with the help of chatOps, the disconnected documents challenge that the company is facing can be solved through continuous automation and collaboration in a transparent manner.

In this thesis, the first step was to explore the challenges faced by the company - *OEV Online Dienste GmbH* due to disconnected document processes and roadblocks in team collaborations, which affects productivity. Then a solution is proposed 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, the developed solution is implemented into the company's cloud native environment and is integrated seamlessly into their existing infrastructure. Thereby making the solution accessible to all their employees.

1.1. 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), there are 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 cloud infrastructure including compute 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 [7]. Companies like Google have also opened up their Natural Language Processing (NLP) platforms like dialogflow where one can build highly advanced chatbots with lifelike conversational AI and integrate them into any platform [8]. Building private Large Language Models (LLM), such as Google's Bidirectional Encoder Representations from Transformers (BERT), which is used to create sophisticated chatbot agents in dialogflow [8] is close to impossible at the cost in which Google offers its dialogflow services for public use. There are companies like Cerebras and Jasper that operate on a supercomputing-as-a-service business model where they rent their supercomputers for Graphic Processing Unit (GPU) AI training [9]. *Figure 1* is a sample price quotation for the cost of training private LLMs on a public supercomputer with Cerebras. Based on the model which needs to be trained, the costs for training LLMs privately could vary between \$2500 to \$2,5 million per training which is really expensive, as compared to Google dialogflow costs, which is priced around \$0.002 to \$0.008 per request [10].

Model	Parameters	Tokens to Train to Chinchilla Point (B)	Cerebras Model Studio CS-2 Day to Train	Cerebras Model Studio Price to Train
GPT3-XL	1.3	26	0.4	\$2,500
GPT-J	6	120	8	\$45,000
GPT-3 6.7B	6.7	134	11	\$40,000
T-5 11B	11	34*	9	\$60,000
GPT-3 13B	13	260	39	\$150,000
GPT NeoX	20	400	47	\$525,000
GPT 70B	70	1,400	85	\$2,500,000
GPT 175B	175	3,500	Contact For Quote	Contact For Quote

* - T5 tokens to train from the original T5 paper. Chinchilla scaling laws not applicable.

Figure 1: Cerebras pricing model for Large Language Model Training¹

Finally, Slack which is a massive instant messaging platform on its own that provides professional and organisational communication services to thousands of companies [11], offers a variety of possibilities to integrate these external services like AWS or Google dialogflow into their platform to send and receive messages [12] using webhook integrations. 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

¹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-one/cerebras-model-training-price-table.jpg>

such as migrating softwares from monolithic architecture to microservices architecture (MSA). This has modified the way in which we build modern scalable applications [13]. As a software developer, these advancements have fascinated me and have been an inspiration for me to harness the power of these digital technologies in building scalable and reliable software applications that can help businesses and people by making their lives easier.

1.2. About Master Thesis

This master thesis has been one such opportunity for me to build a scalable software application using Slack, AWS and Google dialogflow to assist *OEV Online Dienste GmbH* in improving their existing employee management system and documentation process. The challenge that the company is facing at the moment is that their existing employee management system and company wiki does not offer clear visibility to their employees. For example, If an employee needs information regarding a project or a process, they have to manually scan through the 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 persisted in their company wiki, but also for other employee services such as applying sick leaves, submitting vacation requests, submitting invoices, initiating new project ideas, and requesting for project details.

In order to automate these processes, improve knowledge base visibility and boost employee productivity, a slackbot application is built that acts as a chatbot on the slack workspace 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 in the company's slack channel, the chatbot, which is installed in that channel listens to the keyword mentioned by the employee, gets activated and responds to the employee's query inside the same slack channel. This is the basic functionality of the slackbot application and ergo the topic of the master thesis. The goal of the slackbot application is to act as a knowledge distribution system, by guiding employees to information that they require at different stages of software development and also to 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 and business processes by creating a streamlined process of automation.

1.3. Thesis Structure

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 the planning stage till the monitoring stage.

1.3.1. Research and Background

First part of the thesis covers the research and background work involved in planning for the slackbot application. This part consists of two chapters - **Chapter one: Introduction** and **Chapter two: Status Quo**. Chapter one has 6 sections in total. Each section covers a particular topic involved in building the slackbot application.

1. Chapter one starts by giving a brief introduction to the thesis.
2. Following the introduction is **section 1.1**, which explains the motivation behind selecting the topic for the master thesis.
3. Next, **section 1.2**, gives an overview of the application that was built as part of the master thesis.
4. **Section 1.3**, the current section, explains the structure of the thesis so that it is easy for the readers to understand the topics being discussed in each chapter and section.
5. **Section 1.4** contains a general, non technical overview of the collaboration tools and processes used in building the application.
6. **Section 1.5** covers the different approaches used to write the thesis.
7. **Section 1.6** talks about the contributions involved in writing the master thesis.

Chapter two has only 2 sections in total. This chapter first introduces an existing prototype - a Proof-of-Concept(PoC) application which was also built for the same company and explains the reason why there was a need to build the slackbot application. Then it covers the technical configurations of the slackbot application.

1. In chapter two, **section 2.1** gives an overview of the implemented PoC application - its requirements, architecture, functionality, conversational user flows designed to communicate with the chatbot, the drawbacks of the PoC application along with the need for the slackbot application.
2. **Section 2.2** contains the technical overview of the collaboration tools and processes used in building the slackbot application.

1.3.2. Plan and Implementation

The second part of the master thesis discusses the development plan for building the slackbot application for the master thesis along with its architecture and implementation. This part consists of two chapters - **Chapter three: Development Plan** and **Chapter four: Architecture**. Chapter three covers the technical configurations and the step-by-step plan created to build the slackbot application.

1. Chapter three begins with the tech stack of the slackbot application in **section 3.1**.
2. In **section 3.2**, the focus will be on explaining about the software methodology used to build the application - the Software Development Life Cycle (SDLC) along with the reasons why it was chosen.
3. **Section 3.3** discusses the topic of System Requirements Analysis (SRA), Requirements Gathering, and Requirements Analysis that was performed before building the slackbot application.
4. **Section 3.4** will briefly walk through the conversational design made to communicate with the chatbot, a conversational Artificial Intelligence (AI) platform, which is used as part of the slackbot application.
5. In the final section, **section 3.5**, the step-by-step plan of building the slackbot application is explained along with the challenges faced at different stages of building the application.

Chapter four is for bringing together all the components of the slackbot application to create an architecture diagram and explain the functionality of all elements inside the application.

1. **Section 4.1** explains the importance of architecture in building software.

2. **Section 4.2** covers the topics of Microservices Architecture (MSA), what is MSA, advantages of MSA over other architectures, and why MSA was chosen for the slackbot application.
3. In **section 4.3**, the final slackbot application architecture is explained along with all the individual services and components involved in making the end product.
4. When a scalable cloud native application is built and implemented in a public cloud environment, it is important to discuss the potential security threats that the slackbot application could face. This is done in **section 4.4** along with the measures taken to mitigate the security threats.
5. Final section of chapter four, **section 4.5**, is to explain the test results of the End-to-End (E2E) testing that was performed on the application in the production environment.

1.3.3. Ethics and Conclusion

The final part of the master thesis is to have a discussion on ethical concerns regarding LLMs and data protection in the cloud. This part consists of two chapters - **Chapter five: Ethical Concerns** and **Chapter six: Conclusion**. Chapter five covers the dangers posed by LLMs on the environment and energy costs involved in operating and training LLMs.

1. **Section 5.1** covers the problems caused by LLMs to the environment and society.
2. **Section 5.2** explains the importance of data protection in the cloud along with the security risk of data breach.

Chapter six is the conclusion of the master thesis.

1. **Section 6.1** discusses the list of requirements that could not be included for the master thesis.
2. **Section 6.2** gives an overview of what the future of the slackbot application will be along with the architecture diagram for the next version.

1.4. General Overview of Collaboration Tools and Processes

Various collaboration tools and processes like slack, AWS, confluence and jira are already used in the company. There are some new collaboration tools like dialogflow, and processes like Chat Operations (ChatOps) and Software Development Life Cycle (SDLC) being introduced in the company as part of the master thesis. For the slackbot application, both existing and new set of tools and processes are used to build the end product.

1.4.1. Slack

Slack is an instant messaging platform designed by Slack Technologies LLC, and is owned by Salesforce [14]. Users can communicate with voice calls, video calls, text messaging, media and files in private chats or in parts of communities called the workspaces. Slack uses Internet Relay Chat (IRC) style [15] features such as persistent chat rooms, called channels. The slack channels are often organised by the topic discussed in these channels. People can also create private groups or use direct messaging to communicate with others in the slack application. In addition to these online communication features, slack also allows integration with external softwares and applications [16]. This is one of the main advantages of using slack in an organisation. Slack additionally offers chatbot-like features called in-builts apps which can be installed into any slack channel to perform certain tasks [17]. These in-built apps or chatbots can be found and installed from the app directory of slack application. Some examples of such in-built slack apps are: Google calendar, GitHub, AWS which are all connected to their respective external applications where users can use these in-built apps to perform, collaborate or monitor tasks inside slack.

Apart from the in-built apps, slack also allows users to build their own customised apps or chatbots to perform users specific tasks in their slack workspaces [18]. These custom slack apps can be used to perform any task in slack channels based on the user needs. Custom slack apps can also be integrated with external applications like AWS, salesforce, jira, confluence, etc. They can also be integrated with custom built applications like web or mobile applications using OAuth 2.0 authorisation [19]. OAuth 2.0 is the industry-standard protocol for authorisation. OAuth 2.0 focuses on client developer simplicity while providing

specific authorisation flows for web applications, desktop applications, mobile phones, and smart devices [20]. OAuth allows users to install the custom slack app in any slack workspace. For the master thesis, a custom slack app called **Smart Slack Bot** is created in slack to perform certain tasks that would improve team collaboration and productivity in the company. The technical details of the Smart Slack Bot is discussed in *section 2.2.1*.

1.4.2. **Dialogflow**

Dialogflow is a Natural Language Understanding (NLU) platform that makes it easy for developers to design and integrate conversational user interfaces (UI) into mobile applications, web applications, chatbots, interactive voice response system, etc. Dialogflow provides new and engaging ways for users to interact with other applications [21]. It can also be used to analyse multiple types of inputs from the users including text and audio. It responds to the user input, through text or with synthetic speech [21]. Dialogflow provides two types of virtual agent services, each of which has its own agent type, user interface, Application Programming Interfaces (APIs), and client libraries. The two different virtual agents are: **Customer Experience (CX)** and **Essentials (ES)**. Dialogflow Customer Experience (CX) provides an advanced conversational agent which is suitable to manage large and complex conversations. Dialogflow Essentials (ES) provides the standard conversational agent which is suitable to manage small and simple conversations [21]. For the master thesis, to build the slackbot application, dialogflow ES agent was chosen over the CX agent because the company's business requirement and the slackbot's architecture both required only a simple and direct conversation with the users. Therefore dialogflow ES agent was sufficient to handle the conversations and to perform Natural Language Processing (NLP). The technical details of the dialogflow ES agent used for the slackbot application is discussed in the *section 2.2.2* of the thesis document.

1.4.3. **AWS**

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform offering over 200 fully featured services from data centres around the world [22].

Millions of companies, from the fastest-growing startups to largest enterprises, leading government agencies use AWS to lower costs, become more agile, and innovate faster [22]. AWS has significantly more services and features than any other cloud provider - from infrastructure technologies like compute, storage, and databases to emerging technologies such as machine learning and artificial intelligence, data lakes and analytics, and IoT. This makes it faster, easier, and more cost effective for companies to move existing applications to the cloud and run their business from cloud [22]. AWS was selected as the cloud platform to develop the slackbot application because the company has a strategic partnership with AWS and a significant amount of the company's infrastructure is in AWS. Therefore it was the optimal choice to build the application on the company's existing infrastructure on AWS.

1.4.4. ChatOps

Chat Operations (ChatOps) is the use of chat clients and real-time chat tools to facilitate software development and operations [23]. ChatOps is also known as *conversation-driven collaboration* or *conversation-driven DevOps*. ChatOps is designed for fast and simple instant messaging between team members [23]. Throughout the chatOps experience, a chatbot accepts commands and initiates actions with background apps through APIs to optimise IT operations (ITOps) and development operations (DevOps) [23]. ChatOps is designed to eliminate information silos that hinder interdepartmental collaboration and proactive decision-making. As a result, it benefits entire teams with automation, collaboration, engagement, productivity, and transparency. Both individually and collectively, these benefits ultimately strengthen delivering results in the company by speeding up team communications [23] which in turn shortens the development pipelines and incident response time. In the master thesis, chatOps is used as the methodology to build the slackbot application because it perfectly synchronises with the ideologies of the master thesis - improve team collaboration, increase productivity, promote automation, etc. all of which is part of the goal of the master thesis. To implement chatOps in the slack application, 2 chatbots are used - **Smart Slack Bot**, as introduced in *section 1.4.1* and the dialogflow ES agent. Both chatbots are used to perform certain tasks to achieve the goals of the master thesis.

1.4.5. Confluence

Confluence is a web-based corporate wiki developed by the Atlassian software company [24]. Confluence is used widely by tech 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. *OEV Online Dienste GmbH* also uses *confluence* as the company wiki. The company's knowledge base regarding projects, budgets, team members, onboarding and offboarding employees, etc are persisted in the company's confluence page. Confluence is used by the company for also 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. Apart from technical knowledge, the confluence also contains information about employee services such as applying sick leaves, submitting vacation requests, submitting invoices for reimbursement of bills, employee onboarding steps, etc. In the context of the master thesis, ergo the slackbot application, the company confluence is the main source of destination where the users are redirected to whenever a request is made through the slackbot application.

1.4.6. Jira

Jira is a work management tool developed by the same company, Atlassian software, that developed confluence [25]. Jira is mainly used as a tool for team collaboration among tech companies. The company uses Jira for ticket management, logging and resolving bugs, project management activities such as organising agile meetings, maintaining kanban boards, etc. Since the slackbot application is developed as a project within the organisation, it follows the same documentation methodologies as followed in other projects of the company. Hence Jira is used for project management of the slackbot application.

1.4.7. Software Development Life Cycle

Software Development Life Cycle (SDLC) is a structured process that enables production of high quality, low cost software in the shortest possible production time [26]. 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 [26]. 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 are used to develop software. Some of the common ones are waterfall model, iterative model, v-shaped model, spiral model, agile model, etc [27]. For the thesis, the iterative SDLC model is used for developing the application. The technical details of this model are discussed in *section 3.2*.

1.5. Thesis Approach

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

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

1.5.1. Theoretical Analysis

First approach is the theoretical analysis part where a comprehensive background study of the existing collaboration tools, company 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 fit the requirements criteria in solving the company specific problem and not just a general problem. The documentations helped understand the capabilities and functionalities of existing collaboration tools and the research papers gave an in-depth knowledge of how certain methodologies like chatOps can significantly improve an existing system through automation [28]. The theoretical analysis, as a whole, proved to be valuable as it gave an idea of how the slackbot application can be designed. It also provided the

scientific validation that certain methodologies and tools were successful in the past in solving a similar challenge. The results of this analysis are explained in *section 1.6.1*.

1.5.2. Natural Language Processing

The second approach is the Natural Language Processing (NLP) where the best suited conversational AI platform was chosen to build the application. Once chatOps was selected as the methodology to build a conversational experience with the slackbot application, there were two options available - option-1 was to create the LLMs from scratch using a transformer model for the NLU task and build a chatbot on top of it. This would mean that the 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 collected to train the LLMs privately, the datasets had to be labelled and fine tuned regularly with new data in order to have a good responsive NLP based chatbot. Apart from these challenges, there was also the need for intensive compute power like Graphic Processing Units (GPUs) and specialised hardwares to train the LLMs quickly and efficiently.

Even if the necessary data were collected, labelled and trained on new data regularly with the required 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. Hence building LLMs from scratch seemed time consuming, ineffective and more expensive when there were already good LLMs available for the public use offered by cloud providers like Google on the Google Cloud Platform (GCP). Therefore, option-2, which was to select an existing conversational AI that was available for rent which allowed users to build chatbots on top of it for the slackbot . These requirements were matched by dialogflow as it has an advanced LLM and at the same time allowed users to create chatbots that were powered by these advanced LLMs. The cost of the ES agent was also inexpensive as compared to building and training private LLMs as training and running LLMs privately would cost millions of dollars [54] as discussed in *section*

1.1. Therefore it was the clear choice to rent the existing conversation AI platform from Google considering the time, cost and efficiency in building the slackbot application.

At this point, the application architecture had already started to take shape. It was decided that the collaboration tool was Slack, the methodology was chatOps, the NLP platform was the GCP and the chatbot was dialogflow ES agent.

1.5.3. Cloud Computing

The third and final approach of the master thesis was to bring all these services and tools under one roof in a public cloud environment where all of them can be integrated seamlessly to create a scalable, reliable, cloud application. Although there were many public cloud providers in the market like GCP, Microsoft Azure, Oracle cloud, etc. AWS was chosen as the preferred cloud platform to integrate the tools and services. This was mainly due to the reason that the company already had a strategic partnership with AWS as explained in *section 1.4.3*. Therefore it was easier to build the application on their existing infrastructure on AWS and integrate the application seamlessly into their cloud environment. 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 requirements of the application and of the company.

These 3 approaches formed the basis of the master thesis and have contributed equally in building the end product. Their contributions are further subdivided into two categories - Theoretical and Empirical which are explained in the following section.

1.6. Contributions

To explain in terms of contribution to the master thesis, there were 2 contributions that were detrimental. They were:

1. Theoretical Contributions
2. Empirical Contributions

1.6.1. Theoretical contributions

Theoretical contributions come from the theoretical analysis which was performed in *Section 1.5.1*. The results of that analysis and how it contributed to the master thesis is discussed as the theoretical contributions in this section.

1.6.1.1. Business Process Automation (BPA)

The first topic which was studied as part of the 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 master thesis is to automate business processes to ensure more efficiency in the company. BPA describes a situation where a business process is executed without any human intervention, that is, when a task is implemented through software and is executed behind the scenes, on the schedule or automatically [29].

BPA provides certain benefits like higher productivity, improved efficiency, less human error, allows employees to focus on important things, reduces operating costs, etc to companies. Even though BPA offered 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 [30]. Therefore, with the help of theoretical contributions derived from the theoretical analysis, it was possible to mitigate these 2 top pitfalls 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. RPA and no code applications may help 100s of businesses to improve their processes through automation but there is no guarantee that they can solve the company's specific problems. This was the reason why the plan from the beginning was to build a customised, automated solution that solves the

company specific problems rather than a generic solution. There were also other considerations from the theoretical analyses which became part of the theoretical contribution:

1. Not automating broken processes

The current process of knowledge sharing in the company is detailed and well executed. The only step that was missing from making the knowledge sharing process more efficient in the company 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.

2. Picking automation tools that correspond to the company's business model

The tools that were selected for building the slackbot application include slack, AWS, and dialogflow. Slack and AWS, as mentioned in sections 1.4.1 and 1.4.3, were already used by the company due to their strategic partnership and licensing. Dialogflow ES was the best available option for the company specific use case as already discussed in section 1.4.2. Therefore, these automation tools were in line with the company's business model and also with the requirements of the slackbot application.

1.6.1.2. 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 type of the slackbot application. After considering various options, MSA was chosen as the final architecture type over the other common types of architectures like monolithic or layered. This is because each microservice in MSA can be independently developed, updated, deployed, scaled or replaced. This freedom was not possible in other architecture types like monolithic architecture since all the elements in the monolithic architecture are built as a single unified unit. Splitting independent services or scaling them up or down was not possible with breaking down the entire application architecture. This was an important consideration in building the slackbot application for the master thesis 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 dialogflow ES to another product that better suits the company's needs in the future, then they don't have to break down the whole application architecture. They can just replace the dialogflow's API with the new product's API and the application will function perfectly well. This makes MSA agile. Therefore MSA was the best option for the slackbot application and for the company in the future to compete in an ever changing tech environment.

1.6.2. Empirical Contributions

As theoretical contributions came from the theoretical analysis, empirical contributions come from the practical work which was done before writing the master thesis. The empirical contribution encompasses the two remaining concepts of the thesis approach: *Section 1.5.2* and *Section 1.5.3*.

The theoretical analyses provided the knowledge and a hypothesis but there was no definitive proof that these concepts will work in real-life applications. This is where empirical work became very important. It provided an opportunity to test the hypothesis, build scalable applications and deploy it in the production environment where actual users can interact with the application. Before building the slackbot application for the master thesis, a similar but a smaller application was developed as a Proof-of-Concept (PoC) for the company. The PoC application was a predecessor to the current slackbot application. Therefore the same set of platforms which were used for the PoC were also used for building the slackbot applications - slack, AWS and dialogflow ES. Successfully deploying the PoC application in the production environment provided the first empirical contribution with a solid proof that if implemented correctly the slackbot application, which is a larger version of the PoC, will also work in the production environment. The second empirical contribution which formed the basis for the master thesis was in the domain of IT security. Both these empirical contributions are discussed in the following subsections.

1.6.2.1. Proof-of-Concept (PoC)

The functionality of the PoC application was to initiate slack workflows through a chatbot installed in the company's slack channel. The PoC application provided the empirical contribution needed for the slackbot by providing a hands-on experience in chatOps. ChatOps was a theoretical methodology with only theoretical knowledge gathered from studying research papers and documentations. There were no opportunities to test its validity before developing the PoC and implementing chatOps in production. PoC was a successful project. It proved in practice how chatOps contributes to improving company processes through automation. The PoC application improved certain processes inside the company which was the goal of the PoC project. This led to the definitive proof that automation through chatOps could most definitely solve the company's disconnected documents problem and remove roadblocks in team collaboration which was hypothesised in the "Introduction" section of chapter one in this document. ChatOps also includes the topic of workflow automation, another methodology which was tested in the PoC application.

Workflow Automation

A workflow refers to the series of activities needed to complete a task [31]. Workflow automation refers to implementing software that can complete tasks which were manually managed before. Automation software company, *Zapier*, surveyed and found that 94% of the small and medium-sized business workers said they perform repetitive, time-consuming tasks [32]. Zapier also noted that 90% of knowledge workers found that automation improved their jobs and 66% said automation has made them more productive [32]. Automating workflows in the slack application was a crucial requirement from the company's perspective for both PoC and the slackbot application. In order to successfully implement workflow automation in PoC, the below 5 steps were followed which were learnt from the background research performed for building the PoC application [33]:

1. **Identify the process** - Taking a note of all the processes that needs to be automated in the organisation is the first step of successful workflow automation. 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 time and resources for both the automation engineer and 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 an important step in designing the automation software. This is done so that the software covers the entire workflow from the beginning till the end.
3. **Define business goals** - This is to understand the business goal in automating the workflow. As the person designing the automation software, it is important 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 which can be achieved through automation.
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 in improving the workflow automation.

These steps were successful in building the PoC application which successfully automated certain workflows in the company. Therefore these proven 5 steps were also followed to build the slackbot which performs similar workflow automation for the company as part of the master thesis. The in-depth details of the PoC application, its requirements, slack workflows, etc are explained in chapter two of the master thesis.

1.6.2.2. IT Security

Second empirical contribution for the thesis was in the domain of IT security. One of the important concepts that was not much focused while building the PoC was IT security. There were no implicit security systems built to protect the PoC application or the data which was shared across different platforms. PoC was sustained only by the built-in security features of Google dialogflow, slack and AWS services like Lambda and API gateway. Since it was only a proof-of-concept, there were no major impacts due to lack of security systems, but this was

not the case with the slackbot application as it is a complete, scalable application that real users will interact with. IT Security proved to be a valuable empirical contribution not in terms of what was successful but in terms of what was missing and what could be a potential risk in the future. Therefore, for the master thesis, it was planned in advance that there will be more focus on IT security as it was important to protect both the application data and the user data from intentional or accidental breach.

Theoretical contributions provided the knowledge and information necessary to design the application but empirical contributions in the form of PoC gave both, a practical experience and a definitive proof that by combining the tools and methodologies that were studied in theory, a complete microservice-based, secure, scalable, and reliable cloud application can be built and successfully deployed in production environment that will most likely 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*, the slackbot application was not a completely new application but was rather an improvisation on an existing PoC. In this chapter, the existing system which formed the foundation for the slackbot application will be explained in detail.

