Otto-von-Guericke University Magdeburg

Faculty Of Electrical and Information Technology
IIKT-Mobile Dialog Systems

Digital Engineering Project

Developing and evaluating a Chatbot

Authors:
Pritha Ghosal
Jeffy Elson
Preetam Vinod Naik

October 24, 2022

Advisers:

Supervisor	Supervisor
Jun.-Prof. Dr. Ingo Siegert,	Matthias Busch, M.Sc.
IIKT-Mobile Dialog Systems	IIKT-Mobile Dialog Systems
Otto-von-Guericke University	Otto-von-Guericke University
Universitätsplatz 2	Universitätsplatz 2
39106 Magdeburg, Germany	39106 Magdeburg, Germany

Jeffy Elson
Pritha Ghosal
Preetam Vinod Naik
Developing and evaluating a Chatbot
Digital Engineering Project, Otto-von-Guericke University
Magdeburg, 2022.

Contents

Abstract

This project aims at providing a interactive surface for the international students via a chatbot. The one thing that every international student faces being in a different country is to find doctors and their details being new to a place. We tried to solve this problem by developing a simple chatbot that can find user a doctor and all the necessary information regarding it. The chatbot was developed using Google Dialogflow which is used to create intents for training using Natural Language Processing (NLP). We used Flask framework using Python to train the bot and fulfill responses based on the intents that get matched when a user expresses something over the chatbot. We also integrated Firestore database to fetch information about the doctors and to store information about users chat history. In order, for the Dialogflow and the application built using flask to work, we hosted it over PythonAnywhere and finally published on Telegram Platform.

While building the bot we took into consideration few qualitative measures like usability, attractiveness and acceptance tests. For designing the intents, we made flow charts to get a sense of what was needed so that we could structure the flow of the chatbot. This made the chatbot efficient and easy to develop. After the deployment, we conducted a evaluation of the bot using a Google form to understand users perspective and improvising the chatbots functionality and future research. Due to its functionality and structure of data, the final product received generally positive reviews from users.

Chatbots will be your new best friend.

— Christine Crandell (Forbes)

Acknowledgements

We would like express our gratitude and appreciation to all those who gave us the possibility to complete this report. Special thanks is due to our supervisor Jun.-Prof. Dr. Ingo Siegert and Matthias Busch, M.Sc. whose help, stimulating suggestions and encouragement helped us in all time of the development process and in writing this report. We also sincerely thanks for the time spent proofreading and correcting our many mistakes.

Then we would like to thank our friends who have helped us with their valuable suggestions and guidance which have been helpful in various phases of the completion of the project.

1 Introduction

1.1 Introduction

A dialogue system is a computer application that interacts naturally with a human user. An interface is provided through the dialogue system between a user and an application running on a computer, making it possible to interact with the application fairly unforced manner. The System might be a CUI, It can be utilized in GUI, VUI, multimodel, etc. devices like phones, PDAs, automobiles, robots, and website browsers.

Chatbots represent the most basic types of dialogue systems. They are programs that may have prolonged interactions with the aim of replicating the unstructured dialogues or "chats" prevalent in spontaneous human-human interaction. [2]

1.2 Motivation

"Why build a chatbot? and for whom?"

These two questions help us establish the main ground for our project i.e., objective and the target user. We identified potential issues faced by international students in Germany. Some of them were :

- Appointments at the foreigner's office.
- Getting appointments with doctors.
- Appointments for City Registration.
- Course-related difficulties.

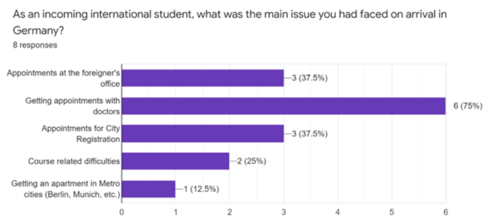


Figure 1.1: Survey for chatbot use case

As an outcome of the survey, we narrowed down that our peers, like us, had faced issues with finding doctors in Germany especially during the initial days. Some of their difficulties were :

- Difficulties with getting an appointment with a specialist.
- Figure out how health insurance coverage works here.
- Linguistic barrier with the medical professionals.
- Lack of emergency contacts.

Getting medical attention starts to seem like a hassle when we are still familiarizing ourselves with the systematic differences. In such bewildering situations, Chatbots can work as a one-stop solution for all our queries. The reasons are -

- There is no scheduled availability, Chatbots are accessible 24*7
- Information is always readily available
- Through feedback and data analysis the services and responses improve
- Repetitive processes like booking appointments etc can be automated, hence reducing human efforts

and many more.

1.3 Objective of the Research

Among some intriguing questions raised by the investigation and review of relevant material, the following question abides by the "FINER"[9] criteria and thus poses as our research question- ***"How do we accumulate all the necessary information on one platform to find basic medical attention in Germany for international students?"***

Systems that are task-oriented are created to aid users in completing tasks as quickly as possible. The talks are notable for adhering to a precisely specified framework that is a result of the domain. Mixed initiative is used in the dialogues; both the user and the system may take the initiative.

Our goal was to build a Rule-based bot as these bots operate according to a predetermined conversation flow that enables the bot to proceed rationally in response to human inputs and selections. Users go through the conversation flow by selecting items from menus, carousels, and buttons, and by providing answers to questions.

Our chatbot is implemented using Dialogflow and Python framework (which are extensively explained in chapter 2. Tools & Framework) aiming to handle the resource-intensive tasks of data collection, manipulation, and periodic analysis. Starting from the data gathering the chatbot diverges into different functional categories each of which deals with one information need of the user. So far 'MediMate' can successfully complete the following tasks -

1. Find specialist doctors nearby
2. Provide information for booking appointments.
   - Name
   - Address
   - Operational hours
   - Language specification
   - Contact information
   - Navigation from nearby landmarks
3. Providing nearby pharmacy emergency contacts
4. Providing nearby Klinikum emergency Room contacts
5. Keeping notes about the previous appointment

2
Evaluation Proposal

Dialogue system evaluation is a difficult endeavor and a hot topic of study. Since Chatbot technologies are still growing and developing, there is no standardized criterion for evaluating chatbots. However, an accurate evaluation might be one of the most important and essential parts of developing a chatbot.

The structured form of the interaction is the foundation for task-oriented dialogue system evaluation. Task success and dialogue efficiency are the two key criteria that are examined because they have been proven to characterize the dialogue's quality[11].

After going through a couple of papers we decided that the focus of our evaluation would be mainly on the following two categories,

- User Experience
- User Satisfaction

2.0.1 User Experience

User experience research is crucial to user-centered[4] design. UX research provides comments and insights that aid in creating customized product experiences for target users. There are numerous UX research techniques that can be implemented to get data successfully. The User Experience Questionnaire (UEQ), another metric, provides a full evaluation of the UX. The Chatbot Usability Questionnaire is the last metric (CUQ). 16 items with both positive and negative implications for the system were distributed to participants. A measure of "Strongly Agree" and "Strongly Disagree" was used to rate the questions on a 5-point scale. [3]

2.0.2 User Satisfaction

In this case, it is assumed that user satisfaction, which can be gauged by surveys, may be used to estimate how usable and novel a system is. These methods seek to mimic human judgments by building models that assign the same scores as human judges. The dialogue system is used for a human evaluation that takes place first. The dialogue system is then evaluated using questionnaires. Finally, a model based on objectively quantifiable properties is fitted using the ratings as target labels (e.g. task success rate) [17]

Task Success Rate

The dialogue's objective or task can be divided into the following two parts:

- A group of restrictions that specify the desired information to be retrieved.
- A collection of requests that specify the data the user seeks.

The task-success rate assesses the degree to which the dialogue system satisfies the information needs to be established by the user's objectives.

2.1 Approach

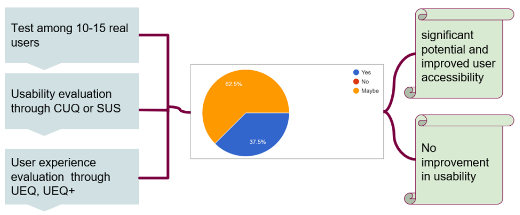

Test among 10-15 real users
Usability evaluation through CUQ or SUS
User experience evaluation through UEQ, UEQ+
62.5%
37.5%
Yes
No
Maybe
significant potential and improved user accessibility
No improvement in usability

Figure 2.1: Initial evaluation Idea

In Figure:2.1 we decided to choose around 15 people to conduct the evaluation with. We also chose to perform *Usability Evaluation* with **CUQ(Chatbot Evaluation Questionnaire)**, **System Usability Scale (SUS)** and *User Experience evaluation* with the standard **UEQ(User Experience Questionnaire)**.

For the participants we focused on people with the following characteristic:

- International students
- Students who do not speak the local language very well
- Students and our peers who had faced medical emergencies

In order to maintain the universal standard of process evaluation we eventually decided to hold to **UEQ(User Experience Questionnaire)** as our tool for explaining the analytics of data.

2.1.1 UEQ (User Experience Questionnaire)

[14] UEQ explains how well the system performs in comparison to competing dialog systems and to what extent it meets user expectations. This questionnaire's rating scales cover every aspect of the user experience. Both traditional user experience elements *(originality, excitement)* and usability elements *(efficiency, perspicuity, dependability)* are measured within 26 items. Attractiveness is solely a valence dimension. While excitement and novelty are *hedonic quality* elements (not goal-directed), perspicuity, efficiency, and dependability are *pragmatic quality* features (goal-directed)

The items take the form of a semantic difference, with two terms with opposing meanings standing in for each item. The terms are randomly arranged for each item so that for a scale, half the items start with the positive term and the other half with the negative term. To eliminate the well-known *central tendency bias* for these kinds of items, we employ a seven-stage scale, which determines a final score representing the chat's *quality of experience (QoE)*. The question structure is shown in the following Figure:2.2 -

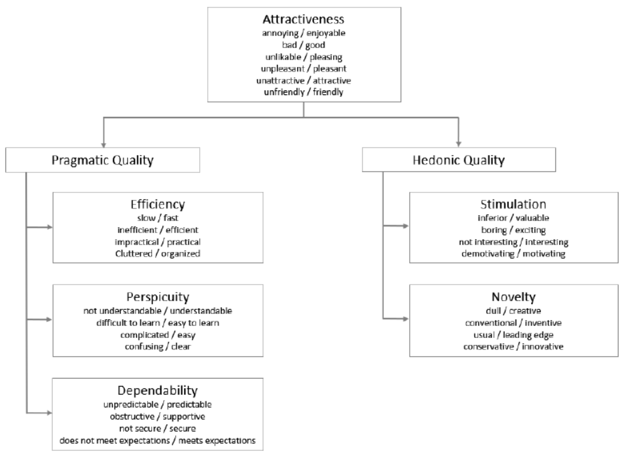


Figure 1: Assumed scale structure of the UEQ.

