AUTOMATIC QUESTION ANSWERING SYSTEM FOR COLLEGE USE

A PROJECT

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN

COMPUTER SCIENCE ENGINEERING

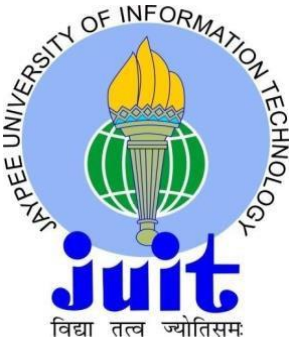
Under the supervision
of

MR. ARIJIT DAS
(ASSISTANT PROFESSOR)

By

GOPAL SINGAL (151353)
RAJAT BAGRI (151379)

to



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
WAKNAGHAT, SOLAN– 173234, HIMACHAL PRADESH INDIA

MAY -2019

STUDENT'S DECLARATION

I hereby declare that the work presented in the Project report entitled **"AUTOMATIC QUESTION ANSWERING SYSTEM FOR COLLEGE USE"** submitted for partial fulfilment of the requirements for the degree of Bachelor of Technology in Computer Science Engineering at **Jaypee University of Information Technology, Waknaghat**is an authentic record of my work carried out under the supervision of **Mr. Arijit Das.** This work has not been submitted elsewhere for the reward of any other degree/diploma. I am fully responsible for the contents of my project report.

Signature of Student

Gopal Singal&Rajat Bagri

Roll No. 151353 & 151379

Department of Computer Science Engineering

Jaypee University of Information Technology, Waknaghat, India

10th May, 2019

ACKNOWLEDGEMENT

I take this opportunity to acknowledge all who has been great sense of support and inspiration throughout the project work successful. There are lots of people who inspired me and helped me in every possible way to provide the detail about various related topics. Thus, making our thesis work a success. Our first gratitude goes to our head of department **Mr. Arijit Das** for his guidance, encouragement and support.

I very grateful to,**Prof. Dr. Satya Prakash Ghrera**head of computer science department for all his diligence, guidance, encouragement and helped throughout the period of project, which has enabled us to complete thesis work in time. I also thank them for the time that they spared for us from their extreme busy schedule. Their insight and creative ideas are always the inspiration for us during the dissertation work.

I am thankful to all the faculty members of the computer science department, for their encouragement during thesis project.

Gopal Singal&Rajat Bagri

(151353 & 151379)

TABLE OF CONTENTS

LIST OF FIGURES

Title **Page No.**

ABSTRACT

Over 2.53 billion people worldwide are using smartphone at this moment and this figure is expected to touch almost 3 billion by the end of next two years. To break this figure even down WhatsApp, one of the leading Instant messenger has 1.3 billion active monthly users and the figure for Facebook messenger is 2.07 billion active monthly users. To put these numbers into perspective 1.324 billion people lives in the world's second most populated country. To tap this big population every information provider is trying to move into that instant messenger you're using at this moment. This will enable it to provide you information in more effective and interactive way without setting up its own infrastructure.

This report deals with the idea of giving 2017 touch up to the existing web kiosk and bringing in even more functionality to it. So, it can add even more value to theexisting system. This report tries to discuss the feasibility of the project, it's architecture, it's scope, it's limitations.

Among many instant messengers we went with the telegram mainly because of the college going youths' inclination toward it and the functionality rich Chabot APIs it has to offer.

Finally, it's working through some screenshots is shown.

1. INTRODUCTION

With the growing functionality and usage of instant messenger, the "College Bot "aims to bring the functionality of the web-kiosk of college to the instant messenger the students use on a daily basis. This would eliminate the need to remember yet another password and bring the required information to the user in an interactive manner.

1.1.0 Existing System:

In the existing system user has to open the browser, go to "www.juit.ac.in" then has to find web-kiosk tab, click on it, wait for it to open and then remember his/her username and password click on login and when it opens user's info. User has to dig through many tabs to find what he's looking for. Technology was invented to make people's life easy not to make them dig through tabs and get frustrated.

1.1.1 Purpose:

- 〉 To eliminate the orthodox way to deal with the information related to student.
- 〉 To make student's phone number password of his web-kiosk.
- 〉 To make the interaction interactive between the user and system.

1.1.2 Scope and Limitations:

- College Bot is designed for colleges, universities, schools.
- There is one to one relation between the roll no. and phone (approved by CITM manually).
- If the database of the student exists in the college then only the bot responds to the query of the student.
- No student can access the details of other student.

1.2 Feasibility Study:

Key consideration:

- **Technical**
- **Behavioural**
- **Economic**

Technical:this assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. As an exaggerated example, an organization wouldn't want to try to put Justice League teleportation system in their building—currently, this project is not technically feasible.

Software:

Front End: Telegram

- Because it is one of the Instant Messenger that offers the functionality to make chat bots.
- One of the most secure Instant Messenger.
- Features offered by telegram are in the line of the uses of students so switching to it would not look like giving up any functionality.

Backend:

- Telegram APIs
- Weka
- R
- JSON
- Python
- SQLite
- Sublime text 3

TELEGRAM: Telegram Messenger is an informing application that works over the web, much the same as WhatsApp or Facebook Messenger.That means you can send messages for free by using a wi-fi connection or your mobile data allowance (providing you have enough data).
As per Telegram, the administration has in excess of 200 million month to month dynamic clients. It propelled in 2013.

Message's primary selling point is security. It asserts every one of its exercises – including talks, gatherings and media – are encoded, which means regardless of whether they are blocked, they won't be noticeable without being deciphered first. In any case, some security specialists have provided reason to feel ambiguous about how secure Telegram is.

Telegram can be utilized on cell phones, tablets, workstations and personal computers. The Telegram application is accessible for Android, iOS, Windows Phone, Windows NT, macOS and Linux.

Telegram APIs

We offer two sorts of APIs for designers. The Bot API enables you to effectively make programs that utilization Telegram messages for an interface. The Telegram API and TDLib enable you to construct your very own tweaked Telegram customers. You are free to utilize both APIs gratis.

You can likewise add Telegram Widgets to your site.

Bot API

This API enables you to associate bots to our framework. Message Bots are uncommon records that don't require an extra telephone number to set up. These records fill in as an interface for code running some place on your server.

To utilize this, you don't have to know anything about how our MTProto encryption convention functions — our go-between server will deal with all encryption and correspondence with the Telegram API for you. You speak with this server through a basic HTTPS-interface that offers a rearranged variant of the Telegram API.

TDLib – construct your own Telegram

Regardless of whether you're searching for most extreme customization, you don't need to make your application starting with no outside help. Attempt our Telegram Database Library (or just TDLib), an apparatus for outsider engineers that makes it simple to assemble quick, secure and include rich Telegram applications.

TDLib deals with all system usage subtleties, encryption and neighborhood information stockpiling, so you can devote more opportunity to structure, responsive interfaces and lovely movements.

TDLib bolsters all Telegram highlights and makes creating Telegram applications a breeze on any stage. It very well may be utilized on Android, iOS, Windows, macOS, Linux and essentially some other framework. The library is open source and good with essentially any programming language

JSON (JavaScript Object Notation): It is a lightweight data exchange position. It is straightforward for individuals to scrutinize and create. It is straightforward for machines to parse and deliver. It relies upon a subset of the JavaScript Programming Language, Standard ECMA-262 third Edition - December 1999. JSON is a substance association that is absolutely language free yet uses demonstrates that are normal to programming architects of the C-gathering of tongues, including C, C++, C#, Java, JavaScript, Perl, Python, and various others. These properties make JSON an ideal data trade language.

JSON depends on two structures:

An amassing of name/regard sets. In various vernaculars, this is recognized as a thing, record, struct, word reference, hash table, keyed summary, or familiar show.

A masterminded summary of characteristics. In numerous tongues, this is recognized as a bunch, vector, summary, or gathering.

These are comprehensive data structures. In every way that really matters all forefront programming lingos reinforce them in some structure. It looks good that a data position that is replaceable with programming lingos moreover be established on these structures.

In JSON, they take on these structures:

An article is an unordered game plan of name/regard sets. A thing begins with { (left help) and completes with } (right prop). Each name is trailed by : (colon) and the name/regard sets are secluded by , (comma).

A bunch is a masterminded aggregation of characteristics. A display begins with [and completions with] . Characteristics are disconnected by,

A regard can be a string in twofold proclamations, or a number, or real or false or invalid, or an article or a bunch. These structures can be settled.

A string is a progression of in any event zero Unicode characters, encased by twofold articulations, using slanted accentuation line escape. A character is addressed as a lone character string. A string is particularly like a C or Java string.

A number is particularly like a C or Java number, of course, really the octal and hexadecimal designs are not used.

Whitespace can be installed between any pair of tokens. In any case, several encoding nuances, that thoroughly portrays the language.

PYTHON: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.Its irregular state worked in data structures, united with dynamic forming and dynamic authority, make it incredibly charming for Rapid Application Development, similarly concerning use as a scripting or glue language to relate existing portions together. Python's basic, simple to learn language structure underlines comprehensibility and in this manner lessens the expense of program support. Python underpins modules and bundles, which empowers program measured quality and code reuse. The Python mediator and the broad standard library are accessible in source or twofold structure without charge for every real stage, and can be uninhibitedly dispersed.

Frequently, software engineers begin to look all starry eyed at Python due to the expanded efficiency it gives. Since there is no aggregation step, the alter test-investigate cycle is unimaginably quick. Troubleshooting Python programs is simple: a bug or terrible information will never cause a division deficiency. Rather, when the mediator finds a blunder, it raises a special case. At the point when the program doesn't get the exemption, the mediator prints a stack follow. A source level debugger permits assessment of nearby and worldwide factors, assessment of subjective articulations, setting breakpoints, venturing through the code a line at any given moment, etc. The debugger is written in Python itself, vouching for Python's reflective power. Then again, frequently the snappiest method to investigate a program is to add a couple of print

proclamations to the source: the quick alter test-troubleshoot cycle makes this basic methodology powerful.

SQLite:SQLite is an in-process library that executes an independent, zero-setup, serverless, value-based SQL database motor. The source code for SQLite exists in the open space and is free for both private and business purposes.

SQLite has ties to a few programming dialects, for example, C, C++, BASIC, C#, Python, Java and Delphi. The COM (ActiveX) wrapper makes SQLite open to scripted dialects on Windows, for example, VBScript and JavaScript, in this manner adding capacities to HTML applications. It is additionally accessible in installed working frameworks, for example, iOS, Android, Symbian OS, Maemo, Blackberry and WebOS in view of its little size and convenience.

