VILNIUS TECH

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

ELEKTRONIKOS FAKULTETAS

ELEKTRONINIŲ SISTEMŲ KATEDRA

EXPO IR MQTT INTEGRACIJA

Išmanieji IP įrenginiai

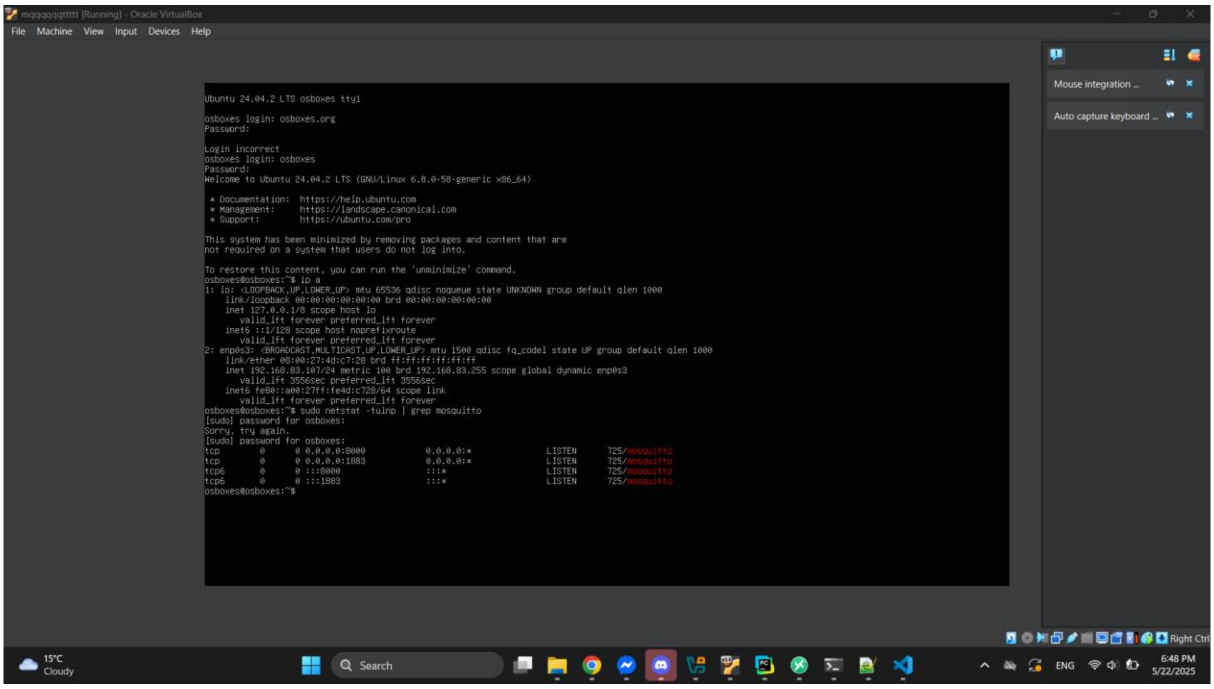
3 laboratorinis darbas

Atliko: DKRf-22 gr. studentas Eimantas Dima

