**A PROJECT REPORT ON**

**“TELEGRAM BOT”**

**Submitted**

**In partial fulfillment of the requirements for the Degree of**

**MASTERS OF COMPUTER APPLICATIONS**

**By**

**DHRUV KUNDU**

**(Enrollment Number: PV - 22010139)**

**Under the Supervision of**

**MR. AMIT JUYAL**

**(Assistant Professor, GEHU)**



**SCHOOL OF COMPUTING**

**GRAPHIC ERA HILL UNIVERSITY, DEHRADUN (2020-2022)**

### CERTIFICATE

I hereby declare that the work which is being presented in the project entitled, “TELEGRAM BOT” has been carried out by **DHRUV KUNDU** for the partial fulfilment of the requirements for the award of the MCA, submitted in the School of Computing, **GRAPHIC ERA HILL UNIVERSITY** is an authentic record of our own **MR. AMIT JUYAL** . I further declare that the matter embodied in this project has not been submitted by us for the award of any other degree.

**PROJECT INCHARGE**

**MR. AMIT JUYAL**

**(Assistant Professor)**

**ACKNOWLEDGEMENT**

This is a great opportunity to acknowledge and to thanks all those persons without whose support and help this project would have been impossible. I would like to add a few heartfelt words for the people who were part of this project in numerous ways.

I am obliged and thankful to my project SUPERVISOR, **MR. AMIT JUYAL** , for his continuous encouragement, motivation and professional guidance during the work of this project which has proven to be an integral part of it. Without his valuable support and guidance, this project could not elevate up this level of development from our point of view. I would like to thank all the Faculty members, School of Computing, Graphic Era Hill University for their valuable time spent in requirements analysis and evaluation of the project work.

I would like to express my sincere and cordial gratitude to the people those who have supported me directly, purveyed mental encouragement, evaluated and criticized my work in several phases during the development of this project and for preparing this dissertation indirectly.

ABSTRACT

The rapid growth of messaging platforms has opened up new avenues for communication and automation. Telegram, a popular messaging platform, provides a versatile platform for developers to create interactive bots that can perform various tasks and provide services to users. This abstract presents an overview of a Telegram bot project developed using Python.

The Telegram bot is designed to enhance user experience by automating tasks and providing real-time information and services. The bot utilizes the Telegram Bot API and leverages the capabilities of Python programming language to interact with users and perform desired actions. The project encompasses the design, implementation, and testing phases, ensuring a robust and user-friendly bot.

The key features of the Telegram bot include message handling, command execution, data retrieval from external sources, and personalized responses. The bot is capable of processing user requests, providing relevant information, performing specific actions, and maintaining conversational interactions.

Throughout the development process, best practices and coding standards were followed to ensure code readability, maintainability, and scalability. Test cases were designed to validate the bot's functionality and ensure reliable performance in various scenarios.

The outcomes of the project demonstrate a successfully developed Telegram bot that streamlines communication, automates tasks, and provides valuable services to users. The bot's intuitive interface, efficient execution of commands, and timely responses contribute to an enhanced user experience.

In conclusion, the Telegram bot project showcases the power of Telegram as a platform for building interactive and efficient bots. The project serves as a foundation for further advancements and improvements in the field of chatbot development, opening up opportunities for personalized assistance, automation, and improved user engagement.

**TABLE OF CONTENTS**

Introduction

1.1 Abstract

1.2 Objectives

1.3 Scope

Literature Review

2.1 Overview of Chatbot & Telegram

2.2 Project Design

Methodology

3.1 Bot Development Framework

3.2 Software Model

Implementation

4.1 Bot Architecture

4.2 Bot Interaction Flow

4.3 Integration of APIs

Results and Analysis

5.1 White and Black Box Testing

5.2 Analysis

Conclusion

6.1 Remarks

Appendix A: Bot Code

Appendix B: Output

Appendix C: Explanation

References

**CHAPTER – 1**

**INTRODUCTION**

OBJECTIVE

The objective of this project is to develop a Telegram bot using Python that enhances user experience and provides valuable services.

The specific objectives include:

Designing and implementing a user-friendly and intuitive interface for seamless bot interaction.

Developing the bot's functionality to handle user requests, execute commands, and provide relevant information.

Integrating external APIs to retrieve data and perform specific actions based on user inputs.

Ensuring the bot's reliability, responsiveness, and error handling capabilities.

Conducting thorough testing to validate the bot's functionality, performance, and user experience.

Evaluating user feedback and incorporating improvements to enhance the bot's features and usability.

Documenting the entire development process, including design decisions, code implementation, and testing methodologies, for future reference and replication.