Figure 2.2: Questionnaire scale structure

An example of an item is:

attractive O O O O O O O unattractive

Scales for the goods range from -3 to +3. So, the most negative response is represented by -3, the neutral response by 0, and the most positive response by 3.

The gathered data is evaluated through **scale means** and **benchmark comparison**. These analytical results help in understanding whether the product meets the standard requirements for user experience or not.

3 Framework & Integration

After having a clear picture on the objective of our chatbot, we had to decide on which technological solutions would be the most appropriate for our usecase. We explored some state of the art technologies such as **JOVO** and **RASA**. Both are open source platforms but have their own limitations.

JOVO mainly focuses on building voice and chat platforms. They mainly help in handling the states of the chat. For our NLU, we considered Google DialogFlow as our first option. Due to the advancement of JOVO from version 3 to version 4, this integration seemed impossible which was even confirmed by their community.

We also explored RASA which has two variants - Core and NLU. For the NLU, we could combine the most suitable Tokenizers, Featurizers and Classifiers. This seemed to be a very good option but due to the complexity and time constraint of our use case we decided not to proceed further . Another factor was that RASA's architecture and knowledge requirement was vast.

Considering all factors, we decided to use Python to design our backend and Google DialogFlow ES as our NLU. A more detailed explanation of the components of all the frameworks are explained in this chapter.

3.1 DialogFlow

Google enables customers to create their own AI-powered Conversational Interface for free. Dialogflow [10] aims to engage the user in intelligent conversations. It makes an effort to comprehend the user's purpose using the training phrases the user provides and automatically reacts to queries through Text-based or voice conversations.

Google Dialogflow has two versions, Dialogflow ES and Dialogflow CX. For our project we went with Dialogflow ES which is free to use and suitable for small case or medium scale applications. Dialogflow ES seemed advantageous to our use case for the following reasons :

- easy to use and understand
- Uses NLP
- single click integrations with many messaging applications

3.1.1 Agents

When a user queries or interacts with the application, the Dialoglfow Agent's job is to translate the text of the end-user into structured data for the application to understand.

3.1.2 Intents and Entities

An Intent helps the dialogflow decide on the intention of the end users query. When an end user queries something, the dialogflow matches these end user queries with the most suitable or best matching intent. These intent define what needs to be done when a end user query is matched to it. This is also called as **Intent Classification**.

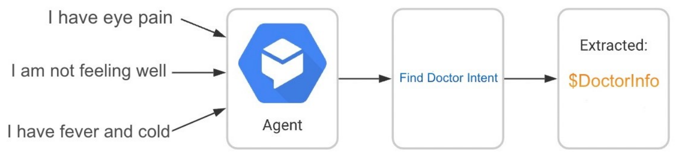


Figure 3.1: Intent Detection

A intent has following:

1. **Training Phrases**: Training phrases are the possible phrases that an end user might use. If any of the entities match with the user expressions that particular intent gets invoked.

2. **Actions and Parameters**: Parameters are the values that can be asked at run time to the user. Actions then match the intent and various actions can be triggered on the basis of that which are defined.

3. **Responses** : Responses are the text or quick replies which is returned when a particular intent is matched.

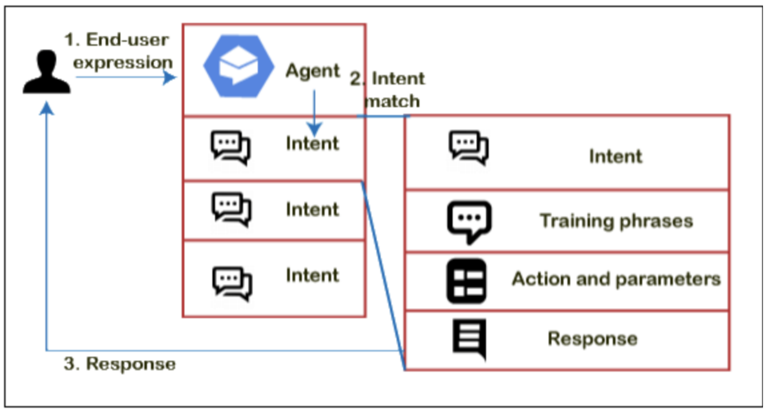


Figure 3.2: Flow for intent matching and response [10]

3.1.3 Fulfillment

By default, the agent matches with the intent and gives out a static response defined in the intent. In order to have a dynamic response, we can either integrate it with messaging platforms or through an API call by enabling a webhook call toggle.

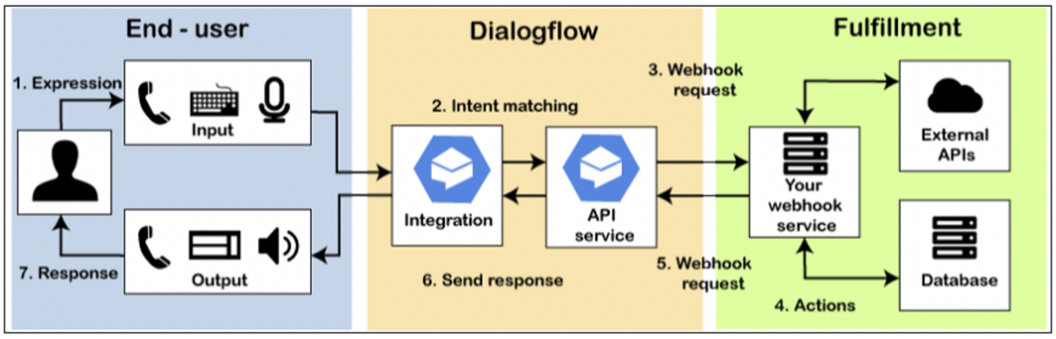
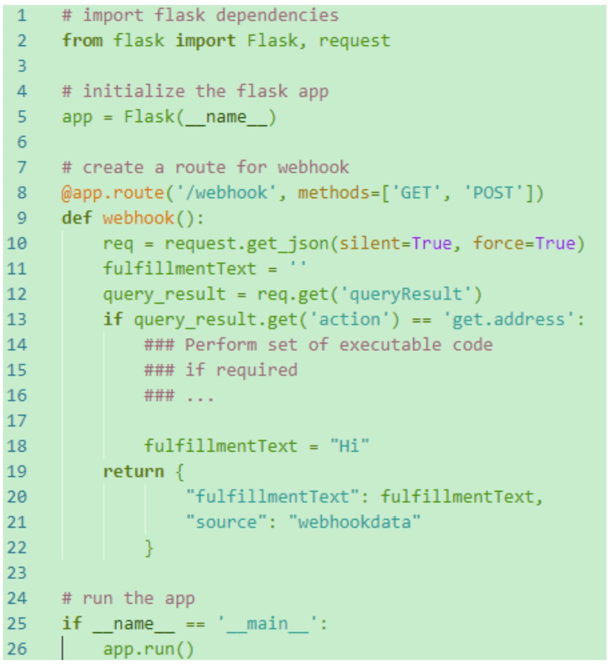


Figure 3.3: Fulfillment processing [7]

3.2 Python Flask

We use Python as our language to code our webhook response logic and to execute it we use a python framework called Flask[16] . We need flask to create a server for it to be integrated with Dialogflow



```python
# import flask dependencies
from flask import Flask, request

# initialize the flask app
app = Flask(__name__)

# create a route for webhook
@app.route('/webhook', methods=['GET', 'POST'])
def webhook():
    req = request.get_json(silent=True, force=True)
    fulfillmentText = ''
    query_result = req.get('queryResult')
    if query_result.get('action') == 'get.address':
        ### Perform set of executable code
        ### if required
        ### ...

        fulfillmentText = "Hi"
    return {
            "fulfillmentText": fulfillmentText,
            "source": "webhookdata"
        }

# run the app
if __name__ == '__main__':
    app.run()
```

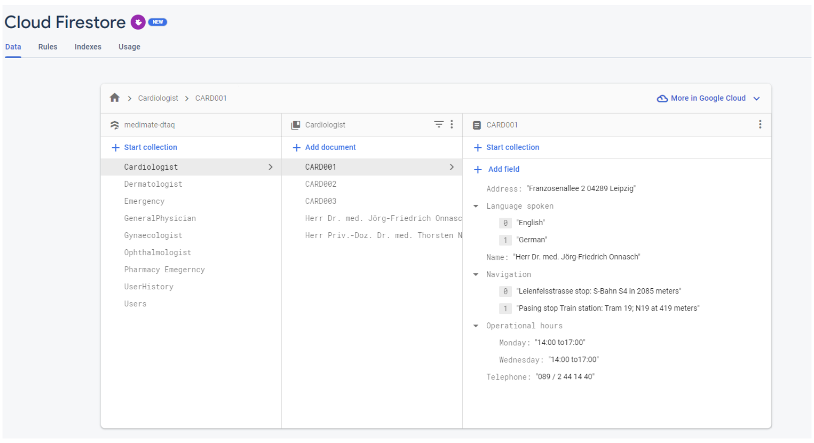
Figure 3.4: Flask app [8]

3.2.1 Ngrok

Ngrok [12] is a cross platform development tool used to expose our local development server to the internet. Ngrok is used to host our Python flask app on to a server. This helps us in integrating our python flask app with Dialogflow.

3.3 Firestore database

Firestore[5] Database is a No SQL cloud database. It is very flexible and scalable. It syncs our data realtime and used in client and server side development. We use firestore database for our chatbot for seamless user experiences and better responses. We can store data into documents which have mappings to the specific values. All the documents are stored in a collection used to organise data.



```
Cardiologist              CARD001
Dermatologist             CARD002
Emergency                 CARD003
GeneralPhysician          Herr Dr. med. Jörg-Friedrich Onnasc
Gynaecologist             Herr Priv.-Doz. Dr. med. Thorsten N
Ophthalmologist
Pharmacy Emegerncy
UserHistory
Users
```

```
Address: "Franzosenallee 2 04289 Leipzig"
▾ Language spoken
    0 "English"
    1 "German"
Name: "Herr Dr. med. Jörg-Friedrich Onnasch"
▾ Navigation
    0 "Leienfelsstrasse stop: S-Bahn S4 in 2085 meters"
    1 "Pasing stop Train station: Tram 19; N19 at 419 meters"
▾ Operational hours
    Monday: "14:00 to17:00"
    Wednesday: "14:00 to17:00"
Telephone: "089 / 2 44 14 40"
```

Figure 3.5: Firestore Database

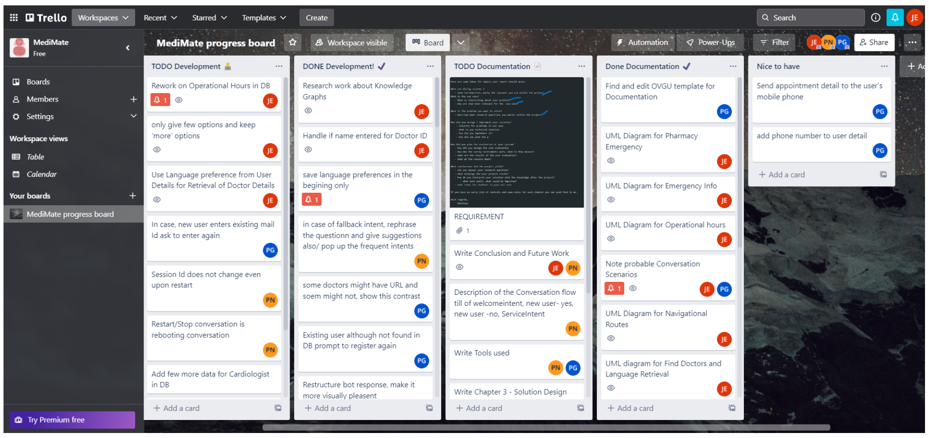
3.4 Telegram

Telegram [15] is a globally accessible, free messaging app which can easily be integrated with Dialogflow using the integration options. Dialogflow Telegram integration allows us to create chatbots with natural language processing abilities with ease and to use rich response messages like Image response, Card response, Quick Reply response and many more. Telegram

3.5 Project Management

The Medimate bot development is a software project and our group consists of three members. Therefore, to manage our project efficiently and divide the work equally amongst ourselves, we used Trello - a Kanban board. To help us be updated with the code, we used Gitlab as our Version Control.