Tikrino: Dr. Tomas Cuzanauskas

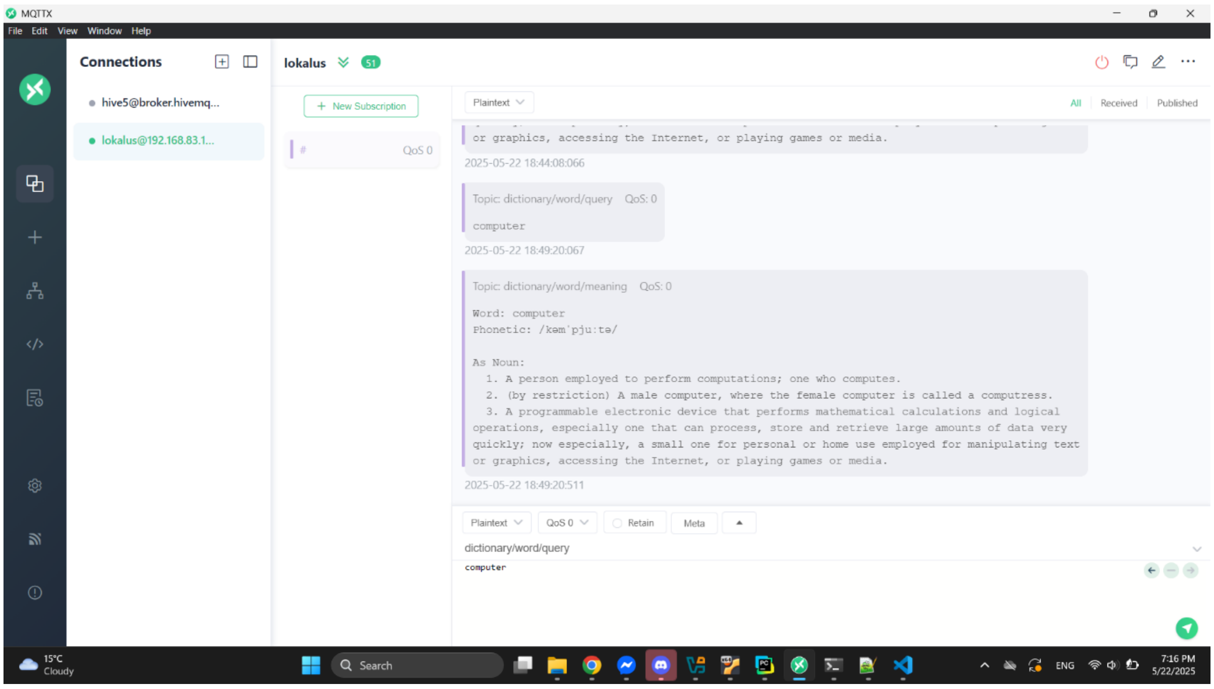
VILNIUS 2025

1. Paruošėme ubuntu virtualią mašiną. Paleidome ją bridged režimu, kad galėtumėme pasiekti tinklą. Bei sukonfiguravome mosquitto 1883 ir 8000 prievadams.