2.1. Implemented PoC

It is important to discuss the existing system to understand the master thesis because this system was the foundation for the slackbot application. The thesis application takes a lot of functionalities from this system and hence it is necessary for the readers to fully comprehend the PoC before introducing the current slackbot application. The PoC application used a custom built slack app as the frontend for the employees of the company to interact with the application. Slack was the frontend of the PoC. The backend was created using AWS and Google dialogflow. Dialogflow ES agent was used for the NLU part similar to the slackbot application. AWS was used for the infrastructure, compute and API services which formed 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 and all 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 was on AWS, Cloud Development Kit (CDK) of AWS was used to deploy the entire PoC infrastructure to the cloud. AWS lambda function served as the compute service in the backend. It was written in python 3.8 and the CDK stack which served as the IaC was

written in typescript. This is the brief overview of the functionality and the software stack of the PoC application.

2.1.1. Requirements Document

Figure 2 was the project goal that was defined by the company as requirements for developing the PoC application.

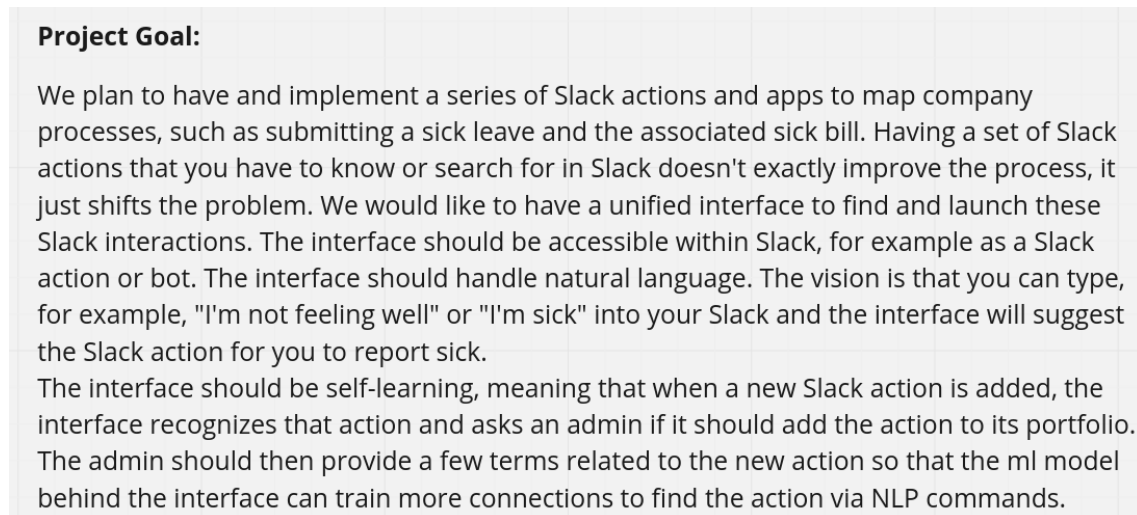


Figure 2: Project Requirement Document for PoC²

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 installed inside the slack channel. The company also requested for an app that can perform NLP actions and understand what the users say without the users having to explicitly mention a keyword to initiate the workflows. Based on the project requirement provided by the company as the problem statement, the PoC application was developed successfully.

2.1.2. Architecture and Functionality

The architecture, in Figure 3, shows how the PoC was built using slack, dialogflow ES agent and just 2 AWS services - lambda and API gateway. These services integrated together to form the PoC application.

² <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/poc/PoC-requirement.png>

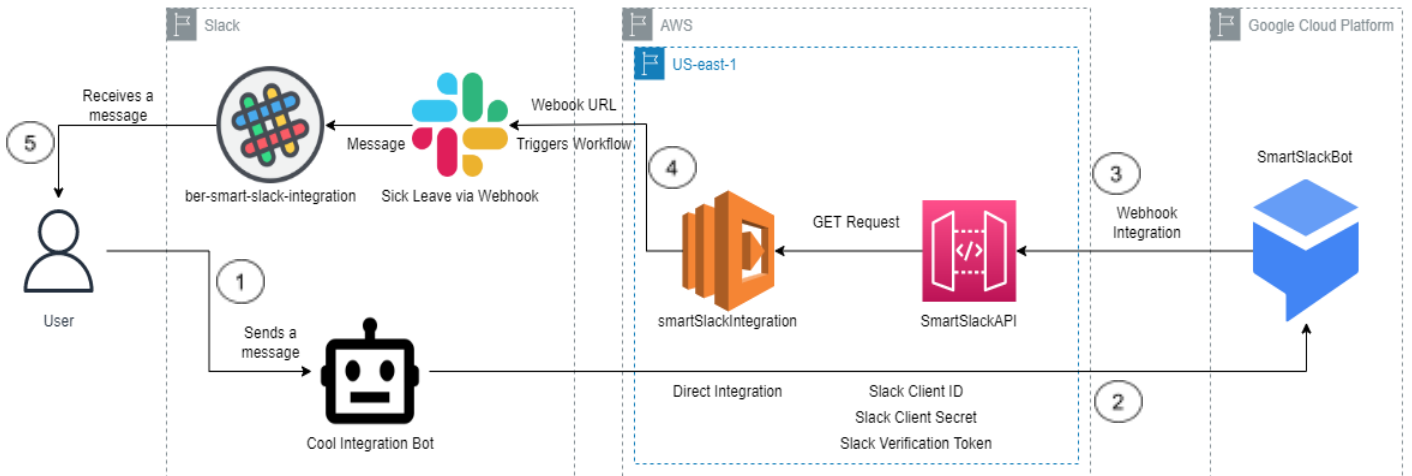


Figure 3: PoC Architecture³

The functionality of the PoC application can be summarised in 5 simple steps. These steps are also marked in the architecture diagram 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 custom slack app that is installed in the company's slack channel by sending a message to it. A *sick* workflow is a slack workflow which was created by the company. It informs users on the steps to be followed to apply for a sick leave in the company.
2. The slack app/chatbot called *Cool Integration Bot* was created and integrated directly with the dialogflow ES agent by sharing its *client ID*, *client secret*, and *verification token*. These 3 tokens are unique to each custom slack app. They are generated when a new app is created in the slack webpage [34]. These 3 tokens provide the basic security to the custom slack apps. When users send a message to the slack app, the message is sent directly to the 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 to AWS through a webhook integration. A webhook integration is used for event-driven integrations and is one of the many ways applications can communicate with each other. They allow users to send real-time data from one system to another when a given event occurs [55]. The response is sent to AWS and not back to the slack app because a decision needs to

³ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/poc/PoC-architecture.png>

be made whether to initiate a slack workflow or not based on each message sent by the user through the slack app. If a user sends a greeting message like “Hi” or “Good Morning”, the slack workflow does not have to be initiated. This decision is made in the AWS lambda function. The API gateway acts as the API between dialogflow and AWS. A webhook integration was made between API gateway and the ES agent. The API gateway service, in this case, was a REST API that was allowed to perform 2 methods - *GET* and *POST*. When the response to a user’s input message was generated in dialogflow, an event is triggered and this event sends the ES agent’s response to the REST API. This is where the dialogflow response is transferred from the GCP to AWS. The ES agent also sends other conversational parameters along with the agent response as part of the chatbot recognition capabilities. These parameters are discussed in detail in chapter three. The response received by the API gateway is a *POST* method and since the API gateway is connected to lambda, the lambda function gets invoked on receiving the JSON payload from the REST API’s *POST* method. Lambda 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.

4. Once the lambda function receives the JSON payload, the decision to whether initiate a workflow or not is made based on the value of the json parameter - *workflow* coming from dialogflow as part of the API payload. There are 3 workflows that were created in dialogflow, one workflow for each 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 payload has any of the 3 above mentioned values. If yes, then 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 ES agent’s advanced NLU capabilities. The lambda service is only used to make the decision and trigger the slack workflows by invoking the workflow’s webhook URLs. Slack workflows have their own webhook URLs which when invoked initiates the workflow in the slack channel [35]. Lambda sends all the responses

received from the dialogflow to the user in the custom slack app through another webhook integration between AWS lambda and the custom slack app. The user receives a response irrespective of whether the slack workflow is initiated or not.

5. The user who initiated the conversation in the slack app waits 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 user is satisfied.

This is the end-to-end functionality of the PoC that was built before starting the master thesis.

2.1.3. Conversational User Flows

The conversation user flow created for the PoC is depicted in the flowchart, in *Figure 4*. The flowchart is created to display the end-to-end interaction between the user and the highly advanced NLP based ES agent. It is also colour coded for easy comprehension of the dialogue structure. There are 3 major components in the flowchart:

- A. The curved green rectangular component denotes the entry or exit points where the user initiates or ends the conversation with the chatbot.
- B. Yellow rectangular components are the actions taken by the ES agent based on user input or existing context of the conversation.
- C. 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 chronologically for the readers to follow the user 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 the conversation ends, all the contexts of the conversation are deleted and a new context is created for each new conversation.

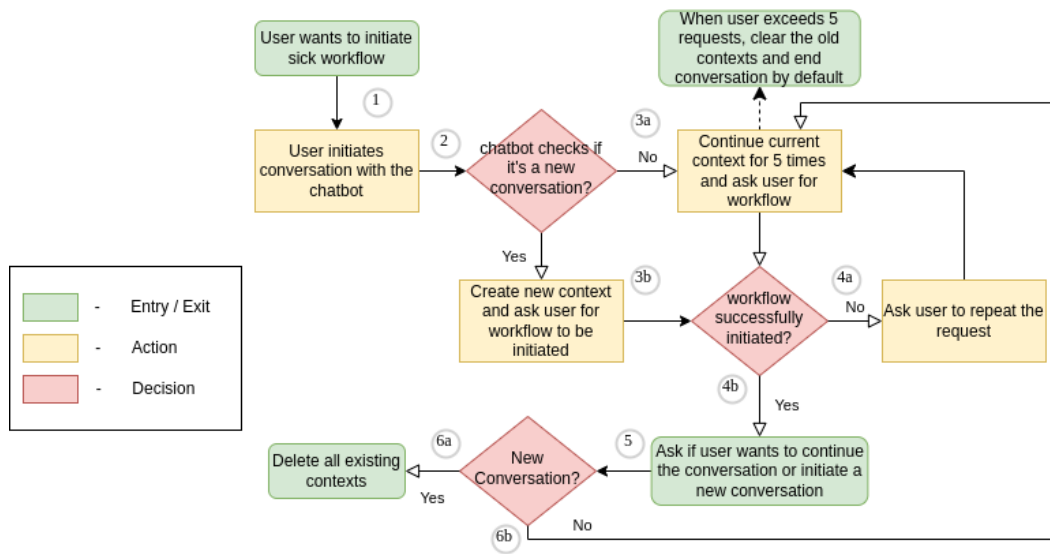


Figure 4: Conversational User Flow for PoC⁴

2.1.4. Additional Requirements for Improvisation

The reason for the PoC being just the first step and not a full application is because it was only semi-automated. Even though the workflow initialisation process in the backend was automated. It was only after the user initiated the conversation with the chatbot in slack the workflow automation began. Therefore it was not as effective as full automation. Users had to manually go to the slack app and initiate conversations with the chatbot to perform the slack workflow initiation process. This was a concern for both parties involved 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 automatically without manual initiation from the slack app. Therefore it was decided that the basic functionality of the slackbot application was to:

1. Listen to all conversations in the slack channels using a custom slack app and provide suggestions to the users to initiate workflows or redirect to confluence pages when

⁴ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/poc/chatbot-flowchart.png>

users talk about a particular scenario in the slack channels - applying sick leave, submitting vacation requests or submitting invoices.

2. Provide project details to the users on the slack when users request for information on a particular project whose details are persisted in the confluence webpage.

2.2. Technical Overview of Existing Tools and Processes

The existing processes and tools used by the company for documenting business processes, team collaborations, and maintaining knowledge bases about projects, teams, organisational procedures and other essential information about the company were already introduced in *section 1.4*. In this section, the technical aspects of each of these collaboration tools and processes are discussed.

2.2.1. Slack

For both PoC and for the master thesis, custom slack apps were built inside the slack platform which was used as the frontend for users to interact with the application. For the master thesis, as already mentioned in *section 1.4.1*, the custom slack app called Smart Slack Bot was used to perform the task of workflow initiation by triggering the slack workflow when a user mentions words or phrases that correspond to a particular workflow [36]. These slack workflows [37] were created by the company to help employees apply for sick leaves, submit vacation requests or submit invoices. Slack workflows are a series of actions and reactions that can be triggered by the users manually or by an external application. These workflows guide users to perform step-by-step activities inside the slack application. Custom slack apps can be created from the slack API⁵ webpage. Below are some of the technical configurations of the Smart Slack Bot which was created as the frontend for the master thesis:

⁵ <https://api.slack.com/>

Basic Information

Figure 5 is a screenshot from the home page of Smart Slack Bot from the slack api website. This is the display information section which determines how users will see the custom slack app in one of the company's slack channels. The custom slack app can be given an app name, short description, long description and an app icon to distinguish it from other apps in the slack workspace. Figure 5 shows the display information of the Smart Slack Bot which was configured to act as the frontend of the slackbot application.

Display Information

This information will be shown in the Slack App Directory and in the Slack App
For more information, view our [App Detail Guidelines](#).

App name	Short description
Smart Slack Bot	Chatbot that offers links to collaboration tools

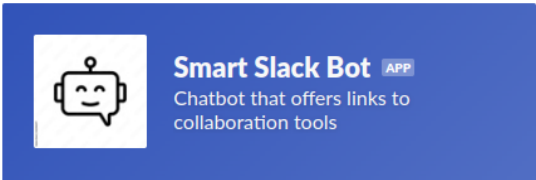
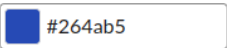
App icon & Preview	Background color
 The preview shows a blue rectangular card. On the left is a white icon of a robot head with a speech bubble. To the right of the icon, the text 'Smart Slack Bot' is displayed in white, followed by a small 'APP' badge. Below this, the text 'Chatbot that offers links to collaboration tools' is shown in a smaller white font.	 A blue square color swatch followed by the hex code #264ab5.

Figure 5: Basic Information of Smart Slack Bot⁶

App Credentials

App credentials for every custom built slack app can be found in the basic information page of the slack api website. It is the same page where the display information was captured from in the previous subsection. For the Smart Slack Bot, Figure 6 shows the app specific credentials generated by slack when the app was created. These credentials are used to connect the app to external applications like dialogflow through oauth which was explained in section 2.1.2. This type of connection was made for the PoC application.

For the master thesis, the Smart Slack Bot is connected to AWS lambda directly and not to the dialogflow agent. The details of this integration is discussed in depth in section 3.5 and section 4.3 of the thesis document. Therefore two different tokens - *Slack App Token* and *Slack Bot Token* were used to integrate the Smart Slack Bot with AWS lambda. App credential

⁶ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/basic-information.png>

information such as App ID and App Creation Date are generic but Client ID, Client Secret, Signing Secret and Verification token are confidential information through which the custom slack app can be accessed, modified and controlled by any person or application. For this security reason, both Client Secret and Signing Secret are hidden by slack by default. Whereas Client ID and Verification token are manually hidden.

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID	Date of App Creation
A04CKDJFQSC	November 21, 2022
Client ID	
[REDACTED]	
Client Secret	
.....	Show Regenerate
You'll need to send this secret along with your client ID when making your <code>oauth.v2.access</code> request.	
Signing Secret	
.....	Show Regenerate
Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.	
Verification Token	
[REDACTED]	Regenerate
This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.	

Figure 6: App Credentials for Smart Slack Bot⁷

Bot Tokens

Slack bot tokens or bot user oauth tokens, as shown in *Figure 7*, are generated automatically when a custom slack app is installed in a slack workspace. This bot token is used to perform oauth integration with other applications, as mentioned in *section 1.4.1*. For the slackbot application, these slack bot tokens are provided in AWS lambda so that the compute service can access the Smart Slack Bot from the cloud environment. Bot tokens always start with "xoxb-" followed by a series of alphabets and numbers which can be used to identify them among other slack configurations.

⁷ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/app-credentials.jpg>

OAuth Tokens for Your Workspace

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. [Learn more.](#)

Bot User OAuth Token

xoxb-

Copy

Access Level: Workspace

Figure 7: Slack Bot Token for Smart Slack Bot⁸

App Tokens

On the other hand, slack app tokens, shown in *Figure 8*, are manually generated from the basic information page of the slack api website if the custom slack app needs to be accessed from outside the slack environment.

slack-app-token

×

Properties of an app level token

Generated By	Date Generated
Rishi Srinivasan	November 22nd, 2022

Token

Copy

Revoke

Scope

connections:write	Route your app's interactions and event payloads over WebSockets
-------------------	--

Figure 8: Slack App Token for Smart Slack Bot⁹

For the slackbot application, the slack app token is provided to AWS lambda so that the compute service can start the app using the app token and listen to the conversations in the slack channel. It is also possible to restrict the app token's scopes to certain functionalities based on the requirements of the custom slack app. For example, in the app token that was created for Smart Slack Bot, the scope is restricted to "connections:write" as shown in *Figure 8*. Therefore, even if AWS lambda can access the custom app through the app token, it can only route the app's interactions and event payloads over websockets. Slack does not allow AWS lambda to perform any other functions outside the app token's scope.

⁸ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/bot-token.jpg>

⁹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/app-token.jpg>

App Scopes

App scopes, as mentioned in the app tokens subsection, define the capabilities and permissions that a custom slack app can have on a slack channel.

Scopes

A Slack app's capabilities and permissions are governed by the scopes it requests.

Bot Token Scopes

Scopes that govern what your app can access.









OAuth Scope	Description	
app_mentions:read	View messages that directly mention @Smart Slack Bot in conversations that the app is in	
channels:history	View messages and other content in public channels that Smart Slack Bot has been added to	
channels:read	View basic information about public channels in a workspace	
chat:write	Send messages as @Smart Slack Bot	
chat:write.customize	Send messages as @Smart Slack Bot with a customized username and avatar	
chat:write.public	Send messages to channels @Smart Slack Bot isn't a member of	
commands	Add shortcuts and/or slash commands that people can use	
groups:history	View messages and other content in private channels that Smart Slack Bot has been added to	
im:history	View messages and other content in direct	

Figure 9: App Scopes for Smart Slack Bot¹⁰

Slack offers many scopes to choose from based on the functionality and requirements of the custom slack app. For Smart Slack Bot, *Figure 9* shows some of the scopes that were selected to define the functionalities of the Smart Slack Bot in the company's slack channel. Each scope has a description of what it allows the custom slack app to do. Thereby making it

¹⁰ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/scopes.png>







easy for the users to select the required app scopes based on the custom slack app functionality. Any number of app scopes can be added or deleted at any point of time while using the custom slack app. The custom slack app needs to be reinstalled after every scope is added or deleted to redefine the functionalities of the custom slack app in the slack channel.

Bot Events Subscription

Bot events are the events that a custom slack app can subscribe to. These events allow the custom app to receive and react to certain events that happen in the slack channel where the custom app is installed.

Subscribe to bot events

Apps can subscribe to receive events the bot user has access to (like new messages in a channel). If you add an event here, we'll add the necessary OAuth scope for you.

Event Name	Description	Required Scope	
app_home_opened	User clicked into your App Home	none	
app_mention	Subscribe to only the message events that mention your app or bot	app_mentions:read	
message.channels	A message was posted to a channel	channels:history	
message.groups	A message was posted to a private channel	groups:history	
message.im	A message was posted in a direct message channel	im:history	
message.mpim	A message was posted in a multiparty direct message channel	mpim:history	

[Add Bot User Event](#)

Figure 10: Bot Events Subscriptions for Smart Slack Bot¹¹

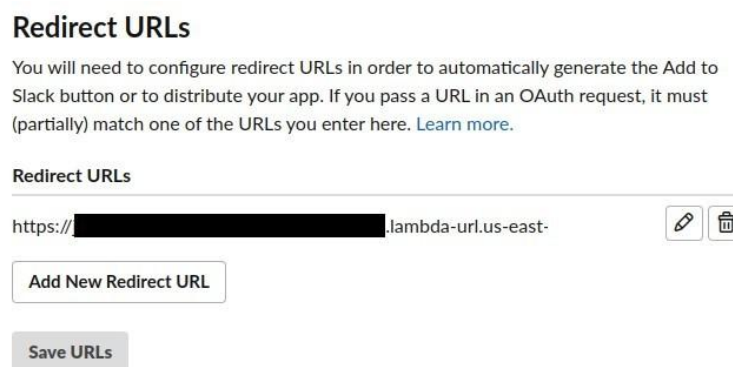
Figure 10 shows the list of events that the Smart Slack Bot is subscribed to. Each event has a name, description and required scope to explain what the event does. Therefore the number

¹¹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/bot-event-subscriptions.png>

of events that Smart Slack Bot has been subscribed to can be seen by looking at each event and their respective descriptions. Slack automatically adds the required oauth scopes to the events that are subscribed by the custom slack app. Therefore only the event name needs to be chosen for the events that the custom slack app wants to receive in the slack channels.

Redirect URLs

Redirect URLs are used to distribute custom slack apps to external applications or to install them in a slack workspace in general. For the slackbot application, AWS lambda was connected to the Smart Slack Bot directly therefore the redirect URL provided in *Figure 11* was the lambda function URL. The redirect URL creates an event based webhook integration between the Smart Slack Bot and AWS lambda keeping the connection alive thereby allowing lambda function to start the Smart Slack Bot and perform actions on it.



The screenshot shows the 'Redirect URLs' configuration page in Slack. At the top, there is a heading 'Redirect URLs' followed by a paragraph explaining that these URLs are used to generate the 'Add to Slack' button or to distribute the app. Below this, there is a section titled 'Redirect URLs' with a text input field containing a partially redacted URL ending in '.lambda-url.us-east-'. To the right of the input field are icons for editing and deleting. Below the input field is a button labeled 'Add New Redirect URL'. At the bottom of the section is a button labeled 'Save URLs'.

Figure 11: Redirect URLs for Smart Slack Bot¹²

2.2.2. Dialogflow

Dialogflow ES agent, as introduced earlier in *section 1.4.2*, has one of the most advanced LLMs available in the market that allows users to build chatbots on top of an advanced conversational AI platform. This was one of the main reasons for using the ES agent in the slackbot application to satisfy the NLU requirements. Dialogflow ES agent created for the master thesis is called *SmartSlackBot* and can be seen in *Figure 12*. The agent is located in a global data center and is bilingual. Users can interact with the agent in both German (de) and in English (en) languages. The default language of the agent is German since it is the

¹² <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/slack/redirect-url.jpg>

official language spoken in the company but the agent can also understand and respond to user inputs in English. A dialogflow agent is a virtual agent that can handle concurrent conversations with end-users. Dialogflow agents need to be designed and built to handle the types of conversations required for the needs of the application [56]. The conversational design and how the SmartSlackBot responds to each user input is discussed in section 3.4.4 of the thesis document.

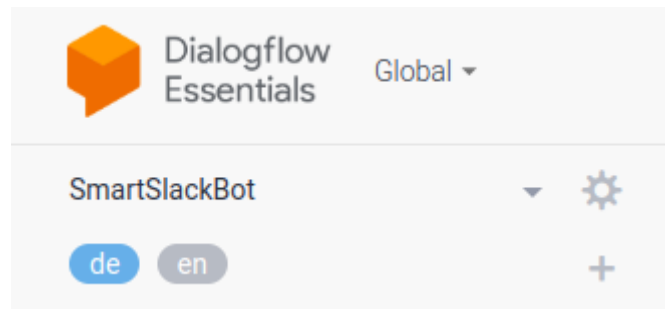


Figure 12: Dialogflow ES agent¹³

In this section, the technical configurations of the dialogflow agent that helps capture user inputs and categorises them, are discussed. Some of the technical configurations created for SmartSlackBot as part of the master thesis are:

Intents

Intents are created in the dialogflow to capture the intent of one conversation turn - one user input and the agent's response. An intent categorises the user's intentions into one of the predefined intents. For each dialogflow agent, intents are defined by the creator of the agent. The intents defined for an agent help understand and categorise user inputs thereby managing the whole conversation. When an end-user writes to the agent, this input is called end-user expression. Dialogflow matches the end-user expression to the best intent available in the agent. This matching of an intent is known as Intent classification. For the SmartSlackBot agent, there are 3 intents that were created specifically to suit the use case scenario of the slackbot application. Apart from the 3 new intents, 2 existing intents were also used and modified to suit the application requirements.

