Engenharia Informática e de Computadores

Computação Distribuída

Ano Letivo 2024 / 2025

Relatório Final de Entrega de Projeto:

TrustWind

Alunos:

Nº 14339, Gonçalo Marinho

Nº 14193, Tiago Farinha

Nº 14060, Bernardo Melo

Paço de Arcos, 18 de junho de 2025

Conteúdo

Resumo

Este relatório tem como objetivo agregar e documentar o trabalho realizado no projeto TrustWind no âmbito da unidade curricular de Computação Distribuída. Neste relatório, vai encontrar uma explicação do projeto e como este foi pensado e desenvolvido, assim como a base teórica e tecnológica por trás do mesmo.

O **TrustWind** é um sistema distribuído para monitoramento meteorológico em tempo real, integrando sensores, comunicação MQTT e HTTP, backend *Node.js*, frontend *Flask* e uma interface web moderna. O objetivo é fornecer dados ambientais e elétricos de forma confiável, escalável e acessível.

1 Arquitetura do Sistema

A arquitetura do TrustWind é composta por vários módulos independentes que comunicam entre si através de protocolos padronizados, como HTTP e MQTT. A Figura 1 ilustra a arquitetura geral.

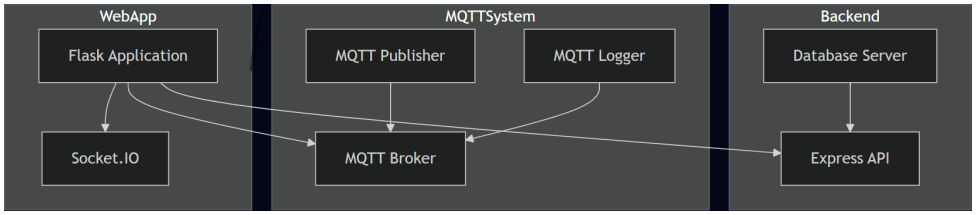

Figura 1: Arquitetura geral do sistema TrustWind

2 Componentes Principais

2.1 Frontend (Flask + SocketIO)

O frontend é responsável por autenticar o utilizador, receber dados em tempo real via WebSocket e apresentar as informações de forma clara e responsiva.

```
   @socketio.on('connect')
def handle_connect():
    if 'utilizador_id' in session:
        logger.info(f"Cliente WebSocket conectado: {request.sid}")
        if mqtt_client and mqtt_connected:
            emit('mqtt_status', {
                'status': 'conectado',
                'mensagem': f'Conectado ao broker MQTT {MQTT_BROKER
                   }:{MQTT_PORT}'
            })
            if cache["dados"] and (time.time() - cache["timestamp"]
                < cache["ttl"]):
                emit('atualizacao_dados', cache["dados"])
        else:
            emit('mqtt_status', {
                'status': 'desconectado',
                'mensagem': 'Não conectado ao broker MQTT'
            })
    else:
        logger.warning(f"Tentativa de conexão WebSocket sem
           autenticação: {request.sid}")
        return False
```

Código 1: Evento de conexão WebSocket

2.2 Backend (Node.js + SQLite)

A API REST em Node.js gere a autenticação e o registo de utilizadores, utilizando uma base de dados SQLite para persistência.

```
app.get("/api/users/:id", async (req, res) => {
  const timestamp = new Date().toLocaleString('pt-BR');
  const userId = req.params.id;
  try {
    console.log(`[${timestamp}] Buscando utilizador com ID: ${
       userId}`);
    const user = await db.get("SELECT * FROM users WHERE id = ?",
       userId);
    if (!user) {
      console.log(`[${timestamp}] utilizador com ID ${userId} não
         encontrado`);
      res.status(404).json({ error: "utilizador não encontrado" });
      console.log(`[${timestamp}] Resposta enviada: 404 Not Found`)
         ;
      return;
    }
    console.log(`[${timestamp}] utilizador encontrado: ${user.email
       }`);
    res.json(user);
    console.log(`[${timestamp}] Resposta enviada: 200 OK`);
  } catch (error) {
    console.error(`[${timestamp}] Erro ao buscar utilizador:`,
       error.message);
    res.status(500).json({ error: "Erro ao buscar utilizador" });
    console.log(`[${timestamp}] Resposta enviada: 500 Internal
       Server Error`);
  }
});
```

Código 2: Gestão de Autenticação

2.3 Broker MQTT

O broker MQTT (`cjsg.ddns.net`) é o núcleo da comunicação em tempo real, recebendo dados meteorológicos dos sensores e distribuindo-os para todos os clientes subscritos.

```
client = mqtt.Client()
client.username_pw_set(USERNAME, PASSWORD)
client.connect(BROKER, PORT, keepalive=60)
client.subscribe('/weather')
```

Código 3: Exemplo de subscrição MQTT em Python

2.4 Scripts Auxiliares

Ao longo do desenvolvimento, foi necessário testar certos componentes à parte para nos certificarmos de que estavam operacionais e que poderiam ser integrados na nossa aplicação.

- `_logger.py`: Subscreve ao tópico MQTT e regista todas as mensagens recebidas.
- `_tester.py`: Publica mensagens de teste no tópico MQTT para simular sensores.