SUBLIME TEXT: Great Text is a cross-arrange substance and source code article chief, with a Python application programming interface (API). Radiant Text is exclusive programming. Its usefulness is extendable with modules. The greater part of the broadening bundles have free-programming licenses and are network manufactured and kept up.

- "Goto Anything," speedy route to documents, images, or lines

- "Order palette" utilizes versatile coordinating for speedy console summon of subjective directions

- Multiple choices: at the same time make the equivalent intuitive changes to different chose territories

- Python-based module API

- Project-explicit inclinations

- Extensive adaptability by means of JSON settings documents, including venture explicit and stage explicit settings

- Cross stage (Windows, OS X, Linux)

- Compatible with numerous language syntaxes from Textmate

Operating System: Can be run on any, be it Windows Phone or Android or iOS.

Hardware: Proposed system doesn't require heavy computing power. It can be run on any device having current generation of snapdragon 4xx or above processor and can be developed on any computer running current generation of AMD or intel processors.

Economic Feasibility: this appraisal regularly includes a cost/benefits investigation of the venture, helping associations decide the practicality, cost, and advantages related with a task before money related assets are assigned. It additionally fills in as an autonomous task appraisal and improves venture believability—helping chiefs decide the positive monetary advantages to the association that the proposed undertaking will give.

Cost:

- One-time developer Fee
- Admin who maintains the database and approves the access of a particular roll no. to particular phone

Benefits:

- No need to remember password of kiosk
- Makes the current system contemporary and interactive

Behavioural Feasibility: This can be easily explained with the help of an example

Imagine a Zara store in a remote village of Uttar Pradesh.

Would it have customers?

Most definitely. There would be a number of people who would buy clothes from there and make them a part of their daily wardrobe.

But the general behaviour of the people in the place is not conducive to a shop like Zara which sells 'socially unacceptable' apparel (strictly for a place like the one described above).

Thus, Zara would not like to invest in a store in such a location as the daily sales will not be would not be enough to cover the basic cost of the store and, thus would not be 'feasible' due to the 'behaviour' of the people.

That, right there, is Behavioural Feasibility.

For the proposed product feasibility can be assumed conducive since no person is losing their job, and if a person is operating a computer and using Instant messenger (WhatsApp, Telegram, Hike etc.) then he/she can use it very easily, there is no learning curve.

2. USER REQUIREMENT DEFINITION

The user requirement for this system is to make the system fast, flexible, less prone to error, reduce expenses and save the time.

- Less human error
- Strength and strain of manual labor can be reduced
- High security

3. SYSTEM ARCHITECTURE

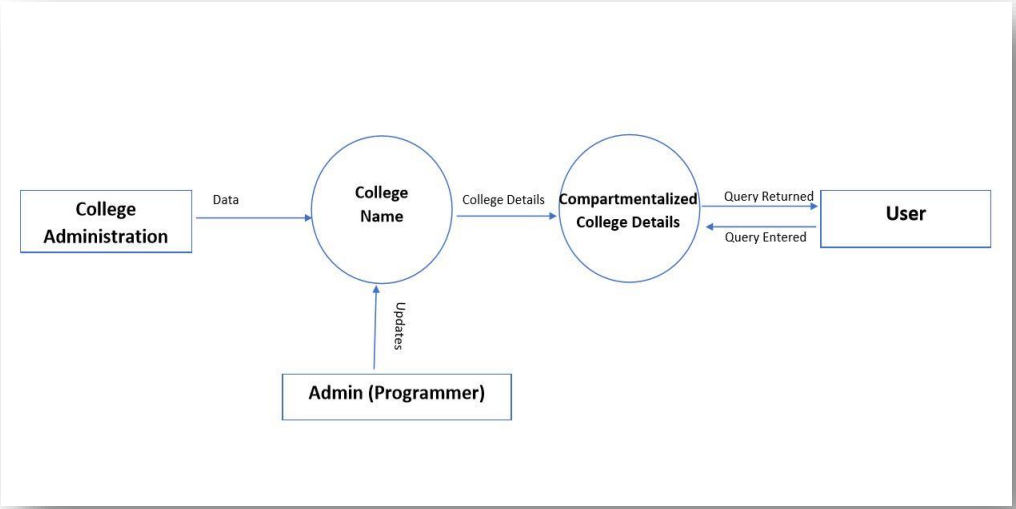


Figure 1

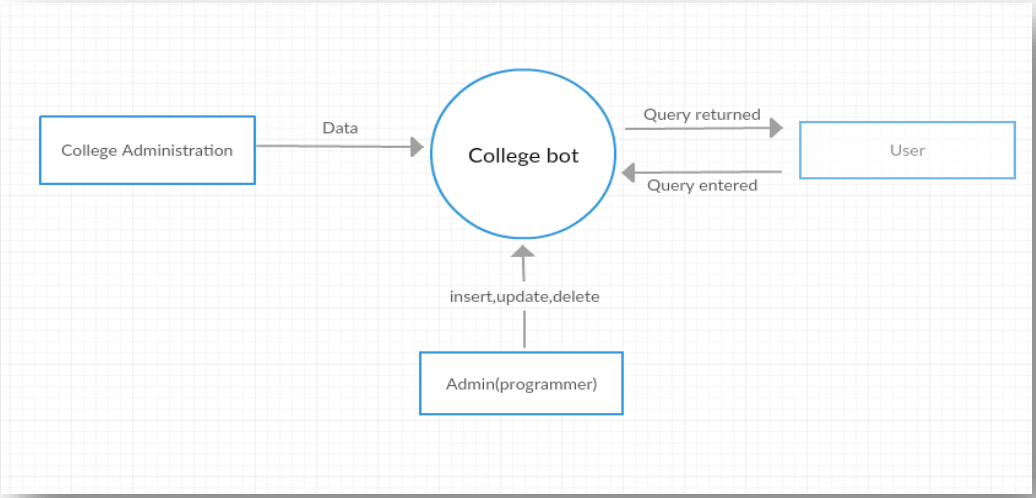


Figure 2

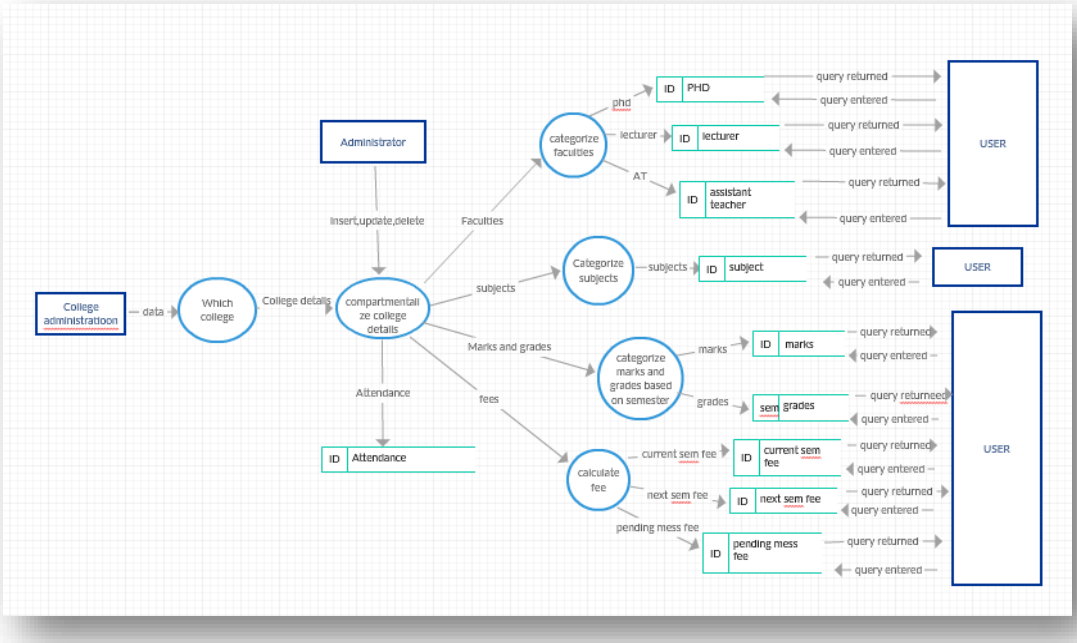

College administratioon
data
Which college
College details
compartmentalize college details
Administrator
insert,update,delete
Faculties
categorize faculties
phd
lecturer
AT
ID
PHD
lecturer
assistant teacher
query returned
query entered
USER
subjects
Categorize subjects
subject
Marks and grades
categorize marks and grades based on semester
marks
grades
sem
Attendance
fees
calculate fee
current sem fee
next sem fee
pending mess fee
current sem fee
next sem fee
pending mess fee

Figure 3

4.SYSTEM REQUIREMENT SPECIFICATION

4.1Functional System Requirement:

〉 Administrator module:

The Administrator can:

1. Manage the whole system
2. Gives permission to a device to link with a roll no.
3. Insert new information to the existing modules
4. Update the particular information in the existing modules
5. Delete particular information from the existing modules

〉 Student/User Module:

Student can

1. Link his/her phone with his/her roll no.
2. Know about the current/previous semester(s) faculties
3. Know about the current/previous semester(s) subjects
4. Know about the current/previous semester(s) marks/grades
5. Know his attendance
6. Know about the current/next semester fee
7. Know about his time table

〉 Subjects module:

It can:

1. Provide information regarding subjects of particular student
2. Provide information regarding previous semester subjects
3. Provide information regarding faculties who is teaching current semester subjects

〉 Faculties module:

It can:

1. Provide information of particular faculty
2. Provide control to edit the details of any particular faculty

〉 Fees Module:

It can:

1. Provide information regarding fee of particular student
2. Provide information regarding fee of next semester.
3. Provide information regarding pending mess fee of the student

〉 Marks module:

It can:

1. Provide information regarding marks obtained in any subject of current semester
2. Provide information regarding marks obtained in any subject of previous semester(s)
3. Provide information regarding grades obtained in any subject of previous semester(s)
4. Tells the current CGPA

〉 Attendance:

It can:

1. Gives attendance of any subject of current semester
2. Notify about the storage of attendance in any subject

4.2 Non-Functional System Requirement:

4.2.1 Performance Requirements

Some Performance requirements identified is listed below:

- The database shall be able to accommodate a thousand record to store.
- The software shall support use of multiple users at a time.
- There are no other specific performance requirements that will affect development.

4.2.2 Safety Requirements

The database may get crashed at any certain time due to virus or operating system failure. Therefore, it is required to take the database backup.

4.2.3 Security Requirements

Some of the factors that are identified to protect the software from accidental or malicious access, use, modification, destruction, or disclosure are described below.
Keep specific log or history data sets
Assign certain functions to different modules
Restrict communications between some areas of the program
Check data integrity for critical variables
Later version of the software will incorporate encryption techniques in the user/license authentication process.