¹³ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/dialogflow-es-agent.png>

An intent basically contains the following parameters in them:

1. Training phrases - These are the example phrases for what the end-users might say to the agent as input. These phrases are created for each intent in an agent. When an end-user's input resembles one of these phrases, dialogflow matches the user input with the closest intent. In other words, this is the training data for the machine learning model used by dialogflow. Not every possible user input needs to be defined in the training phrases of an intent because the NLU capabilities of dialogflow expands the created list of training phrases with other similar phrases using NLP. For the SmartSlackBot agent, there were many training phrases created to resemble the possible user input for each intent.
2. Action - An Action can be defined for each intent created in an agent in dialogflow ES. When an intent is matched, dialogflow provides the action to the external application that is connected to the ES agent through a webhook integration or other form of API integration. The application may use the action sent by dialogflow to trigger any actions defined in its system. For example, in the slackbot application, for each user input, the AWS lambda which is connected to the dialogflow agent receives an action from the dialogflow as part of the agent's response. The slackbot application decides what to do with the action parameter that it receives from the dialogflow ES agent.
3. Parameters - When an intent is matched to the user input at runtime, dialogflow sends certain values extracted from the user's input as parameters. Each parameter has a type called the entity type which dictates exactly how the data is extracted from a user input. Unlike raw user input, parameters are structured data (usually json payload) that can easily be used to perform some logic or generate actions in applications connected to the dialogflow agent. The entity types are important in creating an ES agent. This will be discussed later in this section.
4. Responses - When the users define text, speech, or visual responses to the dialogflow agent. The agents process each user input, extract relevant information like parameters, actions and intent and provide the users with responses that the user expects or if the agent is unable to comprehend the user input, it asks the users

for more information. If the conversation is completed or a certain number of conversational turns are completed then the conversation is terminated by sending the information to the user saying the conversation has ended. This is called the response. For each user input, dialogflow generates a response.

For SmartSlackBot ES agent, there are 5 intents which were created and is being used as shown in *Figure 13*:

1. Default Fallback Intent
2. Default Welcome Intent
3. promptedHelp Intent
4. promptedContinue Intent
5. promptedEndConversation Intent

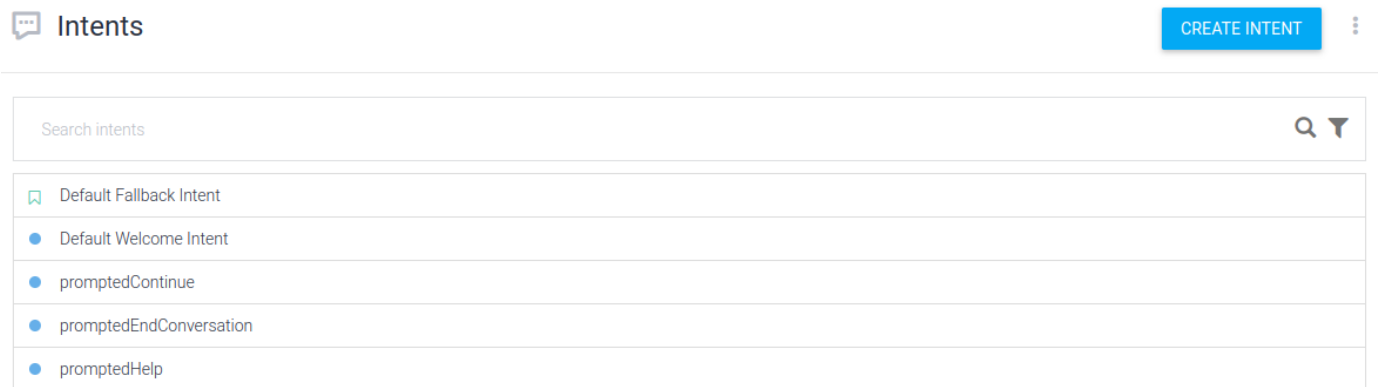


Figure 13: SmartSlackBot Intents¹⁴

1. Default Fallback Intent

The Default Fallback Intent is an existing intent that is automatically created when a virtual agent is created in dialogflow ES. This default fallback intent is triggered if the user's input does not match any of the regular intents or its training phrases. If an agent cannot understand the user input then this intent is matched. This is why it is called the fallback intent. Default fallback intent has its own predefined values like actions and responses that come by default but it can be modified to suit the requirements of each project requirement.

¹⁴ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/fallback-intent.png>

For the slackbot application, the default fallback intent's responses were modified to suit the use case of the master thesis. *Figure 14* shows the 5 different responses that SmartSlackBot could send as a response to the user if the agent could not recognise the user's input.

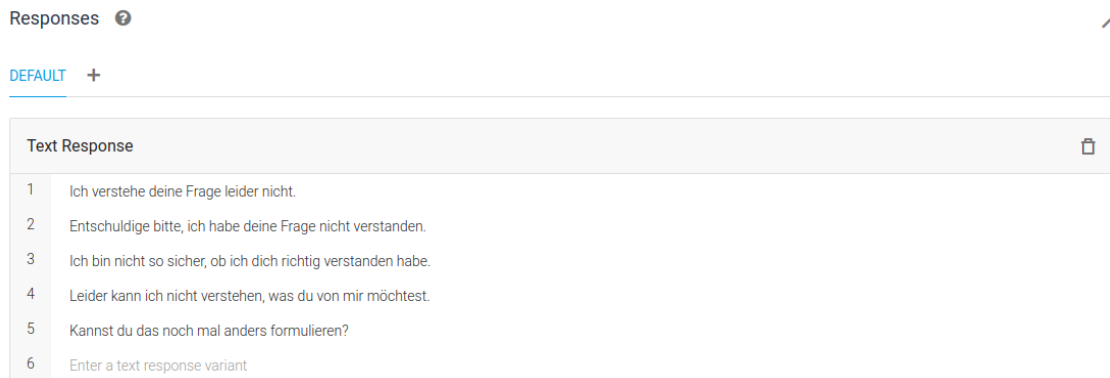


Figure 14: Default Fallback Intent Responses¹⁵

Since the Default Fallback Intent is activated when the agent does not understand the user's input, it does not have any training phrases in them because they are "*negative examples*" and will not match to any user input. This is unique only to Default Fallback Intent. All other intents have training phrases in them.

2. Default Welcome Intent

The Default Welcome Intent is also an existing intent that is automatically created when a virtual agent is created in dialogflow ES. The default welcome intent is matched when the user sends a greeting or welcome message to initiate the conversation with the agent. Default welcome intent is the first intent that an agent recognises from the user's input. For the slackbot application, this intent has been modified to suit the application requirements. *Figure 15* shows the contexts that are assigned to the Default Welcome Intent in SmartSlackBot agent. Contexts are essential to capture the underlying topic of the conversation. It helps connect different user inputs entered at different stages of a conversation. Contexts are explained in detail later in this section. Since Default Welcome Intent is the first intent that the SmartSlackBot agent matches user input to, therefore there is no need for an input context of the conversation as no conversation would have existed before the welcome intent.

¹⁵ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/fallback-response.png>

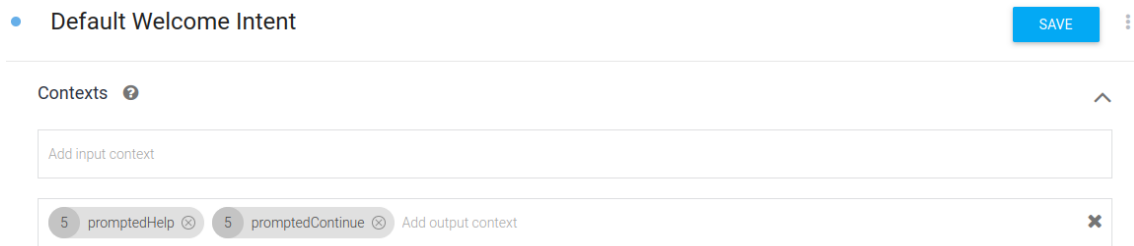


Figure 15: Default Welcome Intent & Contexts¹⁶

Both *promptedHelp* and *promptedContinue* are provided as the output contexts of the Default Welcome Intent because this allows the agent to follow the next user inputs after the greeting or welcome message. Both output contexts are only valid for 5 conversational turns, meaning the SmartSlackBot agent will remember the contexts for 5 total conversational exchanges between the user and the agent. After which the agent clears all contexts and the user must restart the conversation from the beginning. This is done to keep the length of conversation short so that the NLU capability of the ES agent is not expanded over a long conversation. Short conversations help capture contexts more accurately and are sufficient for the use case of the master thesis as the user converses with the dialogflow agent only to initiate or enquire about a few processes. This context length of 5 conversational turns remains the same for all contexts used in all intents of the SmartSlackBot agent. The decision of which context should be assigned to which intent is created in the conversational storylines, a separate topic which is explained in detail in *section 3.4.4*.

Once the user enters a welcome message, the agent will be able to capture the user input and match it with the existing entity type based on the training phrases provided for the welcome intent. For default welcome intent, *Figure 16* lists some of the training phrases created to help the agent match user input with the welcome intent. As seen in the image, different training phrases are created based on the possibility of different user inputs. The LLM model is trained on these training phrases and understands the relationship between the intent and its training phrases. Hence, for example, when a user says “*Hola*” which is a greeting or welcome message that is not provided in training phrases, the agent will still be able to capture the user input, match it with the welcome intent and respond accordingly.

¹⁶ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/welcome-intent.png>

⚠ Template phrases are deprecated and will be ignored in training time. More details here.

When a user says something similar to a training phrase, Dialogflow matches it to the intent. You don't have to create an exhaustive list. Dialogflow will fill out the list with similar expressions. To extract parameter values, use [annotations](#) with available [system](#) or [custom](#) entity types.

” Add user expression
” Hallo Hi
” Sie da
” Heya
” Ich werde nur Hallo sagen
” Hallo ich
” Guten Morgen
” hallo Schatz
” ein guter Tag
” Schöner Tag ist es nicht
” lange nicht gesehen
1 OF 3 →

Figure 16: Default Welcome Intent - Training Phrases¹⁷

After sending the greeting or welcome message, the user receives one of the following responses, shown in *Figure 17*, from the SmartSlackBot agent.

Text Response		
1	Hil Willkommen im Smart Slack Bot. Wenn Sie neu in Smart Slack Bot sind. Bitte geben Sie "Hilfe" ein, um mehr über den Chatbot zu erfahren. Andernfalls geben Sie bitte einen relevanten Satz ein, um Gespräche mit dem Chatbot zu beginnen.	
2	Hallo! Willkommen im Smart Slack Bot. Wenn Sie neu in Smart Slack Bot sind. Bitte geben Sie "Hilfe" ein, um mehr über den Chatbot zu erfahren. Andernfalls geben Sie bitte einen relevanten Satz ein, um Gespräche mit dem Chatbot zu beginnen.	
3	Schönen Tag! Willkommen im Smart Slack Bot. Wenn Sie neu in Smart Slack Bot sind. Bitte geben Sie "Hilfe" ein, um mehr über den Chatbot zu erfahren. Andernfalls geben Sie bitte einen relevanten Satz ein, um Gespräche mit dem Chatbot zu beginnen.	
4	Grüße! Willkommen im Smart Slack Bot. Wenn Sie neu in Smart Slack Bot sind. Bitte geben Sie "Hilfe" ein, um mehr über den Chatbot zu erfahren. Andernfalls geben Sie bitte einen relevanten Satz ein, um Gespräche mit dem Chatbot zu beginnen.	
5	Enter a text response variant	

Figure 17: Default Welcome Intent - Responses¹⁸

¹⁷ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/welcome-training-phrase.png>

¹⁸ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/welcome-responses.png>

3. promptedContinue Intent

The promptedContinue Intent is not a default intent. It was created based on the requirements of the slackbot application. This is the main intent used in the SmartSlackBot agent that detects which workflow the user wants to initiate. promptedContinue intent uses the same context *promptedContinue* as input and output contexts, seen in *Figure 18*. This is created so that the users can continue initiating more than 1 workflow in the same conversation. Users can initiate up to 5 workflows in a conversation as all contexts in the slackbot application are set to a maximum of 5 conversational turns as explained in the previous subsection. That is, if the SmartSlackBot agent manages to recognise all 5 user inputs and matches them all with the respective entity type. After 5 conversational turns, all contexts are deleted and a new conversation needs to be initiated by the user.

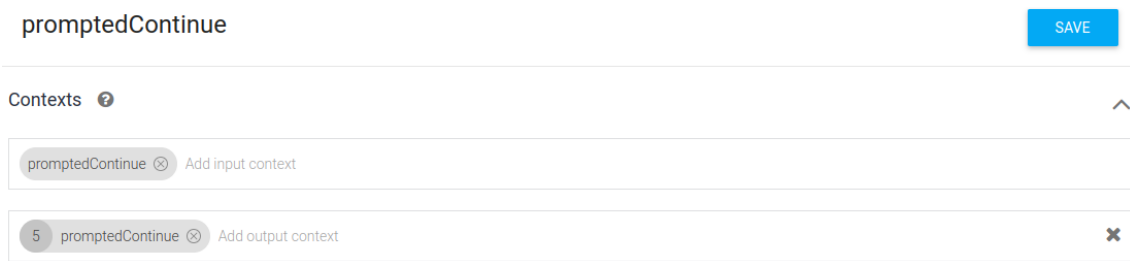


Figure 18: promptedContinue Intent - Contexts¹⁹

Once the user knows which workflow to be initiated, the user can send a message to the SmartSlackBot agent which resembles one of the training phrases, as shown in *Figure 19*, and the agent will respond with an appropriate response, as shown in *Figure 20*. The training phrases, in *Figure 19*, are well detailed and many in number. The screenshot from *Figure 19* is just the first page of the 3 pages of training phrases created for this intent. This is to ensure that the SmartSlackBot agent can understand a variety of user inputs and match the inputs to one of the 3 entities created in the SmartSlackBot agent - sick, vacation and bills. The yellow tags represent the actions being tagged to the training phrases. This was already discussed in this section earlier. Certain words are tagged as action parameters in training phrases so that the chatbot can understand that it is a keyword for the promptedContinue

¹⁹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/continue-context.png>

intent and match the phrase with the correct intent. Once the agent matches the user input with the intent, it sends one of the responses, shown in *Figure 20*, to the users.

Training phrases ⓘ Search training phrases 🔍 ^

⚠️ Template phrases are deprecated and will be ignored in training time. More details [here](#).

When a user says something similar to a training phrase, Dialogflow matches it to the intent. You don't have to create an exhaustive list. Dialogflow will fill out the list with similar expressions. To extract parameter values, use [annotations](#) with available [system](#) or [custom](#) entity types.

” Add user expression

” Ich werde nächsten Monat auf den Malediven **feiern**

” Was mache ich mit meiner **Krankschreibung**?

” Ich möchte meine **Zahlung leisten**

” An wen sende ich meine **Krankschreibung**?

” Ich brauche eine **Pause von der Arbeit**

” Ich mache **Urlaub**

” Ich habe eine **Reise** nächste Woche geplant

” Ich möchte, dass die **Quittung** an mich gesendet wird

” Wo finde ich meine **Quittung**?

” Ich habe eine **Rechnung**

1 OF 3 →

Figure 19: promptedContinue Intent - Training Phrases²⁰

Responses ⓘ ^

DEFAULT +

Text Response 🗑️

1 Vielen Dank, dass Du den \$Workflow workflow ausgewählt hast. Warte bitte, bis der \$Workflow workflow gestartet wurde. Möchtest Du einen weiteren workflow starten?

2 Du hast erfolgreich den \$Workflow workflow initiiert! Bitte warte kurz. Möchtest Du noch einen anderen workflow starten?

3 Enter a text response variant

Figure 20: promptedContinue Intent - Response²¹

In both responses, you can see that there is a variable - “\$Workflow” to which the name of the workflow which is initiated by the agent is assigned. If a sick workflow is initiated then the “\$Workflow” variable is replaced by krank (in German) in the agent’s response. This helps users visually confirm which workflow has been initiated by the SmartSlackBot agent when requesting for workflow initiating from slack.

²⁰ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/continue-trainin.png>

²¹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/continue-response.png>

4. promptedHelp Intent

The *promptedHelp* Intent is also not an existing intent, it was created based on the requirements of the slackbot application. This intent was created to let the first time users know about the SmartSlackBot agent. Users who are new to the slackbot application may not know what to ask the conversational agent. Therefore this intent guides the users on what they can ask the agent and it will give a response to the user explaining its functionalities. *Figure 21* shows the input and output contexts defined for *promptedHelp* intent. *promptedHelp* is given as input context to the *promptedHelp* intent because this structure allows users to ask the agent once more for help if the first response was not clear to the user. At the same time, the user can also continue with the conversation by requesting for a workflow as the output context is *promptedContinue*. The users can, similar to the previous intent, initiate a workflow for 5 conversational turns from the *promptedHelp* intent as shown in the output context of the intent.

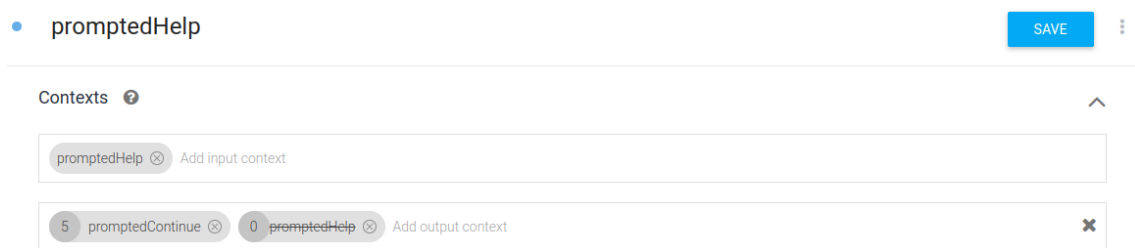


Figure 21: *promptedHelp* Intent - Contexts²²

The training phrases created for this intent, shown in *Figure 22*, include both questions as well as phrases that the users might send as input in order to learn about the SmartSlackBot agent. Once the agent matches the user input to *promptedHelp* intent, the SmartSlackBot agent responds to the user input with the predefined response, shown in *Figure 23*. There is only one response created for *promptedHelp* intent because it covers all the necessary information that the user needs to know to start conversing with the ES agent.

²² <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/help.png>

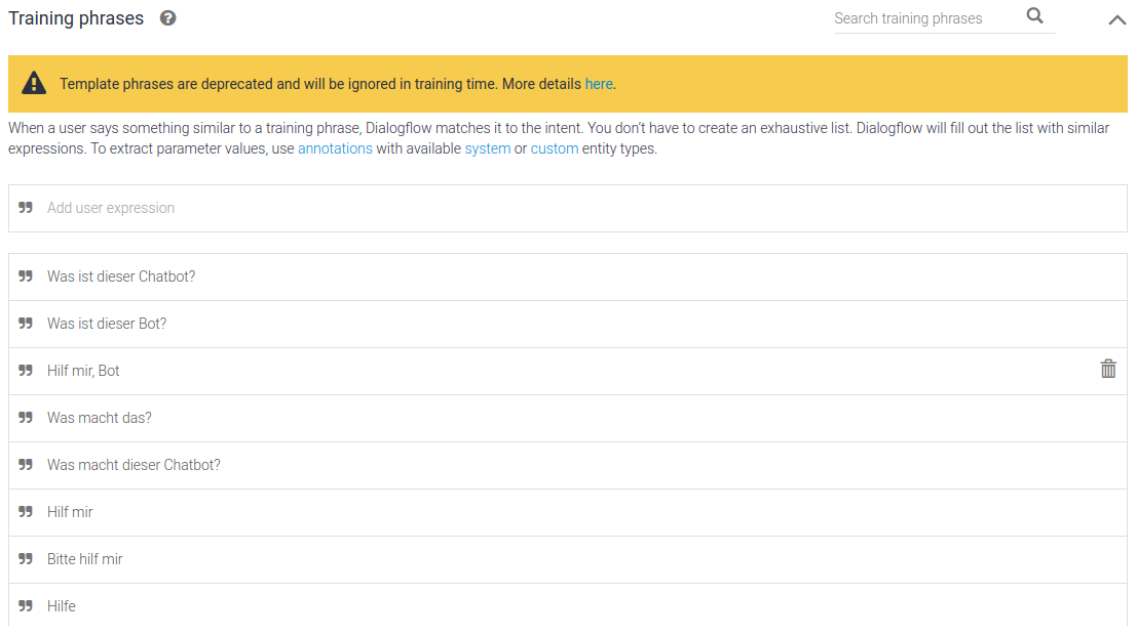


Figure 22: promptedHelp Intent - Training Phrases²³

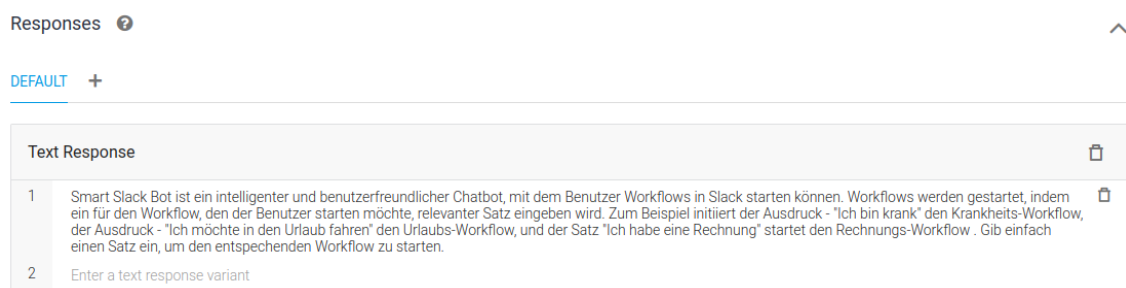


Figure 23: promptedHelp Intent - Response²⁴

5. promptedEndConversation Intent

This is the final intent of the 3 newly created intents for the slackbot application. The purpose of this intent is to end conversation between the user and the SmartSlackBot agent. The *promptedEndConversation* intent has only one input context and no output contexts, as shown in Figure 24. This is because the conversation between the user and the dialogflow agent ends with this intent. Therefore at the end of this intent all contexts are deleted and the conversation is forgotten by the SmartSlackBot agent. The *promptedEndConversation* intent has an extensive list of training phrases created, as shown in Figure 25, in order to allow users to end conversation with the SmartSlackbot agent in different ways.

²³ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/help-train.png>

²⁴ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/help-response.png>

Since the user can use different words and phrases to terminate a conversation in real life. It is necessary to cover other language words that are colloquially used in German in everyday conversations, such as *Ciao* and *Bye*.

The screenshot shows the 'promptedEndConversation' intent configuration in the Dialogflow console. The 'Contexts' section is expanded, showing a search bar and a list of contexts. One context, 'promptedContinue', is listed with a delete icon. A 'SAVE' button is located in the top right corner.

Figure 24: *promptedEndConversation Intent - Contexts*²⁵

The screenshot displays the 'Training phrases' section for the 'promptedEndConversation' intent. A yellow banner at the top states: 'Template phrases are deprecated and will be ignored in training time. More details [here](#).' Below this, a list of training phrases is shown, each with a delete icon. The phrases include 'Ciao', 'tschau', 'tschüss', 'tschüss', 'Bye', 'Stop', 'Breche diese Diskussion bitte ab', 'Breche dieses Gespräch bitte ab', 'Bitte schließe diesen Chat', and 'Bitte beende dieses Gespräch'. A pagination bar at the bottom indicates '1 OF 2'.

Figure 25: *promptedEndConversation Intent - Training phrase*²⁶

Once the SmartSlackBot agent detects and matches the user input with the *promptedEndConversation* intent, it sends the response, shown in Figure 26, to the user and deletes all existing contexts of the current conversation. The response is created in such a way that the user is made aware that the conversational turn has ended with the slackbot application.

²⁵ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/end.png>

²⁶ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/end-training.png>

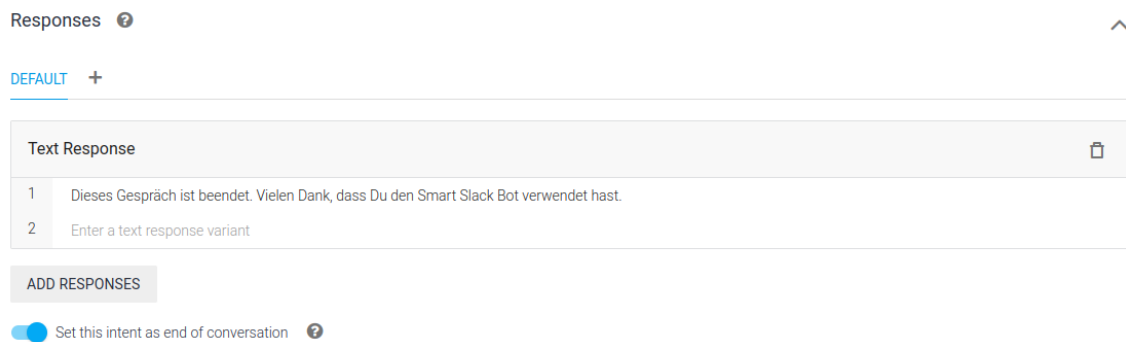


Figure 26: *promptedEndConversation Intent - Responses*²⁷

Entities

Each intent has multiple parameters and one of the important parameter types is called the entity type. Entities dictate how data is exactly extracted from the user input in an ES agent. Dialogflow provides predefined system entities that can match many common types of data. For example, there are system entities for matching dates, times, colours, email addresses, and so on. These are general system entities which do not suit the requirements of the slackbot application. Therefore, custom entities for matching custom data were created in the SmartSlackBot agent. There are 2 entities, shown in *Figure 27*, which were created for the slackbot application - *Workflow* and *endConversation*.

