**1. Tổng quan về UI**

* **Vai trò**: ui.py là thành phần giao diện người dùng (User Interface), cung cấp một web server để:
  1. Hiển thị dữ liệu thời gian thực (Dashboard, Gateways, Beacons, Logs).
  2. Cho phép điều khiển Gateway (bật/tắt).
  3. Quản lý cấu hình hệ thống và đổi mật khẩu.
* **Input**: Dữ liệu từ Redis (beacon\_data, beacon\_state, logs, aws\_queue).
* **Output**:
  1. Giao diện web hiển thị dữ liệu.
  2. Lệnh điều khiển gửi tới NanoMQ qua MQTT.
* **Thư viện**:
  1. flask: Web framework.
  2. flask-socketio: Cập nhật dữ liệu thời gian thực qua WebSocket.
  3. paho-mqtt: Gửi lệnh tới NanoMQ.
  4. redis: Lấy dữ liệu từ Redis.

**2. Cách UI hoạt động**

**2.1. Khởi tạo và cấu hình**

* **Khởi tạo Flask app**:
  + Tạo ứng dụng Flask và thiết lập khóa bí mật:

python

CollapseWrapCopy

app = Flask(\_\_name\_\_)

app.secret\_key = os.getenv("SECRET\_KEY", "your-secret-key")

socketio = SocketIO(app)

* **Load biến môi trường**:
  + Lấy cấu hình từ .env:

python

CollapseWrapCopy

load\_dotenv()

BROKER\_HOST = os.getenv("BROKER\_HOST", "localhost")

BROKER\_PORT = int(os.getenv("BROKER\_PORT", 1883))

QUEUE\_HOST = os.getenv("QUEUE\_HOST", "localhost")

QUEUE\_PORT = int(os.getenv("QUEUE\_PORT", 6379))

* **Kết nối Redis**:
  + Tạo kết nối tới Redis để truy xuất dữ liệu:

python

CollapseWrapCopy

redis\_client = redis.Redis(host=QUEUE\_HOST, port=QUEUE\_PORT, db=0)

* **Kết nối MQTT**:
  + Tạo client MQTT để gửi lệnh điều khiển Gateway:

python

CollapseWrapCopy

mqtt\_client = mqtt.Client()

mqtt\_client.connect(BROKER\_HOST, BROKER\_PORT, 60)

mqtt\_client.loop\_start()

* + loop\_start(): Chạy vòng lặp MQTT trong thread riêng để xử lý gửi tin nhắn.
* **Danh sách Gateway tĩnh**:
  + Một danh sách Gateway mẫu để hiển thị:

python

CollapseWrapCopy

gateways = [

{"id": "GW1", "ip": "192.168.1.1", "status": "Online", "beacons": 0},

{"id": "GW2", "ip": "192.168.1.2", "status": "Online", "beacons": 0},

{"id": "GW3", "ip": "192.168.1.3", "status": "Offline", "beacons": 0}

]

**2.2. Quản lý người dùng**

* **Khởi tạo users**:
  + Nếu Redis chưa có hash users, tạo tài khoản mặc định (admin/admin123):

python

CollapseWrapCopy

def init\_users():

if not redis\_client.exists("users"):

redis\_client.hset("users", "admin", hashlib.md5("admin123".encode()).hexdigest())

init\_users()

* **Decorator login\_required**:
  + Bảo vệ các route yêu cầu đăng nhập:

python

CollapseWrapCopy

def login\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if 'username' not in session:

return redirect(url\_for('login'))

return f(\*args, \*\*kwargs)

return decorated\_function

**2.3. Các route và chức năng**

1. **Login (/login)**:
   * Hiển thị form đăng nhập và xử lý POST:

python

CollapseWrapCopy

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

stored\_password = redis\_client.hget("users", username)

if stored\_password and stored\_password.decode() == hashlib.md5(password.encode()).hexdigest():

session['username'] = username

return redirect(url\_for('dashboard'))

return render\_template('login.html', error="Invalid credentials")

return render\_template('login.html')

* + Kiểm tra mật khẩu bằng hash MD5, lưu username vào session nếu hợp lệ.

