VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi – 590 018, Karnataka

MINIPROJECT REPORT

ON

Music Bot

Submitted in partial fulfillment of the requirements for the course 21CSMP67 – Mini Project, for awarding

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

1JT21CS001	ABHINAV K R
1JT21CS039	DEEKSHA S
1JT21CS053	KRUTHIKA C S
1JT21CS054	KUSHAL S GOWDA

Guide: NAGARAJ. A
Associate Professor
Department of CSE

Department of Computer Science and Engineering

Jyothy Institute of Technology

Tataguni, Off. Kanakapura Road, Bengaluru – 560 082

2023–2024

JYOTHY INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Accredited by NBA, New Delhi

Tataguni, Off. Kanakapura Road,

Bengaluru - 560 082

C E R T I F I C A T E

This is to certify that the Mini Project, titled *Music Bot*, carried out by

1JT21CS001	ABHINAV K R
1JT21CS039	DEEKSHA S
1JT21CS053	KRUTHIKA C S
1JT21CS054	KUSHAL S GOWDA

students of Jyothy Institute of Technology, in partial fulfilment, for the award of **Bachelor of Engineering**, in **Computer Science and Engineering** under Visvesvaraya Technological University, Belagavi, during the year 2023-2024.

The report satisfies the academic requirements in respect of 21CSMP67 – Mini Project.

Signature of Guide	Signature of HOD
....	....
Nagaraj. A	Dr. Prabhanjan. S.
Associate Professor	Prof & Head of Dept - CSE
Dept of CSE	JIT, Bengaluru
JIT, Bengaluru	

External Viva

Name of the Examiners	Signature with Date
1)	
2)	

D E C L A R A T I O N

We, the students of 6th Semester, Computer Science and Engineering, Jyothy Institute of Technology, Bangalore - 560 082, declare that the Mini Project – 21CSMP67, is successfully completed.

This report is submitted in partial fulfillment of the requirements for award of Bachelor Degree in Computer Science and Engineering, during the academic year 2023-2024.

Date : 24/07/2024 Place : Bangalore

1JT21CS001	ABHINAV K R
1JT21CS039	DEEKSHA S
1JT21CS053	KRUTHIKA C S
1JT21CS054	KUSHAL S GOWDA

ACKNOWLEDGEMENT

This Mini Project, is a result of accumulated guidance, direction and support of several important persons. I take this opportunity to express our gratitude to all, who have helped us to complete the Mini Project.

I express my sincere thanks to our Principal Dr K. Gopalakrishna, for providing us adequate facilities to undertake this Mini Project.

I would like to thank Dr. Prabhanjan. S, Head of Dept - CSE, for providing us an opportunity to carry out Mini Project and for his valuable guidance and support.

I would like to thank, Nagaraj. A, Associate Professor, Dept of CSE, for guiding us during the course of Mini Project.

I would like to thank all the faculty members of CSE department for the support extended during the Mini Project.

I would like to thank the non-teaching members of the Dept of CSE, for helping us during the Mini Project.

Last but not the least, I would like to thank my parents and friends without whose constant help, the completion of Mini Project would have not been possible.

1JT21CS001	**ABHINAV K R**
1JT21CS039	**DEEKSHA S**
1JT21CS053	**KRUTHIKA C S**
1JT21CS054	**KUSHAL S GOWDA**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

INTRODUCTION

In today's digital age, music bots have become an integral part of the online social experience, especially in platforms like Discord, where communities gather to chat, play games, and share media. The development of a robust and feature-rich music bot for Discord presents a multifaceted challenge that encompasses various technical and user-experience considerations. This project aims to create a music bot capable of playing, pausing, stopping, skipping, and queuing songs in a Discord voice channel, while ensuring high performance, reliability, and ease of use.

The primary objective of the music bot is to provide users with seamless control over their audio experience within Discord voice channels. This includes allowing users to search for and play songs from various sources such as YouTube and Spotify. The bot must ensure minimal latency to deliver a smooth listening experience, maintain high audio quality, and operate without crashes or interruptions. Key functionalities include:

1. **Playback Control**: The bot should allow users to easily play, pause, stop, skip, and queue songs using simple commands. This control is vital for managing the audio environment in a dynamic voice channel.
2. **Song Search and Integration**: Users should be able to search for songs from multiple popular sources like YouTube and Spotify. This integration ensures a diverse music library, catering to varied tastes and preferences.
3. **Performance and Stability**: Ensuring that the bot runs smoothly is critical. The bot should be resilient to crashes and interruptions, providing a reliable service to users. This requires efficient handling of voice channel operations and user commands through the Discord API.
4. **Cross-Platform Deployment**: The bot must be versatile enough to be deployed and operated on various platforms, including Windows and Linux. This flexibility is essential for reaching a broad user base with different server environments.
5. **Audio Quality and Bandwidth Management**: Maintaining high audio quality while minimizing bandwidth usage is crucial. The bot should handle potential network issues gracefully, ensuring that the listening experience remains uninterrupted and of high quality.
6. **Scalability**: The bot should be designed to handle user requests efficiently, especially in large servers with many concurrent users. This involves optimizing the bot's architecture to support scalability without compromising performance.

OBJECTIVES AND GOALS

The ultimate goal of this project is to deliver a music bot that not only meets the functional requirements but also exceeds user expectations in terms of usability and reliability. This involves:

- **Utilizing the Discord API**: Efficient use of the Discord API is essential for managing voice channel operations and user commands seamlessly.
- **Integration with Music Sources**: Ensuring smooth integration with popular music sources like YouTube and Spotify to provide a wide range of music options.
- **Ensuring Smooth Operation**: The bot should operate without crashes or interruptions, providing a consistent and enjoyable user experience.
- **Platform Flexibility**: The bot should be easily deployable on various operating systems, ensuring broad accessibility.
- **High Audio Quality**: Maintaining high audio quality while minimizing bandwidth usage is essential for user satisfaction.
- **Handling Network Issues**: Designing the bot to handle potential network issues gracefully, ensuring uninterrupted service.
- **Scalability**: Building the bot to scale efficiently, handling high loads and numerous concurrent users without performance degradation.

By addressing these objectives, the project aims to develop a music bot that enhances the Discord experience, providing users with reliable, high-quality, and versatile audio control in their voice channels.

CHAPTER 2

PROBLEM STATEMENT

PROBLEM STATEMENT:

The goal of this project is to develop a Discord music bot with the following capabilities:

- Allow users to play, pause, stop, skip, and queue songs in a Discord voice channel.
- Enable users to search for songs from various sources (e.g., YouTube, Spotify) and play them directly.
- Ensure minimal latency for a smooth listening experience.
- Ensure the bot runs smoothly without crashes or interruptions.
- Utilize the Discord API efficiently for voice channel operations and user commands.
- Integrate with popular music sources like YouTube for a diverse music library.
- Ensure the bot can be deployed and operated on various platforms (Windows, Linux, etc.).
- Maintain high audio quality while minimizing bandwidth usage and handling potential network issues.
- Design a system to handle user requests efficiently, especially in large servers with many concurrent users.

This project aims to develop a feature-rich Discord music bot capable of seamlessly managing audio playback within voice channels. Users can control playback (play, pause, stop, skip, queue) and search for songs from sources like YouTube and Spotify. Ensuring minimal latency, reliability, and smooth operation without crashes, the bot utilizes the Discord API efficiently. It integrates with popular music platforms, maintains high audio quality while optimizing bandwidth, and supports deployment across multiple platforms. Designed to handle user requests effectively, even in large servers, this bot enhances the Discord experience by offering robust music streaming capabilities.

CHAPTER 3

WORK BREAKDOWN STRUCTURE

21CSMP67 - Mini Project
Work breakdown structure

Week No	Date	Details of work done
Week 1	01-05-2024 to 07-05-2024	**Setting Up the Python Environment for Music Bot Development**
		• Defined the scope and main features of Music bot.
		• Set up the environment using Python.
Week 2	08-05-2024 to 15-05-2024	**Discord Bot Setup and Server Configuration**
		• Created Bot in Discord Developer Portal
		• Configured Discord Server in Application
		• Added Bot to the Server
Week 3	16-05-2024 to 23-05-2024	**Setup and Extensions Installed in VS Code**
		• Installed Extensions: ➢ Discord.py[voice] ➢ youtube-dl ➢ discord.py ➢ pynacl ➢ Urllib3
		• Additional Software Installed: • FFmpeg
		FFmpeg Description: FFmpeg is an open-source software project that offers tools for processing audio and video, essential for the functionality of the Discord music bot.

Week 4	24-05-2024 to 31-05-2024	**Initiating Music Bot Development**
		• Began Coding for Music Bot
		• Working on Bot Activation in Discord
		• Encountered Error: 'No module named 'discord components'
		• Commenced Research on Error Resolution
Week 5	01-06-2024 to 07-06-2024	**Resolution of Error in Music Bot Development**
		• Resolved Error: 'No module named 'discord components'
		• Discovered Discord Application Update on May 29th, 2024
		• Note: 'discord-components' Module Removed by Discord Developers
		• Exploring New Libraries and Commands for Discord
Week 6	08-06-2024 to 15-06-2024	**Resuming Music Bot Development**
		• Continued Coding
		• Successfully Deployed Bot Online
Week 7	16-06-2024 to 23-06-2024	**Achievements in Music Bot Development**
		• Achieved: Bot successfully joins Discord voice channels

		• Implemented command prefix '.' for actions: .play (join voice channel), .stop (leave voice channel)
Week 8	24-06-2024 to 30-06-2024	**Issue During Music Playback**
		• Encountered Error: Unable to Extract Uploader ID
		• Bot Successfully Joins Voice Channel but Unable to Play Songs.
Week 9	01-07-2024 to 07-07-2024	**Bug Fixing**
		• Conduct extensive testing in a development server.
		• Debug and fix any issues found during testing.