3.5.1 Trello



TODO Development
- Rework on Operational Hours in DB — JE
- only give few options and keep 'more' options — JE
- Use Language preference from User Details for Retrieval of Doctor Details — JE
- In case, new user enters existing mail Id ask to enter again — PG
- Session Id does not change even upon restart — PN
- Restart/Stop conversation is rebooting conversation — PN
- Add few more data for Cardiologist in DB

DONE Development!
- Research work about Knowledge Graphs — JE
- Handle if name entered for Doctor ID — JE
- save language preferences in the begining only — PG
- in case of fallback intent, rephrase the questionn and give suggestions also/ pop up the frequent intents — PN
- some doctors might have URL and soem might not, show this contrast — PG
- Existing user although not found in DB prompt to register again — PG
- Restructure bot response, make it more visually pleasent

TODO Documentation
- REQUIREMENT
- Write Conclusion and Future Work — JE PN
- Description of the Conversation flow till of welcomeintent, new user- yes, new user -no, ServiceIntent — PN
- Write Tools used — PN PG
- Write Chapter 3 - Solution Design

Done Documentation
- Find and edit OVGU template for Documentation — PG
- UML Diagram for Pharmacy Emergency — JE
- UML Diagram for Emergency Info — JE
- UML Diagram for Operational hours — JE
- Note probable Conversation Scenarios — JE PG
- UML Diagram for Navigational Routes — JE
- UML diagram for Find Doctors and Language Retrieval — JE

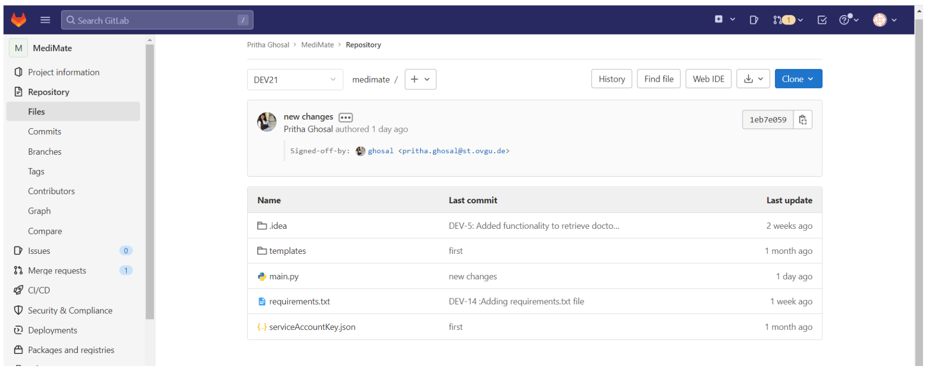
Nice to have
- Send appointment detail to the user's mobile phone — PG
- add phone number to user detail — PG

Figure 3.6: Medimate Trello Progress board

Trello is a quick and easy way to create a digital Kanban board. Setup requires just a few clicks to create digital lists that represent the phases of your Kanban process on a whiteboard view that your entire team can access and manage. [13]

Lists were created as follows in the board such that each contains tasks in the form of Cards with members assigned to the tasks:

3.5.2 Version Control - GitLab



new changes
Pritha Ghosal authored 1 day ago
1eb7e059

```
Signed-off-by: ghosal <pritha.ghosal@st.ovgu.de>
```

Name	Last commit	Last update
.idea	DEV-5: Added functionality to retrieve docto...	2 weeks ago
templates	first	1 month ago
main.py	new changes	1 day ago
requirements.txt	DEV-14 :Adding requirements.txt file	1 week ago
serviceAccountKey.json	first	1 month ago

Figure 3.7: Medimate Gitlab repository

In GitLab, one can create projects to host their codebase. One can also use projects to track issues, schedule work, collaborate on code, and continuously build, test, and deploy their application with built-in CI/CD. We also use Git branching strategies to work collaboratively on our code. In this project, we used Gitlab provided by the university - https://code.ovgu.de/.

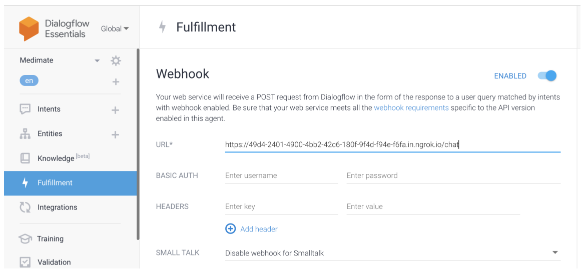
3.6 Integration of components

The integration of the technologies and design decisions that were made to create our ChatBot are the main topics of this chapter.

3.6.1 Integrating Dialogflow and Python

Python code and Dialogflow are integrated by a webhook. The Python code upon hosting on the internet can be integrated to Dialogflow using the fullfiment option in it.

For the purpose of hosting the python code online, we used ngrok. This URL which we get can be used to integrate our code with Dialogflow by enabling the webhook option and using the URL that was generated under the URL option as shown below.



Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

URL* https://49d4-2401-4900-4bb2-42c6-180f-9f4d-f94e-f6fa.in.ngrok.io/chat

BASIC AUTH Enter username Enter password

HEADERS Enter key Enter value

SMALL TALK Disable webhook for Smalltalk

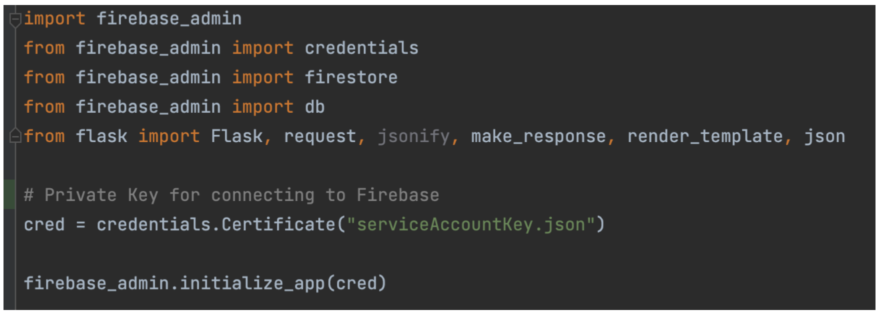
Figure 3.8: Using hosted URL in fulfillment

The code hosted online will receive a POST request from the Dialogflow. These POST requests are handled inside the webhooks that are written inside our python code. Webhooks according to the query and intent matching give out appropriate fulfillment response for our chatbot provided that the webhook is enabled for a intent in dialogflow.

3.6.2 Integrating Python and Firestore

To connect to Google Firestore, we need to install a python package called *“firebase-admin"*. This can be installed like any other python package using pip

To connect to Firestore, Firebase first performs authentication and for this we can generate a private key using firebase console which we include in our code.The credentials library is used to connect to Firestore using the keys file. The app is then initialized with a service account and the admin privileges are granted.



```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from firebase_admin import db
from flask import Flask, request, jsonify, make_response, render_template, json

# Private Key for connecting to Firebase
cred = credentials.Certificate("serviceAccountKey.json")

firebase_admin.initialize_app(cred)
```

Figure 3.9: Assigning Private Key for connection

3.6.3 Integrating Telegram and Dialogflow

In order to connect telegram with Dialogflow, we first need to configure telegram by creating a bot in telegram as given in with the help of BotFather that creates, maintains & modifies all the bot. We connected our telegram bot MediMate by adding the access token of Telegram in Dialogflow and Voila! Our MediMate was live on Telegram.

3.6.4 Hosting code on PythonAnywhere

For hosting the code over pythonanywhere, we need to create a account in pythonanywhere and upload the code from our local to the platform and create a web app in pythonanywhere and configure the web app to run the code that was added to the platform. We also had to turn on the HTTPS module in it as Dialogflow doesnot allow any URl with no SSL certificates.

3.7 Architecture Diagram

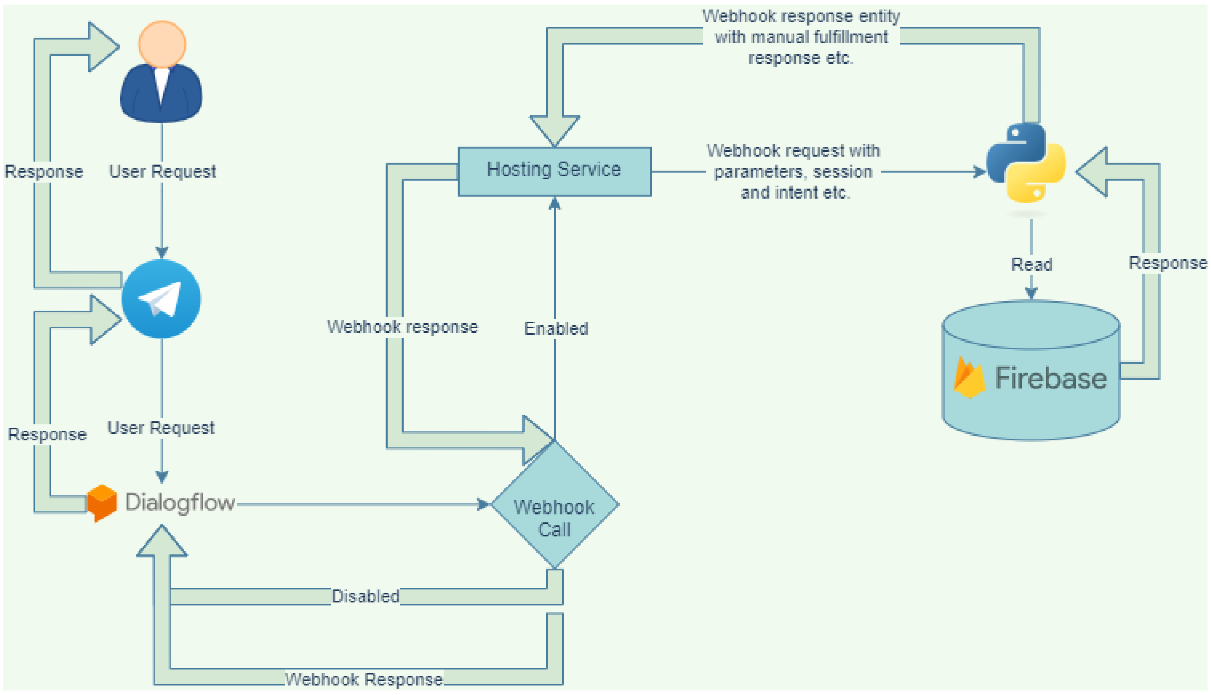


Figure 3.10: System architecture

4
Intent Design

In order to make our chatbot perform various functionalities and detect the user's request, we have designed intents. We have explained in detail, the design need and flow of various intents in this chapter.

