Automated Tweets Powered by BeagleBone AI

Kavyan Nasseri, 217829854

Harnoor Singh, 218020206

Dayeon Lee, 21789763

Faraz Akbarzadeh, 217208638

Project Summary

This project leverages the Internet of Things (IoT) capabilities of the BeagleBone AI board to create an interactive Twitter trivia bot. The system is designed to autonomously generate and post trivia questions on Twitter at regular 5 minute intervals, engaging followers with interactive polls to gather their responses. Trivia questions are sourced from the Open Trivia Database and seamlessly shared as tweets as long as the Python code is actively running on the Cloud 9 IDE hosted on the BeagleBone AI board. In an era where social media engagement is crucial, the Twitter Trivia Bot offers a unique and entertaining way for Twitter users to test their knowledge and have fun within the Twitter platform.

Use Cases of IOT BeagleBone

Smart Home Automation

BeagleBone AI can be used to create a smart home system, enabling users to control lights, thermostats, security cameras, and other devices remotely through IoT applications.
Industrial Monitoring and Control: In industrial settings, the BeagleBone AI can be employed for monitoring and controlling machinery, collecting real-time data on equipment performance, and automating processes to enhance efficiency.

Healthcare Applications

The board can be utilized in healthcare to develop IoT solutions for patient monitoring, tracking medical equipment, and managing health data securely, contributing to the advancement of telemedicine and remote patient care.

Environmental Monitoring

BeagleBone AI can be deployed for environmental monitoring, such as tracking air quality, soil conditions, and weather patterns. This data can be valuable for research, agriculture, and urban planning.

Edge AI and Machine Learning

With its AI capabilities, BeagleBone AI is suitable for edge computing applications. This includes implementing machine learning algorithms for image recognition, speech processing, or predictive maintenance, bringing intelligence to devices at the edge of the network.

Content Scheduler

The BeagleBone AI can be utilized to create a content scheduler that automates the process of creating content at specific intervals.