Week 10	08-07-2024 to 15-07-2024	**Deployment and Project Closure**
		• Deploy bot to production environment.
		• Close project and release resources.

CHAPTER 4

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

- **Processor**: A multi-core processor with at least 2 cores, such as Intel Core i3 or AMD Ryzen 5, to handle concurrent operations effectively.
- **RAM**: Minimum 2 GB of RAM to ensure smooth performance and responsiveness.
- **Storage**: At least 10 GB of available storage space for storing code, dependencies, and media files.
- **Internet Connection**: A stable internet connection with a minimum speed of 1 Mbps to facilitate real-time communication and media streaming.

SOFTWARE REQUIREMENTS

- **Operating System**: Support for Linux, Windows, or macOS, providing flexibility for development and deployment environments.
- **Discord**: Essential for integrating and testing the bot within Discord servers and channels.
- **Visual Studio Code**: Preferred integrated development environment (IDE) for coding and managing project files.
- **FFmpeg**: Open-source software for handling audio and video processing tasks, crucial for media playback and streaming capabilities.

REQUIRED EXTENSIONS

- **Discord.py[voice]**: Extension for Discord.py library supporting voice functionalities, essential for audio streaming and management.
- **Youtube-dl**: Python library for downloading videos from YouTube and other platforms, enabling access to a wide range of music sources.
- **Discord-components**: Library for integrating interactive components in Discord applications, enhancing user interaction capabilities.
- **Urllib3**: Python library for making HTTP requests, used for fetching and handling data from web sources.
- **PyNaCl**: Python bindings for the Networking and Cryptography (NaCl) library, facilitating secure communication and encryption within the bot.

These requirements and extensions are crucial for setting up a robust development environment and ensuring the Discord music bot can effectively handle audio playback, user interactions, and media management.

CHAPTER 5

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS

1. **Authentication with Discord API**: The bot must authenticate using a bot token to access Discord's features and APIs securely.
2. **Role-Based Access Control**: Implement role-based access control to restrict specific commands to authorized user roles, enhancing security and control.
3. **Voice Channel Operations**: Enable the bot to join and leave voice channels as commanded by users, ensuring seamless integration into Discord voice environments.
4. **Voice Channel State Management**: Implement logic to detect if the bot is already in a voice channel to prevent multiple simultaneous joins, ensuring proper channel management.
5. **Queue Management**: Allow users to add songs to a queue and view the current queue status, facilitating organized playback of requested songs.
6. **Song Search and Playback**: Enable users to search for songs by title, artist, or album, and play songs directly from search results, providing flexibility in music selection.
7. **Concurrency Handling**: Design the bot to handle multiple users and commands concurrently without significant performance degradation, ensuring responsiveness even under load.
8. **Resource Management**: Implement efficient resource management to handle high loads and many simultaneous music requests effectively, optimizing bot performance and reliability.

These functional requirements outline the core capabilities necessary for the Discord music bot to provide a seamless and enjoyable music playback experience while ensuring robust performance and user interaction within Discord servers.

NON-FUNCTIONAL REQUIREMENTS

1. Respond to commands within 2 seconds and handle up to 50 concurrent users.
2. Maintain 99.9% uptime with robust error handling and informative error messages.
3. Provide intuitive commands and clear help documentation for users.
4. Ensure bot token and user data are securely stored and protected.
5. Follow coding best practices for modular, well-documented, and easily updatable code.

CHAPTER 6

DESIGN DOCUMENT

Design Document for Discord Music Bot

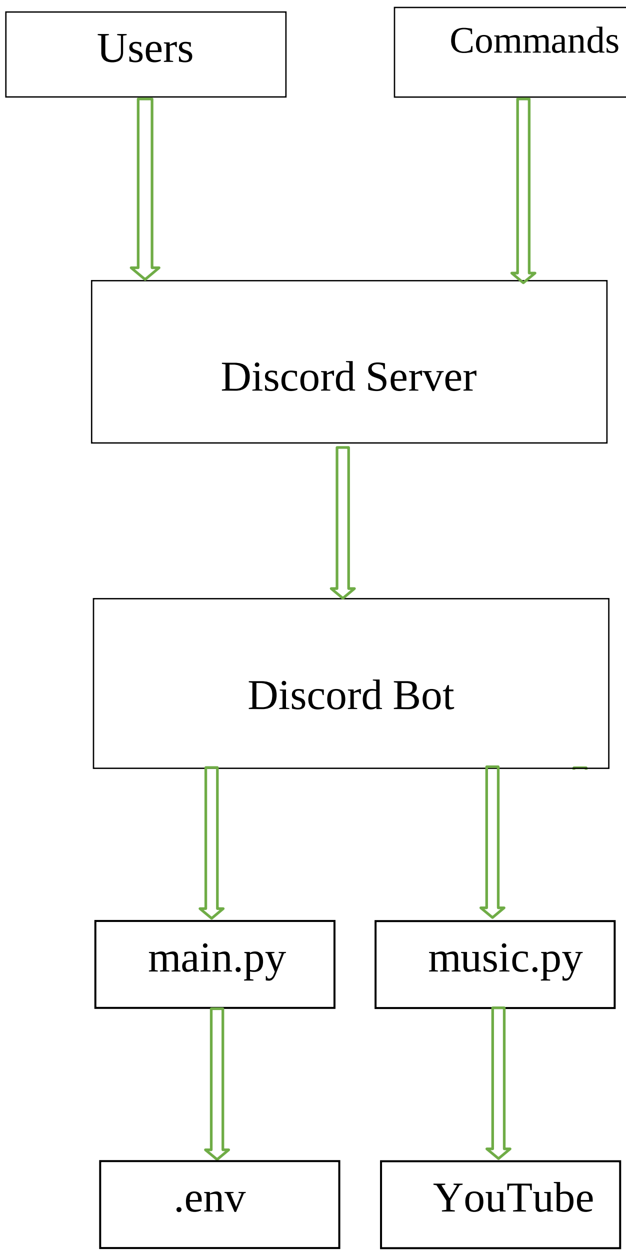


Fig : diagram that outlines the structure and components of the Discord Music Bot

Description of Diagram

1. **Users**: Interact with the bot by sending commands via the Discord server.
2. **Commands**: Examples include .play, .pause, .resume, .queue, .stop, and .help.
3. **Discord Server**: The platform where the bot operates and users send commands.
4. **Discord Bot**: The bot application that processes user commands.
    - **main.py**: Initializes the bot, loads music.py.
    - **musi.py**: Handles music-related commands and manages the music queue.
      Also Provides customized help messages for bot commands.
5. **.env**: Securely stores the bot token required for authentication with the Discord API.
6. **YouTube**: External service used to stream music based on user commands.

This diagram visually represents the structure and interactions of the Discord Music Bot components, making it easier to understand the system's design.

CHAPTER 7

IMPLEMENTATION

Implementation Plan for Discord Music Bot

1. Setup Bot and Load Cogs :

- Initialize Bot: Create the main.py file to initialize the Discord bot client.
- Load Cogs: Set up the bot to load music.py.

2. Implement Music Commands :

- In music.py, implement the .play <songname> command to play music from YouTube.
- Implement commands for .pause, .resume, .resume, and .stop to control music playback.
- Handle a music queue to manage multiple songs.
- Custom Help: Override the default help command to provide detailed descriptions of available commands.

3. Testing and Debugging :

- Write unit tests for each command to ensure they function correctly.
- Test the interaction between commands and ensure they work seamlessly together.
- Identify and fix any issues found during testing

4. Deployment Plan :

- Set up a cloud server to host the bot and ensure .env is securely stored and accessed.
- Monitor the deployment, and if any issues arise, revert to the previous stable version.

5. Maintenance and Support :

- Keep the bot updated with the latest Discord API changes and fix any bugs reported by users.
- Provide a dedicated Discord channel for user support and use a ticketing system to track and resolve issues

CHAPTER 8

TESTING PLAN AND IMPLEMENTATION

Testing Plan for Discord Music Bot

Command	**Test Case**	**Expected Result**	**Actual Result**	**Remarks**
.play	Play a valid YouTube URL	Music starts playing	Music starts playing	PASS
	Play an invalid YouTube URL	Error	Error	PASS
	Play while another song is playing	New song is added to the queue	New song is added to the queue	PASS
.pause	Pause current playing music	Music pauses	Music pauses	PASS
	Pause when no music is playing	Error	Error	PASS
.queue	Show current queue when songs are queued	List of queued songs is displayed	List of queued songs is displayed	PASS
.resume	Resume paused music	Music resumes playing	Music resumes playing	PASS
	Resume when no music is paused	Error	Error	PASS
.stop	Stop current playing music	Music stops and queue is cleared	Music stops and queue is cleared	PASS
	Stop when no music is playing	Error	Error	PASS
.help	Show help for all commands	Help message with descriptions of all commands is displayed	Help message with descriptions of all commands is displayed	PASS

CHAPTER 9

LIST OF FILES

List of Files for Discord Music Bot

1. **main.py**
   - Initializes the bot.
   - Loads music.py.
2. **music.py**
   - Contains commands for music control: .play, .pause, .queue, .resume, .stop.
   - Manages music playback and queue.
   - Customizes the help command.
   - Provides detailed descriptions for all bot commands.
3. **.env**
   - Stores the bot token securely for authentication.

main.py

- **Purpose**: This file is the starting point of the bot. It sets up and runs the bot.
- **Contents**:
    - **Bot Initialization**: Code to create the bot instance and connect it to Discord.
    - **Loading Extensions**: Commands to load additional functionality from other files (music.py).
    - **Basic Events**: Handles basic events, like printing a message when the bot is ready to use.

music.py

- **Purpose**:
    - This file handles all the music-related features of the bot.
    - This file also customizes the help messages that the bot provides.
- **Contents**:
    - **Play Music**: Code to play music from a YouTube URL when a user types a command like .play <songname>
    - **Pause Music**: Code to pause the currently playing music when a user types .pause.
    - **Resume Music**: Code to resume paused music when a user types .resume.
    - **Stop Music**: Code to stop the music and clear the queue when a user types .stop.
    - **Show Queue**: Code to display the list of songs in the queue when a user types .queue.
    - **Custom Help Command**: Code that defines what the bot should reply when a user types .help.
    - **Command Descriptions**: Detailed descriptions of each command, so users understand how to use the bot effectively.