4.1 Medimate Welcome Intent

This intent is designed to welcome the user and for a way to recognise if the user has used the chatbot before or not. The conversation begins, upon any greeting or pre-defined command from the user side. First, the bot makes sure to introduce itself by saying *"Hi! I am MediMate. Your buddy for medical assistance."* and set about the stage of collecting user-detail *"Nice to meet you! Let's find you help ASAP!"*. Next, the user is asked to specify whether he is a 'New User' or 'Existing User' (who is already registered in the bot) through the telegram quick suggestions button. Based off of the user's selection the follow-up intents (New User - Yes, New User - No) get spawned.

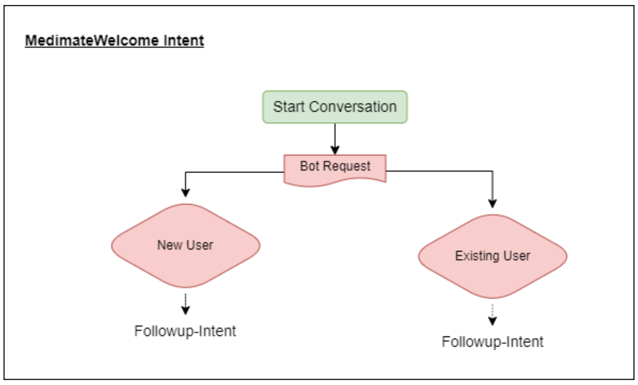


Figure 4.1: UML for Medimate Welcome Intent

4.2 New User - Yes Intent

This is the intent that works for registering the user into the database and personalizing their experience further. Once the user classifies themselves as **'New User'** the bot asks them to enter the necessary details one by one. The name and email-id are requested from the user consecutively and fetched from the request JSON from query parameters **['user_Name']** & **['user_Email']** . We have kept the email-id as a unique key for each user hence **checkUserExistenceByEmail()** function performs a search into the **"Users"** DB to find whether this email-id already exists or not. If it founds the existence of the email, the bot would spawn a response such as *"Looks like this email id is already registered with us, please try a different email Id"*.

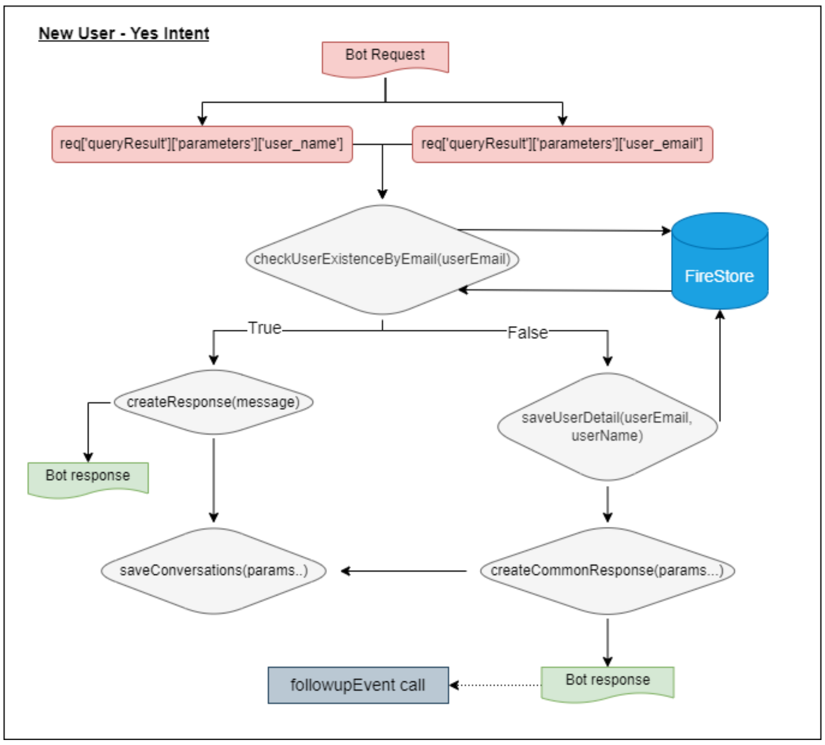

New User - Yes Intent
Bot Request
req['queryResult']['parameters']['user_name']
req['queryResult']['parameters']['user_email']
checkUserExistenceByEmail(userEmail)
FireStore
True
False
createResponse(message)
Bot response
saveUserDetail(userEmail, userName)
saveConversations(params..)
createCommonResponse(params...)
followupEvent call
Bot response

Figure 4.2: UML for New User - Yes Intent

On the contrary when the email-id is unique, the bot performs a number of tasks as follows -

- it would save the new user detail first in the **"Users"** DB.

- It will greet the user and will notify them about their newly created user Id for ease of later access.
- It will start storing the conversations in the **"UserHistory"** DB as well, with respect to the session id and user Id.
- Lastly it would spawn the following **askLanguagePreference** intent by asking the user about their linguistic preference if any. *"Is there any language that you would like your Medical expert to speak in?"*.

4.3 askLanguage & enterLanguagePreference intent

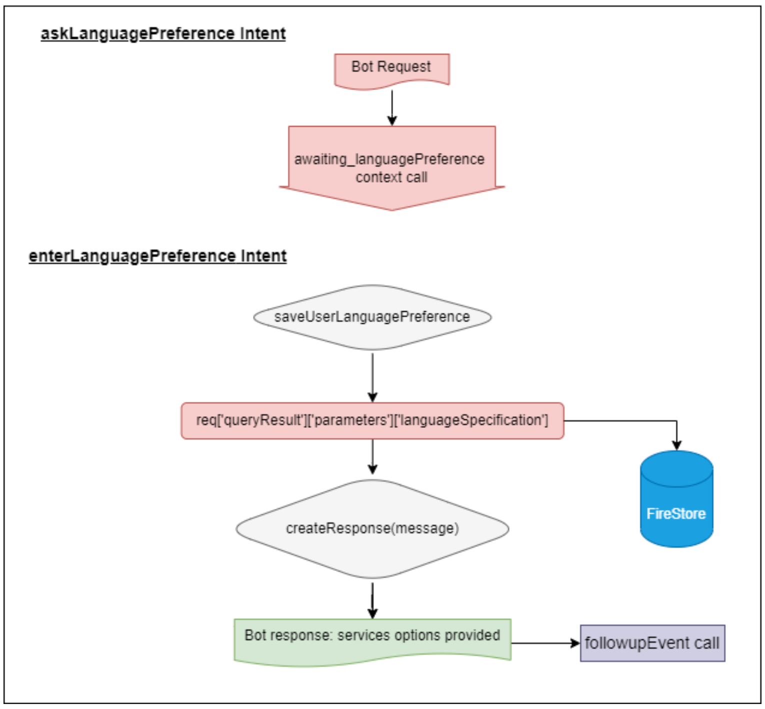

askLanguagePreference Intent
Bot Request
awaiting_languagePreference context call
enterLanguagePreference Intent
saveUserLanguagePreference
req['queryResult']['parameters']['languageSpecification']
FireStore
createResponse(message)
Bot response: services options provided
followupEvent call

Figure 4.3: UML for askLanguage & enterLanguagePreference intent

This intent was designed to have a usecase where a user can set their language preferences in the system as medimate primarly focuses on international students and language is mostly a key parameter that accounts while finding a doctor. These intents primarily deal with collecting the user's language propensity. **saveUserLanguagePreference()** function takes the query parameter **['languageSpecification']** from user's response and stores it under the **’Users’** DB. Further on it proceeds with the options of the services offered by the bot. In case of an existing user they are also provided with option of **changing their preferred language**, in that case these intents are called as well.

4.4 New User - No and getUserId Intent

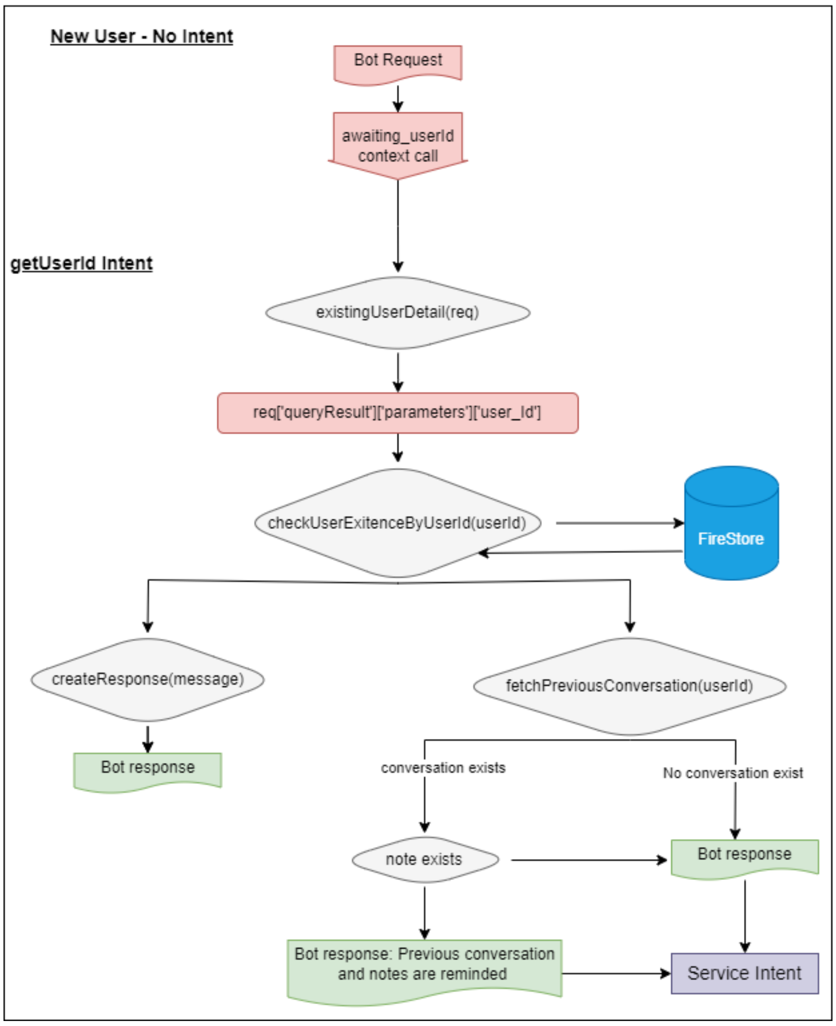

New User - No Intent
Bot Request
awaiting_userId context call
getUserId Intent
existingUserDetail(req)
req['queryResult']['parameters']['user_Id']
checkUserExitenceByUserId(userId)
FireStore
createResponse(message)
Bot response
fetchPreviousConversation(userId)
conversation exists
No conversation exist
note exists
Bot response
Bot response: Previous conversation and notes are reminded
Service Intent