4.3 System Requirement Specification:

4.3.1 Hardware Requirements:

1. Processor: core i3 or greater
2. RAM:4 GB or more
3. Hard drive: 512GB or more
4. Keyboard
5. Monitor
6. LCD

4.3.2 Software Requirements:

1. Python
2. Text editor/IDE like Atom, Sublime or PyCharm
3. Command Prompt/PowerShell
4. Operating System: Microsoft Windows

5.SYSTEM MODEL

In this system we are use waterfall model to apply these ideas. Which is help us to separate each step and when we finish a one phase the output of it is the input to the next phase. Also, we can backwards if there is a new requirement or to apply any update.

Building a database aide :

We'll pursue great coding practice and keep our database-related code separate from the fundamental rationale of our Bot. Any code that contacts the database will be bound to another content, dbhelper.py and we'll bring this content into the fundamental Bot content.

Demonstrating the issue:

At whatever point you have to manufacture a database, the initial step is to ponder precisely how you will speak to the issue. We'll pursue an iterative way to deal with structure this bot, implying that at each progression, we'll influence the issue as straightforward as would be prudent and afterward to tackle that before emphasizing on the task to assemble a more nuanced arrangement.

A schedule comprises of various things with content depictions, for example, "Purchase staple goods", or "Complete the process of composing system instructional exercise". Until further notice, we'll model everything by essentially putting away its depiction in a table in a database. We'll see later on this is an uncommon misrepresentation of the issue, however it'll be adequate to kick us off.

Our System should most likely add another thing to the database of things that it is recollecting for the client, and it will likewise need to erase a thing once the client has denoted that thing as done. Moreover, it'll should probably get every one of the things to show them to the client. At long last, it'll should most likely make the database table, on the off chance that we have to set up another database (for example while moving our bot to another machine). The majority of this usefulness ought to be contained in our dbhelper.py content, as every last bit of it collaborates with the database legitimately.

Writing the code for our Bot

Now we can get to writing Python. Create the file `echobot.py` and add the following code:

```
import json
import requests

TOKEN = "<your-bot-token>"
URL = "https://api.telegram.org/bot{}/".format(TOKEN)


def get_url(url):
    response = requests.get(url)
    content = response.content.decode("utf8")
    return content


def get_json_from_url(url):
    content = get_url(url)
    js = json.loads(content)
    return js


def get_updates():
    url = URL + "getUpdates"
    js = get_json_from_url(url)
    return js


def get_last_chat_id_and_text(updates):
    num_updates = len(updates["result"])
    last_update = num_updates - 1
    text = updates["result"][last_update]["message"]["text"]
    chat_id = updates["result"][last_update]["message"]["chat"]["id"]
    return (text, chat_id)


def send_message(text, chat_id):
    url = URL + "sendMessage?text={}&chat_id={}".format(text, chat_id)
    get_url(url)


text, chat = get_last_chat_id_and_text(get_updates())
send_message(text, chat)
```

We should pull separated what this code does:

In lines 1 and 2, we import the solicitations and json modules. The first is to make web demands utilizing Python and we'll utilize it to interface with the Telegram API (also to what we were utilizing our internet browser for prior). We'll utilize the JSON module to parse the JSON reactions from Telegram into Python word references with the goal that we can separate the bits of information that we need.

The following two lines are worldwide factors, where we characterize our Bot's token that we have to confirm with the Telegram API, and we make the essential URL that we'll be utilizing in the entirety of our solicitations to the API.

The get_url work just downloads the substance from a URL and gives us a string. We include the .decode("utf8") part for additional similarity as this is vital for some Python forms on certain stages. Ordinarily, we'd do some special case taking care of here as this solicitation could fall flat if our web association were down, if Telegram's administration were down, or if there were an issue with our Token. Anyway for effortlessness, here we'll basically expect that everything Always Works (TM).

The get_json_from_url work gets the string reaction as above and parses this into a Python lexicon utilizing json.loads() (loads is short for Load String). We'll generally utilize this one as Telegram will dependably give us a JSON reaction.

get_updates calls similar API order that we utilized in our program before, and recovers a rundown of "refreshes" (messages sent to our Bot).

get_last_chat_id_and_text gives a straightforward yet inelegant approach to get the visit ID and the message content of the latest message sent to our Bot. Since get_updates will dependably send every one of the messages that were as of late sent to our bot, this isn't

perfect, as we will dependably download an entire group of messages when we just need the last one. We'll examine later in more detail how to do this all the more carefully. Until further notice, this capacity restores a tuple of the chat_id which recognizes the particular visit between our Bot and the individual who sent the message, and the content, which is simply the message.

send_message takes the content of the message we need to send (content) and the talk ID of the visit where we need to send the message (chat_id). It at that point calls the sendMessage API direction, passing both the content and the visit ID as URL parameters, in this manner requesting that Telegram send the message to that talk.

The last two lines bring all that we have composed together to really get and communicate something specific. To start with, we get the content and the talk ID from the latest message sent to our Bot. At that point, we call send_message utilizing a similar content that we simply got, successfully "reverberating" the last message back to the client.

The testdb.py code

Create a new file called testdb.py in the same directory as your Chatbot script, and add the following code:

```
import sqlite3


classTestdb:
def__init__(self, dbname=".testdb0sqlite"):
        self.dbname = dbname
        self.conn = sqlite3.connect(dbname)

defsetup(self):
        stmt = "CREATE TABLE IF NOT EXISTS items (description
text)"
        self.conn.execute(stmt)
        self.conn.commit()

defadd_item(self, item_text):
        stmt = "INSERT INTO items (description) VALUES (?)"
        args = (item_text, )
        self.conn.execute(stmt, args)
        self.conn.commit()

defdelete_item(self, item_text):
        stmt = "DELETE FROM items WHERE description = (?)"
        args = (item_text, )
        self.conn.execute(stmt, args)
        self.conn.commit()

defget_items(self):
        stmt = "SELECT description FROM items"
return [x[0] for x in self.conn.execute(stmt)]
```

In this code, we have five strategies:

__init__() takes a database name (naturally store our information in a document called testdb0.sqlite) and makes a database association.

setup() makes another table called items in our database. This table has one section (called description)

add_item() takes the content for the thing and additions it into our database table.

delete_item() takes the content for a thing and expels it from the database

get_items() restores a rundown of the considerable number of things in our database. We utilize a rundown cognizance to take the primary component of every thing, as SQLite will dependably return information in a tuple position, notwithstanding when there is just a single section, so in this precedent every thing we recover from the database will be like ("subject",) (a tuple) which the rundown appreciation changes over to "purchase perishables" (a basic string).

This Testdb class that we've quite recently made can be utilized by our Chatbot to do all that it needs to include, expel, and show things. How about we change the Chatbot code to utilize this new usefulness.

Updating the todobot.py file

The changes here are simpler because of the abstraction provided by testdb. The changes we have to make are in the handle_updates() function, and all we need to do is pass the chat_id along to testdb whenever we call one of the methods we just updated. The new code for handle_updates() looks as follows:

```
def handle_updates(updates):

    for update in updates["result"]:

        text = update["message"]["text"]

        chat = update["message"]["chat"]["id"]

        items = db.get_items(chat)   ##

        if text == "/done":

            keyboard = build_keyboard(items)

            send_message("Select an item to delete", chat,
keyboard)

        elif text in items:

            db.delete_item(text, chat)   ##

            items = db.get_items(chat)   ##

            keyboard = build_keyboard(items)

            send_message("Select an item to delete", chat,
keyboard)
```

```
        else:

            db.add_item(text, chat)   ##

            items = db.get_items(chat)   ##

            message = "\n".join(items)

            send_message(message, chat)
```

The lines where we now have to pass chat along to the testdb have been indicated with ##. The bot should now work with multiple users, assigning each user a new list. Delete the testdb0.sqlite file and start the bot again to recreate the database with the new changes.

5. TEST PLAN

6.1) Diagrams:

6.1.1) Class diagrams:

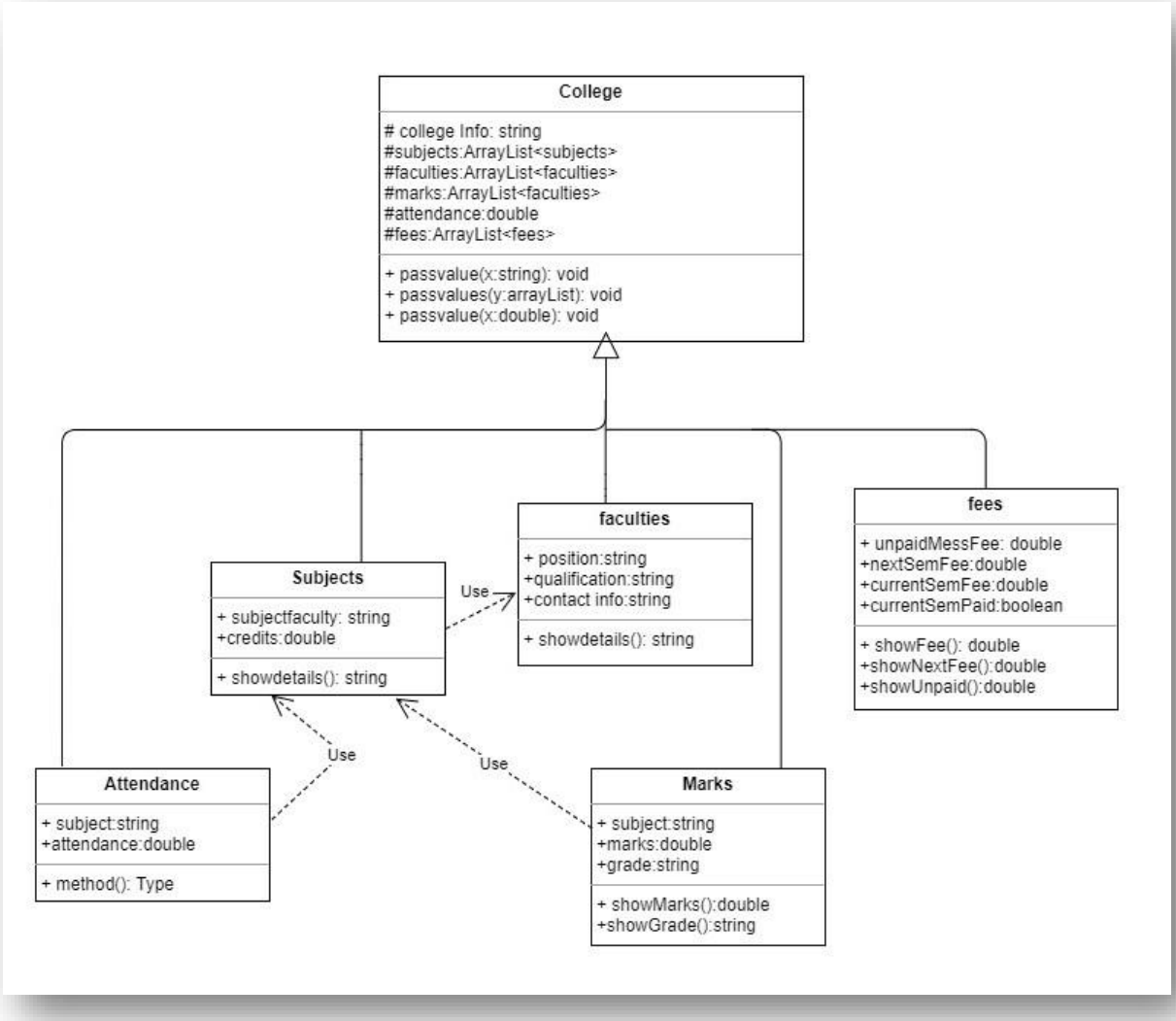


Figure 4

6.1.2) Use Case Diagram:

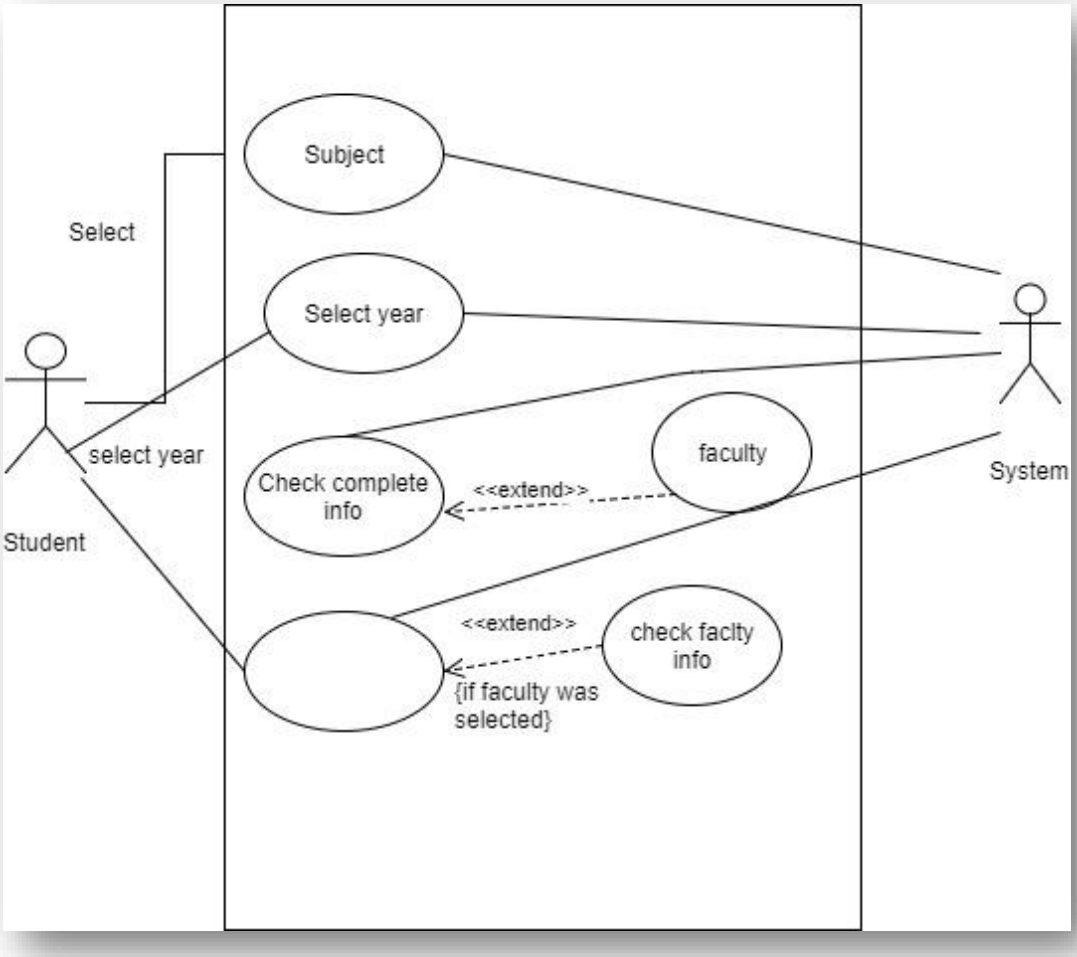


Figure 5

Scenario: when user selects subject option

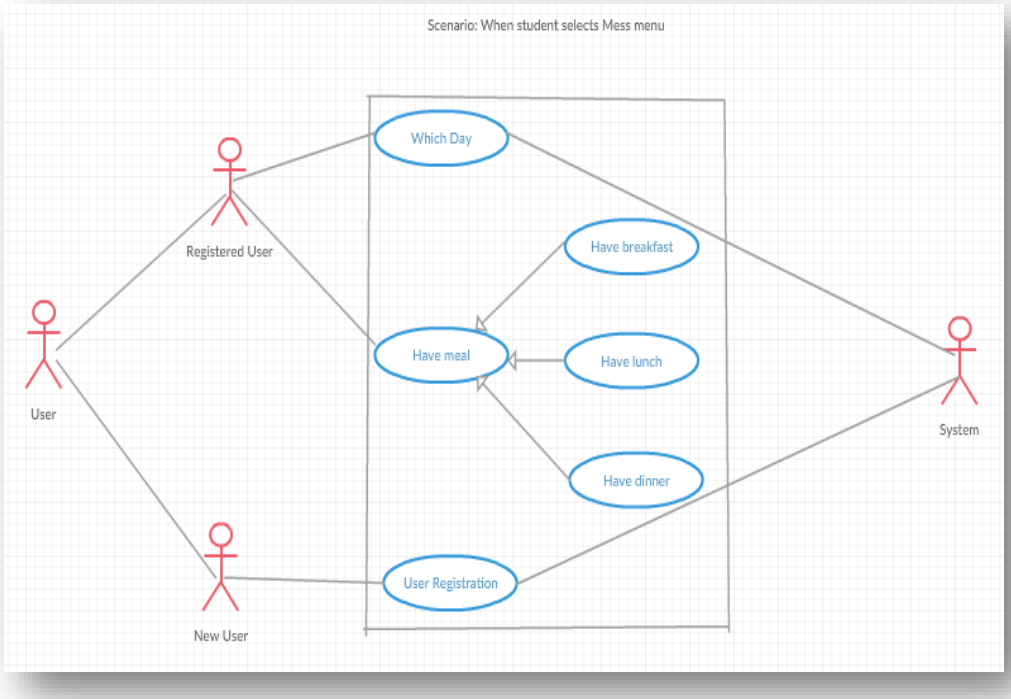

Scenario: When student selects Mess menu
Which Day
Registered User
Have breakfast
Have meal
Have lunch
User
System
Have dinner
User Registration
New User