.env

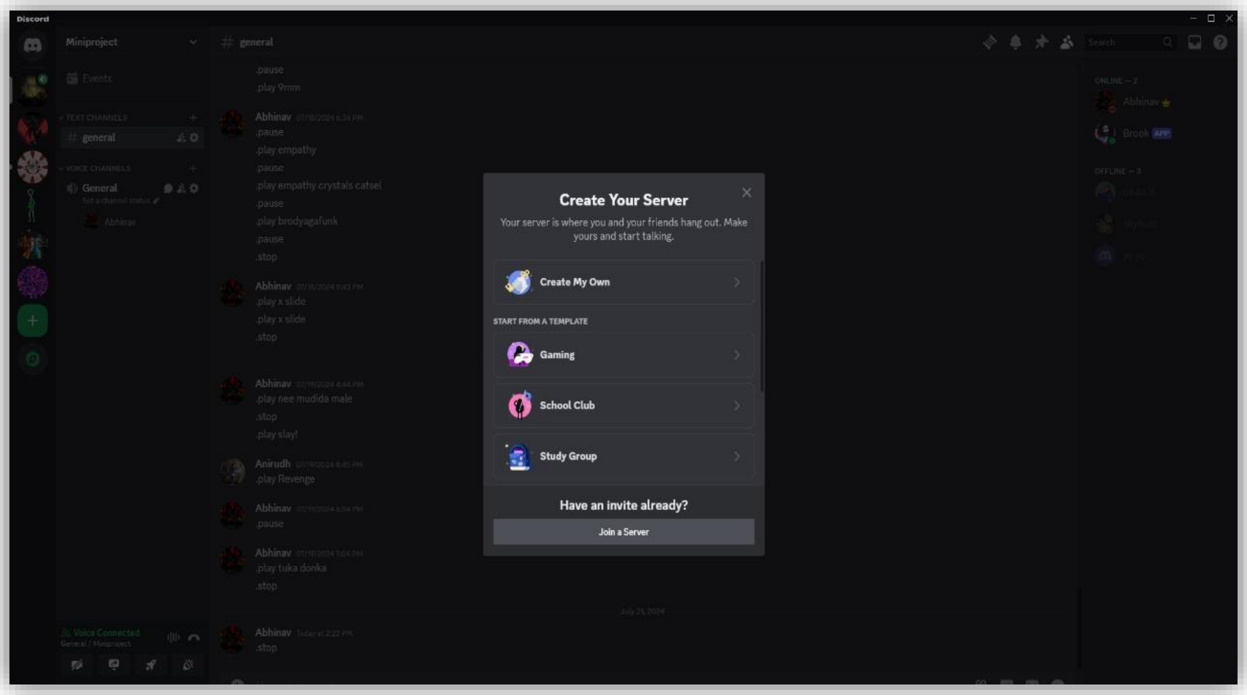
- **Purpose**: This file securely stores the bot token, which is a secret key needed to log in to Discord.
- **Contents**:
    - **Bot Token**: A unique string of characters that allows the bot to connect to Discord. This file is read by mussic.py to authenticate the bot.

CHAPTER 10

SCREENSHOTS

SCREENSHOTS

1. Creating Discord Server



Discord

Miniproject

general

.pause
.play 9mm

Abhinav 07/18/2024 6:36 PM
.pause
.play empathy
.pause
.play empathy crystals catsel
.pause
.play brodyagafunk
.pause
.stop

Abhinav 07/18/2024 9:43 PM
.play x slide
.play x slide
.stop

Abhinav 07/19/2024 4:44 PM
.play nee mudida male
.stop
.play slay!

Anirudh 07/19/2024 4:45 PM
.play Revenge

Abhinav 07/19/2024 6:54 PM
.pause

Abhinav 07/19/2024 7:04 PM
.play tuka donka
.stop

Abhinav Today at 2:22 PM
.stop

Create Your Server

Your server is where you and your friends hang out. Make yours and start talking.

Create My Own

START FROM A TEMPLATE

Gaming

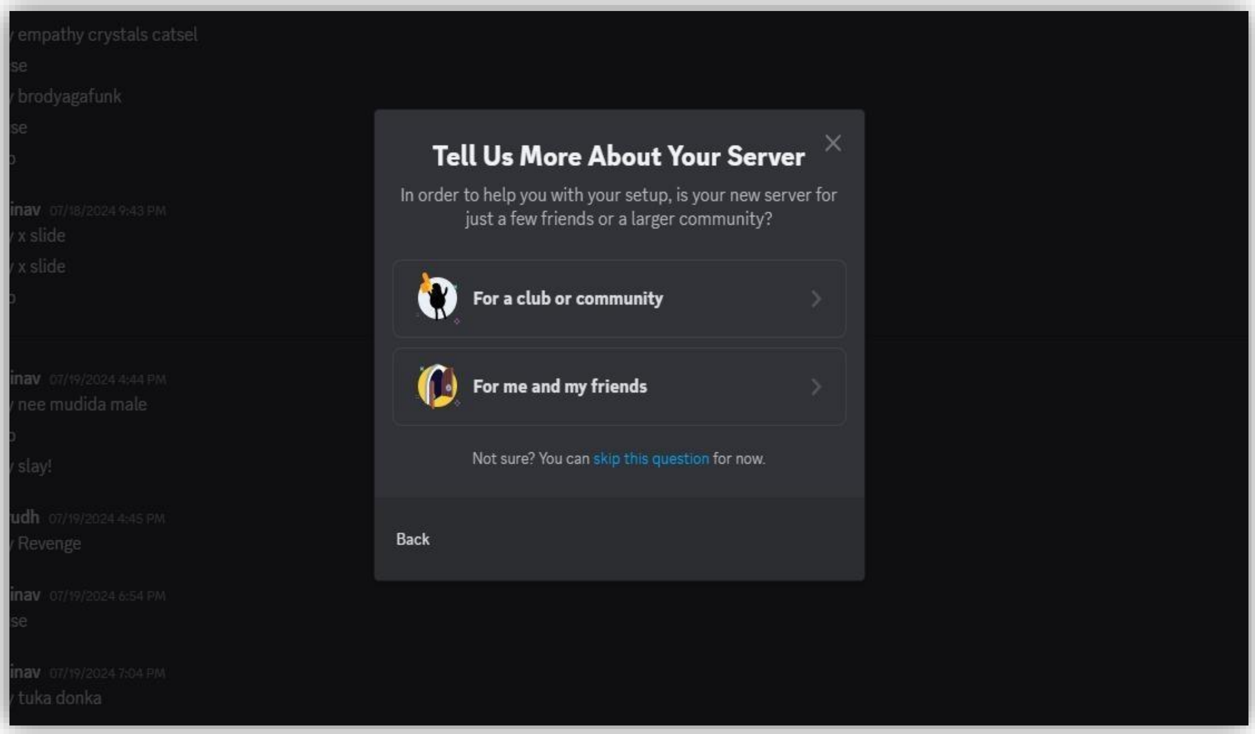
School Club

Study Group

Have an invite already?

Join a Server

Fig : (1.1)



Tell Us More About Your Server

In order to help you with your setup, is your new server for just a few friends or a larger community?

