CHAT CENTER FRAMEWORK BASED ON NLP

GROUP MEMBERS:

HAZIM TALAAT BUKHARI	1740758
AHMED FAHAD ALMUTAIRI	1740898
SULTAN ABDULAZIZ ALFASHKH	1741406

THIS REPORT IS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR DEGREE IN COMPUTER SCIENCE DEPARTMENT

SUPERVISOR NAME:

Dr. MOHAMMED YAHYA DAHAB

COMPUTER SCIENCE DEPARTMENT

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

KING ABDULAZIZ UNIVERSITY

Declaration of Originality

We herby declare that this project report is based on our original work except for citations and questions, which had been duly acknowledged. We also declare that it has not been previously and concurrently submitted for any other degree or award at KAU or other institutions.

Name	ID	Signature
HAZIM BUKHARI	**1740758**	
AHMED ALMUTIRI	**1740898**	
SULTAN ABDULAZIZ ALFASHKH	**1741406**	

Acknowledgement

We are very grateful for everyone who help us to complete the first part of the senior project successfully. Firstly, we would thank Allah for the immense blessing he has given it to us and that led us to continue to the end. We also extend our thanks and gratitude to our supervisor Dr. Mohammed Yahia Dahab, for his support and his encouragement.

Furthermore, special thanks for:

- Our coordinators, for their expertise and guidance.
- Our parents, for trust and faith in our abilities.

We are truly grateful to all, even those who have not been personally mentioned, we thank you from the bottom of the heart.

Abstract

Since the natural language processing application becomes more popular in real-world applications but recently there is a new movement in using natural language processing to retrieve the data from a database which is known as a text to structured query language. This technology can be used to reduce the operational cost and workload on the human agent when dealing with customer queries which can be answered from the database. Also, it helps non-technical people to retrieve the data from the database without knowing the structured query language. It is worth mentioning that the main feature of the chat center framework based on natural language processing is a work in different domains which allows the chat center framework based on natural language processing to retrieve the answer from a database in different domains. Also, the chat center framework based on natural language processing aims to mimic the human conversation to replay to the user as human as possible to encourage the customer to continue the conversation.

الملخص

نظرًا لأن تطبيق معالجة اللغة الطبيعية أصبح أكثر شيوعًا في تطبيقات العالم الحقيقي ، ولكن في الآونة الأخيرة هناك حركة جديدة في استخدام معالجة اللغة الطبيعية لاسترداد البيانات من قاعدة بيانات تُعرف باسم تحويل النص إلى لغة قواعد البيانات. يمكن استخدام هذه التقنية لتقليل التكلفة التشغيلية وعبء العمل على العامل البشري عند التعامل مع استفسارات العملاء التي يمكن الإجابة عليها من قاعدة البيانات. كما أنه يساعد الأشخاص غير التقنيين على استرداد البيانات من قاعدة البيانات دون معرفة لغة قواعد البيانات. ومن الجدير بالذكر أن السمة الرئيسية لإطار عمل مركز الدردشة القائم على معالجة اللغة الطبيعية هو العمل في مجالات مختلفة مما يسمح لإطار مركز الدردشة القائم على معالجة اللغة الطبيعية باسترداد الإجابة من قاعدة بيانات في مجالات مختلفة. أيضًا ، يهدف إطار عمل مركز الدردشة القائم على معالجة اللغة الطبيعية إلى محاكاة المحادثة البشرية للرد على المستخدم كالبشر قدر الإمكان لتشجيع العميل على مواصلة المحادثة.

Contents

List Of Tables

List Of Figures

List Of Abbreviations

- NLP: natural language processing.
- SQL: structured query language.
- Table2answer: Table to answer.
- GPT-2: Generative Pretrained Transformer version 2.
- AI: Artificial Intelligence.
- SParC: Semantic Parsing in Context.
- CoSQL: Conversational text-to-SQL.

List Of Appendices

Chapter 1 Introduction

1.1 Introduction

The chat center framework based on natural language processing to convert the human language in English to structured query language using deep learning techniques in the framework to solve workload and cost operations. Which helps the organizations to respond to their user's queries. In order to identify how the chat center framework based on natural language processing helps the users to answer their questions in an efficient way, the data was gathered based on reading research papers that related to text-to-SQL in the NLP. This data helped to determine the system requirements. In addition, this report contains a description of the proposed system scenario, data requirements, and software requirements. Also, it contains initial structure diagrams that build based on the system requirement. Therefore, the initial structure diagrams explain the relation between the system components and how many steps should be taken to achieve the system goals.

1.2 Aim

The aims of the project are to develop framework that help organizations to improve their business. The framework can help non-technical employees by retrieving information from the database by using English language. Also, they can use it to help their customer by answer their questions and serve them in short period of time and reduce operating costs.

1.3 Problem Definition

Build framework that translate English text to Structured Query Language (SQL) by using Natural language processing (NLP) and then retrieving the answer from database according to a given domain and provide users with the answer in acceptable format.

1.4 Scope

Cover many databases on many different domains.

1.5 Target Users

- Organizations
- Non-technical people
- Anyone who is interested in automating the process of answering questions in a specific domains.

1.6 Suggested Solution

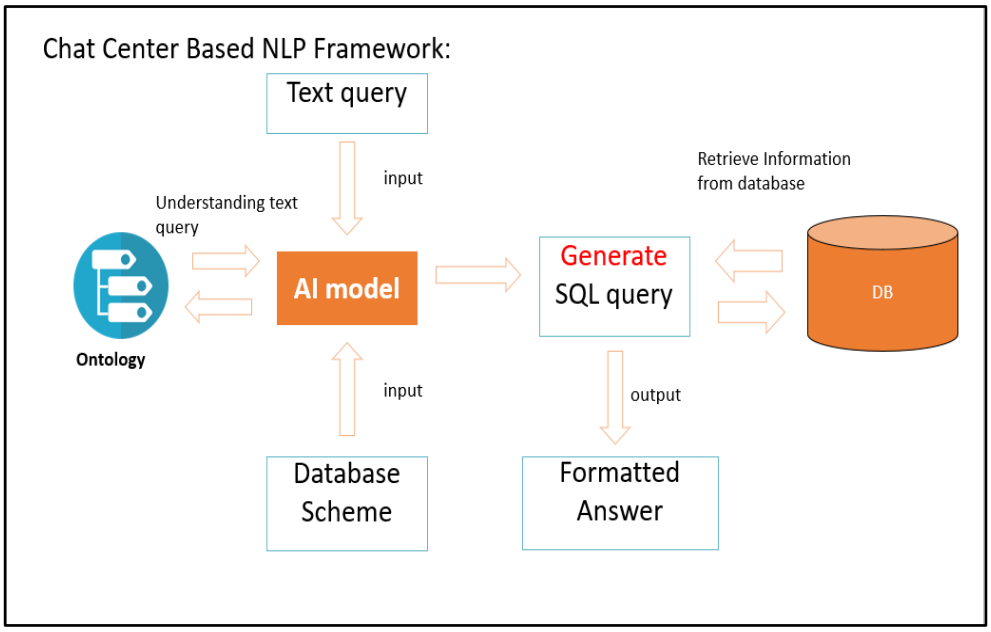


Figure 1 shows the Chat Center framework-based NLP model

There are many different solutions for this problem.

- **The first solution**: is to build a new AI model that translate text query to SQL based on specific patterns that divided the user inputs into a function, and columns like figure 2 below:

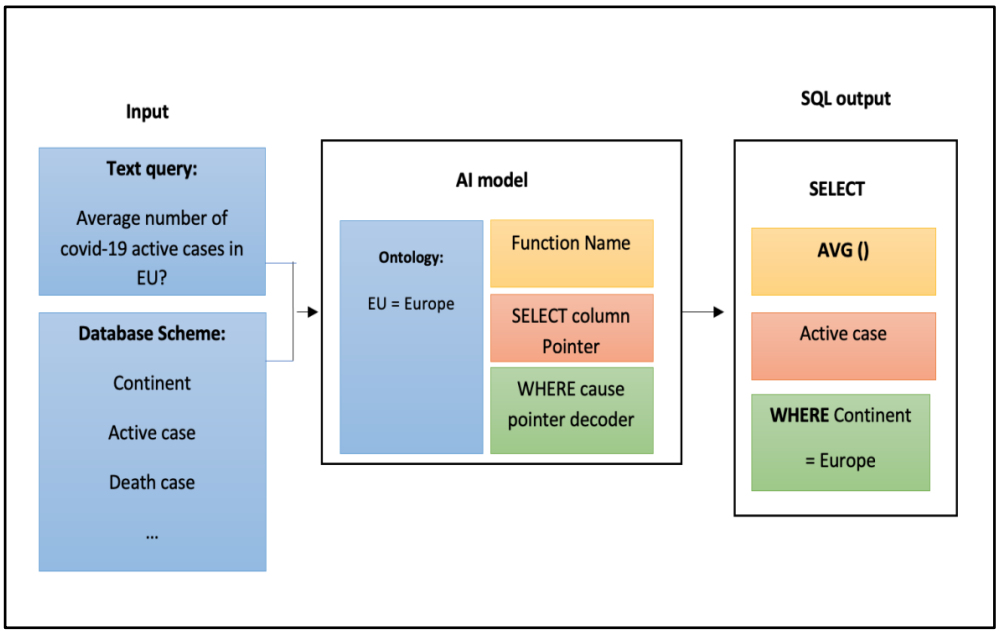


Figure 2 This figure shows the text query as input from the user and database scheme from the database that contains one table all these parameters entered to the AI model that has implicit ontology that understand the text query after that AI model take the text.

- **Second solution**: to modify any existing model such as
  - GPT-2 (Generative Pretrained Transformer 2)
  - TypeSQL (Knowledge-based Type-Aware Neural Text-to-SQL Generation).
  - TAPAS (Weakly Supervised Table Parsing via Pre-training).

1.7 Required Skills

- **Python programming**

  Reason: Using python in this project is substantial because all tools required for this project depends python.

- **Natural language processing (NLP)**

  Reason: It important to understand NLP concepts to format question and answer in good way.

- **Deep learn**

  Reason: Using deep learn tool like Generative Pre-Training (GPT2) in this project is very useful because it will save a lot time and resources.

- **Structured Query Language (SQL)**

  Reason: It necessary to have basic knowledge about SQL because one major part in this project is translate natural language to SQL.