Figure 4.4: UML for New User - No & getUserId Intent

Once the user specifies that they are an existing user the **"New User - No"** intent is spawned and the bot asks for their user Id subsequently. The data value is fetched from the query parameter **"user_Id"**. Following this the bot performs the following tasks one by one -

- The function **checkUserExistenceByUresId()** returns true is user is found in the **"Users"** DB and false otherwise.
- As the name suggests the **fetchPreviousConversation()** function fetches the last interaction details along with saved **Notes** if any. If there is nothing saved under Notes it simply asks the user *"Do you want me to create a note about the appointment?"*
- The functions **createCommonResponse()** & **createResponse()** structures the bot responses accordingly and send to the user through Telegram interface.
- Lastly it would spawn the following intent by providing the options of the services offered by the bot.

4.5 Find Doctors and Language Specification Intent

This intent was designed to have a way for the users to let the chatbot know what they want or what symptoms do they have. This gives us a way to process the user requirement and provide them with necessary information. The Find Doctors Intent is the most important intent in the entire chat bot. It helps the user to find doctors according to their required specification or symptoms. A Telegram suggestion button "Find Doctors" helps us to navigate to this intent. Alternatively, the user can post their query, for example : *"I would like to book an appointment with a Gynaecologist"*. This intent also suggests the required specialization of doctor with respect to the user's symptoms. This may not be very accurate as we would suggest the user to specifically enter the area of specialization.

The request of the user calls the **getListofDoctors()** function. The entity being extracted as specialization is stored in a temporary variable. From the user information collected at the start, the language of preference is fetched. This is beneficial to international students who are not familiar with German. After passing these two information, a call from **getListofDoctors()** is made to the **processLanguage()** function. A query is executed in Firestore where only doctors who match the specialization and language are retrieved. The Bot Response would be a list of Doctors from the specialization and preferred language along with their Doctor Identification numbers. If the user selects *"It's fine"* during storing the initial user data, then all doctors from the chosen specialization are given irrespective of the language.

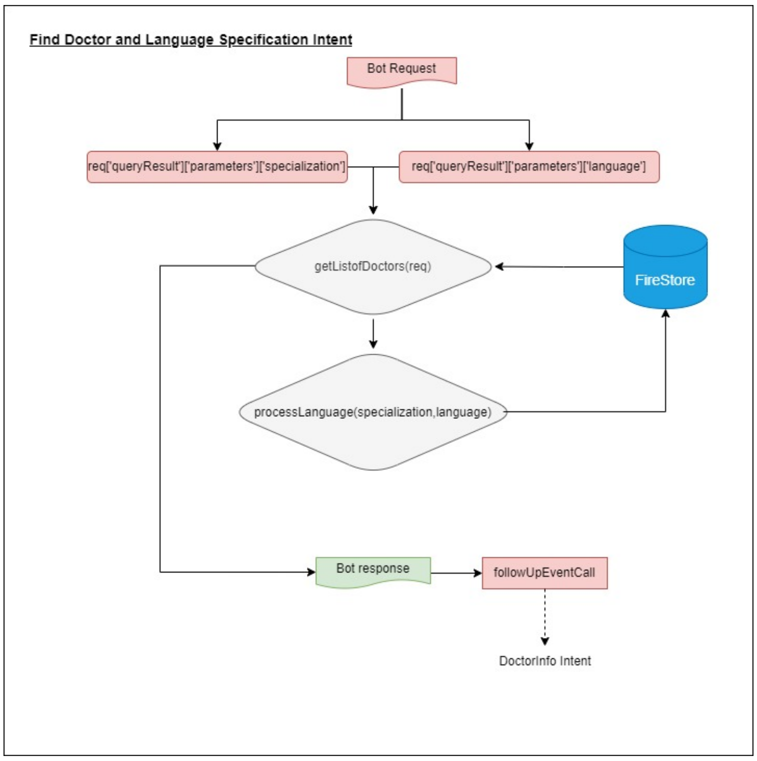

Find Doctor and Language Specification Intent
Bot Request
req['queryResult']['parameters']['specialization']
req['queryResult']['parameters']['language']
getListofDoctors(req)
FireStore
processLanguage(specialization,language)
Bot response
followUpEventCall
DoctorInfo Intent

Figure 4.5: UML for Find Doctors and Language Specification Intent Flow

Phrase	**Entity extracted**	**Intent**
Find Doctor	Doctor	finddoctor
I need an appointment with a gynaecologist	gynaecologist	finddoctor
I have stomach pain	pain	finddoctor
I need an appointment with an opthalmologist who speaks English	opthalmologist ; English	finddoctor

Table 4.1: Example findDoctors Intent Training Phrases and Entities

4.6 Doctor Information Intent

The user gets a list of Doctor names and Doctor IDs where he/she is expected to choose from one of the mentioned IDs for more information. When the user enters a Doctor ID, the Doctor ID entity is extracted. This has been defined in DialogFlow using Regular Expressions. The specialization of the doctor is stored in Find Doctors Intent in a list. This is passed on to the **provideDoctorDetails()** function along with the Doctor ID. Both these parameters help in retrieving the information about the doctor and provides it as a response to the user.

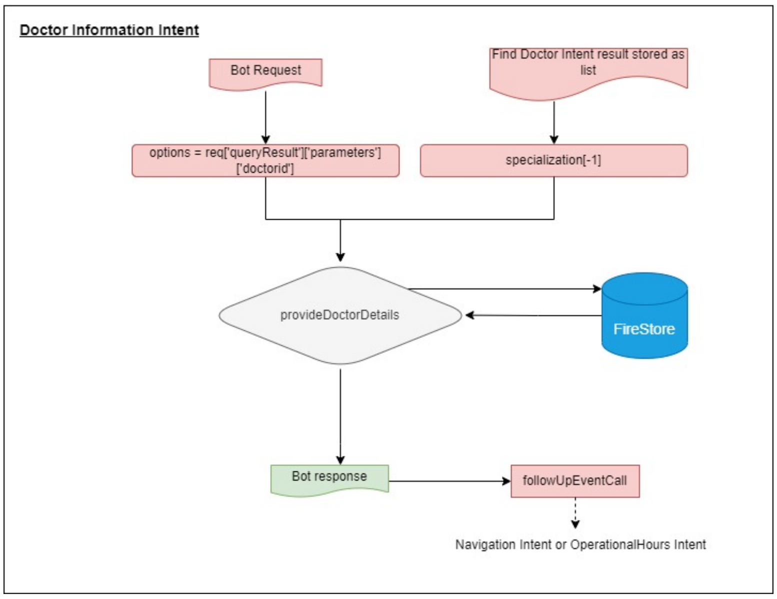

Doctor Information Intent
Bot Request
Find Doctor Intent result stored as list
options = req['queryResult']['parameters']['doctorid']
specialization[-1]
provideDoctorDetails
FireStore
Bot response
followUpEventCall
Navigation Intent or OperationalHours Intent

Figure 4.6: UML for Doctor Information Intent Flow

4.7 Operational Hours Intent

This is a follow-up intent to Doctor Information Intent. The Doctor ID and specialization which is stored from the Find Doctors and Doctor Information Intent is passed to the **provideOperationalHours()** function. This in turn retrieves the Operational hours from the Firestore database and provides the response to the user.The follow-up responses are the navigational routes intent and Exit Intent. The user can choose either to exit or know more information about the doctor.

Phrase	**Entity extracted**	**Intent**
Operational Hours	operational hours	operationalHours
When does the doctor open his clinic	open	operationalHours
availability of the doctor	availability	operationalHours
When does the doctor work	work	operationalHours

Table 4.2: Example operationalHours Intent Training Phrases and Entities

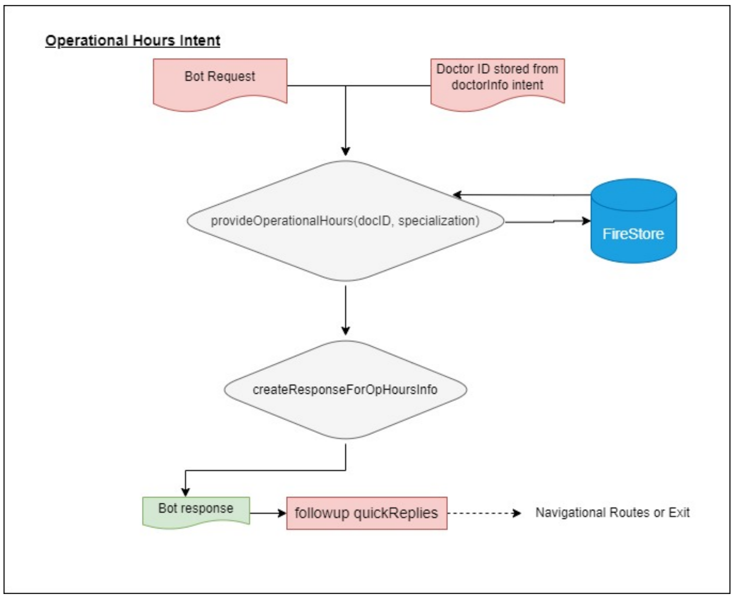

Operational Hours Intent
Bot Request
Doctor ID stored from doctorInfo intent
provideOperationalHours(docID, specialization)
FireStore
createResponseForOpHoursInfo
Bot response
followup quickReplies
Navigational Routes or Exit

Figure 4.7: UML for Operational Hour Intent Flow

4.8 Navigational Routes Intent

The navigational routes intent requires two parameters to retrieve necessary data from the database - the selected Doctor ID from the Doctor Information Intent and the specialization of the Doctor from the Find Doctors Intent. These parameters are passed to the **provideNavigationalRoutes()** function and the retrieved data is sent to the user via **createResponseForNavigationalInfo()**. This function also has quickReplies which helps the user to either exit or find the **OperationalHours()**.

