Experiment No 7

AIM: To develop a simple IoT-based device control system using **Flask** as a web framework and **ngrok** for exposing the local server to the internet, allowing remote control of an LED-like device.

Objectives
- To implement a web-based interface for controlling an IoT device (LED).
- To use **Flask** to create a lightweight API for device control.
- To expose the local Flask server to the internet using **ngrok**.
- To enable remote control of the device via API calls.

Tools Used
- **Python** (Version 3.x)
- **Flask** (for creating the web server)
- **ngrok** (for exposing Flask to the internet)
- **pip** (Python package manager)
- **Web Browser** (for testing API)
- **Postman** (optional, for API testing)
- **Command Prompt/Terminal** (for running the server)

Theory
Introduction to IoT and Web-based Control
The Internet of Things (IoT) enables physical devices, such as sensors and actuators, to be controlled remotely via the internet. In this experiment, we simulate an IoT device (an LED) that can be turned **ON** or **OFF** using a **Flask web server** and control it remotely through **ngrok**.
The main components of this system include:
- **Flask (Python-based Web Framework)**: Handles HTTP requests and processes commands.
- **ngrok (Tunneling Tool)**: Exposes the locally running Flask application to the public internet, allowing remote control.
- **REST API (GET Requests)**: Allows users to send commands to control the LED state.

Why Use Flask?
Flask is a **lightweight and easy-to-use** web framework that allows:
- Handling **HTTP requests** efficiently.
- Creating **RESTful APIs** to interact with IoT devices.
- Providing **JSON-based responses** that can be used in mobile apps or web dashboards.

Understanding Flask API Endpoints
APIs allow **external systems** (e.g., web apps, mobile apps, or other computers) to interact with our IoT device.
In this experiment, we define three **Flask API endpoints**:

Role of ngrok
Flask runs on a **local server** (127.0.0.1:5000), which is only accessible on the local machine. To control the LED **from outside the local network**, we need **ngrok**.
ngrok provides:

Endpoint	Method	Description
/	GET	Displays instructions on how to use the API.
/control?device=LED &state=ON	GET	Turns the LED ON.
/control?device=LED &state=OFF	GET	Turns the LED OFF.
/status	GET	Returns the current state of the LED.

A Public URL for Flask Server
- Converts http://127.0.0.1:5000 → http://your-ngrok-url.ngrok.io
- **Secure Tunneling**
- No need to configure **firewalls or routers**.
- **Remote Access**
- Users can send API requests from **any device**, anywhere in the world.

How Flask and ngrok Work Together
- Flask starts a local web server on 127.0.0.1:5000.
- ngrok creates a public URL, forwarding all requests to the Flask server.
- Users send HTTP requests (via a web browser or API tools like Postman) to control the LED.
- Flask processes the requests and updates the LED state.
- Flask sends a response confirming the action.

Program Code cum Procedure
Step 1: Install Required Packages
Before running the program, install dependencies:
```
pip install flask pyngrok
```

Step 2: Save the Flask Code as app.py
Create a new file named app.py and **copy-paste the following code**:
```
from flask import Flask, request, jsonify
app = Flask(__name__)
device_state = {"LED": "OFF"}  # Initial state of the virtual IoT device
@app.route('/')
def home():
    return "<h1>IoT Device Control</h1><p>Use /control?device=LED&state=ON to control the device.</p>"
@app.route('/control', methods=['GET'])
def control_device():
    device = request.args.get('device')
    state = request.args.get('state')
    if device in device_state and state in ["ON", "OFF"]:
        device_state[device] = state
        return jsonify({"message": f"{device} turned {state}"})
    else:
        return jsonify({"error": "Invalid device or state"})
@app.route('/status', methods=['GET'])
def device_status():
    return jsonify(device_state)
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 3: Run Flask Server

Open Command Prompt (Windows) or Terminal (Linux/Mac) and navigate to the directory where app.py is saved:

```
python app.py
```

Output:

```
* Running on http://127.0.0.1:5000/
```

Step 4: Start ngrok

In a new terminal window, run:

```
ngrok http 5000
```

It will generate a **public URL**, e.g.:

http://your-ngrok-url.ngrok.io -> http://127.0.0.1:5000

Step 5: Test the API

Use your browser or Postman to access the following URLs:

Home Page

http://your-ngrok-url.ngrok.io/

Expected Output:

```
<h1>IoT Device Control</h1><p>Use /control?device=LED&state=ON to control the device.</p>
```

Turn LED ON

http://your-ngrok-url.ngrok.io/control?device=LED&state=ON

Expected JSON Response:

```
{"message": "LED turned ON"}
```

Turn LED OFF

http://your-ngrok-url.ngrok.io/control?device=LED&state=OFF

Expected JSON Response:

```
{"message": "LED turned OFF"}
```

Check LED Status

http://your-ngrok-url.ngrok.io/status

Expected JSON Response:

```
{"LED": "ON"}  # or "OFF" depending on last action
```

Explanation of the Program

Import Required Modules

```
from flask import Flask, request, jsonify
```

- Flask: Used to create a web server.
- request: Extracts parameters from API requests.
- jsonify: Formats responses in **JSON** format.

Initialize Flask App

```
app = Flask(__name__)
```

- Flask(__name__) initializes a **Flask web application**.

Define the LED State

```
device_state = {"LED": "OFF"}  # Initial state of the virtual IoT device
```

- A **dictionary** (device_state) stores the LED's status.
- Initially, the **LED is OFF**.

Define the Home Route (/)

```
@app.route('/')
def home():
    return "<h1>IoT Device Control</h1><p>Use /control?device=LED&state=ON to control the
device.</p>"
```

- Displays instructions for using the API when accessed.

Define the Control Route (/control)

```
@app.route('/control', methods=['GET'])
def control_device():
    device = request.args.get('device')  # Get 'device' parameter from URL
    state = request.args.get('state')    # Get 'state' parameter from URL
    if device in device_state and state in ["ON", "OFF"]:
        device_state[device] = state  # Update the LED state
        return jsonify({"message": f"{device} turned {state}"})
    else:
        return jsonify({"error": "Invalid device or state"})
```

How It Works:

- Extracts **device name** (LED) and **desired state** (ON or OFF) from the **query parameters**.
- If valid, it **updates the device state** and returns a confirmation message.
- If invalid, it returns an **error message**.

Example Usage:

```
http://127.0.0.1:5000/control?device=LED&state=ON
Response:
{"message": "LED turned ON"}
http://127.0.0.1:5000/control?device=LED&state=OFF
Response:
{"message": "LED turned OFF"}
```

Define the Status Route (/status)

```
@app.route('/status', methods=['GET'])
def device_status():
    return jsonify(device_state)  # Returns the current state of the LED
```

- Returns the current state of the LED in JSON format.
- Example Usage:

```
http://127.0.0.1:5000/status
{"LED": "ON"}
```

Start the Flask Server

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Runs Flask on port 5000.
- host='0.0.0.0' allows access from any network.

Exposing Flask via ngrok

After starting Flask, we need to **expose it to the internet**:

Run this command in a new terminal window:

```
ngrok http 5000
```

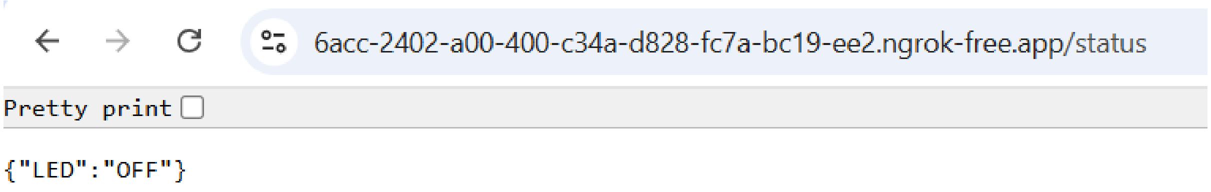
Expected output:

```
Forwarding http://your-ngrok-url.ngrok.io -> http://127.0.0.1:5000
```

Now, use the **ngrok URL** to access the API from **anywhere**.

Outputs:

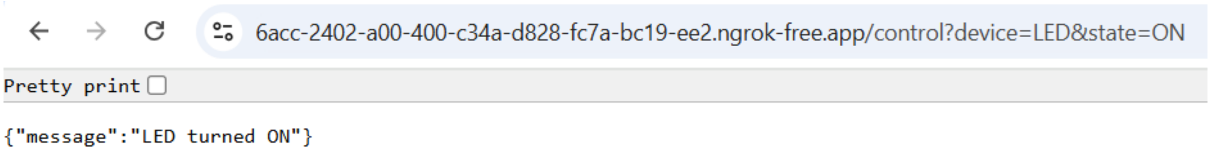
https://6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status


6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

Pretty print