System Architecture

Hardware Requirements

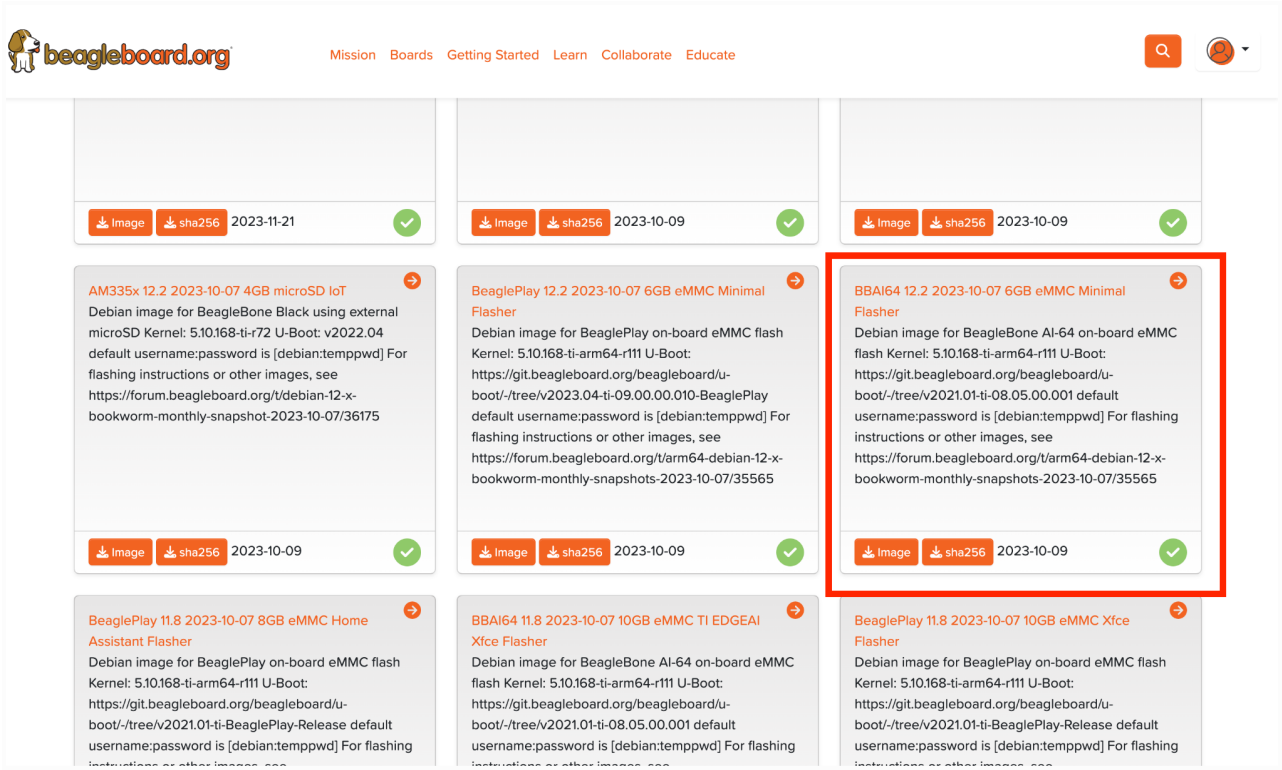
Mini SD Card

USB C Cable

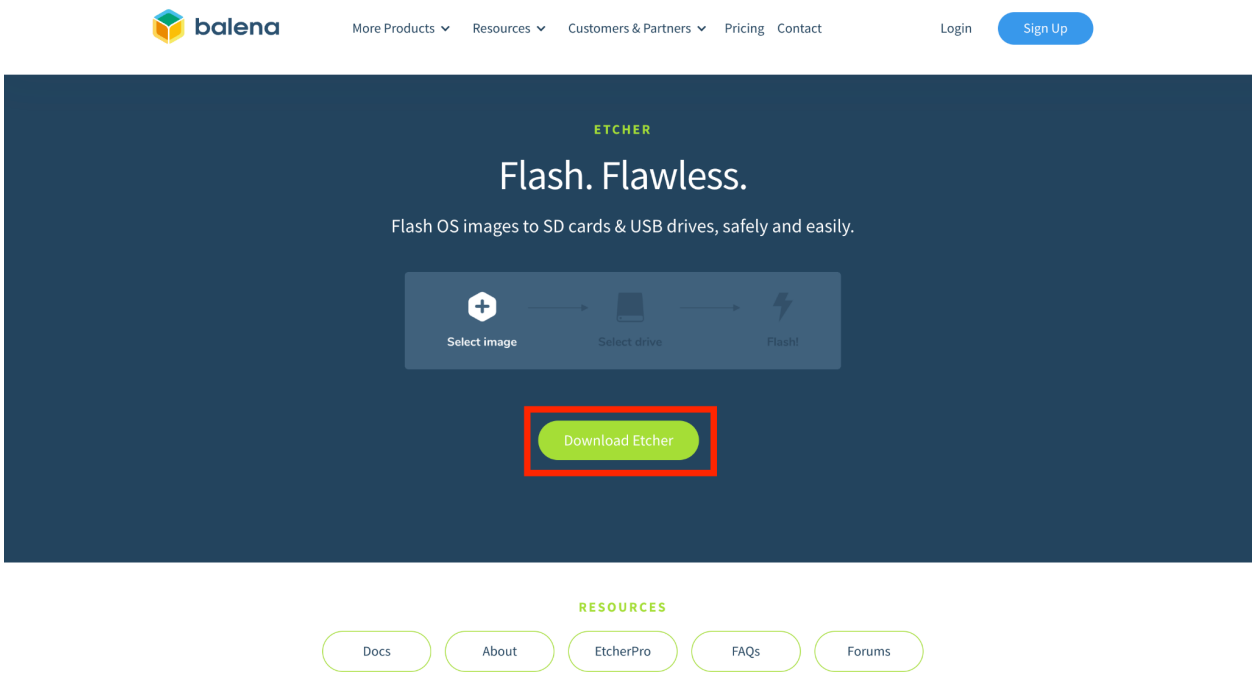
BeagleBone AI

Software Requirements

1. Go to the following webpage https://www.beagleboard.org/distros and download the latest debian image for the BeagleBone AI board, in this case, am57xx-eMMC-flasher-debian-10.3-iot-tidl-armhf-2020-04-06-6gb.img.xz



2. Go to the following link https://etcher.balena.io and download Etcher



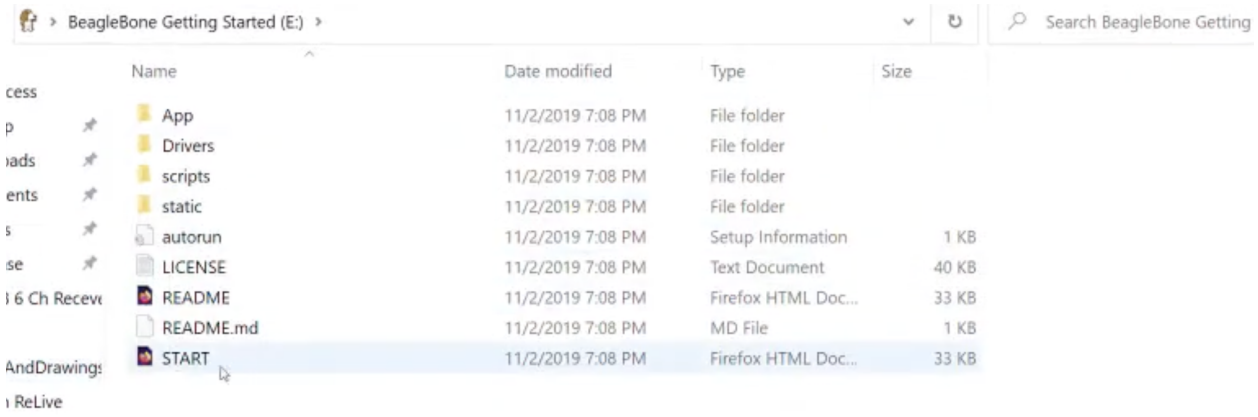
.

Setting up The Board

After downloading Etcher, and the Debian image, insert the SD card into your computer and open up Balena Etcher. Select “Flash From File” and select the downloaded Debian image from your downloads folder. Once the SD card has been flashed you should receive an alert saying that it is no longer readable by your computer.