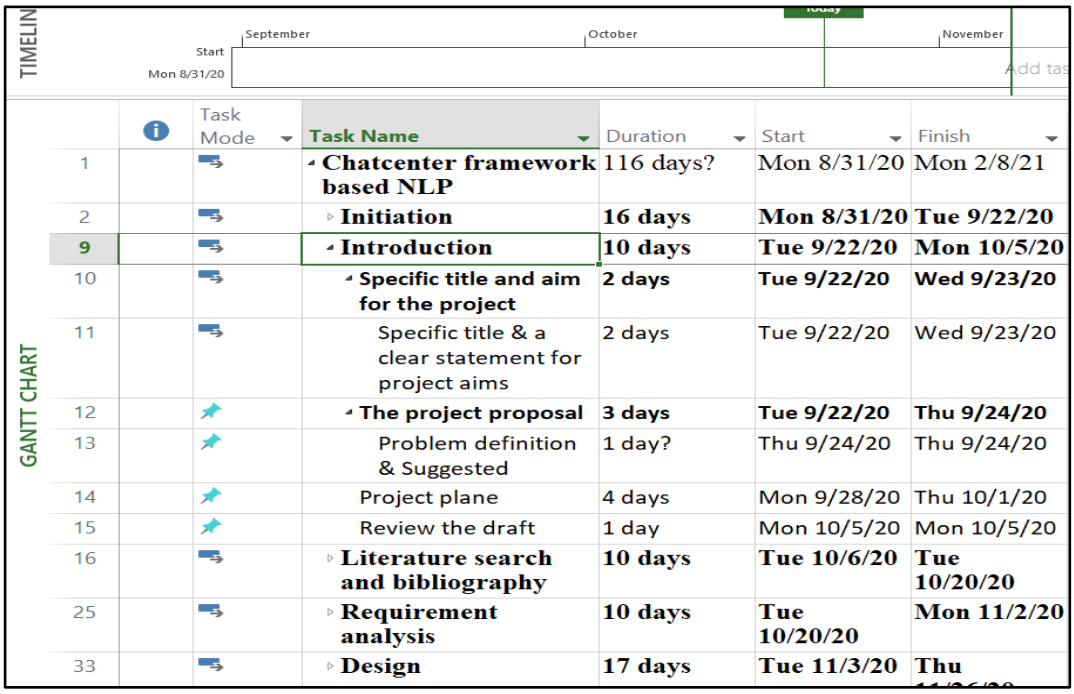
1.8 Project Management

- **Time Schedule**

GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
1		**Chatcenter framework based NLP**	116 days?	Mon 8/31/20	Mon 2/8/21
2		**Initiation**	**16 days**	**Mon 8/31/20**	**Tue 9/22/20**
3		Search for an idea	7 days	Mon 8/31/20	Tue 9/8/20
4		Define the idea	2 days	Wed 9/9/20	Thu 9/10/20
5		Proposed System	1 day	Fri 9/11/20	Fri 9/11/20
6		Determine the objectives	4 days	Mon 9/14/20	Thu 9/17/20
7		send approval form	0 days	Thu 9/17/20	Thu 9/17/20
8		Present the initial presentation.	0 days	Tue 9/22/20	Tue 9/22/20
9		**Introduction**	**10 days**	**Tue 9/22/20**	**Mon 10/5/20**
17		**Literature search and bibliography**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**
26		**Requirement analysis**	**10 days**	**Tue 10/20/20**	**Mon 11/2/20**
34		**Design**	**17 days**	**Tue 11/3/20**	**Thu 11/26/20**

Figure 3 shows plan for initiation project



TIMELIN

September October November

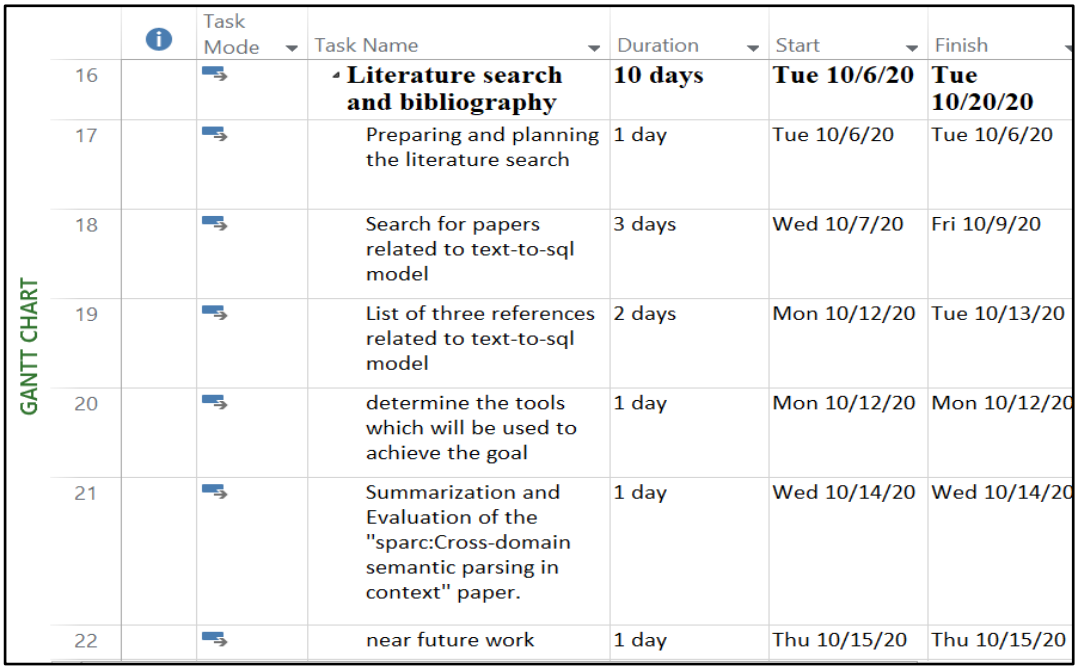
Start
Mon 8/31/20

Add tas

GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
1		**Chatcenter framework based NLP**	116 days?	Mon 8/31/20	Mon 2/8/21
2		**Initiation**	**16 days**	**Mon 8/31/20**	**Tue 9/22/20**
9		**Introduction**	**10 days**	**Tue 9/22/20**	**Mon 10/5/20**
10		**Specific title and aim for the project**	**2 days**	**Tue 9/22/20**	**Wed 9/23/20**
11		Specific title & a clear statement for project aims	2 days	Tue 9/22/20	Wed 9/23/20
12		**The project proposal**	**3 days**	**Tue 9/22/20**	**Thu 9/24/20**
13		Problem definition & Suggested	1 day?	Thu 9/24/20	Thu 9/24/20
14		Project plane	4 days	Mon 9/28/20	Thu 10/1/20
15		Review the draft	1 day	Mon 10/5/20	Mon 10/5/20
16		**Literature search and bibliography**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**
25		**Requirement analysis**	**10 days**	**Tue 10/20/20**	**Mon 11/2/20**
33		**Design**	**17 days**	**Tue 11/3/20**	**Thu**

Figure 4 shows the plan for Introduction



GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
16		**Literature search and bibliography**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**
17		Preparing and planning the literature search	1 day	Tue 10/6/20	Tue 10/6/20
18		Search for papers related to text-to-sql model	3 days	Wed 10/7/20	Fri 10/9/20
19		List of three references related to text-to-sql model	2 days	Mon 10/12/20	Tue 10/13/20
20		determine the tools which will be used to achieve the goal	1 day	Mon 10/12/20	Mon 10/12/20
21		Summarization and Evaluation of the "sparc:Cross-domain semantic parsing in context" paper.	1 day	Wed 10/14/20	Wed 10/14/20
22		near future work	1 day	Thu 10/15/20	Thu 10/15/20

Figure 5 shows the plan for literature search and bibliography

GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
16		**Literature search and bibliography**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**
25		**Requirement analysis**	**10 days**	**Tue 10/20/20**	**Mon 11/2/20**
26		**Data gathering**	**3 days**	**Tue 10/20/20**	**Thu 10/22/20**
27		Brainstorming	3 days	Tue 10/20/20	Thu 10/22/20
28		Define functional requirements and non functional requirements	3 days	Fri 10/23/20	Tue 10/27/20
29		Define system scenario	1 day	Wed 10/28/20	Wed 10/28/20
30		Define data requirements	2 days	Wed 10/28/20	Thu 10/29/20
31		Define software requirement	2 days	Fri 10/30/20	Mon 11/2/20
32		Review the draft	1 day	Fri 10/30/20	Fri 10/30/20
33		**Design**	**17 days**	**Tue 11/3/20**	**Thu 11/26/20**

Figure 6 shows the plan for Requirement

GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
16		**Literature search and bibliography**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**
25		**Requirement analysis**	**10 days**	**Tue 10/20/20**	**Mon 11/2/20**
33		**Design**	**17 days**	**Tue 11/3/20**	**Thu 11/26/20**
34		Use-case diagram	1 day	Tue 11/3/20	Tue 11/3/20
35		text-to-sql model	3 days	Wed 11/4/20	Fri 11/6/20
36		chat-center framework model	3 days	Mon 11/9/20	Wed 11/11/20
37		Review the draft	1 day	Thu 11/12/20	Thu 11/12/20
38		Deliver the second report chapter 3,4: Requirement analysis, Design and Presentation 2	0 days	Thu 11/12/20	Thu 11/12/20
39		Deliver Final report and Final presentation	0 days	Thu 11/26/20	Thu 11/26/20

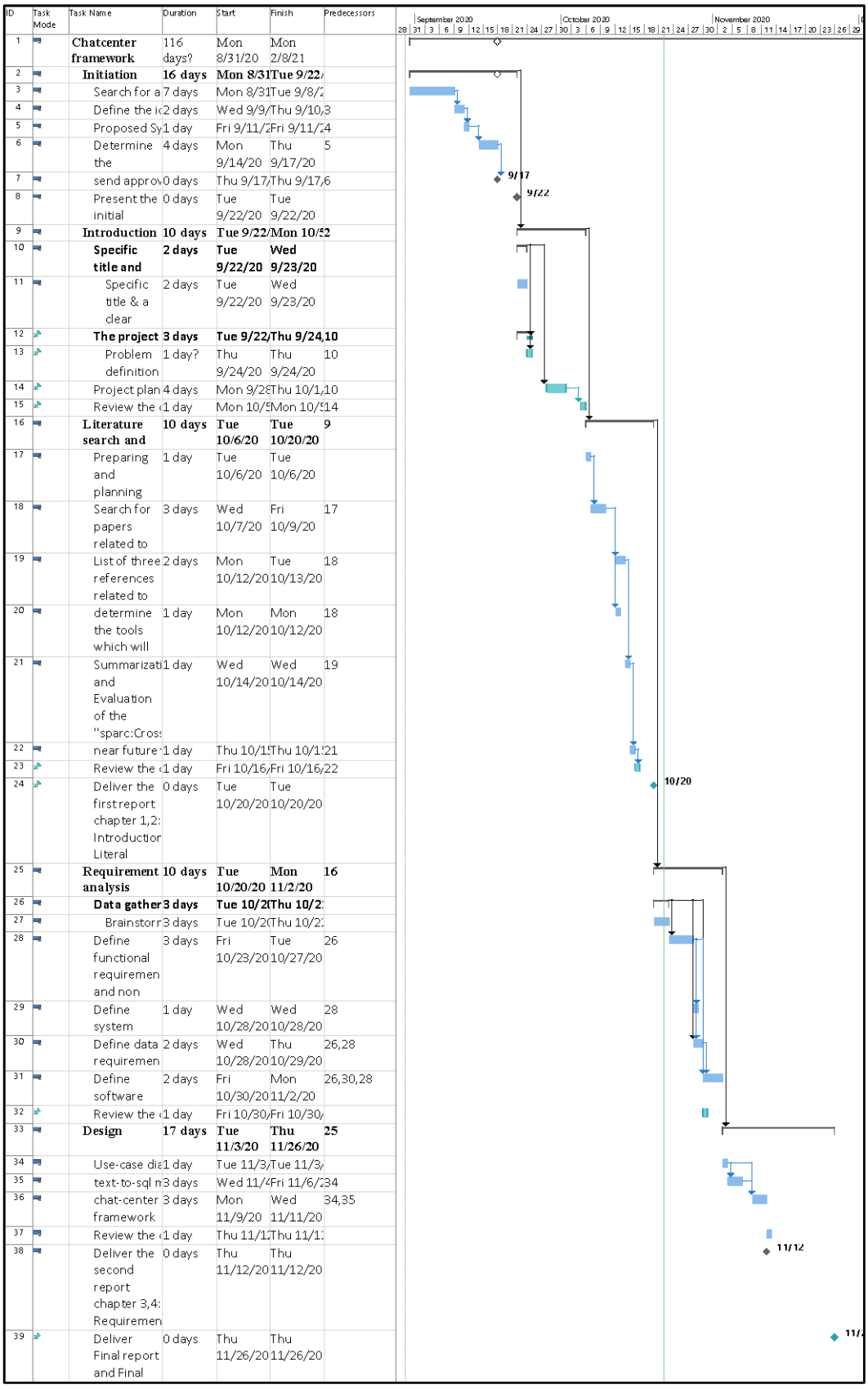
Figure 7 shows the plan for the Design

- **Milestone**

Term	tasks	Milestone#	week#	Tentative date
CPCS-498	Present the initial presentation	1	4	22/09/2020
	Deliver the first report chapter 1,2: Introduction, Literal review and Presentation 1	2	8	20-22/10/2020
	Deliver the second report chapter 3,4: Requirement analysis, Design and Presentation 2	3	12	10-12/11/2020
	*Poster Session	4	13	TBA
	Deliver the final report Chapters: 1,2,3 and 4	5	14	26/11/2020
	Final presentation	6	14	TBA