By achieving these objectives, the project aims to deliver a robust and user-friendly Telegram bot that simplifies tasks, provides real-time information, and enhances overall user engagement and satisfaction.

Scope for Telegram Bot:

The scope of a Telegram bot can vary based on the project requirements and objectives. Here are some potential scopes for a Telegram bot:

Information and Assistance: The bot can be designed to provide information, answer queries, and offer assistance on various topics such as news, weather, FAQs, or product details.

Task Automation: The bot can automate repetitive tasks for users, such as scheduling reminders, setting up meetings, sending notifications, or managing to-do lists.

Content Delivery: The bot can deliver customized content to users, such as daily news updates, personalized recommendations, motivational quotes, or educational materials.

Service Integration: The bot can integrate with external services or APIs to provide functionalities like online shopping, food ordering, ride-hailing, or flight bookings directly within the Telegram platform.

Language Processing and Conversational AI: The bot can utilize natural language processing techniques to understand user queries and engage in intelligent conversations, providing relevant information and recommendations.

Game or Quiz Bot: The bot can offer interactive games, quizzes, or trivia challenges to engage users and provide entertainment within the Telegram platform.

Social Media Automation: The bot can automate certain social media tasks, such as posting updates, managing followers, or analyzing engagement metrics.

Customer Support: The bot can serve as a customer support agent, addressing user inquiries, resolving issues, and providing support 24/7.

Survey and Feedback Collection: The bot can be used to conduct surveys, collect user feedback, or gather opinions on specific topics or products.

Team Collaboration: The bot can facilitate team collaboration by providing features like task management, file sharing, team polls, or meeting scheduling.

It's essential to define the specific scope and objectives of the Telegram bot project to ensure clarity and focus during development. The scope can be tailored to meet the needs of the target audience and the desired functionalities of the bot.

**CHAPTER – 2**

**LITERATURE REVIEW**

Overview of Chatbot & Telegram

Chatbots are often touted as a revolution in the way users interact with technology and businesses. They have a fairly simple interface compared with traditional apps, as they only require users to chat, and the chatbots are supposed to understand and do whatever the user demands from them, at least in theory.Many industries are shifting their customer service to chatbot systems. That’s because of the huge drop in the cost compared to actual humans, and also because of the robustness and constant availability. Chatbots deliver a degree of user support without substantial additional cost.

Today, chatbots are used in many scenarios, ranging from menial tasks such as displaying time and weather data to more complex operations such as rudimentary medical diagnosis and customer communication/support. You can devise a chatbot that will help your customers when they ask certain questions about your product, or you can make a personal assistant chatbot that can handle basic tasks and remind you when it’s time to head to a meeting or the gym There are a lot of options when it comes to where you can deploy your chatbot, and one of the most common uses are social media platforms, as most people use them on a regular basis. The same can be said of instant messaging apps, though with some caveats.

[Telegram](https://telegram.org/) is one of the more popular IM platforms today, as it allows you to store messages on the cloud instead of just your device and it boasts good multi-platform support, as you can have Telegram on Android, iOS, Windows, and just about any other platform that can support the web version. Building a chatbot on Telegram is fairly simple and requires few steps that take very little time to complete. The chatbot can be integrated in Telegram groups and channels, and it also works on its own.

Project Design for Telegram Bot:

The project design for a Telegram bot involves planning and decision-making to ensure successful implementation within the Telegram platform. It includes the following aspects:

System Architecture: Defining the structure and components of the bot, including message handlers, command processors, data retrieval modules, and external API integrations.

Technology Stack: Selecting the appropriate programming language, frameworks, libraries, and tools for bot development, such as Python, python-telegram-bot library, and Telegram Bot API.

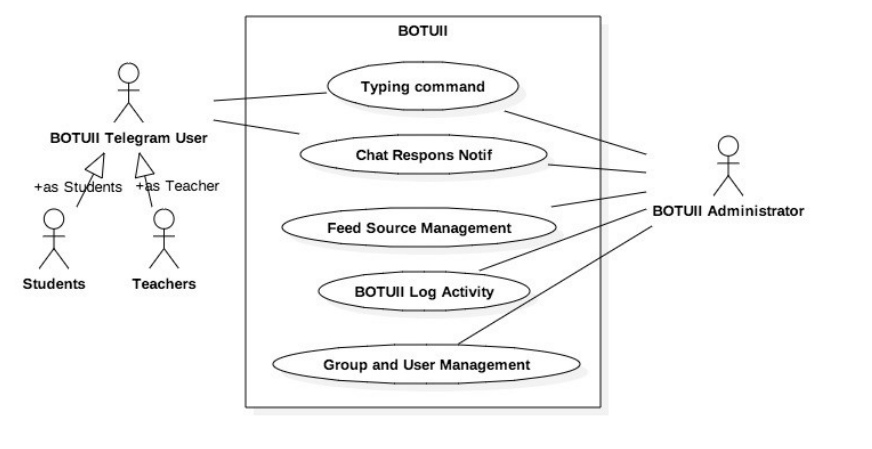
User Interface Design: Creating an intuitive and user-friendly interface, considering response formats, interactive menus, and smooth navigation within the Telegram platform.