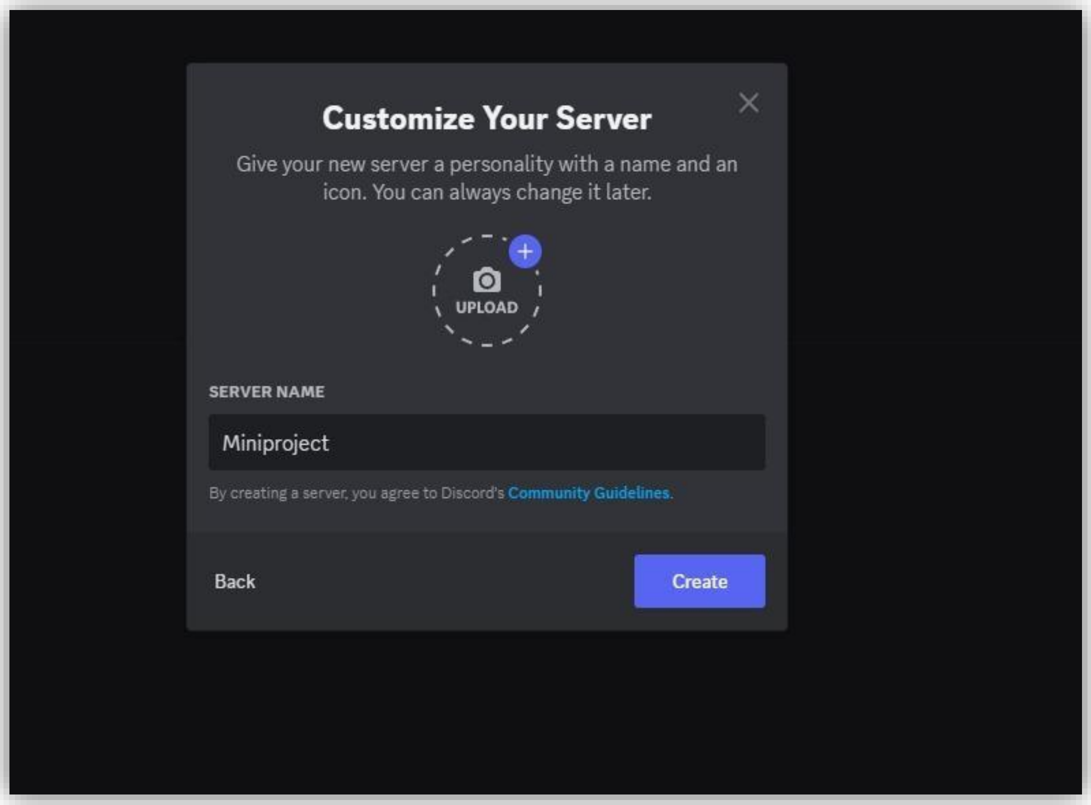
For a club or community

For me and my friends

Not sure? You can skip this question for now.

Back

Fig : (1.2)


Customize Your Server

Give your new server a personality with a name and an icon. You can always change it later.

SERVER NAME

Miniproject

By creating a server, you agree to Discord's Community Guidelines.

Fig : (1.3)

Fig : 1.1, 1.2 and 1.3 illustrates the creation of discord server.

2. Creating Bot

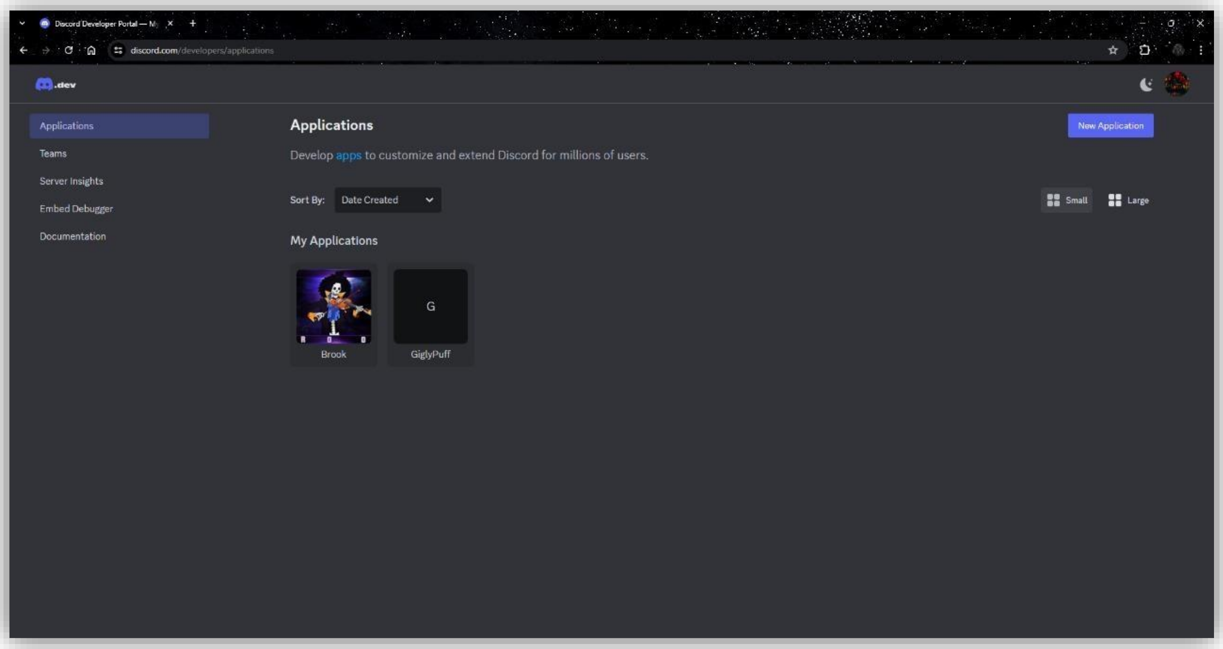


Fig : (2.1)

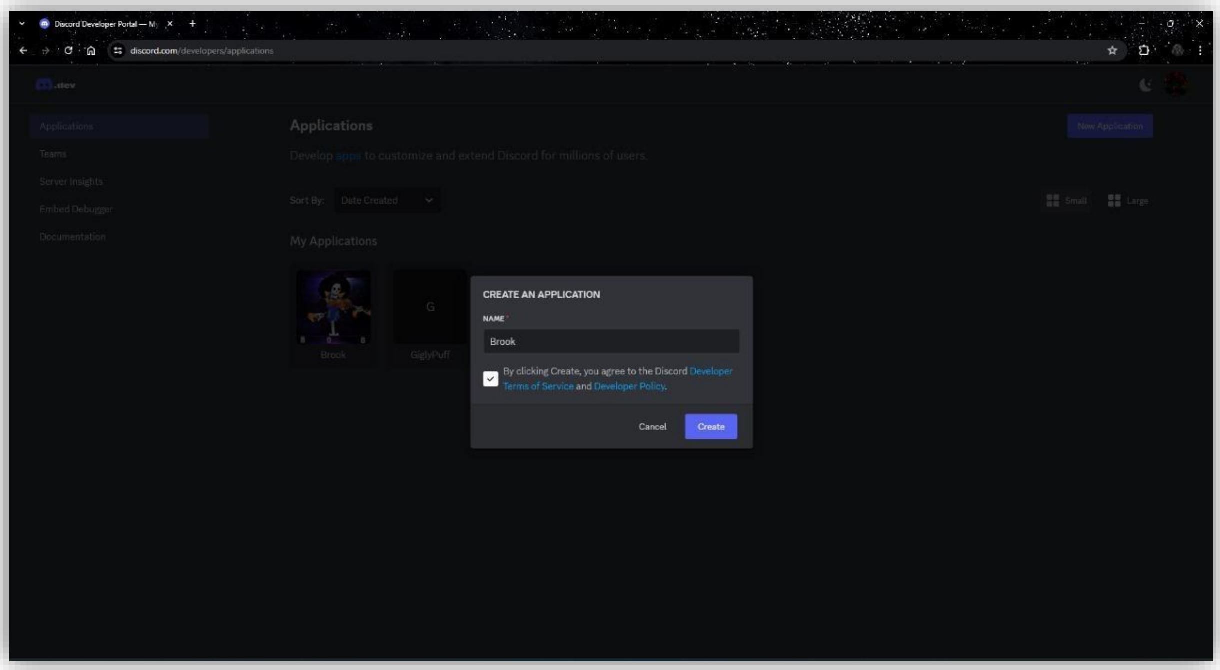
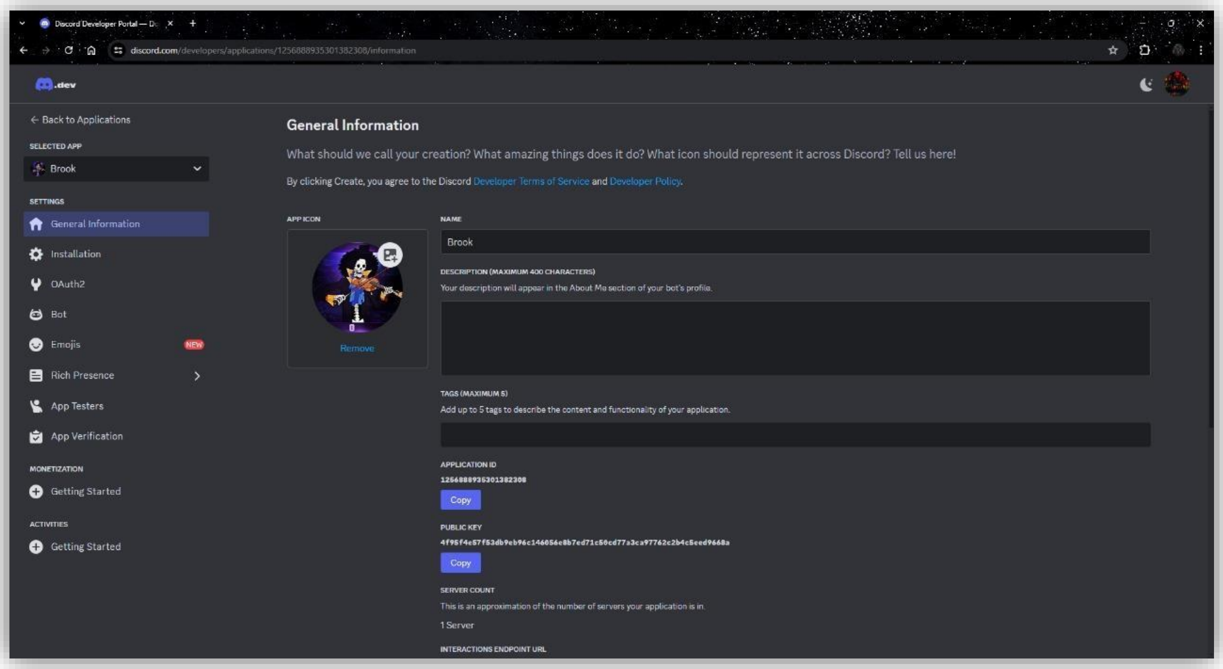

Discord Developer Portal — My
discord.com/developers/applications
CREATE AN APPLICATION
NAME
Brook
By clicking Create, you agree to the Discord Developer Terms of Service and Developer Policy.
Cancel
Create

Fig : (2.2)



General Information

What should we call your creation? What amazing things does it do? What icon should represent it across Discord? Tell us here!