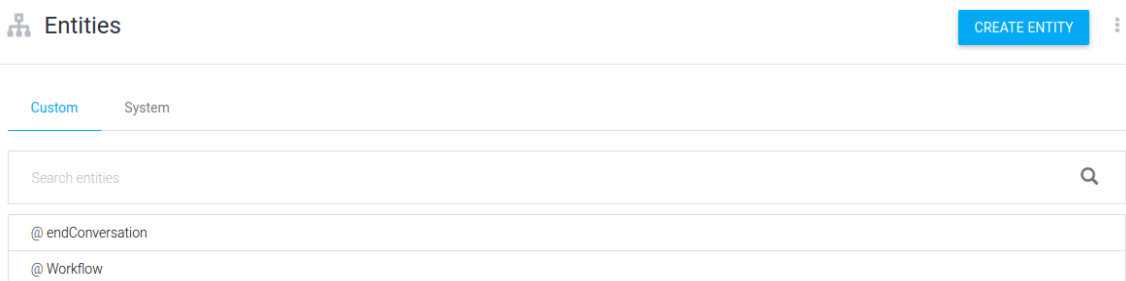


Figure 27: *Entities*²⁸

Workflow entity, shown in *Figure 28*, is used to capture the correct workflow that the user wants to initiate from the slack channel. The entity has reference values (on the left) and a number of synonyms (on the right) for those reference values for all 3 use cases of the slackbot application - Krank (sick), Urlaub (vacation), and Rechnung (invoices).

²⁷ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/end-response.png>

²⁸ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/entities.png>

The reference values are the exact words that the users may use in a message and synonyms are possible alternatives to the reference values that the user might provide in the input.

Workflow

SAVE

☒ Define synonyms ☐ Regex entity ☐ Allow automated expansion ☒ Fuzzy matching

Krank	Krank, Fieber, Kalt, Husten, Grippe, Kopfschmerzen, Nicht gut fühlen, Sich nicht gut fühlen, nicht wohl
Urlaub	Urlaub, Pause, Ferien, Ausserhaus, Reisen, Party, Reise
Rechnung	Rechnung, Zahlung, Kassenbon, Bill

Figure 28: Workflow Entity²⁹

The NLP engine used by dialogflow uses these reference values and synonyms to extract data from the user input in order to match with the correct intent. Selecting the checkboxes in Figure 28 for “Define synonyms” and “Fuzzy matching” are optional. For the slackbot application, both options are selected because the users can use either the exact word or its synonyms to request for the workflow initiation for the 3 use cases. If there were buttons provided to the users to choose from then fuzzy matching or synonyms are not required as users will select only from the given options but since for slackbot application, we provide a complete chat experience the user have the freedom to type the phrases they like and it is the responsibility of the dialogflow agent to be smart enough to capture the intent from those phrases using the workflow entity type. The *Workflow* entity always matches the user input to the *promptedContinue* intent which is the main intent of the SmartSlackBot agent.

Similar to the *Workflow* entity, the *endConversation* entity is used to extract data from the user input and match it to the existing intent but the functionality of the *endConversation* entity varies from the *Workflow* entity. The *endConversation* entity is used to capture the end conversation request from the user. This entity, as shown in Figure 29, has only one reference value - *Ende*, and a list of synonyms to make the user input match with the existing intents to end or terminate the current conversation. When the user sends a message requesting to terminate the conversation, this entity is matched to the user input and the *endConversation* Intent is triggered.

²⁹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/entityiv.png>

endConversation

SAVE

☒ Define synonyms ☐ Regexp entity ☐ Allow automated expansion ☒ Fuzzy matching

Ende	Ende, Ausgang, Beenden, Stop, Abschließen, einstellen, abbrechen, Fertig, Das ist es, Das wird es sein, Das passt, Ok
------	---

Figure 29: *endConversation Entity*³⁰

Contexts

Dialogflow contexts are similar to natural language context. For example, in a general use case, If the user says "they are orange", the agent needs context in order to understand what "they" refers to in the user input. This will help the dialogflow agent to correctly match an intent with the context. Using contexts, the flow of a conversation can be controlled. Contexts can be configured for an intent by setting input and output contexts, which are identified by string names. When an intent is matched, any configured output contexts for that intent become active. While any contexts are active, dialogflow is more likely to match intents that are configured with input contexts that correspond to the currently active contexts. This way the conversation structure can be linked with previous, current and next user inputs. For the SmartSlackBot agent, there are 3 contexts that are most commonly used - *promptedContinue*, *promptedHelp*, and *endConversation*. These 3 contexts control the conversation and guide users to the next intent as already explained earlier in this section.

Fulfillments

By default, the dialogflow agent responds to a matched intent with a static response. By using one of the integration options provided by Google dialogflow, a more dynamic response can be provided to the user. When fulfillment is enabled for an intent, as shown in *Figure 30*, the agent responds to that intent by calling a service that is integrated with the intent. Each intent has a setting to enable or disable fulfillment. If an intent requires some action by an external system or a dynamic response then fulfillment should be enabled for that intent. If an intent without an enabled fulfillment is matched, dialogflow uses the static response that is defined for the intent to respond to the user.

³⁰ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/end-entitivy.png>

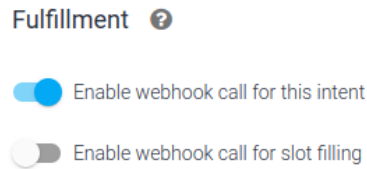


Figure 30: Fulfillment Enabled³¹

Fulfillment is usually enabled through webhook integrations. For the slackbot application, the fulfillment is enabled for only one intent - *promptedContinue* intent. This is because the information about the matched intent and the entity type to which it the intent is matched, is only required by AWS lambda when the agent is expected to initiate the workflow. The intent responsible for initiating workflow in SmartSlackBot agent is the *promptedContinue* intent. Only for the *promptedContinue* intent, the matched entity type is sent to the AWS lambda from the SmartSlackBot agent as a parameter through fulfillment. User inputs that are matched to other intents without fulfillments, only receive the static responses from the SmartSlackBot agent. The intent type or entity type need not be sent to AWS for these intents. Therefore fulfillment can be disabled for intents that do not have to send dynamic responses. This is good practice because most of the conversational data is stored within the dialogflow environment, only essential conversational data is transferred from GCP to AWS. This helps in preventing breaches or loss of data when data is moved between environments. The sections 2.2.1 and 2.2.2 are the important technical configurations set up in slack and dialogflow to enable a smooth conversational experience for the users from the slack application.

2.2.3. Processes

Apart from the collaboration tools and the company wiki used by the company, it is essential to explain in brief how the organisational process for the below 5 use cases happen at the company. This is because the slackbot application's job is to automate the below 5 use cases as part of the master thesis:

1. Applying sick leaves (Krankmeldung)

³¹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-two/dialogflow/fulfillment.png>

2. Submitting vacation requests (Urlaub Erfassen)
3. Submitting invoices (Rechnungen)
4. Initiating project ideas (Meine Idee channel)
5. Requesting project details (Projekte webpage)

2.2.3.1. 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 where employees can manually select the workflow and initiate it. The workflow explains the consecutive steps necessary for the users to complete the sick leave application within the company. The challenge for the company in this process is that new employees or employees who have not applied sick leaves before have no idea that this workflow exists. Some employees find it difficult to find this workflow in the company's slack workspace. Even experienced employees might forget how they initiated the sick workflow a few weeks/months before, since people don't apply for sick leaves quite often. Hence it becomes both time consuming and tedious for the employees to search and find where and how to initiate the *krankmeldung* workflow among the other important things that the employees must focus on within the company. There is also a confluence page called *Krankmeldung* which explains the process of how the slack 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 challenge that the slackbot application is trying to solve as part of the master thesis. Slackbot, when activated, provides the user with 2 buttons - one for redirecting the user to the confluence page and the other button to initiate the workflow directly from the chat window so that the user does not have to search for the *krankmeldung* workflow every time they want to apply for a sick leave.

2.2.3.2. Submitting Vacation Requests

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 slackbot for this use case is similar to applying sick leaves where upon activation, will provide 2 buttons - one to redirect the users on the slack channel to the confluence page and the other to initiate *Urlaubsmeldung* workflow directly.

2.2.3.3. 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 page called *Rechnungen* that explains how much money can be reimbursed without further approvals and how many approvals are needed for purchases exceeding a certain price. The role of the slackbot application, in this use case, will be to provide 2 buttons similar to the previous two 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.

2.2.3.4. Initiating Project Ideas

New project ideas are essential for companies' growth, 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 slackbot for 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 for new project ideas are not lost in the everyday conversations of the general slack channel.

2.2.3.5. 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 ongoing project they need, but the same problem of finding the confluence page among multiple confluence pages exists in this use case too. At the same time searching for the project details is a time consuming and tedious task even inside the *Projekte* confluence page. The role of *slackbot* is to help automate this task by retrieving 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 application 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 covered all the non technical aspects of the master thesis. Chapter two has given a brief explanation of the technical aspects of both PoC and the slackbot application along with an overview of the existing processes in the company. Chapter three and four will cover the step-by-step development plan and architecture of the slackbot application respectively.

CHAPTER

THREE

DEVELOPMENT PLAN

A general overview of the slackbot application along with the technical configurations of its individual components was provided in the first two chapters of the thesis document. This was to give an introduction and provide the background information necessary for the slackbot application. The next step was to start building the slackbot application.

3.1. Tech Stack for Slackbot

The list of finalised technical configurations that were confirmed from the earlier chapters of for building the slackbot application are:

1. **Platforms** - Slack, AWS and Google dialogflow ES
2. **Architecture** - Microservices Architecture (MSA)
3. **Methodology** - ChatOps and SDLC
4. **Programming Languages** - Python, Typescript, HTML, Javascript, and CSS

3.2. Software Development Life Cycle (SDLC)

The type of SDLC model used for developing the slackbot application is the *Iterative SDLC model*. Iterative models provide faster results, require less up-front information, and offer greater flexibility [38]. This model was selected because to start developing the end product, this model does not require the full specifications or requirements of the product at the very beginning. 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 iteration of the model [39].

The iterative model of SDLC can usually include 6 phases [40]:

1. Requirements Gathering & Analysis
2. Design
3. Implementation
4. Testing
5. Deployment
6. Review and Maintenance

Figure 31 is a visual representation of how the iterative SDLC model works in building production ready software applications.

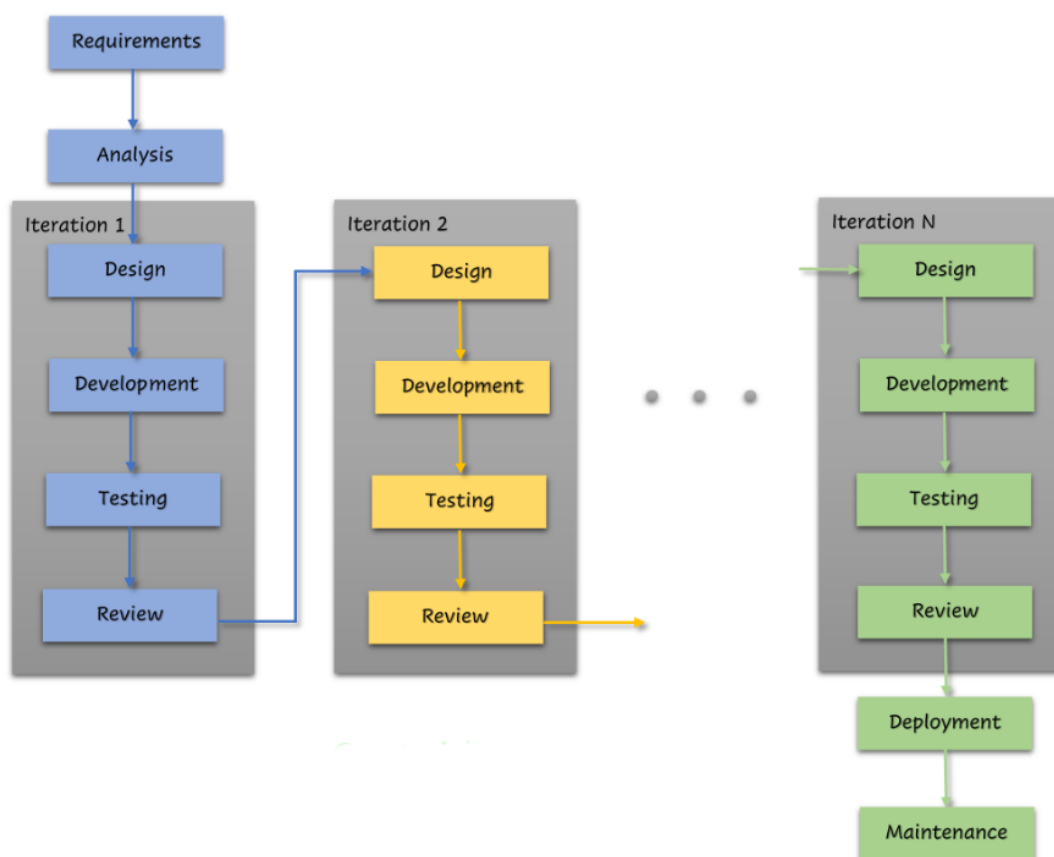


Figure 31: Iterative SDLC Model³²

³² <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/iterative-sdlc.png>

Requirements Gathering & Analysis is the first stage of the iterative SDLC model, followed by multiple iterations of software design, 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 final phase of monitoring & maintenance begins. This was also the sequence of steps that was followed in developing the slackbot application. One of the first and important steps in iterative SDLC methodology is System Requirements Analysis. This is explained in the next section.

3.3. System Requirements Analysis

Requirements are necessary attributes defined for a software prior to the efforts of developing its design [41][pg.7]. System Requirements Analysis (SRA) is a structured and organised methodology of 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 [41][pg.7]. SRA acts as a transformation between the customer's needs and the software design created for development. It is the first phase of the iterative SDLC model and is performed at the very beginning of a project. SRA is performed in two steps: Requirements Gathering and Requirements Analysis.

3.3.1. Requirements Gathering

Requirements Gathering is usually done in collaboration with the stakeholders in order to collect, record and categorise the requirements of the project. This process consists of 5 steps [42]:

Step-1: Identifying Key Stakeholders and End-users

This is the first step of the requirements gathering or analysis phase. In this step, the stakeholders are identified and defined along with who the end users will be for the project. For the slackbot application, both the stakeholders and the customers are from the same company because it is an internal application built within the company to improve the accessibility to knowledge base and also the team collaboration among employees, who are

the end users. Since the company sponsors the project and defines the scope of the application whose end users are the employees of the same company, both the stakeholders and the customers for the slackbot application are the company itself. The goal of the project is to cover the scopes defined by the stakeholders and should satisfy the needs of the end users, the employees of the company. In this step the stakeholders and the end users for the slackbot application were identified.

Step-2: Capture Requirements

In the first step, the stakeholders and end users were identified. In step two, the requirements from the stakeholders and the end users were captured. Both their inputs were essential for defining the scope of the project before starting to build the application. This can be done in multiple ways. First way is by holding one-on-one interviews with the stakeholders and end users of the project. This helps in gathering the requirements directly from the stakeholder and the customers. Another way is to request use cases from the end users as this gives a walkthrough of the application end-to-end from the eyes of the end users. Third way 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 the slackbot application, both one-on-one interviews with the stakeholders and customers from the company were held along with building a prototype of the slack application as PoC. These two methods helped capture the requirements accurately that were necessary for building the slackbot application.

Step-3: Categorise Requirements

The *capture requirements* step (step-2) includes capturing both user requirements and system requirements. These two requirement documentations combine together in officially registering the agreed requirements for the project between the developer and the stakeholders. Difference between the User Requirements Document (URD) and the System Requirements Document (SRD) is that the former is a non-technical document and the latter 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. User requirements are collected in a document called Functional Requirements Document (FRD). This document is written by the developer and is approved by the stakeholder. The FRD also has the goal of the project clearly defined and approved by both the stakeholders and the developer. This is done so that all the work dedicated to building the application can be directed towards the goal agreed upon in the FRD. Any modifications suggested by the stakeholders gets added to the later versions of the applications 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. 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 were not created for the slackbot application. This is because the master thesis was an internal project hence there were no need for official documentation and approvals between the developer and the stakeholders but this step is crucial in regular software development as industry practice.

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 [42]. *Figure 32* shows the Gantt chart created for the slackbot application to track the progress of the project.

DEV Online Dienste GmbH

Rishi Srinivasan Kanaka Sabapathy

Project start date: 07.11.2022

Scrolling increment: 0

Legend:

On track
Low risk
Med risk
High risk
Unassigned

							November							December							January							February							March																				
							7	10	13	16	19	22	25	28	1	4	7	10	13	16	19	22	25	28	31	3	6	9	12	15	18	21	24	27	30	2	5	8	11	14	17	20	23	26	29	1	4	7	10	13	16	19	22	25	28
Milestone description							Category	Assigned to		Progress	Start	Days																																											
Create Slack chatbot locally																																																							
Create Slack App in company workspace							Goal	Rishi		100%	07.11.2022	2																																											
Set up local environment							Milestone	Rishi		100%	09.11.2022	4																																											
Develop Slackbot locally							Goal	Rishi		100%	13.11.2022	13																																											
Test Slackbot locally							Goal	Rishi		100%	26.11.2022	2																																											
Deploy chatbot to AWS																																																							
Set up AWS environment							Milestone	Rishi		100%	28.11.2022	2																																											
Set up individual AWS services							Milestone	Rishi		100%	30.11.2022	3																																											
Connect all services based on planned Architecture							Goal	Rishi		100%	03.12.2022	5																																											
Export local slackbot to AWS							Milestone	Rishi		100%	08.12.2022	4																																											
Deploy Slackbot in AWS							Goal	Rishi		100%	12.12.2022	15																																											
Test Slackbot in AWS							Goal	Rishi		100%	27.12.2022	4																																											
Design Smart Slackbot UI							Milestone	Rishi		100%	31.12.2022	5																																											
Create Chatbot in GCP																																																							
Create Chatbot Agent							Milestone	Rishi		100%	05.01.2023	3																																											
Create Intents and Entities							Milestone	Rishi		100%	08.01.2023	3																																											
Train chatbot with synonyms							Milestone	Rishi		100%	11.01.2023	2																																											
Integrate chatbot to Slack for testing							Goal	Rishi		100%	13.01.2023	10																																											
Integrate chatbot to AWS							Med Risk	Rishi		70%	23.01.2023	5																																											
Thesis Writing																																																							
Write the thesis for submission							Goal	Rishi		100%	28.01.2023	17																																											
Administrator Setting																																																							
Create webpage for Admin								Rishi			14.02.2023	5																																											
Deploy the webpage in S3								Rishi			19.02.2023	5																																											
End-to-End testing																																																							
Test Smart Slackbot E2E & Deploy in PROD								Rishi			24.02.2023	15																																											

The Gantt charts are used in project management to track the progress of all tasks in the project [43]. On the left side of the chart, in *Figure 32*, there are some information such as:

1. Milestone description - This column explains each task involved in building the slackbot application in a brief manner.
2. Category - This is used to categorise each task into a goal (or) milestone (or) risk based on the timeline and priority of the task.
3. Assigned to - This column contains the team members' names who are assigned to the task but since the master thesis is an individual, all the tasks were assigned to one person.
4. Progress - This column is used to track the status or progress of each task for the slackbot application. Completed tasks show 100% in the Gantt chart.
5. Start - This is used to mark the start date of the task.
6. Days - This denotes the approximate number of days required to finish the task.

In the category column, 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 [44]. On the right side of the chart is the visual representation of the tasks' scheduled timelines along with the current status - *completed* or *at risk*. The red diamonds are for goals achieved and yellow flags are for the milestones completed. They signify that the task is completed on the date specified at the top along with the month. Blue and purple bars denote that a task is at risk and is not completed on time. Various shades of blue differentiate between low and medium risks. Purple is for high risk tasks. The milestone "*Integrating chatbot to AWS*" was at risk at the time of pausing the development of the application to start writing the master thesis but this milestone has been completed and the Gantt chart will be updated as soon as the project resumes development. Grey bars signify that the task is unassigned and is planned for the future sprint cycles. Other project related information such as project start date, the current status of the project, the future plan, etc can be inferred from the Gantt chart.

Step-5: Sign off

Once a final decision is made on the requirements, a confirmation from the key stakeholders is ensured regarding the accepted requirements. This is done to make sure that there are no changes or uncontrolled growth in the scope of the project. This was done in the slackbot application too, in deciding the scope and timeline of the MVP and getting the approval of the stakeholders for the MVP which was the first production ready version of the slackbot application with the additional features being developed in the future, as shown in the Gantt chart diagram in *Figure 32*.

3.3.2. 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 the slackbot application, the Business Process Model and Notation (BPMN) technique was used 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. *Figure 33* shows the BPMN flowchart that was created for modelling the processes of the *slackbot* application. In the BPMN flowchart, it is easy to understand that there are 3 different environments - slack, AWS and Google dialogflow. The business process for the slackbot application transcends between these 3 environments. The start event and end event denoted in blue and green colour respectively 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 component which is denoted in a red coloured diamond is 'Yes' and the flow of information becomes a loop when the *Keyword found* decision is 'No'. The processes in the BPMN diagram for the slackbot application are denoted in yellow rectangles. These are the activities performed by the slack application at different stages of the BPMN.

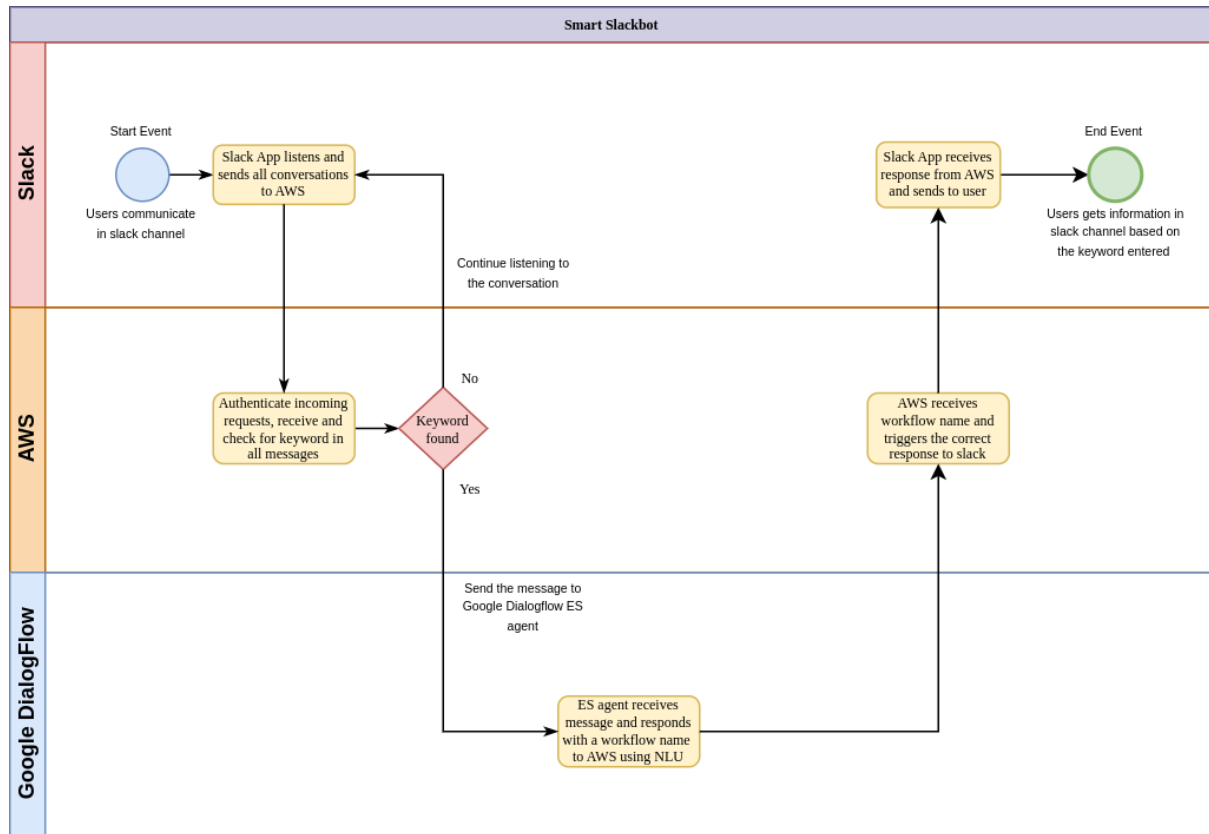


Figure 33: Business Process Model and Notation Diagram³⁴

After gathering and analysing the requirements necessary for building the application, the next step is 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 ES agent. The response users receive from the ES agent is also largely influenced by the conversational user flow design. In the next section, it is explained in detail on how the user flows are designed in dialogflow for the slackbot application.