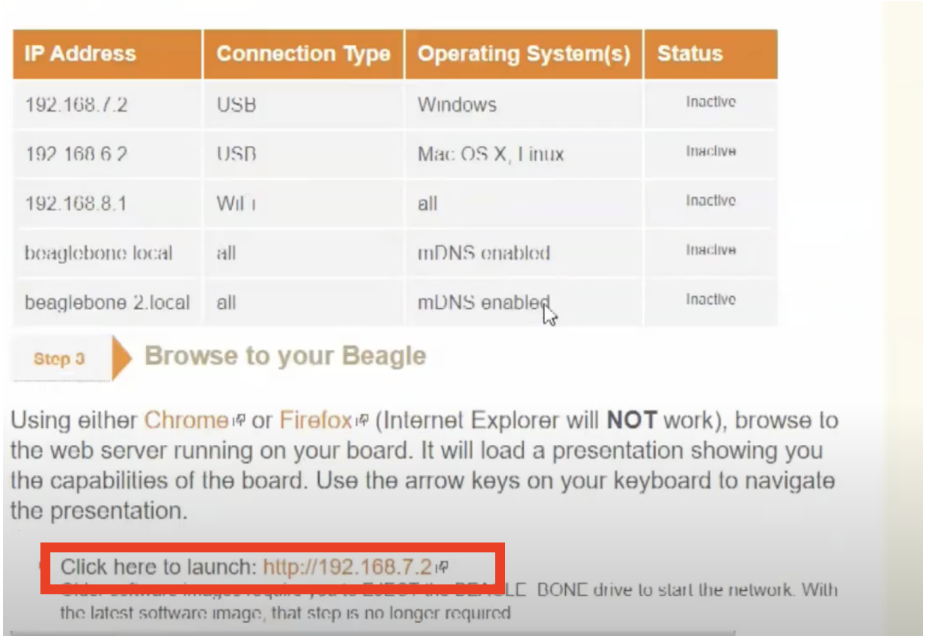
Remove the card, make sure the BeagleBone AI board is powered off, insert the SD card in the correlating port on the board and connect it to power. Within 5 minutes the LED's on the board must start chasing each other, that means it is flashing on EMMC. Once the installation is finished the board turns itself off.

After powering the board on again, it should appear as an image on your device and you will see the following files



Open the start.html file and you will be greeted with the following page, click on the IP address

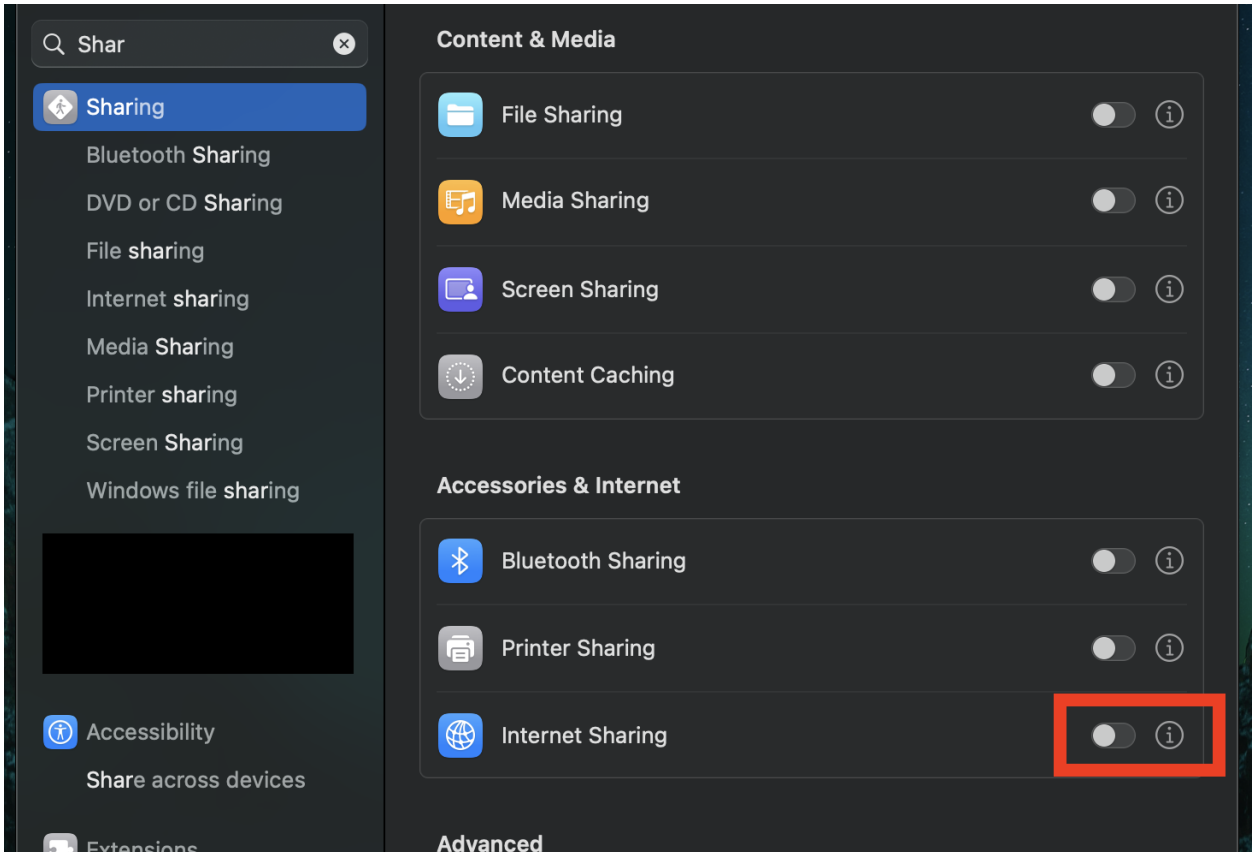
provided in step 3 to open up the Cloud 9 IDE