By clicking Create, you agree to the Discord Developer Terms of Service and Developer Policy.

APP ICON

NAME
Brook

DESCRIPTION (MAXIMUM 400 CHARACTERS)
Your description will appear in the About Me section of your bot's profile.

TAGS (MAXIMUM 5)
Add up to 5 tags to describe the content and functionality of your application.

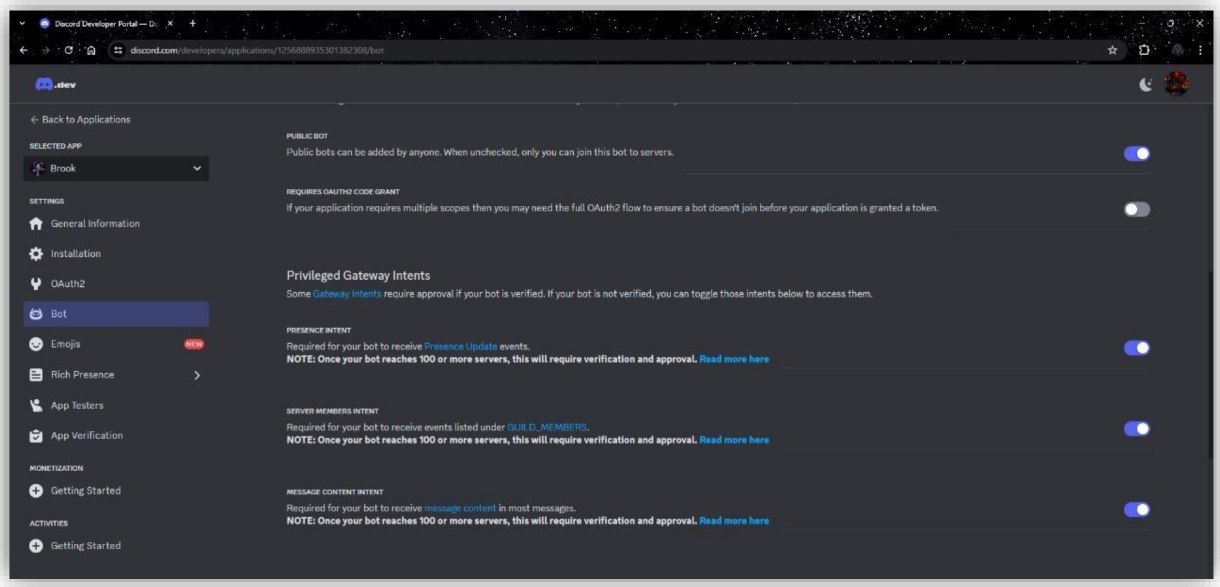
APPLICATION ID
1256888935301382308

PUBLIC KEY
4f95f4e57f53db9eb96c146056e8b7ed71c50cd77a3ca97762c2b4c5eed9668a

SERVER COUNT
This is an approximation of the number of servers your application is in.
1 Server

INTERACTIONS ENDPOINT URL

Fig : (2.3)



PUBLIC BOT
Public bots can be added by anyone. When unchecked, only you can join this bot to servers. [✓]

REQUIRES OAUTH2 CODE GRANT
If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

Privileged Gateway Intents

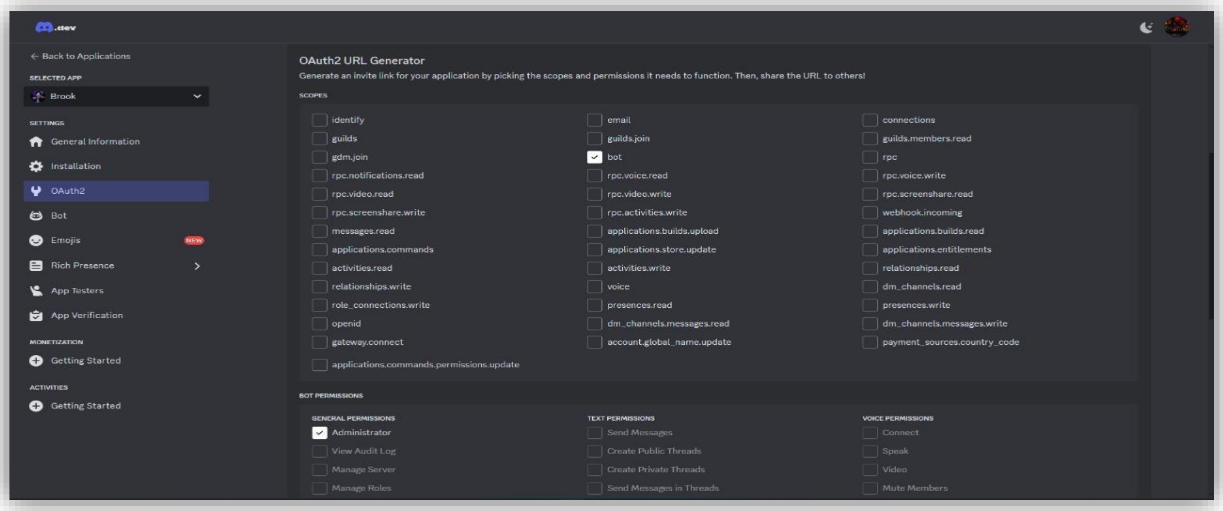
Some Gateway Intents require approval if your bot is verified. If your bot is not verified, you can toggle those intents below to access them.

PRESENCE INTENT
Required for your bot to receive Presence Update events. [✓]
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. Read more here

SERVER MEMBERS INTENT
Required for your bot to receive events listed under GUILD_MEMBERS. [✓]
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. Read more here

MESSAGE CONTENT INTENT
Required for your bot to receive message content in most messages. [✓]
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. Read more here

Fig : (2.4)



OAuth2 URL Generator

Generate an invite link for your application by picking the scopes and permissions it needs to function. Then, share the URL to others!

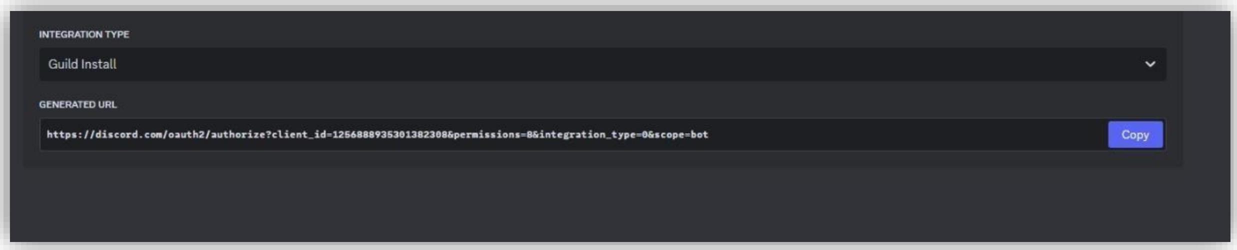
SCOPES

☐ identify	☐ email	☐ connections
☐ guilds	☐ guilds.join	☐ guilds.members.read
☐ gdm.join	☑ bot	☐ rpc
☐ rpc.notifications.read	☐ rpc.voice.read	☐ rpc.voice.write
☐ rpc.video.read	☐ rpc.video.write	☐ rpc.screenshare.read
☐ rpc.screenshare.write	☐ rpc.activities.write	☐ webhook.incoming
☐ messages.read	☐ applications.builds.upload	☐ applications.builds.read
☐ applications.commands	☐ applications.store.update	☐ applications.entitlements
☐ activities.read	☐ activities.write	☐ relationships.read
☐ relationships.write	☐ voice	☐ dm_channels.read
☐ role_connections.write	☐ presences.read	☐ presences.write
☐ openid	☐ dm_channels.messages.read	☐ dm_channels.messages.write
☐ gateway.connect	☐ account.global_name.update	☐ payment_sources.country_code
☐ applications.commands.permissions.update		

BOT PERMISSIONS

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
☑ Administrator	☐ Send Messages	☐ Connect
☐ View Audit Log	☐ Create Public Threads	☐ Speak
☐ Manage Server	☐ Create Private Threads	☐ Video
☐ Manage Roles	☐ Send Messages in Threads	☐ Mute Members

Fig : (2.5)