3.4. Conversational Design

Conversations are an interactive way of communication or social interaction between two or more human beings [45]. Conversations, for centuries, have been structural in building relationships and exchanging ideas among human beings.

³⁴ [https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/bpmn-new.drawio%20\(2\).png](https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/bpmn-new.drawio%20(2).png)

3.4.1. History of Conversational AI

Since the evolution of Artificial Intelligence (AI), 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*" [46], in which he proposed an experiment to evaluate the intelligence of AI 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 important tests used in the past to evaluate an AI system's ability to think. Alan Turing proposed that if a machine can have conversation like humans, then they can think like humans but AI systems of today are not the same as they were 70 years ago. Hence many AI systems have managed to pass the Turing test without having actual human-like intelligence [47]. Since the development of Large Language Models (LLMs), the conversational AI has transformed from rudimentary chatbots that gave pre-fed answers with no intelligence in Alan Turing's 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.

3.4.2. Designing Conversational AI

Conversational AI is the study of techniques for software agents that can engage in natural conversational interactions with humans [48]. Conversational AI 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, the conversations on how humans and chatbots can interact with each other can be designed and structured. This is called conversational storylines [49]. Designing conversational storylines is the first step to building a chatbot as this determines how the chatbot will respond to each interaction with the humans. Since the chatbot is a key player in the slackbot application - dialogflow ES agent, it is explained in this section on how the conversational storylines were designed for building a responsive chat experience with the slackbot application.

3.4.3. Conversational Storylines

In making a conversational storyline, the first step is to determine the overall theme of the conversation [49]. The theme for the conversation that the users will have with the chatbot in the slackbot application, was to determine the type of workflow that the users want to initiate within the company. Therefore the first step was the goal of the chatbot which was to have a conversation with the users and predict which workflow the users wanted to initiate. The overall theme of the conversation was determined as the first step of the conversational storylines. Next step was to list out each scene and characters to illustrate the scene [49]. This helps in clearly defining the roles of chatbot and the users in each scenario where the chatbot interacts with the users. There are 3 scenarios where the ES agent is used in the slackbot application. These 3 scenarios formed the scenes. The character was the role which the users play in each of these scenes for the chatbot to understand what the user wants in each scene.

Below is the table describing how the scene and characters are set for slackbot:

Scene Number	Scene	Character
1	Sick Leave	Sick Employee
2	Vacation Request	Employee going on vacation
3	Submit Invoices	Employee requesting information

Table 1: Chatbot Storylines

3.4.4. Branching Storyline Route

After creating a conversational storyline, the next step was to sketch the storyline branching route [49]. The goal of branching storylines was to anticipate what the user wants and navigate the user through the available options present in the chatbot which are closest to the user needs [49]. This was done through giving *prompts* to the users about the available options so that the users can choose the closest option that they need. The branching storyline diagram created for the slackbot application is represented in *Figure 34*. This

diagram illustrates how the conversational user flows happen between the dialogflow ES agent and the user.

In *Figure 34*, the conversations are colour coded for better understanding of the readers. The user entries are denoted in pink while the chatbot's responses are denoted in violet. The yellow boxes represent intents, contexts and processes. These terminologies - intents, contexts and processes were already explained in detail in chapter 2, *section 2.2.2* of the thesis document. In this section, only the structure of the conversation between the users and the agent is discussed. In the branching diagram, *Figure 34*, there are 4 steps involved as part of the conversation. These steps are highlighted in Teal blue:

Step-1: Welcome or Greeting step

Step-2: Help step or Action step

Step-3: Continue step or End step

Step-4: Continue step or End step

Step-1: Welcome step or Greeting step

All conversations with the ES agent, *SmartSlackBot*, begins with a greeting or welcome from the user. The ES agent is dormant unless the user initiates a conversation with a greeting or a welcome message. The greeting message can be anything like - Hi, hello, hey, Good Morning, etc. Since the dialogflow uses NLU in the background to understand conversations, the different types of welcome messages are recognised by using the *Default Welcome* intent set in the ES agent. The *welcome* Intent determines what types of welcome or greeting messages the agent can comprehend. They also determine what types of responses the agent can send to the users. After receiving the welcome message from the user, the agent responds with an introductory message. Since the agent is developed both in English and German, the examples of conversational user flows are displayed in both languages. The user flows in english are available in the branching diagram, in *Figure 34*, whereas the user flows in german are provided as screenshots directly from the dialogflow console, in *Figure 35*. This is followed for all the 5 steps in the branching storylines section.

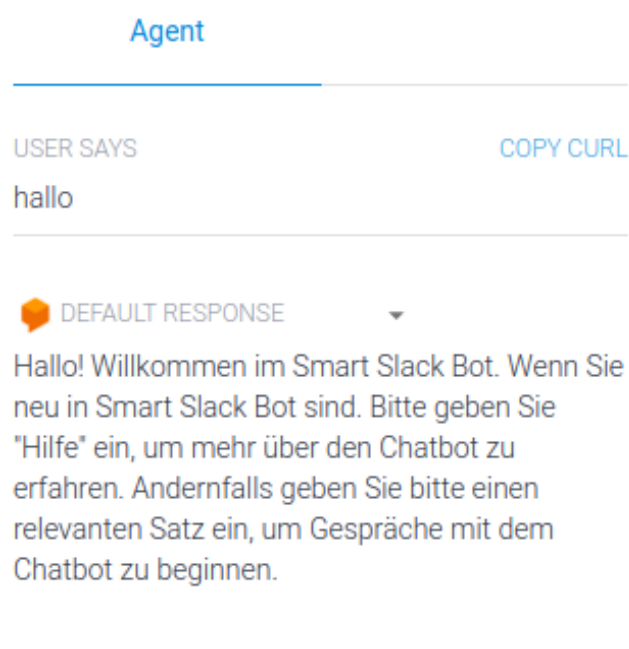


Figure 35: Dialogflow Demo - Welcome Intent³⁶

³⁶

<https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2011-58-29.png>

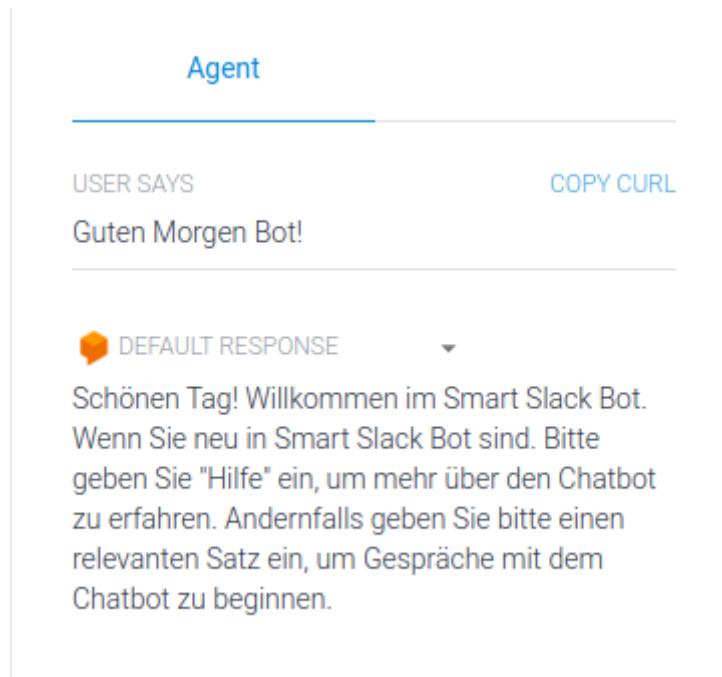


Figure 36: Dialogflow Demo - Welcome Intent for different input³⁷

Different user inputs were also tested to show the variation in response from the chatbot. In *Figure 35* and *Figure 36*, the chatbot responds differently to different welcome messages from the users. This is because both the responses are fed into the welcome intent, as explained in *section 2.2.2* and the chatbot is trained to learn from these phrases and decides the most appropriate response to the user's input based on its advanced NLU capabilities. If the user slightly modifies his input, the chatbot will still be able to understand the user input and detect the correct intent. Therefore giving the correct response to the user.

Step-2: Help step or Action step

The second step in the conversational user flow comes with 2 options: Help step or Action step. First option is when the user is new to the chatbot and has no knowledge of the chatbot. In this case, the user can ask the chatbot to help them before asking it to perform an action. When a user asks for help, the chatbot explains its functionality to the users as seen in the branching diagram, in *Figure 34*. The chatbot explains to the user what it is created for, what it can do and how users can interact with it. This part of the conversation is called the "***promptedHelp***" intent. The responses are registered in the *promptedHelp* intent

³⁷ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2012-32-20.png>

and similar to the *welcome* intent, the chatbot chooses the most appropriate response and sends it to the user based on the user input. Figure 37 is an example of the same *promptedHelp* intent response in German.

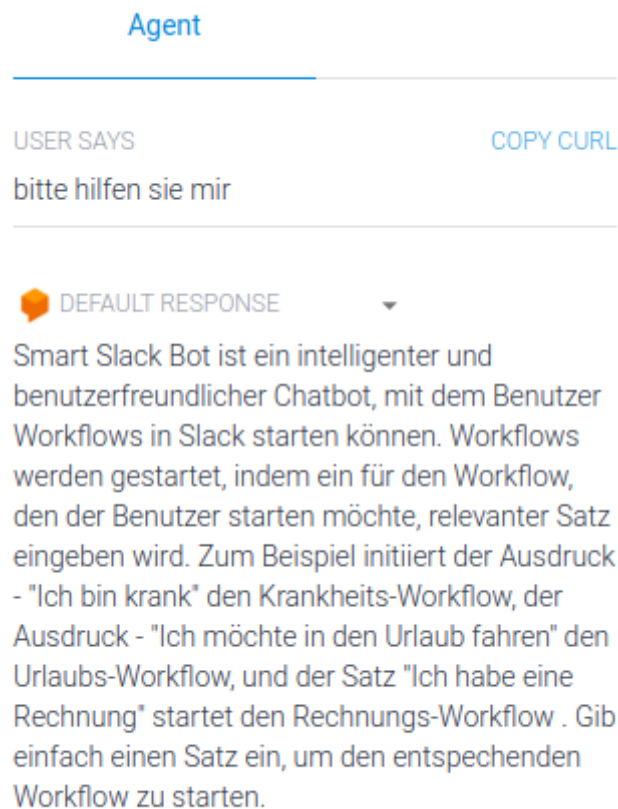


Figure 37: Dialogflow Demo - Help Intent³⁸

Second option of step-2 is for the users to directly ask the chatbot to perform an action. This is called the Action step. This information is stored in the *promptedContinue* intent. If the user is already familiar with the chatbot, then they don't have to necessarily 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 were created for the agent earlier, in *Table 1*, the user can ask the chatbot to perform 3 different actions:

1. initiate sick workflow
2. initiate vacation workflow

³⁸ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2012-51-59.png>

3. Initiate invoice workflow.

These 3 workflows are the scenes created, in *Table 1*, for the current version of the chatbot. 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 in *Figure 38* in German.

Agent

USER SAYS

COPY CURL

Ich habe Fieber

DEFAULT RESPONSE

Vielen Dank, dass Du den Krank workflow ausgewählt hast. Warte bitte, bis der Krank workflow gestartet wurde. Möchtest Du einen weiteren workflow starten?

CONTEXTS

RESET CONTEXTS

promptedcontinue

promptedhelp

INTENT

promptedContinue

ACTION

input.workflow

PARAMETER

VALUE

Workflow

["Krank"]

SENTIMENT

Query Score: 0.1

Figure 38: Dialogflow Demo - promptedContinue Intent for Krank³⁹

³⁹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2013-13-42.png>

In *Figure 38*, the chatbot is smart enough to understand the different ways in which the user can express that they want to initiate a sick workflow. In *Figure 38*, the user does not mention the word “*Krank*” but the chatbot is smart enough to understand that “*Fieber*” refers to the sick workflow and thus initiates the sick workflow for the user. It is also possible to see the additional parameters, as discussed in *section 2.2.2*, being captured by the chatbot such as: *Contexts*, *Intent*, *Parameter* and *Sentiment*. In contexts, the chatbot shows the next possible contexts that the user can possibly choose from. The user can either continue the conversation or end conversation from *promptedContinue* intent.

If the user continues the conversation then the next context is *promptedContinue* or *promptedHelp* context if the user asks for help to the chatbot. Intent denotes the context of the current user input. In *Figure 38*, the chatbot is able to recognise that the user input belongs to *promptedContinue* intent. If the user had asked for help then the chatbot would recognise it as *promptedHelp* intent. Parameter is the entity set by the developer to recognise the action which the chatbot should perform. Entities are variables that dictate how data is extracted from the user input. They identify what needs to be done from a user input. In *Figure 38*, the entity is correctly identified as *Krank* (action to be performed by the chatbot) from the user input of “Ich habe Fieber” and hence sick workflow will be initiated.

The last parameter is the *Sentiment* parameter which is the query score between -1 and +1. The query score defines the sentiment of the user query. The NLP engine performs a sentiment analysis on each user input and displays the sentiment score of -1 to +1 for the *sentiment* parameter. The *sentiment* parameter varies from Negative (0 to -1), Neutral (0) and Positive (0 to +1) sentiments. In this conversation, the NLP engine has predicted that the user input is a neutral sentiment with a sentiment score of 0.1 but the sentiment analysis is outside the scope of the thesis and the query score parameter is not used anywhere in the master thesis.

All this additional information is captured for every user input. The chatbot will only confirm that the workflow has been successfully initiated when it recognises what the user wants, that is, if the entity is captured from the user input. If the chatbot could not recognise the user input, it will ask the user to repeat again for up to 5 times to capture the entity. At the

end of the 5th attempt, the conversation is ended by default, all contexts of the conversation are deleted and the user has to initiate a new conversation with the chatbot.


Step-3: Continue step or End step

Agent

USER SAYS

COPY CURL

Ich reise nächste Woche nach Mallorca

 DEFAULT RESPONSE ▼

Vielen Dank, dass Du den Urlaub workflow ausgewählt hast. Warte bitte, bis der Urlaub workflow gestartet wurde. Möchtest Du einen weiteren workflow starten?

CONTEXTS

RESET CONTEXTS

promptedcontinue

promptedhelp

INTENT

promptedContinue

ACTION

input.workflow

PARAMETER	VALUE
Workflow	["Urlaub"]

SENTIMENT

Query Score: 0.3

Figure 39: Dialogflow Demo - promptedContinue Intent for Urlaub⁴⁰

⁴⁰

<https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2013-48-45.png>

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 user's needs are satisfied. Step-3 covers this process. If the user wants to continue the conversation and initiate another workflow then the user can proceed by directly asking the chatbot to perform the next action. This is called the continue step. This is similar to the second option of step 2 in the branching diagram. An example of this step is shown in the branching diagram in English, with the intent detected as *promptedContinue* intent. In *Figure 39*, the same scenario is tested in German in the dialogflow console.

In *Figure 39*, you can see that the word “*urlaub*” was never used but the chatbot was able to correctly identify the right workflow that the user intended to initiate using the verb “*Reisen*” meaning travel. The entity is also detected correctly as [*“Urlaub”*]. The user can again continue the conversation for a maximum of 5 times from the start of the conversation with the chatbot, similar to step-2, after which all the contexts are deleted and the chatbot forgets the existing conversation. This is an example of the continue step.

Second option in Step-3 is called End step. An English example of this step is shown in the branching diagram as *promptedEndConversation* Intent and the German example is shown in *Figure 40*. If the user wants to end the conversation with the chatbot, they can ask the chatbot to directly end this conversation. For ending the conversation, the chatbot uses the second entity of the 2 predefined entities called *endconversation* whereas the continue step (first option of step-3) uses the *workflow* entity to initiate the vacation workflow. Two entities are created to differentiate between users asking to initiate a workflow (workflow entity) and users choosing to end the conversation (endconversation entity).


In *Figure 40*, the user says “*Nein Keine workflows mehr*” and the chatbot understands that the user wants to end the conversation. In parameter, the chatbot has recognised that the user wants to end conversation with the words mentioned by the user “*Keine workflows*” with the reference value of the *endConversation* entity “*Ende*”. This was also discussed in detail in *section 2.2.2*. The intent used to recognise the *endconversation* entity is the *promptedEndConversation* intent which is also displayed by the dialogflow agent in *Figure 40*. The chatbot then deletes all the contexts and ends the current session with the user.

Agent

USER SAYS

COPY CURL

Nein Keine workflows mehr

 DEFAULT RESPONSE

Dieses Gespräch ist beendet. Vielen Dank, dass Du den Smart Slack Bot verwendet hast.

INTENT

promptedEndConversation

ACTION

input.endConversation

PARAMETER	VALUE
endconversation	["Ende", "Keine workflows"]

SENTIMENT

Query Score: -0.5

Figure 40: Dialogflow Demo - promptedEndConversation Intent⁴¹

Step-4: Continue step or End step

This step has the same name as step-3 because they are both the same steps but they happen in different places of the conversation with the chatbot. When the user is new to the chatbot as explained in step-2, first option, after the user learns about the chatbot through the *promptedHelp* intent, the user can directly initiate a workflow from this point. This is denoted as step 4 in the branching diagram.

⁴¹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-05%2013-56-12.png>

Step 4, in the branching diagram, 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 of the branching diagram, including both the continue step and end step in step 4.

This is the conversational user flow created for the slackbot application using the dialogflow ES agent in Google Cloud Platform. In the next section, the step-by-step plan created to build the slackbot application is discussed.

3.5. Step-by-Step Plan

The slackbot application is a complete scalable application that is deployed in the production environment. Therefore, careful planning had to be involved in building the application so that it 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, individual architectures of all 3 major environments - slack, AWS and Google dialogflow had to be studied first. Therefore the first step of the development plan involved deep analysis and study of individual environments, their architecture and the services that were offered. From the PoC, there was prior knowledge about how the slack, aws and dialogflow could be integrated together but for the slackbot application, the requirements were different and hence the way these 3 environments were integrated was also different. This is because the basic requirement of how a conversation should happen was different between the PoC and the master thesis slackbot application.

In the architecture diagram of the PoC (*Figure 3*), the slack app was directly connected to dialogflow ES agent through a direct integration. This was possible because in the PoC, 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 the slackbot application, the slack app should always be active and listen to all conversations happening in the slack channel. Therefore a direct integration to dialogflow agent was tested and it did not work. A lot of research was involved to find the suitable way in which the custom slack app could listen to all conversations and become active upon listening to a particular keyword mentioned in the slack channel.

This is when the *Bolt framework* came into the architecture. Bolt framework is created by slack which allows developers to build custom slack apps that can be activated based on a message sent by the user in the slack channel [50]. This fit perfectly into the requirements of the slackbot application. The *Bolt for python* software development kit (SDK) was used [51] for building the custom slack app. Once the slack app was custom built, it was able to listen to a particular conversation and respond accordingly. The custom app was tested locally and it was able to respond to one particular keyword “krank” or “urlaub” in the slack channel. Since the project followed the iterative SDLC model, multiple iterative versions of a working application were built. There were a total of 3 iterations for the MVP version of the slackbot application at the end.

3.5.1. Iteration-1

The first iteration was to design, build, test and review the local version of the application running on a local IDE in the local environment. The first iteration was working as expected and passed all 3 phases - design, build and test. Hence the first iteration was reviewed and the decision to move the locally running app infrastructure to the cloud was made. In Iteration-1, the python script written was a basic code which initialised the slack app and listened to a particular keyword in the slack channel - “krank”. Below is the version of the code written for Iteration-1 of the slackbot application:

```
1  import os
2  from slack_bolt import App
3  from slack_bolt.adapter.socket_mode import SocketModeHandler
4
5  # Initializes your app with your bot token and socket mode handler
6  app = App(token=os.environ.get("SLACK_BOT_TOKEN"))
7
8  # Listens to incoming messages that contain "hello"
9  @app.message("krank")
10 def message_hello(message, say):
11     # say() sends a message to the channel where the event was triggered
12     say(f"Hey there <@{message['user']}>!")
13
14 # Start your app
15 if __name__ == "__main__":
16     SocketModeHandler(app, os.environ["SLACK_APP_TOKEN"]).start()
```

This script just initialises the custom slack app in line 15 & 16, and listens to the word “*krank*” in line 13. When a user mentions the word “*krank*” in the slack channel, the chatbot responds in the same channel with the message in line 12, by tagging the user. To initialise or start the app, the code gets the *SLACK_BOT_TOKEN*, a unique token generated for each slack app in the slack application, as explained in *section 2.2.1*, from the environment variables stored in the local environment.

3.5.2. Iteration-2

Next step of the plan was to find a way to integrate the custom slack app into AWS. The compute services of AWS - Lambda is a powerful service that can manage the loads that the slack app would receive from the users of the company. This knowledge was derived from studying the documentations of AWS and also by practically implementing and testing the integration of lambda with the slack app for the PoC. From the documents and prior experience, it was evident that AWS lambda was the best choice to connect to the slack app in terms of reliability and scalability of the slackbot application. For migrating the app from the local environment to the cloud, the Bolt framework packages along with other dependencies of python had to be pushed to the lambda function. For this task, a python library called *Python-Lambda* was used to deploy the local code along with Bolt framework packages and other dependencies to AWS lambda.

The *Python-Lambda* library helps streamline the python and AWS lambda connections from the 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. The command also packaged these dependencies along with the lambda handler function into a zip file and uploaded it to AWS lambda. In the slackbot application, this process was written as Infrastructure-as-Code (IaC) for creating the lambda function, specifying the properties of the lambda function and then deploying the lambda function on AWS. An AWS cloudformation template was created for this task. AWS Cloudformation allows modelling, provisioning, and managing AWS and third-party resources by treating

infrastructure as code. *Figure 41* is an illustration of how AWS Cloudformation works starting from creating the template in the local machine to getting deployed in the AWS.

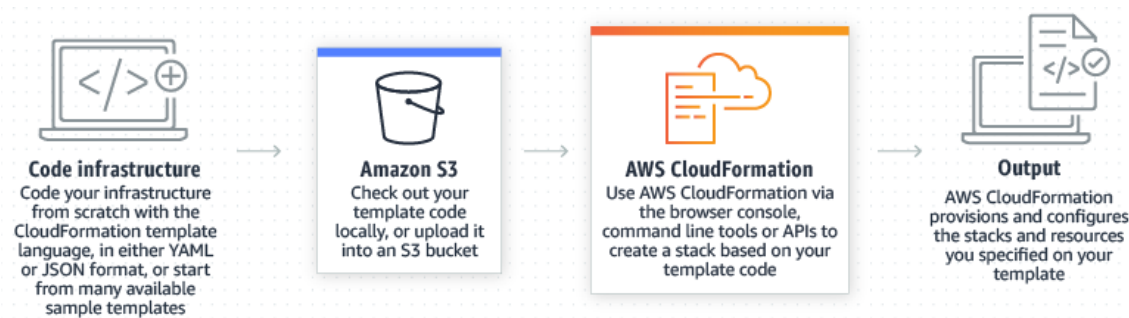


Figure 41: AWS Cloudformation Stack Deployment Process⁴²

Code infrastructure is written in either *YAML* or *JSON* format. For the slackbot application, the IaC was written in *YAML*. When the *YAML* code was executed locally using *python-lambda* library, a cloudformation stack was created in AWS and the stack takes care of creating the services which were written as code in the cloudformation template *YAML* file. *Figure 42* is the AWS cloudformation template file that was written to create the lambda function, *bolt_py_function*, and deploy it in AWS.

```

! bolt_py_cf.yaml > ...
1  # aws region for provisioning lambda
2  region: us-east-1
3
4  # lambda function configurations
5  function_name: bolt_py_function
6  handler: smart_slackbot.handler
7  description: Lambda function containing the Smart Slack chatbot functionality
8  runtime: python3.8
9
10 # iam role for managing the lambda function
11 role: bolt_python_lambda_invocation
12
13 # if access key and secret are left blank, boto will use the credentials
14 # defined in the [default] section of ~/.aws/credentials.
15 aws_access_key_id:
16 aws_secret_access_key:
17
18 # environment variables
19 environment_variables:
20   SLACK_BOT_TOKEN: ${SLACK_BOT_TOKEN}
21   SLACK_SIGNING_SECRET: ${SLACK_SIGNING_SECRET}
22
23
24 # build options
25 build:
26   source_directories: slack_bolt # a comma delimited list of directories in your project root that contains source to package.
27

```

Figure 42: AWS Cloudformation template to provision AWS Lambda⁴³

While writing the cloudformation template file for creating and deploying the lambda function in AWS, there are certain mandatory properties like *region*, *function_name*,

⁴² <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-06%2010-05-01.png>

⁴³ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-06%2010-23-16.png>

handler, *description*, *role*, and *runtime*. These mandatory properties are the configurations of the lambda function which was created in AWS. Whereas other properties like *aws_access_key_id*, *aws_secret_access_key*, *SLACK_BOT_TOKEN* and *SLACK_SIGNING_SECRET* are not mandatory but are specific to the master thesis. A cloudformation template can be written without specifying these properties too. *Figure 43* shows the deploy script that was used to deploy the IaC code into AWS.

```
$ deploy.sh
1  rm -rf slack_bolt && mkdir slack_bolt && cp -pr ../../slack_bolt/* slack_bolt/
2  pip install python-lambda -U
3  lambda deploy \
4  --config-file lazy_aws_lambda_config.yaml \
5  --requirements requirements.txt
```

Figure 43: Shell script code for deploying local codebase to AWS⁴⁴

In line 1, the shell script creates a folder in the local environment and copies all the dependencies created in the virtual environment to the folder. In line 2, the script installs the previously mentioned library *python-lambda* to the virtual environment where all the Bolt Framework packages are stored. Finally, in line 3, the script deploys the cloudformation template "*lazy_aws_lambda_config.yaml*" with the tag "*--config-file*" along with the requirements file "*requirements.txt*" with the tag "*--requirements*" to the AWS environment. The cloudformation template file has all the necessary information required for creating the lambda function in AWS.