```
{"LED":"OFF"}
```

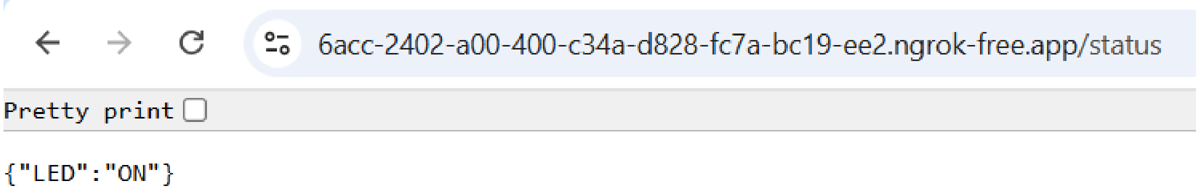
https://6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED&state=ON


6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED&state=ON

Pretty print

```
{"message":"LED turned ON"}
```

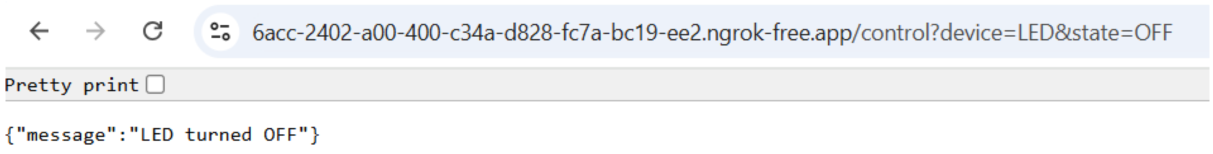
https://6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status


6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

Pretty print

```
{"LED":"ON"}
```

https://6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED&state=OFF


6acc-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED&state=OFF

Pretty print

```
{"message":"LED turned OFF"}
```

Conclusion:

This experiment successfully demonstrates **remote control of an IoT device using Flask and ngrok**. Flask provides a **lightweight, easy-to-use API** for controlling the LED. Ngrok allows **secure remote access without configuring firewalls**. This approach can be extended to **control real IoT hardware (Raspberry Pi, Arduino, ESP32, etc.)**.

Homework Assigned

AIM: To develop an extended IoT-based control system that allows remote control of multiple virtual devices (LED and Fan) using Flask as a web framework and ngrok for exposing the local server.

Objectives:

1. To create a web-based interface for controlling multiple IoT devices (LED and Fan).
2. To use Flask to define REST API endpoints for interacting with these devices.
3. To expose the local Flask server to the internet using ngrok.
4. To enable users to remotely turn the devices ON or OFF and check their status.

Tools Used:

- **Python (Version 3.x)**
- **Flask** (for creating the web server)
- **ngrok** (for exposing Flask to the internet)
- **pip** (Python package manager)
- **Web Browser / Postman** (for testing API)
- **Command Prompt/Terminal** (for running the server)

Program Code cum Procedure

Step 1: Install Required Packages

Before running the program, install dependencies:

```
pip install flask pyngrok
```

Step 2: Save the Flask Code as app.py

Create a new file named app.py and copy-paste the following code:

```
from flask import Flask, request, jsonify
app = Flask(__name__)

# Initial states of the virtual IoT devices
device_state = {"LED": "OFF", "Fan": "OFF"}

@app.route('/')
def home():
    return "<h1>IoT Device Control</h1>" \
           "<p>Use /control?device=LED,Fan&state=ON,OFF to control the devices.</p>"

@app.route('/control', methods=['GET'])
def control_devices():
    devices = request.args.get('device').split(',')
    states = request.args.get('state').split(',')

    if len(devices) != len(states):
        return jsonify({"error": "Mismatch between number of devices and states"})

        messages = []  # To store individual device update messages
    for device, state in zip(devices, states):
        if device in device_state and state in ["ON", "OFF"]:
            device_state[device] = state
            messages.append(f"{device} turned {state}")
        else:
            return jsonify({"error": f"Invalid device or state: {device} - {state}"})

    return jsonify({"messages": messages})

@app.route('/status', methods=['GET'])
def device_status():
    return jsonify(device_state)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 3: Run Flask Server
Open Command Prompt (Windows) or Terminal (Linux/Mac) and navigate to the directory where app.py is saved:

```
python app.py
```

Output:

```
* Running on http://127.0.0.1:5000/
```

Step 4: Start ngrok
In a new terminal window, run:

```
ngrok http 5000
```

It will generate a public URL, e.g.:
http://your-ngrok-url.ngrok.io -> http://127.0.0.1:5000