Figure 8 shows the milestone table

- **Gantt chart**



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Chatcenter framework**	116 days?	Mon 8/31/20	Mon 2/8/21	
2		**Initiation**	**16 days**	**Mon 8/31**	**Tue 9/22/**	
3		Search for a	7 days	Mon 8/31	Tue 9/8/2	
4		Define the ic	2 days	Wed 9/9/	Thu 9/10/3	
5		Proposed Sy	1 day	Fri 9/11/2	Fri 9/11/2	4
6		Determine the	4 days	Mon 9/14/20	Thu 9/17/20	5
7		send approv	0 days	Thu 9/17/	Thu 9/17/	6
8		Present the initial	0 days	Tue 9/22/20	Tue 9/22/20	
9		**Introduction**	**10 days**	**Tue 9/22/**	**Mon 10/5**	**2**
10		**Specific title and**	**2 days**	**Tue 9/22/20**	**Wed 9/23/20**	
11		Specific title & a clear	2 days	Tue 9/22/20	Wed 9/23/20	
12		**The project**	**3 days**	**Tue 9/22/**	**Thu 9/24/**	**10**
13		Problem definition	1 day?	Thu 9/24/20	Thu 9/24/20	10
14		Project plan	4 days	Mon 9/28	Thu 10/1/	10
15		Review the c	1 day	Mon 10/5	Mon 10/5	14
16		**Literature search and**	**10 days**	**Tue 10/6/20**	**Tue 10/20/20**	**9**
17		Preparing and planning	1 day	Tue 10/6/20	Tue 10/6/20	
18		Search for papers related to	3 days	Wed 10/7/20	Fri 10/9/20	17
19		List of three references related to	2 days	Mon 10/12/20	Tue 10/13/20	18
20		determine the tools which will	1 day	Mon 10/12/20	Mon 10/12/20	18
21		Summarizati and Evaluation of the "sparc:Cross	1 day	Wed 10/14/20	Wed 10/14/20	19
22		near future	1 day	Thu 10/15	Thu 10/1	21
23		Review the c	1 day	Fri 10/16/	Fri 10/16/	22
24		Deliver the first report chapter 1,2: Introductior Literal	0 days	Tue 10/20/20	Tue 10/20/20	
25		**Requirement analysis**	**10 days**	**Tue 10/20/20**	**Mon 11/2/20**	**16**
26		**Data gather**	**3 days**	**Tue 10/2**	**Thu 10/2**	
27		Brainstorr	3 days	Tue 10/2	Thu 10/2	
28		Define functional requiremen and non	3 days	Fri 10/23/20	Tue 10/27/20	26
29		Define system	1 day	Wed 10/28/20	Wed 10/28/20	28
30		Define data requiremen	2 days	Wed 10/28/20	Thu 10/29/20	26,28
31		Define software	2 days	Fri 10/30/20	Mon 11/2/20	26,30,28
32		Review the c	1 day	Fri 10/30/	Fri 10/30/	
33		**Design**	**17 days**	**Tue 11/3/20**	**Thu 11/26/20**	**25**
34		Use-case dia	1 day	Tue 11/3/	Tue 11/3/	
35		text-to-sql n	3 days	Wed 11/4	Fri 11/6/2	34
36		chat-center framework	3 days	Mon 11/9/20	Wed 11/11/20	34,35
37		Review the c	1 day	Thu 11/1	Thu 11/1	
38		Deliver the second report chapter 3,4: Requiremen	0 days	Thu 11/12/20	Thu 11/12/20	
39		Deliver Final report and Final	0 days	Thu 11/26/20	Thu 11/26/20	

Figure 9 shows the Gantt Chart

2 Literature search and bibliography

2.1 Planning for literature search

Before starting the search process, we gathered and discussed the core parts of our framework and what problems we need to search before starting to work. For the literature search both Google Scholar and Research Gate will be used in the search process.

2.2 Literature Review

- **Summary of Sparc: Cross-domain semantic parsing in context**

  The mentioned study is about building a large general dataset for text-to-sql, the goal of said dataset is to provide the researchers who are intreated in the text-to-sql task with a data set to train their models and evaluate them. Sparc has an advantage over existing text-to-sql datasets mainly it has data from multiple domains which forces the model to be general also the datasets in the evaluation process is completely new to the model, also it contains complex sql queries which poses a challenge to models.

2.3 Evaluation

The paper mentioned gave us an insight on what type of input/output we need to consider when building the framework. And it is considered a valuable resource in evaluating the AI model we will use in our framework. Finally, it gave us an idea about what kind of results we could expect from out framework.

- **SParC: Cross-Domain Semantic Parsing in Context**

  The paper was submitted on 5 jun 2019 and its goal was to to build a text-to-sql cross domain dataset for anyone who is interested in tackling this challenging task. The information are clearly presented with a semi-easy language with explained diagrams. But some background knowledge are needed especially in SQL and AI evaluation methods. The information in this paper helped us understand what data we need to collect before training/finetuning our AI model.

All the authors are from the Computer Science department in Yale university and salesforce research.

- **Table2answer: Read the database and answer without SQL**

  The paper was submitted on 12 fep 2019 and the goal is tried to approach the problem of text-to-sql from a new angle by ignoring the conversion to SQL. The Information are clear in an understandable language and is backed with examples and the diagrams are simple. Unfortunately an understanding of NLP, deep learning, and evaluation methods are needed. We could use the approach presented to help us understand the internal structure of AI models.

  The research team is made from a Rokid AI which is a Chinese company specialized in AI products, and China Electronic Technology Group Corporation Information Science Academy members.

- **Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions**

  The paper was submitted on 2 sep 2019 and its goal is to build a deep learning model by modifying the user's query before feeding it to the model. Its language was hard to understand especially in the math part, diagrams were clear to understand but knowledge about NLP, deep learning, math are needed to understand the paper. We could use the approach presented to help us understand the internal structure of AI models.

  All the authors are from the Computer Science department in Yale university, Salesforce research, and Cornell university.

2.4 Comparing methods to find information:

Method (1) research papers: Provide a lot of deep information regarding models and approaches which could not be found anywhere else, but it is hard to understand if you lack the necessary information.

Method (2) Blogs & Youtube: Usually has easy to understand explanation of complex concepts with a lot of hand on examples for popular models, but you can't find some really technical information or any explanation of less known models.

2.5 Equipment or software that will be used

Below are the tools which will be used to achieve the goal of this project:

- Python: a programming language with strong libraries in NLP and AI.
- Text-to-sql AI model: an AI model that can convert a natural language query to the corresponding sql query.
- Google Colab: A web based virtual environment Notebook.
- Sparc: A text-to-sql Dataset.
- CoSQL: A text-to-sql Dataset.

3 Requirement Specification

3.1 System Scenario

In the beginning, the user must write his query in the English language. When the user has finished writing his query the system using an artificial intelligence model converts this query to Structured Query Language (SQL) with help of the schema of the dataset and the ontology. When the conversion process has been finished, the system using an AI model should pass the SQL to retrieve the appropriate answer from the database. The system formats the answer to make it more understandable. Finally, the system displays the formatted answer to the user.

3.2 Functional requirements

- Allow the user to enter the query in the English language.
- Convert user query to SQL.
- Integrate with the SQL database.
- Retrieve the appropriate answer from SQL database.
- Formatting the answer.
- Display the formatted answer to the user.

3.3 Non-functional requirements

- The system interface shall be clear and easy to use.
- The system shall be available 24 hours 7 days a week.

3.4 Data requirements

Since the Chat Center Framework Based NLP shall be work in different domains. Therefore, two databases were merged with each other to obtain the largest possible amount of data, both of them have the same type which is a text-to-SQL dataset. The two datasets which have been merged are SparC and CoSQL.

3.5 Software requirements

- Python: a programming language with strong libraries in NLP and AI.
- Google Colab.
- MS Visual studio code.

4 Design

4.1 Initial Flowchart for query answer

As is shown in figure 10 the system gets the schema from the database first, then the system creates the history buffer to store the answered query with its SQL into the buffer. After that the user enters the query into the system, then the system checks if this query has been written in English syntax or not. The system displays an error message for the user if the query was written in a different language and asks him to enter the query again. Else, the system checks if this query in the history buffer if the query already exists the system generates an intelligent response to be shown to the user. Otherwise, the system determines the pattern from the ontology, and then the system generates the SQL by using the user query, schema, and patterns. After that the system passes the SQL to the database, then the system checks whether this SQL is correct or not. If the SQL was incorrect the system asks the user if wants to switch to call service or not? If the user agreed, then the system gets user feedback then switches the user to the call service and stops the program. But if the user does not agree to switch to a call service the system displays a message that asks the user to rephrase the question again. After taking the user query the system checks if this query has been written in English syntax or not. The system displays an error message for the user if the query was written in a different language and asks him to enter the query again. Else, the system replaces the old user query with this query to generate the SQL and do the same steps again if the SQL was incorrect. But if the SQL was correct from the first try the system returns the proper answer from the database. Then the system formatting the answer and save the query and the SQL into the history buffer. After that, the system displays the formatted answer to the user. the system asks the user if he/she wants to exit the system. if the answer of the user was no the system asks the user to enter the query and do the same check query steps. Then the system uses this query with schema and pattern to generate the SQL and repeat the same steps. But if the user wants to exit the program! the system gets feedback from the user and clears the results history buffer. Finally, stop the program.