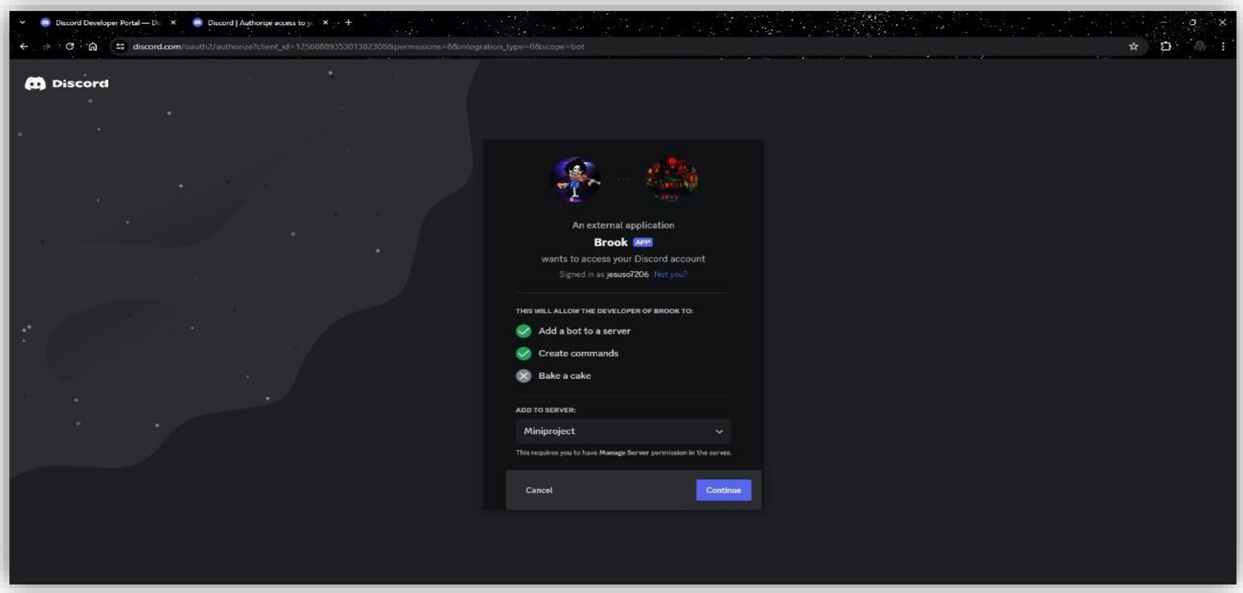
INTEGRATION TYPE

Guild Install

GENERATED URL

```
https://discord.com/oauth2/authorize?client_id=1256888935301382308&permissions=8&integration_type=0&scope=bot
```

Fig : (2.6)



discord.com/oauth2/authorize?client_id=1256888935301382308&permissions=8&integration_type=0&scope=bot

An external application

Brook APP

wants to access your Discord account

Signed in as jesuso7206 Not you?

THIS WILL ALLOW THE DEVELOPER OF BROOK TO:

- ✓ Add a bot to a server
- ✓ Create commands
- ✗ Bake a cake

ADD TO SERVER:

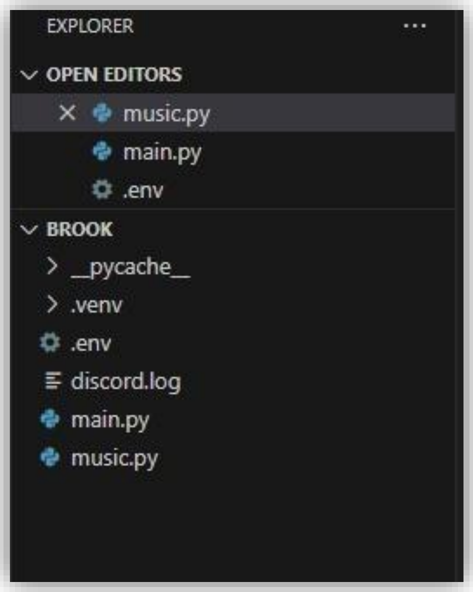
Miniproject

This requires you to have **Manage Server** permission in the server.

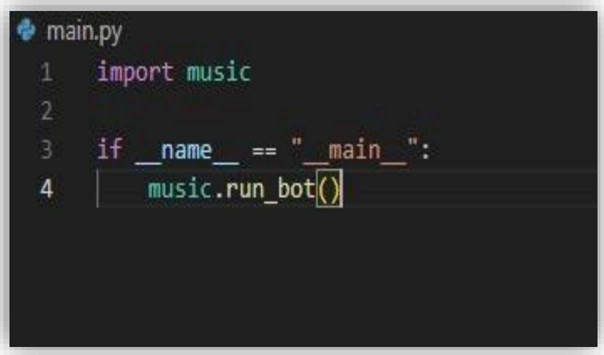
Fig : (2.7)

Fig : 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 and 2.7 illustrates the creation of bot.

3. Code Snippets



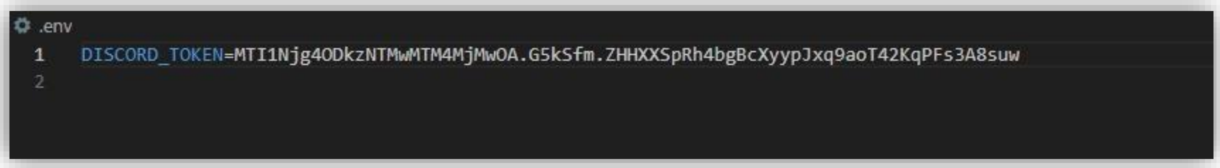
```
EXPLORER
OPEN EDITORS
  × music.py
    main.py
    .env
BROOK
  > __pycache__
  > .venv
  .env
  discord.log
  main.py
  music.py
```



```
main.py
import music

if __name__ == "__main__":
    music.run_bot()
```

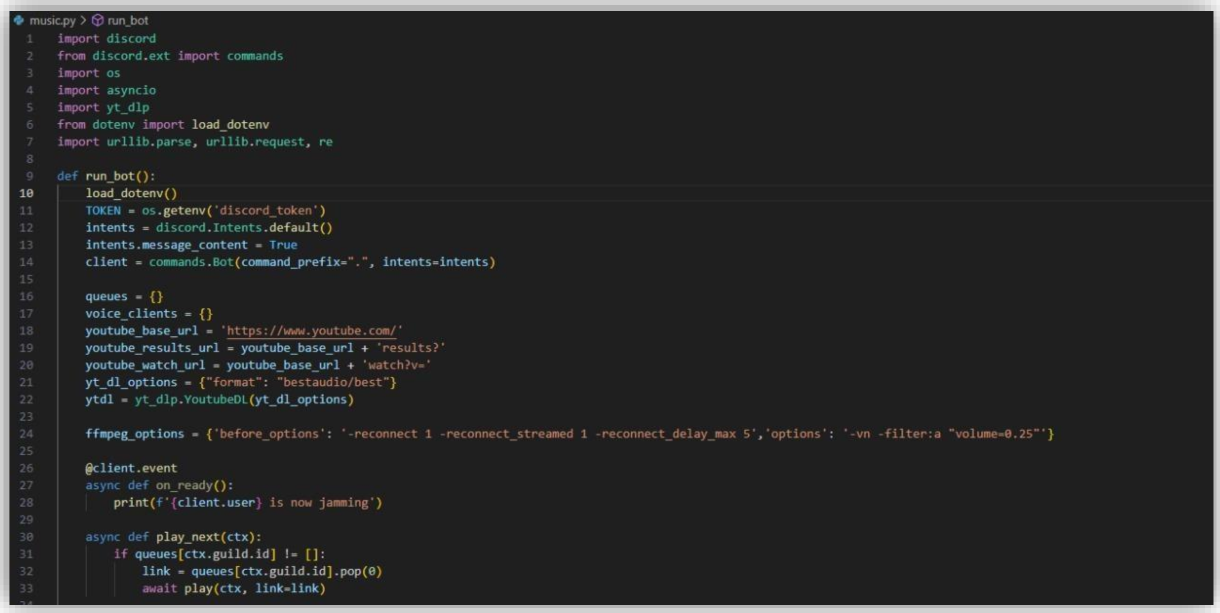
Fig : (3.1)

Fig : (3.2)



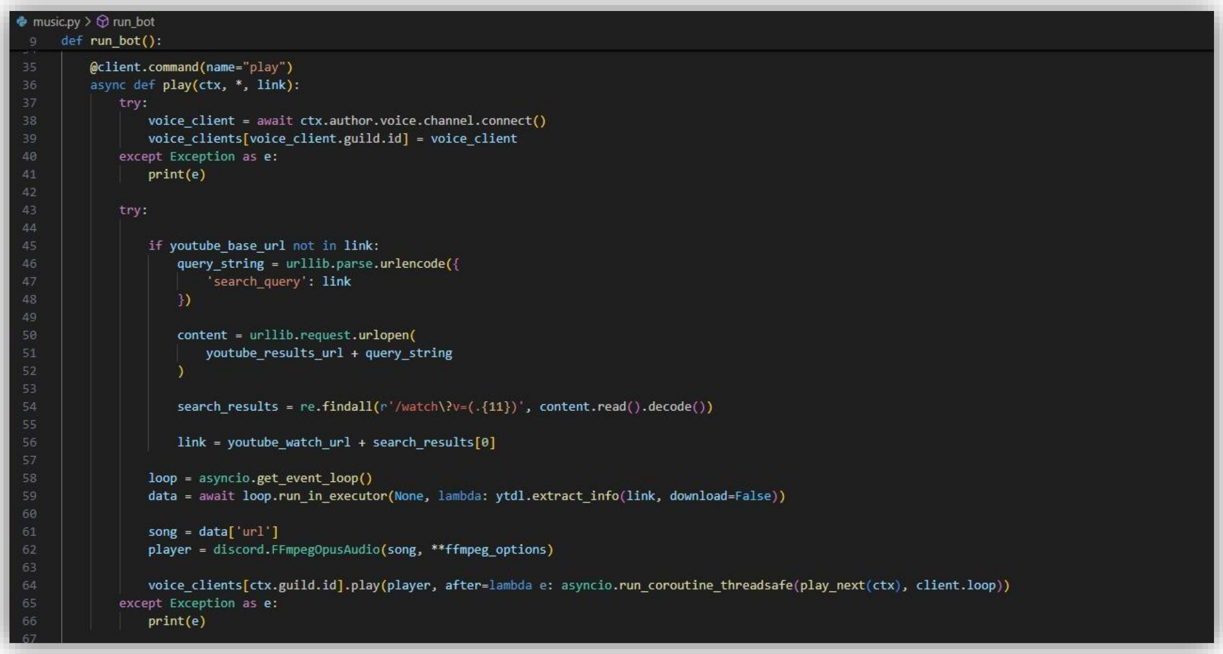
```
.env
DISCORD_TOKEN=MTI1Njg4ODkzNTMwMTM4MjMwOA.G5kSfm.ZHHXXSpRh4bgBcXyypJxq9aoT42KqPFs3A8suw
```