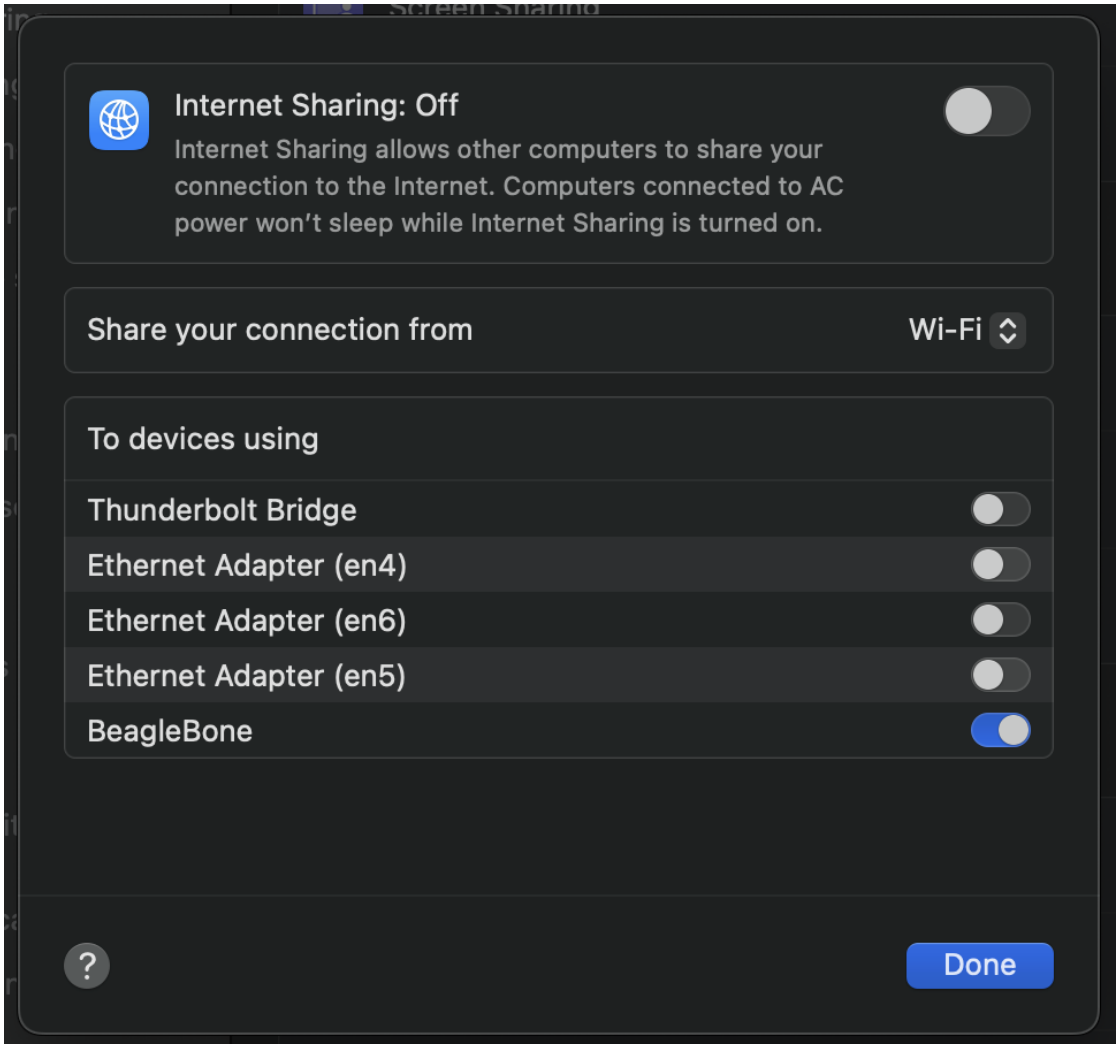
Cloud 9 is the IDE where we will be running our python code on.

Providing Internet To The Board

In order to install the required modules to successfully run the code we need to provide internet to the board. To do so, we can share the Wifi connection of our device through the USB C cable. On a Mac, go to System Settings>Sharing and click on the information icon next to Internet Sharing.



In the pop-up window choose “Wi-fi” from the dropdown list of connections and turn on BeagleBone so the connection could be shared to it. At the end, turn on Internet Sharing.



Establishing Connection

It's important to note that the device that is sharing its wifi cannot be connected to the board over wifi for obvious reasons. Use another device to perform the following steps, and make sure to repeat these steps every single time you are restarting the connection.

Open up terminal and enter the following command :

```
ssh debian@beaglebone.local
```

For the password enter : temppwd

```
deepakg@Deepaks-MacBook-Air-2 ~ % ssh debian@beaglebone.local
Debian GNU/Linux 10

BeagleBoard.org Debian Buster IoT Image 2021-10-01
Support: https://bbb.io/debian
default username:password is [debian:temppwd]

debian@beaglebone.local's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Once an SSH connection is established run the following command :

```
sudo ifconfig
```

See which port the BeagleBone AI board is connected to, by seeing where the address

192.168.6.2 appears, in this case it is usb1

```
        TX packets 328  bytes 23040 (22.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.7.2  netmask 255.255.255.0  broadcast 192.168.7.255
        inet6 fe80::e615:f6ff:fefc:24ec  prefixlen 64  scopeid 0x20<link>
        ether e4:15:f6:fc:24:ec  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.6.2  netmask 255.255.255.0  broadcast 192.168.6.255
        inet6 fd62:e6a2:cc98:6fcd:e615:f6ff:fefc:24f0  prefixlen 64  scopeid 0x0
<global>
        inet6 fe80::e615:f6ff:fefc:24f0  prefixlen 64  scopeid 0x20<link>
        ether e4:15:f6:fc:24:f0  txqueuelen 1000  (Ethernet)
        RX packets 288  bytes 75622 (73.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 152  bytes 30924 (30.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Now run `sudo dhclient usb1` and ping google.com to check the board has access to internet.

```
debian@beaglebone:~$ sudo dhclient usb1
debian@beaglebone:~$ ping www.google.com
PING www.google.com (142.250.67.36) 56(84) bytes of data.
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=1 ttl=116 time=
50.9 ms
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=2 ttl=116 time=
12.5 ms
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=3 ttl=116 time=
11.4 ms
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=4 ttl=116 time=
10.8 ms
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=5 ttl=116 time=
23.1 ms
64 bytes from maa05s12-in-f4.1e100.net (142.250.67.36): icmp_seq=6 ttl=116 time=
11.8 ms
```