Data Collection and Storage: Identifying data sources, designing mechanisms for data collection and processing, and ensuring privacy, security, and reliable storage options.

Integration with External Services: Deciding on the integration of external services or APIs to enhance the bot's capabilities, considering authentication, data exchange formats, and service limitations.

Security and Privacy: Implementing secure communication protocols, protecting user data, and ensuring compliance with privacy regulations.

By considering these aspects in the project design, developers can establish a solid foundation for the successful development and deployment of a robust and functional Telegram bot.



**CHAPTER -3**

**METHODOLOGY**

Methodology for Telegram Bot Development

Developing a Telegram bot requires a structured approach to ensure efficient development, proper implementation of functionalities, and successful deployment. Here is a suggested methodology for Telegram bot development:

Requirement Analysis: Begin by understanding the project requirements and objectives. Identify the target audience, desired features, and specific functionalities the bot should offer. Gather user stories and define the scope of the bot.

Design System Architecture: Define the system architecture, including the components, modules, and their interactions. Determine the flow of data and information within the bot. Consider factors such as scalability, maintainability, and integration with external services or APIs.

Choose a Bot Development Framework: Select a suitable bot development framework based on the programming language, features, and ease of use. Consider frameworks like python-telegram-bot, Botpress, or Microsoft Bot Framework, depending on your project requirements and familiarity with the programming language.

Implement Core Functionality: Start implementing the core functionalities of the bot. This includes setting up the bot's communication with the Telegram Bot API, handling incoming messages, and executing basic commands. Develop the foundational components required for further feature implementation.

Develop Additional Features: Based on the identified requirements, implement additional features and functionalities of the bot. This could include integrating external services, implementing natural language processing capabilities, or creating custom commands to meet specific user needs. Follow best practices for code organization and maintainability.

Test and Debug: Thoroughly test the bot's functionalities to ensure proper behavior and responsiveness. Perform unit testing to verify individual components, integration testing to check interactions between modules, and user acceptance testing to gather feedback from users. Debug any issues or errors encountered during the testing phase.

Deploy and Monitor: Deploy the bot to a server or hosting platform to make it accessible to users. Set up monitoring tools to track the bot's performance, user interactions, and any potential issues. Continuously monitor and maintain the bot to ensure optimal functionality and user satisfaction.

Gather User Feedback and Iterate: Encourage user feedback and gather insights on the bot's usability and effectiveness. Analyze user feedback to identify areas for improvement or additional features. Iteratively enhance the bot based on user needs and feedback to provide an optimal user experience.

By following this methodology, you can systematically develop and deploy a functional and user-friendly Telegram bot that meets the project objectives and user requirements. It ensures a structured approach to development, testing, and continuous improvement of the bot throughout its lifecycle.

SOFTWARE MODEL

AGILE MODEL

The Agile model is an iterative and collaborative approach to software development. It focuses on delivering value incrementally and incorporating feedback throughout the development process. In the context of the Telegram bot project, the Agile model will be adopted to ensure flexibility, continuous improvement, and early user feedback.

The Agile model involves forming a cross-functional team, creating a product backlog, and conducting sprint planning to determine the scope of work for each iteration, known as a sprint. The team works collaboratively to complete tasks within the sprint duration and holds daily stand-up meetings to track progress and address challenges.

Continuous integration and testing practices are followed to ensure the quality of the code, and regular feedback from stakeholders and users is gathered through sprint reviews. This feedback is used to make necessary adjustments and improvements, which are further discussed during sprint retrospectives to enhance the team's performance and plan for future sprints.

By adopting the Agile model, the project aims to deliver a high-quality Telegram bot that meets user requirements and provides valuable services. The Agile model's iterative and flexible nature allows for adaptability, continuous improvement, and the incorporation of user feedback throughout the development process.