Step 5: Test the API
Use your browser or Postman to access the following URLs:
http://your-ngrok-url.ngrok.io/
Expected Output:

```
<h1>IoT Device Control</h1>
<p>Use /control?device=LED,Fan&state=ON,OFF to control the devices.</p>
```

1. **Control Both Devices at Once:**
   Turn LED ON and Fan OFF:

http://your-ngrok-url.ngrok.io/control?device=LED,Fan&state=ON,OFF
Expected JSON Response:

```
{
    "messages": ["LED turned ON" , "Fan turned OFF"]
}
```

2. **Check the Device Status:**

http://your-ngrok-url.ngrok.io/status
Expected JSON Response:

```
{
  "LED": "ON",
  "Fan": "OFF"
}
```

Explanation of the Program

Import Required Modules:

```
from flask import Flask, request, jsonify
```

- **Flask:** Used to create a web server.
- **request:** Extracts parameters from API requests.
- **jsonify:** Formats responses in JSON format.

Initialize Flask App:

```
app = Flask(__name__)
```

- **Flask(name)** initializes the Flask web application.

Define the Devices' States:

```
device_state = {"LED": "OFF", "Fan": "OFF"}
```

- A dictionary (device_state) stores the status ("ON"/"OFF") of both devices.
- Initially, both devices are OFF.

Define the Home Route (/):

```
@app.route('/')
def home():
  return "<h1>IoT Device Control</h1>" \
       "<p>Use /control?device=LED,Fan&state=ON,OFF to control the devices.</p>"
```

- This route displays instructions for using the API.

Define the Control Route (/control):

```
@app.route('/control', methods=['GET'])
def control_devices():
  devices = request.args.get('device').split(',')
  states = request.args.get('state').split(',')
```

- Splits the device and state parameters into lists based on commas.

Validation:

```
if len(devices) != len(states):
  return jsonify({"error": "Mismatch between number of devices and states"})
```

- Ensures that the number of devices matches the number of states.

Updating Device States:

```
messages = []  # To store individual device update messages

  for device, state in zip(devices, states):
    if device in device_state and state in ["ON", "OFF"]:
      device_state[device] = state
      messages.append(f"{device} turned {state}")
    else:
      return jsonify({"error": f"Invalid device or state: {device} - {state}"})
  return jsonify({"messages": messages, "device_states": device_state})
```

- Loops through the devices and their corresponding states, updating the device_state dictionary.

Define the Status Route (/status):

```
@app.route('/status', methods=['GET'])
def device_status():
    return jsonify(device_state)
```

- Returns the current state of all devices in JSON format.

Start the Flask Server:

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Runs the Flask app on port 5000, allowing access from any network.

Exposing Flask via ngrok

After starting Flask, we need to **expose it to the internet**:

Run this command in a new terminal window:

```
ngrok http 5000
```

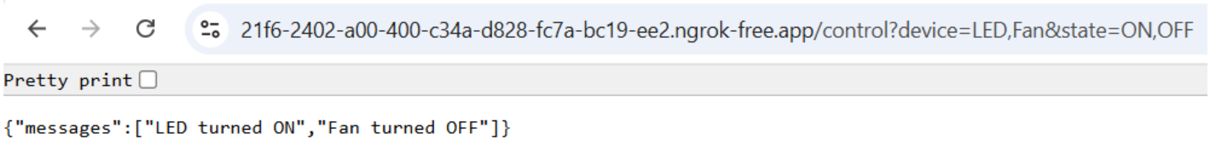
Expected output:

```
Forwarding http://your-ngrok-url.ngrok.io -> http://127.0.0.1:5000
```

Now, use the **ngrok URL** to access the API from **anywhere**.

Outputs:

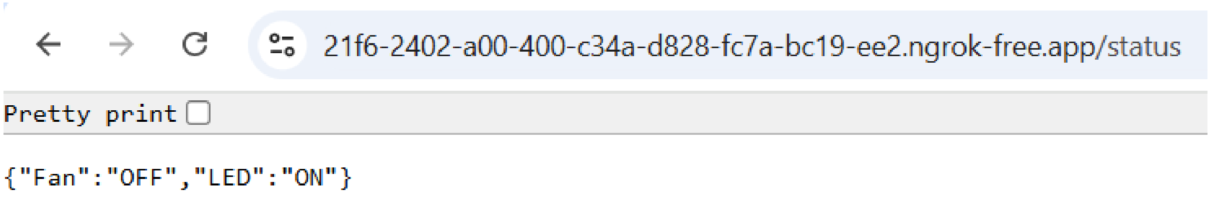
https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=ON,OFF


