Trabalho unic beira rio
Aluno: Lucas da Silva Melo

Seguindo esse projeto da Unic, ele é focado no desenvolvimento do protótipo de casa inteligente, com maior ênfase em monitorar automação básica e ambiental, com uma arquitetura utilizando gateway de borda (edge) em Python/FastAPI, banco local SQLite/SQLAlchemy, mensageria MQTT(broker mosquitto e clente Paho), e frontend web em react, vite e recharts, o que permite a visualização das regras simples. O objetivo desse projeto é simples, uma demonstração da coleta e dados mockeado com um sistema que gera valores aleatórios, para visualização da automação reativa (dependendo de regras). Devido ao tempo limitado do aluno, nem todos os requerimentos foram alcançados.

A automação, com seu uso de MQTT como protocolo e de web APIs, permite a replicação desse protótipo para demonstrar a automação residencial que integra redes, desenvolvimento web/embarcado e integridade eletrônica, e como uma replicação, se difere da forma física do projeto, mantendo-se como um projeto de baixo custo. Isso é alcançado com monitoramento de temperatura, umidade, solo e movimento, e automação via regras implementadas pelo usuário. A metodologia foi usada em base da implementação de um backend dee borda com coleta, presistência e websocket para tempo real (FastAPI e Uvicorn), integrando MQTT (Paho e Mosquitto) para ingestão das leituras no backend, para que seja visível no frontend, permitindo a criação de dashboards com Reacts para manter formas de gráficos e tabela, com os sistemas de Recharts, react router e vite. Por fim, a modelagem dos dados e regras no banco, foi com a assistência de SQLAlchemy ORM, permitindo a documentação dos processos e avaliação de desempenhos no próprio sistema.

Em geral, as tecnologias usadas incluí Python 3.12 (FastAPI/Uvicorn/SQLAlchemy), Mosquitto, React/Vite/Recharts. Por estes motivos, para ser possível compilar esse projeto, será necessário essas tecnologias,

winget install -e --id Python.Python.3.12
winget install -e --id OpenJS.NodeJS.LTS
winget install -e --id EclipseFoundation.Mosquitto

Em seguida, com essas ferramentas instaladas, um PowerShell deve ser ativado na pasta do repositório para a criação do ambiente do backend e, como um serviço FastAPI/Uvicorn, esses comandos criam uma venv, e isso automaticamente instalam as dependências restantes ao subir a API local.

(Localização_da_Pasta)\iot-smart-env\edge
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
& "C:\Program Files\mosquitto\mosquitto.exe" -v
uvicorn app.main:app --host 127.0.0.1 --port 8000

Agora, para ativar o dashboard (um aplicativo react + vite), as dependências do react devem ser instaladas para executar o servidor de desenvolvimento.

(Localização_da_Pasta)\iot-smart-env\frontend\dashboard
npm install
(Se faltar roteador, instale: npm i react-router-dom)
'VITE_API_URL=http://127.0.0.1:8000
VITE_USE_MOCK=1' | Out-File -FilePath .env.local -Encoding utf8
npm run dev, e então, abra o navegador em http://localhost:5173/

O cronograma do desenvolvimento do projeto teve início duas semanas atrás, com a instação do python e a criação de venv, sendo o primeiro projeto sério criado pelo aluno que normalmente, é mais especializado em outras linguagens de computação. As dependências só foram utilizadas depois, com fastAPI, Uvicorn, SQLAlchemy, Paho e Mosquitto. Os boots do frontend foram baixados apenas depois, na última semana. Os arquivos desenvolvidos na primeira semana, foram os artefatos de back end, iniciando pelo /edge/app/main.py que foi o ponto de entrada do backend (fastAPI) que inicializa o aplicativo, CORS e registra rotas REST e WEBSOCKET, subindo o WorkerMQTT em um singleton, e é possível fazer a validação disso com uvicorn app.main:app --host 127.0.0.1 --port 8000, acessando /docs e /health. Há a possibilidade de ajustar utilizando settings.ALLOW_ORIGINS, para incluir ambos api_router e ws_router.

/edge/app/core/config.py foi o segundo à ser desenvolvido, que expõe DB_URL, MQTT_HOST, MQTT_PORT, MQTT_TOPIC, ALLOW_ORIGINS e ADMIN_BEARER_TOKEN. Supostamente, isso foi projetado para permitir apenas o 'dono' do servidor a ser capaz de editar dados mockados no frontend, mas como isso foi um detalhe muito avançado, não foi terminado. Em seguida, foi criado o /edge/.env, o que parametriza o edge sem editar código.

No terceiro dia, foi criado o /edge/app/db/models.py com ORM e SQLAlchemy, e o /edge/app/db/db.py e /edge/app/api/routes.py. O primeiro prepara o SQLite com a função init_deb(), para abrir edge_readings.db após o primeiro run, e expor contagens no /health. No caso do routes.py, sua função foi definir api_router e o endpoint /health com consulta no banco, com contagem de leituras e nós.

Com o fim da primeira semana, o /edge/app/mqtt/client.py foi criado como um worker MQTT usando Paho, conectando ao broker settings.MQTT_HOST:MQTT_PORT), assina settings.MQTT_TOPIC, com um callback que decodifica JSON, e processa um serviço de payload (process_incoming_payload()) por uma thread consumidora, permitindo que o Mosquitto seja processado, com mosquitto.exe -v) e publicar com mosquitto_pub e, assim, verificar logs do edge.

No início da segunda semana, o /edge/app/services/ingest.py foi criado, recebendo dicionário decodificado com o fim de validar campos e gravar Reading com SQLAlchemy, e após publicar, GET /readings retorna registros para ser usados depois em /edge/app/api/routes.py, que foi ampliado logo depois, acrescentando o GET /readings (mas com filtros, sendo eles node_id, since, until, limit), para validar o curl "http://127.0.0.1:8000/readings?limit=50" e trazer uma lista em ordem temporal. O motivo dos filtros, é para se tratar de exceções de possíveis payloads malformados no worker MQTT, afim de evitar derrubamento do thread.

Na última semana, com tempo mais limitado, trabalhei na automação do CRUD de regras com modelagem e armazenação das regras de automação e o frontend, trabalhando novamente com o arquivo /edge/app/api/routes.py, em qual usa o swagger /docs com CRUD usando token admin.

Referências:

ECLIPSE FOUNDATION. Paho MQTT (Python) — Documentação do módulo client.https://eclipse.dev/paho/files/paho.mqtt.python/html/client.html

REACT ROUTER. Documentação oficial (v6+). https://reactrouter.com/

VITE. Vite — Documentação oficial (ferramenta de build/dev). https://vite.dev/

ECLIPSE FOUNDATION. Eclipse Mosquitto — Documentação do broker.https://mosquitto.org/documentation/

OASIS OPEN. MQTT Version 5.0 — OASIS Standard (texto completo). https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

PYTHON SOFTWARE FOUNDATION. sqlite3 — DB-API 2.0 interface for SQLite databases. https://docs.python.org/3/library/sqlite3.html