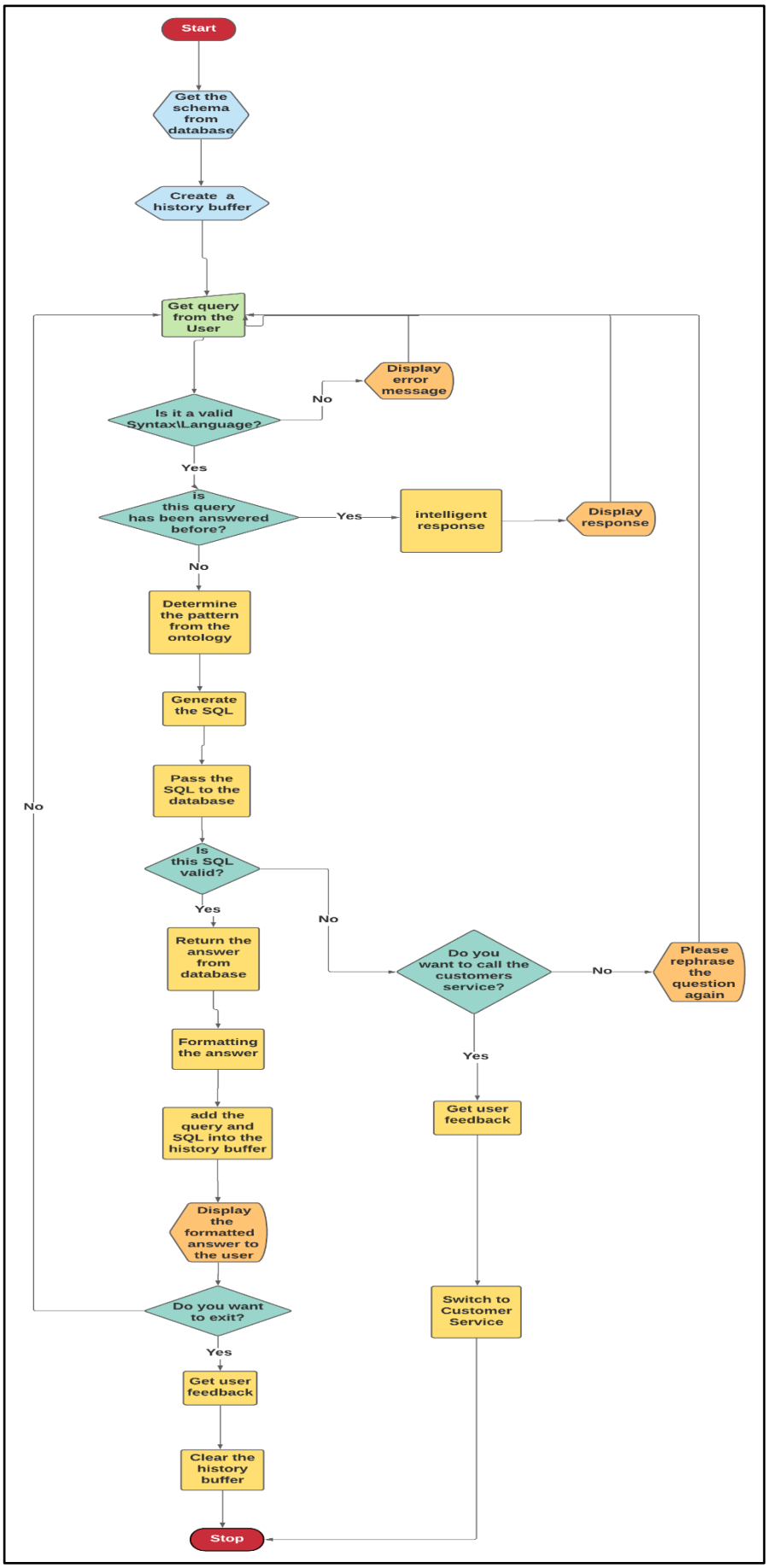


Figure 10 shows the initial flowchart for query answer

4.2 Class diagrams

As figure 11 shown there are three classes include Chat Center UI, SQL Generator, and Chat Center Database. The main class is Chat Center UI which responsible to take the user query to be passed to SQL Generator class and display the formatted answer to the user. The second class is the SQL Generator which responsible to generate the appropriate SQL to be passed to the Chat Center Database class and formatting the answer to be returned to the chat center UI class. the last class is the Chat Center Database which holds all information, this class responsible to return the database schema to be used to generate the SQL and it is responsible to return the appropriate answer from the database.

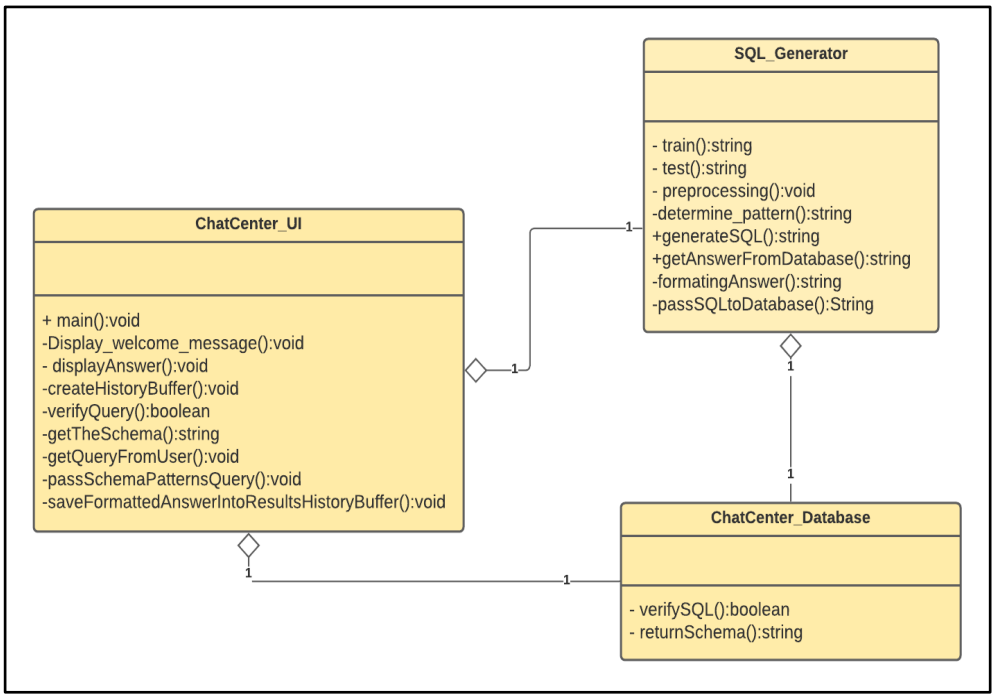


Figure 11 shows the classes and the relation between them

4.3 Sequence Diagram

As is shown in figure 12 the user opens the website application. Then the Chat Center UI gets the schema from the Database first, after that the Chat Center UI creates the history buffer to store the query and the SQL. After that the chat center UI asks user to enter the query into the system. After the user entered the query the chat center UI checks if this query has been written in English syntax or not and . The chat center UI displays an error message for the user if the query was written in a different language and asks him to enter the query again. Else, the chat center UI pass the user query and the schema to the SQL Generator. The SQL Generator determines the pattern and then the SQL Generator generates the SQL. After that the SQL Generator passes the SQL to the Database, then the Database checks whether this SQL is correct or not. If the SQL valid return the answer from Database. Else, the Database returns error message which contain couldn't find table name or column name to the SQL Generator which returns the error message that contain Invalid SQL statement to the chat center that informs user that SQL Generator couldn't understand the query. After returned the answer, the SQL generator formatting the answer and save the query and SQL in the history buffer. After that, the Chat Center UI displays the formatted answer to the user.

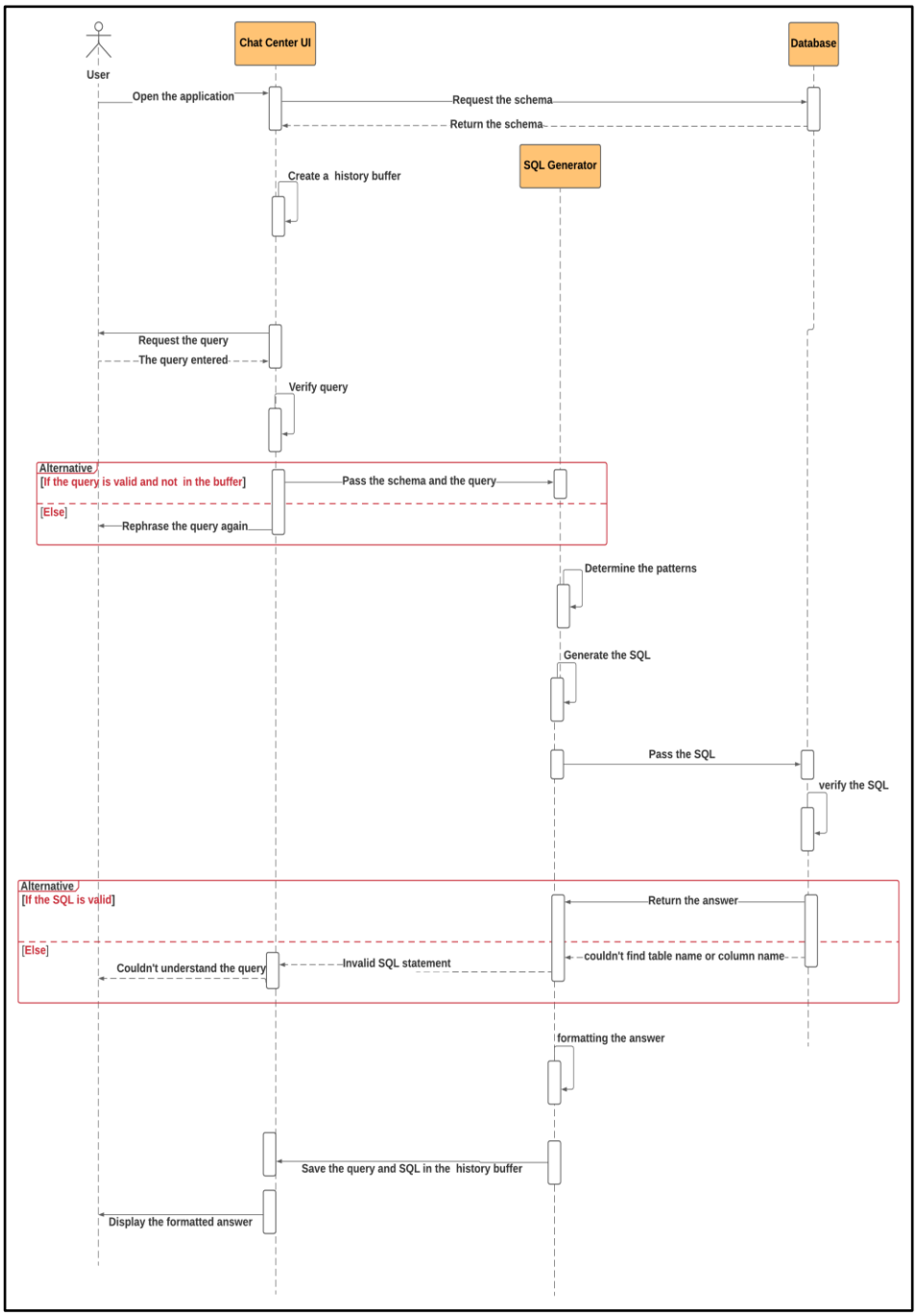


Figure 12 Shows the initial sequence diagram

4.4 Prototype

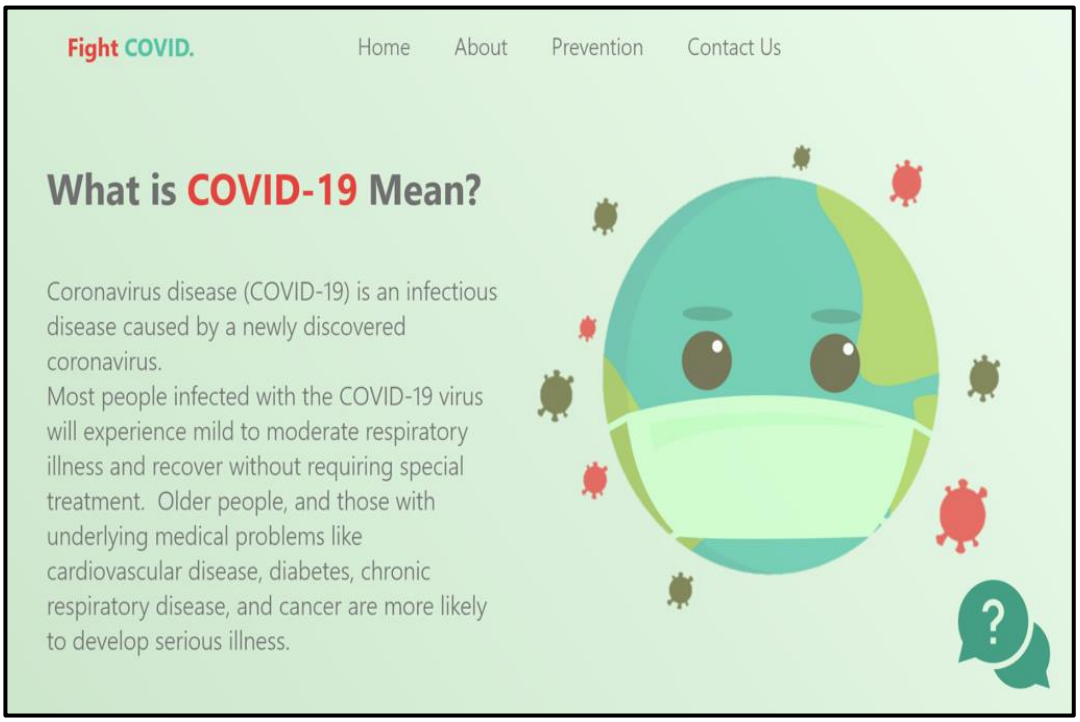


Figure 13 this figure shows the chat center icon and also it shows that the chat center can integrate with any website in any domain.