Installing Modules

Our python code uses a few libraries that do not come preinstalled, so we will need to install those through the terminal. But in order to be able to install any python libraries we will need to install pip first. Now that the SSH connection is established, run the following commands to install pip.

```
curl https://bootstrap.pypa.io/get-pip.p -o get-pip.py
```

```
python get-pip.py
```

To install Tweepy (Twitter's API) and requests run the following commands

```
pip install tweepy
```

```
pip install requests
```

Creating a Developer Twitter Account

Using the Twitter API requires a developer Twitter account and a Twitter application, the following steps explain how to create them.

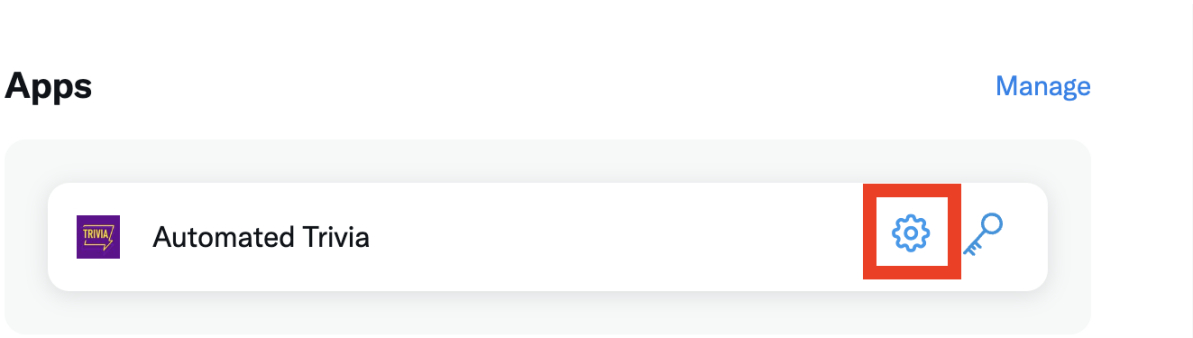
Sign in to your Twitter account and sign up for a free developer account on

https://developer.twitter.com/en/portal/petition/essential/basic-info

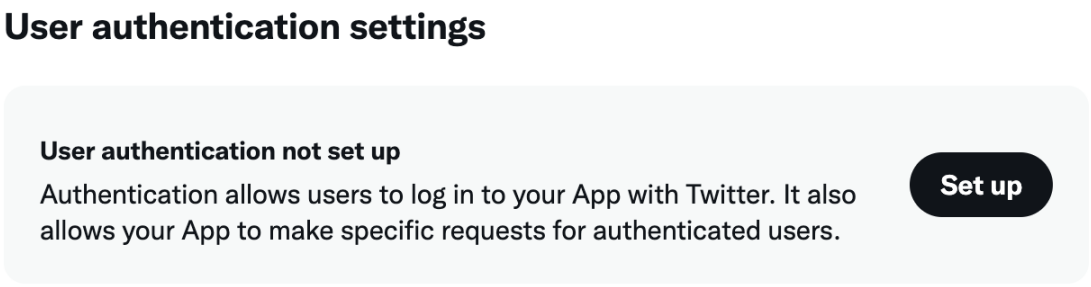
Then go to the Twitter developer portal

https://developer.twitter.com/en/portal/dashboard

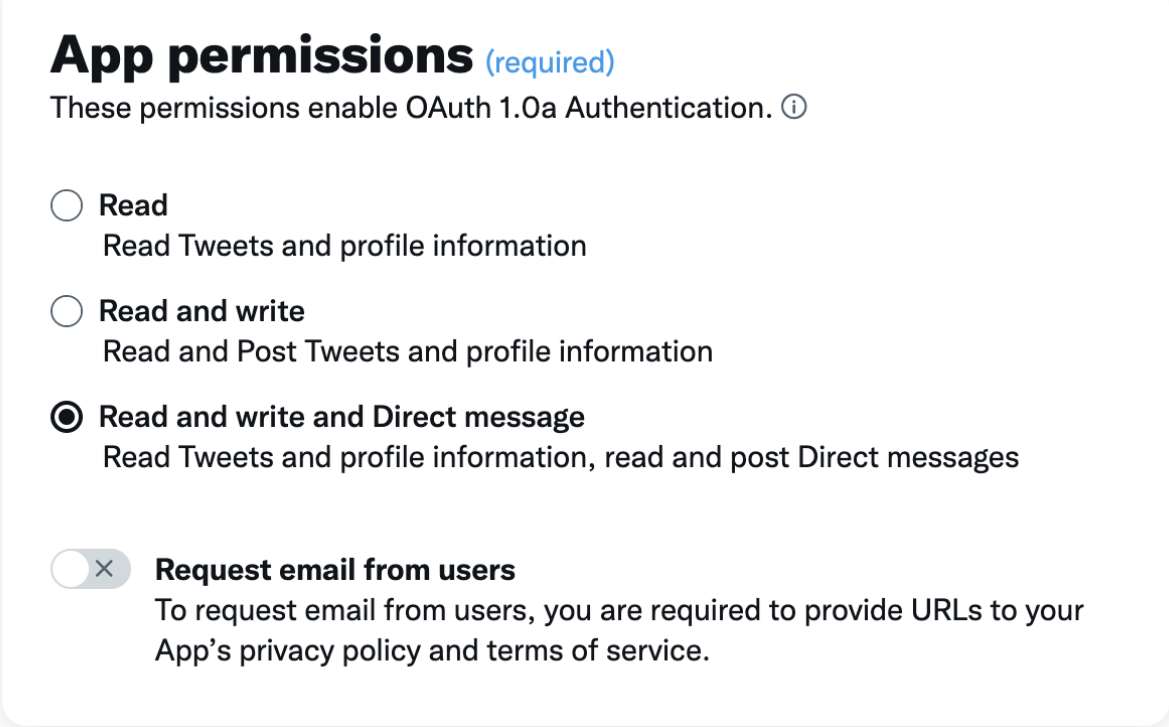
There must be a default project with a default application inside, in your dashboard, you can change the names to your preferred names. After giving the application a name, open up its settings