The codebase was deployed to AWS, the third step of iteration-2 in the development plan was to connect the custom slack app to the lambda function to test if the application is working as per requirements from the AWS cloud. The same 3 stages of iterative SDLC model - Design, Build and Test were necessary to be passed from AWS, to build a successful working iteration of the application. There were two options available to connect AWS lambda to the 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 was the traditional method and provided a loose coupling advantage as part of the microservices architecture. Second option was to directly connect AWS lambda to slack app using the AWS lambda function URL.

⁴⁴ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-06%2009-51-34.png>

The lambda function URL is a new feature which was made public in 2022 by AWS [78]. There were some advantages of using function URL over API gateway which will be discussed in *section 4.4* as part of the whole application architecture.

Challenges in Iteration-2

Once the slack app and AWS were connected using the function URL, the slackbot application had to be tested from the cloud environment. This was one of the most challenging parts of the master thesis because of the difficulties that were faced to create a working iteration of the application from the cloud. When the code was tested from AWS, the same code that was working in iteration-1 in the local environment did not work from the cloud. This was because AWS lambda could not recognise the dependency packages of the Bolt framework which were uploaded along with other python dependencies. It took a few days, as seen in the Gantt chart (*Figure 32*), which tracked the project progress on a daily basis as part of the project management, to figure out the problem and understand what was causing this issue. After about 10 days, the issue was diagnosed. The issue was caused in AWS due to a version mismatch between the python code and the bolt dependency packages. The code did not work in AWS because the python version that was used in the local machine was the latest version *python 3.10* hence all the dependencies that created in the local environment for the Bolt framework were for *python 3.10* but AWS lambda did not support the latest python version as it had the provision to only use python 3.8. Therefore when the code was pushed from the local environment to AWS, both python and bolt framework dependencies were using *python 3.10* version. Hence It did not work in AWS. The iteration-1 had to be repeated after downgrading the local environment to *python 3.8*. All the project dependencies had to be downloaded again for *python 3.8* version, the iteration-1 was tested again after version change. Once iteration-1 was successful, the code and all dependency packages were again deployed to AWS using the cloudformation template file for iteration-2. The python version in lambda can also be seen in the cloudformation template file, in *Figure 42, line 8*. Even though the python version should not have caused the problem this was confirmed as the issue when the newly deployed code executed without any errors.

The slack app functionality was tested and it finally worked. The slack app was finally able to listen to the conversations in the slack channel from AWS. During this process, lots of documents, github issues and stackoverflow queries were referred to find the root cause of the problem.

3.5.3. Iteration-3

The slack app was working from the cloud and was able to react to the exact keywords “krank” and “urlaub” at the end of iteration-2. In iteration-3, the plan was to connect the dialogflow ES agent to the slack app. This was because at the end of iteration-2, the slack app could only react to the exact keywords but the app was also expected to respond to similar words used by the employees in the company to initiate the slack workflows - sick, vacation and invoice. This step included getting the dialogflow agent involved with either the slack app directly or through AWS. Different possibilities were analysed to build both versions’ architecture to see which offers more advantages. The test results showed that connecting dialogflow to AWS gave more flexibility as it allowed customisation of the slackbot application based on the project requirements. It also made AWS lambda into a middleware between slack and dialogflow, as shown in *Figure 44*, where MSA was still implemented through loose coupling between the dialogflow agent, AWS and slack.

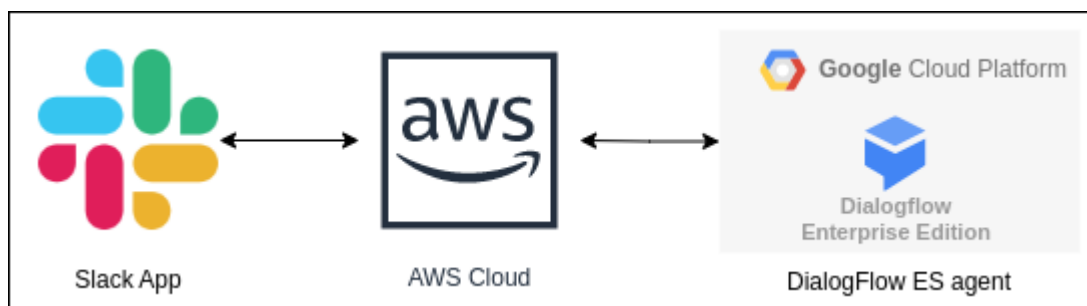


Figure 44: System Design Architecture for Slackbot - High level overview⁴⁵

There were two ways to connect AWS lambda and dialogflow ES agent. It was either through an API gateway similar to the PoC architecture (*Figure 3*) but that involved an additional AWS service - API Gateway to the existing architecture which impacted both the cost and the

⁴⁵ [https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/high-level-arch.drawio%20\(1\).png](https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/high-level-arch.drawio%20(1).png)

simplicity of the architecture. The application needs to be scalable and reliable but at the same time should have a simplistic architecture that has all the necessary functionalities. It should also be built in the most cost efficient way. This would help in improving the application security as mentioned in *section 1.6.2.2*. It was clear after considering the cost and simplicity of the application architecture, that the best option was to use the Google dialogflow API which was offered by the Google Cloud Platform (GCP) to help developers build conversational interfaces with their services [57]. Dialogflow API offered by GCP allowed the lambda function in AWS to directly invoke the dialogflow API when the user sends a message in the slack channel. Both user input and agent response were able to be sent through the dialogflow API. Therefore in iteration-3, the goal was to connect dialogflow with AWS using dialogflow API to send the user message to the ES agent and retrieve the response and workflow from the ES agent through the same API. First the dialogflow API needed to be tested locally and then iteratively pushed to AWS once the code and the API service was functioning as expected. This is the similar process followed for the slack app by iterating from iteration-1 to iteration-3.

Google Cloud Console

In order to access the dialogflow API from GCP, service account keys need to be downloaded from the Google cloud console. The service account keys are public-private keys that are generated using the Google cloud console. A service account in GCP is a special kind of account used by an application or a compute workload such as a Compute Engine virtual machine (VM) instance rather than individual users. The service account is identified by its email address which is unique to the GCP account. Applications use the service accounts to make authorised API calls by authenticating as either the service account itself or as Google workspace or as cloud identity users through domain-wide delegation [58]. To use a service account from outside of GCP, for example from AWS lambda in case of the slackbot application, an identity of the service account must be established [59]. Public or private key pairs provide a secure way of accomplishing this goal. When a service account key is created in the Google cloud console, the public key is stored on Google Cloud, while the private key

is available for download [59]. Therefore this process was followed and the private key was downloaded and stored as an environmental variable in the local virtual environment.

After setting up the GCP console, the local development stage for accessing dialogflow agent began. Similar to the Bolt framework libraries which were downloaded for the custom slack app in the local environment first, and then pushed to AWS, Google dialogflow libraries had to be downloaded for accessing the ES agent. Since the development programming language was python, the *google-cloud-dialogflow* package was downloaded and installed in the local environment. For the local development, Python Flask, a micro web framework for developing web applications in python was also installed [60]. This was done to test dialogflow API locally as an independent application. Below is the code developed locally to test dialogflow API using python flask packages and google-cloud-dialogflow packages.

```
1  from Flask import Flask
2  from Flask import request, jsonify
3  import google.cloud.dialogflow_v2 as dialogflow
4  from google.api_core.exceptions import InvalidArgument
5  import os
6
7  # Retrieve private key from local environment
8  os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = 'private_key.json'
9
10 # Google Cloud Console Project Details
11 DIALOGFLOW_PROJECT_ID = 'smartslackbot-nhxxh'
12 DIALOGFLOW_LANGUAGE_CODE = 'de'
13
14 # Initializes your app with your bot token and socket mode handler
15 app = Flask(__name__)
16 app.config["DEBUG"] = True
17
18 # Test if the app responds to GET request
19 @app.route("/")
20 def root():
21     return "Hello World"
22
23 # Create a POST method to Dialogflow ES agent and receive response
24 @app.route('/api/getMessage', methods=['POST'])
25 def home():
26     message=request.form.get('Body')
27     #user=request.form.get('From')
28     session_client = dialogflow.SessionsClient()
29     session =
30     session_client.session_path(DIALOGFLOW_PROJECT_ID,SESSION_ID)
31     text_input = dialogflow.types.TextInput(text=message,
32     language_code=DIALOGFLOW_LANGUAGE_CODE)
33     query_input = dialogflow.types.QueryInput(text=text_input)
```

```

34 try:
35     response = session_client.detect_intent(session=session,
36     query_input=query_input)
37 except InvalidArgument:
38     raise print("Query
39 text:", response.query_result.query_text)
40 print("Detect Intent:", response.query_result.intent.display_name)
41 print("Detected Intent
42 Confidence:", response.query_result.intent_detection_confidence)
43 print("Fulfillment text:", response.query_result.fulfillment_text)
44 return response.query_result.fulfillment_text
45
46 # Start your app
47 if __name__ == "__main__":
48     app.run()

```

The private key file which was downloaded from the Google cloud console to the local environment, *private_key.json*, is accessed in line 8 by the OS library of python. The private key is then extracted from the json file and is stored in the local environmental variable, *GOOGLE_APPLICATION_CREDENTIALS*. Line 11 & 12 denote the dialogflow specific configurations - *project ID* and *language*. “*Smartslackbot-nhxxh*” is the project in which the dialogflow ES agent is built and hence it can be accessed only by specifying the project name. The language can be switched between “*de*” for German and “*en*” for English to communicate with the dialogflow agent. Since German is the official language being used in the company, the language code for dialogflow is set as “*de*”. The app is started using *app.run()* command in line 48.

The local testing for the dialogflow API was done using Postman. Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration to create better APIs quickly [61]. The local app was running and was hosted in port 5000 locally. Therefore the dialogflow API was accessible through the URL:

<http://localhost:5000/api/getMessage>

This URL was tested in Postman to check if the dialogflow ES agent was accessible from the local environment, *Figure 45* shows that the same port 5000 URL with a POST method was working. When the sample user input - “*Heute fühle ich mich nicht gut*” was sent as text in the *Body* of the API request, the response is received from the dialogflow agent as seen in the response body.

This confirmed that the SmartSlackBot agent built inside Google Cloud Platform was accessible from the local machine through dialogflow API.

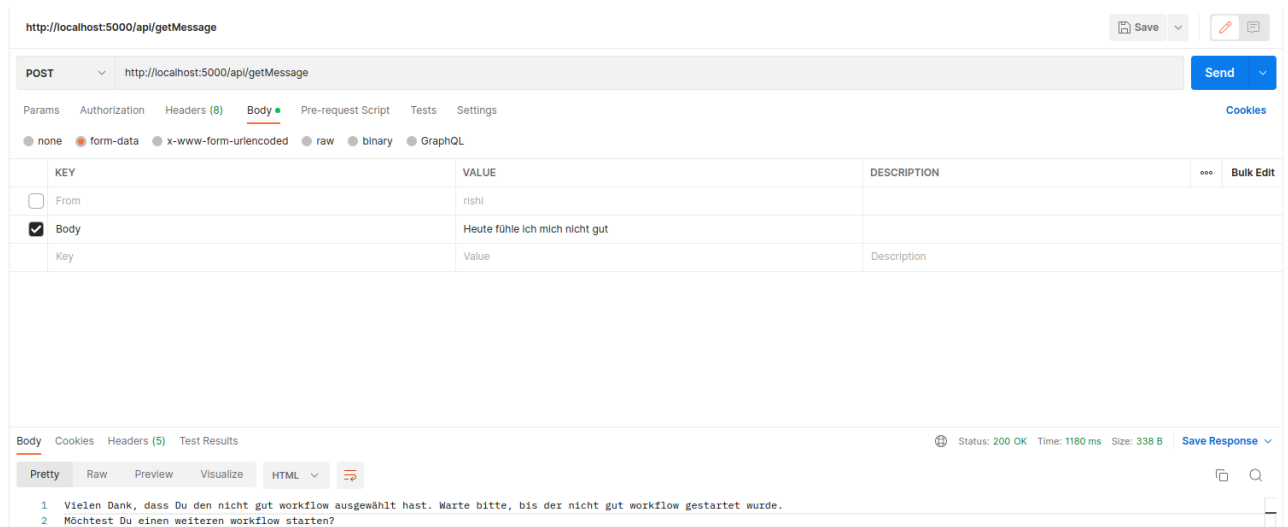


Figure 45: Postman - Dialogflow API test⁴⁶

A dummy function, in line 20, in the local dialogflow code was also created to test if the web application created by Flask was working correctly for the GET method. Figure 46 shows the response from the postman service on sending a GET method to the URL:

<http://localhost:5000/>

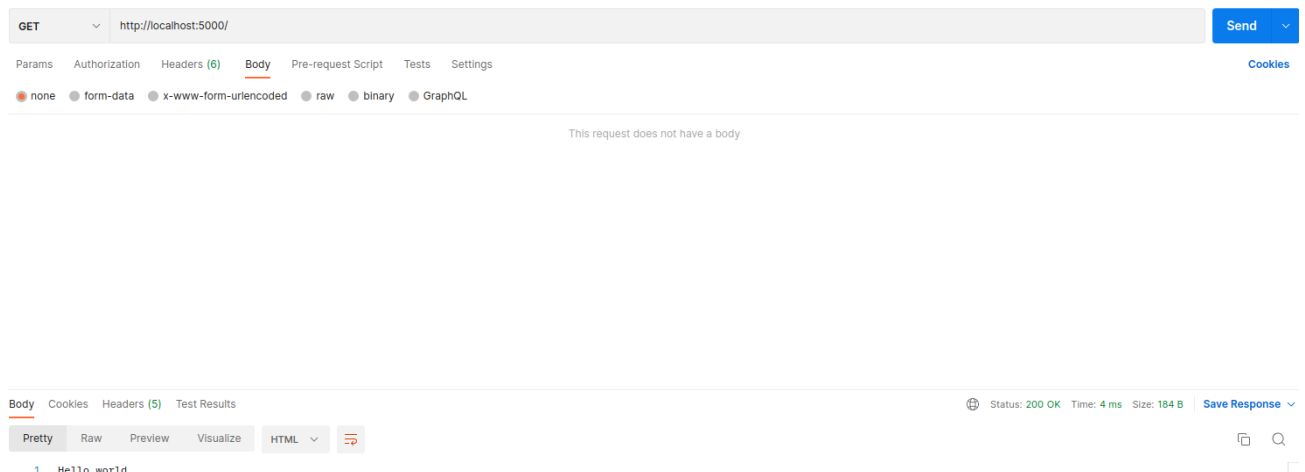


Figure 46: Postman - Dummy function test⁴⁷

⁴⁶ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-12%2012-34-35.png>

⁴⁷ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-three/Screenshot%20from%202023-02-12%2013-01-10.png>

The response body shows “*Hello world*” which was the expected result from line 21 of the local dialogflow code. Iteration-1 was successful for the dialogflow API.

Therefore the next step was to push the local dialogflow code to AWS and perform testing from the AWS environment. This step was similar to iteration-1 where the local code was pushed along with all its dependencies to AWS using the AWS cloudformation template. Therefore the same cloudformation template was used to migrate the local dialogflow code along with its dependencies to a new AWS lambda function, *smart-slack-dgflow*, for the dialogflow functionality. Since all the local code has been migrated to the cloud environment, the next step of the master thesis was to design the complete architecture of the application and build the Minimum Viable Product (MVP) that could be deployed in the production environment.

CHAPTER

FOUR

ARCHITECTURE

The slackbot application was built across 3 iterations to reach a working prototype that works as per the requirements of stakeholders. In order to build a scalable, reliable and robust application, the architecture must be well planned and designed. Therefore architecture plays an important role in the success of an application.

4.1. Importance of Architecture

Software architecture represents a high-level abstraction of the application that most of the application's stakeholders can use as a basis for creating mutual understanding, forming consensus and communicating with each other [63]. Architecture of a software represents the embodiment of the earliest set of design decisions made about an application. These early bindings carry weight far out of proportion to their individual gravity with respect to the application's remaining development, its service in deployment, and its maintenance life [63]. Architecture contains a relatively small, easily understandable model of the application, how it is structured and how its components work together. This model is transferable across systems. It can be applied to other systems which have similar requirements and can promote large scale reuse [63].

When designing the architecture, it is important to have a clear, well structured architecture that supports the workload of the application and at the same time offers the possibility to modify the architecture to adapt to the future technological advancements. In a business environment where companies follow the *“build, test and iterate quickly”* methodology [64], this agility in being able to modify the application architecture quickly based on future needs

provides an added advantage to companies. This is easier in startups or small companies where smaller products are created and deployed in the production environment. In large companies like Google or Facebook where the core application is built with 1000s of lines of code it is a complex and tedious process to modify the application architecture. This is one of the reasons why a more robust and agile model of architecture was chosen to build the slackbot application for the master thesis. Well designed serverless applications are decoupled, stateless and use minimal code. The simplicity of design and low-code implementation must be maintained even when the project grows. These are some of the best practices of a good serverless architecture.

4.2. Microservices Architecture

Microservices Architecture (MSA) was chosen over other types of architectures like monolithic or layered because each microservice in MSA can be independently developed, updated, deployed, scaled or replaced. This is because each microservice is independent of each other [62]. This freedom is not possible in monolithic architecture because if one element in a monolithic architecture fails, the whole architecture collapses. This is because all the elements in a monolithic architecture are built as a single unified unit [62]. This was an important consideration in building the slackbot application for the master thesis since the current version is only the MVP and the application will be developed further in the future with more features. 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 requirements in the future, then they don't have to break down the whole architecture. The company can just replace the dialogflow API with the new product's API and the application will function perfectly well. Therefore to build scalable applications that don't need major modifications in the future, MSA is preferred over monolithic architecture [53]. This makes MSA agile and the best option for building the slackbot application in an ever changing tech environment.

In developing the architecture for the slackbot application, the goal was to implement the DevOps ideology from the early stage of SDLC as it provides valuable insights into the

production environment in which the application will be deployed. DevOps is a collaborative and multidisciplinary organisational effort to automate continuous delivery of new software updates while guaranteeing their correctness and reliability [65]. DevOps is an evolution of the agile movement. DevOps proposes a complementary set of agile practices to enable the iterative delivery of software in short cycles effectively [66]. Thereby helping in improving the software quality [52]. MSA emerged from a common set of DevOps ideology that came into existence in companies in 2009 [67]. Since, implementing automation was the foundation of the master thesis, it made sense to implement automation in also building the application. This included fully automated pipelines from code repository in Github to production. Automation includes release pipelines with continuous integration (CI) [68], automated testing, and continuous delivery (CD) [69]. The goal was to build the slackbot application by following these DevOps ideologies thereby automating the whole application life cycle from planning stage to monitoring stage.

4.3. Slackbot Architecture

The Slackbot application built for the master thesis consists of 3 main environments - slack, AWS and Google Cloud Platform. Slack acts as the frontend for the users to interact with the application whereas the AWS and Google Cloud Platform acts as the backend of the application. These environments interact with each other through HyperText Transfer Protocol (HTTP/HTTPS) endpoints or through API calls. The AWS environment acts as a middleware as explained in iteration-3 in *section 3.5.3*. AWS connects to both slack and dialogflow to receive and send data across all 3 platforms. *Figure 47* is the architecture diagram for the MVP version of the slackbot application. The user sends a message in the slack channel, *#ber-allegemein*, where the custom slack app, Smart Slack Bot, is installed. The configuration details of the custom slack app has already been discussed in *section 2.2.1*. Smart slack bot has a unique token called *Bot Token* which is generated by the slack when the app is created. This token is essential to validate the requests coming into AWS from the smart slack bot. It is stored inside the S3 bucket, *slackbot-helper*.

key stored in AWS console using its key ID. Each key generated by AWS KMS has a unique key ID. Therefore the slack bot token was encrypted locally using the KMS key in the AWS CLI. The encryption is a 256-bit AES-GCM encryption after which the encrypted bot token was generated. This encrypted bot token was copied and stored in a json file, *slack-config.json*.

4.3.2. Amazon S3 Service

The json file containing the encrypted bot token also contains other variables such as KMS region, KMS key alias, app mentioned, slash command, app home opened. These variables are used by the lambda function, *bolt_py_function*, for different purposes which will be explained later in this section. This json file was stored inside the s3 bucket, *slackbot-helper*, which was created for the slackbot application. S3 buckets are object storage services which offer industry-leading scalability, data availability, security, and performance with cost-effective pricing. It is possible to store and protect any amount of data for any use case, such as data lakes, cloud-native applications, and mobile apps inside s3 buckets. For the slackbot application, s3 buckets were only used to store the configuration file which will be accessed by the lambda function. A new directory *"/config"* was created inside the s3 bucket and the json file was stored inside this directory. The json file will be accessed by the lambda function, *bolt_py_function*, every time this lambda function is invoked.

4.3.3. AWS Lambda Service

Lambda is a compute service offered by AWS that lets users run code without provisioning or managing servers. Lambda runs the code on a high-availability compute infrastructure and performs all of the administration of the compute resources including server and operating system maintenance, capacity provisioning, automatic scaling, and logging. With Lambda, users can run code for virtually any type of application or backend service. The slackbot application uses 2 lambda services - *bolt_py_function* and *smart-slack-dgflow*. The *bolt_py_function* lambda function is connected to the custom slack app, Smart Slack Bot, which is located in the slack application and to the Amazon Simple Notification Service (SNS). The *smart-slack-dgflow* function, on the other hand, is connected to the dialogflow agent in GCP and to Amazon SNS.

Both the lambda services have memory of 512mb and 128mb respectively and have a timeout of 15 seconds each. These configurations are essential from the cost perspective for the company. Both lambda functions are written in python 3.8. Although there are many similarities between the two lambda functions, their functionalities differ significantly. The *bolt_py_function* is the main compute service used in the slackbot application. This function is connected to the frontend (Slack) using the function URL. The *smart-slack-dgflow* function is used to send the user messages to the dialogflow agent in the GCP using dialogflow API and receive the response from the agent and send it to the *bolt_py_function* lambda function.

Function URL

A function URL is a dedicated HTTP(S) endpoint for the lambda function. A function URL can be created and configured through the Lambda console or the Lambda API. When a function URL is created, the lambda automatically generates a unique URL endpoint for each user. For the slackbot application, the function URL was created to connect to the Smart Slack Bot app on the slack application. In PoC architecture (*Figure 3*), the slack app was connected to the lambda function using the API gateway service but for slackbot application, it was decided that an additional service was not necessary. The same functionality that the API gateway service provides is also offered by the lambda function URL. The function URL also has some major advantages over API gateway and other AWS services:

1. API Gateway or Load Balancer only gives a Domain Name Service (DNS) in addition to the functionalities of a function URL. This is to protect confidential information such as Account Resource Number (ARN) or AWS config details. ARN has some confidential information such as account ID, region info, etc. which should not be exposed to unauthorised personnel. For the slackbot application, although lambda was connected to an external application, slack, the custom slack app, Smart Slack Bot, is being protected by the slack application. Only authorised users can access the custom slack app's configuration page. Therefore even if the ARN information about the AWS account is shared to the slack app, the information is protected at the slack end. The AWS account in which the infrastructure is set up is also the company's

account therefore only the company system admin can access both the slack app and the AWS account apart from the developer.