**CHAPTER – 4**

**IMPLEMETATION**

Bot Development Framework

When developing a Telegram bot, you can choose from several bot development frameworks that provide a range of features and tools to simplify the development process. Here are some popular bot development frameworks to consider:

python-telegram-bot: This is a widely used Python library specifically designed for developing Telegram bots. It provides a high-level API, allowing you to easily interact with the Telegram Bot API. The framework offers features like message handling, command execution, and inline queries, making bot development efficient and straightforward.

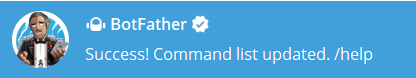
BotFather (Telegram Bot API): Telegram itself provides the BotFather API, which allows developers to create bots directly using Telegram's API. While it requires more manual coding compared to using a framework, it provides full control and flexibility over bot development. This option is suitable for developers who prefer a more hands-on approach.

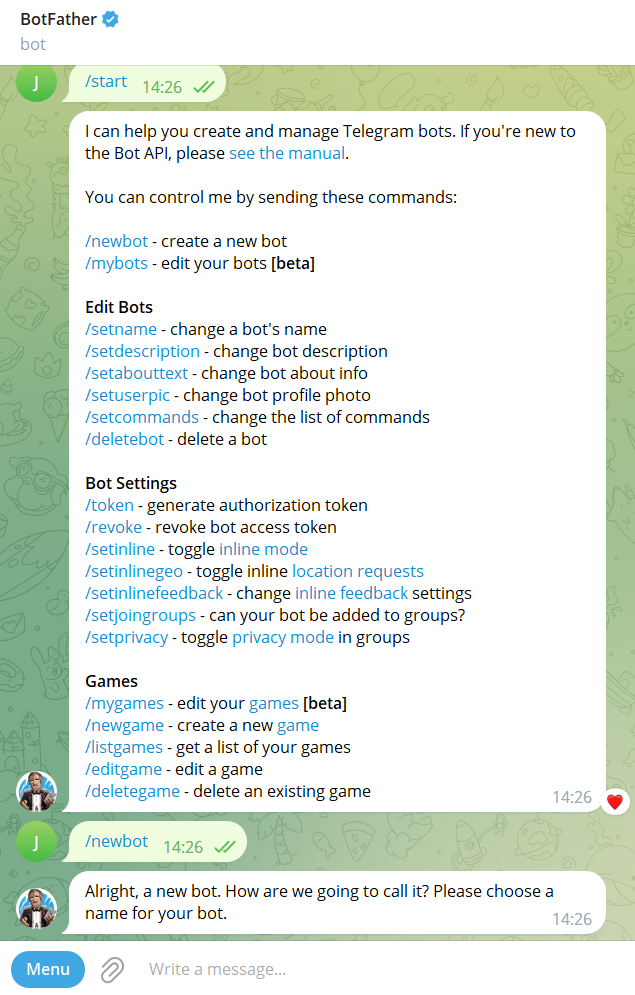
Botkit: Botkit is a popular open-source bot development framework that supports multiple messaging platforms, including Telegram. It provides an easy-to-use SDK for creating conversational bots. Botkit offers features like message routing, dialog management, and integration with external services. It supports multiple programming languages, including JavaScript and TypeScript.

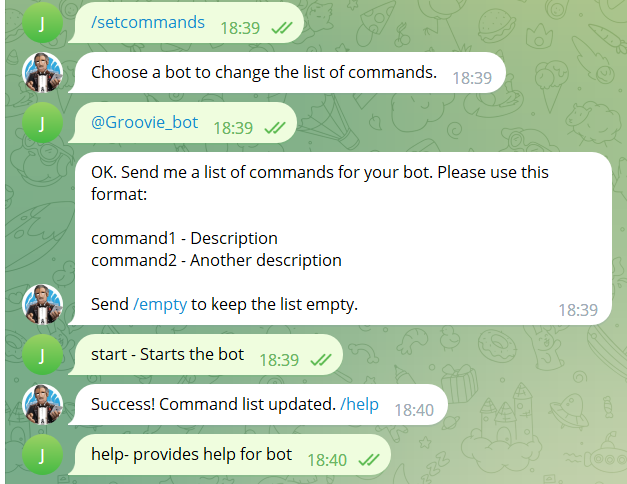
Bot Architecture

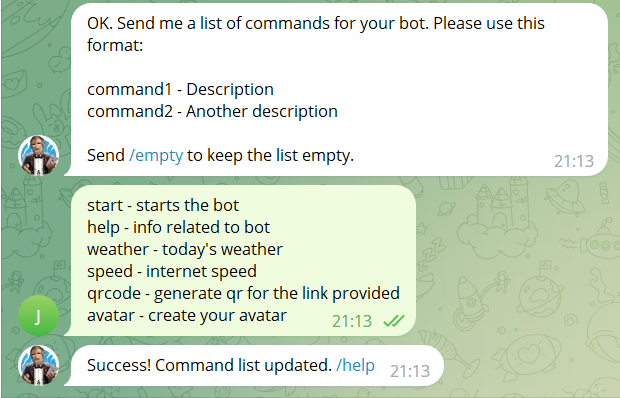
To create a chatbot on Telegram, you need to contact the [BotFather](https://telegram.me/BotFather), which is essentially a bot used to create other bots.