Fig : (3.3)



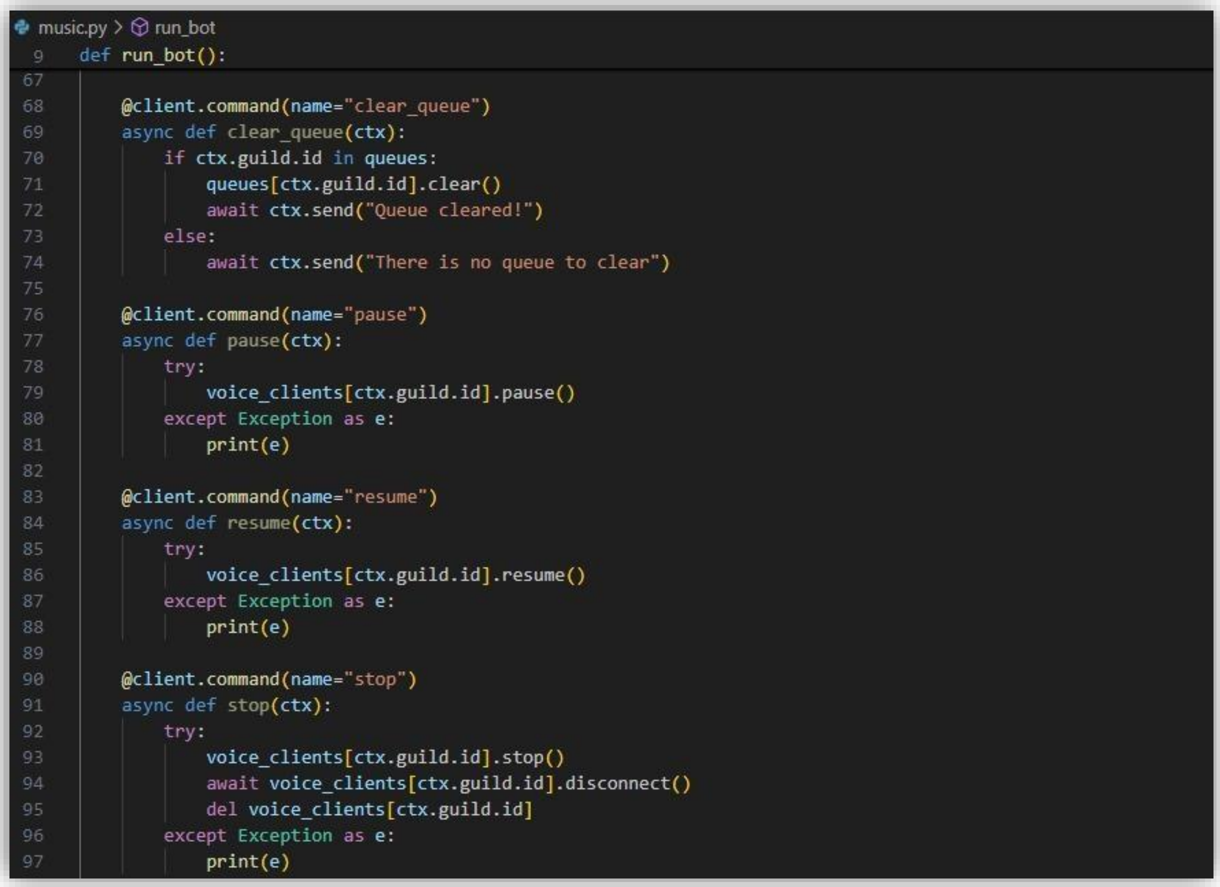
```
music.py > run_bot
import discord
from discord.ext import commands
import os
import asyncio
import yt_dlp
from dotenv import load_dotenv
import urllib.parse, urllib.request, re

def run_bot():
    load_dotenv()
    TOKEN = os.getenv('discord_token')
    intents = discord.Intents.default()
    intents.message_content = True
    client = commands.Bot(command_prefix=".", intents=intents)

    queues = {}
    voice_clients = {}
    youtube_base_url = 'https://www.youtube.com/'
    youtube_results_url = youtube_base_url + 'results?'
    youtube_watch_url = youtube_base_url + 'watch?v='
    yt_dl_options = {"format": "bestaudio/best"}
    ytdl = yt_dlp.YoutubeDL(yt_dl_options)

    ffmpeg_options = {'before_options': '-reconnect 1 -reconnect_streamed 1 -reconnect_delay_max 5','options': '-vn -filter:a "volume=0.25"'}

    @client.event
    async def on_ready():
        print(f'{client.user} is now jamming')

    async def play_next(ctx):
        if queues[ctx.guild.id] != []:
            link = queues[ctx.guild.id].pop(0)
            await play(ctx, link=link)
```

Fig : (3.4)



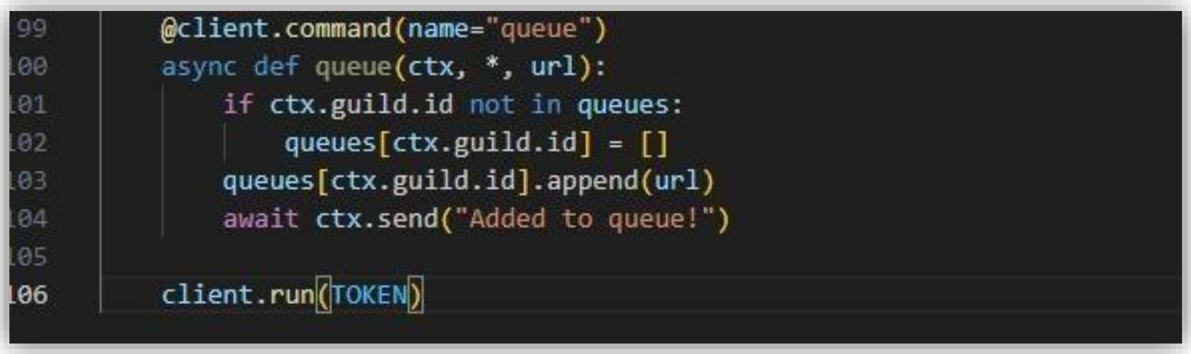
```
music.py > run_bot
def run_bot():

    @client.command(name="play")
    async def play(ctx, *, link):
        try:
            voice_client = await ctx.author.voice.channel.connect()
            voice_clients[voice_client.guild.id] = voice_client
        except Exception as e:
            print(e)

        try:

            if youtube_base_url not in link:
                query_string = urllib.parse.urlencode({
                    'search_query': link
                })

                content = urllib.request.urlopen(
                    youtube_results_url + query_string
                )

                search_results = re.findall(r'/watch\?v=(.{11})', content.read().decode())

                link = youtube_watch_url + search_results[0]

            loop = asyncio.get_event_loop()
            data = await loop.run_in_executor(None, lambda: ytdl.extract_info(link, download=False))

            song = data['url']
            player = discord.FFmpegOpusAudio(song, **ffmpeg_options)

            voice_clients[ctx.guild.id].play(player, after=lambda e: asyncio.run_coroutine_threadsafe(play_next(ctx), client.loop))
        except Exception as e:
            print(e)
```

Fig : (3.5)



```
music.py > run_bot
def run_bot():

    @client.command(name="clear_queue")
    async def clear_queue(ctx):
        if ctx.guild.id in queues:
            queues[ctx.guild.id].clear()
            await ctx.send("Queue cleared!")
        else:
            await ctx.send("There is no queue to clear")

    @client.command(name="pause")
    async def pause(ctx):
        try:
            voice_clients[ctx.guild.id].pause()
        except Exception as e:
            print(e)

    @client.command(name="resume")
    async def resume(ctx):
        try:
            voice_clients[ctx.guild.id].resume()
        except Exception as e:
            print(e)

    @client.command(name="stop")
    async def stop(ctx):
        try:
            voice_clients[ctx.guild.id].stop()
            await voice_clients[ctx.guild.id].disconnect()
            del voice_clients[ctx.guild.id]
        except Exception as e:
            print(e)
```

Fig : (3.6)

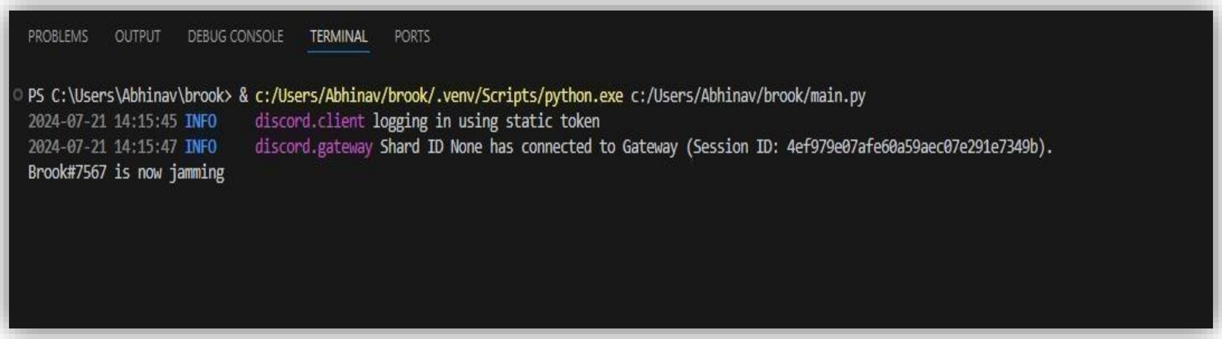


```python
@client.command(name="queue")
async def queue(ctx, *, url):
    if ctx.guild.id not in queues:
        queues[ctx.guild.id] = []
    queues[ctx.guild.id].append(url)
    await ctx.send("Added to queue!")

client.run(TOKEN)
```

Fig : (3.7)

Fig : 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7 are a few Code Snippets.