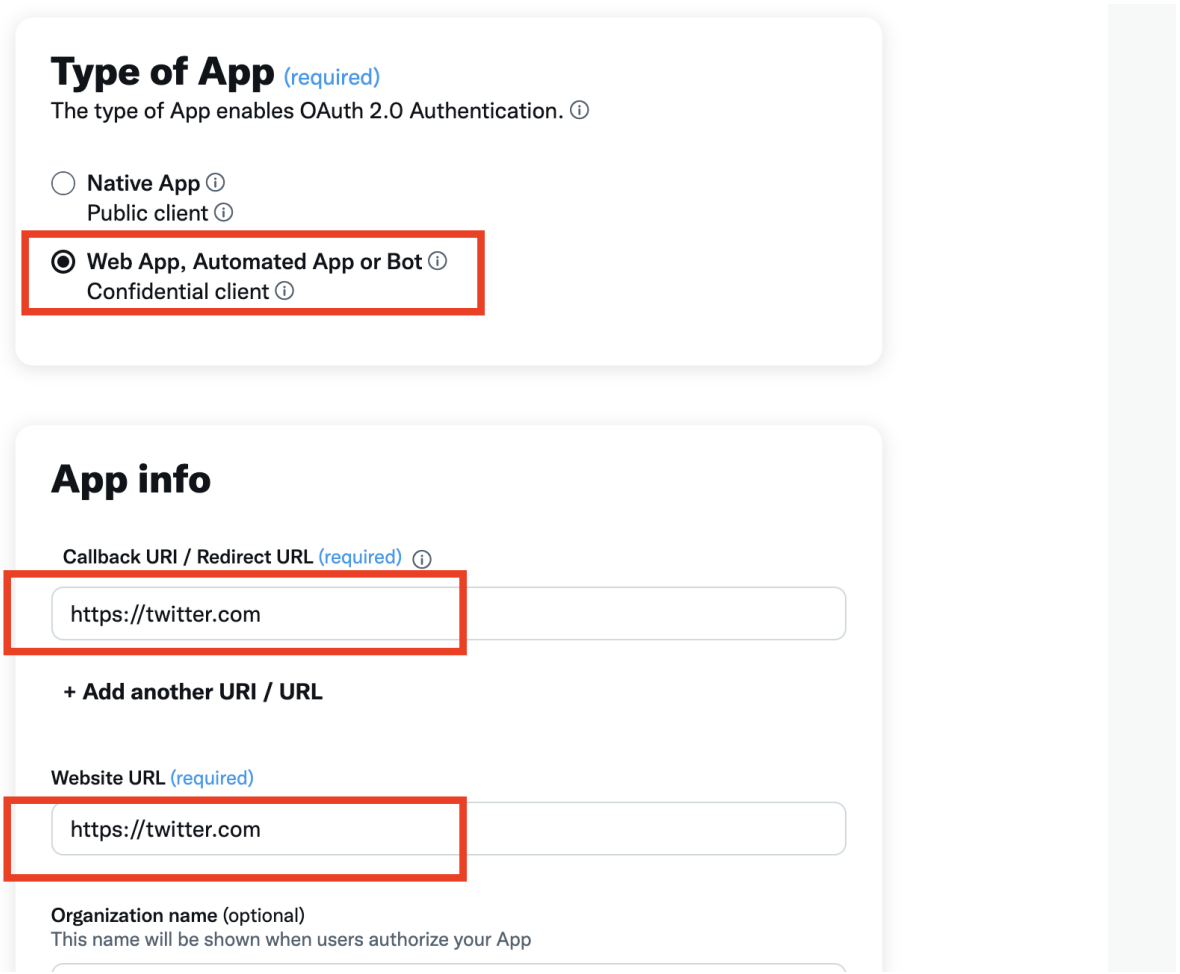
Under User Authentication Settings click on “Set up”