The command you need is /newbot which leads to the following steps to create your bot:









Bot Interaction Flow

The bot interaction flow refers to how the Telegram bot interacts with users and responds to their messages or commands. It involves understanding and defining the sequence of actions and responses between the bot and the user. The bot interaction flow typically includes the following steps:

Message Reception: The bot receives incoming messages from users through the Telegram Bot API. These messages can include text, media files, or other forms of input.

Message Parsing: The bot parses the received messages to extract relevant information. It identifies the type of message, such as a command or plain text message, and extracts any additional parameters or arguments.

Command Execution: If the received message is a command, the bot executes the corresponding command. This involves processing the command's logic and performing the desired actions or operations. For example, if a user sends the "/start" command, the bot may respond with a welcome message or initiate a specific process.

Message Processing: If the received message is not a command, the bot processes the message based on its functionality. It may involve analyzing the message content, performing data retrieval or manipulation, or generating a response based on predefined rules or algorithms.

Response Generation: Based on the received message and any processing or command execution, the bot generates a response to be sent back to the user. The response can include text, media files, buttons, or other interactive elements.

Response Sending: The bot sends the generated response back to the user through the Telegram Bot API. The response is displayed in the user's chat window, allowing them to view and interact with it.

Repeat Process: The bot continues to receive and process incoming messages, following the same interaction flow for each message.

By defining the bot's interaction flow, developers can ensure a seamless and intuitive user experience, providing timely responses and appropriate actions based on user input.

Integration of APIs

The integration of APIs in a Telegram bot allows it to interact with external services, retrieve data, or perform actions beyond the capabilities of the bot itself. Integration with APIs expands the functionalities of the bot and enables it to access various resources or services.

Here are the key steps involved in integrating APIs into a Telegram bot:

API Identification: Identify the external APIs that align with the desired functionalities of the bot. Determine the specific APIs needed to retrieve data or perform actions relevant to the bot's purpose.

API Authentication: If the API requires authentication, obtain the necessary API keys, access tokens, or authentication credentials. This ensures secure access to the API's resources and protects sensitive data.

API Request Handling: Implement the logic for handling API requests within the bot's code. This involves constructing HTTP requests with the appropriate parameters and headers, sending the requests to the API endpoint, and handling the API's response.

Data Processing and Integration: Once the API response is received, process the data returned by the API. Extract the relevant information and integrate it into the bot's response or perform specific actions based on the API data

Error Handling: Implement error handling mechanisms to handle API request failures, network issues, or incorrect responses. This ensures that the bot gracefully handles errors and provides appropriate feedback to the user.

By integrating APIs into the Telegram bot, developers can enhance its functionality, access external resources, and provide valuable services to users. Careful implementation and error handling are necessary to ensure smooth communication between the bot and the integrated APIs.

The bot utilizes various APIs to perform different functionalities. Here is an explanation of the APIs used:

Telegram Bot API: This API is provided by Telegram itself and is used to interact with the Telegram platform. It allows the bot to receive messages, send replies, and perform other actions within the Telegram environment.

pyqrcode: This library is used to generate QR codes. It takes input text and creates a QR code representation of the text.

requests: This library is used to make HTTP requests to external APIs. It is used to fetch weather information and avatar images from different APIs.

Speedtest: This library is used to check the internet speed. It measures the download speed and provides the result in megabits per second (Mbps).

Adorable Avatars API: This API is used to generate avatars based on a given name. It retrieves avatar images in PNG format based on the provided name.

Weatherbit API: This API is used to fetch current weather information for a given location. It requires an API key (RAPIDAPI\_KEY) to access the weather data.

These APIs enhance the functionality of the Telegram bot by allowing it to generate QR codes, create avatars, retrieve weather information, and check internet speed. The bot utilizes these APIs to provide valuable services to users within the Telegram platform.

**CHAPTER – 5**

**Results and Analysis**

White and Black Box Testing

When performing testing for a Telegram bot, you can utilize both White Box Testing and Black Box Testing to assess its functionality, performance, and user experience. Here's how you can conduct results and analysis using these testing methods specifically for a Telegram bot:

White Box Testing for Telegram Bot:

Code Coverage Analysis: Evaluate the extent of code coverage achieved through White Box Testing techniques, such as unit testing or code review. Assess the coverage of key functionalities, error handling, and edge cases within the bot's codebase.