Figure 6

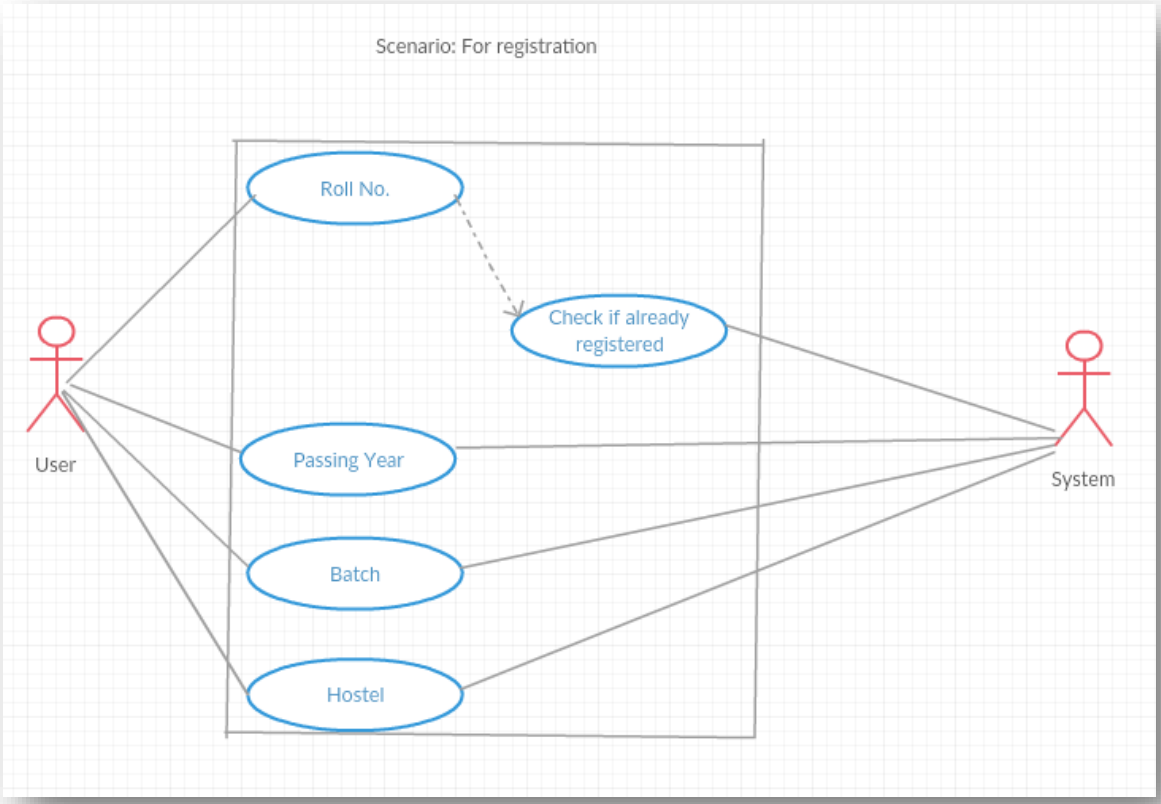


Figure 7

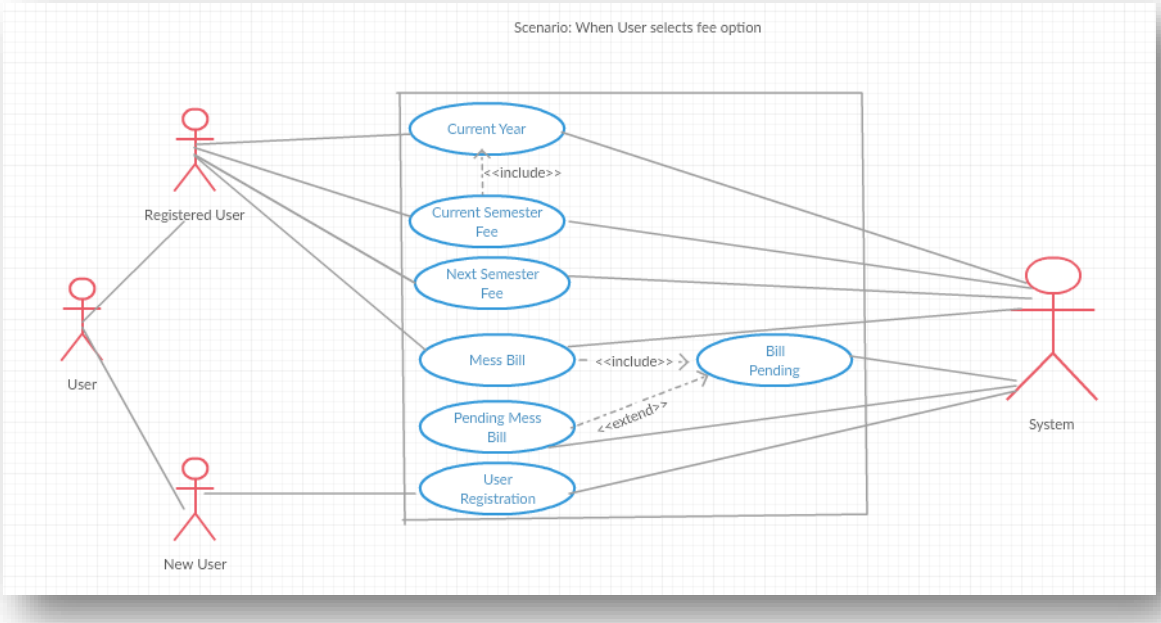


Figure 8

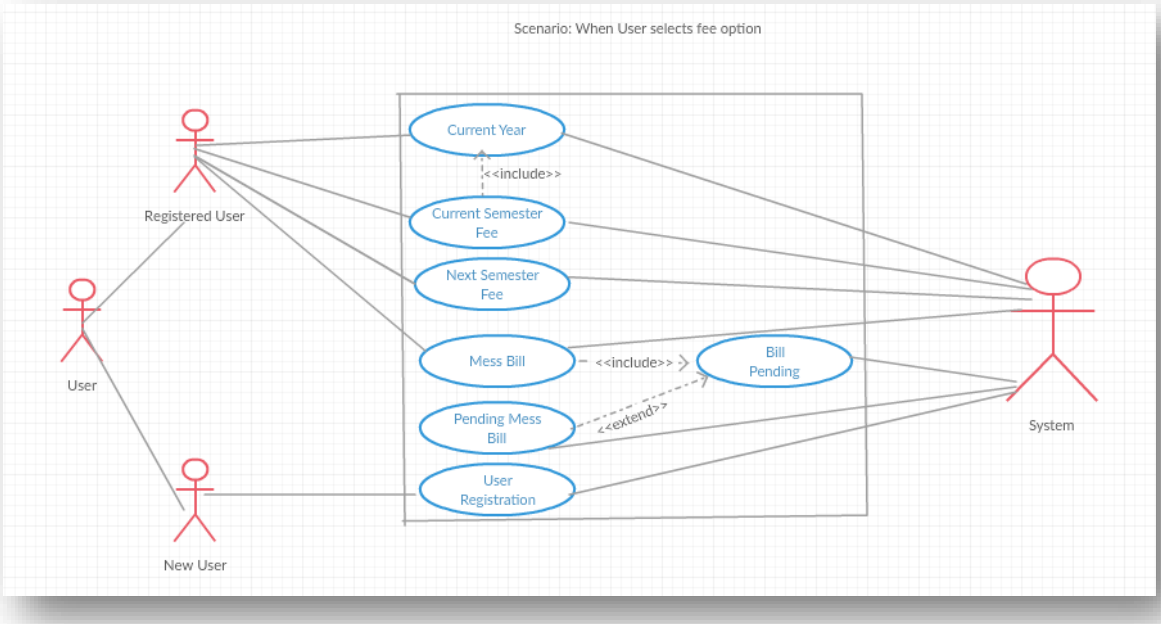

Scenario: When User selects fee option
Registered User
User
New User
Current Year
<<include>>
Current Semester Fee
Next Semester Fee
Mess Bill
<<include>>
Bill Pending
Pending Mess Bill
<<extend>>
User Registration
System

Figure 9

6.1.3) Activity Diagram:

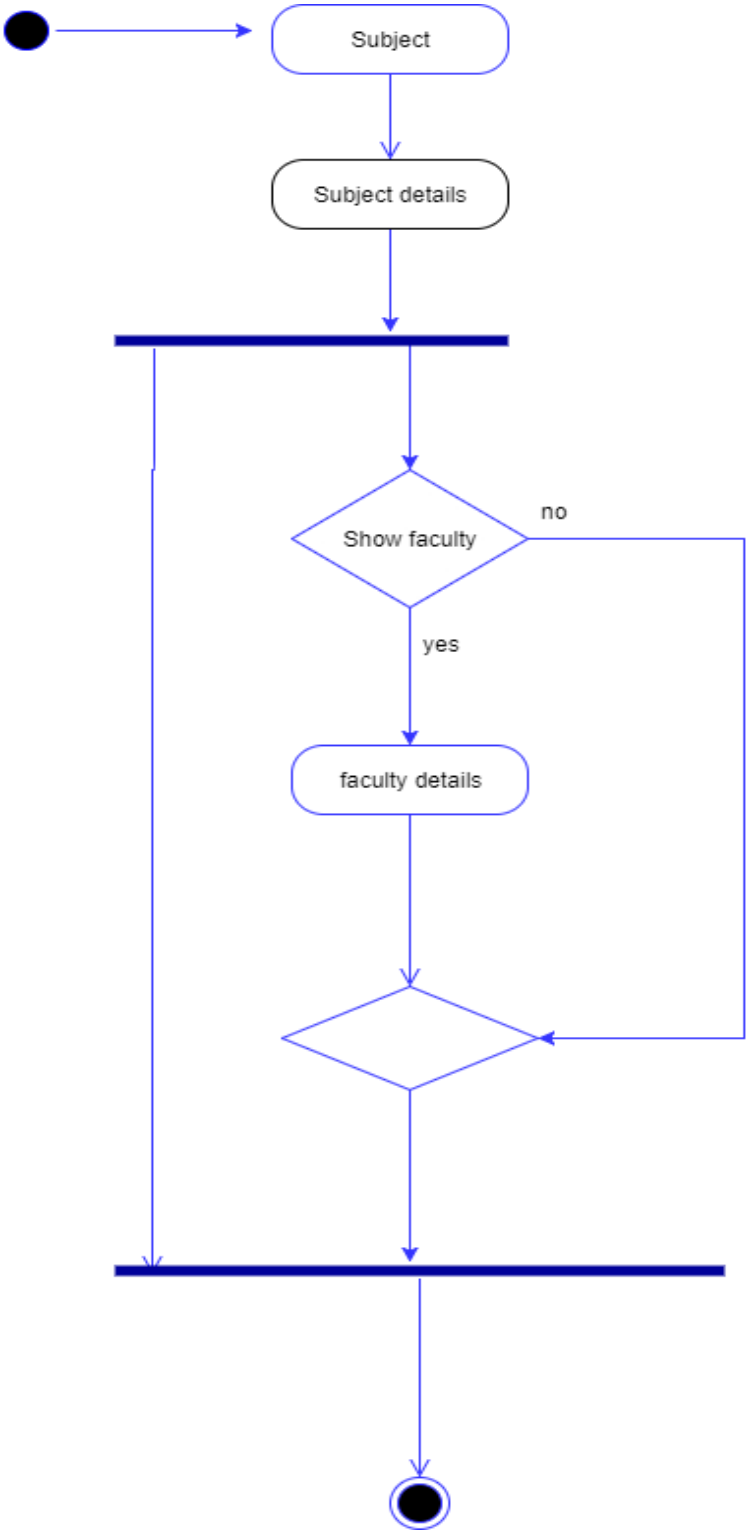


Figure 10

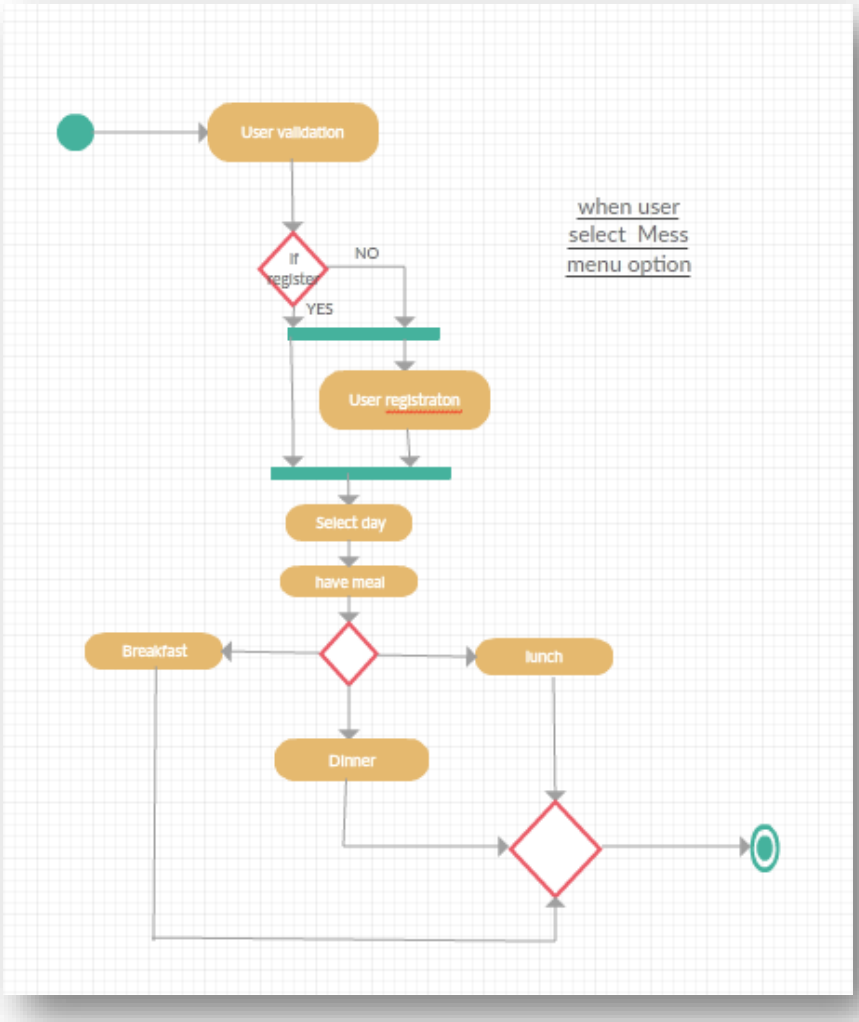

User validation
when user
select Mess
menu option
If
register
NO
YES
User registraton
Select day
have meal
Breakfast
lunch
Dinner