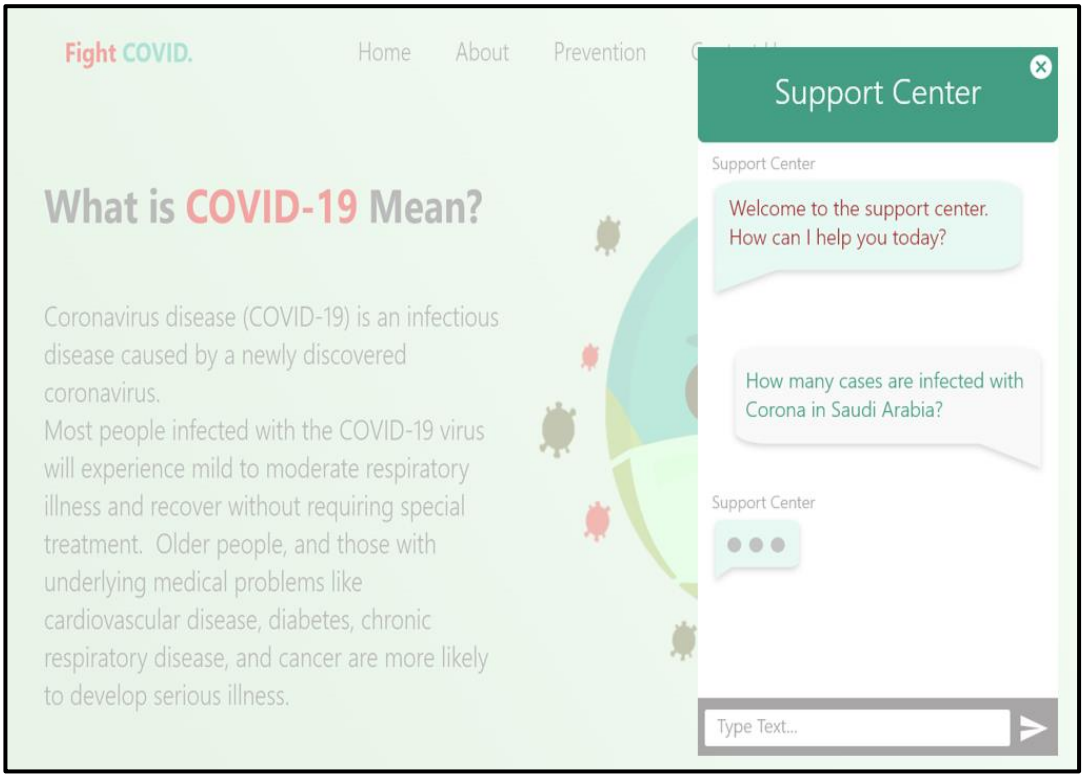


Figure 14 shows the chat appearance, the greeting message, a user query, and how the chat center processing the answer.

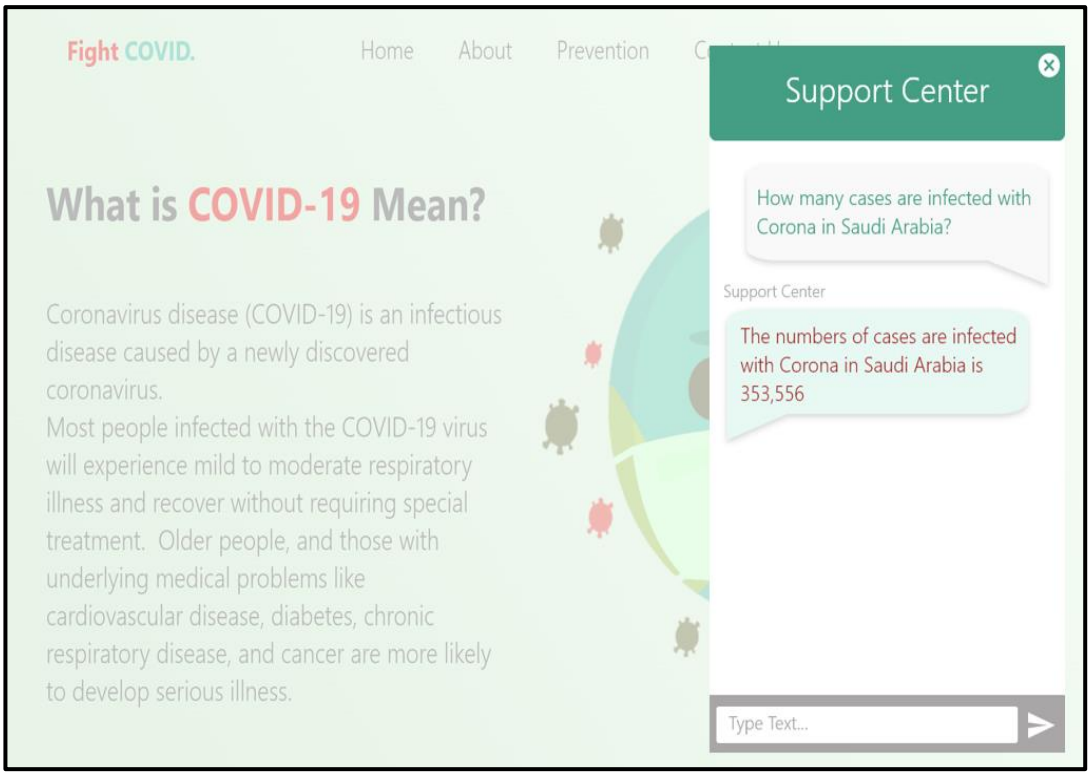


Figure 15 shows the chat center response.

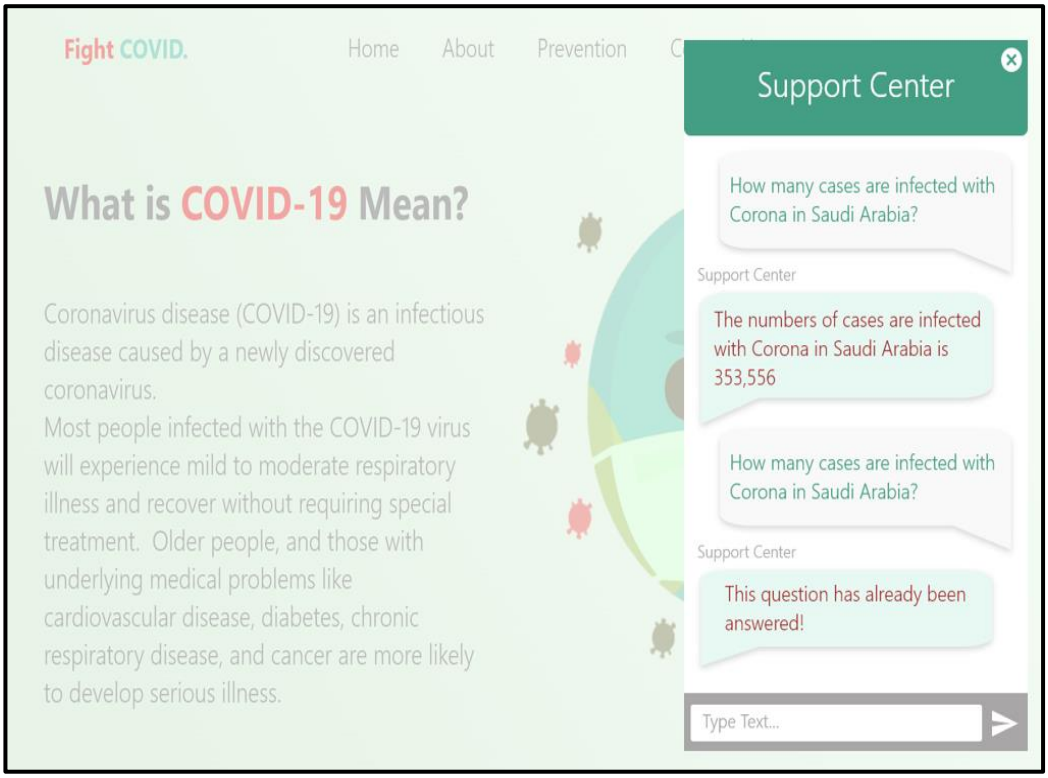


Figure 16 shows the case that when the user repeats the same question and how the chat center will deal with it.

4.5 The software or modelling tools:

- Adobe XD desktop application.
- Lucid website.

4.6 Near future work

We can divide our near future work into two parts:

- **Knowledge part.**

  Where we get to study more about the project field and gain more information about it.

- **Practical part.**

  Get more practice with python and learn how to write proper programs in it. Also, get more experience with Generative Pre-Training (GPT2) Model and learn how to use it.

5 Implementation

5.1 Tools and technologies:

- Python: Easy to use and has powerful libraries that deals with AI. And it has a large supporting community.
- Google colab: Provides free resources such as RAM, GPU, CPU, and storage.
- Gpt-2: A text generating model which will be used to generate the SQL and the response.
- GPT_2_simple: An API to control the GPT-2.
- Flask: Building a website with python as a back end.
- JS/ CSS/ HTML: Controlling the layout, exchanging data between front/back end using json, building website, and styling the GUI.
- SQLite: connect python to a database and it supports all the major databases formats.

5.2 Interface:

A GUI was devolved to demonstrate how the framework could be integrated into a website.

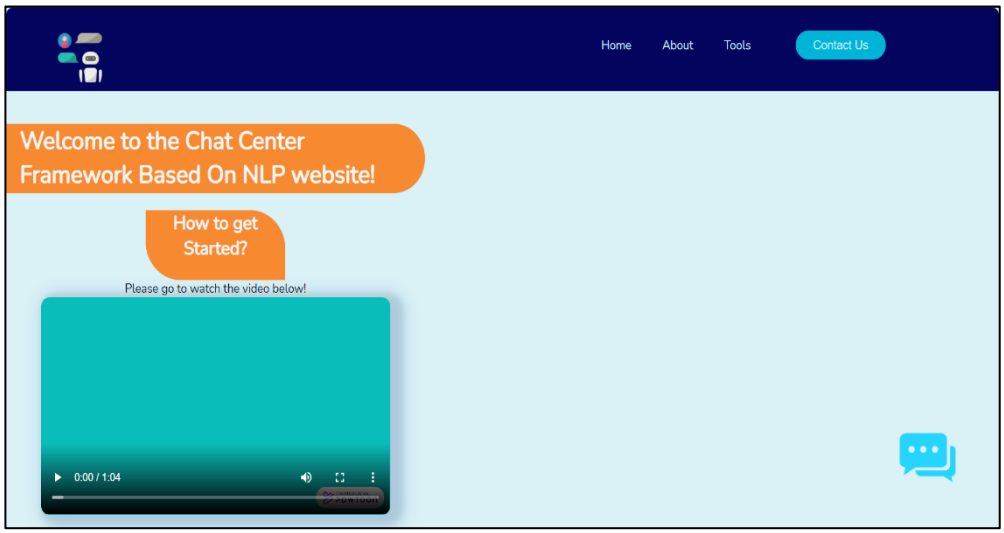


Figure 17 This figure shows the chat center icon

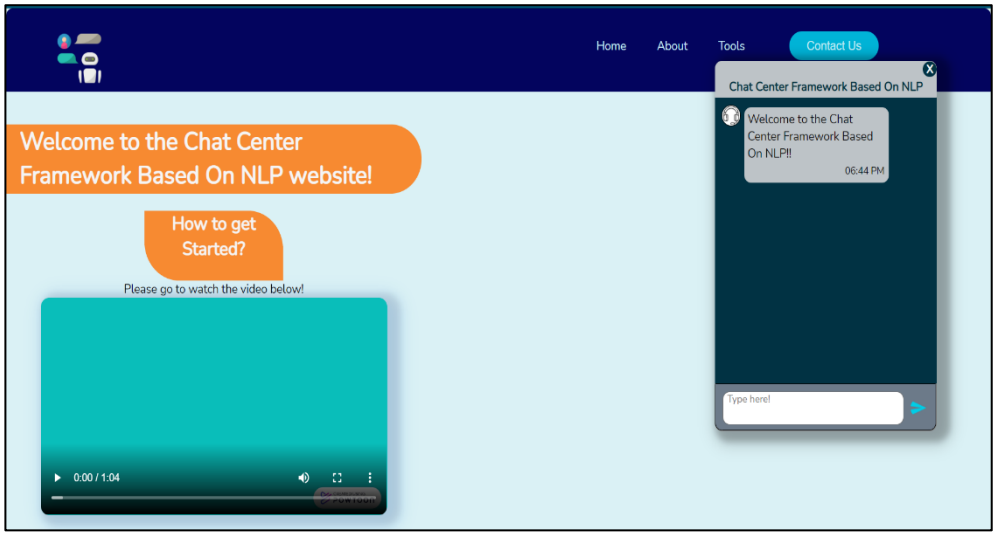


Figure 18 This screenshot shows when user Click on the icon.