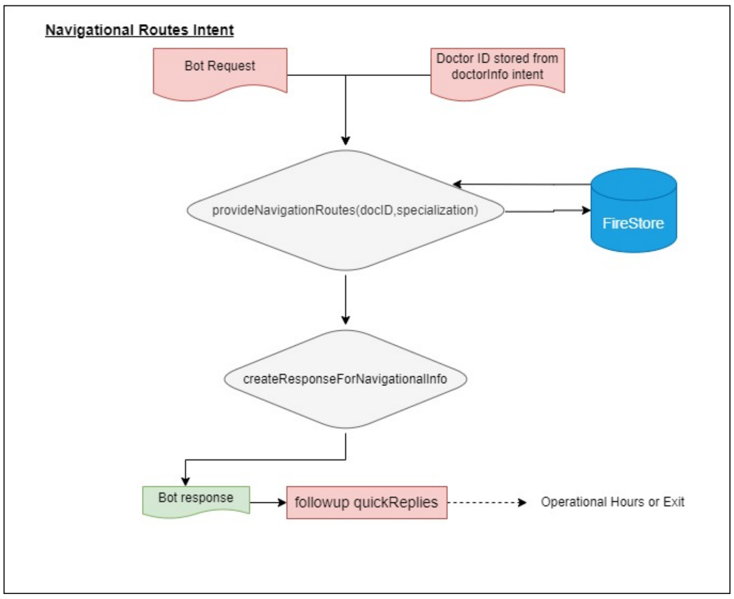

Navigational Routes Intent
Bot Request
Doctor ID stored from doctorInfo intent
provideNavigationRoutes(docID,specialization)
FireStore
createResponseForNavigationalInfo
Bot response
followup quickReplies
Operational Hours or Exit

Figure 4.8: UML for Navigational Hour Intent Flow

4.9 Pharmacy Emergency Intent

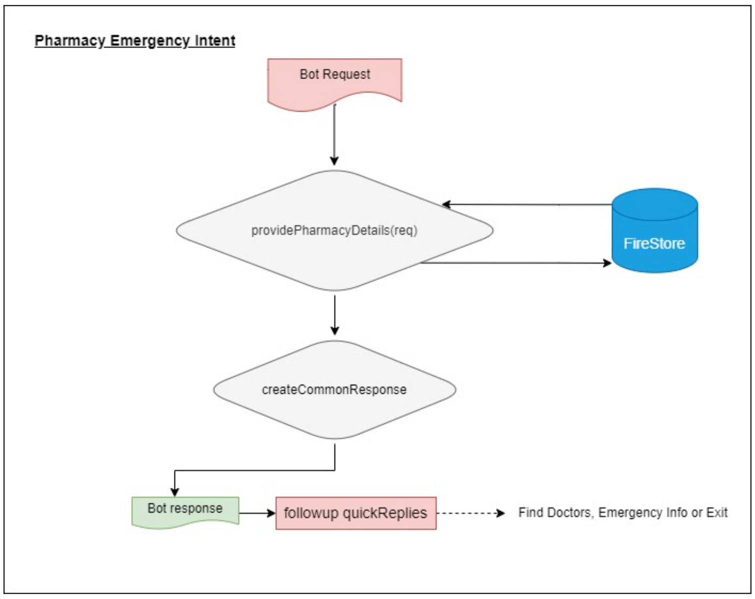

Pharmacy Emergency Intent
Bot Request
providePharmacyDetails(req)
FireStore
createCommonResponse
Bot response
followup quickReplies
Find Doctors, Emergency Info or Exit

Figure 4.9: UML for Pharmacy Emergency Intent Flow

The Pharmacy Emergency Service is one among the services provided in the service Intent. When the user selects the Pharmacy Emergency option, the bot calls the **providePharmacyDetails()** function which retrieves the emergency pharmacy contacts from the Firestore Database. At the end of this intent, the bot allows the user to visit other services or exit.

4.10 Emergency Information Intent

Medimate offers an emergency information service. When the user chooses Emergency Information Intent from the Services, the bot retrieves all the emergency information from the Firestore Database. It has followUp Quick Replies to find the Emergency Information details , Find Doctors or to exit the bot.

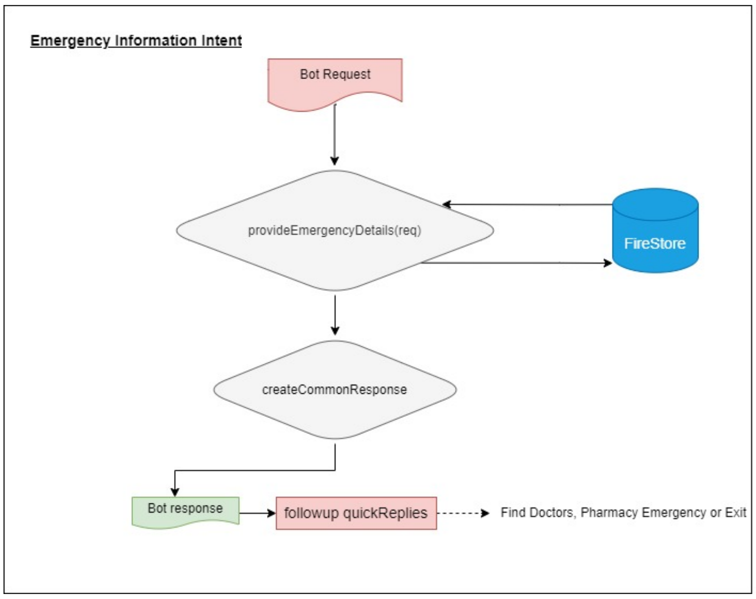

Emergency Information Intent
Bot Request
provideEmergencyDetails(req)
FireStore
createCommonResponse
Bot response
followup quickReplies
Find Doctors, Pharmacy Emergency or Exit

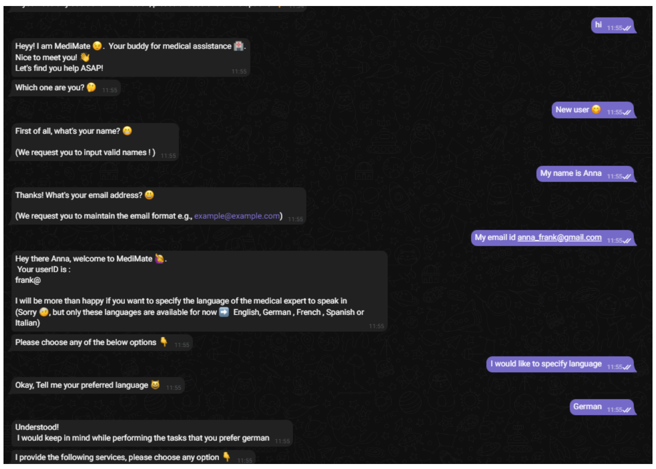
Figure 4.10: UML for Emergency Information Intent Flow

5 Conversation Scenarios

5.1 Scenario: User tries to log in or register

As soon as the user initiates a chat with the chatbot, the user is provided with two options registering themselves with the system or logging in. This is a key factor in our chatbot as we use this to store user history with the chatbot. For logging in, the chatbot asks users for their user-id. If it is present a user is given different options to find doctors.

In the case of registration, the user is prompted to provide their Name, Email Address. Upon successful registration, the user is redirected to the next step of finding a doctor based on their symptoms, or to emergency and pharmacy emergency contacts followed by helping them setting their choice of language If the user provides an email address that is already being used by another user, they will be prompted to use a different one.



hi 11:55

Heyy! I am MediMate 😔. Your buddy for medical assistance 🏥.
Nice to meet you! 👋
Let's find you help ASAP! 11:55

Which one are you? 🤔 11:55

New user 😋 11:55

First of all, what's your name? 😬

(We request you to input valid names !) 11:55

My name is Anna 11:55

Thanks! What's your email address? 😃

(We request you to maintain the email format e.g., example@example.com) 11:55

My email id anna_frank@gmail.com 11:55

Hey there Anna, welcome to MediMate 🙋.
Your userID is :
frank@

I will be more than happy if you want to specify the language of the medical expert to speak in
(Sorry 🥺, but only these languages are available for now ➡️ English, German , French , Spanish or Italian) 11:55

Please choose any of the below options 👇 11:55

I would like to specify language 11:55

Okay, Tell me your preferred language 😺 11:55

German 11:55

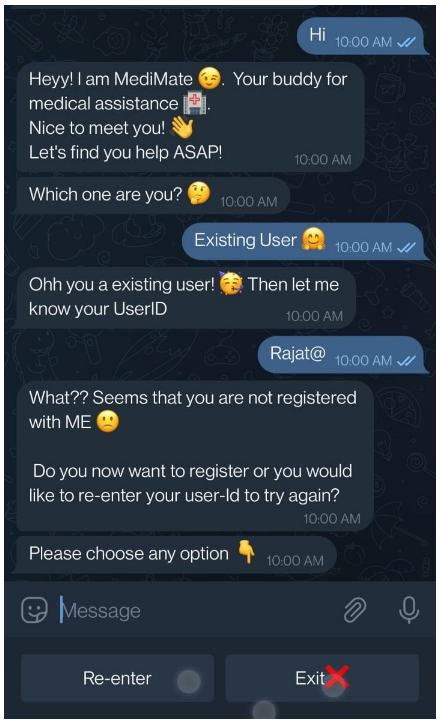
Understood!
I would keep in mind while performing the tasks that you prefer german 11:55

I provide the following services, please choose any option 👇 11:55

Figure 5.1: Register user

5.2 Scenario: User tries to log in when he is not registered with the system

If a user tries logging into the chatbot without registering themselves with the system, then the chatbot lets the user know that he is not registered with us and gives them an option of reentering the user id in case of any typos.



Hi 10:00 AM

Heyy! I am MediMate 😉. Your buddy for medical assistance 🏥.
Nice to meet you! 👋
Let's find you help ASAP! 10:00 AM

Which one are you? 🤔 10:00 AM

Existing User 🤗 10:00 AM

Ohh you a existing user! 🥳 Then let me know your UserID 10:00 AM

Rajat@ 10:00 AM

What?? Seems that you are not registered with ME 🙁

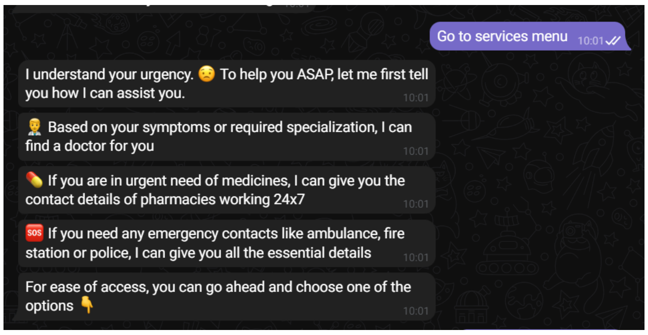
Do you now want to register or you would like to re-enter your user-Id to try again? 10:00 AM

Please choose any option 👇 10:00 AM

Figure 5.2: Wrong user id entry

5.3 Scenario: User wants all details of an Ophthalmologist

After the user is successfully logged in, the user enters the phrase which has an entity matched to an Ophthalmologist. The Doctor IDs are listed and the user enters the Doctor ID listed above. Successful retrieval of the doctor's details is done within the current day's operational hours. Additionally, the user clicks on Operational Hours to know all the Operational Hours of the doctor. Following this, the user also seeks the routes for reaching the doctor and exits the conversation.