4. Working of Music Bot



```
PS C:\Users\Abhinav\brook> & c:/Users/Abhinav/brook/.venv/Scripts/python.exe c:/Users/Abhinav/brook/main.py
2024-07-21 14:15:45 INFO     discord.client logging in using static token
2024-07-21 14:15:47 INFO     discord.gateway Shard ID None has connected to Gateway (Session ID: 4ef979e07afe60a59aec07e291e7349b).
Brook#7567 is now jamming
```

Fig : (4.1)

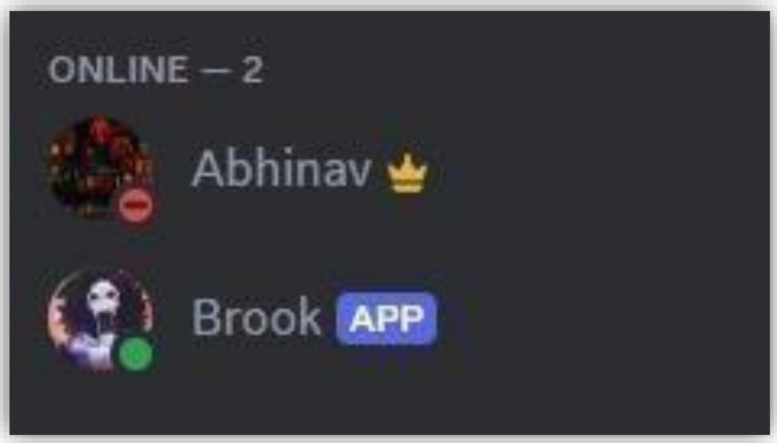
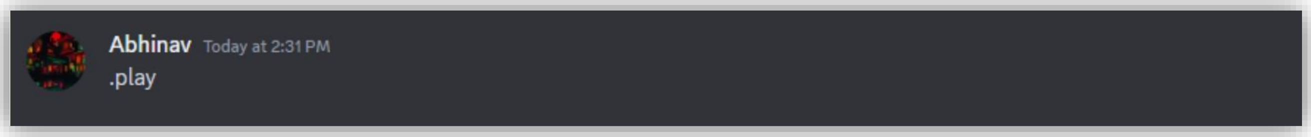


Fig : (4.2)



Abhinav Today at 2:31 PM
.play

Fig : (4.3)

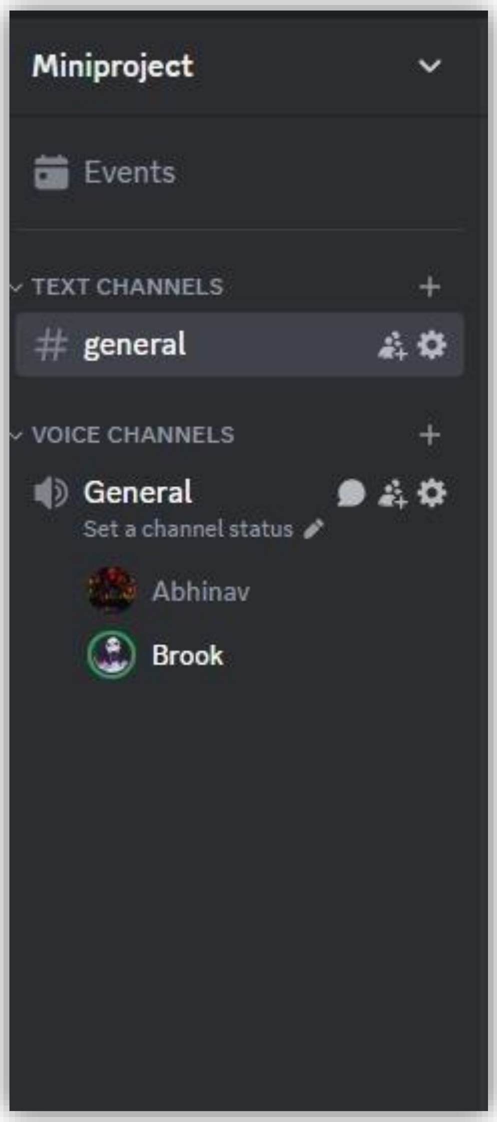
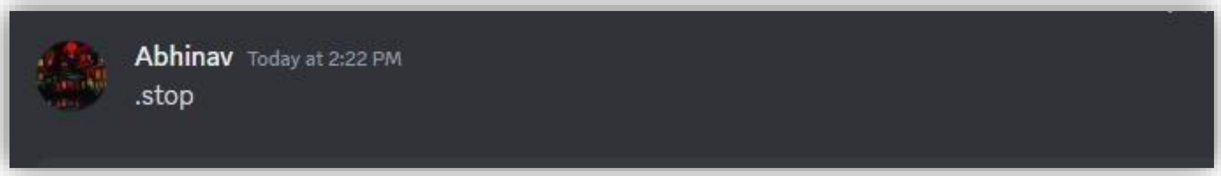


Fig : (4.4)



Abhinav Today at 2:22 PM
.stop

Fig : (4.5)

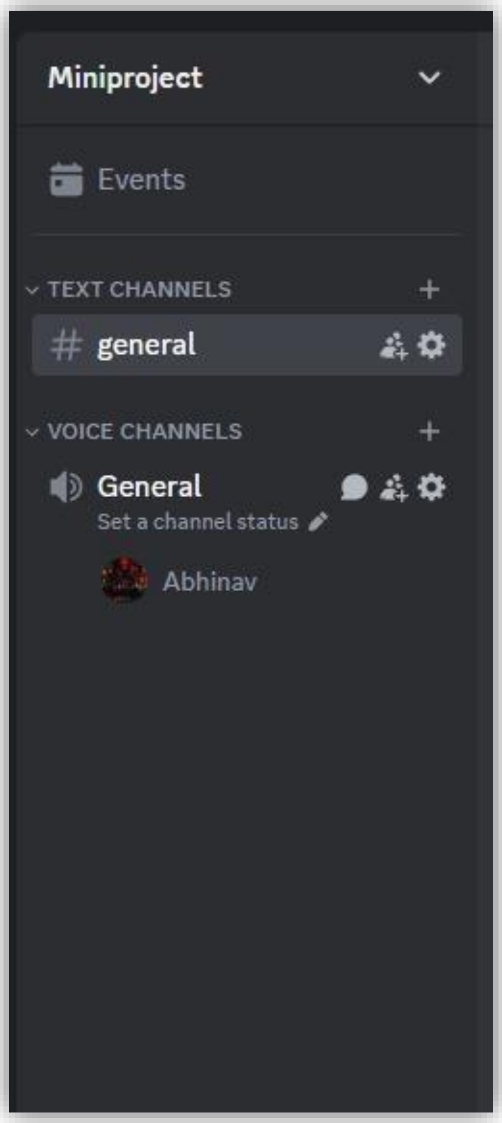
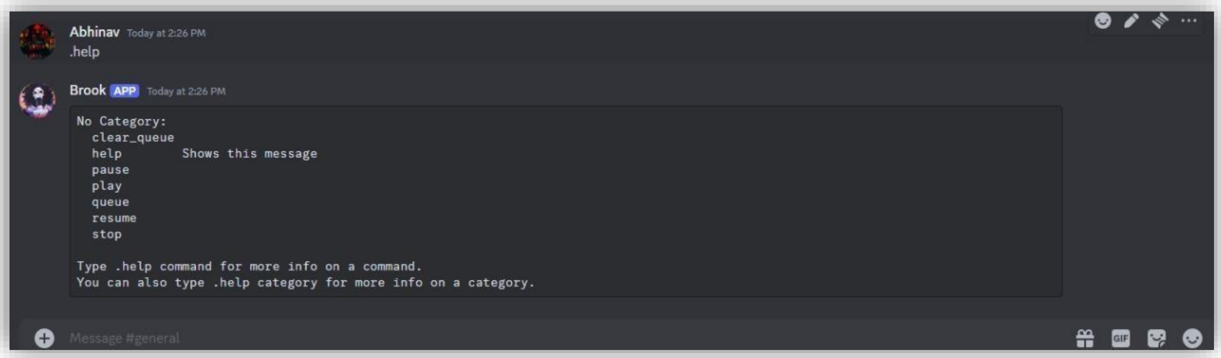


Fig : (4.6)



Abhinav Today at 2:26 PM
.help

Brook APP Today at 2:26 PM

```
No Category:
  clear_queue
  help        Shows this message
  pause
  play
  queue
  resume
  stop

Type .help command for more info on a command.
You can also type .help category for more info on a category.
```

Fig : (4.7)

Fig : 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 illustrates the Working of Bot.