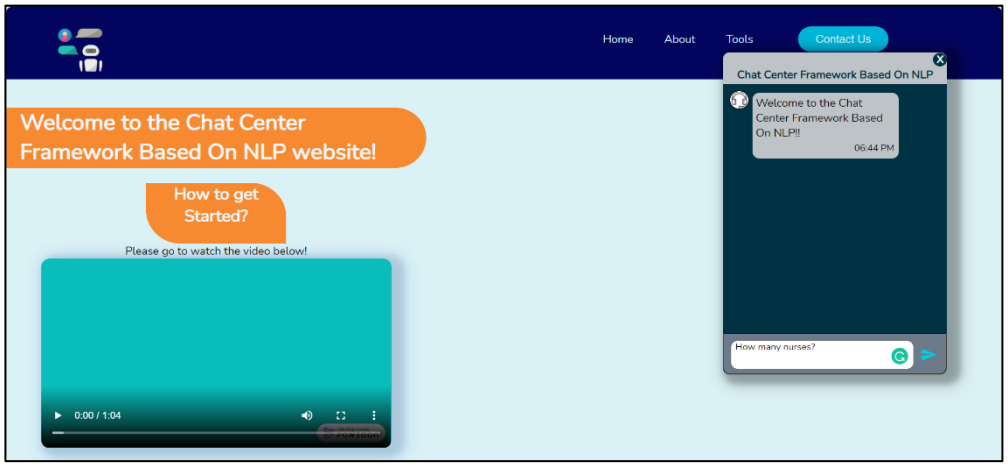


Figure 19 Press the send button to send your message to the server.

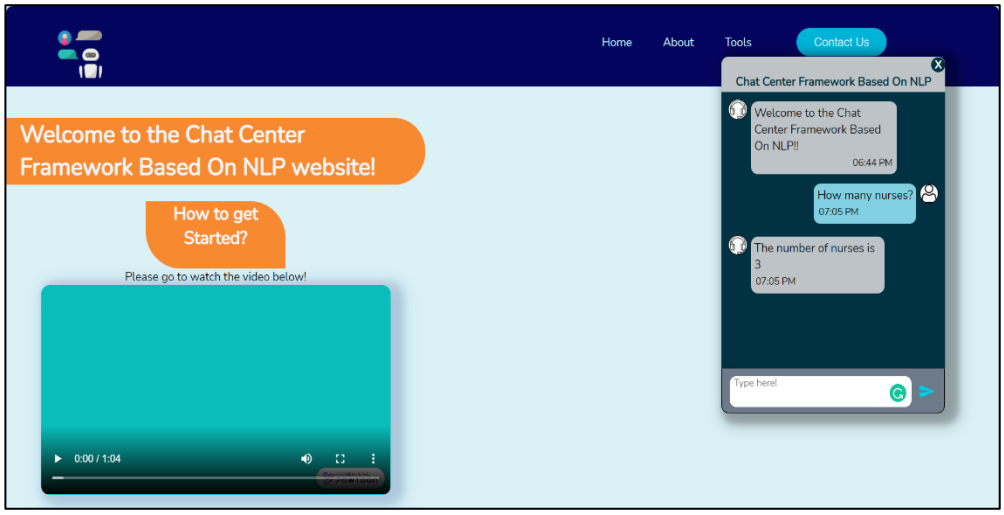


Figure 20 An answer to your question will be displayed in the left side.

5.3 How to get started?

1- Click on the logo at the right button to start the app.

2- Write your question which is related to the database.

3- Press the send button to send your message to the server.

4- Wait for the response.

5- An answer to your question will be displayed in the left side.

5.4 snippets of code:

- Back-End code

```python
import gpt_2_simple as gpt2
from database import DB
import random
import tensorflow as tf
import os
from datetime import datetime
from google.colab import files
import string

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}

class Framework(object):

    def __init__(self, db_path):
        self.db= DB(db_path)
        self.db_schema= self.db.get_schema()
        self.model_name= "124M-V4-10k"
        self.train_file=''
        self.memory=[]#store previos (Q,SQL,ANS) tuples
        self.int_res=["I gave the answer already!","Check my last replay","The answer did not change since last t
        self.sess = gpt2.start_tf_sess()

    def close(self):
        """
        close connection to database
        """
        self.db.close_connection()

    def print_memory(self):
        """
        print the content of memory list.
        used for debuging.
        """
        print("#################### Memory Content ##########################")
```

```python
def fetch_answer(self,question):
    """
    return an answer from database base on a natural language question.
    """
    hazim_check(question)#make sure question is in EN
    #phase 1
    #remove pancuation
    question_clean=question.translate(str.maketrans('', '', string.punctuation))
    if len(self.memory)!=0:

        for block in self.memory:#search memory
            if block[0] == question_clean.lower():
                #match is found
                sql_ans= self.db.get_result(block[1])
                if len(sql_ans)==0:
                    raise Exception("69")

                if sql_ans == block[2]:#ans did not change
                    #iniligent responce
                    return self.int_res[random.randint(0,len(self.int_res)-1)]
                else:#ans changed
                    return f"{block[3]} {sql_ans}"
```

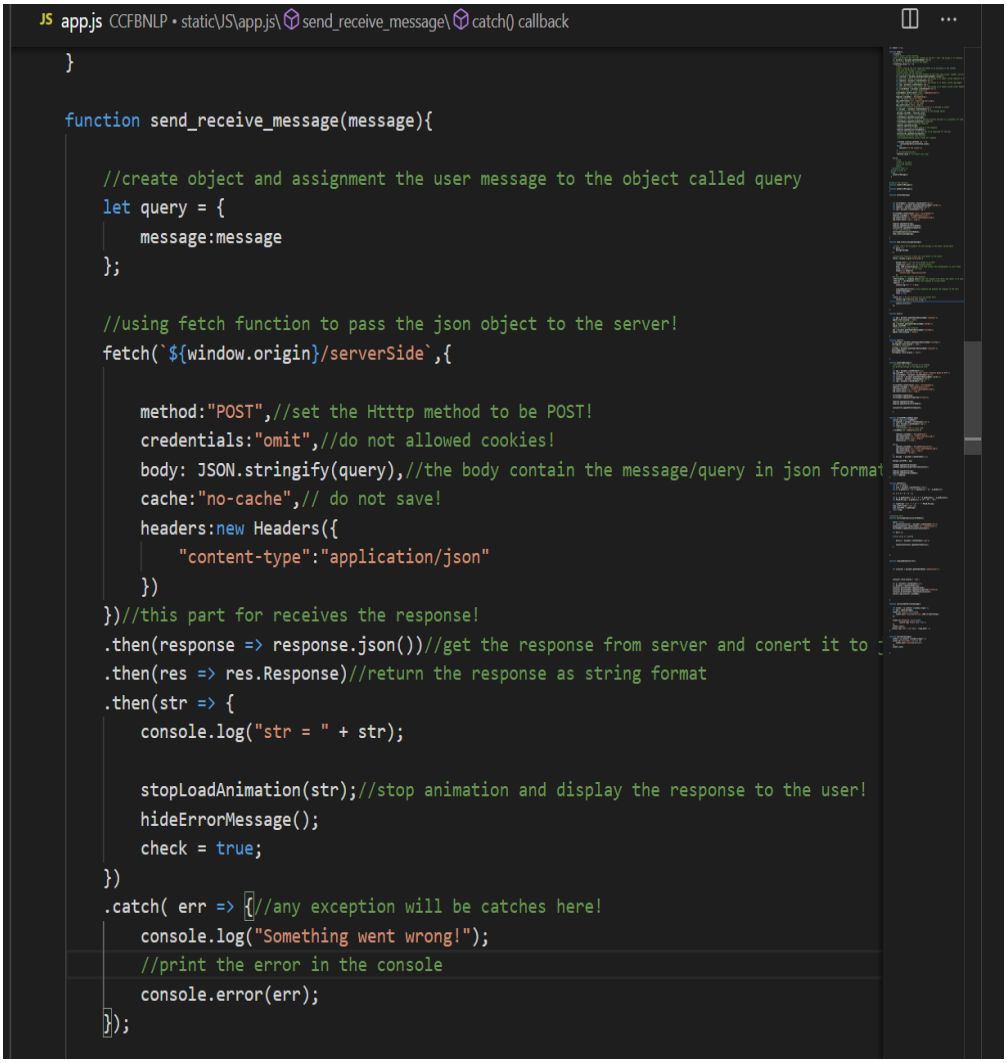
```
#phase 2
#load model and get SQL
gpt2.load_gpt2(self.sess, run_name=self.model_name)
ans= gpt2.generate(self.sess, run_name=self.model_name, truncate='<|endoftext|>', prefix=self.db_schema+question, return_as_list=True)[0]
#extract SQL
sql=ans.split("A: ")[1].split("\n")[0]
#res=ans.split("R: ")[1].split("\n")[0]

result=self.db.get_result(sql)

self.memory.append((question_clean.lower(), sql, result))#save data to memory
tf.get_variable_scope().reuse_variables()
if len(result)==0:
    raise Exception("69")

answer=f""

for an in result:
    for i in an:
        answer+=f"{i}, "
    answer+=f"\n"

return answer
```

- Front-end code

```
<!--App section-->
<div class="startApp" onclick="start()">
    <img src="static/img/logo2.png" alt="app logo!">
</div>
<div class="container">

    <header>
        <button class="exit" onclick="exit()">
            <strong>X</strong>
        </button>
        <br>
        <h4 >
            Chat Center Framework Based On NLP
        </h4>
    </header>

    <div class="waitMessage">


        <p id="error">Please wait for the response!</p>

    </div>

    <div class="chatBox">

    </div>
    <div class="footerChatBox">
     <textarea name="text" id="text" cols="32"
     rows="1" placeholder="Type here!"></textarea>

        <button onclick="send()" id="send">
            <img src="static/img/send.png" alt="send button">
        </button>

    </div>

</div>
```

```
var isTheServerFree = true;


function send(){
    //Check if the server is busy or not!
    if(isTheServerFree){
    //the server is ready to serve!
    //create object called textArea,
    //And from document get the html element by the Id = 'text' and assign it to textArea.
    let textArea = document.getElementById('text');
    //check if the textArea value is not empty!
    if(textArea.value !== ''){
        //Then
        /*1- start creating the user image and bubble to be displayed in the chatBox
          2-create object called chatBox,
              And from document get the html element by the class name called 'chatBox',
              and assign it to chatBox object.
          3-send the message to the server.*/
        let chatBox = document.getElementsByClassName('chatBox');

          let speachBubble = whoIsTheSender('Client',textArea.value);
         let template = SpeashBubbleMaker(speachBubble);
         //append the tempalte to the chat box to be displayed for the user.
         addMessageToChatBox(template,chatBox);

         //invoking the server side function
         serverSide(speachBubble.message,chatBox);//wait for response
         textArea.value = "";//cleare text area!

    }else{
        //Else
        //there is no query
        //exit the function!
        return;
    }//end of inner if
  }//end of outer if
  else{
    //the server is busy!
    //display Error message!
    showErrorMessage();
  }


}//end of the function!
```

```

  function showErrorMessage(){
      document.getElementsByClassName('waitMessage')[0].style.display = 'block';
  }
  function hideErrorMessage(){
      document.getElementsByClassName('waitMessage')[0].style.display = 'none';
  }

  function serverSide(msg,chatBox){

      //start load animation
      startLoadAnimation(chatBox);
      send_receive_message(msg,chatBox);


  }
```