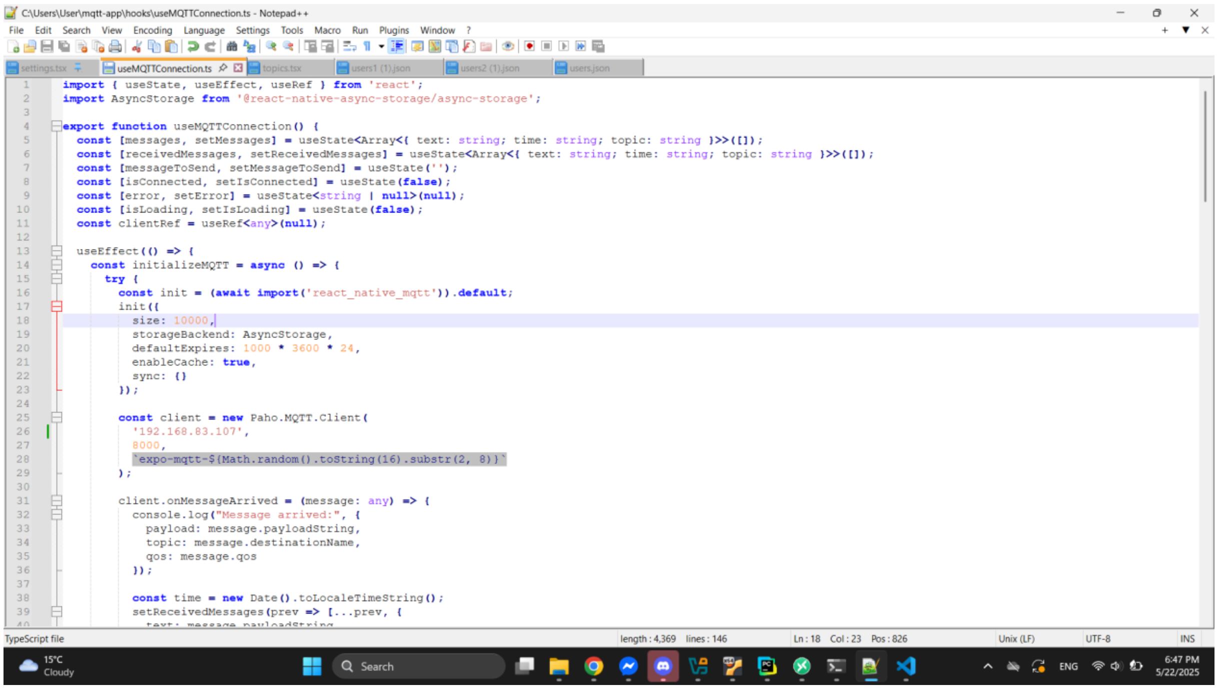


```
Ubuntu 24.04.2 LTS osboxes tty1

osboxes login: osboxes.org
Password:

Login incorrect
osboxes login: osboxes
Password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
osboxes@osboxes:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:4d:c7:28 brd ff:ff:ff:ff:ff:ff
    inet 192.168.83.107/24 metric 100 brd 192.168.83.255 scope global dynamic enp0s3
       valid_lft 3556sec preferred_lft 3556sec
    inet6 fe80::a00:27ff:fe4d:c728/64 scope link
       valid_lft forever preferred_lft forever
osboxes@osboxes:~$ sudo netstat -tulnp | grep mosquitto
[sudo] password for osboxes:
Sorry, try again.
[sudo] password for osboxes:
tcp        0      0 0.0.0.0:8000            0.0.0.0:*               LISTEN      725/mosquitto
tcp        0      0 0.0.0.0:1883            0.0.0.0:*               LISTEN      725/mosquitto
tcp6       0      0 :::8000                 :::*                    LISTEN      725/mosquitto
tcp6       0      0 :::1883                 :::*                    LISTEN      725/mosquitto
osboxes@osboxes:~$
```

2. Paleidome expo programą, bei sukonfigūravome programos kode, kad MQTT užklausos kreiptūsi į virtualios mašinos IP adresą. Taip pat pakeičiau MQTT temas į dictionary/word/query užklausoms, ir dictionary/word/meaning užklausų grąžinimui.