2. Other AWS services add to the cost of infrastructure for every message coming from slack to the AWS cloud whereas function URL does not have additional costs and is associated with the costs of the lambda function. This not only reduces costs for the company but also helps in reducing the complexity of the application architecture which is an important criteria in building scalable modern applications.
3. Function URL allows version control of the lambda function. Function URL doesn't change with changes in lambda code or configurations. Therefore a lambda function can have multiple Function URLs where one URL can be used for the latest version of lambda and the other URL for alias version.
4. Function URL is HTTPS therefore it uses AWS provided certificates for a secured connection. It also provides an additional layer to use IAM as authenticator where a secret access key and a secret key can be used to invoke the url. A designated role can be assumed within the aws account and only that role will have the permission to call the lambda. If the lambda function is directly called from a browser then a CORS is also available to invoke the lambda url similar to API gateway service. For the slackbot application, the additional layer of IAM authentication was not selected because it increases the latency of the application. The slackbot application also doesn't use a browser as frontend hence CORS functionality was also not required.

Due to these advantages, the function URL of *bolt_py_function* lambda function was used to connect to the slack app, Smart Slack Bot. The function URL is provided in the “*Redirect URLs*” section of the Smart Slack Bot's home page. The topic of Redirect URLs was already discussed in the subsection of *section 2.2.1*.

Lambda Function - *bolt_py_function*

The *bolt_py_function* lambda function is the main compute function of the slackbot application. The majority of backend code is written inside this function. Therefore this function has 7 functionalities as the main backend compute service of the slackbot application:

1. The first functionality is to start the custom slack app installed in the company slack channel from the AWS environment using the bolt framework packages. The custom slack app then listens to all the messages in the slack channel. The messages come into the AWS environment to the lambda function using the function URL.
2. The second functionality is to authenticate the incoming messages. The function performs authentication of all messages that come from the slack environment to the lambda function. This is to ensure that there is no unauthorised access to the backend of the application - AWS or the GCP environment, from unauthorised slack channels or applications. This authentication is done by accessing the s3 bucket, *slackbot-helper*, to retrieve the json file - *slack-config.json* which is stored inside the *"/config"* directory of the s3 bucket. The values in the json file - encrypted bot token, KMS region, KMS key alias, app mentioned, slash command, and app home opened are copied and stored in local variables inside the function. The encrypted key, KMS region and KMS key alias values are used to decrypt the custom slack app's bot token which was encrypted in the local machine earlier using AWS KMS. The decrypted bot token returns the original bot token value which is compared with the incoming request.

Since the slack app and AWS lambda are connected through a webhook integration. When the users send a message in the slack channel in which the slack app is installed, an event is triggered and the message is sent as a webhook payload along with other details such as the custom app ID, slack channel ID, user ID, etc. The custom app ID that comes into the lambda function as webhook payload is the bot token that is encrypted and stored in the s3 bucket. Therefore, the lambda function takes the custom app ID from the incoming webhook payload and compares it with the decrypted bot token value which was stored in the s3 bucket. If both values match then the authentication is successful. This means that the message coming from the slack app is the authorised slack app whose bot token is stored in the s3 bucket. If the values don't match then the authentication has failed and an error message is sent to the user in slack.

3. Once the authentication is successful, the third functionality of the lambda function is to send the user message to the *smart-slack-dgflow* function through the SNS topic. This is because the lambda function already performs a lot of functionality and to also connect to the dialogflow in the GCP environment is not a best practice while building MSA based applications. Therefore the message is sent to the *smart-slack-dgflow* lambda function which sends the response from dialogflow agent back to the *bolt_py_function* function along with the workflow type detected by the dialogflow agent.
4. The fourth functionality is to trigger the correct workflow based on the workflow type detected by dialogflow agent. The workflow is triggered inside the lambda function using the workflow's webhook url. If the workflow type does not match with any of the predefined workflow types - sick, vacation or invoice then the response is sent to the user without initiating the workflow.
5. Fifth functionality of the lambda function is to allow users to redirect to the confluence page since this is also part of the requirement that the users are given an option to choose between initiating a workflow or reading the documentation in the confluence web page. This is done by storing the confluence web page URL inside a button click event.
6. Sixth functionality is to allow users to communicate with the app using different ways. For example, a custom slack app can be accessed in 4 different ways in slack:
 1. listening and responding to a keyword
 2. direct messaging to the slack app
 3. using the slash ('/') command in a slack channel
 4. mentioning the app (using '@') in a slack channel.

The rest of the values stored in the json file inside the s3 bucket are used for this purpose. The app mentioned and slash command values are used to start the app and allow the users to communicate with the app manually. When a user mentions the app or uses the slash command *"/smartslackbot"*, in a slack channel, the chatbot will respond the same way. The slash command functionality also needs to be configured in the custom slack app's slash command page, like in *Figure 48*.

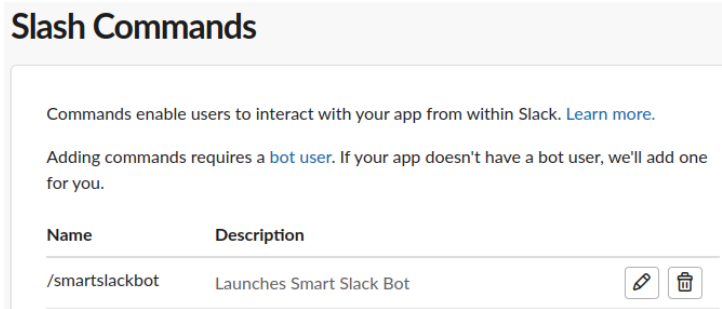


Figure 48: Slash command for Smart Slack Bot⁴⁹

- The seventh and final functionality is to create the user interface of the custom slack app that the users can initiate conversations with. Creating buttons and texts that pop up along with the slack app when the app recognises a keyword is done using the Block Kit Builder provided by slack. Block kit is a clean and consistent UI framework for slack apps [73]. Block kit can be used to customise the order and appearance of information in the slack app to guide users through the app's capabilities by composing, updating, sequencing, and stacking blocks. Blocks are the reusable components of the block kit framework that work almost everywhere in Slack [73]. The app's response to each workflow type and the app home page is designed using the block kit framework in this function. The UI of the app will be shown in section 4.7.

Lambda Function - smart-slack-dgflow

The *smart-slack-dgflow* function has a very specific functionality unlike the *bolt_py_function* function. The *smart-slack-dgflow* function is only used to connect to the dialogflow agent in the GCP environment. The lambda function is invoked using a push notification service - SNS. When the *bolt_py_function* function receives the message from the custom slack app, it authenticates the incoming payload and after successful authentication triggers the SNS topic, *smart-slack-sns*, by publishing a message across to the SNS topic. The user message is also published to SNS when the *bolt_py_function* function triggers the SNS topic. The *smart-slack-dgflow* function receives the published message from the SNS topic after being

⁴⁹ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-four/Screenshot%20from%202023-02-13%2003-38-15.png>

invoked by the push notification service and sends the received user message to the dialogflow agent through the dialogflow API, used in iteration-3 of the *section 3.5.3*.

This function contains all the packages related to Google cloud as explained in iteration-3 of the *section 3.5.3*. Therefore when the function is invoked, an API request with POST method is sent to the dialogflow agent along with the user message. The API response from the dialogflow agent contains the workflow type. This workflow type is sent back to the main compute function "*bolt_py_function*" through the SNS topic.

4.3.4. AWS SNS Service

Amazon Simple Notification Service (SNS) is a fully managed publish-subscribe (pub/sub) service [74] for application-to-application (A2A) and application-to-person (A2P) messaging. SNS sends notifications in two ways - A2A and A2P. A2A is a kind of message traffic where an application receives a message from another application. It provides high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. These applications include Amazon Simple Queue Service (SQS), Amazon Kinesis Data Firehose, AWS Lambda, and other HTTPS endpoints. A2P is a kind of message traffic where a person receives a message from an application like One Time Password (OTP), marketing messages etc. A2P functionality allows the applications to send messages to their customers with SMS texts, push notifications, and email. For the slackbot application functionality, only A2A messaging is used.

Amazon SNS is an event-driven hub that has native integration with a wide variety of AWS event sources and event destinations. Event-driven computing is a model in which subscriber services automatically perform work in response to events triggered by publisher services [75]. This paradigm can be applied to automate workflows while decoupling the services that collectively and independently work to fulfill these workflows. Therefore an event-driven architecture is applied between the two lambda functions and the SNS topic in the backend of the slackbot application which is built based on MSA. The SNS topic created for the slackbot application is called *smart-slack-sns*. This is a standard topic whose functionality is to trigger an event when the *bolt_py_function* lambda function sends the

user message to the topic by publishing a message. Upon receiving the message, the SNS topic triggers an event invoking another lambda function *smart-slack-dgflow*, and passes the message to the invoked lambda function. The SNS topic invokes the lambda function asynchronously, meaning the SNS does not wait for a response from the *smart-slack-dgflow* lambda function. The message is handed-off to the lambda and the event is completed at the SNS end. This is called asynchronous invocation. Similarly the workflow type from the *smart-slack-dgflow* lambda function is asynchronously pushed to the main compute function *bolt_py_function*. SNS provides flexibility, reliability and enables loose coupling thereby making the application robust.

4.3.5. List of AWS Services Used

Below is the list of AWS services used in building the slackbot application along with its estimated costs. These costs are collected from the official AWS website.

S.No	Services	Costs
1	Lambda	\$0.20 per 1 million requests for first 6 billion GB-seconds
2	S3 Bucket	\$0.023 per GB for first 50TB/month
3	KMS	\$0.03 per 10,000 requests
4	SNS	No charge for deliveries to Lambda
5	Cloudwatch	1,000,000 free requests/month for storing Lambda logs

Table 2: List of AWS services used and their costs

4.3.6. Estimated Cloud Costs

The entire backend infrastructure of the slackbot application is hosted in the cloud environment. Two different cloud services are used for the slackbot application - AWS and GCP. It is important to consider the costs associated with using the cloud services when developing cloud applications. Costs related to AWS and its individual services that are used for the slackbot application are covered in *section 4.3.5*. In the case of GCP, Google dialogflow costs between \$0.002 to \$0.008 per request, as discussed in *section 1.1*. For the

slackbot application, the Essentials (ES) type of dialogflow agent is used for the NLU requirements of the application. Therefore the cost for using dialogflow ES agent is \$0.002 per request based on the Google cloud official website.

Both AWS and GCP costs are estimated based on the number of requests received from the users. Therefore the cloud costs for the slackbot application also depends on the number of messages sent by the employees in the slack channel of the company. This is because all messages from the slack channel pass through both AWS and GCP services to capture the keyword sent by the users. By estimating the approximate number of messages sent in the slack channel where the slackbot application is installed, the approximate cloud costs can be calculated. Since the monthly user count for the application is pretty low as compared to customer facing applications, most of the AWS services that are used by the slackbot application are covered under *free tier*. Below is the estimated cloud cost for the slackbot application at an average estimate of 50 messages per day in the slack channel.

Period	Approximate No. of Messages in Slack	AWS	GCP
Daily	50	\$0,45	\$0,1
Monthly	1520	\$0,45	\$3,04
Yearly	18.250	\$0,73	\$36,5
	Total Costs	\$38 per annum	

Table 3: Estimated Cloud Costs for Slackbot

For AWS, all individual services listed in *Table 2* are included in the costs estimated in *Table 3*. As mentioned earlier, AWS costs are covered under the *free tier* for lower number of requests both monthly and annually. Since the costs are pretty low, annual costs for AWS is calculated which comes around \$0,73 USD. Therefore AWS services definitely will fall under the *free tier* category. For GCP, there are fixed costs unlike AWS. The annual costs for using the Google dialogflow ES agent is \$36,5 USD for 18.250 requests annually. This is the average estimate of 50 messages sent in the slack channel per day which is higher than the actual numbers. Therefore the combined estimated cloud costs for the slackbot application per year is \$38 for the company.

4.4. Cloud Application Security

Cloud application security is defined as a set of policies, governance, tools and processes used to govern and secure information exchanged within collaborative cloud environments and applications deployed to the cloud [76]. As part of security measures discussed in *section 1.6.2.2*, keeping the application as compact as possible within easily migratable cloud areas was the goal to enhance the cloud application security. Cloud application security issues are cyber threats that a cloud-based application is or could be exposed to. These threats include [76]:

1. Unauthorised access to application functionality or data
2. Exposed application services due to misconfigurations
3. Hijacking of user accounts because of poor encryption and identity management
4. Data leakage from insecure APIs or other infrastructure endpoints
5. Distributed denial of service (DDos) attacks related to poorly managed resources

The best practices for effective security and protection against these cyber threats for cloud native application are [76]:

1. Identity access management
2. Encryption
3. Threat monitoring
4. Data privacy & compliance
5. Automated security testing

Some of the cyber security threats against which measures were taken for the slackbot application are:

1. Unauthorised access to application functionality or data
2. Exposed application services due to misconfigurations

4.4.1. Unauthorised access to application functionality or data

This is managed by clearly defining the roles and access for each environment and each user within the organisation. In the slack environment, the custom slack app configurations or

slack user data can only be accessed by the admin and the developer. Each slack workspace has account level restrictions which allows no unauthorised user to access the custom slack app or the data that is being sent through the custom slack. Apart from account level permissions, slack also has in-built security features [77] that protect user data and secure against cyber threats.

In AWS, Identity Access Management (IAM) best practices are followed by creating roles and users with only minimal permissions required to perform the necessary operations in the AWS environment. All the services provisioned inside AWS for slackbot application only have these restricted roles as the execution role. Apart from these best practices, an explicit authentication system using AWS KMS is also built inside the slackbot architecture as explained in *section 4.4.3*, under *Lambda Function - bolt_py_function* subsection.

In GCP, the dialogflow API is protected by creating a service account role which manages the permissions to access the private key which was downloaded as part of private/public key authentication to connect to the dialogflow API. This was already covered in iteration-3, under *Google Cloud Console* subsection of *section 3.5.3*. Only users with both the public and private key can access the dialogflow API.

4.4.2. Exposed application services due to misconfigurations

This threat is common when using IaC to provision the infrastructure of applications. In a complex application where there are multiple services and a large infrastructure, a misconfiguration in the infrastructure security could be a potential threat to the application's security. This is prevented in the slackbot application since the entire infrastructure for the backend was not provisioned using IaC. Only the lambda functions and the code inside the lambda functions inside AWS were provisioned using IaC. Despite this there is an authentication system in place for all entries into the *bolt_py_function* lambda function using AWS KMS. The *smart-slack-dgflow* lambda function can only be accessed by the SNS and dialogflow. The response coming from the dialogflow is protected by the API keys downloaded from the Google cloud console. Hence a layer of protection is available for both the lambda functions against this security threat.

No explicit security measures are taken for the other three threats. The slackbot application is dependent on in-built security features of AWS, GCP and slack against the remaining 3 cyber security threats.

4.5. End-to-End Testing

Once the application has gone through multiple iterations of build, test and iterate as a best practice to build scalable applications. The final step of the SDLC is to do an End-to-End (E2E) testing. End-to-end testing is done to test the functionality and performance of an application under product-like circumstances and data to replicate live settings. The goal is to simulate what a real user scenario looks like from start to finish. The completion of this testing is not only to validate the system under test, but to also ensure that its subsystems work and behave as expected. For the slackbot application, the E2E testing is done by sending messages as the end user from the slack channel where the custom slack app is installed and checking if the received results are as expected.

4.5.1. Benefits of End-To-End Testing

Conducting E2E testing helps ensure that the product is production-ready and avoids risks post-release. This process is essential to the application's success for a few key reasons:

1. **Confirms the Application Health:** E2E testing validates that the software is functional at every level - from frontend to backend as well as provides perspective on its performance across different environments.
2. **Expands Test Coverage:** By incorporating many different subsystems in the testing process, the test coverage is effectively expanded and additional test cases that may have not previously been considered are created.
3. **Detects Bugs and Increases Application Productivity:** In E2E testing, the application is usually tested after every iteration, which means issues can be found and fixed faster. This reduces the chances of bugs making it further into the testing process and also in production, thereby ensuring the application workflows operate seamlessly.

4. Reduces Testing Efforts and Costs: With fewer bugs, breakdowns, and comprehensive testing each step of the way, E2E testing also decreases the need to repeat tests and ultimately the costs and time associated with doing so.

4.5.2. Test Results

The slackbot application was tested E2E from the slack channel *#test-slack-bot* where the custom slack app *Smart Slack Bot* was installed. *Figure 49* shows the results of the E2E test performed by simulating an end user. The slack app was listening to the exact words or synonyms of krank, urlaub and rechnung.

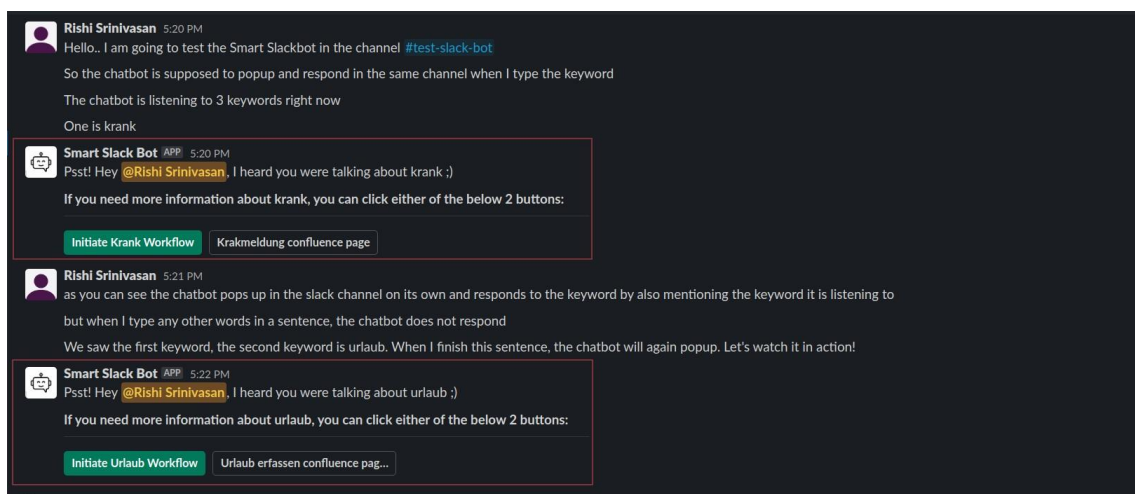


Figure 49: Slackbot - E2E Testing Results⁵⁰

Message 1: “Hello.. I am going to test the Smart Slackbot in the channel #test-slack-bot”

There was no response from the slack app since the message has no words or synonyms related to krank, urlaub or rechnung.

Message 2: “So the chatbot is supposed to pop up and respond in the same channel when I type the keyword”

There was no response again from the slack app since the message has no words or synonyms related to krank, urlaub or rechnung.

Message 3: “The chatbot is listening to 3 keywords right now”

⁵⁰ <https://github.com/rishi-srinivasan/master-thesis/blob/main/images/chapter-four/Screenshot%20from%202023-02-04%2017-22-49.jpg>

There was no response from the slack app since the message has no words or synonyms related to krank, urlaub or rechnung.

Message 4: *“One is Krank”*

There was a response from the slack app after message 4. This is because the word krank was used in message 4. The *Smart Slack Bot* tags the user and mentions the keyword that the user had used in the message *krank* and asks if the user wants to select either of the 2 options. The *Smart Slack Bot* always provides two options as discussed in *section 2.2.3.1*.

Message 5: *“as you can see the chatbot pops up in the slack channel on its own and responds to the keyword by also mentioning the keyword it is listening to”*

Again there was no response from the slack app since it has no words or synonyms related to krank, urlaub or rechnung.

Message 6: *“but when I type any other words in a sentence, the chatbot does not respond”*

There was no response from the slack app since it has no words or synonyms related to krank, urlaub or rechnung.

Message 7: *“We saw the first keyword, the second keyword is urlaub. When I finish this sentence, the chatbot will again popup. Let’s watch it in action!”*

After message 7 was sent, there was a response from the *Smart Slack Bot* again. Since the word urlaub was mentioned in message 7, the *Smart Slack Bot* was listening to the messages and was able to react to the keywords mentioned in the slack channel. The *Smart Slack Bot* again tags the user and mentions the keyword that the user had entered in the message *urlaub* and asks if the user wants to select either of the 2 options.

If the user clicked on the *initiate workflow* button then the workflow would directly initiate from the slack channel. If the user clicked on the *confluence page* button then the user will be redirected to the respective confluence page. These were the expected functionalities of the slackbot application. Hence the E2E testing was successful.

CHAPTER

FIVE

ETHICAL CONCERNS

Some of the digital technologies that were involved in building the slackbot application are chatbot agents, cloud services, LLMs, etc. As much as these technologies are revolutionary and have modified how modern applications are built, they also bring considerable amounts of risks and ethical concerns with them. The two main areas of focus in the master thesis will be on LLMs and data protection in the cloud environment.

5.1. Large Language Models

Large language models are deep learning algorithms that can recognize, predict, generate text, etc. The LLMs can also generate other contents based on knowledge gained from large datasets [79]. LLMs are among the most successful applications of the transformer models. In addition to accelerating NLP applications like translation, chatbots and AI assistants, LLMs are used in healthcare, software development and in many other fields [79].

5.1.1. How do Large Language Models work

LLMs learn from huge volumes of data. The volumes have also changed over time. LLM models of today's top AI based companies like Open AI's GPT-3 or Google's Bard are typically trained on datasets extracted from the entire internet. The training data consists of almost all the texts available on the internet. These massive amounts of text data are fed into the AI algorithms using unsupervised learning. Through this method, a LLM learns words, their relationships between them, and the concepts behind them.

LLMs can also be customised for specific use cases including techniques like fine-tuning or prompt-tuning, which is the process of feeding the model small bits of data to focus on to train it for a specific application. The transformer model architecture is the building block behind the largest and most powerful LLMs in the world.

5.1.2. Environemental costs from operating LLM

LLMs have the potential to create significant environmental costs through energy consumption, carbon emissions for training and operating the models, demand for water to cool the data centres where computations are run [80][pg.32]. These demands have impacted on the ecosystems and the climate, including the risk of environmental resource depletion [80][pg.32]. Although LLMs are often talked about, it has received less attention that most AI based companies spend more energy on operating the LLMs (performing inference) than training them: AWS claimed that 90% of cloud ML demand is for inference, and Nvidia claims 80-90% of the total ML workload is for inference [80][pg.32]. Therefore it is expected that companies providing LLM services may spend more energy, money and time on operating such models than on training them. Based on this, it can be expected that the environmental costs of operating LLMs may exceed the energy cost of training them which is a significant environmental problem [80][pg.32].

There have been researches to estimate the approximate amount of CO₂ being emitted in training the LLMs. While the average human is responsible for an estimated 5t CO₂ per year, a Transformer model with neural architecture search emitted 284t of CO₂ [81][pg.612]. Training a single BERT transformer based model without hyperparameter tuning on GPUs was estimated to require as much energy as a trans-American flight [81][pg.612]. Although some of this energy comes from renewable sources, the majority of energy used in training or operating the LLMs are from non-renewable sources [81][pg.612].

5.2. Data Protection in Cloud

Protecting data in cloud environments is a challenging task. When on-premise infrastructure is used to build applications, the complete responsibility of securing the data is with the

on-premise company but when a company uses cloud infrastructure the responsibility is divided between the company and the cloud vendor. This is where the topic of cloud security comes into play. Cloud security is the set of strategies and practices for protecting data and applications that are hosted in the cloud environment. Cloud security is a broad area in the field of security and it is highly impossible to prevent all varieties of cyber threats but a well designed cloud security strategy can vastly reduce the risk of cyber attacks [82]. Even with the risks involved, cloud computing is often more secure than on-premise computing. Most cloud providers have more resources for keeping data secure than individual businesses do which lets cloud providers keep infrastructure up to date and patch vulnerabilities as soon as possible [82].

In the slackbot application, when multiple cloud environments like AWS, GCP and slack are used, the best way to protect the data is to encrypt the data. Encrypting data in transit addresses both data travelling between a cloud and a user, and data travelling from one cloud to another. In terms of encrypting the data, in the slackbot application, data at transit is never encrypted explicitly as no personal or confidential data sent between different environments. The data from slack to AWS contains information about the custom slack app along with the user message. This information is protected by AWS certified HTTPS protocol of the lambda service's function URL which is used as integration between slack and AWS. This is explained in *section 4.3.3*. When data is sent between AWS and GCP, the data shared are dialogflow agent specific parameters and user message. This data is protected by the dialogflow API's security which implements Google certified HTTPS protocol to protect the data at transit. Since no data is stored in the cloud, data protection at storage is not applicable for the slackbot application.