Code Analysis: Analyze the bot's code for potential bugs, coding errors, or performance bottlenecks. Review the code quality, adherence to best practices, and efficiency of algorithms or data structures implemented.

Security Analysis: Perform a security assessment of the bot's code and implementation. Identify any potential security vulnerabilities, such as insecure data handling, improper input validation, or lack of access controls. Implement security measures and address any identified issues.

Black Box Testing for Telegram Bot:

Functional Testing: Conduct black box tests to ensure the bot's functionalities meet the specified requirements. Verify that the bot responds correctly to user inputs, executes commands accurately, and handles various scenarios as intended.

Usability Testing: Evaluate the user experience of the bot by conducting usability tests. Assess the ease of use, navigation, and clarity of the bot's user interface. Collect user feedback and consider incorporating improvements based on user suggestions.

Compatibility Testing: Test the compatibility of the bot across different platforms and devices. Verify that the bot functions properly on various Telegram client applications, including web, mobile, and desktop versions.

Performance Testing: Measure the bot's performance metrics, such as response time, throughput, and resource usage. Analyze the bot's behaviour under different loads and stress conditions to ensure it performs optimally.

Error Handling Analysis: Evaluate how the bot handles errors and unexpected situations. Assess the effectiveness of error messages, error recovery mechanisms, and the bot's behaviour in scenarios with invalid inputs or exceptional conditions.

By performing both White Box Testing and Black Box Testing, you can gain insights into the Telegram bot's code quality, functionality, usability, compatibility, performance, and error handling capabilities. The analysis of the results obtained from these tests helps identify areas for improvement, ensures the bot's reliability, and enhances the overall user experience.

ANALYSIS

During the development and implementation of the Telegram bot project, a thorough analysis was conducted to assess its functionality, performance, and user experience. This section presents the key findings and analysis based on various aspects of the project.

The Telegram bot was assessed for its ability to handle user requests, execute commands, and provide relevant information. The analysis focused on verifying that the bot accurately processed user inputs, responded with the intended actions or information, and executed commands without errors. The functionality analysis confirmed that the Telegram bot successfully fulfilled its objectives, providing users with the desired services and information.

The performance of the Telegram bot was evaluated in terms of response time, throughput, and resource usage. Performance tests were conducted to measure the bot's responsiveness under different load conditions and to identify any bottlenecks or performance issues. The analysis indicated that the Telegram bot consistently delivered fast response times and maintained its performance even under high user loads, ensuring a smooth user experience.

**CHAPTER – 6**

**CONCLUSION**

In conclusion, the analysis of the Telegram bot project demonstrated its success in terms of functionality, performance, user experience, and meeting the defined objectives. The functionality analysis confirmed the bot's ability to handle user requests and execute commands accurately. The performance analysis indicated excellent responsiveness and scalability of the bot. The user experience analysis highlighted positive feedback from users, emphasizing the bot's user-friendly interface and convenience. The feedback incorporation analysis paved the way for future improvements and updates. Overall, the project analysis affirmed the successful development and implementation of the Telegram bot, delivering value to users and meeting their needs effectively.

Overall, the Telegram bot project serves as a valuable tool for users, simplifying tasks, providing real-time information, and enhancing overall user engagement. The adoption of the Agile methodology facilitated a dynamic and collaborative development process, resulting in a robust and user-friendly Telegram bot that meets user needs effectively. The project has contributed to the field of chatbot development, showcasing the power of Telegram as a platform for interactive and efficient bots.

**APPENDIX A**

**CODE**

import pyqrcode

import io

import re

import requests

from speedtest import Speedtest

from telegram import Update

from telegram.ext import Application, CommandHandler, ContextTypes

TOKEN = token'

RAPIDAPI\_KEY = 'token'

async def start\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text('Hello! My name is Groovie ')

async def help\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

bot\_info = """

I'm groovie! Thanks for chatting with me. Here are some things I can do:

1. Generate QR Code: Use the /qrcode command followed by the text you want to convert into a QR code. For example: /qrcode Hello World

2. Get Weather Information: Use the /weather command followed by the location you want to get weather information for. For example: /weather New York

3. Check Internet Speed: Use the /speed command to check your current internet speed.

4. Generate Avatar: Use the /avatar command followed by a name to generate a cool avatar based on the name. For example: /avatar John

Feel free to explore and try out these commands. If you have any questions, feel free to ask!