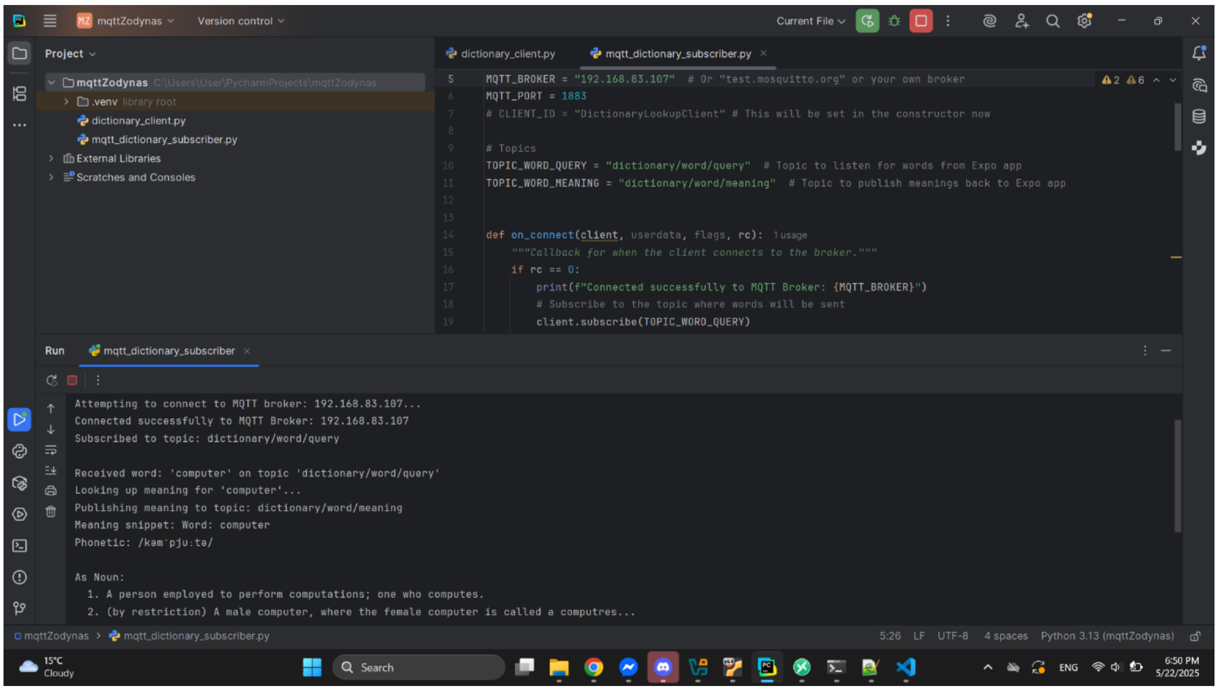


```
or graphics, accessing the Internet, or playing games or media.
2025-05-22 18:44:08:066

Topic: dictionary/word/query   QoS: 0
computer
2025-05-22 18:49:20:067

Topic: dictionary/word/meaning   QoS: 0
Word: computer
Phonetic: /kəmˈpjuːtə/

As Noun:
  1. A person employed to perform computations; one who computes.
  2. (by restriction) A male computer, where the female computer is called a computress.
  3. A programmable electronic device that performs mathematical calculations and logical
operations, especially one that can process, store and retrieve large amounts of data very
quickly; now especially, a small one for personal or home use employed for manipulating text
or graphics, accessing the Internet, or playing games or media.
2025-05-22 18:49:20:511

Plaintext   QoS 0   Retain   Meta
dictionary/word/query
computer
```



```typescript
import { useState, useEffect, useRef } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';

export function useMQTTConnection() {
  const [messages, setMessages] = useState<Array<{ text: string; time: string; topic: string }>>([]);
  const [receivedMessages, setReceivedMessages] = useState<Array<{ text: string; time: string; topic: string }>>([]);
  const [messageToSend, setMessageToSend] = useState('');
  const [isConnected, setIsConnected] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState(false);
  const clientRef = useRef<any>(null);

  useEffect(() => {
    const initializeMQTT = async () => {
      try {
        const init = (await import('react_native_mqtt')).default;
        init({
          size: 10000,
          storageBackend: AsyncStorage,
          defaultExpires: 1000 * 3600 * 24,
          enableCache: true,
          sync: {}
        });

        const client = new Paho.MQTT.Client(
          '192.168.83.107',
          8000,
          `expo-mqtt-${Math.random().toString(16).substr(2, 8)}`
        );

        client.onMessageArrived = (message: any) => {
          console.log("Message arrived:", {
            payload: message.payloadString,
            topic: message.destinationName,
            qos: message.qos
          });

          const time = new Date().toLocaleTimeString();
          setReceivedMessages(prev => [...prev, {
```