Go to services menu 10:01

I understand your urgency. 😟 To help you ASAP, let me first tell you how I can assist you. 10:01

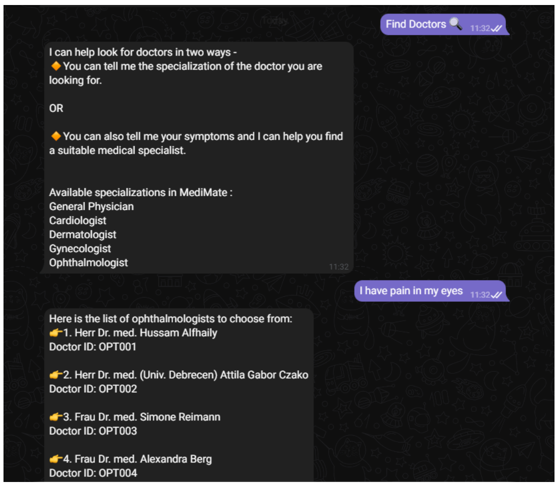
👨‍⚕️ Based on your symptoms or required specialization, I can find a doctor for you 10:01

💊 If you are in urgent need of medicines, I can give you the contact details of pharmacies working 24x7 10:01

🆘 If you need any emergency contacts like ambulance, fire station or police, I can give you all the essential details 10:01

For ease of access, you can go ahead and choose one of the options 👇 10:01

Figure 5.3: Services list are explained



Find Doctors 🔍 11:32

I can help look for doctors in two ways -
🔶 You can tell me the specialization of the doctor you are looking for.

OR

🔶 You can also tell me your symptoms and I can help you find a suitable medical specialist.

Available specializations in MediMate :
General Physician
Cardiologist
Dermatologist
Gynecologist
Ophthalmologist 11:32

I have pain in my eyes 11:32

Here is the list of ophthalmologists to choose from:
👉1. Herr Dr. med. Hussam Alfhaily
Doctor ID: OPT001

👉2. Herr Dr. med. (Univ. Debrecen) Attila Gabor Czako
Doctor ID: OPT002

👉3. Frau Dr. med. Simone Reimann
Doctor ID: OPT003

👉4. Frau Dr. med. Alexandra Berg
Doctor ID: OPT004

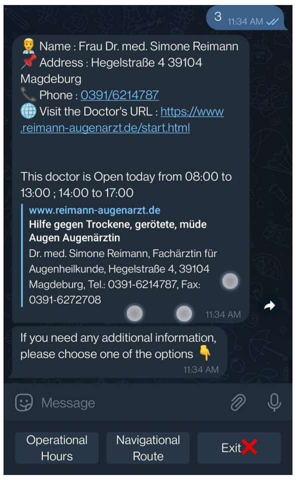
Figure 5.4: User queries for doctor

5.4 Scenario: User enters a number instead of Doctor ID

Ideally, we do not encourage the user to enter a number but instead, enter the Doctor's ID. In case, the user fails to understand the required input, there are two sub-scenarios. The user can enter a number that indicates the position of the doctor in the list provided from the Find Doctors Intent. Alternatively, the user can enter any other number. Both scenarios are discussed below.

5.4.1 User enters number pertaining to the position of Doctor

In this case, the user enters the position of the doctor on the list. To handle this case, we have used a list data structure containing the list of Doctor IDs and retrieved the details based on the position given as input.



3 11:34 AM

👨‍⚕️ Name : Frau Dr. med. Simone Reimann
📌 Address : Hegelstraße 4 39104 Magdeburg
📞 Phone : 0391/6214787
🌐 Visit the Doctor's URL : https://www.reimann-augenarzt.de/start.html

This doctor is Open today from 08:00 to 13:00 ; 14:00 to 17:00

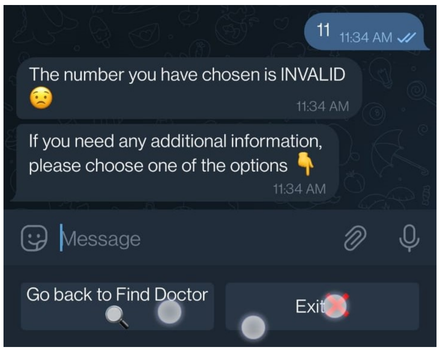
www.reimann-augenarzt.de
Hilfe gegen Trockene, gerötete, müde Augen Augenärztin
Dr. med. Simone Reimann, Fachärztin für Augenheilkunde, Hegelstraße 4, 39104 Magdeburg, Tel.: 0391-6214787, Fax: 0391-6272708
11:34 AM

If you need any additional information, please choose one of the options 👇
11:34 AM

Figure 5.5: Fetching through doctor serial no

5.4.2 User enters number larger than the number of doctors listed

As this does not make sense to the bot and does not provide any information about which Doctor ID to retrieve, an error message is provided to the user requesting to input the Doctor ID and that the number entered is invalid.



11 11:34 AM

The number you have chosen is INVALID 😟
11:34 AM

If you need any additional information, please choose one of the options 👇
11:34 AM

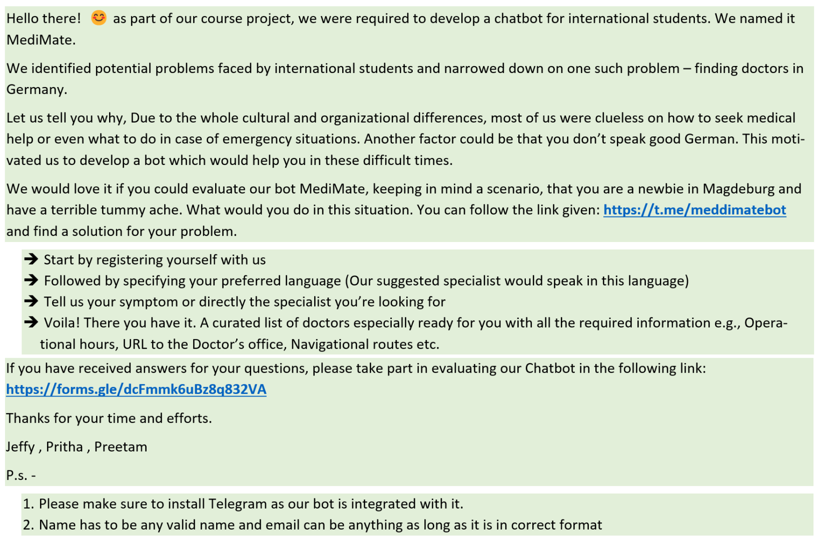
Figure 5.6: Fetching through invalid doctor serial no

6 Detailed Results of Evaluation

Followed by the successful development of our chatbot we proceeded with the evaluation. This chapter deals with the procedure and findings of our experiement. About 17 volunteers participated in our user evaluation.

6.1 Evaluation Procedure

We prepared a brief introduction of our experiment, its motives, and tasks and gave out them prior to the experiments. The following Figure: 6.1 is one of such introductions given to our volunteers:



Hello there! 😊 as part of our course project, we were required to develop a chatbot for international students. We named it MediMate.

We identified potential problems faced by international students and narrowed down on one such problem – finding doctors in Germany.

Let us tell you why, Due to the whole cultural and organizational differences, most of us were clueless on how to seek medical help or even what to do in case of emergency situations. Another factor could be that you don't speak good German. This motivated us to develop a bot which would help you in these difficult times.

We would love it if you could evaluate our bot MediMate, keeping in mind a scenario, that you are a newbie in Magdeburg and have a terrible tummy ache. What would you do in this situation. You can follow the link given: **https://t.me/meddimatebot** and find a solution for your problem.

- ➔ Start by registering yourself with us
- ➔ Followed by specifying your preferred language (Our suggested specialist would speak in this language)
- ➔ Tell us your symptom or directly the specialist you're looking for
- ➔ Voila! There you have it. A curated list of doctors especially ready for you with all the required information e.g., Operational hours, URL to the Doctor's office, Navigational routes etc.

If you have received answers for your questions, please take part in evaluating our Chatbot in the following link: **https://forms.gle/dcFmmk6uBz8q832VA**

Thanks for your time and efforts.

Jeffy , Pritha , Preetam

P.s. -

1. Please make sure to install Telegram as our bot is integrated with it.
2. Name has to be any valid name and email can be anything as long as it is in correct format

Figure 6.1: Evaluation form introduction

We assigned different tasks to the participants e.g.,

- find doctors based on their language preference and symptoms
- find doctors based on their choice of specialization
- find emergency contacts
- find pharmacy emergency contacts
- save notes about their prior appointment

Followed by the distribution of the introduction and them successfully performing the task we then handed out the links to our Google form [6] which contained the UEQ.

6.2 Results

[14]We inserted the data acquired through the form inside the data analytics tool provided by UEQ and interpreted the result.

6.2.1 Scale Means

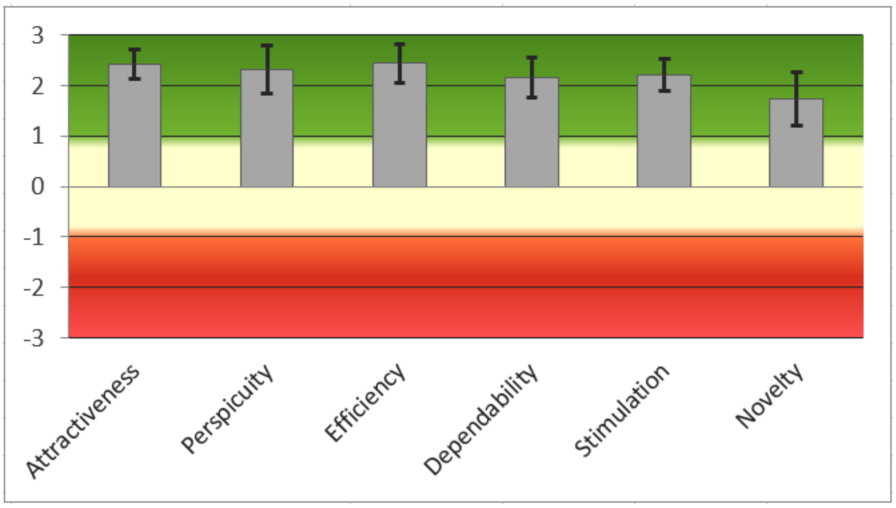


Figure 6.2: Scale means evaluation

Scale means is a general barplot of the values of the resulting scales. The scales ranges from +3 (extremely good) to -3(horribly bad). In accordance with the accepted interpretation, the scale's values between -0.8 and 0.8 signify a neural evaluation of the relevant scale, whereas values more than or equal to 0.8 indicate a positive evaluation and values less than or equal to 0.8 indicate a negative evaluation. In our case in Figure: 6.2 we noticed a very positive evulation throughout all the categories with a value typically over 2.0 except in novelty.