1. **Change Password (/change-password)**:
   * Cho phép người dùng đổi mật khẩu:

python

CollapseWrapCopy

@app.route('/change-password', methods=['GET', 'POST'])

@login\_required

def change\_password():

if request.method == 'POST':

old\_password = request.form['old\_password']

new\_password = request.form['new\_password']

username = session['username']

stored\_password = redis\_client.hget("users", username).decode()

if hashlib.md5(old\_password.encode()).hexdigest() == stored\_password:

redis\_client.hset("users", username, hashlib.md5(new\_password.encode()).hexdigest())

return redirect(url\_for('dashboard'))

return render\_template('change\_password.html', error="Old password incorrect")

return render\_template('change\_password.html')

1. **Dashboard (/ và /api/dashboard)**:
   * Hiển thị giao diện chính và dữ liệu thời gian thực:

python

CollapseWrapCopy

@app.route('/')

@login\_required

def dashboard():

return render\_template('dashboard.html')

@app.route('/api/dashboard', methods=['GET'])

@login\_required

def api\_dashboard():

msg\_rate = redis\_client.llen("beacon\_data")

aws\_rate = redis\_client.llen("aws\_queue")

beacons\_detected = len(redis\_client.hkeys("beacon\_state") or [])

system\_status = {

"gateways": len(gateways),

"gateways\_online": sum(1 for gw in gateways if gw["status"] == "Online"),

"beacons\_detected": beacons\_detected,

"broker": "Online" if mqtt\_client.is\_connected() else "Offline",

"redis": "Online" if redis\_client.ping() else "Offline",

"listener": "Running",

"processor": "Running",

"publisher": "Running",

"alerts": ["GW3 offline for 5 minutes"] if any(gw["status"] == "Offline" for gw in gateways) else []

}

return jsonify({"status": system\_status, "msg\_rate": msg\_rate, "aws\_rate": aws\_rate})

* + **Realtime update**:
    - Thread riêng gửi dữ liệu qua SocketIO mỗi giây:

python

CollapseWrapCopy

def update\_realtime\_data():

while True:

msg\_rate = redis\_client.llen("beacon\_data")

aws\_rate = redis\_client.llen("aws\_queue")

beacons\_detected = len(redis\_client.hkeys("beacon\_state") or [])

system\_status = {...} *# Như trên*

socketio.emit('update\_dashboard', {'status': system\_status, 'msg\_rate': msg\_rate, 'aws\_rate': aws\_rate})

time.sleep(1)

threading.Thread(target=update\_realtime\_data, daemon=True).start()

1. **Gateways (/gateways và /api/gateways)**:
   * Hiển thị danh sách Gateway và điều khiển:

python

CollapseWrapCopy

@app.route('/gateways')

@login\_required

def gateways\_list():

return render\_template('gateways.html')

@app.route('/api/gateways', methods=['GET'])

@login\_required

def api\_gateways():

status\_filter = request.args.get('status')

id\_filter = request.args.get('id', '').lower()

filtered\_gateways = gateways

if status\_filter:

filtered\_gateways = [gw for gw in filtered\_gateways if gw["status"] == status\_filter]

if id\_filter:

filtered\_gateways = [gw for gw in filtered\_gateways if id\_filter in gw["id"].lower()]

for gw in filtered\_gateways:

gw["beacons"] = sum(1 for state in redis\_client.hvals("beacon\_state")

if json.loads(state.decode())["gateway"] == gw["id"])

return jsonify(filtered\_gateways)

@app.route('/api/gateways/control/<gateway\_id>', methods=['POST'])

@login\_required

def api\_control\_gateway(gateway\_id):

action = request.json.get('action')

for gw in gateways:

if gw["id"] == gateway\_id:

gw["status"] = "Online" if action == "on" else "Offline"

mqtt\_client.publish(f"control/{gateway\_id}", json.dumps({"action": action}))

return jsonify({"success": True, "gateway": gw})

return jsonify({"success": False, "error": "Gateway not found"}), 404

1. **Beacons (/beacons và /api/beacons)**:
   * Hiển thị thông tin Beacon:

python

CollapseWrapCopy

@app.route('/beacons')

@login\_required

def beacons\_list():

return render\_template('beacons.html')

@app.route('/api/beacons', methods=['GET'])