Although there is a high possibility of data breach, explicit cloud security measures are not taken into account for the slackbot application. This is because no confidential information like personal or health or financial data of employees are being used by the slackbot application. Therefore only a minimal security is provided to the application at this point. The application is dependent on in-built security of the cloud infrastructure against most common attacks.

CHAPTER

SIX

CONCLUSION

The goal of the slackbot application was to improve the accessibility to company wiki and increase the visibility of certain workflows by automating the business processes within the organisation. The current version of the slackbot application managed to stay active in the slack channel and listened to the keywords mentioned by the users. The slackbot application also responded to the specific keywords or synonyms mentioned by the users in the slack channel, by sending relevant links to the users. These links guide users to either the confluence web page containing information related to the keywords mentioned by the user or directly initiates the slack workflows created by the company. By testing the application end-to-end in the production environment and analysing the test results from *section 4.5.2*, it can be concluded that the slackbot application is working as intended. The slackbot application has successfully achieved the following results:

1. Improve accessibility to company wiki
2. Increase visibility of slack workflows
3. Increase productivity of employees by automating processes

6.1. Missed Requirements

These were some of the requirements from the stakeholders and they were successfully implemented in the slackbot application. Although the slackbot application was successful in achieving some of the requirements provided by the stakeholders, not all requirements were possible to be implemented for the master thesis of the slackbot application. The requirements that were not developed as part of the master thesis are:

1. Retrieve project details from confluence web page and send to the users in slack channel based on project number specified by the user in slack
2. Add a webpage as another frontend for the company admin to add new workflows directly to the slackbot application, over the existing 3 workflows.

6.2. Next Steps

The reason for these requirements not being completed in the master thesis of the slackbot application is because the project development for the slack application was paused on January 28, 2023 as seen in the Gantt chart, in *Figure 32*, in order to start writing the master thesis. This was a mutual decision taken between the stakeholders and the developer based on the time available and time taken to implement those 2 features. Therefore the mutual decision to build a working MVP of the slackbot application was confirmed for the master thesis. Although not all requirements were completed for the master thesis, the goal is to iterate further and improve the slackbot application by adding additional features which were provided as requirements by the stakeholders and also new exciting features that could benefit the company and its employees. The complete slackbot application with all the requirements provided by the stakeholders covered is shown in *Figure 50*. The additional features include:

1. Creating a web page frontend using HTML, CSS and Javascript with login credentials and hosting this web page in a new s3 bucket, *slack-web-page*.
2. The company admin can log into this webpage and add the new slack workflows to be added to the slackbot application's functionality. The webpage is connected to an API gateway, *slack-lambda-trig*, and when the admin saves the newly entered workflows along with its functionality, a POST method API call is made to the lambda function, *slack-update-wf*, is that is connected to the API gateway lambda function.
3. The lambda function, *slack-update-wf*, receives the new workflow and stores it in the dynamo db table, *slack-wf-db*, this dynamo db is created as a trigger for the lambda function, *slack-new-wf*, this lambda function gets invoked by the dynamo db as soon as a new item is stored in it.

CHAPTER

SEVEN

REFERENCES

Below is the list of references used in the thesis document. These references cite scientific papers, journals, documentations and articles from the internet. The style used for citing all references consistently throughout the document is the **APA** citation. Some documentations and articles do not have author names and/or year/date. For those references, “name of the organisation. (n.d.). title. link” - This style of APA referencing is followed.⁵² For Wikipedia references the reference format is “title of article. (Year, Month Date). In Wikipedia. link ”.⁵³

1. Webster, M. (2015). *Bridging the Document Disconnect in Sales, An IDC White Paper, Sponsored by Adobe*. Adobe Inc. <https://esign.adobe.com/rs/345-TTI-184/images/idc-bridging-the-document-disconnect-sales.pdf>
2. Schaubroeck, R., Holsztein Tarczewski, F., & Theunissen, R. (2016). *Making collaborations across functions a reality*. McKinsey & Company. <https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/making-collaboration-across-functions-a-reality>
3. Romao, M., Costa, J., & Costa, C. J. (2019). *Robotic process automation: A case study in the banking industry. In 2019 14th Iberian Conference on information systems and technologies (CISTI) (pp. 1-6)*. IEEE. https://www.researchgate.net/profile/Carlos-Costa/publication/334482701_Robotic_Process_Automation_A_Case_Study_in_the_Banking_Industry/links/6034f0c2299b1cc26e4c057/Robotic-Process-Automation-A-Case-Study-in-the-Banking-Industry.pdf
4. William, J. (2021). *How to bring a culture of automation in your organisation*. Forbes Inc. <https://www.forbes.com/sites/forbesbusinesscouncil/2021/02/10/how-to-bring-a-culture-of-automation-into-your-organization/?sh=17be76397c65>
5. Gaba, I. (2022). *What is the Slack app and how can it improve productivity?*. Simplilearn Solutions. <https://www.simplilearn.com/tutorials/slack-tutorial/what-is-slack-app-uses-to-improve-productivity>
6. Regan, S. (2016). *What is ChatOps? A guide to its evolution, adoption, and significance*. Atlassian Corporation. <https://www.atlassian.com/blog/software-teams/what-is-chatops-adoption-guide>
7. Linardatos, D. (2021). *TCO in private and public cloud*. Cambrian Technologies. <https://www.cambriantechnologies.com/blog/2021/02/14/comparing-teo-in-private-and-public-cloud/#:~:text=The%20public%20cloud%20is%20significantly,used%20components%20as%20managed%20services.>
8. Google Cloud. (2023). *Dialogflow ES Basics*. Google LLC. <https://cloud.google.com/dialogflow/es/docs/basics#intro>

⁵²

<https://www.scribbr.com/apa-examples/website/#:~:text=APA%20website%20citations%20usually%20include,time%2C%20add%20a%20retrieval%20date.>

⁵³ <https://aus.libguides.com/c.php?g=1131783&p=8260786>

9. Prickett Morgan, T. (2022). *Counting the costs of training Large Language Models*. The Next platform. Stackhouse Publishing Inc. <https://www.nextplatform.com/2022/12/01/counting-the-cost-of-training-large-language-models/>
10. Google Cloud. (2023). *Dialogflow Editions*. Google LLC. <https://cloud.google.com/dialogflow/docs/editions>
11. Slack api. (n.d.). *An introduction to the Slack platform*. Slack Technologies LLC. <https://api.slack.com/start/overview#what>
12. Slack api. (n.d.). *Sending Messages through Incoming Webhooks*. Slack Technologies LLC. <https://api.slack.com/messaging/webhooks>
13. Ponce, F., Márquez, G., & Astudillo, H. (2019). *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. https://www.researchgate.net/profile/Gaston-Marquez-2/publication/335716451_Migrating_from_monolithic_architecture_to_microservices_A_Rapid_Review/links/5d778d8292851cacdb2e23/Migrating-from-monolithic-architecture-to-microservices-A-Rapid-Review.pdf
14. Slack (Software). (2023, January 23). In Wikipedia. [https://en.wikipedia.org/wiki/Slack_\(software\)](https://en.wikipedia.org/wiki/Slack_(software))
15. Internet Relay Chat. (2023, February 3). In Wikipedia. https://en.wikipedia.org/wiki/Internet_Relay_Chat
16. Slack Technologies LLC. (n.d.). *Integrations*. <https://slack.com/integrations>
17. Slack Help Center. (n.d.). *Guide to apps in slack*. Slack Technologies LLC. <https://slack.com/help/articles/360001537467-Guide-to-apps-in-Slack>
18. Slack api. (n.d.). *Planning your slack app*. Slack Technologies LLC. <https://api.slack.com/start/planning>
19. Slack api. (n.d.). *Installing with OAuth*. Slack Technologies LLC. <https://api.slack.com/authentication/oauth-v2>
20. Parecki, A. (n.d.). *OAuth 2.0*. OAuth. <https://oauth.net/2/>
21. Google Cloud. (2023). *Dialogflow*. Google LLC. <https://cloud.google.com/dialogflow/docs>
22. Amazon Web Services. (2023). *Cloud Computing with AWS*. Amazon.com, Inc. https://aws.amazon.com/what-is-aws/?nc1=f_cc
23. IBM Cloud Education. (2021). *What are the benefits of chatOps?*. IBM Cloud. International Business Machines Corporation. <https://www.ibm.com/cloud/blog/benefits-of-chatops>
24. Confluence (software). (2022, November 24). In Wikipedia. [https://en.wikipedia.org/wiki/Confluence_\(software\)](https://en.wikipedia.org/wiki/Confluence_(software))
25. Jira Software. (2023). *What is Jira used for?*. Atlassian Corporation. <https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for>
26. Synopsis Glossary. (n.d.). *Software Development Life Cycle (SDLC)*. Synopsis, Inc. <https://www.synopsys.com/glossary/what-is-sdlc.html#:~:text=Definition.all%20customer%20expectations%20and%20demands>
27. Bassil, Y. (2012). *A simulation model for the waterfall software development life cycle*. arXiv preprint arXiv:1205.6904. https://d1wqtxts1xzle7.cloudfront.net/34194116/V115-0008-libre.pdf?1405312886=&response-content-disposition=inline%3B+filename%3DA_Comparative_Study_of_Different_Softwar.pdf&Expires=1675812986&Signature=A28N5G7jNyz-DxOGICiMOFc6aNLenY6eGlpYahDWVe2DVITNqinCsUw2ZEDVux0tyACsePH13XrPsEBBO6uCBiOSLL9RR3LYEXCuSunXYJAPflzh-DYrVKmA7pEb1CvoWpGB412wJxaaIKY83-ePuoaKZ1yJ3rWxw-DjBiGm9s~cvyUbiCvL7AH1PbLZoJd~4cypWimNf3Ag9HkkmbLv0cPtA~vZyXgoBUChPpETa4jsult-CchzZ1qtus-LhLOFgFHsNyvuWNCjxVR~bhYNK6tra2D~mDw6zipap5yKELQgy6tSJ1F46RJC0d8dD~YK9rDQLyCWvjgvvb~30xUh4Q_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
28. Adamek, Á., Farkas, K., & Szabó, G. (2022). *Cloud Native ChatOps Development*. HÍRKÖZLÉSI ÉS INFORMATIKAI TUDOMÁNYOS EGYESÜLET (hte). https://www.hte.hu/documents/10180/4734660/Adamek_Adam-Dolgozat.pdf

29. Lawton, G., & Tucci, L. (2022). *Business Process Automation (BPA)*. TechTarget CIO Definition. TechTarget.
<https://www.techtarget.com/searchcio/definition/business-process-automation>
30. Goasduff, L. (2022). *Top 10 Automation mistakes to avoid*. Gartner, Inc.
<https://www.gartner.com/en/articles/10-automation-mistakes-to-avoid>
31. Pratt, M.K. (2022). *workflow*. TechTarget CIO Definition. TechTarget.
<https://www.techtarget.com/searchcio/definition/workflow>
32. Zapier Editorial Team. (2021). *Zapier report: The 2021 state of business automation*. Zapier, Inc.
<https://zapier.com/blog/state-of-business-automation-2021/>
33. Pratt, M.K.(2022). *Workflow Automation*. TechTarget Content Management Definition. TechTarget.
<https://www.techtarget.com/searchcontentmanagement/definition/workflow-automation>
34. Slack api. (n.d.). *Using OAuth 2.0*. Slack Technologies LLC. <https://api.slack.com/legacy/oauth>
35. Slack Help Center. (n.d.). *Create more advanced workflows using webhooks*. Slack Technologies LLC.
<https://slack.com/help/articles/360041352714-Create-more-advanced-workflows-using-webhooks>
36. Slack Help Center. (n.d.). *Create more advanced workflows using webhooks*. Slack Technologies LLC.
<https://slack.com/help/articles/360041352714-Create-more-advanced-workflows-using-webhooks>
37. Slack Help Center. (n.d.). *Guide to workflow builder*. Slack Technologies LLC.
<https://slack.com/help/articles/360035692513-Guide-to-Workflow-Builder>
38. Munassar, N. M. A., & Govardhan, A. (2010). *A comparison between five models of software engineering*. International Journal of Computer Science Issues (IJCSI), 7(5), 94.
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3a4a2cb2328e2f416be0be012e5d580975943554#page=115>
39. Rastogi, V. (2015). *Software development life cycle models-comparison, consequences*. International Journal of Computer Science and Information Technologies, 6(1), 168-172.
https://d1wtxtslxle7.cloudfront.net/40003520/ijcsit2015060137-libre.pdf?1447530769=&response-content-disposition=inline%3B+filename%3D37_SDLC_comparison_consequences.pdf&Expires=1675318114&Signature=edv9WD-Trk5YpSgE3IqCg2251S4tAtgJLRqkXXKCanlp8rEZWMYLMtTe-3r439BMV1ReWUioEBXxoTm6Q~h3TsIRiY1MvK5nMzKYCGz~1tDkeA3WFLBDDY-Gy7s9yigiCBh7-1jDKrcARSu8604vx10-n9xNhiOdUXvbyurB7CR9cZbzW-dWrbIZawSp9B51fO0mxbsUewn7Pp2Aq-BP8XIIoFUOvrHi3YUch0rsM-o6XppkJY9TcQAoauUkznTICUrQe-yOPkZduzEvQVwDgmLr-uhF1YEi~09hP~vYPM-DYXF9bFHes8CIHQH49BTCZehdDGIDYTRNtEtKFQ_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
40. Larman, C., & Basili, V. R. (2003). *Iterative and incremental developments. a brief history*. Computer, 36(6), 47-56. <https://www.it.uu.se/edu/course/homepage/acsd/vt08/SE1.pdf>
41. Grady, J. O. (2010). *System requirements analysis*. Elsevier.
https://books.google.de/books?hl=en&lr=&id=FkpqAnHUNLYC&oi=fnd&pg=PP2&dq=requirements+analysis&ots=SjXCtvuO_&sig=KF64GWBAglec3N5CSL6ue1nftn4&redir_esc=y#v=onepage&q=requirements%20analysis&f=false
42. Simplilearn. (2023). *What is Requirements Analysis? Overview, Applications, and more*. Simplilearn Solutions.
<https://www.simplilearn.com/what-is-requirement-analysis-article>
43. Simplilearn. (2021). *What is a Gantt chart and why you need it for project management*. Simplilearn Solutions.
<https://www.simplilearn.com/what-is-a-gantt-chart-article>
44. Skusa, M. (2022). *How to write project milestones*. Filestage GmbH.
<https://filestage.io/blog/project-milestones/#:~:text=These%20two%20words%20can%20be,necessary%20to%20achieve%20a%20goal>
45. Conversation. (2023, January 16). In Wikipedia. <https://en.wikipedia.org/wiki/Conversation>
46. Turing, A. M. (2009). *Computing machinery and intelligence (pp. 23-65)*. Springer Netherlands.
https://watermark.silverchair.com/lix-236-433.pdf?token=AOECAHi208BE49Ooan9kkhW_Ercy7Dm3ZL_9Cf3qfKA485ysgAAAtEwgeLNBgkqhkiG9w0BBwaggeK-MIIcugIBADCCARMGCSqGSIb3DOEHATAeBgIghkgBZOMEAS4wEOOMqw-FZrdaCfQ9ewICAgeEQgIICH09V7MJuPK47sZ-LLdBDt81VPRohUAmO0wW83XicDn10H-eYj-tSiBUPZsPzRyVYajG1oVSFLaxci1

[RWUABD8OF6T0hP3O5SA1tBg-6Q_b9Xoq_OF4Dr3U_80IigOzhcvaSQ5AOHfPFvXI67-l_ZWqB8V0vAHlQeiKmSHo7-4Dg_xUdgSMOkvZRG05E9_pCOIRElJfBkAkNYymmFuX9Wf0vXrzTvyIJx8OGXFYhin17nQeedi51vxUdlt0Ni-Cxf0DBwKiUEme_nStL6-iWWf1NsnES-hOhk9rPP5sB40V6dwqEsCQC8Z8Wcd8Mm0XP0HO4cjQaWCxbj1J5snwINconeFVjXCgpEpJnZcy6D4N_vtI2_ivMx70V3CfOxgnSHxOHqgk1ryezTpxqGS9sfq6lKNGvXnYOY22410ce5Ta_ap-E_hi2UYXaPXw14rqDCOrYS3_xDvB6_GNiXfl0o2vXSGkwy-O2J8P1jt2XbWS661uoZRg6jqzMF2AO52RwJABXsbTevCgPmd4lmFt7B3DLTEpa5cGiVzImA-KeFi7Je_kZ4Nidmtk2DFjo314K8fGGwhJk2ViXrYsu1_rUJ3PB-gA55UtsiN85mZQ8mR1oNNLX6jPZgAJH4UA6WB9JwXrSQ2joSJp4_AWgyIW2829fGp3A909dEvGy2ANjY27ztdXySbbGAjwbwQh3xanik96111w-pYlhW5GUHhJfDNqQ8xk2hgKS-fWjAHl0X_X_6TdDfNFp33DKAOWLL7fxOIqZDEPj8n06Q4tywqYOpBGraQ1KquKkQps-rx3CQ80RMIBgCgCOOL94YP8yMCHna9bviQey_g57nP5z958DpUGoE1434neiGiGpf](#)

47. Oremus, W. (2022). *Google's AI passed a famous test — and showed how the test is broken*. Washington Post. <https://www.washingtonpost.com/technology/2022/06/17/google-ai-lamda-turing-test/>
48. Ram, A., Prasad, R., Khatri, C., Venkatesh, A., Gabriel, R., Liu, Q., ... & Pettigrew, A. (2018). *Conversational ai: The science behind the alexa prize*. arXiv preprint arXiv:1801.03604. <https://arxiv.org/pdf/1801.03604.pdf?fbclid=IwAR3dwqy06PSlmpFkyx2oTHCYt>
49. Muhammad, A. F., Susanto, D., Alimudin, A., Adila, F., Assidiqi, M. H., & Nabhan, S. (2020). *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
50. Slack api. (n.d.). *The Bolt family of SDKs*. Slack Technologies LLC. <https://slack.dev/bolt-python/concepts>
51. Slack api. (n.d.). *Getting started with Bolt for Python*. Slack Technologies LLC. <https://slack.dev/bolt-python/tutorial/getting-started>
52. Goel, A. (2021). *Part 4 Including DevOps in microservices migration*. Capgemini Engineering. Capgemini SE. <https://capgemini-engineering.com/us/en/insight/part-4-including-devops-in-the-microservices-migration/>
53. Gos, K., & Zabierowski, W. (2020). *The comparison of microservice and monolithic architecture*. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 150-153). IEEE. https://www.researchgate.net/profile/Wojciech-Zabierowski/publication/341956559_The_Comparison_of_Microservice_and_Monolithic_Architecture/links/5edf80fe299b1d20bdb24e2/The-Comparison-of-Microservice-and-Monolithic-Architecture.pdf
54. Dickson, B. (2022). *Democratising the hardware side of Large Language Models*. TechTalks. <https://bdtchttalks.com/2022/08/01/cerebras-large-language-models/>
55. Twigg, D. (2022). *What is a Webhook?*. Cyclr Systems Limited. <https://cyclr.com/blog/what-is-a-webhook>
56. Google Cloud. (2023). *Dialogflow ES Basics*. Google LLC. <https://cloud.google.com/dialogflow/es/docs/basics>
57. Google Cloud. (2022). *Dialogflow API*. Google LLC. <https://cloud.google.com/dialogflow/es/docs/reference/rest/v2-overview>
58. Google Cloud. (2023). *Service Accounts*. Google LLC. <https://cloud.google.com/iam/docs/service-accounts>
59. Google Cloud. (2023). *Create and Manage Service Account Keys*. Google LLC. <https://cloud.google.com/iam/docs/creating-managing-service-account-keys#creating>
60. Ronacher, A. (n.d.). *Welcome to Flask*. Pallets. <https://flask.palletsprojects.com/en/2.2.x/#>
61. Postman Learning Center. (2023). *Sending your first request*. Postman API Platform. Postman, Inc. <https://www.postman.com/home>
62. Gitlab community. (2022). *What are the benefits of microservices architecture?*. Gitlab, Inc. <https://about.gitlab.com/blog/2022/09/29/what-are-the-benefits-of-a-microservices-architecture/#:~:text=The%20benefit%20of%20a%20microservice,the%20resilience%20of%20the%20infrastructure>

63. Perry, D. E., & Wolf, A. L. (1992). *Foundations for the study of software architecture*. ACM SIGSOFT Software engineering notes, 17(4), 40-52. <https://apps.dtic.mil/sti/pdfs/ADA305470.pdf>
64. Google Cloud. *Google Cloud for Startups*. Google LLC . <https://cloud.google.com/solutions/startups>
65. Leite, L., Rocha, C., Kon, F., Milojevic, D., & Meirelles, P. (2019). *A survey of DevOps concepts and challenges*. ACM Computing Surveys (CSUR), 52(6), 1-35. <https://arxiv.org/pdf/1909.05409.pdf>
66. Bærbak Christensen, H. (2016). *Teaching DevOps and Cloud Computing Using a Cognitive Apprenticeship and Story-Telling Approach*. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16). ACM, 174–179. Code: A47. <https://archive.headconf.org/head17/wp-content/uploads/pdfs/5607.pdf>
67. Jeremy, H. (2022). *4 Microservice Examples: Amazon, Netflix, Uber, Etsy*. DreamFactory Software, Inc. <https://blog.dreamfactory.com/microservices-examples/#:~:text=In%202009%2C%20Netflix%20began%20the.servers%20as%20an%20independent%20microservice>
68. Jacobs, M., Petersen, T., & Kaim, E. (2022). *Use continuous integration*. Microsoft Learn. Microsoft Corporation. <https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration>
69. Jacobs, M., Petersen, T., & Kaim, E . (2022). *What is continuous delivery?*. Microsoft Learn. Microsoft Corporation. <https://learn.microsoft.com/en-us/devops/deliver/what-is-continuous-delivery>
70. Hardware Security Module. (2022, December 19). In Wikipedia. https://en.wikipedia.org/wiki/Hardware_security_module
71. Cryptographic Module Validation Program (CMVP). (2022). *Certificate #4177*. National Institute of Standards and Technology (NIST). <https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4177>
72. Galois/Counter Mode (GCM). (2023, January 10). In Wikipedia. https://en.wikipedia.org/wiki/Galois/Counter_Mode
73. Slack api. (n.d.). *Block Kit*. Slack Technologies LLC. <https://api.slack.com/block-kit>
74. Publish-subscribe pattern. (2023, October 31). In Wikipedia. https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
75. Amazon Web Services. (n.d.). *What is an Event-Driven Architecture?*. Amazon, Inc. <https://aws.amazon.com/event-driven-architecture/>
76. Cloud Native Security. (n.d.). *5 cloud application security best practices*. Synk Limited. <https://synk.io/learn/cloud-application-security/>
77. Slack security white paper. (2020). *Security at Slack*. Slack Technologies LLC. https://a.slack-edge.com/964df/marketing/downloads/security/Security_White_Paper_2020.pdf
78. Casalboni, A. (2022). *Announcing AWS Lambda Function URLs: Built-in HTTPS Endpoints for Single-Function Microservices*. AWS News Blog. Amazon, Inc. <https://aws.amazon.com/blogs/aws/announcing-aws-lambda-function-urls-built-in-https-endpoints-for-single-function-microservices/>
79. Lee, A. (2023). *What are Large Language Models used for?*. Nvidia blogs. Nvidia Corporation. <https://blogs.nvidia.com/blog/2023/01/26/what-are-large-language-models-used-for/>
80. Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P. S., ... & Gabriel, I. (2021). Ethical and social risks of harm from language models. arXiv preprint arXiv:2112.04359. <https://arxiv.org/pdf/2112.04359.pdf>
81. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In Proceedings of the 2021 ACM conference on fairness, accountability, and transparency (pp. 610-623). <https://arxiv.org/pdf/2112.04359.pdf>
82. Cloudflare learning. (n.d.). *What is cloud security?*. Cloudflare, Inc. <https://www.cloudflare.com/learning/cloud/what-is-cloud-security/>

Declaration of Independence

I hereby certify that I have written this thesis independently and only using the specified sources and aids.
The work has not been submitted to any other examination authority in the same or a similar form.

A handwritten signature in blue ink, consisting of a large, stylized 'R' followed by a smaller 'S' and a horizontal line.

Place, Date, Name

Berlin, February 14,. 2023, Rishi Srinivasan Kanaka Sabapathy