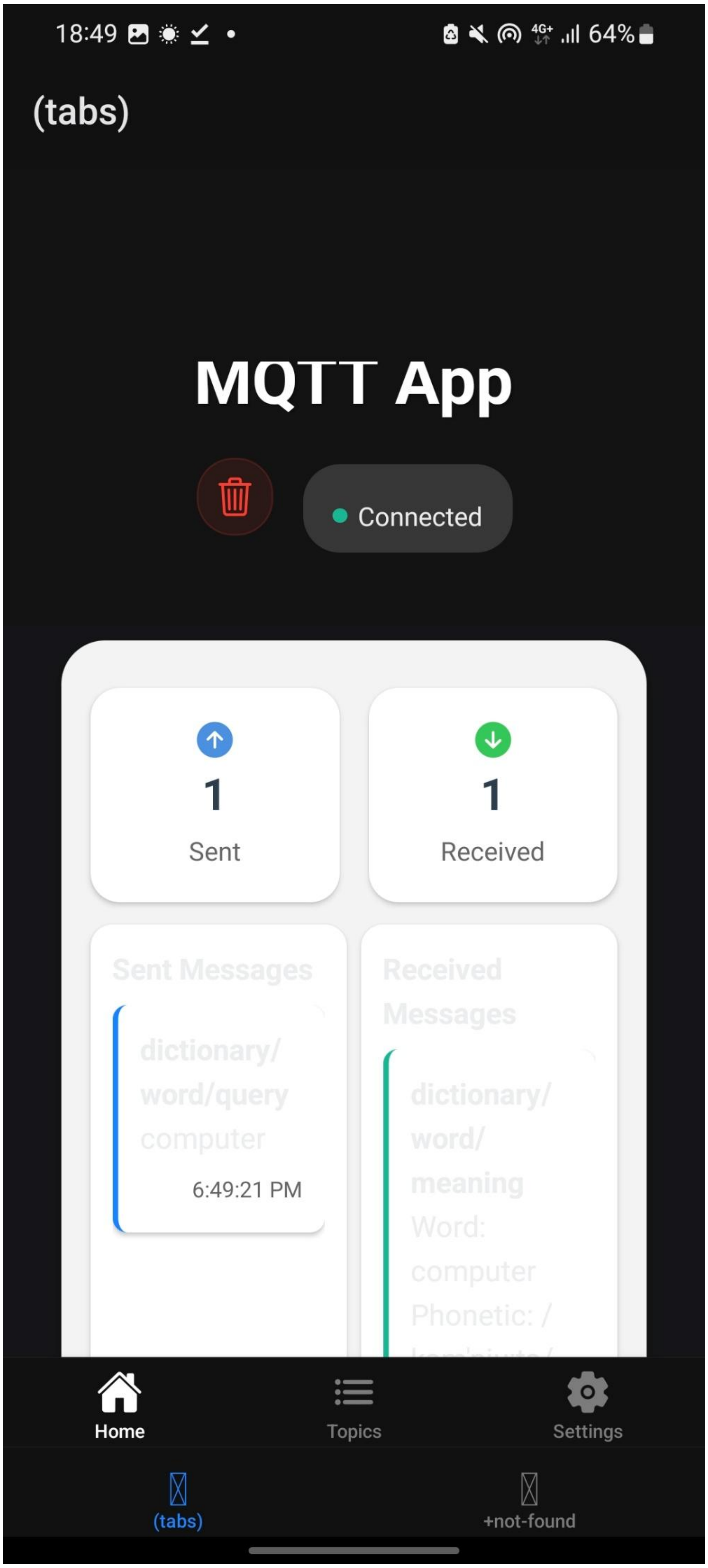
3. Parašėme python backend kodą, kuris pateikia API užklausas į http://api.dictionaryapi.dev/ ir gauna žodyno atsakymus iš https://en.wiktionary.org/wiki/[žodis]



```python
MQTT_BROKER = "192.168.83.107"  # Or "test.mosquitto.org" or your own broker
MQTT_PORT = 1883
# CLIENT_ID = "DictionaryLookupClient" # This will be set in the constructor now

# Topics
TOPIC_WORD_QUERY = "dictionary/word/query"  # Topic to listen for words from Expo app
TOPIC_WORD_MEANING = "dictionary/word/meaning"  # Topic to publish meanings back to Expo app


def on_connect(client, userdata, flags, rc):
    """Callback for when the client connects to the broker."""
    if rc == 0:
        print(f"Connected successfully to MQTT Broker: {MQTT_BROKER}")
        # Subscribe to the topic where words will be sent
        client.subscribe(TOPIC_WORD_QUERY)
```

```
Attempting to connect to MQTT broker: 192.168.83.107...
Connected successfully to MQTT Broker: 192.168.83.107
Subscribed to topic: dictionary/word/query

Received word: 'computer' on topic 'dictionary/word/query'
Looking up meaning for 'computer'...
Publishing meaning to topic: dictionary/word/meaning
Meaning snippet: Word: computer
Phonetic: /kəmˈpjuːtə/

As Noun:
  1. A person employed to perform computations; one who computes.
  2. (by restriction) A male computer, where the female computer is called a computres...
```