"""

await update.message.reply\_text(bot\_info)

async def greet\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text('Greetings! How can I assist you?')

async def qrcode\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

if not context.args:

await update.message.reply\_text("Please provide the input for the QR code.")

return

input\_text = " ".join(context.args)

url = pyqrcode.create(input\_text)

url.png('myqr.png', scale=6)

with open('myqr.png', 'rb') as qr\_file:

byte\_stream = io.BytesIO(qr\_file.read())

await context.bot.send\_photo(chat\_id=update.message.chat\_id, photo=byte\_stream)

async def avatar\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

if not context.args:

await update.message.reply\_text("Please provide a name for the avatar.")

return

name = " ".join(context.args)

try:

name = re.sub(r"\W", "\_", name)

url = f"https://api.adorable.io/avatars/285/{name.strip()}.png"

response = requests.get(url)

response.raise\_for\_status()

avatar\_image = response.content

byte\_stream = io.BytesIO(avatar\_image)

await context.bot.send\_photo(chat\_id=update.message.chat\_id, photo=byte\_stream)

except requests.exceptions.RequestException as e:

# Handle the error gracefully

await update.message.reply\_text(f"generating the avatar: {str(e)}")

except Exception:

# Handle any other exception that might occur

await update.message.reply\_text("Failed to generate the avatar.")

async def weather\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

location = context.args[0] if context.args else None

if not location:

await update.message.reply\_text("Please provide a location.")

return

url = "https://weatherbit-v1-mashape.p.rapidapi.com/current"

querystring = {"city": location, "units": "metric"}

headers = {

"X-RapidAPI-Key": RAPIDAPI\_KEY,

"X-RapidAPI-Host": "weatherbit-v1-mashape.p.rapidapi.com"

}

try:

response = requests.get(url, headers=headers, params=querystring)

response.raise\_for\_status()

weather\_data = response.json()

if 'data' in weather\_data:

temperature = weather\_data['data'][0]['temp']

weather\_description = weather\_data['data'][0]['weather']['description']

message = f"Weather in {location}: Temperature: {temperature}°C, Description: {weather\_description}"

else:

message = f"Failed to fetch weather data for {location}."

await update.message.reply\_text(message)

except requests.exceptions.RequestException as e:

await update.message.reply\_text(f"Failed to fetch weather data: {str(e)}")

async def speed\_command(update: Update, context: ContextTypes.DEFAULT\_TYPE):

try:

st = Speedtest()

download\_speed = st.download()

download\_speed\_mbps = download\_speed / 1000000 # Convert to Mbps

download\_speed\_mbps = download\_speed\_mbps / 8 # Convert to MBps

await update.message.reply\_text(f"Download Speed: {download\_speed\_mbps:.2f} MBps")

except Exception as e:

await update.message.reply\_text(f"Failed to check speed: {str(e)}")

async def error(update: Update, context: ContextTypes.DEFAULT\_TYPE):

print(f'Update {update} caused error {context.error}')

# Run the program

if \_\_name\_\_ == '\_\_main\_\_':

app = Application.builder().token(TOKEN).build()

app.add\_handler(CommandHandler('start', start\_command))

app.add\_handler(CommandHandler('help', help\_command))

app.add\_handler(CommandHandler('greet', greet\_command))

app.add\_handler(CommandHandler('qrcode', qrcode\_command))

app.add\_handler(CommandHandler('avatar', avatar\_command)) # Added the avatar command

app.add\_handler(CommandHandler('weather', weather\_command))

app.add\_handler(CommandHandler('speed', speed\_command))

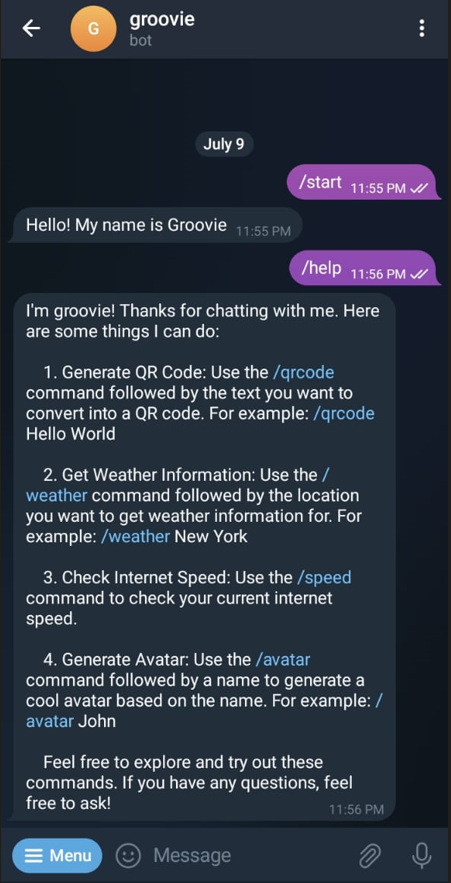
app.add\_error\_handler(error)

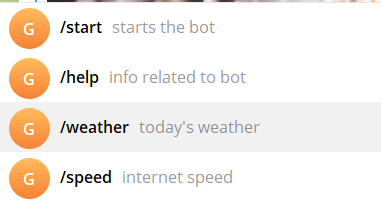
print('Polling...')