@login\_required

def api\_beacons():

gateway\_filter = request.args.get('gateway')

detected\_filter = request.args.get('detected')

id\_filter = request.args.get('id', '').lower()

beacons = {}

for data in redis\_client.lrange("beacon\_data", -100, -1):

d = json.loads(data.decode())

beacon\_id = d["beacon\_id"]

beacons[beacon\_id] = {

"gateway": d["gateway\_id"],

"rssi": d["rssi"],

"last\_seen": d["timestamp"],

"detected": 1 if redis\_client.hexists("beacon\_state", beacon\_id) else 0

}

filtered\_beacons = {k: v for k, v in beacons.items()

if (not gateway\_filter or v["gateway"] == gateway\_filter) and

(detected\_filter is None or v["detected"] == int(detected\_filter)) and

(not id\_filter or id\_filter in k.lower())}

return jsonify(filtered\_beacons)

1. **Config (/config và /api/config)**:
   * Quản lý cấu hình hệ thống:

python

CollapseWrapCopy

@app.route('/config')

@login\_required

def config():

return render\_template('config.html')

@app.route('/api/config', methods=['GET', 'POST'])

@login\_required

def api\_config():

config\_keys = ["WINDOW\_SIZE", "RSSI\_THRESHOLD", "FREQ\_THRESHOLD", "MAX\_FREQ", "W1", "W2", "MAX\_WORKERS", "MAX\_BUFFER\_PER\_BEACON", "QUEUE\_HOST", "QUEUE\_PORT"]

if request.method == 'GET':

config = {key: redis\_client.get(f"config:{key}").decode() if redis\_client.exists(f"config:{key}") else os.getenv(key, "") for key in config\_keys}

return jsonify(config)

elif request.method == 'POST':

for key, value in request.json.items():

if key in config\_keys:

redis\_client.set(f"config:{key}", value)

return jsonify({"success": True})

1. **Logs (/logs và /api/logs)**:
   * Hiển thị log hệ thống:

python

CollapseWrapCopy

@app.route('/logs')

@login\_required

def logs():

return render\_template('logs.html')

@app.route('/api/logs', methods=['GET'])

@login\_required

def api\_logs():

service\_filter = request.args.get('service')

time\_filter = request.args.get('time')

logs = [json.loads(log.decode()) for log in redis\_client.lrange("logs", -50, -1)]

if service\_filter:

logs = [log for log in logs if log["service"] == service\_filter]

if time\_filter:

logs = [log for log in logs if time\_filter in log["time"]]

stats = {"processing\_time": "50ms", "latency": "200ms"}

return jsonify({"logs": logs, "stats": stats})

1. **SocketIO Connect**:
   * Xác nhận kết nối WebSocket:

python

CollapseWrapCopy

@socketio.on('connect')

def handle\_connect():

if 'username' not in session:

return False

print("Client connected")

**2.4. Chạy server**

* Chạy Flask server với SocketIO:

python

CollapseWrapCopy

if \_\_name\_\_ == "\_\_main\_\_":

socketio.run(app, debug=True, host="0.0.0.0", port=5000)

**3. Luồng hoạt động chi tiết**

1. **Khởi động**:
   * Tạo Flask app, kết nối Redis, MQTT, khởi tạo thread realtime.
2. **Đăng nhập**:
   * Người dùng truy cập /login, nhập admin/admin123, lưu session.
3. **Hiển thị dữ liệu**:
   * **Dashboard**: Lấy từ /api/dashboard, cập nhật qua SocketIO.
   * **Gateways**: Lấy từ /api/gateways, hiển thị danh sách và số Beacon.
   * **Beacons**: Lấy từ /api/beacons, hiển thị RSSI và trạng thái.
   * **Logs**: Lấy từ /api/logs, hiển thị log và số liệu.
4. **Điều khiển Gateway**:
   * Gửi POST tới /api/gateways/control/<gateway\_id> với action (on/off).
   * Publish lệnh tới NanoMQ qua topic control/<gateway\_id>.
5. **Cấu hình**:
   * GET /api/config: Hiển thị cấu hình.
   * POST /api/config: Cập nhật cấu hình vào Redis.

**4. Ví dụ minh họa**

* **Đăng nhập**: Truy cập http://localhost:5000, nhập admin/admin123.
* **Dashboard**: Thấy "Gateways: 3, Beacons Detected: 2".
* **Điều khiển Gateway**: POST {"action": "off"} tới /api/gateways/control/GW1 → MQTT publish control/GW1.

**5. Tóm tắt cách hoạt động**

* **Hiển thị**: Lấy dữ liệu từ Redis, render template HTML, cập nhật realtime qua SocketIO.
* **Điều khiển**: Gửi lệnh tới NanoMQ qua MQTT.
* **Quản lý**: Xử lý đăng nhập, đổi mật khẩu, cấu hình.