app.js CCFBNLP • static\JS\app.js\ send_receive_message\ catch() callback
}
function send_receive_message(message){
//create object and assignment the user message to the object called query
let query = {
message:message
};
//using fetch function to pass the json object to the server!
fetch(`${window.origin}/serverSide`,{
method:"POST",//set the Htttp method to be POST!
credentials:"omit",//do not allowed cookies!
body: JSON.stringify(query),//the body contain the message/query in json format
cache:"no-cache",// do not save!
headers:new Headers({
"content-type":"application/json"
})
})//this part for receives the response!
.then(response => response.json())//get the response from server and conert it to
.then(res => res.Response)//return the response as string format
.then(str => {
console.log("str = " + str);
stopLoadAnimation(str);//stop animation and display the response to the user!
hideErrorMessage();
check = true;
})
.catch(err => {//any exception will be catches here!
console.log("Something went wrong!");
//print the error in the console
console.error(err);
});

```
102
103 function exit(){
104
105     let app = document.getElementsByClassName('container');
106     app[0].style.display = 'none';
107     //clear the chatBox message
108     let chatBox = document.getElementsByClassName('chatBox');
109     //chatBox[0].innerHTML = "";
110     clearChatBox(chatBox);
111     //switch to start app container
112     app = document.getElementsByClassName("startApp");
113     app[0].style.display = 'block';
114
115 }
116
117 function clearChatBox(chatBox){
118     //delete all the chat history!
119     while(chatBox[0].children.length > 0){
120         //keep deleting till the length of the children of chatbox == 0
121         chatBox[0] = chatBox[0].removeChild(chatBox[0].firstElementChild);
122     }
123 }
124
125 function start(){
126    let startApp = document.getElementsByClassName('startApp');
127    //hide the logo!
128    startApp[0].style.display = 'none';
129    //switch to container (chat center framework based on NLP application!)
130    startApp = document.getElementsByClassName('container');
131    //display the greeting Message!
132    greetingMessage();
133    //display the chat center framework based on NLP application (CCFBONLP)
134    startApp[0].style.display = 'block';
135
136
137 }
```

```
function start(){
   let startApp = document.getElementsByClassName('startApp');
   //hide the logo!
   startApp[0].style.display = 'none';
   //switch to container (chat center framework based on NLP application!)
   startApp = document.getElementsByClassName('container');
   //display the greeting Message!
   greetingMessage();
   //display the chat center framework based on NLP application (CCFBONLP)
   startApp[0].style.display = 'block';


}

function greetingMessage(){
    /*The mean job of this function is to display
    the greeting message at the beginning only!
    */
    let msg = document.createElement('p');
    msg.innerHTML = "Welcome to the Chat Center Framework Based On NLP!!";
    let serverBubble = document.createElement("div");
    let container = document.getElementsByClassName('chatBox');
    let template = document.createElement('div');
    let img = document.createElement('img');

    serverBubble.setAttribute('class',"serverBubble");
    template.className = 'messageHolding-server';
    img.setAttribute('src',"static/img/headphone.png");
    img.setAttribute('class','Limg');

    serverBubble.append(msg);
    serverBubble.appendChild(getTime("R-time"));

    template.appendChild(img);
    template.appendChild(serverBubble);

    container[0].appendChild(template);

    }



function getTime(n){
    //this fnction job is to set the time at 12 hours system!
    let date = new Date();
    let time = document.createElement('time');
    let hour = date.getHours() > 12 ? date.getHours() - 12 : date.getHours();

    hour = hour <= 9 ? "0" + hour : hour;

    let minutes = date.getMinutes() <= 9 ? '0' + date.getMinutes() : date.getMinutes();
    let PM_AM_Periods = date.getHours() >= 12 ? 'PM' : "AM";

    let timeFormat = hour + ":" + minutes + " " + PM_AM_Periods;
    time.setAttribute('class',n);
    time.innerHTML = timeFormat;
    return time;
}
```

```
//Animation Part!
function startLoadAnimation(chatBox){

    let serverBubble = document.createElement("div");
    //let container = document.getElementsByClassName('chatBox');
    let template = document.createElement('div');
    let img = document.createElement('img');

    serverBubble.setAttribute('class',"serverBubble");
    template.className = 'messageHolding-server';
    img.setAttribute('src',"static/img/headphone.png");
    img.setAttribute('class','Limg');

    template.appendChild(img);
    template.appendChild(serverBubble);
    chatBox[0].appendChild(template);


    let animationContainer = document.createElement('div');
    animationContainer.setAttribute('id','LoadContainer');
    serverBubble.appendChild(animationContainer);

    let dots= [];

    for(let i= 0; i < 3;i++){

        dots[i] = document.createElement('span');

        animationContainer.appendChild(dots[i]);
    }

    //Make the server busy!
    isTheServerFree = false;
}


function stopLoadAnimation(){


    //get the load animation container to delete it
    let container = document.getElementById('LoadContainer');

    let parent = container.parentElement;
    let grandParent = parent.parentElement.parentElement.removeChild(parent.parentElement);
    console.log('the grand parent is '+ grandParent.className);


}
```

```
function SpeashBubbleMaker(sender){
//create html element tag called 'div' and assign it to object called template to holds the both of image and speach bubble!
let template = document.createElement('div');
//create html element tag called 'div' and assign it to object called img(image)!
let img = document.createElement('img');
//create html element tag called 'div' and assign it to object called client bubble to holds the clent message!
let BubbleSpeash = document.createElement('div');



    //set class name for client Bubble div
    BubbleSpeash.setAttribute('class',sender.className);
    //set class name for template div
    template.className = sender.templateName;
    //set the phath source for image
    img.setAttribute('src',sender.imageSrc);
    //set the class name for the image
    img.setAttribute('class',sender.imageClassName);
    //create html element pragraph and assign it to message as object
    let message  = document.createElement('p');
    //assign the user input from teaxArea to the message object
    message.innerHTML = sender.message;
    //append the message to BubbleSpeash
    BubbleSpeash.appendChild(message);
    //appent the time by invoking a getTime function and give it a prarmeter for time location inside the bubble.
    BubbleSpeash.appendChild(getTime(sender.timeSide));
    //append the image inside the template.
    template.appendChild(img);
    //append the client bubble speach to the template.
    template.appendChild(BubbleSpeash);

    console.log(`from speashBubbleMaker is ${template.className}`);
    //return the tempalte to be append in the HTML
    return template;
}

function whoIsTheSender(senderName,query){
    //Check the sender's identity to apply the proper style!
    if(senderName === 'Client'){
        // create the senderBubbleStyle object to apply the client bubble speash style!
        let senderBubbleStyle = {
            className:'msgBubbleClient',
            templateName:'messageHolding',
            imageSrc:"static/img/user1.png",
            imageClassName:'Rimg',
            timeSide:'L-time',
            message: query
        };
        //return senderBubbleStyle for the client style
        return senderBubbleStyle;
    }//else create the senderBubbleStyle object to apply the server bubble speash style!
    let senderBubbleStyle = {
        className:'serverBubble',
        templateName:'messageHolding-server',
        imageSrc:"static/img/headphone.png",
        imageClassName:'Limg',
        timeSide:'L-time',
        message: query,
    };
     //return senderBubbleStyle for the server style
     return senderBubbleStyle;
}

function addMessageToChatBox(template,chatBox){
    console.log(template.className);
    chatBox[0].appendChild(template);
}
```

```css
body{
    background-color: #bdc3c7;
    font-family: 'Nunito', sans-serif, Arial;
}

.startApp{
    width: 5em;
    height: 5em;
    border-radius: 15px;
    position: fixed;
    top: 80%;
    left: 90%;
    bottom: 0;
    right: 0;

}

.startApp:hover{
    background-color: rgba(129, 157, 184, 0.856);
}
.container{
    text-align: left;
    background-color:  gray;
    border-radius: 10px;
    width: 20em;
    height: 30em;
    display: none;
    position: absolute;
    top: 5%;
    margin-top: 0%;
    margin-left: 72%;
    margin-bottom: 0;
    margin-right: 0;
    box-shadow: 10px 4px 8px 10px rgba(0, 0, 0, 0.2), 10px 6px 20px 10px rgba(0, 0, 0, 0.19);
    z-index: 300;
}
h4{
    text-align: center;
}
```

```css
/*
Application part!
*/
.startApp:hover{
    background-color: rgba(129, 157, 184, 0.856);
}
.container{
    text-align: left;
    background-color:  gray;
    border-radius: 10px;
    width: 20em;
    height: 30em;
    display: none;
    position: absolute;
    top: 10%;
    margin-top: 0%;
    margin-left: 72%;
    margin-bottom: 0;
    margin-right: 0;
    box-shadow: 10px 4px 8px 10px rgba(0, 0, 0, 0.2), 10px 6px 20px 10px rgba(0, 0, 0, 0.19);
    z-index: 300;
}
h4{
    text-align: center;
}
header{
    background-color: #bdc3c7;
    border: 2px solid inherit;
    border-radius: 10px 10px 1px 1px;
    width: inherit;
    height: 3em;
    position: relative;
    margin: auto;
    padding: 0;
    color: #013243;
    padding: 0;
    font-family: 'Nunito', sans-serif, Arial;
}
.chatBox{

    background-color: #013243;
    width: 20em;
    height: 370px;
    position: relative;
    overflow: auto;
    margin: 0;
    box-sizing: content-box;
}


.msgBubbleClient{
    color: black;
    background-color: #81cfe0;
    border-radius: 10px;
    position: relative;
    word-break: break-word;
    font-family: 'Nunito', sans-serif, Arial;
    clear: both;
    padding: 5px;
    margin: 0 0 3% 0;
}
```

```css
.serverBubble{
    color: rgb(10, 10, 10);
    background-color: #bdc3c7;
    border-radius: 10px;
    position: relative;
    word-break: break-word;
    font-family: 'Nunito', sans-serif, Arial;
    clear: both;
    padding: 5px;
    margin: 1% 2% 1% 0%;
    }


.exit{
    width: 20px;
    height: 20px;
    border-radius: 50%;
    float: right;
    background-color: #013243;
    color: white;
    font-family: Arial;
    border: none;
    font-size: 15px;
    margin: 1px 0px 0px 0px;
    font-family: 'Nunito', sans-serif, Arial;
}
.exit:hover{
    background-color: rgb(170, 170, 190);
    color: black;
}

.footerChatBox{
    background-color: #6c7a89;
    width: 319px;
    border: 1px solid black;
    border-radius: 1px 1px 10px 10px;
    height: 61px;
    position: relative;
    top: 0px;
    box-sizing: border-box;
    padding: 0;
    display: flex;
}

#text{
    resize: none;
    margin: 3% 0 0 3%;
    border-radius: 10px;
    font-family: 'Nunito', sans-serif, Arial;
    padding-left: 6px;
    width: 80%;
    height: 70%;
}
#text:focus, #send:focus, .exit:focus, #errorBtn{
    outline: none;
}
#text:focus{
    border: 2px solid #81cfe0;
}

```

```css
#send{
    width: 23px;
    height: 23px;
    position: relative;
    text-align: right;
    margin-top: 6%;
    margin-left: 3%;
    border: none;
    padding: 0;
    border-radius: 50%;
    background-color: inherit;
}

#send img{
    width: 23px;
    height: 23px;
    margin: 0;
    position: relative;
    background-color: inherit;
    border-radius: 50%;
}
.messageHolding{

    float: right;
    display: flex;
    flex-direction: row-reverse;
    border-radius: 10px;
    margin: 1% 3% 0 20%;
    padding: 0%;

}
.messageHolding-server{
    float: left;
    display: flex;
    flex-direction: row;
    border-radius: 10px;
    margin: 3% 20% 2% 3%;
    padding: 0%;
}
 .Rimg{

    width: 25px;
    height: 25px;
    border-radius: 50%;
    border: 1px solid black;
    margin: 0;
    margin-left: 5px;
    position: relative;
    background-color: white;
 }
 .Limg{

    width: 25px;
    height: 25px;
    border-radius: 50%;
    border: 1px solid black;
    margin: 0;
    margin-right: 5px;
    position: relative;
    background-color: white;
 }
```

```
.R-time{
    float: right;
    color: black;
    font-size: 13px;
    margin: 2px;

}
.L-time{
    float: left;
    color: black;
    font-size: 13px;
    margin: 2px;
}

.startApp img{
    width: 5em;
    height: 5em;
    position: relative;
}


.waitMessage{
    position: relative;
    width: 20em;
    color: aliceblue;
    background-color: red;
    margin: 0px;
    padding: 0px;
    display: none;
}

#error{
   text-align: center;
   justify-self: center;
}
```