4. Užklausas galime pateikti per expo telefone.



18:49 64%

(tabs)

MQTT App

Connected

1
Sent

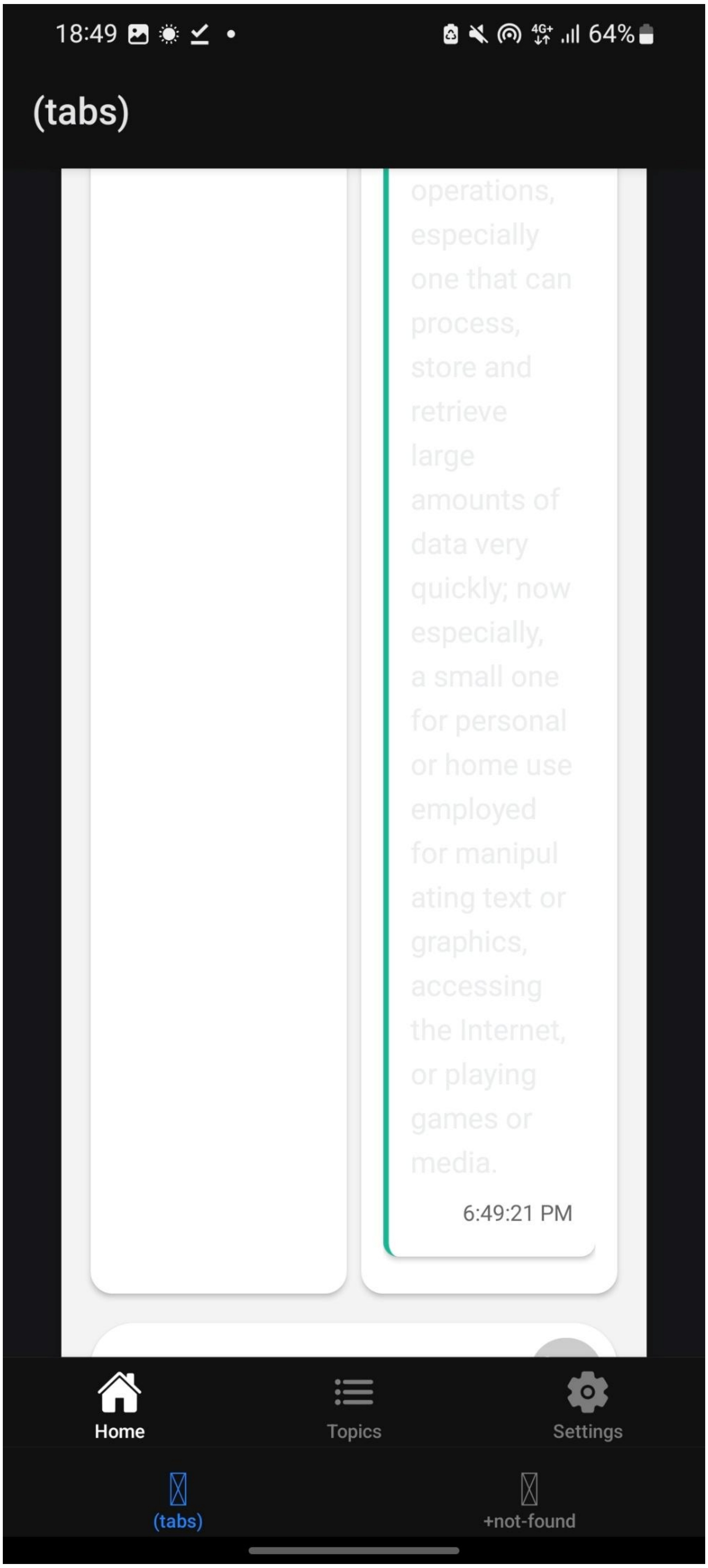
1
Received

Sent Messages

dictionary/word/query
computer
6:49:21 PM

Received Messages

dictionary/word/meaning
Word: computer
Phonetic: /



18:49 64%

(tabs)

operations,
especially
one that can
process,
store and
retrieve
large
amounts of
data very
quickly; now
especially,
a small one
for personal
or home use
employed
for manipul
ating text or
graphics,
accessing
the Internet,
or playing
games or
media.

6:49:21 PM

Prisegu programos kodą:

Dictionary_client.py:

```
import requests
import json

# The API endpoint for dictionaryapi.dev
API_BASE_URL = "https://api.dictionaryapi.dev/api/v2/entries/en/"

def get_word_meaning(word):
    """
    Fetches word meanings from dictionaryapi.dev.
    """
    if not word:
        return "Error: No word provided."

    url = f"{API_BASE_URL}{word}"
    try:
        response = requests.get(url)
        response.raise_for_status()  # Raises an HTTPError for bad
responses (4XX or 5XX)

        data = response.json()
        return format_api_response(data, word)

    except requests.exceptions.HTTPError as http_err:
        if response.status_code == 404:
            return f"Sorry, couldn't find a definition for '{word}'."
        return f"HTTP error occurred: {http_err} - {response.text}"
    except requests.exceptions.RequestException as req_err:
        return f"Error fetching definition: {req_err}"
    except json.JSONDecodeError:
        return "Error: Could not decode the server's response."
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def format_api_response(api_data, original_word):
    """
    Formats the JSON response from dictionaryapi.dev into a readable
string.
    """
    if not api_data or not isinstance(api_data, list):
        # The API might return an object with 'title', 'message',
'resolution' on 404
        if isinstance(api_data, dict) and "title" in api_data:
            return f"{api_data.get('title', 'Error')}:
{api_data.get('message', 'Could not find the word.')}"
        return f"No definitions found for '{original_word}' or unexpected
API response format."