3 Fluxo de Dados

Processos de Autenticação

1. O utilizador submete credenciais através do frontend.
2. O backend Flask envia um pedido HTTP à API Node.js.
3. A API Node.js valida as credenciais contra o banco SQLite.
4. O resultado da autenticação é retornado ao frontend via Flask.

Fluxo Principal de Dados, via MQTT

1. Sensores publicam dados no tópico MQTT.
2. O backend Flask subscreve ao tópico e atualiza o frontend via WebSocket.
3. O frontend exibe os dados em tempo real.
4. Estes dados são os dados meteorológicos.

Fluxo Auxiliar de Dados, via HTTP

1. O client faz pedidos a rotas para obter dados e imagens.

2. O client recebe os dados e atualiza o frontend.
3. O frontend exibe os dados em tempo real.
4. Estes dados são referentes aos dados elétricos e em relação a Câmera.

4 Interface Web

A interface web utiliza TailwindCSS e Google Fonts, focando na usabilidade e clareza. Permite:

- Visualização de dados meteorológicos (temperatura, humidade, localização).
- Visualização de dados elétricos (voltagem, corrente, potência, etc).
- Acesso a stream de câmeras.
- Logs em tempo real das conexões MQTT e WebSocket e HTTP.

5 Contêinerização com Docker

Para facilitar a implantação, portabilidade e escalabilidade do TrustWind, o sistema foi preparado para execução em contentores Docker. Cada componente principal (API Node.js, aplicação Flask) pode ser executado de forma isolada, garantindo ambientes reproduzíveis e independentes do sistema operacional do host.

5.1 Vantagens do Docker em Sistemas Distribuídos

- **Portabilidade:** O mesmo contentor pode ser executado em qualquer máquina com Docker, independentemente do sistema operacional.
- **Isolamento:** Cada serviço roda em seu próprio ambiente, evitando conflitos de dependências.
- **Escalabilidade:** É fácil replicar serviços ou adicionar novos nós apenas ajustando o arquivo de orquestração.
- **Facilidade de Deploy:** O ambiente é definido por código, facilitando CI/CD e testes automatizados.

5.2 Exemplo de Orquestração com `docker-compose`

No Código 4 está um exemplo simplificado de arquivo `docker-compose.yml` para orquestrar os principais serviços do TrustWind.

```yaml
version: '3.8'

networks:
  app-network:
    driver: bridge

services:
  python-server:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./app/mqtt_configs:/app/mqtt_configs
    environment:
      - FLASK_ENV=production
      - API_BASE_URL=http://js-server:3000
    networks:
      - app-network
    depends_on:
      js-server:
        condition: service_healthy
    command: >
      sh -c "sleep 10 && gunicorn --bind 0.0.0.0:5000 --workers 2
         app:app"

  js-server:
    build:
      context: ./database
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./database/data:/app/data
    environment:
      - NODE_ENV=production
    networks:
```

Código 4: docker-compose.yml

5.3 Execução dos Serviços

Para iniciar todos os serviços em contentores, basta executar:

```
docker-compose up --build
```

Com isso, todo o sistema TrustWind pode ser iniciado e gerido facilmente, reforçando as vantagens dos sistemas distribuídos e modernos.

6 Vantagens dos Sistemas Distribuídos

6.1 Comparação com Sistemas Centralizados

Característica	**Centralizado**	**Distribuído (TrustWind)**
Escalabilidade	Limitada pelo servidor central	Fácil de escalar adicionando novos nós
Tolerância a falhas	Ponto único de falha	Falha de um nó não compromete o sistema
Desempenho	Pode sofrer com sobrecarga	Balanceamento de carga entre componentes
Flexibilidade	Difícil de adaptar a novos requisitos	Fácil integração de novos módulos

Tabela 1: Comparação entre sistemas centralizados e distribuídos

7 Execução do Projeto

1. Execute o script `start.bat` no Windows.
2. O backend Node.js será iniciado automaticamente.
3. O Flask ira iniciar junto com o Node.js.
4. Acesse a interface web em `http:localhost:5000`.

8 Considerações Finais

O TrustWind demonstra a integração de múltiplas tecnologias para monitoramento distribuído, com autenticação segura, comunicação em tempo real e interface amigável. O

uso de uma arquitetura distribuída traz vantagens claras em termos de escalabilidade, resiliência e flexibilidade.

Referências

[1] A. Stanford-Clark and A. Nipper, "Mqtt - a publish/subscribe protocol for wireless sensor networks," *2004 IEEE International Conference on Communications*, pp. 1–7, 2004.

[2] S. Powers, *Learning Node.js Development*. Birmingham, UK: Packt Publishing, 2nd ed., 2020.

[3] A. Ronacher, "Flask web development." https://flask.palletsprojects.com/, 2024. Accessed: 2025-06-18.

[4] "Mqtt protocol." https://mqtt.org/, 2025. Accessed: 2025-06-18.

[5] Internet Engineering Task Force (IETF), "Hypertext transfer protocol version 2 (http/2)." https://datatracker.ietf.org/doc/html/rfc7540, 2015. RFC 7540, May 2015. Accessed: 2025-06-18.

[6] SQLite Consortium, *SQLite Documentation*. SQLite, 2025. Accessed: 2025-06-18.

[7] Tailwind Labs, "Tailwind css documentation." https://tailwindcss.com/docs, 2025. Accessed: 2025-06-18.