Figure 11

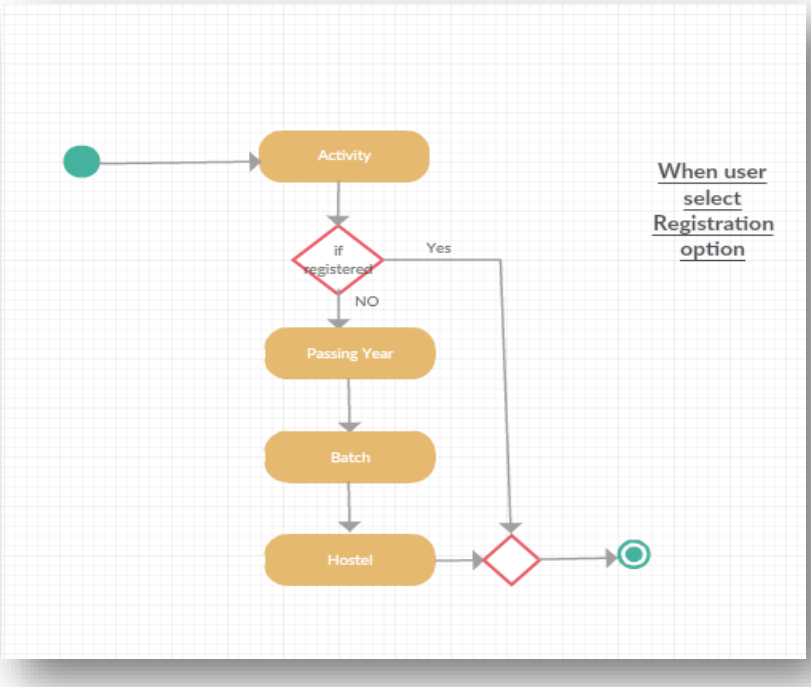


Figure 12

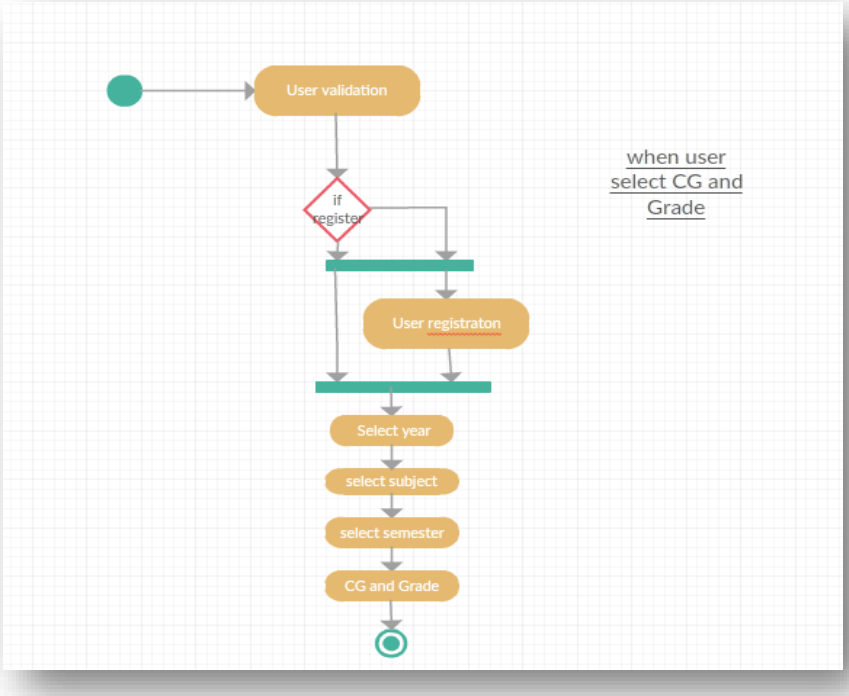


Figure 13

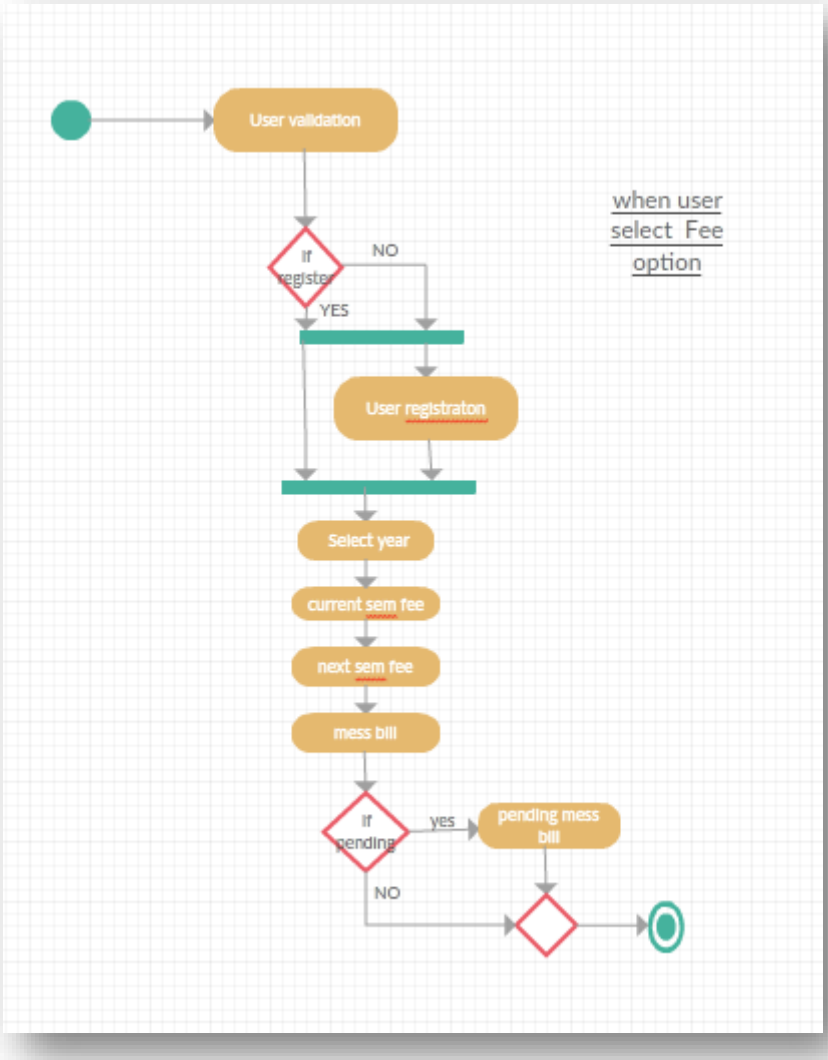

User validation
when user select Fee option
If register
NO
YES
User registraton
Select year
current sem fee
next sem fee
mess bill
If pending
yes
pending mess bill
NO

Figure 14

6.1.4) ER Diagram:

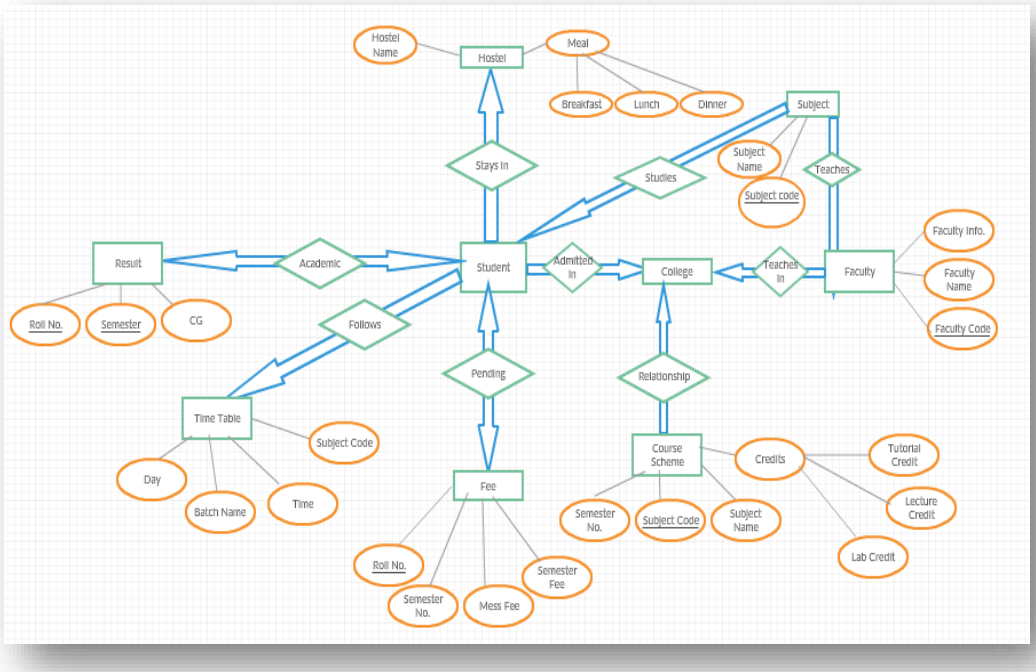


Figure 15

6.2) Screenshots:

6.2.1) Screenshots of the test cases:

Finding bot on Telegram:

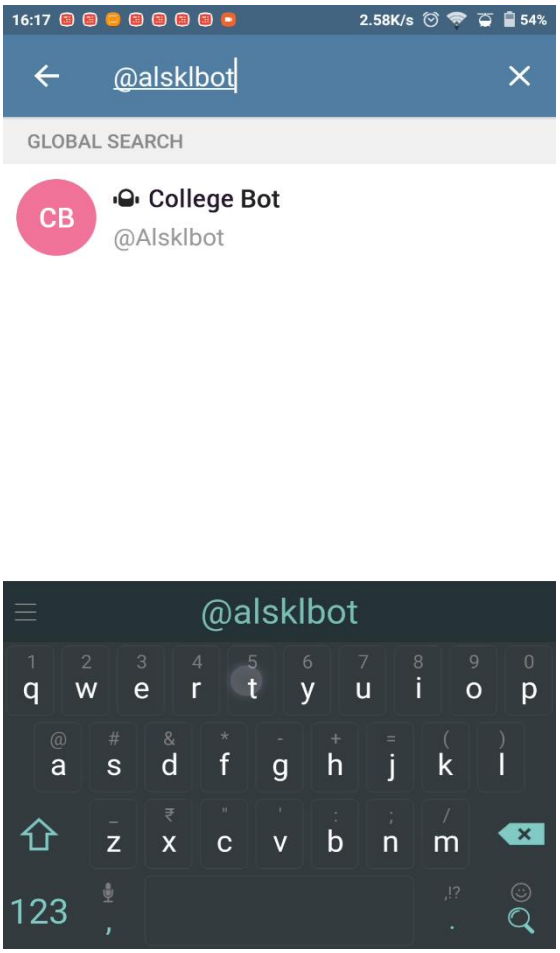


Figure 16

Registration Process

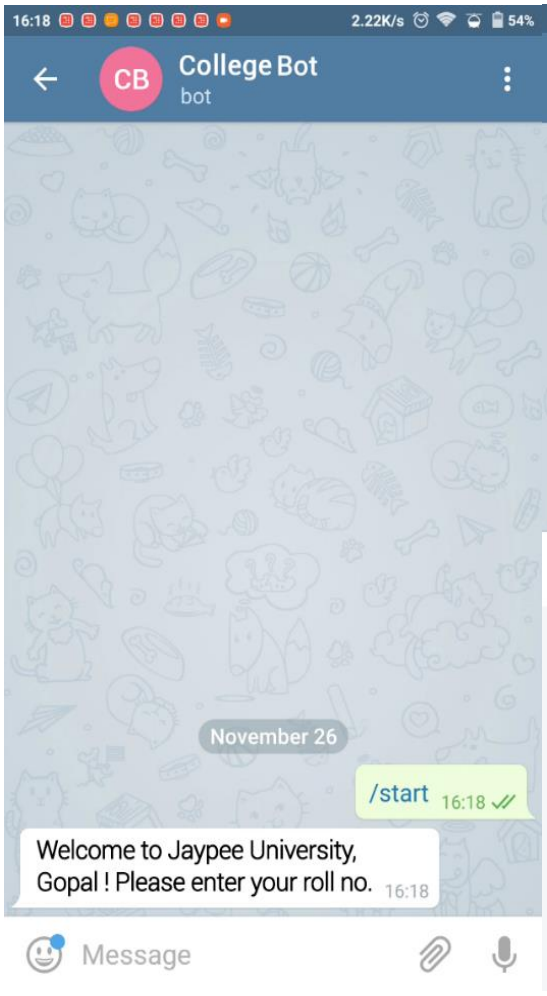


Figure 17

Figure 18

Options Available:

Figure 19

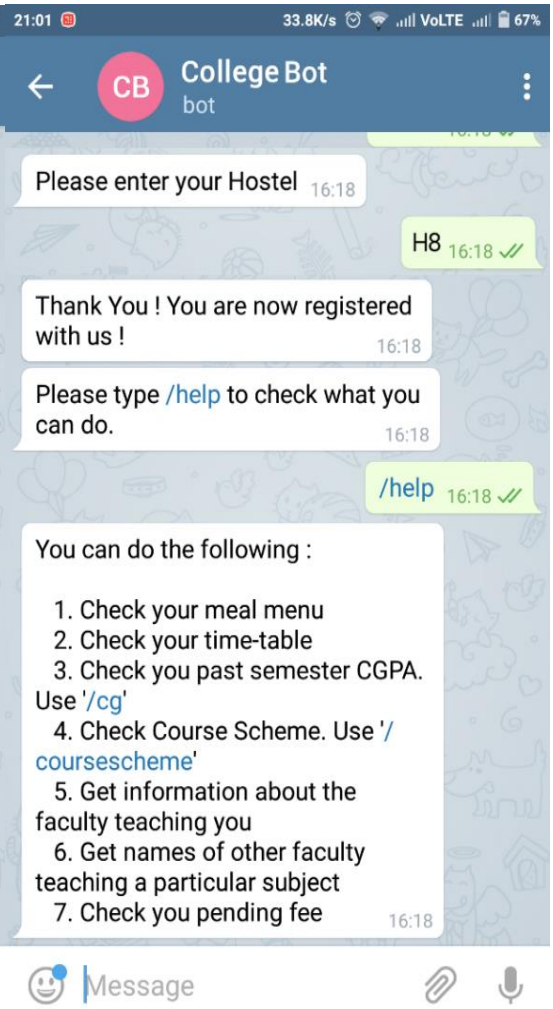


Figure 20

Using Tags and Telegram Keyboard:Unknown Input:

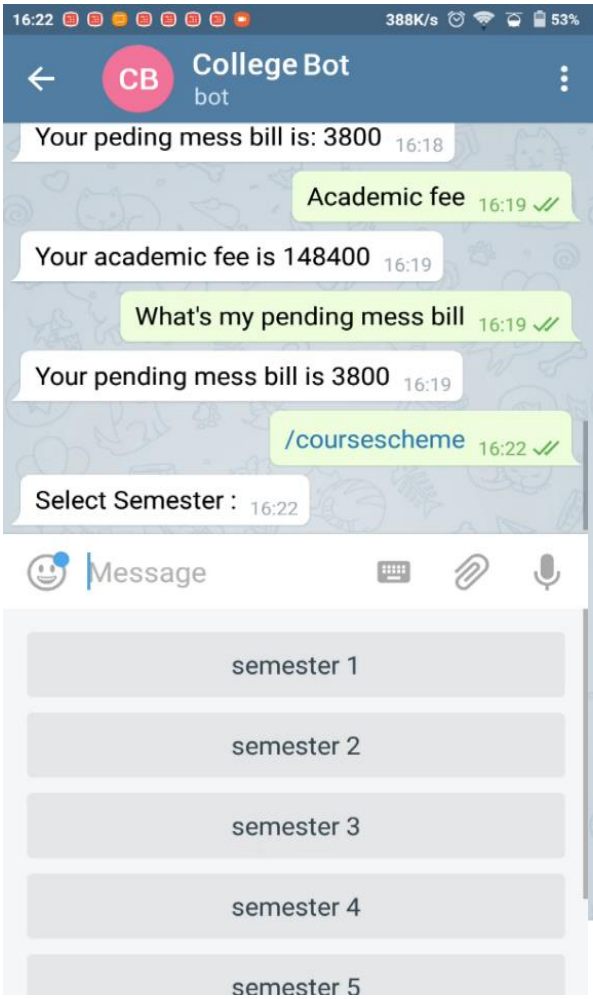


Figure 21

Figure 22


21:03
57.3K/s
VoLTE
65%
CB
College Bot
bot
16:28
I'm not sure, what to do with this ! 16:28
Hey 16:28
hie 16:28
What's your name? 16:29
My name is Alskl bot ! :D 16:29
What's your age? 16:29
I am 18 days old! 16:29
When were you born? 16:30
I was conceptualised on November 12, 2018. 16:30
Message

Figure 17

6.2.2) Screenshots from the Database:

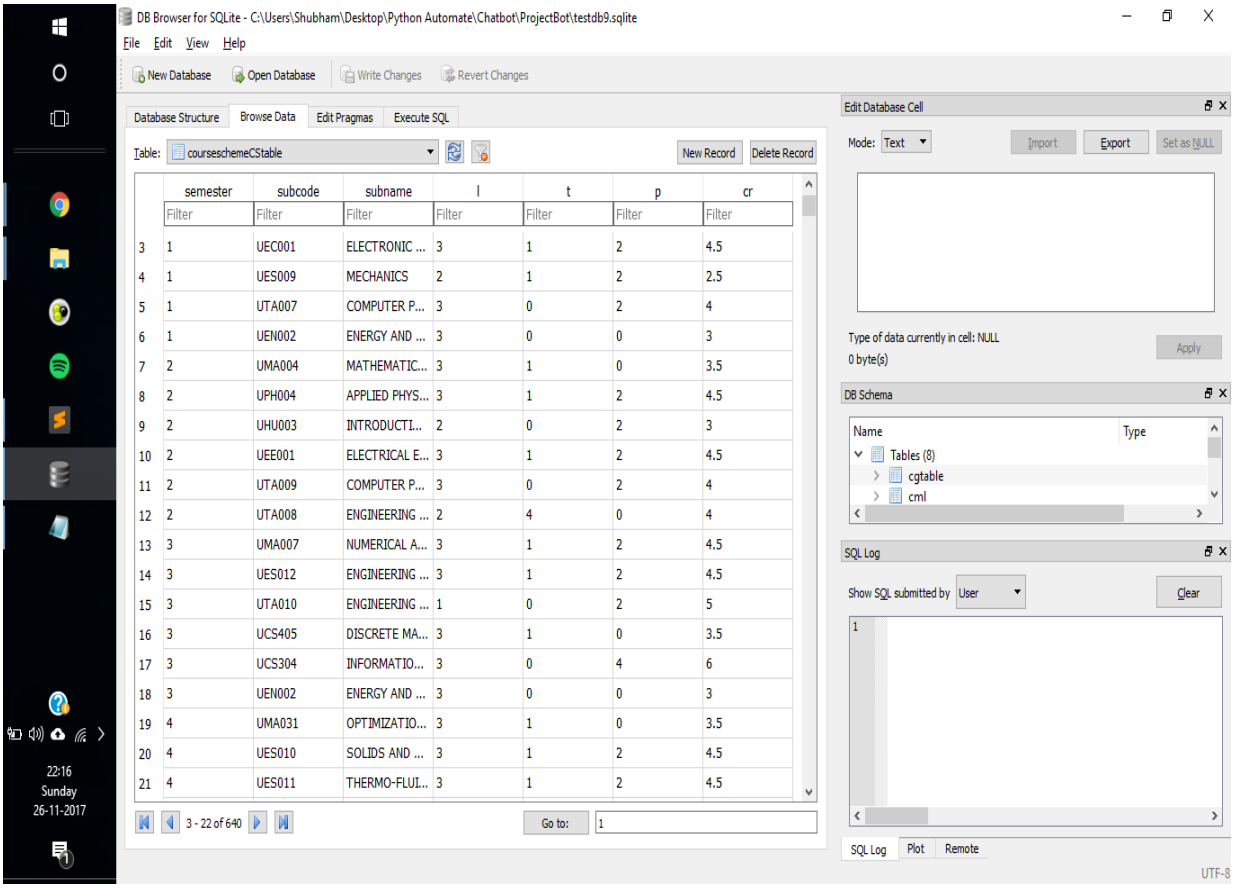


Figure 18

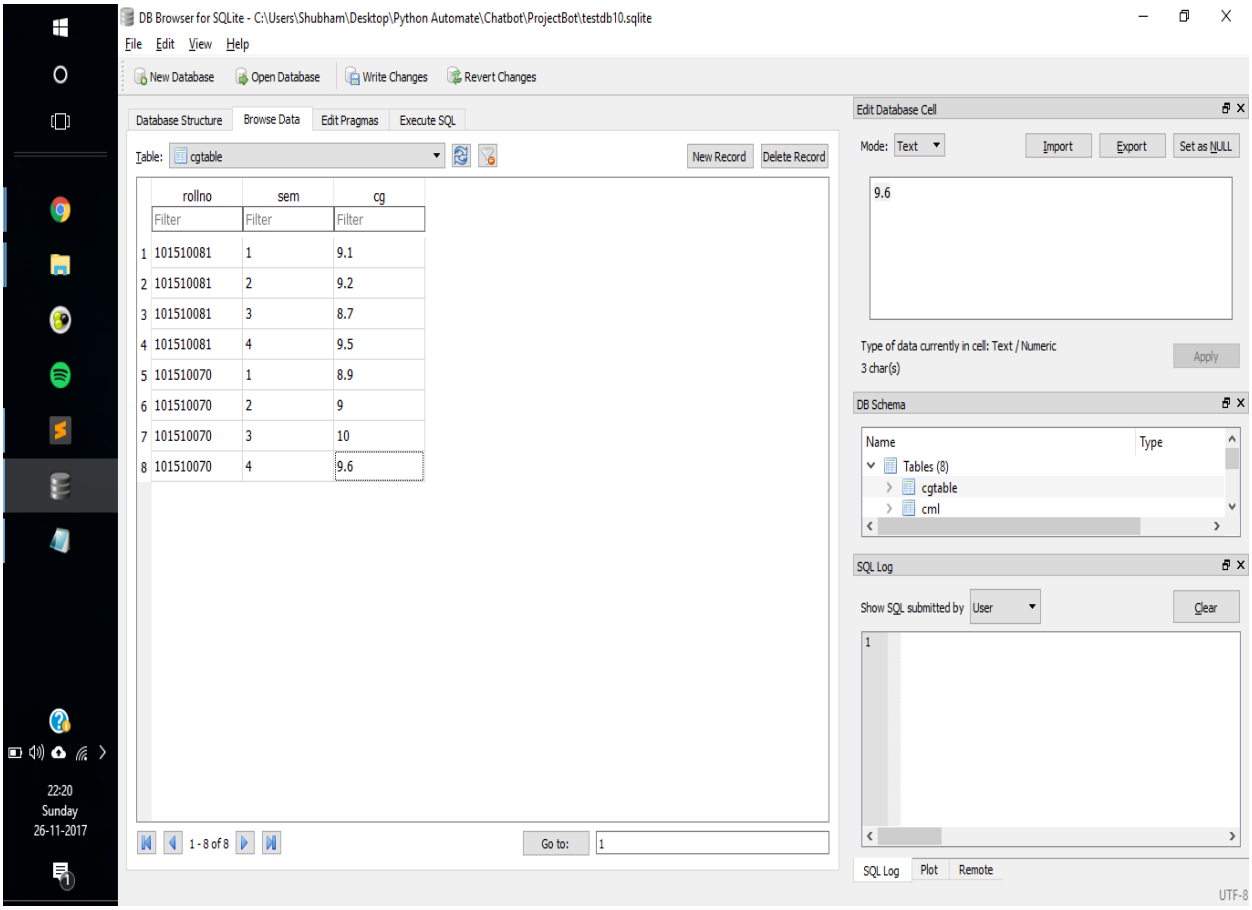


Figure 19

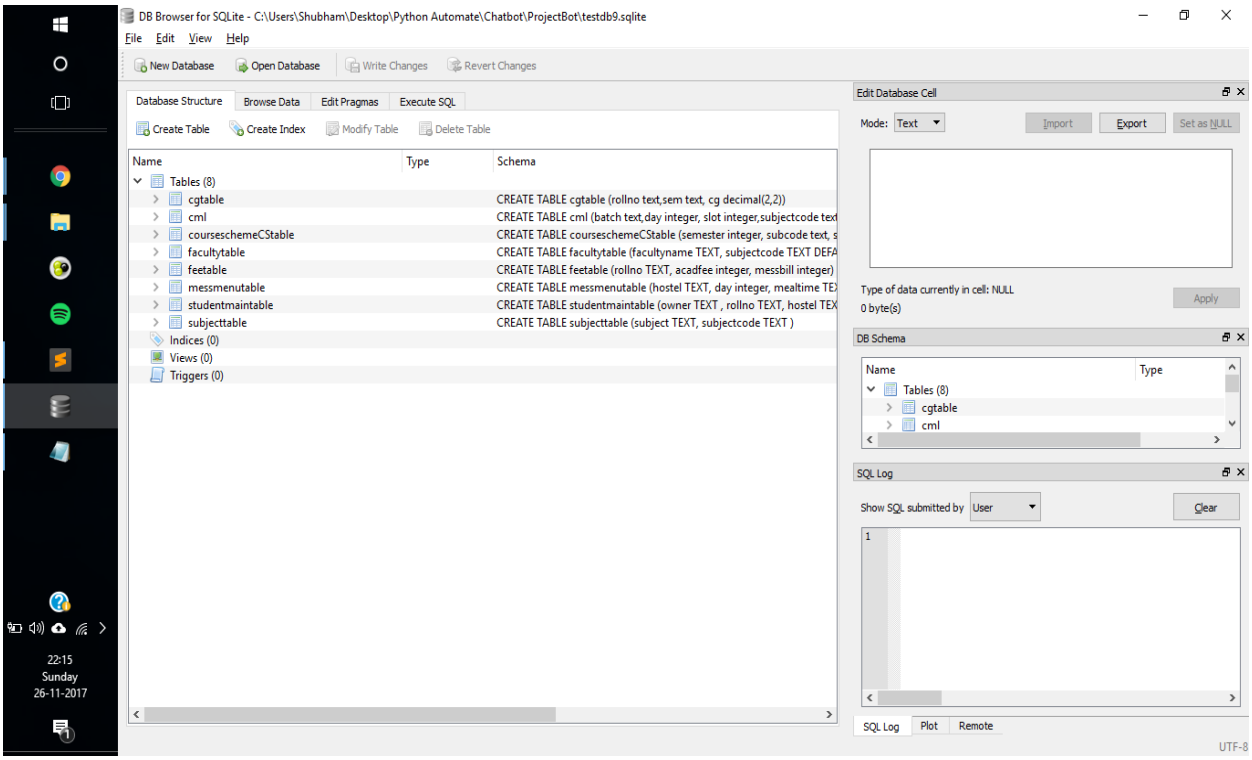


Figure 20

7. CONCLUSION

The proposed system works as expected. But its working relies heavily on telegram, which is expected but brings some amount of instability to the whole system. As in worst case, if telegram goes down then the system goes down. Though the proposed idea when discussed with the peers garnered positive response. Which further cemented the need to bring functionality of web kiosk to the instant messenger.

Functionality of certain features introduced in the project solely depends on regular update to the database. As attendance feature would only be fruitful to a person if it shows latest percentage. Same is the case with pending mess fee.

As with every new project some functionalities are buggy and certain functionalities need optimization. There is always room for improvements. Many could be fixed by continuous testing and debugging. Preferred deployment should be in phases, first making it available to 100 then adding 200 more students and so on.

In future machine learning could be implemented in the model which could predict the CGPA of the student. Also it can be expanded to encompass many colleges instead of one.

REFERENCES

[1] Girju, R. (2003, July). Automatic detection of causal relations for question answering. In Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering-Volume 12 (pp. 76-83). Association for Computational Linguistics

[2] Wang, J. H., Chung, E. S., & Jang, M. G. (2008). U.S. Patent No. 7,428,487. Washington, DC: U.S. Patent and Trademark Office.

[3] Soricut, R., & Brill, E. (2004). Automatic question answering: Beyond the factoid. In Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004.

[4] Soricut, R., & Brill, E. (2006). Automatic question answering using the web: Beyond the factoid. Information Retrieval, 9(2), 191-206.

[5] Green, C. C. (1969). The application of theorem proving to question-answering systems (No. CS-138). STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE.

[6] Ferrucci, D. A., & Zadrozny, W. W. (2012). U.S. Patent No. 8,280,838. Washington, DC: U.S. Patent and Trademark Office.

[7] Ravichandran, D., Ittycheriah, A., & Roukos, S. (2003). Automatic derivation of surface text patterns for a maximum entropy based question answering system. In Companion Volume of the Proceedings of HLT-NAACL 2003-Short Papers.

[8] Zhang, D., & Lee, W. S. (2003, July). Question classification using support vector machines. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval (pp. 26-32). ACM.

[9] Srihari, R., & Li, W. (2000). A question answering system supported by information extraction. In Sixth Applied Natural Language Processing Conference.

[10] Harabagiu, S., & Hickl, A. (2006, July). Methods for using textual entailment in open-domain question answering. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (pp. 905-912). Association for Computational Linguistics.

[11] Malinowski, M., & Fritz, M. (2014). A multi-world approach to question answering about real-world scenes based on uncertain input. In Advances in neural information processing systems (pp. 1682-1690).

[12] Fu, J., Xu, J., & Jia, K. (2009, January). Domain ontology based automatic question answering. In 2009 International Conference on Computer Engineering and Technology (Vol. 2, pp. 346-349). IEEE.

[13] Torres-Moreno, J. M., St-Onge, P. L., Gagnon, M., El-Beze, M., & Bellot, P. (2009). Automatic summarization system coupled with a question-answering system (qaas). arXiv preprint arXiv:0905.2990.

[14] Bian, J., Liu, Y., Agichtein, E., & Zha, H. (2008, April). Finding the right facts in the crowd: factoid question answering over social media. In Proceedings of the 17th international conference on World Wide Web (pp. 467-476). ACM.

[15] https://www.codementor.io/garethdwyer/building-a-telegram-bot-using-python-part-1-goi5fncay

[16] https://www.codementor.io/garethdwyer/building-a-chatbot-using-telegram-and-python-part-2-sqlite-databse-backend-m7o96jger

[17] https://www.codementor.io/officehours/5519386452/build-chatbot-python-deploy-vps?icn=post-m7o96jger&ici=OH130