6.2.2 Benchmark Evaluation

In order to have a better understanding of the relative quality of our Med-iMate, it was also required to compare the measured user experience to the outcomes of a benchmark data collection encompassing rather varied typical products that was already present in the UEQ tool. This benchmark dataset includes the results of 452 UEQ product evaluations (with a total of 20190 participants in all evaluations).

In standard practice the benchmark classifies products into 5 categories (per scale):

- Excellent: In the range of the 10% best results.
- Good: 10% of the results in the benchmark data set are better and 75% of the results are worse.
- Above average: 25% of the results in the benchmark are better than the result for the evaluated product, 50% of the results are worse.
- Below average: 50% of the results in the benchmark are better than the result for the evaluated product, 25% of the results are worse.
- Bad: In the range of the 25% worst results.

The following Figure: 6.3 shows the benchmark evaluation of MediMate -

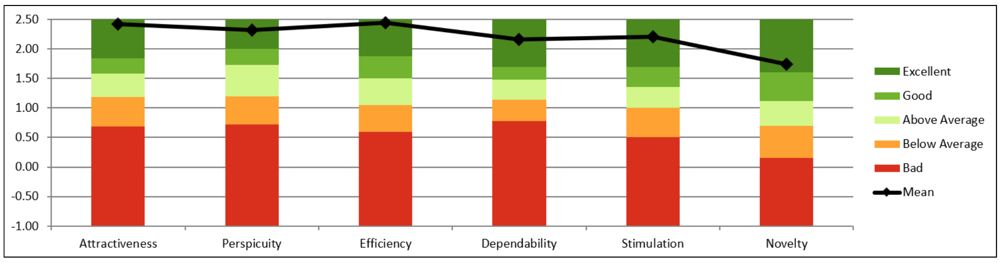
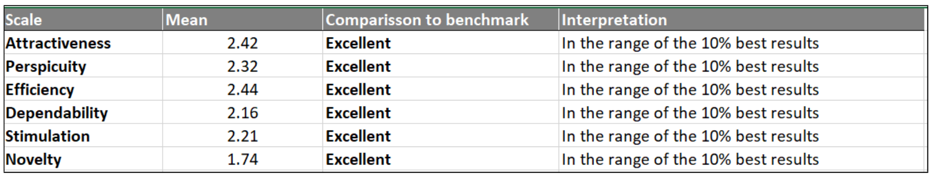


Figure 6.3: Benchmark evaluation

It can be determined from the above figure 6.3 that MediMate is classified as an **Excellent** product. The quantitative reasoning can be found in the following Figure: 6.4 -



Scale	Mean	Comparisson to benchmark	Interpretation
Attractiveness	2.42	**Excellent**	In the range of the 10% best results
Perspicuity	2.32	**Excellent**	In the range of the 10% best results
Efficiency	2.44	**Excellent**	In the range of the 10% best results
Dependability	2.16	**Excellent**	In the range of the 10% best results
Stimulation	2.21	**Excellent**	In the range of the 10% best results
Novelty	1.74	**Excellent**	In the range of the 10% best results

Figure 6.4: Benchmark evaluation measures

Although we are much aware that peer-evaluation with 17 participants cannot produce very liable results and hence we are still trying to improve MediMate's performance.

As a means of determining the areas of improvement our participants were also asked the question *"What one thing about MediMate would you change, and why?"*. The quantitative analysis of user experience cannot directly address this question. So we considered few of the frequent responses and based our future improvement scope on them, some of which were:

- to include more "symptoms and diseases" for the bot.
- to include more better quick replies for navigation.
- to include other platforms like Whatsapp which was used more by international students over telegram
- to have doctors suggestions based on the location of the users

6.3 Evaluation Outcome

The confidence intervals must be small in order to evaluate a consistent opinion. The overall intervals for attractiveness, clarity, efficiency, reliability, motivation and novelty are small for the entire system. From the results, we can conclude that the performance measures seem to be Excellent. The Evaluation has also led to extensions in our research question which would be covered as Future Work. As participants, evaluated the bot immediately after usage, we could also identify room for potential improvement with respect to Response Quality and Design. The evaluation, thus proved that Medimate is a chatbot providing above-average performance to the user.

6.4 Conclusion

Our research was mainly to accommodate all the necessary information to help international students in seeking information for medical help. In our benchmark evaluation, we see that our **Dependability** score comes under the category of **Excellent** with a mean of **2.162**. It comprises all the necessary information such as the contact details, navigational routes , operational hours and filters according to the language of communication of the user. Although, our bot does not cover all symptoms and specializations, we have touched the most common specializations and also emergency data. This has therefore led users to depend on the bot in times of need. Despite the fact that some insurance companies have a feature to filter out with respect to specialization and language, they lack popularity. In addition, our bot also comforts the user by having an interactive conversation. Overall, we have included the main functionalities required but on adding more, our chatbot, Medimate, would prove to be more beneficial for the growing population of incoming international students in Germany.

7 Learning Outcomes

7.1 Issues and Resolutions

While progressing through the development and testing phase of our chatbot build we dealt with challenges that were sometimes posed as a hindrance to the implementation of functions. Although maintaining a standardized coding practice and continuous testing we could overcome most of those. Following are a few significant roadblocks that slowed our progress at some stages -

- **Web Hosting Services:** We initially used **ngrok** where we shared the server on the local machine which ended up throwing several *502 bad gateway, 404 Errors*. At last, we decided to host it on a global host called *PythonAnywhere*.
- **Username:** Defining the name provided by the user was tricky. Reasons are,
    1. User might choose to provide an abstract name. Initially, we kept the Dialogflow user_name parameter as the type *any*. Later we realized that this way entire sentences are getting accepted as names.
    2. User might feel like wrapping their name in a sentence. To avoid that we should keep the system-parameter as the type *given_name* and *last_name*. In this case, none of the abstract names were accepted. Hence at last we settled for parameter type *person* which accepts globally acceptable common names provided irrespective of in-sentence formation or as an individual.

- **Choosing Doctor's Serial No instead of ID:** If a user misses to follow the conversation and mistakenly chooses the serial no. of the doctor instead of the Doc ID as asked, the system wouldn't recognize the response. We later accommodated the serials as well.

7.2 Positive Outcomes

Though we had faced a couple of issues, we also had positive learning outcomes from Medimate. Following are the positive outcomes:

- DialogFlow ES is extensively used for small scale applications. We managed to efficiently model it in such a way , that it doesnt act as an FAQ bot.
- As we used Python as our backend, we could efficiently use all the necessary data structures like lists and dictionaries for the design of our dialog manager.

8 Future Work

We would like to include several functionalities in our chatbot. During our implementation, we encountered various new ideas to expand our research question. Some of them are enlisted below:

- In our current work, we have selected only a limited number of doctors in the region of Magdeburg. We would like to **extend this to other parts of Germany** to help all international students and filter based on **location** of the user.
- Another extension for our bot is to get the input of the users appointment and add it to Google Calendar with a follow up immediately after the appointment.As the bot is hosted online on Pythonanywhere, it can match the system date with the **Google Calendar** date and ask the user to input notes on the doctor visit for further use.
- **SMS and Mailing** services are a possible future scope as we are using Firestore and just by getting an additional information - phone number and EMail ID, users can get details without needing to open Telegram.
- A major issue we faced during development and evaluation was to include several symptoms and doctor specializations. There are several **APIs** such as SymptomChecker and **medical corpuses** such as SciSpacy available. This could be essential in advancement of the bot.
- We would also like to include information about one's **insurance coverage** as a functionality of MediMate.

The above mentioned advancements are vast and would prove to beneficial to the incoming student community. This would therefore widen Germany's welcoming arms to expats.

A
List of Tools used

- Diagrams - [1]
- Dialogflow - [10]
- Firebase Firestore - [5]
- ngrok - [12]

B
List of Figures

C
List of Tables

D
Bibliography

[1] [ALDER] **Diagrams.net**. `https://app.diagrams.net/`.

[2] [DANIEL und MARTIN] **Speech and Language Processing Chatbots Dialogue Systems**.

[3] [DERIU et al. 2019] J. Deriu, A. Rodrigo, A. Otegi, G. Echegoyen, S. Rosset, E. Agirre und M. Cieliebak. **Survey on Evaluation Methods for Dialogue Systems**. 2019.

[4] [DJAMASBI und STRONG 2019] S. Djamasbi und D. Strong. **User Experience-driven Innovation in Smart and Connected Worlds**. AIS Transactions on Human-Computer Interaction, pp. 215–231, 2019.

[5] [FIRESTORE] **Firestore**. `https://cloud.google.com/firestore`.

[6] [FORM] **MediMate evaluation**.

[7] [GOOGLE] **Dialogflow Basics**. `https://www.tutorialspoint.com/flask/index.htm`.

[8] [GOOGLE] **Python integration with Dialogflow and Firestore**. `https://marcusyatim.medium.com/google-dialogflow-with-fulfillment-webhook-and-python-flask-6b3c44dbeef0`.

[9] [HULLEY SB 2007] B. W. G. D. N. T. D. C. R. L. W. . W. . Hulley SB, Cummings SR. **FINER-Criteria-for-Developing-Research-Questions-1**. 2007.

[10] [JAVAPOINT] **Dialogflow**. `https://www.javatpoint.com/dialogflow`.

[11] [Maroengsit et al. 2019] W. Maroengsit, T. Piyakulpinyo, K. Phonyiam, S. Pongnumkul, P. Chaovalit und T. Theeramunkong. **A survey on evaluation methods for chatbots**. Vol. Part F148391, pp. 111–119. 2019, Association for Computing Machinery.

[12] [NGROK] **Ngrok**. `https://ngrok.com/`.

[13] [REHKOPF] **What is a kanban board?** `https://www.atlassian.com/agile/kanban/boards`.

[14] [Schrepp] **User Experience Questionnaire Handbook**.

[15] [Telegram] **Telegram**. `https://telegram.org/`.

[16] [TuturialsPoint] **Flask App**. `https://www.tutorialspoint.com/flask/index.htm`.

[17] [Yadav und Kaushik 2022] S. Yadav und A. Kaushik. **Do You Ever Get Off Track in a Conversation? The Conversational System's Anatomy and Evaluation Metrics**. Knowledge, Vol. 2:55–87, 2022.

Declaration of Academic Integrity

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Datum: ....

(Signature)