    formatted_output = []

    # The API returns a list, usually with one main entry for the word
    for entry_index, entry in enumerate(api_data):
        word = entry.get("word", original_word)
        phonetic = entry.get("phonetic", "")

        if entry_index == 0: # Main header for the first entry
            formatted_output.append(f"Word: {word}")
            if phonetic:
                formatted_output.append(f"Phonetic: {phonetic}")
        elif len(api_data) > 1: # If multiple entries (e.g. different
etymologies)
```

```
            formatted_output.append(f"\n--- Alternative Entry for {word} --
-")
            if phonetic:
                formatted_output.append(f"Phonetic: {phonetic}")

        if "meanings" in entry:
            for meaning in entry["meanings"]:
                part_of_speech = meaning.get("partOfSpeech", "N/A")
                formatted_output.append(f"\nAs
{part_of_speech.capitalize()}:")

                for i, definition_obj in
enumerate(meaning.get("definitions", [])):
                    definition = definition_obj.get("definition", "No
definition text.")
                    example = definition_obj.get("example", "")

                    formatted_output.append(f"  {i+1}. {definition}")
                    if example:
                        formatted_output.append(f"     Example:
\"{example}\"")
        else:
            formatted_output.append("  No specific meanings found in this
entry.")

    if not formatted_output or (len(formatted_output) == 2 and "Phonetic"
in formatted_output[1] and not "meanings" in api_data[0]):
        return f"No detailed definitions found for '{original_word}'."

    return "\n".join(formatted_output)

# --- Example Usage (for testing this file directly) ---
if __name__ == "__main__":
    test_word = "hello"
    print(f"--- Looking up: {test_word} ---")
    meaning = get_word_meaning(test_word)
    print(meaning)

    print("\n--- Looking up: programming ---")
    meaning_programming = get_word_meaning("programming")
    print(meaning_programming)

    print("\n--- Looking up: nonexistingwordxyz123 ---")
    meaning_nonexistent = get_word_meaning("nonexistingwordxyz123")
    print(meaning_nonexistent)
```