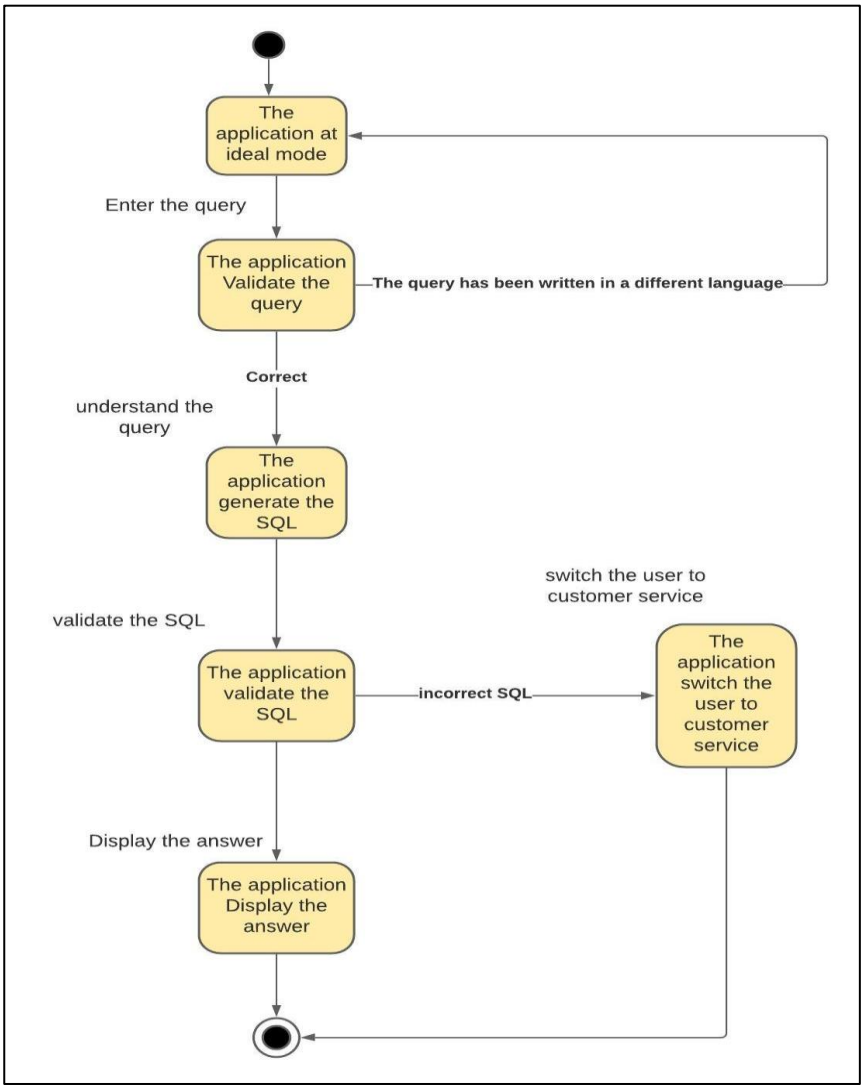
```
/* Animation part*/

#LoadContainer{
    display: flex;
    align-items: center;
    justify-content: center;
    background-color: #bdc3c7;
}
#LoadContainer span{
    height: 15px;
    width: 15px;
    margin: 0 5px;
    background-color: rgb(96, 96, 100);
    border-radius: 50%;
    display: inline-block;
    animation-name: dots;
    animation-duration: 3s;
    animation-iteration-count: infinite;
    animation-timing-function: ease-in-out;
}
#LoadContainer span:nth-child(2){
    animation-delay: 0.4s;
}
#LoadContainer span:nth-child(3){
    animation-delay: 0.8s;
}
@keyframes dots{
    50%{
        opacity: 0;
        transform: scale(0.8) translateZ(20px);
    }
}
```

6 Testing

6.1 Equivalence Partitioning Technique for language entry:

Invalid Partition	Valid Partition
• Any language except English language.	• English language.

6.2 State transition Testing:



6.3 Evaluation

Due to the nature of the used model results are not consisting. This could also be because the smallest model was used for both speed and size constraint.

Examples:

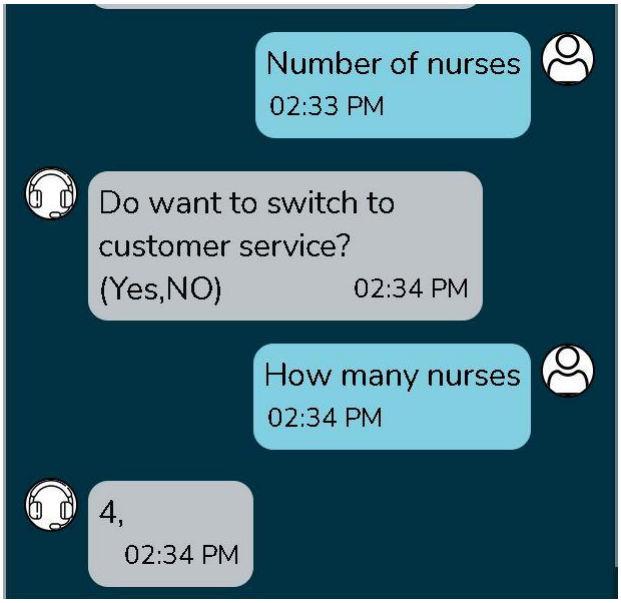


Figure 21 In the above picture shows that asking the same question in a different format can help steer the model to the right output.

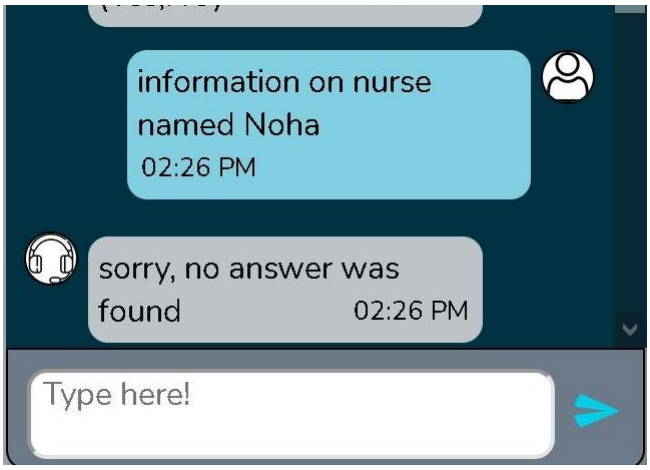


Figure 22 Here the model generated a valid SQL but it does not return an answer even though that Noha does exist in the database.

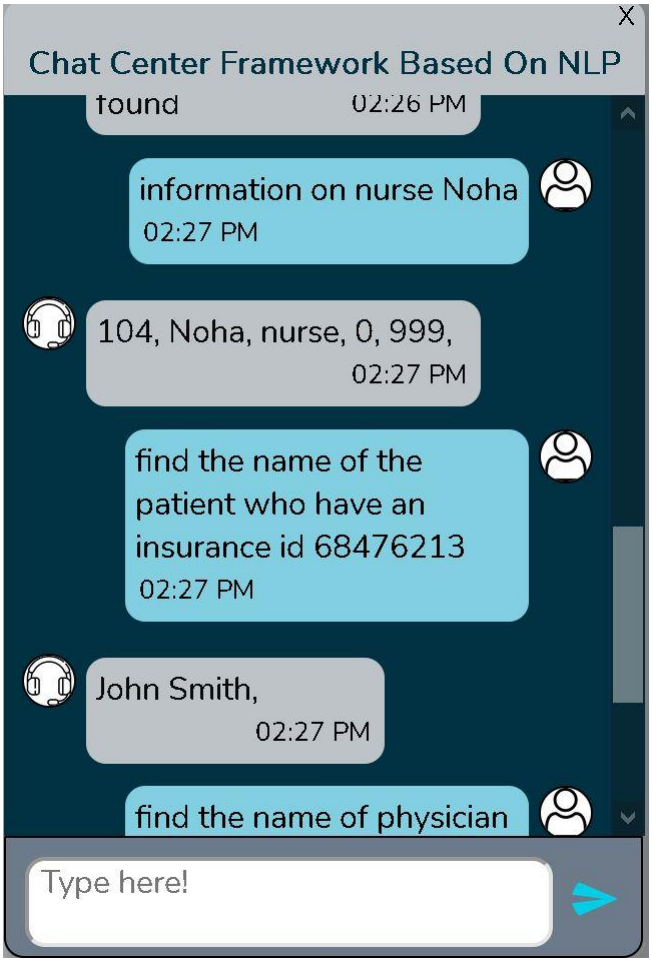
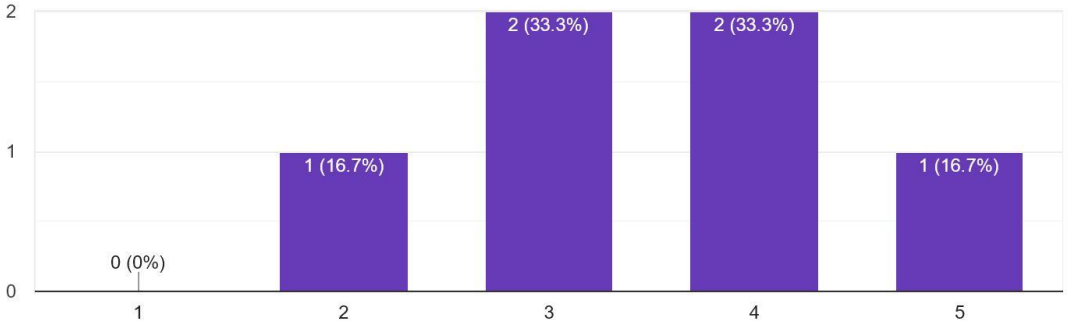


Figure 23 Here the model understood the search conditions and applied them correctly to retrieve the answers.

6.4 Customer Satisfaction:

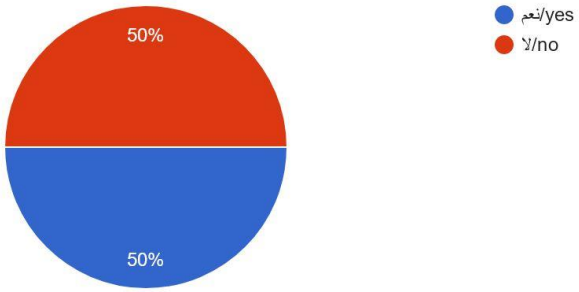
1- are you satisfied? هل انت راضي

6 ردود



2-were all your answers answered? هل تمت الاجابة على جميع اسئلتك؟

6 ردود



6.5 References

[1] Yu, T., Zhang, R., Yasunaga, M., Tan, Y., Lin, X., Li, S., Er, H., Li, I., Pang, B., Chen, T., Ji, E., Dixit, S., Proctor, D., Shim, S., Kraft, J., Zhang, V., Xiong, C., Socher, R. and Radev, D., 2019. Sparc: CrossDomain Semantic Parsing In Context. [online] arXiv.org. Available at: https://arxiv.org/abs/1906.02285.

[2] Guo, T. and Gao, H., 2019. Table2answer: Read The Database And Answer Without SQL. [online] arXiv.org. Available at: https://arxiv.org/abs/1902.04260.

[3] Zhang, R., Yu, T., Er, H., Shim, S., Xue, E., Lin, X., Shi, T., Xiong, C., Socher, R., Radev, D., 2019.Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions. [online] arXiv.org. Available at: https://arxiv.org/abs/1909.00786.