# Project 1: Django and MQTT

# Project 1: Django and MQTT - Report

## Project Overview

This project demonstrates the Django web framework with WebSocket support and MQTT integration. It consists of two main tasks: - **Task 1**: Django server with basic HTTP, WebSocket, and MQTT listener capabilities - **Task 2**: Python application that fetches data from an API, saves it to JSON, and publishes to MQTT

## Task 1: Django Framework Implementation

### Task 1.1: Basic Django Web Server

**Implementation:** - Created Django project `django_mqtt` with app `myapp` - Configured basic HTTP view returning server status

**Code (myapp/views.py):**

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Django MQTT Server</h1><p>Server is
        running!</p>")
```

**Running:**

```
python manage.py runserver 127.0.0.1:8000
```

**Result:**

Basic Django Server
Basic Django Server

The server successfully starts and returns the HTML response when accessed via browser.

## Task 1.2: Django WebSocket Server

**Implementation:** - Added Django Channels and Daphne for ASGI/WebSocket support - Created WebSocket consumer that echoes messages back to client - Built HTML page with JavaScript WebSocket client

### Code (myapp/consumers.py):

```python
import json
from channels.generic.websocket import WebsocketConsumer

class EchoConsumer(WebsocketConsumer):
    def connect(self):
        self.accept()
        self.send(text_data=json.dumps({'message': 'Connected to
        WebSocket!'}))

    def receive(self, text_data):
        data = json.loads(text_data)
        message = data.get('message', '')
        self.send(text_data=json.dumps({'message': f'Echo:
        {message}'}))
```

### ASGI Configuration (django_mqtt/asgi.py):

```python
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from myapp.routing import websocket_urlpatterns

application = ProtocolTypeRouter({
    "http": django_asgi_app,
    "websocket":
        AuthMiddlewareStack(URLRouter(websocket_urlpatterns)),
})
```

### Running:

```
daphne -b 127.0.0.1 -p 8000 django_mqtt.asgi:application
```

### Result:

WebSocket Test
WebSocket Test

The WebSocket connection establishes successfully, and messages are echoed back from the server in real-time.

## Task 1.3: Django MQTT Listener

**Implementation:** - Created Django management command `mqtt_listener` - Uses paho-mqtt library to subscribe to all MQTT topics - Prints received messages to console

**Code (management/commands/mqtt_listener.py):**

```
from django.core.management.base import BaseCommand
import paho.mqtt.client as mqtt

class Command(BaseCommand):
    def handle(self, *args, **options):
        def on_connect(client, userdata, flags, rc, properties=None):
            self.stdout.write(self.style.SUCCESS(f'Connected to MQTT
        broker'))
            client.subscribe('#')  # Subscribe to all topics

        def on_message(client, userdata, msg):
            self.stdout.write(f'[{msg.topic}] {msg.payload.decode()}')

        client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
        client.on_connect = on_connect
        client.on_message = on_message
        client.connect('localhost', 1883, 60)
        client.loop_forever()
```

**Running:**

```
# Terminal 1
python manage.py mqtt_listener

# Terminal 2 - send test messages
docker exec mosquitto mosquitto_pub -t "test/topic" -m "Hello from
        MQTTx!"
```

**Result:**

MQTT Listener

MQTT Listener

The listener successfully connects to the MQTT broker, subscribes to all topics, and displays received messages with their topic names.

## Task 2: API to MQTT Integration

**Implementation:** - Python script that fetches weather data from wttr.in API - Saves complete response to `output.json` - Extracts key weather metrics and publishes to MQTT

**Code (api_fetch.py):**

```
import json
import requests
import paho.mqtt.client as mqtt

CITY = "Vilnius"
API_URL = f"https://wttr.in/{CITY}?format=j1"
MQTT_TOPIC = "weather/data"

def fetch_weather():
    response = requests.get(API_URL, timeout=10)
    return response.json()

def save_to_file(data, filename="output.json"):
    with open(filename, 'w') as f:
        json.dump(data, f, indent=2)

def publish_to_mqtt(data):
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    client.connect("localhost", 1883, 60)

    current = data.get('current_condition', [{}])[0]
    weather_info = {
        'city': CITY,
        'temperature_c': current.get('temp_C'),
        'humidity': current.get('humidity'),
        'pressure': current.get('pressure'),
        'weather_desc': current.get('weatherDesc', [{}])
        [0].get('value'),
    }
    client.publish(MQTT_TOPIC, json.dumps(weather_info))
    client.disconnect()
```

**Running:**

```
python api_fetch.py
```

**Result:**

API to MQTT
API to MQTT

The script successfully: 1. Fetches weather data from wttr.in API for Vilnius 2. Saves complete JSON response to `output.json` file 3. Publishes extracted weather metrics to MQTT topic `weather/data`

## Technologies Used

- **Django 6.0** - Web framework
- **Django Channels 3.0.4** - WebSocket support
- **Daphne** - ASGI server
- **paho-mqtt** - MQTT client library
- **Eclipse Mosquitto** - MQTT broker (Docker)
- **wttr.in API** - Weather data source

## Conclusion

This project successfully demonstrates: 1. **Django basics** - HTTP request handling and routing 2. **Real-time communication** - WebSocket implementation with bidirectional messaging 3. **IoT integration** - MQTT protocol for publish/subscribe messaging 4. **API integration** - External data fetching and transformation 5. **Data persistence** - JSON file storage

The combination of Django's web framework with MQTT messaging creates a foundation for building IoT dashboard applications that can receive sensor data, display it via web interface, and enable real-time updates through WebSockets.

### Key Learning Points

- Django Channels extends Django to handle WebSocket connections

- ASGI (Asynchronous Server Gateway Interface) replaces WSGI for async capabilities
- MQTT's pub/sub pattern is ideal for IoT device communication
- Integration between different protocols (HTTP, WebSocket, MQTT) in one application