mqtt_dictionary_subscriber.py:

```
import paho.mqtt.client as mqtt
from dictionary_client import get_word_meaning  # Import the function from
our other file

# MQTT Configuration
MQTT_BROKER = "192.168.83.107"  # Or "test.mosquitto.org" or your own
broker
MQTT_PORT = 1883
# CLIENT_ID = "DictionaryLookupClient" # This will be set in the
constructor now

# Topics
TOPIC_WORD_QUERY = "dictionary/word/query"  # Topic to listen for words
from Expo app
TOPIC_WORD_MEANING = "dictionary/word/meaning"  # Topic to publish meanings
back to Expo app
```

```
def on_connect(client, userdata, flags, rc):
    """Callback for when the client connects to the broker."""
    if rc == 0:
        print(f"Connected successfully to MQTT Broker: {MQTT_BROKER}")
        # Subscribe to the topic where words will be sent
        client.subscribe(TOPIC_WORD_QUERY)
        print(f"Subscribed to topic: {TOPIC_WORD_QUERY}")
    else:
        print(f"Failed to connect, return code {rc}\n")


def on_message(client, userdata, message):
    """Callback for when a message is received from the broker."""
    try:
        word_to_search = message.payload.decode("utf-8")
        print(f"\nReceived word: '{word_to_search}' on topic
'{message.topic}'")

        if word_to_search:
            print(f"Looking up meaning for '{word_to_search}'...")
            meaning = get_word_meaning(word_to_search)

            print(f"Publishing meaning to topic: {TOPIC_WORD_MEANING}")
            # Log first 200 chars of meaning if it's long
            log_meaning = meaning if len(meaning) < 200 else meaning[:200]
+ "..."
            print(f"Meaning snippet: {log_meaning}")

            client.publish(TOPIC_WORD_MEANING, str(meaning))
        else:
            print("Received an empty message. No word to search.")
            client.publish(TOPIC_WORD_MEANING, "Error: Received an empty
word to search.")

    except Exception as e:
        print(f"Error processing message or fetching meaning: {e}")
        client.publish(TOPIC_WORD_MEANING, f"Error processing request:
{e}")


# --- Main script execution ---
if __name__ == "__main__":
    # MODIFICATION HERE: Specify callback_api_version and client_id
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1,
client_id="DictionaryLookupClient")

    client.on_connect = on_connect
    client.on_message = on_message

    print(f"Attempting to connect to MQTT broker: {MQTT_BROKER}...")
    try:
        client.connect(MQTT_BROKER, MQTT_PORT, 60)
    except Exception as e:
        print(f"Could not connect to MQTT broker: {e}")
        exit()

    # Start the loop to process network traffic, dispatch callbacks, and
handle reconnecting.
    try:
        client.loop_forever()
    except KeyboardInterrupt:
        print("\nDisconnecting from MQTT broker...")
        client.disconnect()
        print("Disconnected.")
```