21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=ON,OFF

Pretty print

```
{"messages":["LED turned ON","Fan turned OFF"]}
```

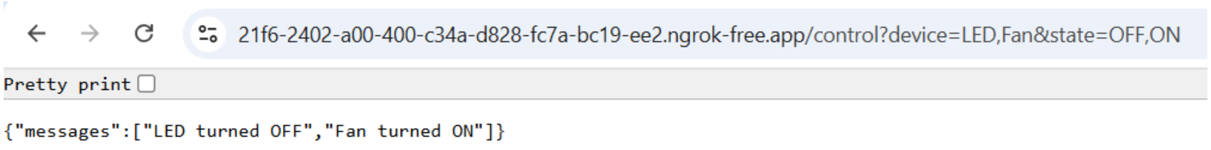
https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status


21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

Pretty print

```
{"Fan":"OFF","LED":"ON"}
```

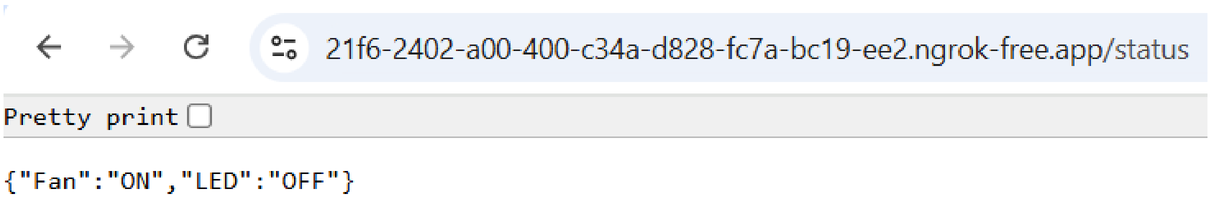
https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=OFF,ON


21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=OFF,ON

Pretty print

```
{"messages":["LED turned OFF","Fan turned ON"]}
```

https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

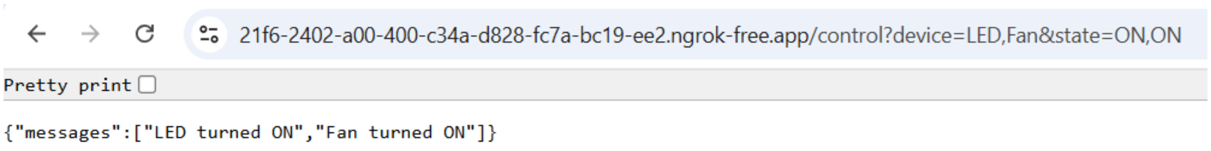


21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

Pretty print

```
{"Fan":"ON","LED":"OFF"}
```

https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=ON,ON

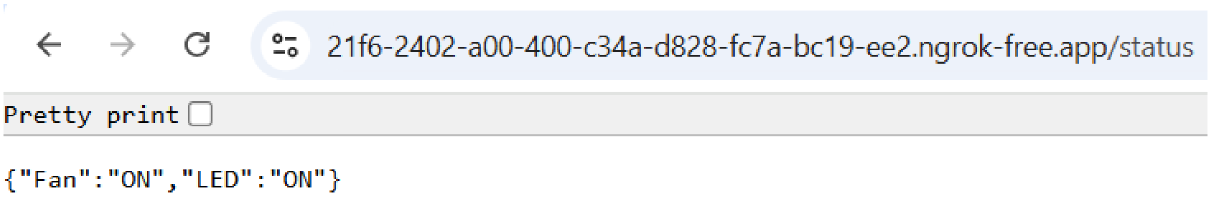


21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/control?device=LED,Fan&state=ON,ON

Pretty print

```
{"messages":["LED turned ON","Fan turned ON"]}
```

https://21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status



21f6-2402-a00-400-c34a-d828-fc7a-bc19-ee2.ngrok-free.app/status

Pretty print

```
{"Fan":"ON","LED":"ON"}
```

Conclusion:
This experiment demonstrates controlling multiple IoT devices (**LED and Fan**) at the same time using **Flask** and **ngrok**. It successfully demonstrates **remote control of IoT devices using Flask and ngrok**. Flask provides a **lightweight, easy-to-use API** for controlling the devices. Ngrok allows **secure remote access without configuring firewalls**. This approach can be extended to **control real IoT hardware (Raspberry Pi, Arduino, ESP32, etc.)**.