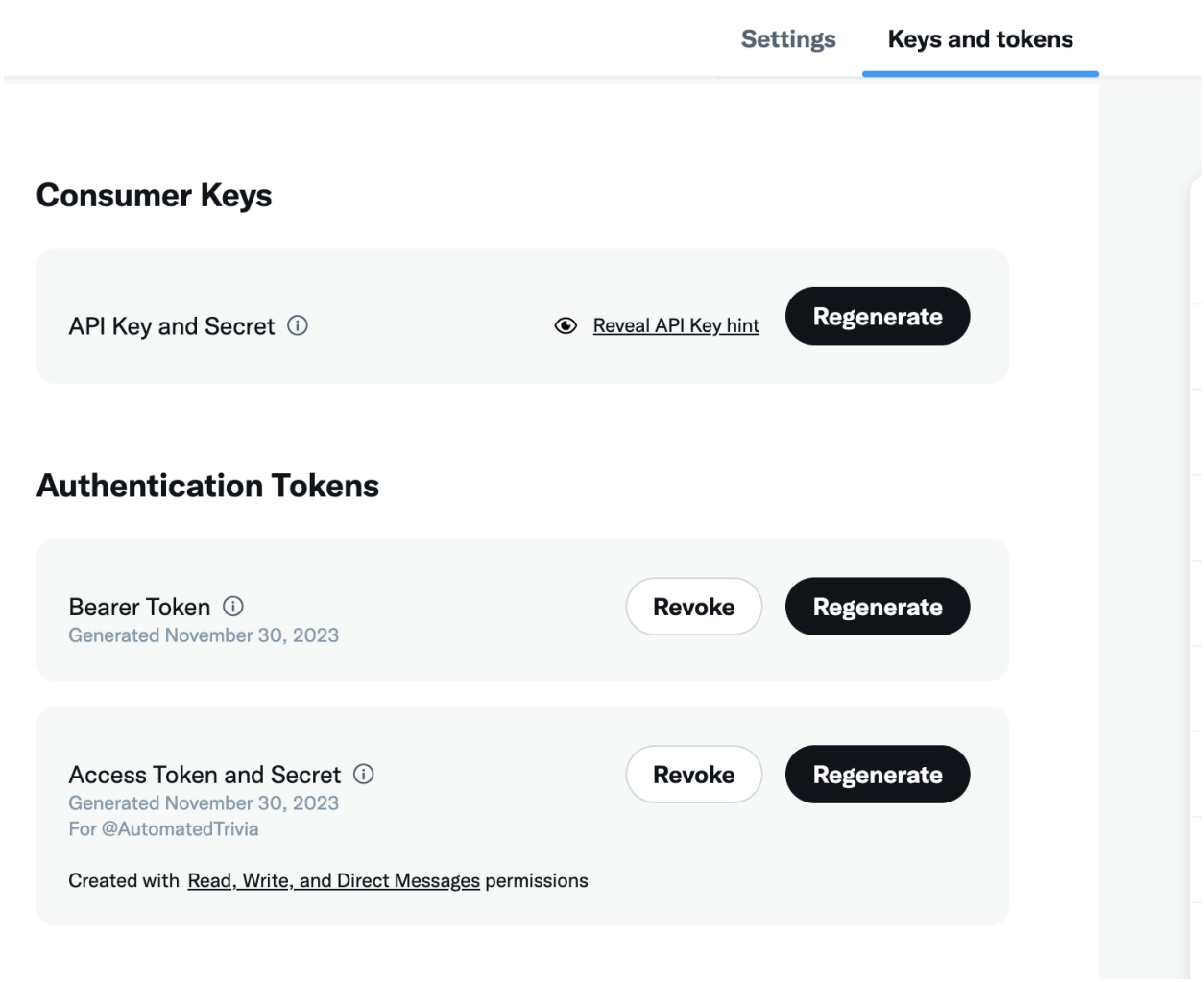
Make sure to give your application read and write permissions



Fill in the other fields like so :



For the final step, go to the Keys and Tokens tab of your application and generate your API Secret and Key, Bearer Token, Access Token and Secret, Client ID and Secret, make sure to store them all somewhere secure.



Python Code

Refer to the Appendices for the full code. Here Sections of the code are explained :

Importing Required Libraries

```
import tweepy
import random
```

```
import html
import requests
import json
import time
```

Obtaining The Questions

This function fetches trivia questions from the Open Trivia Database API based on the specified amount, category, and difficulty level.

It returns a list of question objects.

```
def get_questions(amount: int, category: int) -> list:
    url =
f"https://opentdb.com/api.php?amount={amount}&category={category}&difficulty=easy&type
=multiple"
    response = requests.get(url)
    response_json = response.json()
    return response_json["results"]
```

Shuffling Choices

This function shuffles the answer choices for a given question.

It truncates or pads each choice to ensure it's not longer than max_length.

It returns a shuffled list of choices.

```
def shuffle_choices(choices: list, max_length: int = 25) -> list:
    # Truncate or pad each choice to ensure it's not longer than max_length
    truncated_choices = [choice[:max_length] for choice in choices]
    padded_choices = [choice.ljust(max_length) for choice in truncated_choices]

    random.shuffle(padded_choices)
    return padded_choices
```

Start Trivia

This function plays the trivia game and tweets questions with shuffled answer choices.

It uses the get_questions function to get a list of trivia questions.

For each question, it extracts the question text and answer choices, shuffles the choices, and then tweets the question as a poll.

If an error occurs during the process, it prints an error message and waits for 5 minutes before retrying.

```
def play_trivia_and_tweet(amount: int, category: int) -> None:
    while True:
        try:
            questions = get_questions(amount, category)
            for question in questions:
                question_text = html.unescape(question["question"])
                choices = question["incorrect_answers"]
                choices.extend([question["correct_answer"]])
                shuffled_choices = shuffle_choices(choices)
                client.create_tweet(
                    text=question_text,
                    poll_duration_minutes=100,
                    poll_options=shuffled_choices
                )
                time.sleep(300)  # Sleep for 5 minutes before the next tweet
        except Exception as e:
            print(f"An error occurred: {str(e)}")
            time.sleep(300)  # Sleep for 5 minutes before retrying
```

Twitter API Verification

Twitter API credentials are provided (replace "..." with actual credentials).

tweepy.Client is used to initialize a Twitter client for creating tweets.

tweepy.OAuth1UserHandler is used to authenticate with the Twitter API.

```
api_key = ""
api_secret = ""
bearer_token = ""
access_token = ""
access_token_secret = ""

client = tweepy.Client(bearer_token, api_key, api_secret, access_token,
access_token_secret)
auth = tweepy.OAuth1UserHandler(api_key, api_secret, access_token,
access_token_secret)
api = tweepy.API(auth)
```

Main Section

The script is executed only if it is the main module.