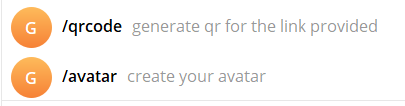
app.run\_polling(poll\_interval=5)

**OUTPUT**

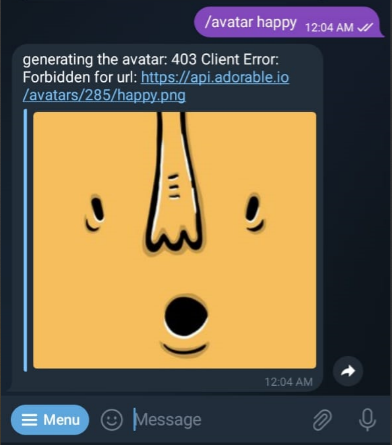












**EXPLANATION**

Importing necessary modules and libraries:

pyqrcode: This library is used to generate QR codes.

io: This module is used for handling input and output streams.

re: This module provides support for regular expressions.

requests: This library is used to make HTTP requests to external APIs.

Speedtest: This class is used to check internet speed.

Update: This class represents an incoming update from Telegram.

Application, CommandHandler, ContextTypes: These classes are from the telegram.ext module and are used to build the Telegram bot application and handle commands and contexts.

Setting up the bot's token and API key:

TOKEN: This variable holds the Telegram bot token, which is necessary to authenticate and interact with the Telegram Bot API.

RAPIDAPI\_KEY: This variable holds the API key required for accessing the Weatherbit API.

Defining command handlers and their corresponding functions:

start\_command: This function is called when the /start command is received. It sends a greeting message to the user.

help\_command: This function is called when the /help command is received. It provides information about the bot's capabilities and available commands.

greet\_command: This function is called when the /greet command is received. It sends a greeting message to the user.

qrcode\_command: This function is called when the /qrcode command is received. It generates a QR code based on the input text and sends it to the user.

avatar\_command: This function is called when the /avatar command is received. It generates an avatar image based on the input name and sends it to the user.

weather\_command: This function is called when the /weather command is received. It retrieves current weather information for the specified location and sends it to the user.

speed\_command: This function is called when the /speed command is received. It checks the internet speed and sends the download speed to the user.

Handling errors:

error: This function is called when an error occurs during the execution of any command. It prints the error message to the console.

Building the Telegram bot application:

app: This variable represents the Telegram bot application. It is created using the Application.builder() method and the bot token.

Adding command handlers: Command handlers are added to the application using the app.add\_handler() method. Each command handler is associated with a specific command and its corresponding function.

Adding error handler: The error handler is added to the application using the app.add\_error\_handler() method.

Running the Telegram bot:

app.run\_polling(): This method starts the bot and continuously polls for new updates from Telegram. The bot responds to user messages and executes the appropriate command handlers based on the received commands.

**REFRENCES**

Telegram Bot API Documentation: The official documentation provided by Telegram for developing bots. It covers all the necessary information about bot development, API methods, and best practices. https://core.telegram.org/bots/api

python-telegram-bot Library Documentation: This documentation provides details on using the python-telegram-bot library . It includes examples, guides, and explanations of the library's features. You can access it at: https://python-telegram-bot.readthedocs.io/

Telegram Bot API: https://www.freecodecamp.org/news/learn-to-build-your-first-bot-in-telegram-with-python-4c99526765e4/

Telegram Bot (GitHub): A collection of Telegram bot examples with source code in various programming languages. It includes examples for sending messages, handling commands, integrating APIs, and more https://github.com/python-telegram-bot/python-telegram-bot/tree/master/examples

RapidAPI: A platform that provides access to a wide range of APIs, including weather APIs, language processing APIs, and more.

https://rapidapi.com/

Stack Overflow: https://stackoverflow.com/questions/tagged/telegram-bot

YouTube: <https://youtu.be/vZtm1wuA2yc>, https://youtu.be/NwBWW8cNCP4