The default values for amount (number of questions) and category (Trivia category) are set.

The play_trivia_and_tweet function is called in an infinite loop to keep playing and tweeting trivia questions.

```
if __name__ == "__main__":
    amount = 50
    category = 11
    while True:
        play_trivia_and_tweet(amount, category)
```

Challenges And Solutions

Limited Internet Access

Challenge: The BeagleBone AI faced constraints in accessing the university's public internet directly.

Solution: A mobile hotspot was used to provide internet access, enabling the BeagleBone AI to connect to external services like the Twitter API.

Hardware Issues

Challenge: Some BeagleBone devices in the lab were non-functional, requiring troubleshooting. The first board would not connect to the laptop while the second board would not power on.

Solution: replacing the board with another board helped us bypass this issue.

Dependency on Mobile Hotspot:

Challenge: Using a mobile hotspot introduced potential connectivity issues, where the hotspot would cut in and out if the phone was ever turned off and created a dependency on mobile devices.

Solution: The phone setting was changed so that the phone would not turn off even through prolonged inactivity. Despite limitations, leveraging a mobile hotspot provided a practical workaround.

Conclusion

In conclusion, this project successfully harnesses the Internet of Things (IoT) capabilities of the BeagleBone AI board to create a dynamic and engaging Twitter trivia bot. By integrating the Open Trivia Database and leveraging the Cloud 9 IDE hosted on the BeagleBone AI, the system autonomously delivers trivia questions to Twitter followers, fostering interactive engagement through polls. The project exemplifies the adaptability of IoT technology in enhancing social media interactions, offering a unique and entertaining experience for users within Twitter.

The interactive nature of the Twitter Trivia Bot not only aligns with the contemporary emphasis on social media engagement but also provides a valuable tool for fostering a sense of community and entertainment. Users can test their knowledge, share their responses, and participate in a shared experience, contributing to a more vibrant and interactive online environment.

Future Enhancements

While the current implementation demonstrates the core functionality of the Twitter Trivia Bot, there are possibilities for future enhancements to further enhance the user experience:

- Enhanced Question Variety: Expand the trivia question database to include a broader range of topics, ensuring a diverse and appealing set of questions for a wider audience.

- User Interaction Features: Implement features allowing users to suggest trivia questions, share their scores, and interact with other participants, fostering a sense of community around the trivia bot.
- Personalized Engagement: Introduce algorithms that personalize trivia content based on user preferences and historical interactions, tailoring the trivia experience to individual interests.
- Real-time Analytics: Incorporate real-time analytics to track user engagement, analyze response patterns, and derive insights for refining and optimizing the trivia bot's performance.
- Integration with Other Platforms: Extend the functionality to other social media platforms, broadening the reach and impact of the trivia bot beyond Twitter.

By pursuing these future enhancements, the Twitter Trivia Bot can evolve into a more sophisticated and versatile platform, staying at the forefront of social media engagement trends and delivering a consistently enjoyable experience for its users.

Appendices

Video Link: https://www.youtube.com/watch?v=3-dKYTAXRR0

```
import tweepy
import random
import html
import requests
import json
import time


# Get Trivia Questions
def get_questions(amount: int, category: int) -> list:
    url =
f"https://opentdb.com/api.php?amount={amount}&category={category}&difficulty=easy&type
=multiple"
    response = requests.get(url)
    response_json = response.json()
    return response_json["results"]

# Shuffle Answers
def shuffle_choices(choices: list, max_length: int = 25) -> list:
    # Truncate or pad each choice to ensure it's not longer than max_length
    truncated_choices = [choice[:max_length] for choice in choices]
    padded_choices = [choice.ljust(max_length) for choice in truncated_choices]

    random.shuffle(padded_choices)
    return padded_choices


# Start Trivia
def play_trivia_and_tweet(amount: int, category: int) -> None:
    while True:
        try:
            questions = get_questions(amount, category)
```

```
            for question in questions:
                question_text = html.unescape(question["question"])
                choices = question["incorrect_answers"]
                choices.extend([question["correct_answer"]])
                shuffled_choices = shuffle_choices(choices)
                client.create_tweet(
                    text=question_text,
                    poll_duration_minutes=100,
                    poll_options=shuffled_choices
                )
                time.sleep(300)  # Sleep for 5 minutes before the next tweet
        except Exception as e:
            print(f"An error occurred: {str(e)}")
            time.sleep(300)  # Sleep for 5 minutes before retrying


api_key = ""
api_secret = ""
bearer_token = ""
access_token = ""
access_token_secret = ""

client = tweepy.Client(bearer_token, api_key, api_secret, access_token,
access_token_secret)
auth = tweepy.OAuth1UserHandler(api_key, api_secret, access_token,
access_token_secret)
api = tweepy.API(auth)


if __name__ == "__main__":
    amount = 50
    category = 11
    while True:
        play_trivia_and_tweet(amount, category)
```